Adaptive Checkpoint Adjoint Method for Gradient Estimation in Neural ODE

Juntang Zhuang [1]   **Nicha Dvornek** [1 2]   **Xiaoxiao Li** [1]   **Sekhar Tatikonda** [3]   **Xenophon Papademetris** [1 2 4]
James Duncan [1 2 4]

Abstract

The empirical performance of neural ordinary differential equations (NODEs) is significantly inferior to discrete-layer models on benchmark tasks (e.g. image classification). We demonstrate an explanation is the inaccuracy of existing gradient estimation methods: the *adjoint* method has numerical errors in reverse-mode integration; the *naive* method suffers from a redundantly deep computation graph. We propose the *Adaptive Checkpoint Adjoint* (ACA) method: ACA applies a *trajectory checkpoint* strategy which records the forward-mode trajectory as the reverse-mode trajectory to guarantee accuracy; ACA deletes redundant components for shallow computation graphs; and ACA supports *adaptive* solvers. On image classification tasks, compared with the adjoint and naive method, ACA achieves half the error rate in half the training time; NODE trained with ACA outperforms ResNet in both accuracy and test-retest reliability. On time-series modeling, ACA outperforms competing methods. Furthermore, NODE with ACA can incorporate physical knowledge to achieve better accuracy.

1. Introduction

Conventional neural networks with discrete layers have achieved great success in various tasks, such as image classification (He et al., 2016), segmentation (Long et al., 2015) and machine translation (Sutskever et al., 2014). However, it's difficult for these discrete-layer networks to model continuous processes. The recently proposed neural ordinary differential equation (NODE) (Chen et al., 2018) views the model as an ordinary differential equation (ODE), whose derivative is parameterized by the network. NODE can be viewed as an initial value problem (IVP), whose initial condition is input to the model. NODE achieves great success in free-form reversible generative models (Grathwohl et al., 2018), time series analysis (Rubanova et al., 2019) and system identification (Quaglino et al., 2019; Ayed et al., 2019). However, the empirical performance of NODE is significantly inferior to discrete-layer models on benchmark classification tasks (error rate: 19% (NODE) vs 5% (ResNet) on CIFAR10) (Dupont et al., 2019; Gholami et al., 2019).

We demonstrate that performance is adversely affected by inaccurate gradient estimation for NODEs using existing methods. NODEs are typically trained with the *adjoint method* (Pontryagin, 1962; Chen et al., 2018), which is memory-efficient but sensitive to numerical errors; because the forward-mode and reverse-mode trajectories are treated as two separate IVPs, they are not accurately equal, causing error in gradient estimation (Gholami et al., 2019). The *naive method* directly back-propagates through ODE solvers; however, it has a redundantly deep computation graph when adaptive solvers search for optimal stepsize (Wanner & Hairer, 1996).

We propose the *adaptive checkpoint adjoint* (ACA) method to accurately estimate the gradient for NODEs. ACA supports adaptive ODE solvers. In automatic differentiation, ACA applies a *trajectory checkpoint* strategy, which stores the forward-mode trajectory with minimal memory; the forward-mode trajectory is used as the reverse-mode trajectory to guarantee numerical accuracy. ACA deletes redundant components during the backward-pass for a shallow computation graph and accurate gradient estimation. We provide the PyTorch implementation of ACA: `https://github.com/juntang-zhuang/torch-ACA`.

Our contributions can be summarized as:
(1) We theoretically analyze the numerical error with the adjoint and naive methods, and propose ACA to accurately estimate gradients of NODEs.
(2) On image classification tasks, compared with the adjoint and naive methods, ACA achieves twice the speed and half the error rate; furthermore, to our knowledge, ACA is the

---

[*]Equal contribution [1]Department of Biomedical Engineering, Yale University, New Haven, CT USA [2]Department of Radiology & Biomedical Imaging, Yale School of Medicine, New Haven, CT USA [3]Department of Statistics and DataScience, Yale University, New Haven, CT USA [4]Department of Electrical Engineering, Yale University, New Haven, CT USA. Correspondence to: Juntang Zhuang <j.zhuang@yale.edu>, james Duncan <james.duncan@yale.edu>.
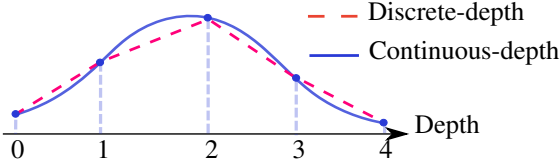
Figure 1. From discrete-depth model to continuous depth model.

first to enable NODE with adaptive solvers to outperform ResNet in both accuracy and test-retest reliability. On time series modeling, ACA outperforms other methods.
(3) We show that NODE can incorporate physical knowledge and improve accuracy when trained with ACA. Furthermore, ACA can be applied to general ODEs.

2. Preliminaries

2.1. Neural Ordinary Differential Equations

NODE views the model as an ordinary differential equation, whose derivative is parameterized by a neural network. NODE can be represented as:

$$\frac{dz(t)}{dt} = f(z(t), t, \theta), \; s.t. \; z(0) = x, \; t \in [0, T] \quad (1)$$

where $z(t)$ is the hidden state, T is the end time, and f is the network with parameters θ. $z(0)$ is the initial condition, which equals input x. Output of the model is $z(T)$.

We draw the connection between NODE and conventional networks in Fig. 1, where a discrete-depth model takes integer depths, and a continuous-depth model has values at all real-number depths. Compared to discrete-layer models, the feature map evolves smoothly with depth in NODE.

2.2. Analytical Form of Gradient for NODE

We formulate the training process of NODE as an optimization problem:

$$argmin_\theta \frac{1}{N} \sum_{i=1}^{N} J(\hat{y}_i, y_i) \quad (2)$$

$$s.t. \; \frac{dz_i(t)}{dt} = f(z_i(t), t, \theta), \; z_i(0) = x_i, \quad (3)$$

$$\hat{y}_i = z_i(T), \; t \in [0, T], \; i = 1, 2, ..N \quad (4)$$

where J is the loss function (e.g. cross-entropy, L_2 loss).

