Appendix A, ACA as an AutoDiff Function in PyTorch Style

Forward $(f, T, z_0, tolerance)$:

$t = 0, z = z_0$

$state_0 = f.state\_dict()$, cache.save($state_0$)

Select initial step size $h = h_0$ (adaptively with adaptive step-size solver).

$time\_points = empty\_list()$

$z\_values = empty\_list()$

While $t < T$:

$state = f.state\_dict(), \;\; accept\_step = False$

While Not $accept\_step$:

$f.load\_state\_dict(state)$

with $grad\_disabled$:

$z\_new, error\_estimate = \psi(f, z, t, h)$

If $error\_estimate < tolerance$:

$accept\_step = True$

$z = z\_new, \;\; t = t + h,$

$z\_values.append(z), \;\; time\_points.append(t)$

else:

reduce stepsize h according to $error\_estimate$

delete $error\_estimate$ local computation graph

cache.save($time\_points, \;\; z\_values$)

return z, cache

Backward $(f, T, tolerance, cache, \frac{\partial J}{\partial z(T)})$:

Initialize $\lambda = -\frac{\partial J}{\partial z(T)}, \;\; \frac{\partial L}{\partial \theta} = 0$

$\{z_0, z_1, z_2, ...z_{N-1}, z_N\} = cache.z\_values$

$\{t_0, t_1, t_2, ...t_{N-1}, t_N\} = cache.time\_points$

For t_i in $\{t_N, t_{N-1}, ..., t_1, t_0\}$:

Local forward $\hat{z}_i = \psi(f, z_{i-1}, t_{i-1}, h_i = t_i - t_{i-1})$

Local backward

$\frac{\partial L}{\partial \theta} \leftarrow \frac{\partial L}{\partial \theta} - \lambda^\top \frac{\partial \hat{z}_i}{\partial \theta}$

$\lambda \leftarrow \lambda^\top \frac{\partial \hat{z}_i}{\partial z_{i-1}}$

delete local computation graph

return $\frac{\partial L}{\partial \theta}, \;\; \lambda$

Appendix B, Proof of Theorem 3.2

We refer readers to Fig. 2 in the main paper for a graphical interpretation, and Sec. 2.3 for a list of notations.

Lemma 0.1 *Suppose f is composed of a finite number of ReLU activations and linear transforms,*

$$f(t,z) = Linear_1 \circ ReLU \circ Linear_2 \circ ... \circ Linear_N(z)$$

if the spectral norm of linear transform is bounded, then the IVP defined above has a unique solution on a bounded region.

Proof: f does not depend on t explicitly, hence is continuous in t. ReLU (and other activation functions such as sigmoid, tanh, ...) is uniformly continuous; a linear transform Wz is also uniformly continuous if the spectral norm of W is bounded. From Picard-Lindelöf Theorem, the IVP has a unique solution on a bounded region.

Flow map Denote $\Phi_{t_0}^T(z_0)$ as the *oracle* solution to the IVP at time T, with the initial condition (t_0, z_0). Then $\Phi_{t_0}^T(z_0)$ satisfies:

$$\Phi_{t_2}^{t_3} \circ \Phi_{t_1}^{t_2} = \Phi_{t_1}^{t_3} \tag{1}$$

$$\frac{d}{dt}\Phi_{t_0}^t(z_0) = f(t, \Phi_{t_0}^t(z_0)) \tag{2}$$

$$\Phi_{t_0}^t(z_0) = z_0 + \int_{t_0}^{t} f\big(s, \Phi_{t_0}^s(z_0)\big) ds \tag{3}$$

Variational flow The derivative $w.r.t$ initial condition is called the *variational flow*, denoted as $D\Phi_{t_0}^t$, then it satisfies:

$$D\Phi_{t_0}^t(z_0) = \frac{d\Phi_{t_0}^t(z_0)}{dz_0}, \;\; D\Phi_{t_0}^{t_0} = I \tag{4}$$

$$D\Phi_{t_0}^{t_0+h} = I + O(h), \; if \; h \; is \; small. \tag{5}$$

From Eq. 1 and 5, using the chain rule, we have:

$$D\Phi_{t_0}^t(z_0) = \frac{d\Phi_{t_0}^t(z_0)}{dz_0} = \frac{d\Phi_{t_0+h}^t(\Phi_{t_0}^{t_0+h}(z_0))}{d\Phi_{t_0}^{t_0+h}(z_0)} \frac{d\Phi_{t_0}^{t_0+h}(z_0)}{dz_0} = D\Phi_{t_0+h}^t + O(h) \tag{6}$$

Local truncation error Denote the step function of a one-step ODE solver as $\psi_h(t,z)$, with step-size h starting from (t,z). Denote the local truncation error as:

$$L_h(t,z) = \psi_h(t,z) - \Phi_t^{t+h}(z) \tag{7}$$

For a solver of order p, the error is of order $O(h^{p+1})$, and can be written as

$$L_h(t,z) = h^{p+1} l(t,z) + O(h^{p+2}) \tag{8}$$

where l is some function of order $O(1)$.

Global error Denote the global error as $G(T)$ at time T, then it satisfies:

$$G(T) = z_N - \Phi_{t_0}^T(z_0) = \sum_{k=0}^{N-1} R_k \tag{9}$$

where

$$R_k = \Phi_{t_{k+1}}^T(z_{k+1}) - \Phi_{t_k}^T(z_k) \tag{10}$$

$$= \Phi_{t_{k+1}}^T\big(\Phi_{t_k}^{t_{k+1}}(z_k) + L_{h_k}(t_k, z_k)\big) - \Phi_{t_{k+1}}^T(\Phi_{t_k}^{t_{k+1}}(z_k)) \tag{11}$$

Lemma 0.2 (Approximation of R_k)

$$R_k = D\Phi^T_{t_{k+1}}\big(\Phi^{t_{k+1}}_{t_k}(z_k)\big)L_{h_k}(t_k, z_k) + O(h_k^{2p+2}) \tag{12}$$

Lemma 0.2 can be viewed as a Taylor expansion of Eq. 11, with detailed proof in (Niesen et al., 2004).

