Opening the Blackbox: Accelerating Neural Differential Equations by Regularizing Internal Solver Heuristics

Avik Pal [1 2] **Yingbo Ma** [2] **Viral Shah** [2] **Christopher Rackauckas** [2 3 4 5]

Abstract

Democratization of machine learning requires architectures that automatically adapt to new problems. Neural Differential Equations (NDEs) have emerged as a popular modeling framework by removing the need for ML practitioners to choose the number of layers in a recurrent model. While we can control the computational cost by choosing the number of layers in standard architectures, in NDEs the number of neural network evaluations for a forward pass can depend on the number of steps of the adaptive ODE solver. But, can we force the NDE to learn the version with the least steps while not increasing the training cost? Current strategies to overcome slow prediction require high order automatic differentiation, leading to significantly higher training time. We describe a novel regularization method that uses the internal cost heuristics of adaptive differential equation solvers combined with discrete adjoint sensitivities to guide the training process towards learning NDEs that are easier to solve. This approach opens up the blackbox numerical analysis behind the differential equation solver's algorithm and directly uses its local error estimates and stiffness heuristics as cheap and accurate cost estimates. We incorporate our method without any change in the underlying NDE framework and show that our method extends beyond Ordinary Differential Equations to accommodate Neural Stochastic Differential Equations. We demonstrate how our approach can halve the prediction time and, unlike other methods which can increase the training time by an order of magnitude, we demonstrate similar reduction in training times. Together this showcases how the knowledge embedded within
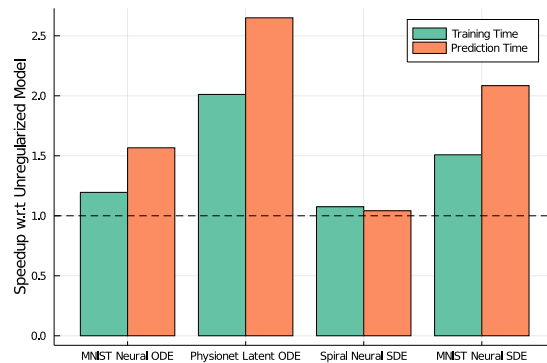
[1]Indian Institute of Technology Kanpur [2]Julia Computing [3]Massachusetts Institute of Technology [4]Pumas AI [5]University of Maryland Baltimore. Correspondence to: Avik Pal <avikpal@cse.iitk.ac.in>, Christopher Rackauckas <crackauc@mit.edu>.

Figure 1. **Training and Prediction Performance of Regularized NDEs** We obtain an average training and prediction speedup of 1.45x and 1.84x respectively for our best model on supervised classification and time series problems.

state-of-the-art equation solvers can be used to enhance machine learning.

1. Introduction

How many hidden layers should you choose in your recurrent neural network? Chen et al. (2018) showed that the answer could be found automatically by using a continuous reformulation, the neural ordinary differential equation, and allowing an adaptive ODE solver to effectively choose the number of steps to take. Since then the idea was generalized to other domains such as stochastic differential equations (Liu et al., 2019; Rackauckas et al., 2020) but one fact remained: solving a neural differential equation is expensive, and training a neural differential equation is even more so. In this manuscript we show a generally applicable method to force the neural differential equation training process to choose the least expensive option. We open the blackbox and show how using the numerical heuristics baked inside of these sophisticated differential equation solver codes allows for identifying the cheapest equations without requiring extra computation.

Our main contributions include:

- We introduce a novel regularization scheme for neural differential equations based on the local error estimates and stiffness estimates. We observe that by
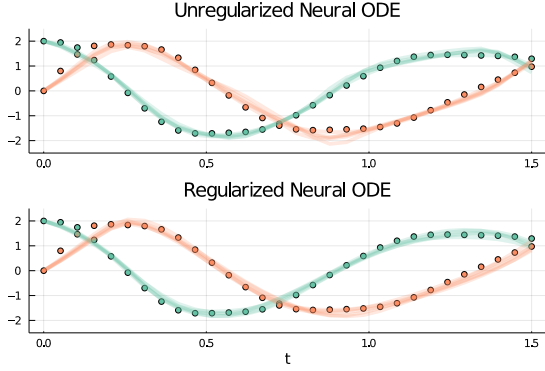
Figure 2. **Error and Stiffness Regularization Keeps Accuracy.** We show the fits of the unregularized/regularized Neural ODE variants on the Sprial equation. However, the unregularized variant requires 1083.0 ± 57.55 NFEs while the one regularized using the stiffness and error estimates requires only 676.2 ± 68.20 NFEs, reducing prediction time by nearly 50%.

white-boxing differential equation solvers to leverage pre-computed statistics about the neural differential equations, we can obtain faster training and prediction time while having a minimal effect on testing metrics.

- We compare our method with various regularization schemes (Kelly et al., 2020; Behl et al., 2020), which often use higher order derivatives and are difficult to incorporate within existing systems. We empirically show that regularization using cheap statistics can lead to as efficient predictions as the ones requiring higher order automatic differentiation (Kelly et al., 2020; Finlay et al., 2020) without the increased training time.

- We release our code[1], implemented using the Julia Programming Language (Bezanson et al., 2017) and SciML Software Suite (Rackauckas et al., 2019), with the intention of wider adoption of the proposed methods in the community.

2. Background

2.1. Neural Ordinary Differential Equations

Ordinary Differential Equations (ODEs) are used to model the instantaneous rate of change ($\frac{dz(t)}{dt}$) of a state $z(t)$. Initial Value Problems (IVPs) are a class of ODEs that involve finding the state at a later time t_1, given the value z_0 at time t_0. This state, $z(t_1) = z_0 + \int_{t_0}^{t_1} f_\theta(z(t), t)dt$, generally cannot be computed analytically and requires numerical solvers. Lu et al. (2018) observed the similarity between fixed time-step discretization of ODEs and Residual Neural Networks (He et al., 2015). Chen et al. (2018) proposed the Neural ODE framework which use neural networks to

---

[1] https://github.com/avik-pal/RegNeuralODE.jl

model the ODE dynamics $\frac{dz(t)}{dt} = f_\theta(z(t), t)$. Using adaptive time stepping allows the model to operate at a variable continuous depth depending on the inputs. Removal of the fixed depth constraint of Residual Networks provides a more expressive framework and offer several advantages in problems like density estimation (Grathwohl et al., 2018), irregularly spaced time series problems (Rubanova et al., 2019), etc.

