Adaptive Checkpoint Adjoint Method for Gradient Estimation in Neural ODE

Juntang Zhuang, Nicha C. Dvornek, Xiaoxiao Li, Sekhar Tatikonda, Xenophon Papademetris, James Duncan
Yale University

Background

- Neural ordinary differential equation (NODE) is a continuous-depth model, and parameterizes the derivative of hidden states with a neural network. (Chen et al., 2018)

NODE achieves great success in free-form reversible generative models (Grathwohl et al., 2018), time series analysis (Rubanova et al., 2019)

- However, on benchmark tasks such as image classification, the empirical performance of NODE is significantly inferior to state-of-the-art discrete-layer models (Dupont et al., 2019; Gholami et al., 2019).

- We identify the problem is numerical error of gradient estimation for continuous models, and propose a new method for accurate gradient estimation in NODE.

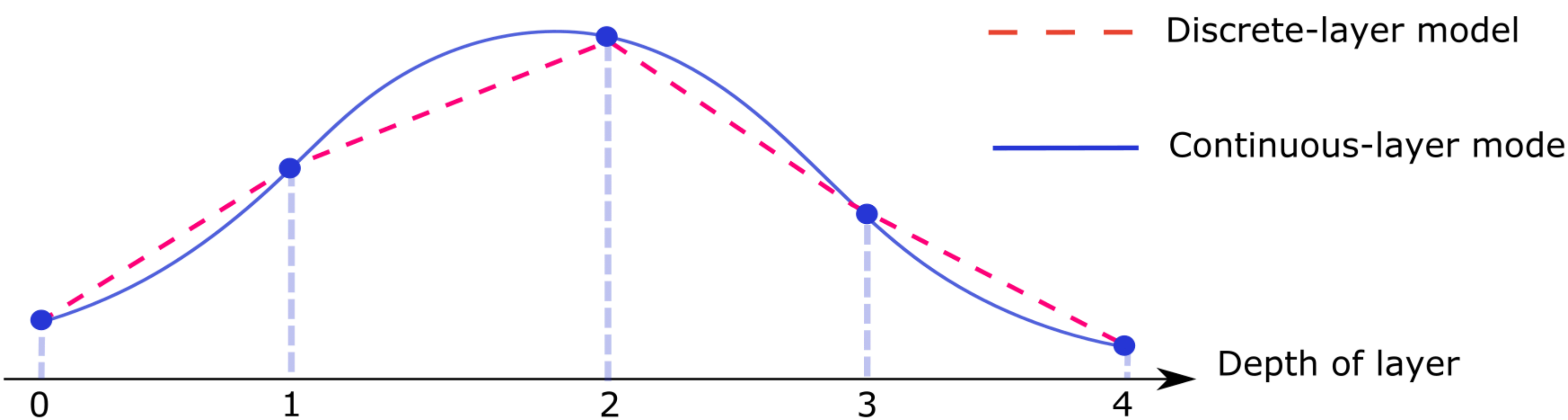
Recap: from discrete-layer ResNet to Neural ODE

Discrete-layer ResNet

$$y = x + f_\theta(x)$$

Continuous-layer model

$$\frac{dz(t)}{dt} = f(z(t), t, \theta), \quad s.t. \quad z(0) = x$$



We call t “continuous depth” or “continuous time” interchangeably.

Chen, Tian Qi, et al. "Neural ordinary differential equations." Advances in neural information processing systems. 2018.

Forward pass of an ODE

Input: $z(0) = x$

Output: $\hat{y} = z(T) = z(0) + \int_0^T f(z(t), t, \theta) dt$

Loss: $L(\hat{y}, y) = L(z(T), y)$

Analytical form of adjoint method to determine grad w.r.t. θ

(1) Solve $z(t)$ from $t = 0$ to $t = T$
Determine $\lambda(T)$

$$\lambda(T) = -\frac{\partial L}{\partial z(T)}$$

(2) Solve $\lambda(t)$ from $t = T$ to $t = 0$

$$\frac{d\lambda(t)}{dt} + \left(\frac{\partial f(z(t), t, \theta)}{\partial z(t)}\right)^\top \lambda(t) = 0 \quad \forall t \in (0, T)$$

(3) Determine $\frac{dL}{d\theta}$ in an integral form

$$\frac{dL}{d\theta} = \int_T^0 \lambda(t)^\top \frac{\partial f(z(t), t, \theta)}{\partial \theta} dt$$

Pontryagin, L. S. Mathematical theory of optimal processes. Routledge, 1962.