We use the Lagrangian Multiplier Method to solve the problem defined in Eq. 4. For simplicity, considering only one example (can be easily extended to the multiple examples case), the Lagrangian is

$$L = J(z(T), y) + \int_0^T \lambda(t)^\top [\frac{dz(t)}{dt} - f(z(t), t, \theta)]dt \quad (5)$$

Theorem 2.1 *The gradient derived from Karush-Kuhn-Tucker (KKT) conditions for Eq. 5 is:*

$$\frac{\partial J}{\partial z(T)} + \lambda(T) = 0 \quad (6)$$

$$\frac{d\lambda(t)}{dt} + \left(\frac{\partial f(z(t), t, \theta)}{\partial z(t)}\right)^\top \lambda(t) = 0 \; \forall t \in (0, T) \quad (7)$$

$$\frac{dL}{d\theta} = \int_T^0 \lambda(t)^\top \frac{\partial f(z(t), t, \theta)}{\partial \theta} dt \quad (8)$$

Detailed proofs are in Appendix C. $\lambda(t)$ also corresponds to the negative *adjoint variable* in optimal control (Pontryagin, 1962; Chen et al., 2018).

Summary The analytical solution can be summarized as:
(1) Solve $z(t)$ in time $0 \rightarrow T$.
(2) Determine $\lambda(T)$ with Eq. 6.
(3) Solve $\lambda(t)$ in time $T \rightarrow 0$, following Eq. 7 and boundary condition $\lambda(T)$.
(4) Calculate parameter gradient by Eq. 8.

Note that in order to calculate Eq. 8, $\lambda(t)$ and $z(t)$ are required for every t. Since $\lambda(t)$ and $z(t)$ are solved in opposite directions, we need to either memorize $z(t)$, or find a method to recover $z(t)$. Note that Eq. 8 is the analytical form, and needs to be numerically calculated in practice.

2.3. Numerical Integration

ODE solvers aim to numerically calculate

$$z(T) = z(0) + \int_0^T f(z(t), t, \theta)dt \quad (9)$$

We mainly consider adaptive stepsize solvers. Compared to constant-stepsize solvers, adaptive solvers can estimate error and adaptively control stepsize (Press et al., 1988).

Notations We summarize notations here, which are also demonstrated in Fig. 2 and Fig. 3:

- $z_i(t_i)/\bar{z}(\tau_i)$: hidden state in forward/reverse time trajectory at time t_i/τ_i.
- $\Phi_{t_i}^t(z_i)$: the *oracle* solution of the ODE at time t, starting from (t_i, z_i). Black dashed curve in Fig. 2 and Fig. 3. Φ is called the *flow map*.
- $\psi_{h_i}(t_i, z_i)$: the *numerical* solution at time $t_i + h_i$, starting from (t_i, z_i). Blue solid line in Fig. 2.
- $L_{h_i}(t_i, z_i)$: local truncation error between numerical approximation and oracle solution, where

$$L_{h_i}(t_i, z_i) = \psi_{h_i}(t_i, z_i) - \Phi_{t_i}^{t_i + h_i}(z_i) \quad (10)$$

- R_i: the local error $L_{h_i}(t_i, z_i)$ propagated to end time.

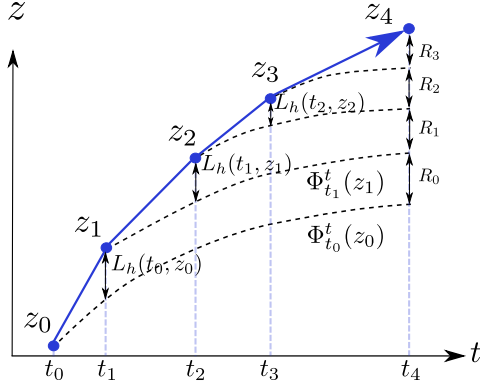$$R_i = \Phi_{t_{i+1}}^T(z_{i+1}) - \Phi_{t_i}^T(z_i) \quad (11)$$

Figure 2. Forward-time integration. The global error is the sum of local error $L_h(t_i, z_i)$ propagated to end time. This picture is called *Lady Windermere's Fan* (Wanner & Hairer, 1996). Details of notations summarized in Sec. 2.3

Figure 3. Reverse-time integration. Blue curve is the same trajectory as in forward-time integration. Both naive method and ours accurately recover the forward-time trajectory, while adjoint method forgets the forward-time trajectory.

- N_f: number of layers in f in Eq. 1.
- N_t / N_r: number of discretized points (outer iterations in Algo. 1) in forward / reverse integration. It varies with input and error tolerance for adaptive solvers.
- m: average number of inner iterations in Algo. 1 to find an acceptable 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

The numerical integration algorithm is summarized in Algo. 1 and Fig. 2. The ODE solver progressively advances in time, and adapts the stepsize according to the error estimate. Note that for a given start point (t_i, z_i), the solver might need to execute the inner *while* loop in Algo. 1 many times until the stepsize is small enough, such that the error estimate is below a certain threshold. This process will generate a very redundant deep computation graph, where only the final h is needed.

3. Methods

In this section we describe different methods to compute the gradient. A summary comparing the methods is given in Table 1 and Fig. 3. We refer readers to the summary part of Sec. 2.2 for the analytical solution; the following
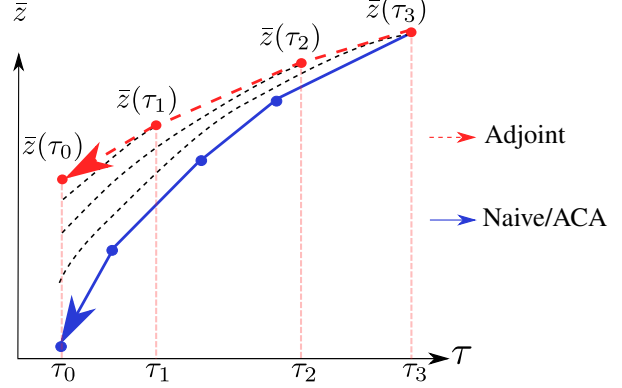
methods are different numerical implementations. Note that forward-time integration is the same for these methods.