Lemma 0.3 *If $L_h(t, y) = O(h^{p+1})$, then $G_h(T) = O(h^p)$*

Proof for Lemma 0.3 is in (Niesen et al., 2004).

Plug Eq. 8 and Eq. 6 into Eq. 12, we have

$$R_k = \big[D\Phi^T_{t_k}(z_k) + O(h_k)\big]L_{h_k}(t_k, z_k) + O(h_k^{2p+2}) \tag{13}$$

$$= \big[D\Phi^T_{t_k}(z_k) + O(h_k)\big]\big[h_k^{p+1}l(t_k, z_k) + O(h_k^{p+2})\big] + O(h_k^{2p+2}) \tag{14}$$

$$= h_k^{p+1}D\Phi^T_{t_k}(z_k)l(t_k, z_k) + O(h_k^{p+2}) \tag{15}$$

Plug Eq. 15 into Eq. 9, then we have:

$$G(T) = \sum_{k=0}^{N-1} R_k = \sum_{k=0}^{N-1}\big[h_k^{p+1}D\Phi^T_{t_k}(z_k)l(t_k, z_k) + O(h_k^{p+2})\big] \tag{16}$$

$$= \sum_{k=0}^{N-1}\big[h_k^{p+1}D\Phi^T_{t_k}(z_k)l(t_k, z_k)\big] + O(h_{max}^{p+1}) \tag{17}$$

Global error of the adjoint method If we solve an IVP forward-in-time from $t = 0$ to T, then take $z(T)$ as the initial condition, and solve it backward-in-time from T to 0, the numerical error can be written as:

$$G(t_0 \to T \to t_0) = \sum_{k=0}^{N_t-1}\big[h_k^{p+1}D\Phi^T_{t_k}(z_k)l(t_k, z_k)\big] + \sum_{J=0}^{N_r-1}\big[(-h_j)^{p+1}D\Phi^{\tau_j}_T(\overline{z_j})\overline{l(\tau_j, \overline{z_j})}\big] + O(h_{max}^{p+1}) \tag{18}$$

$$= G(t_0 \to T) + G(T \to t_0) + O(h^{p+1}) \tag{19}$$

where $G(t_0 \to T)$ represents the numerical error of forward-in-time (t_0 to T) solution (discretized at step k, denoted as z_k); and $G(T \to t_0)$ denotes the numerical error of reverse-in-time solution (T to t_0) (discretized at step j, denoted as $\overline{z_j}$). $G(t_0 \to T \to t_0)$ represents the error in reconstructed initial condition by the adjoint method. Note that generally z does not overlap with $\overline{z}$. The local error of forward-in-time and reverse-in-time numerical integration is represented as l and $\overline{l}$ respectively.

Although going backward is equivalent to a negative stepsize, which might cause the second term to have different signs compared to the first term in Eq. 18, we demonstrate that generally their sum cannot cancel.

For the ease of analysis, we assume the forward and reverse-in-time calculation are discretized at the same grid points, with a sufficiently small constant stepsize (For a variable-stepsize solver, we can modify it to a constant-stepsize solver, whose stepsize is the minimal step in variable-stepsize solver. With this modification, the constant stepsize solver should be *no worse than* adaptive stepsize solver). Then Eq. 18 can be written as:

$$G(t_0 \to T \to t_0) = \sum_{k=0}^{N-1}\big[h_k^{p+1}D\Phi^T_{t_k}(z_k)l(t_k, z_k)\big] + \sum_{k=0}^{N-1}\big[(-h_k)^{p+1}D\Phi^{t_k}_T(\overline{z_k})\overline{l(t_k, \overline{z_k})}\big] + O(h_{max}^{p+1}) \tag{20}$$

$$= \sum_{k=0}^{N-1}\big[h_k^{p+1}D\Phi^T_{t_k}(z_k)l(t_k, z_k) + (-h_k)^{p+1}D\Phi^{t_k}_T(\overline{z_k})\overline{l(t_k, \overline{z_k})}\big] + O(h^{p+1}) \tag{21}$$

If the stepsize is sufficiently small, we can assume

$$z_k = \overline{z_k} + O(h) \tag{22}$$

$$D\Phi_T^{t_k}(z_k) = D\Phi_T^{t_k}(\overline{z_k}) + O(h) \tag{23}$$

$$l(t_k, z_k) = \overline{l(t_k, \overline{z_k})} + O(h) \tag{24}$$

Assume the existence and uniqueness conditions are satisfied on $t \in [0, T]$, so $\Phi_{t_0}^T$ defines a homeomorphism, hence:

$$D\Phi_T^{t_k} = (D\Phi_{t_k}^T)^{-1} \tag{25}$$

Plug Eq. 22 to Eq. 25 into Eq. 21, we have

$$G(t_0 \to T \to t_0) = \sum_{k=0}^{N-1} h^{p+1} l(t_k, z_k) e_k + O(h^{p+1}) \tag{26}$$

$$e_k = D\Phi_{t_k}^T(z_k) + (-1)^{p+1}(D\Phi_{t_k}^T(z_k))^{-1} \tag{27}$$

Reverse accuracy for all t_0 requires $e_k = 0$ for all k. If p is odd, then the two terms in e_k are the same sign, and thus cannot cancel to 0; if p is even, then $e_k = 0$ requires $D\Phi_{t_k}^T(z_k) = D\Phi_{t_k}^T(z_k)^{-1} = I$, which is generally not satisfied with a trained network (otherwise the network is an identity function with a constant bias).