Numerical implementation of the adjoint method

Analytical Form

(1) Solve $z(t)$ from $t = 0$ to $t = T$
Determine $\lambda(T)$

Forward-time

(2) Solve $\lambda(t)$ from $t = T$ to $t = 0$

Reverse-time

(3) Determine $\frac{dL}{d\theta}$ in an integral form

Numerical implementation

(1) Solve $z(T)$ with numerical ODE solvers.
Determine $\lambda(T) = -\frac{\partial L(z(T),y)}{\partial z(T)}$
Delete forward-time trajectory $z(t), 0 < t < T$ on the fly

(2) Numerically solve the following augmented ODE from $t = T$ to $t = 0$

$$\frac{dz(t)}{dt} = f(z(t), t, \theta) \qquad z(T) = z(T)$$

$$\frac{d\lambda(t)}{dt} = -\frac{\partial f}{\partial z}^T \lambda(t) \quad s.t. \quad \lambda(T) = -\frac{\partial L(z(T), y)}{\partial z(T)}$$

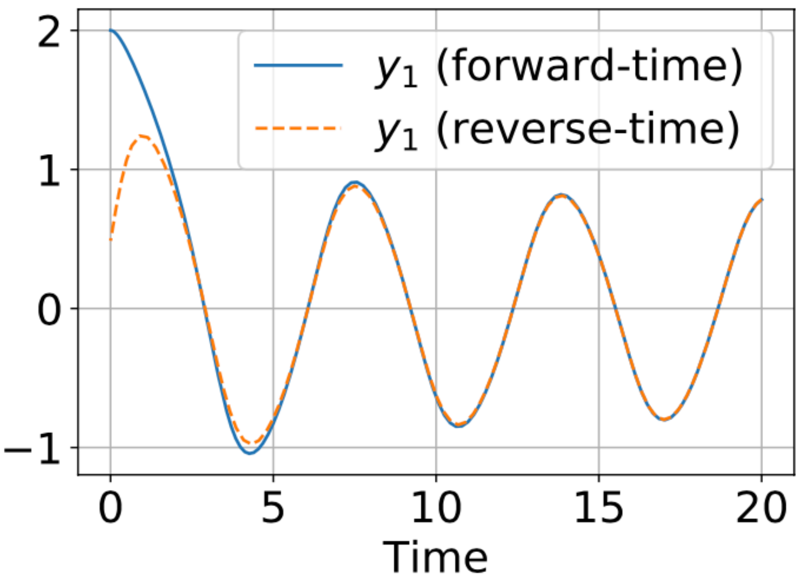
$$\frac{d}{dt}\left(\frac{dL}{d\theta}\right) = -\lambda(t)^T \frac{\partial f}{\partial \theta} \qquad \frac{dL}{d\theta}\bigg|_{t=T} = 0$$

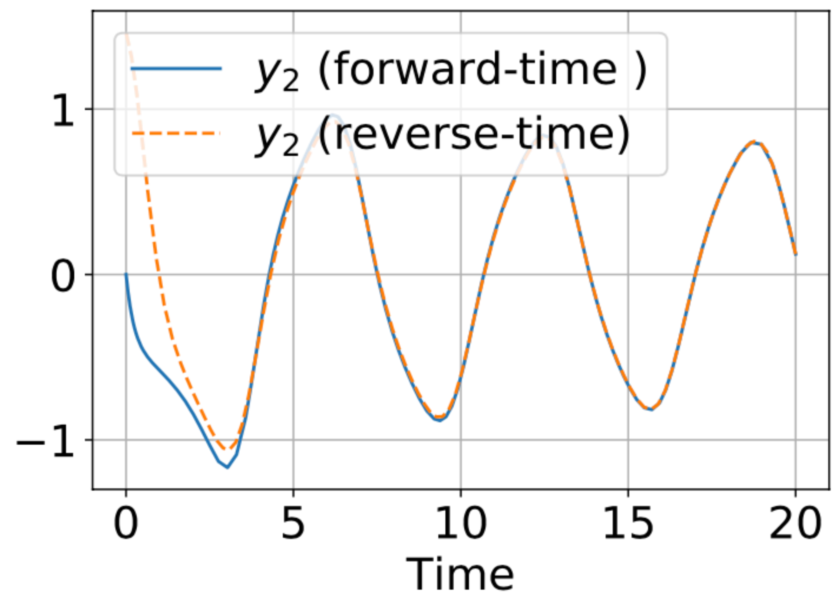
Solve augmented ODE in reverse-time

Chen, Tian Qi, et al. "Neural ordinary differential equations." Advances in neural information processing systems. 2018.