3.1. Summary of Different Methods

Naive Method: direct back-prop through solver The simplest way is to treat the numerical ODE solver as a very deep discrete-layer network, and directly back-propagate. We call it the "naive" method. Because all computation graphs (including searching for optimal stepsize) are recorded in the memory for back-prop, the memory cost and depth are $O(N_f \times N_t \times m)$. The computation cost is doubled considering both forward and reverse integration. The memory cost of the naive method can quickly explode, because an accurate solution requires a very small stepsize, and hence a very large N_t.

Adjoint Method: forget forward-time trajectory To solve the memory issue with the naive method, Chen et al. (2018) proposed the *adjoint method*, originally illustrated by Pontryagin (1962). The adjoint method forgets the forward-time trajectory $z(t)$; instead, it remembers boundary condition $z(T)$ and $\lambda(T)$, then solves $z(t)$ and $\lambda(t)$ in reverse-time $T \to 0$. We use $\bar{z}$ to denote reverse-time solution. Because $\lambda(t)$ and $\overline{z(t)}$ are solved in the same direction, the integration in Eq. 8 only records current values, achieving $O(N_f)$ memory cost. Since the adjoint method needs to solve $\overline{z(t)}$ in reverse-time, it requires extra $O(N_f \times N_r \times m)$ computation, so the total computation cost is $O(N_f \times (N_t + N_r) \times m)$. Note that $\overline{z(t)}$ is not the same as $z(t)$ (as in Fig. 3) due to numerical errors, which will cause error in gradient estimation. We will explain in detail in Sec. 3.2.

Adaptive Checkpoint Adjoint (ACA): record $z(t)$ with minimal memory ACA tries to record $z(t)$ to avoid numerical errors, while also controlling memory cost. ACA

supports both adaptive and constant stepsize ODE solvers. It is summarized in Algo. 2, with a detailed version in Appendix A. Note that the forward-pass computation is the same as Algo. 1 for all three methods, so we omit common parts and focus on the unique part.

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 Eq. 7 and Eq. 8.
      (3) Delete local computation graphs.

During the forward-pass, to save memory, ACA deletes redundant computation graphs to search for the optimal stepsize. Instead, ACA applies the "trajectory checkpoint" strategy, recording the discretization points t_i (equivalently, the accepted stepsize $h_i = t_{i+1} - t_i$) and values z_i (not computation graph $\psi_{h_i}(t_i, z_i)$) at a memory cost $O(N_t)$. Considering $O(N_f)$ memory cost for one evaluation of ψ, the total memory cost is $O(N_f + N_t)$.

During the backward-pass, going reverse-time, ACA performs the forward-pass and backward-pass *locally* from t_i to t_{i+1}, and updates λ and $\frac{dL}{d\theta}$. Computations are evaluated at saved discretization points $\{t_0, ...t_{N_t}\}$, using saved values $\{z_0, ..., z_{N_t}\}$, to guarantee accuracy between forward-time and reverse-time trajectory. We only need to search for optimal stepsize during the forward-pass, with m inner iterations in Algo. 1; during the backward-pass we reuse saved stepsizes, so the total computation cost is $O(N_f \times N_t \times (m + 1))$.

3.2. Adjoint Method has Numerical Errors

Numerical Experiments Due to memory consideration, the adjoint method forgets forward-time trajectory $z(t)$, and instead solves reverse-time trajectory $\overline{z(\tau)}$ with initial condition $\overline{z(T)} = z(T)$. Thus, $z(t)$ and $\overline{z(\tau)}$ could mismatch due to numerical errors, as demonstrated in Fig. 3. We performed numerical experiments with *ode45* solver under default settings in MATLAB. We experimented with the *van der Pol equation* (Van der Pol, 1960), and a convolutional function with a random 3×3 kernel. Results are shown in Fig. 4 and 5. These examples validate our argument about the numerical error of the adjoint method.
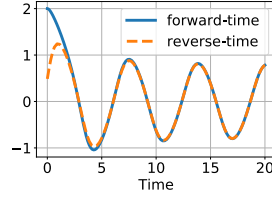

Figure 4. Forward time and reverse time trajectory for numerical solution to *van der Pol equation.*

Figure 5. Input (left) and reverse-time reconstruction (right) for an ODE defined by a convolution function.

Analysis of Numerical Errors We analyze the numerical errors of the adjoint method. Our results are extensions of Niesen & Hall (2004). We start from the following theorem:

Theorem 3.1 (Picard-Lindelöf Theorem) *(Lindelöf, 1894) Consider the initial value problem (IVP):*

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

Suppose in a region $R = [t_0 - a, t_0 + a] \times [z_0 - b, z_0 + b]$, f is bounded ($\|f\| \leq M$), uniformly continuous in z with Lipschitz constant L, and continuous in t; then there exists a unique solution for the IVP, valid on the region where $a < min\{\frac{b}{M}, \frac{1}{L}\}$.

The Picard-Lindelöf Theorem states a sufficient condition for existence and uniqueness for an IVP. Okamura (1942) stated a necessary and sufficient condition. Without going deeper, we emphasize that Theorem 3.1 has a validity region, outside this region the theorem may not hold.

It is trivial to check NODE satisfies the above conditions; see the proof in Appendix B. For simplicity, we assume Theorem 3.1 always holds on $t \in [0, T]$. (If $[0, T]$ is outside the region of validity, the adjoint method cannot recover the forward-time trajectory, while the naive method and ACA record the trajectory in memory with "checkpoint".)