In short, solving an IVP from t_0 to T with $z(0) = z_0$, then taking $z(T)$ as initial condition and solving it from T to t_0 and getting $\overline{z(0)}$, generally $z(0) \neq \overline{z(0)}$ because of numerical errors.

Appendix C. Proof of Theorem 2.1

In this section we derive the gradient in NODE from an optimization perspective.

Notations With the same notations as in the main paper, we use $z(t)$ to denote hidden states z at time t. Denote parameters of f as θ, and input as x, target as y, and predicted output as $\hat{y}$. Denote the loss as $J(\hat{y}, y)$. Denote the integration time as 0 to T.

Problem setup The continuous model is defined to follow an ODE:

$$\frac{dz(t)}{dt} = f(z(t), t, \theta), \quad s.t. \;\; z(0) = x \tag{28}$$

We assume f is differentiable almost everywhere, since f is represented by a neural network in our case. The forward pass is defined as:

$$\hat{y} = z(T) = z(0) + \int_0^T f(z(t), t, \theta) dt \tag{29}$$

The loss function is defined as:

$$J(\hat{y}, y) = J(z(T), y) \tag{30}$$

We formulate the training process as an optimization problem:

$$\underset{\theta}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} J(\hat{y}_i, y_i) \;\; s.t. \;\; \frac{dz_i(t)}{dt} = f(z_i(t), t, \theta), \;\; z_i(0) = x_i, \;\; i = 1, 2, ...N \tag{31}$$

For simplicity, Eq. 31 only considers one ODE block. In the case of multiple blocks, $z(T)$ is the input to the next ODE block. As long as we can derive $\frac{dLoss}{d\theta}$ and $\frac{dLoss}{dz(0)}$ when $\frac{dLoss}{dz(T)}$ is given, the same analysis here can be applied to the case with a chain of ODE blocks.

Lagrangian Multiplier Method We use the Lagrangian Multiplier Method to solve the problem defined in Eq. 31. For simplicity, only consider one example (can be easily extended to the multiple examples case), then the Lagrangian is

$$L = J(z(T), y) + \int_0^T \lambda(t)^\top [\frac{dz(t)}{dt} - f(z(t), t, \theta)]dt \tag{32}$$

Karush-Kuhn-Tucker (KKT) conditions are necessary conditions for a solution to be optimal. In the following sections we start from the KKT conditions and derive our results.

Derivative w.r.t. λ At optimal point, we have $\frac{\delta L}{\delta \lambda} = 0$. Note that λ is a function of t, and we derive the derivative from calculus of variation.

Consider a continuous and differentiable perturbation $\overline{\lambda(t)}$ on $\lambda(t)$, and a scalar ϵ, L now becomes a function of ϵ,

$$L(\epsilon) = J\big(z(0) + \int_0^T f(z(t), t, \theta), y\big) + \int_0^T (\lambda(t) + \epsilon\overline{\lambda(t)})^\top [\frac{dz(t)}{dt} - f(z(t), t, \theta)]dt \tag{33}$$

It's easy to check the conditions for Leibniz integral rule, and we can switch integral and differentiation, thus:

$$\frac{dL}{d\epsilon} = \int_0^T \overline{\lambda(t)}^\top [\frac{dz(t)}{dt} - f(z(t), t, \theta)]dt \tag{34}$$

At optimal $\lambda(t)$, $\frac{dL}{d\epsilon}|_{\epsilon=0} = 0$ for all continuous differentiable $\overline{\lambda(t)}$.

Therefore,

$$\frac{dz(t)}{dt} - f(z(t), t, \theta) = 0, \ \ \forall t \in (0, T) \tag{35}$$

Derivative w.r.t z Consider perturbation $\overline{z(t)}$ on $z(t)$, with scale ϵ. With similar analysis:

$$L(\epsilon) = J(z(T) + \epsilon\overline{z(T)}, y) + \int_0^T \lambda(t)^\top [\frac{dz(t) + \epsilon\overline{z(t)}}{dt} - f(z(t) + \epsilon\overline{z(t)}, t, \theta)]dt \tag{36}$$

Take derivative w.r.t ϵ, it's easy to check conditions for Leibniz integral rule are satisfied, when f and $\overline{z(t)}$ are Lipschitz continuous differentiable functions:

(1) $f(z(t), t, \theta)$ is a Lebesgue-integrable function of θ for each $z(t) \in \mathbf{R}^d$, since we use a neural network to represent f, which is continuous and differentiable almost everywhere.

(2) for almost all θ, $\frac{\partial f(z(t),t,\theta)}{\partial z(t)}$ exists for almost all $x \in \mathbf{R}^d$.

(3) $\frac{\partial f(z(t),t,\theta)}{\partial z(t)}$ is bounded by $g(\theta)$ for all $z(t)$ for almost all θ.