2.2. Neural Stochastic Differential Equations

Stochastic Differential Equations (SDEs) couple the effect of noise to a deterministic system of equations. SDEs are popularly used to model fluctuating stock prices, thermal fluctuations in physical systems, etc. In this paper, we only discuss SDEs with Diagonal Multiplicative Noise, though our method trivially extends to all other forms of SDEs. Liu et al. (2019) propose an extension to Neural ODEs by stochastic noise injection in the form of Neural SDEs. Neural SDEs jointly train two neural networks f_θ and g_ϕ, such that, the dynamics $dz(t) = f_\theta(z(t), t)dt + g_\phi(z(t), t)dW$. Stochastic Noise Injection regularize the training of continuous neural models and achieves significantly better robustness and generalization performance.

2.3. Regularizing Neural ODEs for Speed

Given the map $z(0) \rightarrow z(1)$ does not uniquely define the dynamics, it is possible to regularize the training process to learn differential equations that can be solved using fewer evaluations of f_θ. In the case of continuous normalizing flows (CNF), the ordinary differential equation:

$$\frac{dz(t)}{dt} = f_\theta(z(t), t) \tag{1}$$

$$\frac{dy(t)}{dt} = -\text{tr}\left(\frac{df_\theta}{dz}\right) \tag{2}$$

where $y(t)$ evolves a log-density (Chen et al., 2018). The FFJORD method improves the speed of CNF evaluations by approximating $tr(\frac{df_\theta}{dz})$ via the Hutcheson trace estimator, i.e. $tr(\frac{df_\theta}{dz}) = \mathbb{E}[\epsilon^T \frac{df_\theta}{dz}\epsilon]$ where $\epsilon \sim \mathcal{N}(0, 1)$ (Hutchinson, 1989; Grathwohl et al., 2018). Subsequent research showed that this trace estimator could be used to regularize the Frobenius norm of the Jacobian $\|\frac{df_\theta}{dz}\| = \epsilon^T \frac{df_\theta}{dz}$ (Finlay et al., 2020). While $\epsilon^T \frac{df_\theta}{dz}$ is computationally expensive as it requires a reverse mode automatic differentiation evaluation in the model (leading to higher order differentiation), in the specific case of FFJORD this term is already required and thus this estimate is a computationally-free regularizer.

It was later shown that this form of regularization can be extended beyond FFJORD by using higher order automatic differentiation (Kelly et al., 2020). This was done by regularizing a heuristic for the local error estimate, namely $\mathcal{R}_K(\theta) = \int_{t_0}^{t_f} \|\frac{d^K z(t)}{dt^K}\|_2^2 dt$. The authors showed Taylor-

mode automatic differentiation improves the efficiency of calculating this estimator to a $\mathcal{O}(k^2)$ cost where k is the order of derivative that is required, though this still implies that obtaining the 5 derivatives requires is a significant computational increase. In fact, the authors noted that "when we train with adaptive solvers we do not improve overall training time", and in fact giving a 1.7x slower training time. In this manuscript we show that this is all the way up to 10x on the PhysioNet challenge problem.

Here we show how to arrive at a similar regularization heuristic that is applicable to all neural ODE applications with suitable adaptive ODE solvers and requires no higher order automatic differentiation. We will show that this form of regularization is able to significantly improve training times and generalizes to other architectures like neural SDEs.

2.4. Adaptive Time Stepping using Local Error Estimates

Runge-Kutta Methods (Runge, 1895; Kutta, 1901) are widely used for numerically approximating the solutions of ordinary differential equations. They are given by a tableau of coefficients $\{A, c, b\}$ where the stages s are combined to produce an estimate for the update at $t + h$:

$$k_s = f\left(t + c_s h, z(t) + \sum_{i=1}^{s} a_{si} k_i\right)$$
$$z(t + h) = z(t) + h \sum_{i=1}^{s} b_i k_i \tag{3}$$

For adaptivity, many Runge-Kutta methods include an alternative linear combiner $\tilde{b}_i$ such that $\tilde{z}(t + h) = z(t) + h \sum_{i=1}^{s} \tilde{b}_i k_i$ gives rise to an alternative solution, typically with one order less convergence (Wanner & Hairer, 1996; Fehlberg, 1968; Dormand & Prince, 1980; Tsitouras, 2011). A classic result from Richardson extrapolation shows that $E = \|\tilde{z}(t + h) - z(t + h)\|$ is an estimate of the local truncation error (Ascher & Petzold, 1998; Hairer et al., 1993). The goal of adaptive step size methods is to choose a maximal step size h for which this error estimate is below user requested error tolerances. Given the absolute tolerance $atol$ and relative tolerance $rtol$, the solver satisfies the following constraint for determining the time stepping:

$$E \leq \text{atol} + \max(|z(t)|, |z(t + h)|) \cdot \text{rtol} \tag{4}$$

The proportion of the error against the tolerance is thus:

$$q = \left\| \frac{E}{\text{atol} + \max(|z_n|, |z_{n+1}|) \cdot \text{rtol}} \right\| \tag{5}$$

If $q < 1$ then the proposed time step h is accepted, else it is rejected and reduced. In either case, a proportional error

control scheme (P-control) proposes $h_{\text{new}} = \eta q h$, while a standard PI-controller of explicit adaptive Runge-Kutta methods can be shown to be equivalent to using:

$$h_{\text{new}} = \eta q_{n-1}^{\alpha} q_n^{\beta} h \tag{6}$$

where η is the safety factor, q_{n-1} denotes the error proportion of the previous step, and (α, β) are the tunable PI gain hyperparameters (Wanner & Hairer, 1996). Similar embedded methods error estimation schemes have also been derived for stochastic Runge-Kutta integrators of SDEs (Rackauckas & Nie, 2017; 2020).