Recall $\Phi_{t_i}^t(z_i)$ is the *flow map*, which is the *oracle* solution starting from (t_i, z_i). Define the *variational flow* as:

$$D\Phi_{t_0}^t = \frac{d\Phi_{t_0}^t(z_0)}{dz_0} \tag{12}$$

Consider an ODE solver of order p. The local truncation error $L_h(t_i, z_i)$ is of order $O(h^{p+1})$ and can be written as

$$L_h(t_i, z_i) = \psi_h(t_i, z_i) - \Phi_{t_i}^{t_i+h}(z_i) = h^{p+1}l(t_i, z_i) + O(h^{p+2}) \tag{13}$$

where l is some function of $O(1)$. Denote the global error

as $G(T)$ at time T, then it satisfies:

$$G(T) = z_{N_t} - \Phi_{t_0}^T(z_0) = \sum_{k=0}^{N_t-1} R_k \qquad (14)$$

Eq. 14 is explained by Fig. 2: global error is the sum of all local errors propagated to the end time. R_k is the propagated local error defined by Eq. 11. For simplicity of analysis, we consider constant stepsize solvers with sufficiently small stepsize h, and let $N_t = N_r = N$.

Theorem 3.2 *If the conditions of the Picard-Lindelöf theorem are satisfied, then for an ODE solver of order p, the global error at time T is:*

$$G(t_0 \to T) = \sum_{k=0}^{N-1} \left[h_k^{p+1} D\Phi_{t_k}^T(z_k) l(t_k, z_k) \right] + O(h^{p+1})$$
$$(15)$$

and the error of the reconstructed initial value by the adjoint method is:

$$G(t_0 \to T \to t_0) = G(t_0 \to T) + G(T \to t_0)$$
$$= \sum_{k=0}^{N-1} \left[h_k^{p+1} D\Phi_{t_k}^T(z_k) l(t_k, z_k) + \right.$$
$$\left. (-h_k)^{p+1} D\Phi_T^{t_k}(\overline{z_k}) \overline{l(t_k, z_k)} \right] + O(h^{p+1}) \quad (16)$$

where $G(t_0 \to T \to t_0)$ represents the global error of integration from t_0 to T, then from T to t_0. Terms for reverse-time trajectory are overlined $(\bar{l}, \bar{z})$ to differentiate from forward-time trajectory.

Proofs are in Appendix B. Eq. 16 can be divided into two parts. $G(t_0 \to T)$ corresponds to forward-time error, as shown in Fig. 2; $G(T \to t_0)$ corresponds to reverse-time error, as shown in Fig. 3. When h is small, assume:

$$\overline{z_k} = z_k + O(h^p) \qquad (17)$$
$$\overline{l(t_k, z_k)} = l(t_k, z_k) + O(h^p) \qquad (18)$$
$$D\Phi_T^{t_k}(\overline{z_k}) = D\Phi_T^{t_k}(z_k) + O(h^p) \qquad (19)$$

Note that when existence and uniqueness are satisfied, Φ defines a bijective mapping between $z(t_k)$ and $z(T)$, hence

$$D\Phi_T^{t_k} = (D\Phi_{t_k}^T)^{-1} \qquad (20)$$

Plugging Eq. 17-20 into Eq. 16,

$$G(t_0 \to T \to t_0) = \sum_{k=0}^{N-1} h^{p+1} l(t_k, z_k) e_k + O(h^{p+1})$$
$$(21)$$
$$e_k = D\Phi_{t_k}^T(z_k) + (-1)^{p+1} (D\Phi_{t_k}^T(z_k))^{-1} \qquad (22)$$

Reverse accuracy for all t_0 requires $e_k = 0$ for all k. If p is odd, the two terms in Eq. 22 are of the same sign; thus, e_k cannot be 0. If p is even, $e_k = 0$ requires $D\Phi_{t_k}^T(z_k) = I$, which requires NODE to be an identity function; in this case the model learns nothing. Hence, the adjoint method has numerical errors caused by truncation errors of numerical ODE solvers.
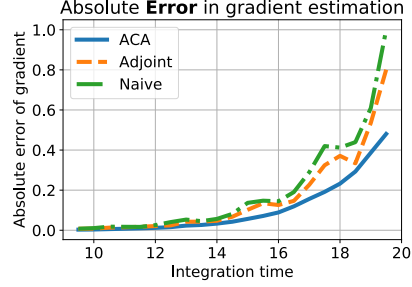

Figure 6. Absolute value of error in gradient estimation for different methods. Problem defined by Eq. 27 to Eq. 29.

3.3. Naive Method has Deep Computation Graph

Note that for each step advance in time, there are on average m steps to find an acceptable stepsize, as in Algo. 1. We give an example below:

$$out_1, h_1, error_1 = \psi(t, h_0, z) \qquad (23)$$
$$out_2, h_2, error_2 = \psi(t, h_1, z) \qquad (24)$$
$$... \qquad (25)$$
$$out_m, h_m, error_m = \psi(t, h_m, z) \qquad (26)$$

Suppose it takes m steps for find an acceptable stepsize such that $error_m < tolerance$. The naive method treats h_m as a recursive function of h_0, and back-propagates through all m steps in the computation graph; while ACA takes h_m as a constant, and back-propagates only through the final accepted step (Eq. 26); therefore, the depth of computation graph is $O(N_f \times N_t)$ for ACA, and $O(N_f \times N_t \times m)$ for the naive method. Note that the output of the forward pass is the same for both methods; the backward pass is different.

The very deep computation graph in naive method takes more memory. More importantly, it might cause vanishing or exploding gradient (Pascanu et al., 2013), since there's no special structure such as residual connection to deal with the deep structure: specifically, in Eq. 23 to Eq. 26, only h_i is passed to the next step, and typically in the form $h_{i+1} = h_i / error_i^p$.

3.4. ACA Guarantees Reverse-accuracy and has Shallow Computation Graph

Table 1 compares the gradient estimation methods. Adjoint method suffers from numerical error in reverse-mode trajectory; naive method suffers from vanishing or exploding gradient caused by deep computation graph ($O(N_f \times N_t \times m)$).