Forward-time trajectory $z(t)$ and reverse-time trajectory $\overline{z(t)}$ might mismatch due to numerical errors

Experiment with *van der Pol* equation, using ode45 solver in MATLAB





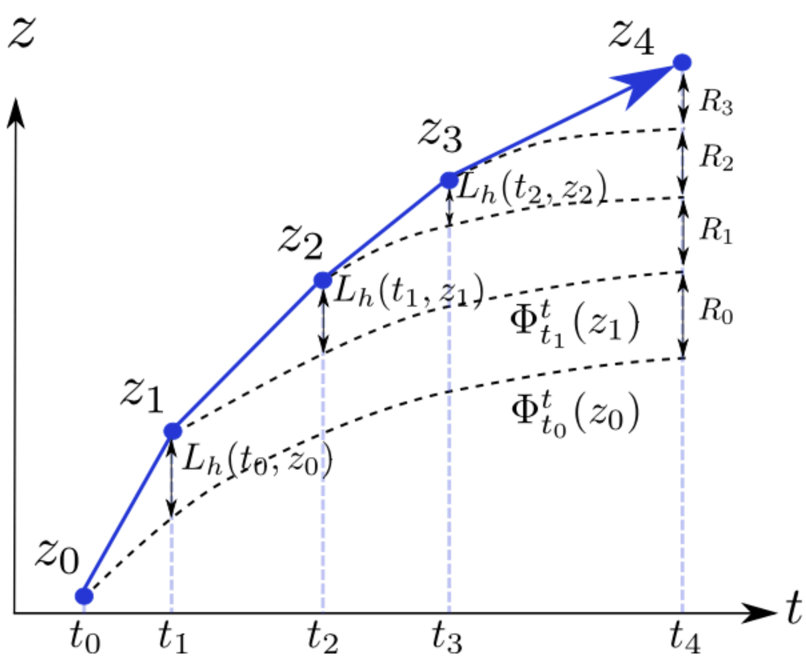
Forward-time trajectory $z(t)$ and reverse-time trajectory $\overline{z(t)}$ might mismatch due to numerical errors

Experiment with an ODE defined by convolution, using ode45 solver in MATLAB

Input

Reverse-time reconstruction

Recap: Numerical ODE solvers with adaptive stepsize



Algorithm 1 Numerical Integration

Input: data x, end time T, error tolerance $etol$, initial stepsize h
Initialize: $z = x, t = 0$, error estimate $\hat{e} = \infty$
while $t < T$ **do**
  while $\hat{e} > etol$ **do**
    $h \leftarrow h \times decay\_factor(\hat{e})$
    $\hat{e}, \hat{z} = \psi_h(t, z)$
  end while
  $t \leftarrow t + h, z \leftarrow \hat{z}$
end while

$z_i(t_i)$ Hidden state at time t_i

h_i The stepsize in time

$\Psi_{h_i}(t_i, z_i)$ The *numerical* solution at time $t_i + h_i$, starting from (t_i , z_i). It returns both the numerical approximation of $z(t_i + h_i)$ and an estimate of truncation error $\hat{e}$.

Adaptive checkpoint adjoint (ACA) method

Algorithm 2 ACA: Record $z(t)$ with Minimal Memory

Forward-pass:

(1) Keep accepted discretization points $\{t_0, ...t_{N_t}\}$

(2) Keep z values $\{z_0, z_1, ...z_{N_t}\}$ (Not $\psi_{h_i}(t_i, z_i)$)

(3) Delete local computation graphs to search for optimal stepsize

Backward-pass:

Initialize $\lambda(T)$, $\frac{dL}{d\theta} = 0$

For N_t to 1:

(1) local forward: $z_{i+1} = \psi(t_i, z_i)$ with stepsize $h_i = t_{i+1} - t_i$