2.5. Stiffness Estimation

While there is no precise definition of stiffness, the definition used in practice is "stiff equations are problems for which explicit methods don't work" (Wanner & Hairer, 1996; Shampine & Gear, 1979). A simplified stiffness index is given by:

$$S = \max \|Re(\lambda_i)\| \tag{7}$$

where λ_i are the eigenvalues of the local Jacobian matrix. We note that various measures of stiffness have been introduced over the years, all being variations of conditioning of the pseudospectra (Shampine & Thompson, 2007; Higham & Trefethen, 1993). The difficulty in defining a stiffness metric is that in each case, some stiff systems like the classic Robertson chemical kinetics or excited Van der Pol equation may violate the definition, meaning all such definitions are (useful) heuristics. In particular, it was shown that for explicit Runge-Kutta methods satisfying $c_x = c_y$ for some internal step, the term

$$\|\lambda\| \approx \left\| \frac{f(t + c_x h, \sum_{i=1}^{s} a_{xi}) - f(t + c_y h, \sum_{i=1}^{s} a_{yi})}{\sum_{i=1}^{s} a_{xi} - \sum_{i=1}^{s} a_{yi}} \right\| \tag{8}$$

serves as an estimate to S (Shampine, 1977). Since each of these terms are already required in the Runge-Kutta updates of Equation 3, this gives a computationally-free estimate. This estimate is thus found throughout widely used explicit Runge-Kutta implementations, such as by the dopri method (found in suites like SciPy and Octave) to automatically exit when stiffness is detected (Wanner & Hairer, 1996), and by switching methods which automatically change explicit Runge-Kutta methods to methods more suitable for stiff equations (Rackauckas & Nie, 2019).

3. Method

3.1. Regularizing Local Error and Stiffness Estimates

Section 2.4 describes how larger local error estimates E lead to reduced step sizes and thus a higher overall cost in the neural ODE training and predictions. Given this, we propose regularizing the neural ODE training process by the

total local error in order to learn neural ODEs with as large step sizes as possible. Thus we define the regularizing term:

$$R_E = \sum_j E_j |h_j| \qquad (9)$$

summing over j the time steps of the solution. This was done by accumulating the E_j from the internals of the time stepping process at the end of each step. We note that this is similar to the regularization proposed in (Kelly et al., 2020), namely:

$$R_K = \int_{t_0}^{t_1} \left\| \frac{d^K z(t)}{dt^K} \right\| dt \qquad (10)$$

where integrating over the Kth derivatives is proportional to the principle (largest) truncation error term of the Runge-Kutta method (Hairer et al., 1993). However, this formulation requires high order automatic differentiation (which then is layered with reverse-mode automatic differentiation) which can be an expensive computation (Zhang et al., 2008) while Equation 9 requires no differentiation.

Similarly, the stiffness estimates at each step can be summed as:

$$R_S = \sum_j S_j \qquad (11)$$

giving a computational heuristic for the total stiffness of the equation. Notably, both of these estimates E_j and S_j are already computed during the course of a standard explicit Runge-Kutta solution, making the forward pass calculation of the regularization term computationally free.

3.2. Adjoints of Internal Solver Estimates

Notice that $E_j = \sum_{i=1}^{s} (b_i - \tilde{b}_i) k_i$ cannot be constructed directly from the $z(t_j)$ trajectory of the ODE's solution. More precisely, the k_i terms are not defined by the continuous ODE but instead by the chosen steps of the solver method. Continuous adjoint methods for neural ODEs (Chen et al., 2018; Zhuang et al., 2021) only define derivatives in terms of the ODE quantities. This is required in order exploit properties such as allowing different steps in reverse and reversibility for reduced memory, and in constructing solvers requiring fewer NFEs (Kidger et al., 2020). Indeed, computing the adjoint of each stage variable k_i can be done, but is known as discrete sensitivity analysis and is known to be equivalent to automatic differentiation of the solver (Zhang & Sandu, 2014). Thus to calculate the derivative of the solution simultaneously to the derivatives of the solver states, we used direct automatic differentiation of the differential equation solvers for performing the experiments (Innes, 2018). We note that discrete adjoints are known to be more stable than continuous adjoints (Zhang & Sandu, 2014) and in the context of neural ODEs have been shown to stabilize the training process leading to better fits (Gholami et al., 2019; Onken & Ruthotto, 2020). While more memory

intensive than some forms of the continuous adjoint, we note that checkpointing methods can be used to reduce the peak memory (Dauvergne & Hascoët, 2006). We note that this is equivalent to backpropagation of a fixed time step discretization if the step sizes are chosen in advance, and verify in the example code that no additional overhead is introduced.

4. Experiments

In this section, we consider the effectiveness of regularizing Neural Differential Equations (NDEs) on their training and prediction timings. We consider the following baselines while evaluating our models:

1. **Vanilla Neural (O/S)DE** with discrete sensitivities.

2. **STEER**: Temporal Regularization for Neural ODE models by stochastic sampling of the end time during training (Behl et al., 2020).

3. **TayNODE**: Regularizing the Kth order derivatives of the Neural ODEs (Kelly et al., 2020)[2].

We test our regularization on four tasks – supervised image classification (Section 4.1.1) and time series interpolation (Section 4.1.2) using Neural ODE, and fitting Neural SDE (Section 4.2.1) and supervised image classification using Neural SDE (Section 4.2.2). We use DiffEqFlux (Rackauckas et al., 2019) and Flux (Innes et al., 2018) for our experiments.

4.1. Neural Ordinary Differential Equations

In the following experiments, we use a Runge Kutta 5(4) solver (Tsitouras, 2011) with absolute and relative tolerances of 1.4×10^{-8} to solve the ODEs. To measure the prediction time, we use a test batch size equal to the training batch size.

4.1.1. SUPERVISED CLASSIFICATION

Training Details We train a Neural ODE and a Linear Classifier to map flattened MNIST Images to their corresponding labels. Our model uses a two layered neural network f_{θ_1}, as the ODE dynamics, followed by a linear classifier g_{θ_2}, identical to the architecture used in Kelly et al. (2020).

$$z_{\theta_1}(x, t) = \tanh(W_1[x; t] + B_1) \qquad (12)$$

---

[2]We use the original code formulation of the TayNODE in order to ensure usage of the specially-optimized Taylor-mode automatic differentiation technique (Bettencourt et al., 2019) in the training process. Given the large size of the neural networks, most of the compute time lies in optimized BLAS kernels which are the same in both implementations, meaning we do not suspect library to be a major factor in timing differences beyond the AD specifics.