Then we calculate $\frac{dL(\epsilon)}{d}$, note that we can switch integral and derivative:

$$\frac{dL}{d\epsilon}|_{\epsilon=0} = \frac{\partial J}{\partial z(T)}^\top \overline{z(T)} + \frac{d}{d\epsilon}\int_0^T \lambda(t)^\top \big[\frac{dz(t)+\epsilon\overline{z(t)}}{dt} - f(z(t)+\epsilon\overline{z(t)}, t, \theta)\big]dt \tag{37}$$

$$= \frac{\partial J}{\partial z(T)}^\top \overline{z(T)} + \int_0^T \lambda(t)^\top \big[\frac{d\overline{z(t)}}{dt} - \frac{\partial f(z(t),t,\theta)}{\partial z(t)}\overline{z(t)}\big]dt \tag{38}$$

$$= \frac{\partial J}{\partial z(T)}^\top \overline{z(T)} + \int_0^T \big[\lambda(t)^\top \frac{d\overline{z(t)}}{dt} + \frac{d\lambda(t)}{dt}^\top \overline{z(t)} - \frac{d\lambda(t)}{dt}^\top \overline{z(t)} - \lambda(t)^\top \frac{\partial f(z(t),t,\theta)}{\partial z(t)}\overline{z(t)}\big]dt \tag{39}$$

$$= \frac{\partial J}{\partial z(T)}^\top \overline{z(T)} + \lambda(t)^\top \overline{z(t)}|_0^T - \int_0^T \overline{z(t)}^\top \big[\frac{d\lambda(t)}{dt} + \frac{\partial f(z(t),t,\theta)}{\partial z(t)}^\top \lambda(t)\big]dt \tag{40}$$

$$= \frac{\partial J}{\partial z(t)}^\top \overline{z(T)} + \lambda(t)^\top \overline{z(T)} - \lambda(0)^T \overline{z(0)} - \int_0^T \overline{z(t)}^\top \big[\frac{d\lambda(t)}{dt} + \frac{\partial f(z(t),t,\theta)}{\partial z(t)}^\top \lambda(t)\big]dt \tag{41}$$

$$= \big(\frac{\partial J}{\partial z(T)} + \lambda(T)\big)^\top \overline{z(T)} - \lambda(0)^T \overline{z(0)} - \int_0^T \overline{z(t)}^\top \big[\frac{d\lambda(t)}{dt} + \frac{\partial f(z(t),t,\theta)}{\partial z(t)}^\top \lambda(t)\big]dt \tag{42}$$

Since the initial condition $z(0) = x$ is given, perturbation $\overline{z(0)}$ at $t = 0$ is 0, then we have:

$$\frac{dL}{d\epsilon}|_{\epsilon=0} = \big(\frac{\partial J}{\partial z(T)} + \lambda(T)\big)^\top \overline{z(T)} - \int_0^T \overline{z(t)}^\top \big[\frac{d\lambda(t)}{dt} + \frac{\partial f(z(t),t,\theta)}{\partial z(t)}^\top \lambda(t)\big]dt = 0 \tag{43}$$

for any $\overline{z(t)}$ $s.t.$ $\overline{z(0)} = 0$ and $\overline{z(t)}$ is differentiable.

The solution is:

$$\frac{\partial J}{\partial z(T)} + \lambda(T) = 0 \tag{44}$$

$$\frac{d\lambda(t)}{dt} + \frac{\partial f(z(t),t,\theta)}{\partial z(t)}^\top \lambda(t) = 0 \ \ \forall t \in (0,T) \tag{45}$$

Derivative w.r.t θ From Eq. 32,

$$\frac{dL}{d\theta} = -\int_0^T \lambda(t)^\top \frac{\partial f(z(t),t,\theta)}{\partial \theta} dt \tag{46}$$

To sum up, we first solve the ODE forward-in-time with Eq. 28, then determine the boundary condition by Eq. 44, then solve the ODE backward with Eq. 45, and finally calculate the gradient with Eq. 46. In fact λ corresponds to the negative $adjoint\ variable$.

Appendix D, Experimental Details

1. Experiments with van der Pol Equation

For experiments with the *van der Pol equation*, the ODE is defined as:

$$\frac{dy_1}{dt} = y_2 \tag{47}$$

$$\frac{dy_2}{dt} = (0.15 - y_1^2) \times y_2 - y_1 \tag{48}$$

with initial condition $y_1(0) = 2, y_2(0) = 0$. Experiments are performed in MATLAB with *ode45* solver under default settings. Results are shown as follow:

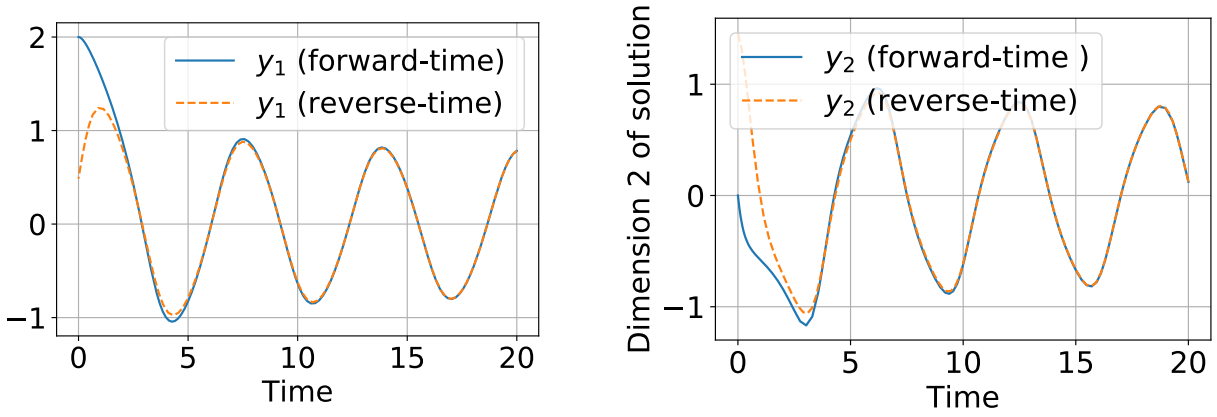

Figure 1. Results on simulation with van der Pol equation.

