Locally Regularized Neural Differential Equations: Some Black Boxes Were Meant to Remain Closed!

Avik Pal [1 2] **Alan Edelman** [1 3] **Christopher Rackauckas** [1]

Abstract

Implicit layer deep learning techniques, like Neural Differential Equations, have become an important modeling framework due to their ability to adapt to new problems automatically. Training a neural differential equation is effectively a search over a space of plausible dynamical systems. However, controlling the computational cost for these models is difficult since it relies on the number of steps the adaptive solver takes. Most prior works have used higher-order methods to reduce prediction timings while greatly increasing training time or reducing both training and prediction timings by relying on specific training algorithms, which are harder to use as a drop-in replacement due to strict requirements on automatic differentiation. In this manuscript, *we use internal cost heuristics of adaptive differential equation solvers at stochastic time-points to guide the training towards learning a dynamical system that is easier to integrate*. We "close the blackbox" and allow the use of our method with any adjoint technique for gradient calculations of the differential equation solution. We perform experimental studies to compare our method to global regularization to show that we attain similar performance numbers without compromising on the flexibility of implementation on ordinary differential equations (ODEs) and stochastic differential equations (SDEs). *We develop two sampling strategies to trade-off between performance and training time*. Our method reduces the number of function evaluations to $0.556 \times -0.733\times$ and accelerates predictions by $1.3 \times -2\times$.

[1]CSAIL MIT [2]Department of Electrical Engineering and Computer Science MIT [3]Department of Mathematics MIT. Correspondence to: Avik Pal <avikpal@mit.edu>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

1. Introduction

Implicit Models, such as Neural Ordinary Differential Equations (Chen et al., 2018) and Deep Equilibrium Models (Bai et al., 2019; Pal et al., 2022), have emerged as a promising technique to determine the depth of neural networks automatically. To maximize performance on a dataset, explicit models are tuned to the "hardest" training sample, which hurts the inference timings for "easier" – more abundant – samples. Using adaptive differential equation solvers allow these implicit models to choose the number of steps they need effectively. This idea of representing neural networks as ODEs has since been generalized to Stochastic Differential Equations (Liu et al., 2019; Rackauckas et al., 2020) and other architectures to improve robustness.

Despite the rapid progress in these methods, the core problem of the scalability of these models is still persistent. Several solutions to these have been proposed:

- Kelly et al. (2020); Finlay et al. (2020) use higher order derivatives for regularization.
- Poli et al. (2020) learn neural solvers to solve Neural ODEs faster.
- Pal et al. (2021) proposed a "zero-cost" global regularization scheme.
- Behl et al. (2020) randomize the integration stop time to "smoothen" the dynamics.

All these methods have definite tradeoffs (See Section 5 for more details).

This paper presents a generally applicable method to force the neural differential equation training process to choose the least expensive option. We build upon the global regularization scheme proposed in Pal et al. (2021) and "close" the blackbox allowing our method to work across various sensitivity algorithms. Our main contributions include the following[1]:

[1]Our code is publicly available at `https://github.com/avik-pal/LocalRegNeuralDE.jl`

Sensitivity Algorithm	Memory Requirement	Memory Requirement with Local Regularization
Backsolve Adjoint (Chen et al., 2018)	$\mathcal{O}(s)$	$\mathcal{O}(s \times (1 + \text{stages}))$
Backsolve Adjoint with Checkpointing (Chen et al., 2018)	$\mathcal{O}(s \times c)$	$\mathcal{O}(s \times (c + \text{stages}))$
Interpolating Adjoint (Hindmarsh et al., 2005)	$\mathcal{O}(s \times t)$	$\mathcal{O}(s \times (t + \text{stages}))$
Interpolating Adjoint with Checkpointing (Hindmarsh et al., 2005)	$\mathcal{O}(s \times c)$	$\mathcal{O}(s \times (c + \times\text{stages}))$
Quadrature Adjoint (Kim et al., 2021)	$\mathcal{O}((s + p) \times t)$	$\mathcal{O}((s + p) \times t + s \times \text{stages})$
Direct Reverse Mode Differentiation	$\mathcal{O}(s \times t \times \text{stages})$	$\mathcal{O}(s \times (t + 1) \times \text{stages})$