Method	Train Accuracy (%)	Test Accuracy (%)	Train Time (hr)	Prediction Time (s)	NFE
Vanilla NODE	100.0 ± 0.00	97.94 ± 0.02	0.98 ± 0.03	0.094 ± 0.010	253.0 ± 3.46
STEER	100.0 ± 0.00	97.94 ± 0.03	1.31 ± 0.07	0.092 ± 0.002	265.0 ± 3.46
TayNODE	98.98 ± 0.06	97.89 ± 0.00	1.19 ± 0.07	0.079 ± 0.007	080.3 ± 0.43
SRNODE (Ours)	100.0 ± 0.00	98.08 ± 0.15	1.24 ± 0.06	0.094 ± 0.003	259.0 ± 3.46
ERNODE (Ours)	99.71 ± 0.28	97.32 ± 0.06	0.82 ± 0.02	0.060 ± 0.001	177.0 ± 0.00
STEER + *SRNODE*	100.0 ± 0.00	97.88 ± 0.06	1.55 ± 0.27	0.101 ± 0.009	275.0 ± 12.5
STEER + *ERNODE*	99.91 ± 0.02	97.61 ± 0.11	1.37 ± 0.11	0.086 ± 0.018	197.0 ± 9.17
SRNODE + *ERNODE*	99.98 ± 0.03	97.77 ± 0.05	1.37 ± 0.04	0.081 ± 0.006	221.0 ± 17.3

Table 1. **MNIST Image Classification using Neural ODE** Using ERNODE obtains a training and prediction speedup of 16.33% and 37.78% respectively, at only 0.6% reduced prediction accuracy. SRNODE doesn't help in isolation but is effective when combined with ERNODE to reduce the prediction time by 14.44% while incurring a reduced test accuracy of only 0.17%.
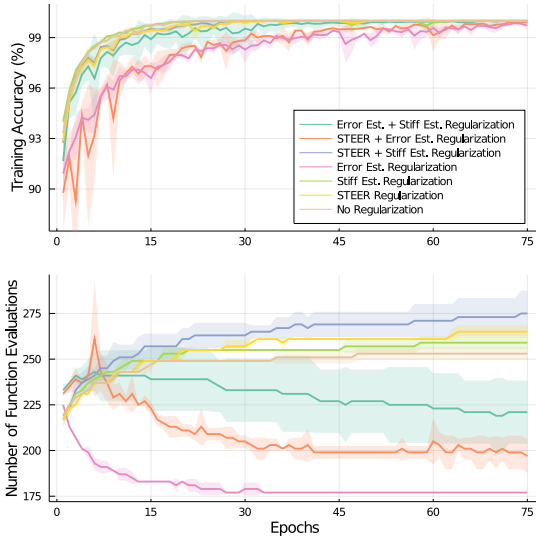

Figure 3. **Number of Function Evaluations and Training Accuracy for Supervised MNIST Classification** Regularizing using ERNODE is the most consistent way to reduce the overall number of function evaluations. Using SRNODE alongside ERNODE stabilizes the training at the cost of increased prediction time.

$$f_{\theta_1}(x, t) = \tanh(W_2[z_{\theta_1}(x, t); t] + B_2) \qquad (13)$$

$$g_{\theta_2}(x, t) = \sigma(W_3 x + B_3) \qquad (14)$$

where the parameters $W_1 \in \mathbb{R}^{100 \times 785}$, $B_1 \in \mathbb{R}^{100}$, $W_2 \in \mathbb{R}^{784 \times 101}$, $B_2 \in \mathbb{R}^{784}$, $W_3 \in \mathbb{R}^{10 \times 784}$, and $B_3 \in \mathbb{R}^{10}$. We use a batch size of 512 and train the model for 75 epochs using Momentum (Qian, 1999) with learning rate of 0.1 and mass of 0.9, and a learning rate inverse decay of 10^{-5} per iteration. For Error Estimate Regularization, we perform exponential annealing of the regularization coefficient from 100.0 to 10.0 over 75 epochs. For Stiffness Regularization, we use a constant coefficient of 0.0285.

Baselines For the STEER baseline, we train the models by stochastically sampling the end time point from $\mathcal{U}(T - b, T + b)$ where $T = 1.0$ and $b = 0.5$[3]. We observe no

---

[3]$b = 0.25$ was also considered but final results were comparable

training improvement but there is a minor improvement in prediction time. For the TayNODE baseline, we train the model with a reduced batch size of 100[4], $\lambda = 3.02 \times 10^{-3}$, and regularizing 3rd order derivatives.

Results Figure 3 visualizes the training accuracy and number of function evaluations over training. Table 1 summarizes the metrics from the trained baseline and proposed models – Error Estimate Regularized Neural ODE (*ERNODE*) and Stiffness Regularized Neural ODE (*SRNODE*). Additionally, we perform ablation studies by composing various regularization strategies.

4.1.2. TIME SERIES INTERPOLATION

Training Details We use the Latent ODE (Chen et al., 2018) model with RNN encoder to learn the trajectories for ICU Patients for Physionet Challenge 2012 Dataset (Silva et al., 2012). We use the preprocessed data provided by Kelly et al. (2020) to ensure consistency in results. For every independent run, we perform an $80 : 20$ split of the data for training and evaluation.

Our model architecture is similar to the encoder-decoder models used in Rubanova et al. (2019). We use a 20-dimensional latent state and a 40-dimensional hidden state for the recognition model. Our ODE dynamics is given by a 4-layered neural network with 50 units and tanh activation. We train our models for 300 epochs with a batchsize of 512 and using Adamax (Kingma & Ba, 2014) with a learning rate of 0.01 and an inverse decay of 10^{-5}. We minimize the negative log likelihood of the predictions and perform KL annealing with a coefficient of 0.99.

For Error Estimate Regularization, we perform exponential annealing of the regularization coefficient from 1000.0 to 100.0 over 300 epochs. We note that using $R_E = \sum_j E_j^2$, instead of $R_E = \sum_j E_j |h_j|$, yields similar results with a constant regularization coefficient of 100.0. For Stiffness Regularization, we use a constant coefficient of 0.285.