2. Experiments on supervised image classification

Experimental settings All experiments were performed with PyTorch 0.4.1 on a single GTX-1080Ti GPU. To generate Fig. 7 (a) and (b) in the main paper, we trained a NODE18 model for 90 epochs, with the initial learning rate 0.01 and decayed by a factor of 0.1 at epoch 30 and 60. Training images were augmented with random crop and horizontal flip. For ACA, we used *HeunEuler* solver for training, with $rtol = 10^{-2}$ and $atol = 10^{-2}$; for the adjoint and naive method, we used the solver implemented by (Chen et al., 2018) (`https://github.com/rtqichen/torchdiffeq`), with a *Dopri5* solver, setting $rtol = 10^{-5}$ and $atol = 10^{-5}$; we tried larger error tolerance (10^{-2}) for the adjoint method, but it led to divergence during training. Batchsize is set as 128 for ACA and the adjoint method, and 32 for the naive method.

To generate Fig. 7(c) and (d), and Table 3 in the main paper, we trained NODE18-ACA and ResNet with the following settings: initial learning rate is 0.1, decayed by 0.1 at epoch 30 and 60, total training epoch is 90; batchsize is set as 128.

To generate Table 2 in the main paper, we trained NODE18-ACA for 350 epochs, with initial learning rate 0.1, and decayed by 0.1 at epoch 150 and 250.

Extra experiments on impact of model depth We performed experiments on CIFAR10 dataset to measure the influence of model depth. We trained models using the same settings as described above, and tested with different solvers *without* re-training.

During test, constant stepsize solvers using different stepsizes are equivalent to different model depths, for example, with a stepsize of 0.2, the model depth is 5 times deeper than with a stepsize of 1.0 ($\frac{1.0}{0.2} = 5$); higher-order solvers evaluates the function more times than low-order solvers, for example, using the same stepsize, RK4 evaluates 4 times wile RK2 evaluates twice at each step. Adaptive stepsize solvers evaluate the function using a finer grid for smaller error tolerance, hence a deeper computation graph.

Constant Stepsize Solvers					Adaptive Stepsize Solvers			
stepsize	1.0	0.5	0.2	0.1	rtol / atol	1e-1	1e-2	1e-3
Euler	**+0.0**	+1.14	+3.62	+4.84	HeunEuler	+5.32	+6.33	+6.42
RK2	+7.69	+6.43	+6.38	+6.43	RK23	+6.03	+6.31	+6.44
RK4	+5.69	+6.31	+6.30	+6.47	RK45	+6.30	+6.47	+6.46

Table 1. **Increase in error rate** of a ResNet18 (equivalently, NODE using 1-step Euler method with integration time [0,1]) when tested with different solvers. When trained and tested with the same method, the error rate is 8.47%, the increase in error rate is 0 as bold fonted. When tested with different solvers **without** re-training, the increase in error rate is reported, with a **smaller** difference represents better robustness.

Constant Stepsize Solvers					Adaptive Stepsize Solvers			
stepsize	1.0	0.5	0.2	0.1	rtol / atol	1e-1	1e-2	1e-3
Euler	+8.31	+1.57	+0.67	+0.57	HeunEuler	+1.29	**+0.0**	+0.18
RK2	+6.61	+0.57	+0.42	+0.39	RK23	+0.46	+0.07	+0.40
RK4	+1.09	+0.48	+0.39	+0.37	RK45	+1.75	+0.44	+0.16

Table 2. **Increase in error rate** of a NODE18 when trained using HeunEuler with $rtol = atol = 10^{-2}$ and tested with different solvers. When trained and tested with the same method, the error rate is 4.85%, the increase in error rate is 0 as bold fonted. When tested with different solvers **without** re-training, the increase in error rate is reported, with a **smaller** difference represents better robustness.

To sum up, depth of the computation graph is determined by both the *stepsize* and *order* for constant stepsize solvers, and determined by *error tolerance* and *order* for adaptive stepsize solvers.

We performed experiments on a ResNet18; equivalently, ResNet18 is NODE18 using one-step Euler solver, with integration time $[0, 1]$. We also experimented with a NODE18, trained with HeunEuler solver with $rtol = etol = 10^{-2}$. Results for their performance using different solvers *without* re-training are summarized in Table. 1 and Table. 2.

NODE generally achieves a lower error rate than ResNet (4.85% v.s 8.47% when trained and tested using the same method). Ignoring the absolute value of error rate, to measure the robustness to solvers, on the *same* model, we focus on the *increase* in test error rate using different methods compared to using the same method as training.

For ResNet, when tested with different solvers, most results have a $\sim 7\%$ increase in error rate; for NODE when trained with HeunEuler with $rtol = 10^{-2}$, and tested with different methods, most results have a $\sim 1\%$ increase in error rate. This results show that training as ResNet is sensitive to model depth during test; while training as NODE with adaptive solvers are robust to different solvers (hence different model depth) during test.

3. Experiments on time-series modeling with irregularly sampled data

We performed experiments using the official implementation by (Rubanova et al., 2019) (`https://github.com/YuliaRubanova/latent_ode`). All models are trained for 300 epochs on the *Mujoco* dataset provided by (Rubanova et al., 2019). For the ease of visualization, we plot the test MSE curve for epochs 0-100.