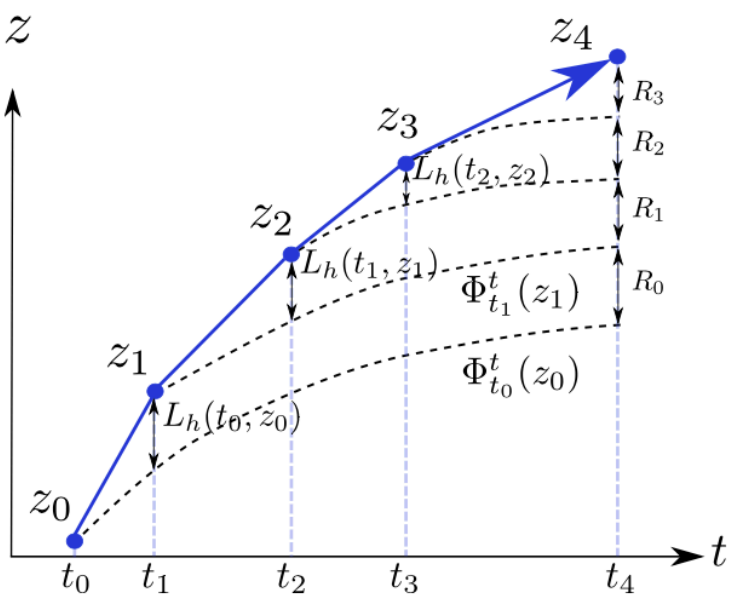
(2) local backward, update λ and $\frac{dL}{\theta}$ according to discretization of adjoint equations

(3) Delete local computation graphs.

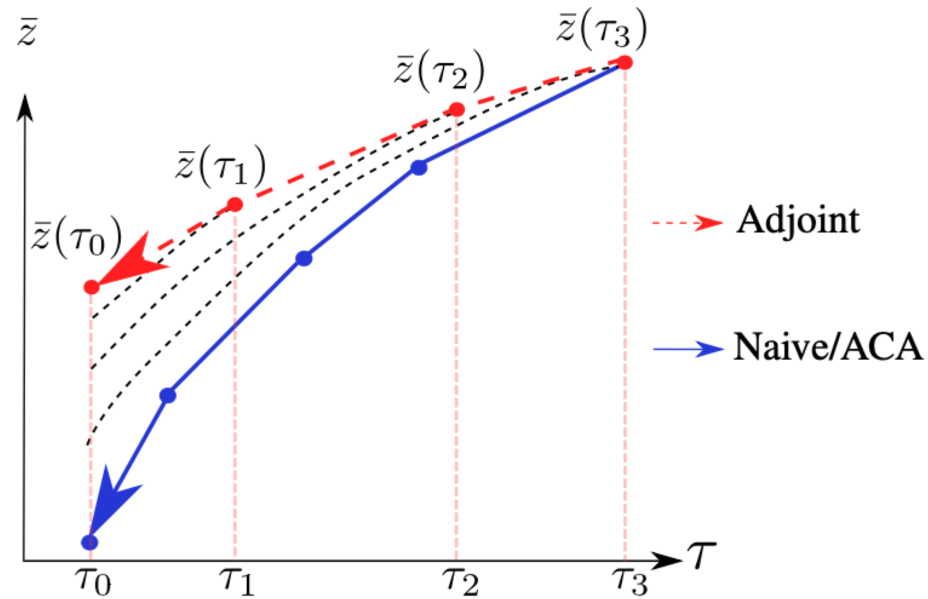
Record $z(t)$ to guarantee numerical accuracy

Delete redundant computation graph and recollect memory

Comparison of different methods



Forward-time trajectory



Reverse-time trajectory

Comparison with naïve method (direct back-prop through ODE solver)

Forward-pass of a single numerical step:

Suppose it takes m steps to find an acceptable stepsize h_m, such that the estimated error is below tolerance $error_m < tolerance$

$$out_1, h_1, error_1 = \psi(t, h_0, z)$$
$$out_2, h_2, error_2 = \psi(t, h_1, z)$$
$$...$$
$$out_m, h_m, error_m = \psi(t, h_m, z)$$

Backward-pass of a single numerical step:

Naïve method	ACA (ours)
Take h_m as a recursive function of h_0 and z	Take h_m as a constant
Equivalent depth of computation graph is $O(m)$	Equivalent depth is $O(1)$
The deeper computation graph might cause numerical errors in gradient estimation (vanishing or exploding gradient)	The exploding and vanishing gradient issue is alleviated

Comparison of different methods

	Naive	Adjoint [1]	ACA (Ours)
Computation Cost	$O(N_f \times N_t \times m \times 2)$	$O(N_f \times (N_t + N_r) \times m)$	$O(N_f \times N_t \times (m+1))$
Memory Consumption	$O(N_f \times N_t \times m)$	$O(N_f)$	$O(N_f + N_t)$
Depth of computation graph	$O(N_f \times N_t \times m)$	$O(N_f \times N_r)$	$O(N_f \times N_t)$
Reverse accuracy	✓	✗	✓

N_f: Number of layers (or parameters) in f

N_t: Number of discretized time points in forward-time numerical integration

N_r: Number of discretized time points in reverse-time numerical integration. Note that N_r is only meaningful for adjoint method [1]

m: Average number of iterations to find an acceptable stepsize (whose estimated error is below error tolerance)

[1] Chen, Tian Qi, et al. "Neural ordinary differential equations." Advances in neural information processing systems. 2018.