---

[4]Batch Size was reduced to ensure we reach a comparable train/test accuracy as the other trained models.

Method	Train Loss ($\times 10^{-3}$)	Test Loss ($\times 10^{-3}$)	Train Time (hr)	Prediction Time (s)	NFE
Vanilla NODE	3.48 ± 0.00	3.55 ± 0.00	1.75 ± 0.39	0.53 ± 0.12	733.0 ± 84.29
STEER	3.43 ± 0.02	3.48 ± 0.01	1.62 ± 0.26	0.54 ± 0.06	699.0 ± 141.1
TayNODE	4.21 ± 0.02	4.21 ± 0.01	12.3 ± 0.32	0.22 ± 0.02	167.3 ± 11.93
SRNODE (Ours)	3.52 ± 1.44	3.58 ± 0.05	0.87 ± 0.09	0.20 ± 0.01	273.0 ± 0.000
ERNODE (Ours)	3.51 ± 0.00	3.57 ± 0.00	0.94 ± 0.13	0.21 ± 0.02	287.0 ± 17.32
STEER + *SRNODE*	3.67 ± 0.02	3.73 ± 0.02	0.89 ± 0.08	0.20 ± 0.01	271.0 ± 12.49
STEER + *ERNODE*	3.41 ± 0.02	3.48 ± 0.01	1.03 ± 0.25	0.24 ± 0.05	269.0 ± 33.05
SRNODE + *ERNODE*	3.48 ± 0.11	3.56 ± 0.03	1.12 ± 0.08	0.21 ± 0.01	263.0 ± 12.49

Table 2. **Physionet Time Series Interpolation** All the regularized variants of Latent ODE (except STEER) have comparable prediction times. Additionally, the training time is reduced by $36\% - 50\%$ on using one of our proposed regularizers, while TayNODE increases the training time by 7x. Overall, SRNODE has the best training and prediction timings while incurring an increased 0.85% test loss.
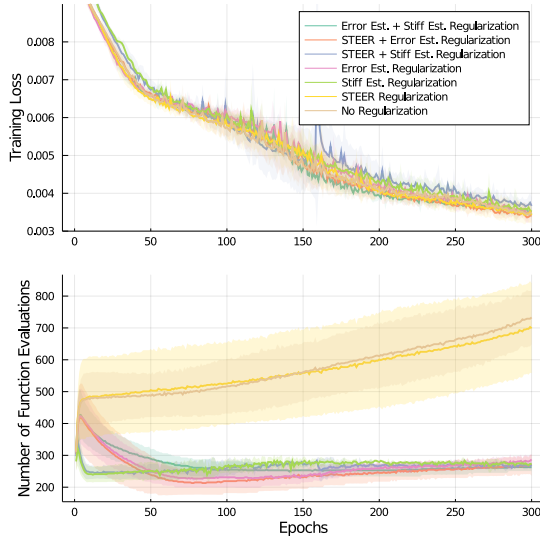

Figure 4. **Number of Function Evaluations and Training Loss for Physionet Time Series Interpolation** Regularized and Unregularized variants of the model have very similar trajectories for the training loss. We do notice a significant difference in the NFE plot. Using either Error Estimate Regularization or Stiffness Regularization is able to bound the NFE to < 300, compared to ~ 700 for STEER or unregularized Latent ODE.

Baselines For STEER Baseline, we stochastically sample the timestep to evaluate the difference between interpolated and ground truth data. Essentially for the interval (t_i, t_{i+1}), we evaluate the model at $\mathcal{U}(t_{i+1} - \frac{t_{i+1}-t_i}{2}, t_{i+1} + \frac{t_{i+1}-t_i}{2})$ and compare with the truth at t_{i+1}. We sample end points after every iteration of the model. STEER reduces the training time but has no significant effect on the prediction time. TayNODE was trained by regularizing the 2nd order derivatives and a coefficient of 0.01 for 300 epochs and a batchsize of 512. TayNODE had an exceptionally high training time $\sim 7\times$ compared to the unregularized baseline.

Results Figure 4 shows the training MSE loss and the NFE counts for the considered models. Table 2 summarizes the metrics and wall clock timings for the baselines, proposed regularizers and their compositions with previously pro-

posed regularizers. We observe that SRNODE provides the most significant speedup while ERNODE attains similar losses at slightly higher training and prediction times.

4.2. Neural Stochastic Differential Equations

In these experiments, we use SOSRI/SOSRI2 (Rackauckas & Nie, 2020) to solve the Neural SDEs. The wall clock timings represent runs on a CPU.

4.2.1. FITTING SPIRAL DIFFERENTIAL EQUATION

Training Details In this experiment, we consider training a Neural SDE to mimic the dynamics of the Spiral Stochastic Differential Equation with Diagonal Noise (DSDE). Spiral DSDE is prescribed by the following equations:

$$
\begin{aligned}
du_1 &= -\alpha u_1^3 dt + \beta u_2^3 dt + \gamma u_1 dW \\
du_2 &= -\beta u_1^3 dt - \alpha u_2^3 dt + \gamma u_2 dW
\end{aligned}
\tag{15}
$$

where $\alpha = 0.1$, $\beta = 2.0$, and $\gamma = 0.2$. We generate data across 10000 trajectories at 30 uniformly spaced points between $t \in [0, 1]$ (Figure 5). We parameterize our drift and diffusion functions using neural networks f_θ and g_ϕ via:

$$
\begin{aligned}
f_\theta(x, t) &= W_2 \tanh(W_1 x^3 + B_1) + B_2 \\
g_\phi(x, t) &= W_3 x + B_3
\end{aligned}
\tag{16}
$$

where the parameters $W_1 \in \mathbb{R}^{50 \times 2}$, $B_1 \in \mathbb{R}^{50}$, $W_2 \in \mathbb{R}^{2 \times 50}$, $B_2 \in \mathbb{R}^2$, $W_3 \in \mathbb{R}^{2 \times 2}$, and $B_3 \in \mathbb{R}^2$. For fitting the drift and diffusion functions to the simulated data, we used a generalized method of moments loss function (Lück & Wolf, 2016; Jeisman, 2006). Our objective is to train these parameters to minimize the L_2 distance between the mean (μ) and variance (σ^2) of predicted and real data. Let, $\hat{\mu}_i$'s and $\hat{\sigma}_i^2$'s denote the means and variances respectively of the multiple predicted trajectories.