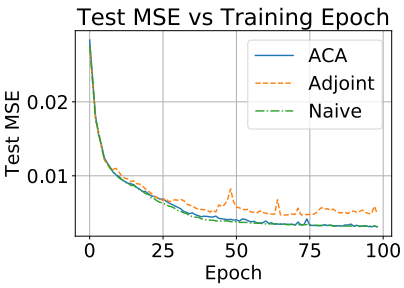

Figure 2. 10% training data

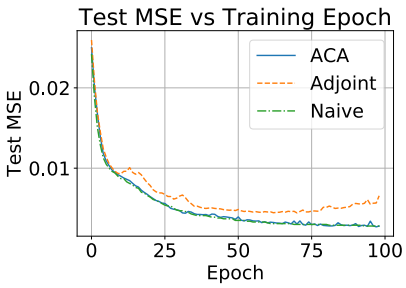

Figure 3. 20% training data

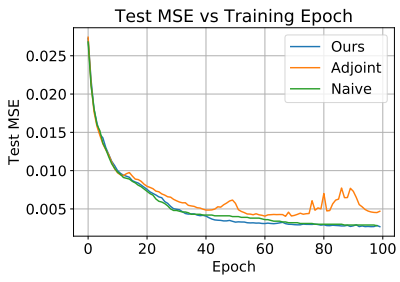

Figure 4. 50% training data

4. Experiments on the three-body problem

We summarize the training details in the following table. All models are trained with Adam optimizer. Learning rate schedule is:

$$lr = InitialLR \times decay^{epoch} \tag{49}$$

For all LSTM models, initial learning rate is 0.01 with a $decay$ of 0.999, trained for 5,000 epochs; all NODEs are trained with initial learning rate 0.1 for 100 epochs, with $decay$ 0.99. We set a much larger epoch and smaller learning rate for LSTM, because we found in practice the training of LSTM is much harder to converge. For adjoint and naive method, we use *Dopri5* solver by (Chen et al., 2018) with $rtol = atol = 10^{-5}$; for ACA, we implemented $Dopri5$ solver with $rtol = atol = 10^{-5}$.

We simulate a 3-body system with unequal mass and arbitrary initial condition, use time range [0,1] year for training, and measure the mean MSE of trajectory on [0,2] years. Training data has 1,000 equally sampled points, cut into sequences of 20 points as input to LSTM models. During inference, 1 initial point is fed to NODE and ODE, and first 10 points are fed to LSTM. Results are shown in figures below and videos in the supplementary material, with numerical measures in the main paper.

	LSTM	LSTM-aug-input	NODE			ODE		
			adjoint	naive	ACA	adjoint	naive	ACA
Epoch	5,000	5,000	100	100	100	100	100	100
InitialLR	0.01	0.01	0.1	0.1	0.1	0.1	0.1	0.1
decay	0.999	0.999	0.99	0.99	0.99	0.99	0.99	0.99

Table 3. Training details for the three-body problem

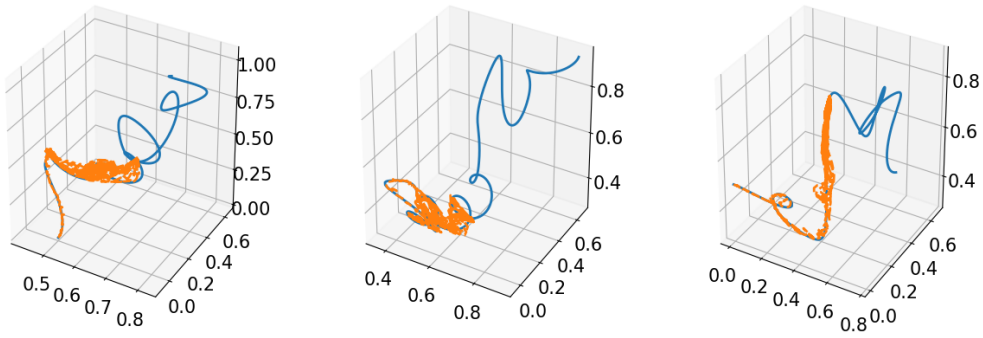

Figure 5. Fitted trajectory (orange dashed) and ground truth (blue solid) in 3D space, from left to right are results for 3 planets. Results for LSTM model.

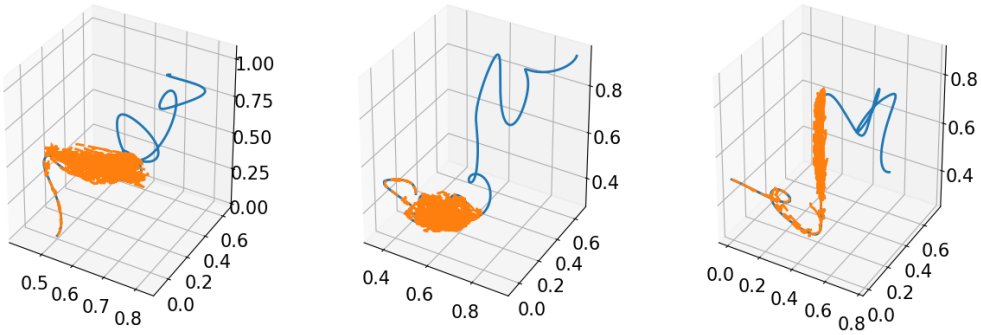

Figure 6. Fitted trajectory (orange dashed) and ground truth (blue solid) in 3D space, from left to right are results for 3 planets. Results for LSTM-aug-input.