Compared with the adjoint method, ACA guarantees accuracy of reverse-mode trajectory by recording the forward-mode trajectory. Compared with the naive method, ACA deletes redundant components, hence has a shallower computation graph (ACA v.s. naive: $O(N_f \times N_t)$ v.s. $O(N_f \times N_t \times m)$) and smaller memory cost. Finally, ACA has the

	Naive	Adjoint	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	✓	✗	✓

Table 1. Comparison between different methods to derive parameter gradient. Note N_r is only meaningful for adjoint method. Our method achieves the lowest computation cost, accuracy in reverse-time trajectory, and a shallow computation graph, with medium memory cost.

lowest computation cost at a medium memory cost.

4. Experiments

4.1. Toy Example

Consider the following toy problem:

$$\frac{dz(t)}{dt} = kz(t), \ \ z(0) = z_0 \tag{27}$$

$$L(z(T)) = z(T)^2 = z_0^2 exp(2kT) \tag{28}$$

$$\frac{dL}{dz_0} = 2z_0 exp(2kT) \tag{29}$$

We plot the absolute value of error between the analytical solution in Eq. 29 and numerical results from various methods as a function of T in Fig. 6. All numerical methods use the *Dopri5* (Dormand & Prince, 1980) solver with error tolerance 10^{-5}. ACA consistently outperforms the naive method and adjoint method, which agrees with our analysis in Table 1.

4.2. Supervised Learning on Image Classification

Network Structure For a fair comparison with state-of-the-art discrete-layer models, we modify a ResNet18 into a NODE18 with the *same* number of parameters. A residual block is defined as:

$$y = x + f(x, \theta) \tag{30}$$

The corresponding ODE-Block is:

$$z(T) = z(0) + \int_0^1 f(z(t), \theta) dt \tag{31}$$

where a residual-block and ODE-Block have the same structure of f (e.g. a sequence of conv-bn-relu layers).

Comparison of gradient estimation methods for NODE We trained the same NODE structure to perform image classification on the CIFAR10 dataset using different gradient estimation methods. The relative and absolute error tolerance are set as 1e-5 for the adjoint and naive method, with *Dopri5* solver implemented by Chen et al. (2018). All methods are trained with SGD optimizer. For each method, we perform 3 runs and record the mean and variance of

test accuracy varying with training process. All models are trained for 90 epochs, with initial learning rate of 0.01, and decayed by a factor of 0.1 at epoch 30 and 60. The adjoint method and ACA use a batchsize of 128, while the naive method uses a batchsize of 32 due to its large memory cost.

Test accuracy varying with training epoch is plotted in Fig. 7(a). For the same number of training epochs, ACA ($\sim 5\%$ error rate) outperforms the adjoint and naive method ($\sim 10\%$ error rate) by a large margin.

Test accuracy varying with training time is shown in Fig. 7(b). To train for 90 epochs on a single GTX 1080Ti GPU, ACA takes about 9 hours, while the adjoint method takes about 18 hours, and the naive method takes more than 30 hours. The running time validates our analysis on computation cost in Table 1.

Overall, for the same NODE model, ACA significantly outperforms the adjoint and naive method, with twice the speed and half the error rate.

Accuracy comparison between NODE and ResNet We also compare the performance between ResNet and NODE. Note that both models have the same number of parameters.

We trained both models for 10 runs with random initialization using the SGD optimizer. All models are trained for 90 epochs. Results are summarized in Fig. 7(c) and (d). On both CIFAR10 and CIFAR100 datasets, NODE significantly outperforms ResNet when trained with ACA.

We then re-initialized and re-trained for 350 epochs for a fair comparison with ResNet reported by Liu (2017), and summarize the results in Table 2. On image classification tasks, compared to the adjoint method, ACA reduces the error rate of NODE18 from 10% (30%) to 5% (23%) on CIFAR10 (CIFAR100). Furthermore, NODE18 has the same number of parameters as ResNet18, but outperforms deeper networks such as ResNet101 on both datasets.

Robustness to ODE solvers We implemented adaptive ODE solvers of different orders, as shown in Table 2. HeunEuler, RK23, RK45 are of order 1, 2, 4 respectively, i.e., for each step of ψ, f is evaluated 1, 2, 4 times respectively. During inference, using different solvers is equivalent to changing model depth (**without** re-training the network). For discrete-layer models, it would generally cause huge er-
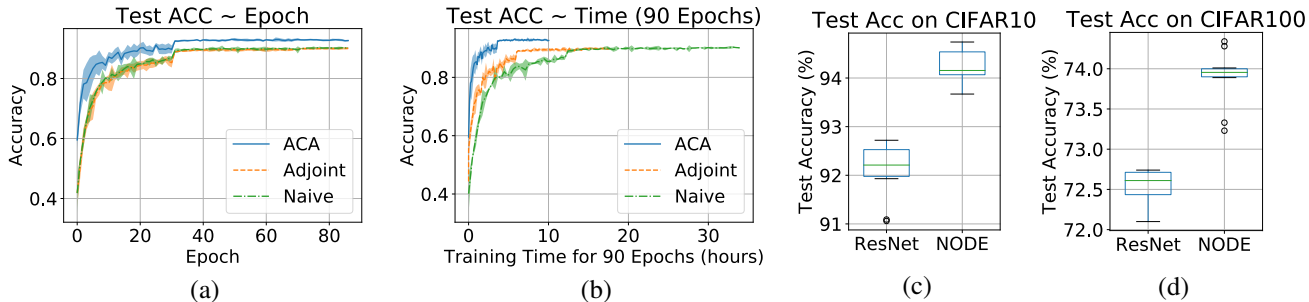
Figure 7. From left to right: (a) Test accuracy vs epoch curve on CIFAR10, for NODE18 trained with different methods. (b) Test accuracy vs running time curve on CIFAR10, for NODE18 trained for 90 epochs. (c) Distribution of test accuracy of 10 runs on CIFAR10. NODE18 is trained with ACA. (d) Distribution of test accuracy of 10 runs on CIFAR100. NODE18 is trained with ACA.