$$
\mathcal{L}(u_0; \theta, \phi) = \sum_{i=1}^{30} \left[(\mu_i - \hat{\mu}_i)^2 + (\sigma_i^2 - \hat{\sigma}_i^2)^2 \right] + \lambda_r R_E
\tag{17}
$$

Method	Mean Squared Loss	Train Time (s)	Prediction Time (s)	NFE
Vanilla NSDE	0.0217 ± 0.0088	178.95 ± 20.22	0.07553 ± 0.0186	528.67 ± 6.11
SRNSDE (Ours)	0.0204 ± 0.0091	166.42 ± 14.51	0.07250 ± 0.0017	502.00 ± 4.00
ERNSDE (Ours)	0.0227 ± 0.0090	173.43 ± 04.18	0.07552 ± 0.0008	502.00 ± 4.00

Table 3. **Spiral SDE** The ERNSDE attains a relative loss of 4% compared to vanilla Neural SDE while reducing the training time and number of function evaluations. Using SRNSDE reduces both the training and prediction times by 7% and 4% respectively.
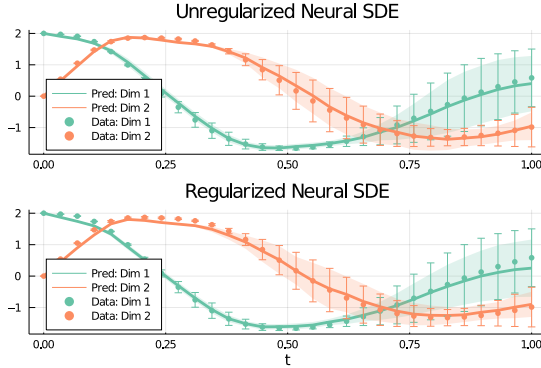

Figure 5. **Fitting a Neural SDE on Spiral SDE Data.** Regularizing has minimal effect on the learned dynamics with reduced training and prediction cost.

The models were trained using AdaBelief Optimizer (Zhuang et al., 2020) with a learning rate of 0.01 for 250 iterations. We generate 100 trajectories for each iteration to compute the $\hat{\mu}_i$s and $\hat{\sigma}_i^2$s.

Results Table 3 summarizes the final results for the trained models for 3 different random seeds. We notice that even for this "toy" problem, we can marginally improve training time while incurring a minimal penalty on the final loss.

4.2.2. SUPERVISED CLASSIFICATION

Training Details We train a Neural SDE model to map flattened MNIST Images to their corresponding labels. Our diffusion function uses a two layered neural network f_{θ_2} and the drift function is a linear map g_{θ_3}. We use two additional linear maps – a_{θ_1} mapping the flattened image to the hidden dimension and b_{θ_4} mapping the output of the Neural SDE to the logits.

$$a_{\theta_1}(x, t) = W_1 x + B_1 \tag{18}$$
$$f_{\theta_2}(x, t) = W_3 \tanh(W_2 x + B_2) + B_3 \tag{19}$$
$$g_{\theta_3}(x, t) = W_4 x + B_4 \tag{20}$$
$$b_{\theta_4}(x, t) = W_5 x + B_5 \tag{21}$$

where the parameters $W_1 \in \mathbb{R}^{32 \times 784}$, $B_1 \in \mathbb{R}^{32}$, $W_2 \in \mathbb{R}^{32 \times 64}$, $B_2 \in \mathbb{R}^{64}$, $W_3 \in \mathbb{R}^{32 \times 64}$, $B_3 \in \mathbb{R}^{32}$, $W_4 \in \mathbb{R}^{10 \times 32}$, and $B_3 \in \mathbb{R}^{10}$. We use a batch size of 512 and train the model for 40 epochs using Adam (Kingma & Ba, 2014) with learning rate of 0.01, and an inverse decay of 10^{-5} per iteration. While making predictions we use the
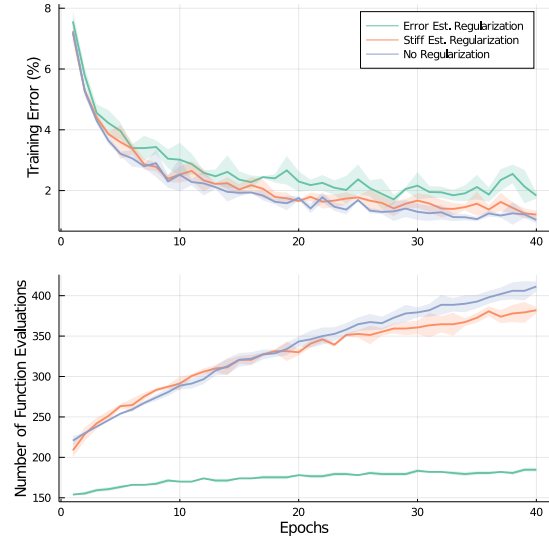

Figure 6. **Number of Function Evaluations and Training Error for Supervised MNIST Classification using Neural SDE** ERNSDE reduces the NFE below 300 with minimal error change while the unregularized version has NFE ~ 400.

mean logits across 10 trajectories. For Error Estimate and Stiffness Regularization, we use constant coefficients 10.0 and 0.1 respectively.

Results Figure 6 shows the variation in NFE and Training Error during training. Table 4 summarizes the final metrics and timings for all the trained models. We observe that SRNSDE doesn't improve the training/prediction time, similar to the MNIST Neural ODE Experiment 4.1.1. However, ERNSDE gives us a training and prediction speedup of 33.7% and 52.02% respectively, at the cost of 0.7% reduced test accuracy.

5. Discussion

Numerical analysis has had over a century of theoretical developments leading to efficient adaptive methods for solving many common nonlinear equations such as differential equations. Here we demonstrate that by using the knowledge embedded within the heuristics of these methods we can accelerate the training process of neural ODEs.