Table 1: **Memory Requirements for various Sensitivity Algorithms for ODEs with Local Regularization**: Local Regularization requires $\mathcal{O}(s \times (c + \text{stages}))$ memory in the best case, while Global Regularization (Pal et al., 2021) always requires $\mathcal{O}(s \times t \times \text{stages})$ memory.

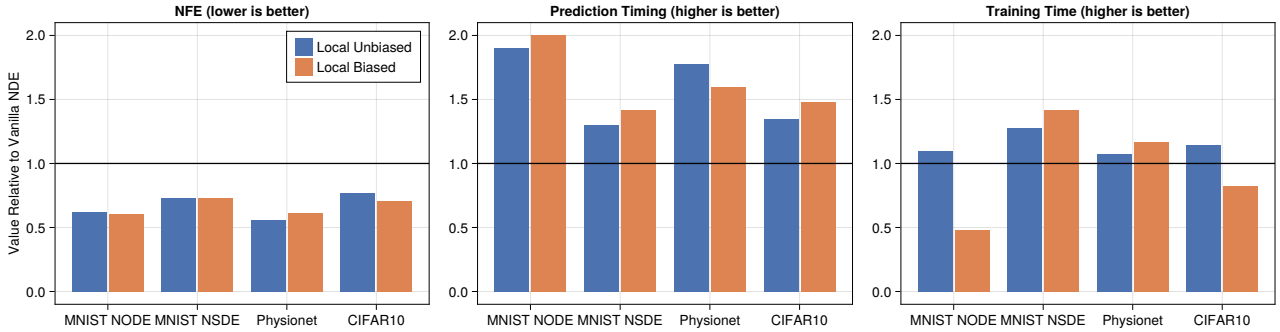

Figure 1: **Locally Regularized NDE leads to faster predictions and faster training compared to vanilla NDE.**

1. We show that our local regularization method – building upon the primitives proposed in Pal et al. (2021) – performs at par with global regularization (See Section 4).

2. We present two sampling methods that trade-off small computational costs for consistently better performance (See Section 3).

3. Using local regularization allows our models to leverage optimize-then-discretize in the backward pass (in addition to discretize-then-optimize methods). Our method works around the several engineering limitations of automatic differentiation (AD) systems (Rackauckas, 2022) that are needed to make global regularization work.

4. We empirically show that regularizing solver heuristics with biased sampling stabilizes the training of larger neural ODEs (See Section 4.3).

2. Background

2.1. Neural Ordinary Differential Equations

Neural ODEs use an explicit neural network to parameterize the dynamical system. These involve finding the state at a later time t_1, given the value z_0 at time t_0. The final 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. Prior works had observed the similarity between fixed time-step discretization of ODEs and Residual Neural Networks Lu et al. (2018), which was later extended in the Neural ODE framework by Chen et al. (2018).

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

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 offers 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. SDE noise can have different variations; however, we exclusively focus on diagonal multiplicative noise for this paper. Liu et al. (2019) modified Neural ODEs by stochastic noise injection

in the form of Neural SDEs. Liu et al. (2019) empirically showed that stochastic noise injection improves the robustness and generalization performance of neural ODEs. In Neural SDEs, we simultaneously train two explicit neural networks – drift $f_\theta(z(t), t)$ and diffusion $g_\phi(z(t), t))$ s.t.:

$$dz(t) = f_\theta(z(t), t)dt + g_\phi(z(t), t))dW$$

2.3. Adaptive Time Stepping for Differential Equations

Runge-Kutta (RK) Methods (Runge, 1895; Kutta, 1901) are widely used to approximate the solutions of ordinary differential equations numerically. Given a tableau of coefficients $\{A, c, b\}$, these methods combine s stages to obtain the estimate at $t + dt$.