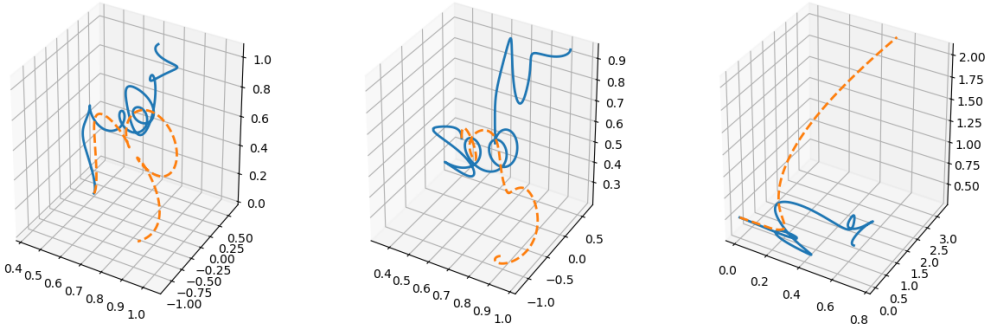

Figure 7. Fitted trajectory (orange dashed) and ground truth (blue solid) in 3D space, from left to right are results for 3 planets. Results for NODE-adjoint.

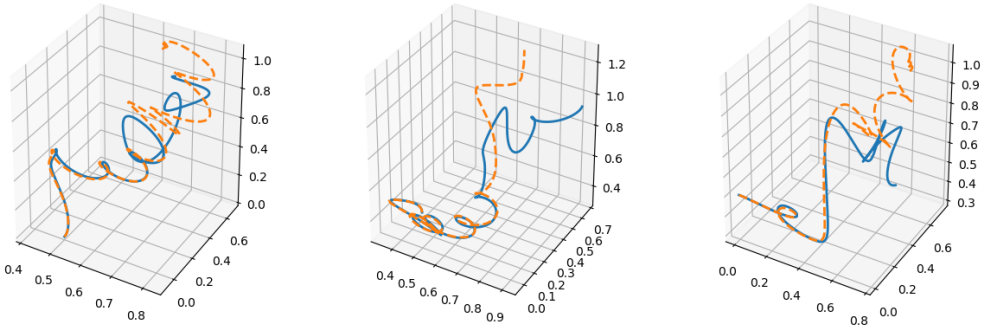

Figure 8. Fitted trajectory (orange dashed) and ground truth (blue solid) in 3D space, from left to right are results for 3 planets. Results for NODE-naive.

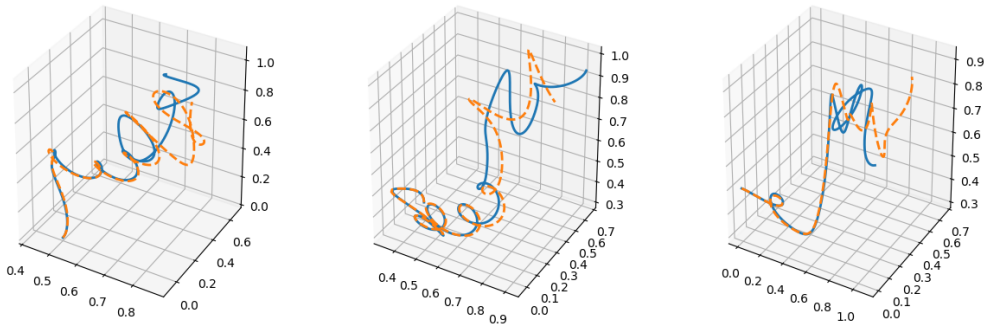

Figure 9. Fitted trajectory (orange dashed) and ground truth (blue solid) in 3D space, from left to right are results for 3 planets. Results for NODE-ACA.

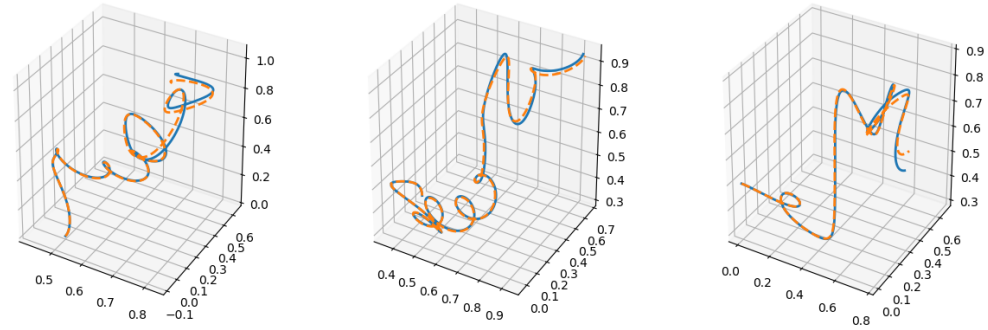

Figure 10. Fitted trajectory (orange dashed) and ground truth (blue solid) in 3D space, from left to right are results for 3 planets. Results for ODE-adjoint.

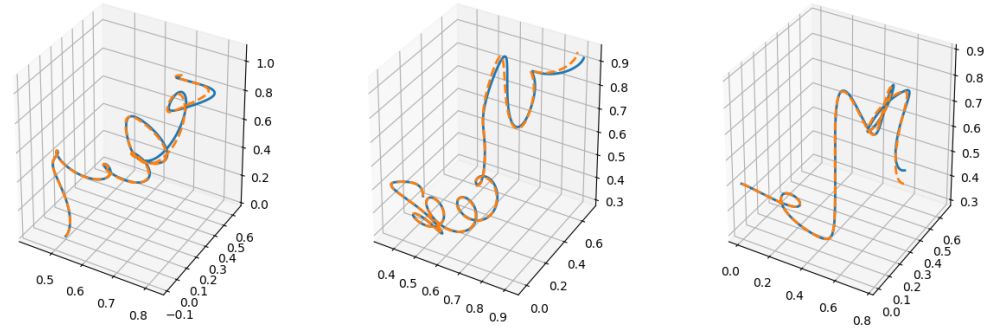

Figure 11. Fitted trajectory (orange dashed) and ground truth (blue solid) in 3D space, from left to right are results for 3 planets. Results for ODE-naive.