We note that on the larger sized PhysioNet and MNIST examples we saw significant speedups while on the smaller differential equation examples we saw only minor perfor-

Method	Train Accuracy (%)	Test Accuracy (%)	Train Time (hr)	Prediction Time (s)	NFE
Vanilla NSDE	98.97 ± 0.11	96.95 ± 0.11	6.32 ± 0.19	15.07 ± 0.93	411.33 ± 6.11
SRNSDE (Ours)	98.79 ± 0.12	96.80 ± 0.07	8.54 ± 0.37	14.50 ± 0.40	382.00 ± 4.00
ERNSDE (Ours)	98.16 ± 0.11	96.27 ± 0.35	4.19 ± 0.04	07.23 ± 0.14	184.67 ± 2.31

Table 4. **MNIST Image Classification using Neural SDE** ERNSDE obtains a training and prediction speedup of 33.7% and 52.02% respectively, at only 0.7% reduced prediction accuracy.

mance improvements. This showcases how the NFE becomes a better estimate of the total compute time as the cost of the ODE f (and SDE g) increase when the model size increases.

This result motivates efforts in differentiable programming (Wang et al., 2018; Abadi & Plotkin, 2019; Rackauckas et al.) which enables direct differentiation of solvers since utilizing the solver's heuristics may be crucial in the development of advanced techniques. This idea could be straightforwardly extended not only to other forms of differential equations, but also to other "implicit layer" machine learning methods. For example, Deep Equilibrium Models (DEQ) (Bai et al., 2019) model the system as the solution to an implicit function via a nonlinear solver like Bryoden or Newton's method. Heuristics like the ratio of the residuals have commonly been used as a convergence criterion and as a work estimate for the difficulty of solving a particular nonlinear equation (Wanner & Hairer, 1996), and thus could similarly be used to regularize for learning DEQs whose forward passes are faster to solve. Similarly, optimization techniques such as BFGS (Kelley, 1999) contain internal estimates of the Hessian which can be used to regularize the stiffness of "optimization as layers" machine learning architectures like OptNet (Amos & Kolter, 2017). However, in these cases we note that continuous adjoint techniques have a significant computational advantage over discrete adjoint methods because the continuous adjoint method can be computed directly at the point of the solution while discrete adjoints would require differentiating through the iteration process. Thus while a similar regularization would exist in these contexts, in the case of differential equations the continuous and discrete adjoints share the same computational complexity which is not the case in methods which iterate to convergence. Further study of these applications would be required in order to ascertain the effectiveness in accelerating the training process, though by extrapolation one may guess that at least the forward pass would be accelerated.

6. Limitations

While these experiments have demonstrated major performance improvements, it is pertinent to point out the limitations of the method. One major point to note is that this only applies to learning neural ODEs for maps $z(0) \mapsto z(1)$ as is used in machine learning applications of the architecture (Chen et al., 2018). Indeed, a neural ODE as an "implicit layer" for predictions in machine learning does not

require identification of dynamical mechanisms. However, if the purpose is to learn the true governing dynamics a physical system from timeseries data, this form of regularization would bias the result, dampening higher frequency responses leading to an incorrect system identification. Approaches which embed neural networks into solvers could be used in such cases (Shen et al., 2020; Poli et al., 2020). Indeed we note that such Hypereuler approaches could be combined with the ERNODE regularization on machine learning prediction problems, which could be a fruitful avenue of research. Lastly, we note that while either the local error and stiffness regularization was effective on each chosen equation, neither was effective on all equations and at this time there does not seem to be a clear a priori indicator as to which regularization is necessary for a given problem. While it seems the error regularization was more effective on the image classification tasks while the stiffness regularization was more effective on the time series task, we believe more experiments will be required in order to ascertain whether this is a common phenomena, possibly worthy of theoretical investigation.

7. Conclusion

Our studies reveal that error estimate regularization provides a consistent way to improve the training/prediction time of neural differential equations. In our experiments, we see an average improvement of 1.4x training time and 1.8x prediction time on using error estimate regularization. Overall we provide conclusive evidence that cheap and accurate cost estimates obtained by white-boxing differential equation solvers can be as effective as expensive higher-order regularization strategies. Together these results demonstrate a generalizable idea for how to combine differentiable programming with algorithm heuristics to improve training speeds in a way that cannot be done with continuous adjoint techniques. Thus, even if a derivative can be defined for a given piece of code, our approach shows that differentiating the solver can still have major advantages because the solver internal details in terms of stability and performance.

8. Acknowledgements

References

Abadi, M. and Plotkin, G. D. A simple differentiable programming language. *Proceedings of the ACM on Programming Languages*, 4(POPL):1–28, 2019.

Amos, B. and Kolter, J. Z. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pp. 136–145. PMLR, 2017.

Ascher, U. M. and Petzold, L. R. *Computer methods for ordinary differential equations and differential-algebraic equations*, volume 61. Siam, 1998.

Bai, S., Kolter, J. Z., and Koltun, V. Deep equilibrium models. *arXiv preprint arXiv:1909.01377*, 2019.

Behl, H., Ghosh, A., Dupont, E., Torr, P., and Namboodiri, V. Steer: simple temporal regularization for neural odes. pp. 1–13. Neural Information Processing Systems Foundation, Inc., 2020.

Bettencourt, J., Johnson, M. J., and Duvenaud, D. Taylor-mode automatic differentiation for higher-order derivatives in jax. 2019.

Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. B. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017. doi: 10.1137/141000671.

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

Dauvergne, B. and Hascoët, L. The data-flow equations of checkpointing in reverse automatic differentiation. In *International Conference on Computational Science*, pp. 566–573. Springer, 2006.

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.

Fehlberg, E. *Classical fifth-, sixth-, seventh-, and eighth-order Runge-Kutta formulas with stepsize control*. National Aeronautics and Space Administration, 1968.

Finlay, C., Jacobsen, J.-H., Nurbekyan, L., and Oberman, A. M. How to train your neural ode. *arXiv preprint arXiv:2002.02798*, 2020.

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., Bettencourt, J., Sutskever, I., and Duvenaud, D. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

Hairer, E., Norsett, S., and Wanner, G. *Solving Ordinary Differential Equations I: Nonstiff Problems*, volume 8. 01 1993. ISBN 978-3-540-56670-0. doi: 10.1007/978-3-540-78862-1.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition, 2015.