$$k_s = f\left(t + c_s \cdot dt, z(t) + \sum_{i=1}^{s-1} a_{si} \cdot k_i\right)$$

$$z(t + dt) = z(t) + dt \cdot \left(\sum_{i=1}^{s} b_i \cdot k_i\right)$$

Adaptive solvers need to maximize the step size (dt) while keeping the error estimate below the user-specified tolerances, i.e., they need to satisfy:

$$\mathtt{E}_{\mathtt{Est}} \leq \mathtt{atol} + \mathtt{max}\left(|z(t)|, |z(t + dt)|\right) \cdot \mathtt{rtol}$$

where $\mathtt{E}_{\mathtt{Est}}$ is the local error estimate. Adaptive solvers utilize an additional linear combiner $\tilde{b}_i$ to get an alternate solution, typically with one order less convergence (Wanner & Hairer, 1996; Fehlberg, 1968; Dormand & Prince, 1980; Tsitouras, 2011).

$$\tilde{z}(t + dt) = z(t) + dt \cdot \left(\sum_{i=1}^{s} \tilde{b}_i \cdot k_i\right)$$

A classic result from Richardson extrapolation shows that $\mathtt{E}_{\mathtt{Est}} = \|\tilde{z}(t + dt) - z(t + dt)\|$ is an estimate of the local truncation error (Hairer et al., 1993; Ascher & Petzold, 1998). The new step size is determined using the following:

$$q = \left\| \frac{\mathtt{E}_{\mathtt{Est}}}{\mathtt{atol} + \mathtt{max}\left(|z(t)|, |z(t + dt)|\right) \cdot \mathtt{rtol}} \right\|$$

- If $q < 1$, dt is accepted.
- Otherwise, it is rejected and reduced. A standard PI controller proposes the new step size to be:

  $$dt_{new} = \eta \cdot q_{n-1}^{\alpha} \cdot q_n^{\beta} \cdot dt$$

  where η is the safety factor, q_{n-1} is the error proportion of the previous step, and (α, β) are tunable PI-gain hyperparameters (Wanner & Hairer, 1996).

We defer the discussion of error estimation schemes for stochastic RK integrators to Rackauckas & Nie (2017; 2020).

2.4. Global Regularization using Local Error Estimates

In Section 2.3, we observe how larger local error estimates $\mathtt{E}_{\mathtt{Est}}$ contribute to reduced step sizes and, in turn, higher training and prediction times for Neural Differential Equations. Pal et al. (2021) proposed a regularization scheme to minimize the "total local error in order to learn Neural ODEs with as large step sizes as possible." They compute the regularization term $(\mathcal{R}_E)_g$ as:

$$(\mathcal{R}_E)_g = \sum_j (\mathtt{E}_{\mathtt{Est}})_j \cdot |dt_j|$$

where the summation is over the j time steps of the solution. All the local error estimates $\mathtt{E}_{\mathtt{Est}}$ are precomputed by adaptive differential equation solvers, which makes the forward pass effectively have zero additional overhead.

Continuous adjoint methods (Chen et al., 2018) define the derivatives in terms of the ODE quantities. The terms $(\mathtt{E}_{\mathtt{Est}})_j = \sum_{i=1}^{s}(b_i - \tilde{b}_i) \cdot k_i$ cannot be constructed directly from the trajectory of the ODE solution, since k_i terms are defined by the solver method (and not by the continuous ODE solution). As a result Pal et al. (2021) performs discrete sensitivity analysis to compute the derivatives w.r.t. the regularization terms.