Comparison of different methods

	Naive	Adjoint [1]	ACA (Ours)
Computation Cost	$O(N_f \times N_t \times m \times 2)$	$O(N_f \times (N_t + N_r) \times m)$	$O(N_f \times N_t \times (m+1))$
Memory Consumption	$O(N_f \times N_t \times m)$	$O(N_f)$	$O(N_f + N_t)$
Depth of computation graph	$O(N_f \times N_t \times m)$	$O(N_f \times N_r)$	$O(N_f \times N_t)$
Reverse accuracy	✓	✗	✓

Take-home message:

(1) Compare with adjoint method, ACA guarantees the accuracy of reverse-time trajectory.

(2) Compared with naïve method, ACA has a shallower computation graph, hence is more robust to vanishing and exploding gradient issue.

[1] Chen, Tian Qi, et al. "Neural ordinary differential equations." Advances in neural information processing systems. 2018.

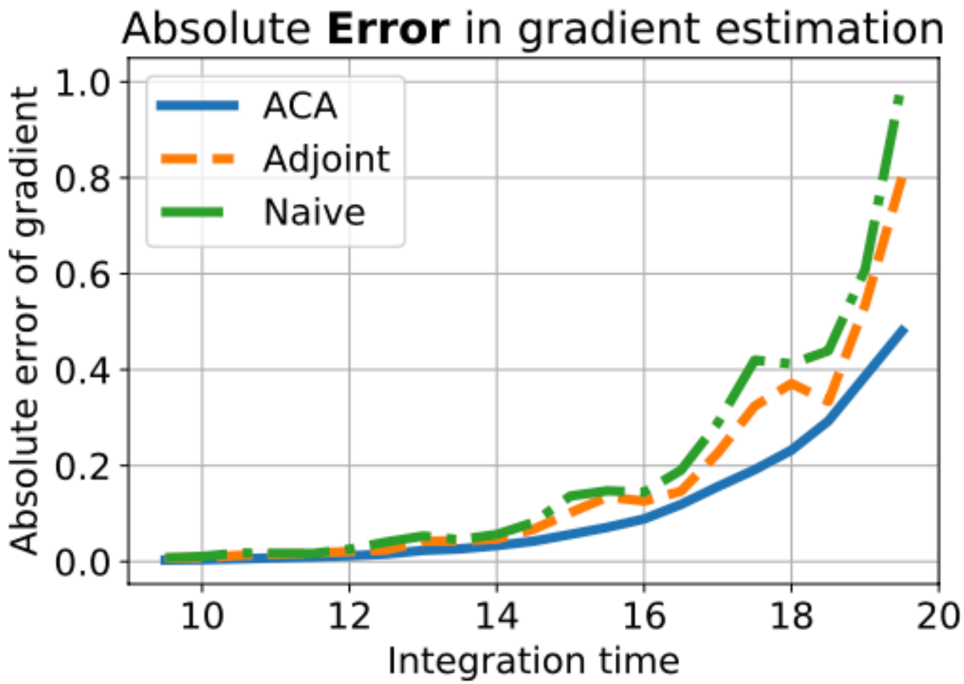
Comparison of different methods

Consider a toy example whose gradient can be analytically solved

$$\frac{dz(t)}{dt} = kz(t), \;\; z(0) = z_0$$

$$L(z(T)) = z(T)^2 = z_0^2 exp(2kT)$$

$$\frac{dL}{dz_0} = 2z_0 exp(2kT)$$



Experimental results

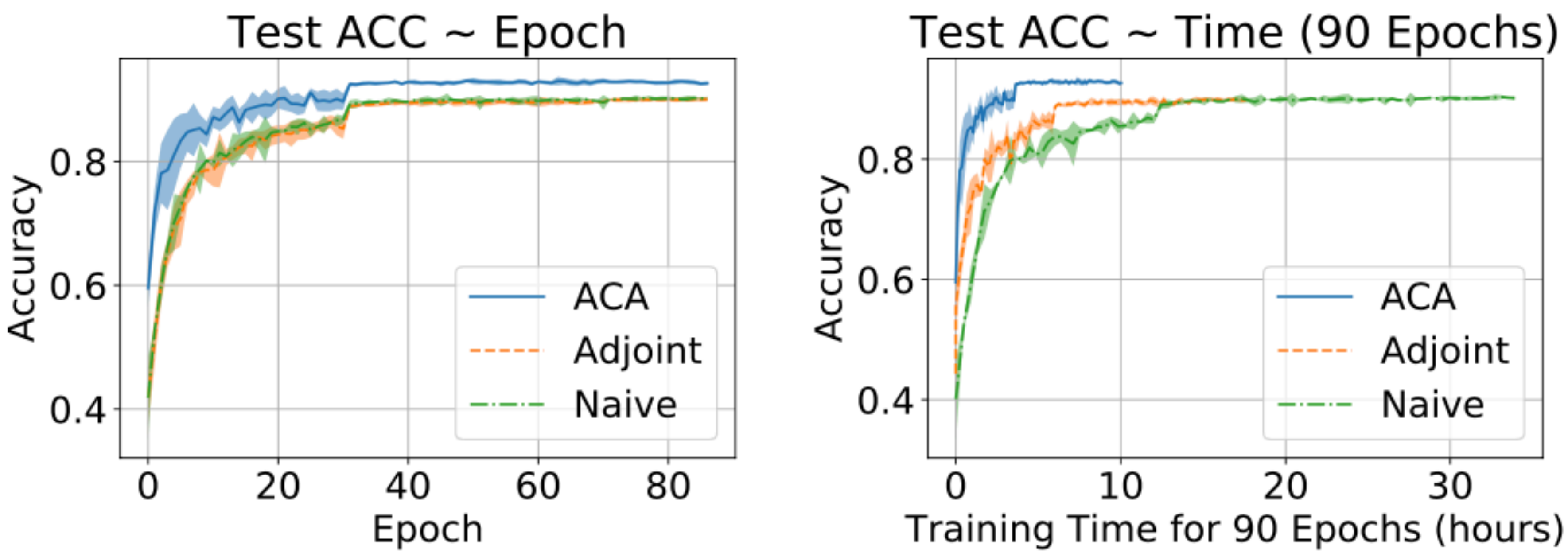
Supervised image classification

We directly modify a ResNet18 into its corresponding NODE counterpart

In a residual block: $y = x + f(x)$

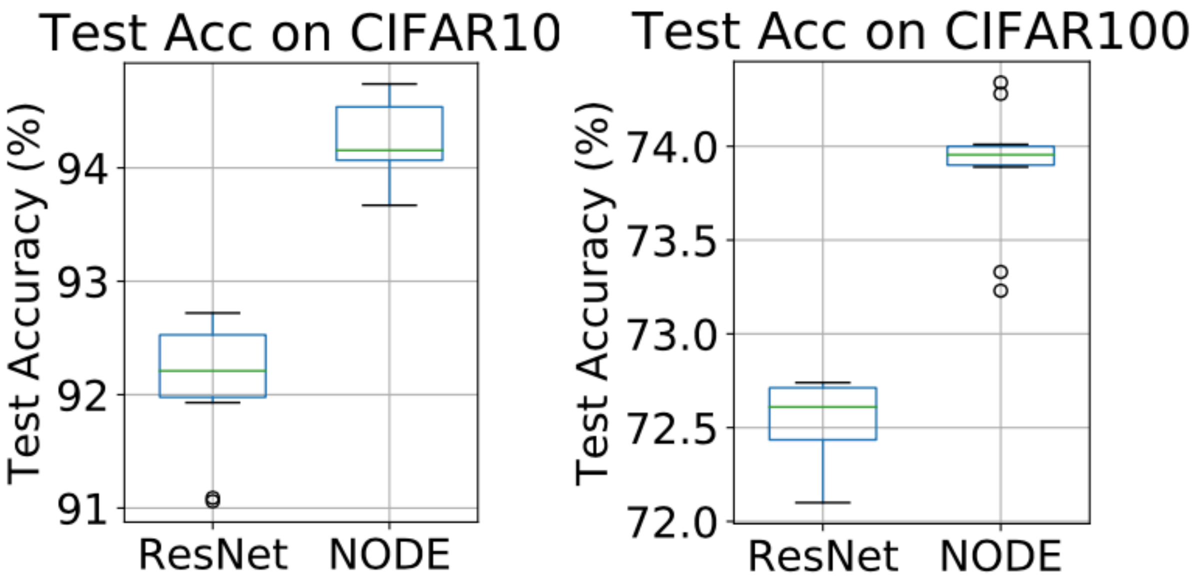
In a NODE block: $y = z(T) = z(0) + \int_0^T f(t, z) dt$, $z(0) = x$