Higham, D. J. and Trefethen, L. N. Stiffness of odes. *BIT Numerical Mathematics*, 33(2):285–303, 1993.

Hutchinson, M. F. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 18(3):1059–1076, 1989.

Innes, M. Don't unroll adjoint: Differentiating ssa-form programs. *arXiv preprint arXiv:1810.07951*, 2018.

Innes, M., Saba, E., Fischer, K., Gandhi, D., Rudilosso, M. C., Joy, N. M., Karmali, T., Pal, A., and Shah, V. Fashionable modelling with flux. *CoRR*, abs/1811.01457, 2018. URL http://arxiv.org/abs/1811.01457.

Jeisman, J. I. *Estimation of the parameters of stochastic differential equations*. PhD thesis, Queensland University of Technology, 2006.

Kelley, C. T. *Iterative methods for optimization*. SIAM, 1999.

Kelly, J., Bettencourt, J., Johnson, M. J., and Duvenaud, D. Learning differential equations that are easy to solve. *arXiv preprint arXiv:2007.04504*, 2020.

Kidger, P., Chen, R. T. Q., and Lyons, T. "hey, that's not an ode": Faster ode adjoints with 12 lines of code, 2020.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization, 2014. URL http://arxiv.org/abs/1412.6980. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

Kutta, W. Beitrag zur naherungsweisen integration totaler differentialgleichungen. *Z. Math. Phys.*, 46:435–453, 1901.

Liu, X., Xiao, T., Si, S., Cao, Q., Kumar, S., and Hsieh, C.-J. Neural sde: Stabilizing neural ode networks with stochastic noise, 2019.

Lu, Y., Zhong, A., Li, Q., and Dong, B. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In *International Conference on Machine Learning*, pp. 3276–3285. PMLR, 2018.

Lück, A. and Wolf, V. Generalized method of moments for estimating parameters of stochastic reaction networks. *BMC systems biology*, 10(1):1–12, 2016.

Onken, D. and Ruthotto, L. Discretize-optimize vs. optimize-discretize for time-series regression and continuous normalizing flows. *arXiv preprint arXiv:2005.13420*, 2020.

Poli, M., Massaroli, S., Yamashita, A., Asama, H., and Park, J. Hypersolvers: Toward fast continuous-depth models. *arXiv preprint arXiv:2007.09601*, 2020.

Qian, N. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

Rackauckas, C. and Nie, Q. Adaptive methods for stochastic differential equations via natural embeddings and rejection sampling with memory. *Discrete and continuous dynamical systems. Series B*, 22(7):2731, 2017.

Rackauckas, C. and Nie, Q. Confederated modular differential equation apis for accelerated algorithm development and benchmarking. *Advances in Engineering Software*, 132:1–6, 2019.

Rackauckas, C. and Nie, Q. Stability-optimized high order methods and stiffness detection for pathwise stiff stochastic differential equations. In *2020 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–8. IEEE, 2020.

Rackauckas, C., Edelman, A., Fischer, K., Innes, M., Saba, E., Shah, V. B., and Tebbutt, W. Generalized physics-informed learning through language-wide differentiable programming.

Rackauckas, C., Innes, M., Ma, Y., Bettencourt, J., White, L., and Dixit, V. Diffeqflux.jl - A julia library for neural differential equations. *CoRR*, abs/1902.02376, 2019. URL http://arxiv.org/abs/1902.02376.

Rackauckas, C., Ma, Y., Martensen, J., Warner, C., Zubov, K., Supekar, R., Skinner, D., Ramadhan, A., and Edelman, A. Universal differential equations for scientific machine learning, 2020.

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

Runge, C. Über die numerische auflösung von differentialgleichungen. *Mathematische Annalen*, 46(2):167–178, 1895.

Shampine, L. F. Stiffness and nonstiff differential equation solvers, ii: Detecting stiffness with runge-kutta methods. *ACM Trans. Math. Softw.*, 3(1):44–53, March 1977. ISSN 0098-3500. doi: 10.1145/355719.355722. URL https://doi.org/10.1145/355719.355722.

Shampine, L. F. and Gear, C. W. A user's view of solving stiff ordinary differential equations. *SIAM review*, 21(1): 1–17, 1979.

Shampine, L. F. and Thompson, S. Stiff systems. *Scholarpedia*, 2(3):2855, 2007.

Shen, X., Cheng, X., and Liang, K. Deep euler method: solving odes by approximating the local truncation error of the euler method. *arXiv preprint arXiv:2003.09573*, 2020.

Silva, I., Moody, G., Scott, D. J., Celi, L. A., and Mark, R. G. Predicting in-hospital mortality of icu patients: The physionet/computing in cardiology challenge 2012. In *2012 Computing in Cardiology*, pp. 245–248. IEEE, 2012.

Tsitouras, C. Runge–kutta pairs of order 5(4) satisfying only the first column simplifying assumption. *Computers Mathematics with Applications*, 62(2):770 – 775, 2011. ISSN 0898-1221. doi: https://doi.org/10.1016/j.camwa.2011.06.002. URL http://www.sciencedirect.com/science/article/pii/S0898122111004706.

Wang, F., Decker, J., Wu, X., Essertel, G., and Rompf, T. Backpropagation with callbacks: Foundations for efficient and expressive differentiable programming. *Advances in Neural Information Processing Systems*, 31: 10180–10191, 2018.

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

Zhang, H. and Sandu, A. Fatode: a library for forward, adjoint, and tangent linear integration of odes. *SIAM Journal on Scientific Computing*, 36(5):C504–C523, 2014.

Zhang, H., Xue, Y., Zhang, C., and Dong, L. Computing the high order derivatives with automatic differentiation and its application in chebyshev's method. In *2008 Fourth International Conference on Natural Computation*, volume 1, pp. 304–308. IEEE, 2008.

Zhuang, J., Tang, T., Ding, Y., Tatikonda, S., Dvornek, N., Papademetris, X., and Duncan, J. Adabelief optimizer: Adapting stepsizes by the belief in observed gradients. *Conference on Neural Information Processing Systems*, 2020.

Zhuang, J., Dvornek, N. C., sekhar tatikonda, and s Duncan, J. {MALI}: A memory efficient and reverse accurate integrator for neural {ode}s. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=blfSjHeFM_e.