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

Table 2. **Error rate** on test set. NODE18-ACA is trained with HeunEuler solver, and tested with different solvers (including fixed-stepsize and adaptive stepsize solvers of various orders) *without* re-training. Other models are trained and tested with the same method. "*" represents results from the literature (Gholami et al., 2019; He et al., 2016; Liu, 2017), note that our reproduced baseline (adjoint) is better than the literature.

Dataset	Model	Whole Test Set		Misclassified Test Data	
		ICC1	ICC1k	ICC1	ICC1k
CIFAR10	ResNet18	0.932-0.935	0.992-0.993	0.581-0.608	0.933-0.939
	NODE18	**0.943-0.945**	**0.993-0.994**	**0.650-0.675**	**0.949-0.954**
CIFAR100	ResNet18	0.759-0.768	0.969-0.971	0.553-0.571	0.925-0.930
	NODE18	**0.767-0.776**	**0.971-0.972**	**0.570-0.587**	**0.930-0.934**

Table 3. ICC (95% confidence region, $[\mu - 2\sigma, \mu + 2\sigma]$) for ResNet and NODE-ACA among 10 runs with random initialization, tested on CIFAR10 (top) and CIFAR100 (bottom). **Higher** is better.

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).

rors (see results in Appendix D); for continuous models, we observe only ∼1% increase in error rate. Thus, our method is robust to different solvers.

Test-retest reliability Test-retest reliability measures the agreement between multiple raters, and is crucial for clinical practices (Bland & Altman, 1986; Williams et al., 1992). For machine learning, test-retest reliability quantifies the stability of a model under random initialization and re-training. Intraclass correlation coefficient (ICC) (Weir, 2005) is widely used to quantify test-retest reliability. ICC is between 0 and 1, with higher values for better agreement.

We take the results of 10 runs with independent initialization, as in Fig. 7(c) and (d), and measure ICC with the $psych$ package (Revelle, 2017). We report two types of coefficient in Table 3: $ICC1$ (randomly selected judges) and $ICC1k$ (average of raters).

As Table 3 shows, NODE consistently generates higher ICC than ResNet on both datasets. To remove the effect caused by different accuracy, we also measure ICC only on misclassified data points, and NODE produces a significantly higher ICC.

Summary NODE trained with ACA generates superior performance on benchmark classification tasks. Compared with the adjoint and naive method, ACA is faster and more accurate. Compared with ResNet, to our knowledge, ACA is the first to enable NODE with adaptive solvers to achieve higher accuracy and better test-retest reliability. The better performance comes from two reasons: (1) accurate gradient estimation with ACA, (2) feature maps evolve smoothly with depth (Fig. 1); this property of NODE might be helpful for better generalization (Jin et al., 2019) and optimization (Nesterov, 2005).

4.3. Time-series Model for Irregularly-sampled Data

Standard recurrent neural networks (RNN) have difficulties modelling time series data with non-uniform intervals. The recently proposed latent-ODE model (Rubanova et al., 2019) is a generalization of NODE to time-series models, and can handle arbitrary time gaps.
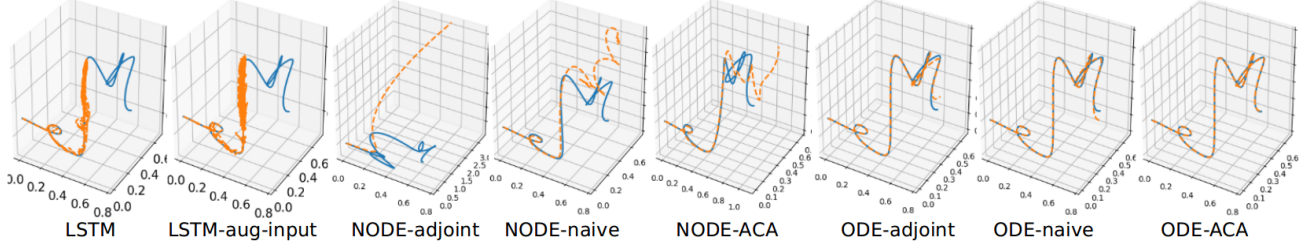
Figure 8. Fitted trajectory (orange dashed curve) and ground truth (blue solid curve) for one planet in 3D space. Two trajectories almost overlap in the rightmost figure. Display is adaptively determined in Python for each figure, but ground truth is the same. Time range is [0,1] year for training, and [0,2] years for visualization. Figures are in the same order as Table. 5.

	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.

We validate our method on the *Mujoco* dataset (Tassa et al., 2018) under the same setting as in Rubanova et al. (2019), with the only difference being the gradient estimation method. We report the mean squared error (MSE) of interpolation. As shown in Table 4, ACA consistently outperforms other methods.

4.4. Incorporate Physical Knowledge into Models

Differential equations are important tools for modern physics (Sommerfeld, 1949), chemistry (Strogatz, 2018), quantitative biology (Jones et al., 2009), system control (Lee & Markus, 1967) and engineering (Lyapunov, 1992). In practice, for many problems a large training set is *unavailable*, but some physical knowledge is known. It is straightforward to incorporate such knowledge into NODE: set f in the form of physical knowledge.

Problem definition We give an example with the three-body problem (Barrow-Green, 1997). 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). The underlying dynamics governing their motion is:

$$\ddot{\mathbf{r}_i} = -\sum_{j \neq i} Gm_j \frac{\mathbf{r_i} - \mathbf{r_j}}{|\mathbf{r_i} - \mathbf{r_j}|^3} \qquad (32)$$

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

We consider the following problem: given observations of trajectory $\mathbf{r_i(t)}$, $t \in [0, T]$, predict future trajectories $\mathbf{r_i}(t)$, $t \in [T, 2T]$, when mass m_i is *unknown*.