f is the same for two types of blocks



Performance of NODE trained with different methods

Supervised image classification



Comparison between ResNet18 and NODE-18 on Cifar10 and Cifar100. We report the results of 10 runs for each model.

Code for ResNet is from: https://github.com/kuangliu/pytorch-cifar

Supervised image classification

Error rate on test set of Cifar10

Dataset	NODE18-ACA						NODE18 - adjoint	NODE18 - naive	ANODE18	ResNet		
	Adaptive Stepsize Solvers			Fixed Stepsize Solvers						ResNet18	ResNet50	ResNet101
	HeunEuler	RK23	RK45	Euler	RK2	RK4						
CIFAR10	**4.85**	4.92	5.29	5.52	5.27	5.24	9.8 (*19)	9.3	6.8	*6.98	*6.38	*6.25
CIFAR100	**22.66**	24.13	23.56	24.44	24.44	24.43	30.6 (*37)	29.4	22.7	*27.08	*25.73	*24.84

Results reported in the literature are marked with *

- We trained a NODE18 with ACA and Heun-Euler ODE solver.
- NODE-ACA generates the best overall performance (NODE-18 outperforms ResNet-101).
- NODE is robust to ODE solvers. During test, we used different ODE solvers ***without*** re-training, and still achieve comparable results

Time series modeling for irregularly sampled data

Percentage of Training Data	RNN	RNN-GRU	Latent-ODE		
			adjoint	naive	ACA
10%	*2.45	*1.97	0.47	*0.36	**0.31**
20%	*1.71	*1.42	0.44	*0.30	**0.27**
50%	*0.79	*0.75	0.40	*0.29	**0.26**

Table 4. Test MSE ($\times 10^{-2}$) for irregularly sampled time series data on *Mujoco* dataset. * are reported by Rubanova et al. (2019).

Incorporate physical knowledge into modeling

Three-body problem:

Consider three planets (simplified as ideal mass points) interacting with each other, according to Newton's law of motion and Newton's law of universal gravitation (Newton, 1833).

$$\ddot{\mathbf{r}}_{\mathbf{i}} = -\sum_{j \neq i} G m_j \frac{\mathbf{r}_{\mathbf{i}} - \mathbf{r}_{\mathbf{j}}}{|\mathbf{r}_{\mathbf{i}} - \mathbf{r}_{\mathbf{j}}|^3}$$

where G is the gravitation constant; $\mathbf{r}_{\mathbf{i}}$ is the location for planet i, each is of dimension 3; $\ddot{\mathbf{r}}_{\mathbf{i}}$ is the 2nd derivative $w.r.t$ time; m_i is the mass of planet i.

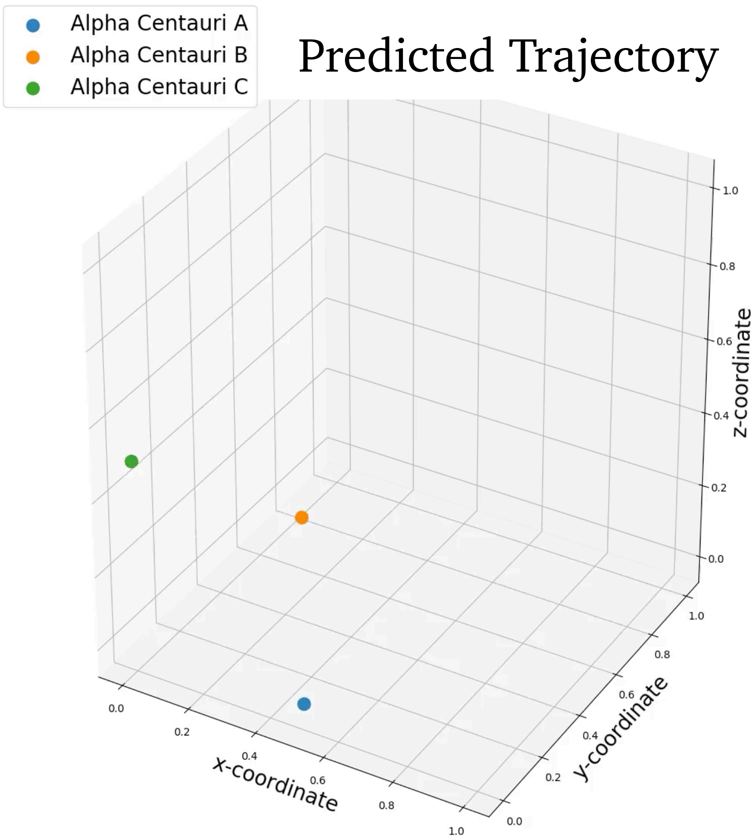
Problem definition:
given observations of trajectory $\boldsymbol{r}_{\boldsymbol{i}}(\boldsymbol{t}), t \in [0, T]$, predict future trajectories $\boldsymbol{r}_{\boldsymbol{i}}(\boldsymbol{t}), t \in [T, 2T]$, when mass m_i is unknown.