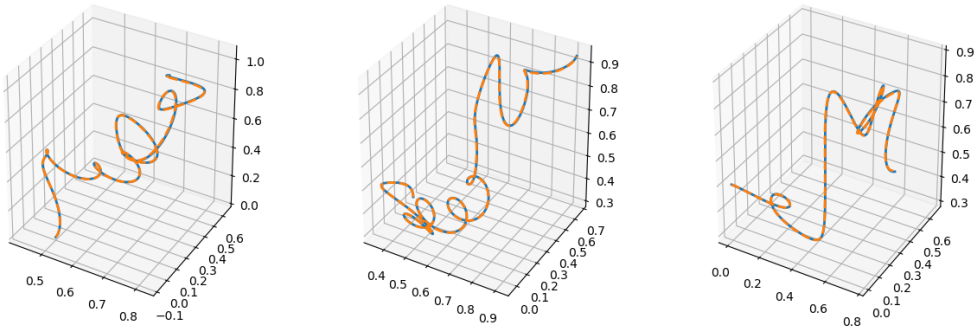

Figure 12. Fitted trajectory (orange dashed) and ground truth (blue solid) in 3D space, from left to right are results for 3 planets. Results for ODE-ACA.

More results of ODE-ACA with different initial conditions

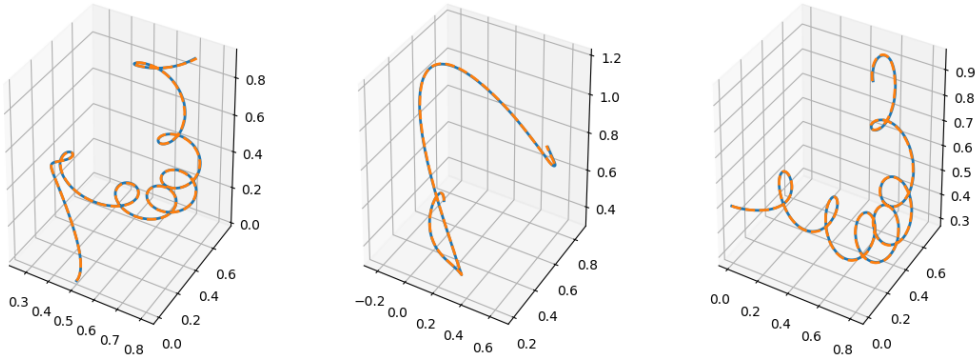

Figure 13. Fitted trajectory (orange dashed) and ground truth (blue solid) in 3D space, from left to right are results for 3 planets. Results for ODE-ACA.

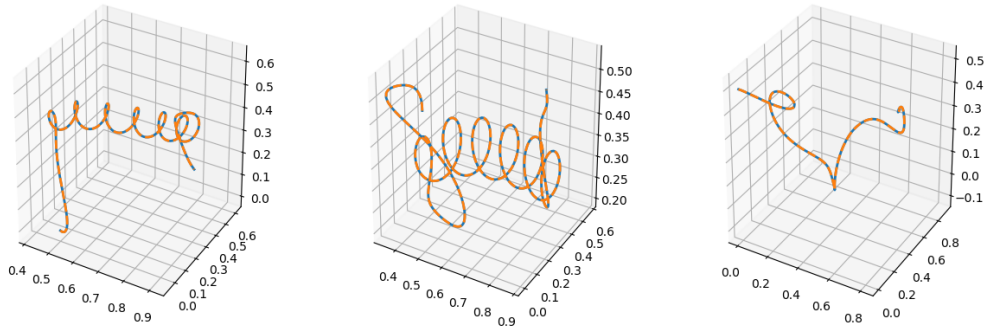

Figure 14. Fitted trajectory (orange dashed) and ground truth (blue solid) in 3D space, from left to right are results for 3 planets. Results for ODE-ACA.

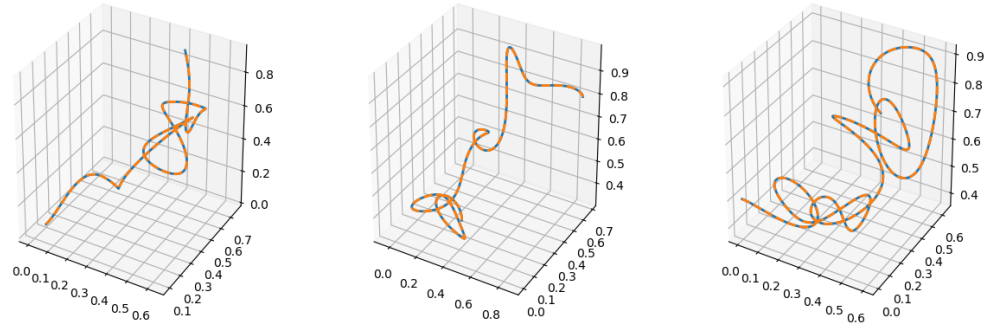

Figure 15. Fitted trajectory (orange dashed) and ground truth (blue solid) in 3D space, from left to right are results for 3 planets. Results for ODE-ACA.

References

Chen, T. Q. et al. Neural ordinary differential equations. In *Advances in neural information processing systems*, pp. 6571–6583, 2018.

Niesen, J. et al. On the global error of discretization methods for ordinary differential equations. In *Citeseer*, 2004.

Rubanova, Y. et al. Latent ordinary differential equations for irregularly-sampled time series. In *Advances in Neural Information Processing Systems*, pp. 5321–5331, 2019.