Models We consider different models:
(1) LSTM with trajectory $\mathbf{r_i}(t)$ as input;

(2) LSTM-aug-input, with augmented input defined as:

$$Aug = \{\mathbf{r_i}, \mathbf{r_i} - \mathbf{r_j}, \frac{\mathbf{r_i} - \mathbf{r_j}}{|\mathbf{r_i} - \mathbf{r_j}|^1}, \frac{\mathbf{r_i} - \mathbf{r_j}}{|\mathbf{r_i} - \mathbf{r_j}|^2}, \frac{\mathbf{r_i} - \mathbf{r_j}}{|\mathbf{r_i} - \mathbf{r_j}|^3}\}, j \neq i \qquad (33)$$

(3) NODE, where f is parameterized as a fully-connected layer using augmented input:

$$\ddot{\mathbf{r}} = FC(Aug) \qquad (34)$$

(4) ODE, with f in the form of Eq. 32. In this case, only 3 parameters, *the mass of planets*, are unknown.

With augmented input, the model knows partial information: the trajectory is related to the distance between planets. The ODE model has full knowledge of the system.

Results We simulate a 3-body system with *unequal masses* and *arbitrary initial conditions*, use time range [0,1] year for training, and measure the mean MSE of trajectory on [0,2] years. Results are reported in Fig. 8 and Table 5, with details in Appendix D. We provide videos [1] for better visualization. With no knowledge, the LSTM model fails due to the chaotic nature (Barrow-Green, 1997) of the three-body system and limited training data. With partial knowledge, NODE-ACA outperforms LSTM. With full knowledge, ODE performs the best. ACA outperforms adjoint and naive method, and supports general ODEs.

5. Scopes and limitations

Considering computation burden, this project only investigates explicit general-purpose one-step ODE solvers. There exists rich literature on other solvers including multi-step and implicit solvers (Wanner & Hairer, 1996; Rosenbrock, 1963; Hindmarsh, 1980; Brown et al., 1989). Acceleration

---

[1]https://www.youtube.com/playlist?list=PL7KkG3n9bER4ODAMzAKzfXIaF0ndUxK-N

methods, such as spectral element methods (Patera, 1984) and parallel-methods (Farhat & Chandesris, 2003), can be used to further improve ACA.

6. Related works

Training of NODE Quaglino et al. (2019) proposed to use spectral element method to train NODE. However, it requires ground truth for the entire trajectory, and therefore is not suitable for tasks like image classification. Gholami et al. (2019) proposed ANODE to deal with reverse-inaccuracy of the adjoint method, by discretizing the integration range into a *fixed* number of steps. Equivalently, ANODE can be viewed as a fixed-depth discrete-layer network with shared weights. Therefore, ANODE is equivalently using a constant stepsize solver, and loses the ability of error control with adaptive solvers, while ACA supports adaptive solvers. Dupont et al. (2019) proposed to augment NODE to a higher dimension for better performance. However, they are not dealing with gradient estimation, and the empirical performance is still inferior to discrete-layer networks.

Dynamics and Physics Many works incorporate physical priors or learn hidden dynamics from data (Ramsay et al., 2007; de Avila Belbute-Peres et al., 2018; Jia et al., 2018; Sienko et al., 2002; Chen & Pock, 2016; Weinan, 2017; Lu, 2017; Sonoda & Murata, 2017). Breen et al. (2019) used deep learning models to solve the three-body problem; however, their methods are limited to *equal masses with zero initial velocities*, while ours do not have these restrictions. Other works try to connect neural networks with dynamical systems (Ruthotto & Haber, 2018; Chang et al., 2018).

Gradient Checkpointing The gradient checkpointing (GC) strategy is used to train large networks with limited memory budget (Chen et al., 2016; Gruslys et al., 2016). However, ACA is not a GC version of the naive method: mathematically, naive-GC has the same computation graph depth of $O(N_f \times N_t \times m)$ as naive method, while ACA uses a simplified computation graph of depth $O(N_f \times N_t)$; hence both naive-GC and naive method suffer from exploding or vanishing gradient problem, while ACA is more numerically stable; if provided with unlimited memory, naive-GC achieves the same accuracy as naive method, while ACA achieves a higher accuracy.

7. Conclusion

We analyzed the inaccuracy of the adjoint and naive method for NODE, and proposed ACA for accurate gradient estimation. We demonstrated NODE trained with ACA is accurate, fast and robust to initialization. Furthermore, NODE can incorporate physical knowledge for better accuracy. We implemented ACA as a package. We hope the good practical performances of ACA, theoretical properties of NODE, and

our easy-to-use package, can inspire new ideas.

References

Ayed, I., de Bézenac, E., Pajot, A., Brajard, J., and Gallinari, P. Learning dynamical systems from partial observations. *arXiv preprint arXiv:1902.11136*, 2019.

Barrow-Green, J. *Poincaré and the three body problem*. Number 11. American Mathematical Soc., 1997.

Bland, J. M. and Altman, D. Statistical methods for assessing agreement between two methods of clinical measurement. *The lancet*, 327(8476):307–310, 1986.

Breen, P. G., Foley, C. N., Boekholt, T., and Zwart, S. P. Newton vs the machine: solving the chaotic three-body problem using deep neural networks. *arXiv preprint arXiv:1910.07291*, 2019.

Brown, P. N., Byrne, G. D., and Hindmarsh, A. C. Vode: A variable-coefficient ode solver. *SIAM*, 1989.

Chang, B., Meng, L., Haber, E., Ruthotto, L., Begert, D., and Holtham, E. Reversible architectures for arbitrarily deep residual neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Chen, T., Xu, B., Zhang, C., and Guestrin, C. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

Chen, T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. In *Advances in neural information processing systems*, pp. 6571–6583, 2018.

Chen, Y. and Pock, T. Trainable nonlinear reaction diffusion: A flexible framework for fast and effective image restoration. *IEEE transactions on pattern analysis and machine intelligence*, 39(6):1256–1272, 2016.

de Avila Belbute-Peres, F., Smith, K., Allen, K., Tenenbaum, J., and Kolter, J. Z. End-to-end differentiable physics for learning and control. In *Advances in Neural Information Processing Systems*, pp. 7178–7189, 2018.