Incorporate physical knowledge into modeling

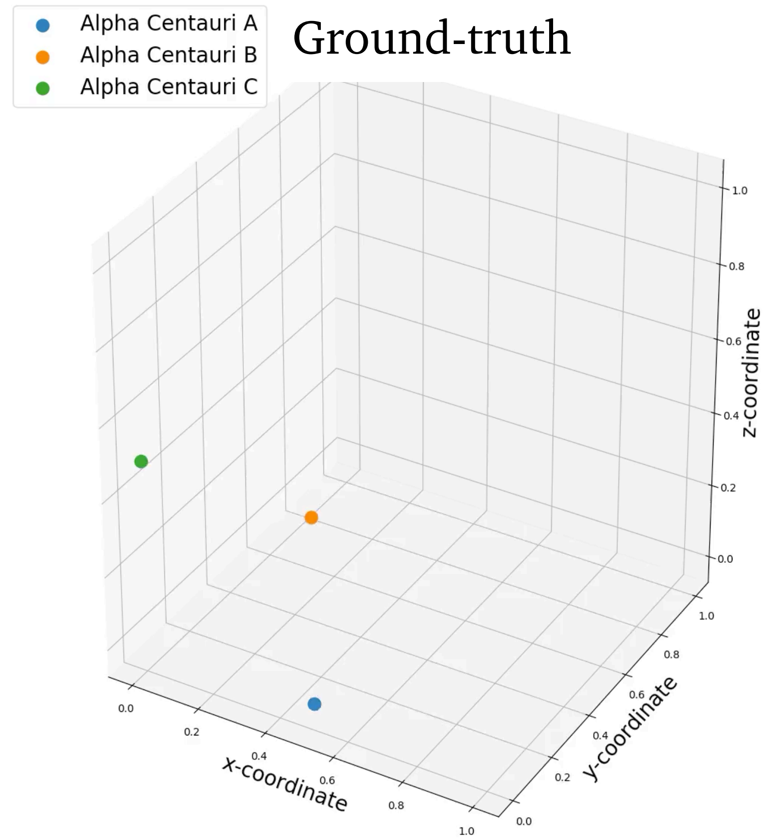
	LSTM	LSTM-aug-input	NODE			ODE		
			adjoint	naive	ACA	adjoint	naive	ACA
Test MSE	0.59+-0.12	0.49+-0.06	3.47+-0.67	0.21+-0.11	**0.16+-0.06**	0.0025+-0.0012	0.0025+-0.0013	**0.0007+-0.0005**

Table 5. Results of 3 runs for three-body problem. Training data time range is [0,1] year, MSE is measured on range [0,2] years.

Predicted Trajectory



Ground-truth



Conclusions

- We identify the numerical error with adjoint method to train NODE.

- We propose Adaptive Checkpoint Adjoint to accurately estimate the gradient in NODE.

In experiments, we demonstrate NODE training with ACA is both fast and accurate. To our knowledge, it's the first time for NODE to achieve ResNet-level accuracy on image classification.

- We provide a ***PyTorch*** package https://github.com/juntang-zhuang/torch_ACA, which can be easily plugged into existing models, with support for ***multi-GPU training*** and ***higher-order derivative***.
  (Reach out by email: *j.zhuang@yale.edu* or twitter: *JuntangZhuang*)

```
from torch_ACA import odesolve
options.update({'t_eval': [a1, a2, a3, ... an]})
out = odesolve(odefunc, x, options)
```