Dormand, J. R. and Prince, P. J. A family of embedded runge-kutta formulae. *Journal of computational and applied mathematics*, 6(1):19–26, 1980.

Dupont, E., Doucet, A., and Teh, Y. W. Augmented neural odes. *arXiv preprint arXiv:1904.01681*, 2019.

Farhat, C. and Chandesris, M. Time-decomposed parallel time-integrators: theory and feasibility studies for fluid, structure, and fluid–structure applications. *International Journal for Numerical Methods in Engineering*, 58(9): 1397–1434, 2003.

Gholami, A., Keutzer, K., and Biros, G. Anode: Unconditionally accurate memory-efficient gradients for neural odes. *arXiv preprint arXiv:1902.10298*, 2019.

Grathwohl, W., Chen, R. T., Betterncourt, J., Sutskever, I., and Duvenaud, D. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

Gruslys, A., Munos, R., Danihelka, I., Lanctot, M., and Graves, A. Memory-efficient backpropagation through time. In *Advances in Neural Information Processing Systems*, pp. 4125–4133, 2016.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hindmarsh, A. C. Lsode and lsodi, two new initial value ordinary differential equation solvers. *ACM Signum Newsletter*, 15(4):10–11, 1980.

Jia, X., Karpatne, A., Willard, J., Steinbach, M., Read, J., Hanson, P. C., Dugan, H. A., and Kumar, V. Physics guided recurrent neural networks for modeling dynamical systems: Application to monitoring water temperature and quality in lakes. *arXiv preprint arXiv:1810.02880*, 2018.

Jin, P., Lu, L., Tang, Y., and Karniadakis, G. E. Quantifying the generalization error in deep learning in terms of data distribution and neural network smoothness. *arXiv preprint arXiv:1905.11427*, 2019.

Jones, D. S., Plank, M., and Sleeman, B. D. *Differential equations and mathematical biology*. Chapman and Hall/CRC, 2009.

Lee, E. B. and Markus, L. Foundations of optimal control theory. Technical report, Minnesota Univ Minneapolis Center For Control Sciences, 1967.

Lindelöf, E. Sur l'application de la méthode des approximations successives aux équations différentielles ordinaires du premier ordre. *Comptes rendus hebdomadaires des séances de l'Académie des sciences*, 116(3):454–457, 1894.

Liu, K. Train cifar10 with pytorch. 2017. URL https://github.com/kuangliu/pytorch-cifar.

Long, J., Shelhamer, E., and Darrell, T. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015.

Lu, Y. e. a. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. *arXiv*, 2017.

Lyapunov, A. M. The general problem of the stability of motion. *International journal of control*, 55(3):531–534, 1992.

Nesterov, Y. Smooth minimization of non-smooth functions. *Mathematical programming*, 103(1):127–152, 2005.

Newton, I. *Philosophiae naturalis principia mathematica*, volume 1. G. Brookman, 1833.

Niesen, J. and Hall, T. *On the global error of discretization methods for ordinary differential equations*. PhD thesis, Citeseer, 2004.

Okamura, H. Condition nécessaire et suffisante remplie par les équations différentielles ordinaires sans points de peano. *Mem. Coll. Sci., Kyoto Imperial Univ., Series A*, 24:21–28, 1942.

Pascanu, R., Mikolov, T., and Bengio, Y. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pp. 1310–1318, 2013.

Patera, A. T. A spectral element method for fluid dynamics: laminar flow in a channel expansion. *Journal of computational Physics*, 54(3):468–488, 1984.

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

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. Numerical recipes in c, 1988.

Quaglino, A., Gallieri, M., Masci, J., and Koutník, J. Accelerating neural odes with spectral elements. *arXiv preprint arXiv:1906.07038*, 2019.

Ramsay, J. O., Hooker, G., Campbell, D., and Cao, J. Parameter estimation for differential equations: a generalized smoothing approach. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69(5):741–796, 2007.

Revelle, W. R. psych: Procedures for personality and psychological research. 2017.

Rosenbrock, H. Some general implicit processes for the numerical solution of differential equations. *The Computer Journal*, 5(4):329–330, 1963.

Rubanova, Y., Chen, R. T., and Duvenaud, D. Latent odes for irregularly-sampled time series. *arXiv preprint arXiv:1907.03907*, 2019.

Ruthotto, L. and Haber, E. Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision*, pp. 1–13, 2018.

Sienko, W., Citko, W. M., and Wilamowski, B. M. Hamiltonian neural nets as a universal signal processor. In *IEEE 2002 28th Annual Conference of the Industrial Electronics Society. IECON 02*, volume 4, pp. 3201–3204. IEEE, 2002.

Sommerfeld, A. *Partial differential equations in physics*. Academic press, 1949.

Sonoda, S. and Murata, N. Double continuum limit of deep neural networks. In *ICML Workshop Principled Approaches to Deep Learning*, 2017.

Strogatz, S. H. *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering*. CRC press, 2018.

Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. d. L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Van der Pol, B. A theory of the amplitude of free and forced triode vibrations, radio rev. 1 (1920) 701-710, 754-762; selected scientific papers, vol. i, 1960.

Wanner, G. and Hairer, E. *Solving ordinary differential equations I*. Springer Berlin Heidelberg, 1996.

Weinan, E. A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 5(1):1–11, 2017.

Weir, J. P. Quantifying test-retest reliability using the intraclass correlation coefficient and the sem. *The Journal of Strength & Conditioning Research*, 19(1):231–240, 2005.

Williams, J. B., Gibbon, M., First, M. B., Spitzer, R. L., Davies, M., Borus, J., Howes, M. J., Kane, J., Pope, H. G., Rounsaville, B., et al. The structured clinical interview for dsm-iii-r (scid): Ii. multisite test-retest reliability. *Archives of general psychiatry*, 49(8):630–636, 1992.