Discrete Sensitivity analysis is more stable than continuous adjoints (Zhang & Sandu, 2014) and stabilizes the training process of Neural ODEs (Onken & Ruthotto, 2020). However, these methods are more memory intensive than continuous adjoints. Another shortcoming is the advanced tooling mechanism needed to differentiate through differential equation solvers (Rackauckas, 2022), making this method hard to adopt. Since adaptive ODE solvers change the number of steps based on the values in the differential equation, discrete sensitivity analysis requires dynamic AD tooling which is generally less optimized than those built for static graph cases. We note that there are checkpointing methods to reduce the memory overhead (Dauvergne & Hascoët, 2006), though these generally have focused on static graph AD systems.

2.5. Multiscale Neural ODE with Input Injection

Multiscale modeling (Burt & Adelson, 1987) has been the central theme for several deep computer vision applications (Farabet et al., 2012; Yu & Koltun, 2015; Chen et al., 2016; 2017). Standard Neural ODEs set the initial condition as input x from the previous layer. Multiscale Neural ODE with Input Injection is based on the Multiscale Deep Equilibrium Models (DEQs) (Bai et al., 2019; 2020; Pal et al., 2022). If our model operates at n scales, the initial condition $\{(z_0)_i\}_{i \in [n-1]}$ is defined as $(z_0)_0 = x$ and $(z_0)_{i \in \{1, \ldots n-1\}} = 0$. Every feature set is upsampled or downsampled and fed into the other scales using a neural network. In line with Multiscale DEQs, we also repeatedly

Algorithm 1 Unbiased Sampling: Training Phase

Data: $\mathtt{x}$, f_θ, t_{span}
Result: $\mathtt{sol}$, $\mathtt{r}$
Define $\frac{du}{dt} = f_\theta(u, t)$
$t_0, t_1 \leftarrow t_{span}$
$\mathtt{t_{reg}} \sim \mathbb{U}[t_0, t_1]$
$\mathtt{sol} \leftarrow \mathtt{solve}(\frac{du}{dt},\ \mathtt{DE\ Solver},\ t_{span})$
$u_{\mathtt{t_{reg}}} \leftarrow \mathtt{sol}(\mathtt{t_{reg}})$
Run single step for the solver with time-span $(\mathtt{t_{reg}}, t_1)$
$\mathtt{r} \leftarrow$ Local Error Estimate $@t = \mathtt{t_{reg}}$

Algorithm 2 Biased Sampling: Training Phase

Data: $\mathtt{x}$, f_θ, t_{span}
Result: $\mathtt{sol}$, $\mathtt{r}$
Define $\frac{du}{dt} = f_\theta(u, t)$
$t_0, t_1 \leftarrow t_{span}$
$\mathtt{sol} \leftarrow \mathtt{solve}(\frac{du}{dt},\ \mathtt{DE\ Solver},\ t_{span})$
$\mathtt{t_{reg}} \sim \mathbb{U}(\mathtt{sol}.t)$
$u_{\mathtt{t_{reg}}} \leftarrow \mathtt{sol}(\mathtt{t_{reg}})$
Run single step for the solver with time-span $(\mathtt{t_{reg}}, t_1)$
$\mathtt{r} \leftarrow$ Local Error Estimate $@t = \mathtt{t_{reg}}$

inject the original input x at every step of the dynamical system.

3. Methods

In Section 2.4, we discussed the downsides of using global regularization with local error estimates. To summarize:

1. Global Regularization relies on discrete sensitivity analysis, which is *more memory intensive*.
2. Global Regularization depends on AD tooling to support dynamic compute graphs in an efficient way, making it *hard to incorporate into existing codebases*.

To get around these limitations, we developed a new technique using local sampling of error estimates at specific time points, rather than globally over the full interval. We deal with sampling the "appropriate" time point for regularization by two strategies:

- **Algorithm 1 Unbiased Sampling**: We random uniformly sample the time-point in the integration time span. Intuitively, since we will perform the training for "a large number of steps," the learned dynamical system would end up being faster to solve "everywhere" over the time span.
- **Algorithm 2 Biased Sampling**: Adaptive Time-Stepping Differential Equation Solvers naturally take more steps around the area, which is harder to integrate. We can bias the regularization to operate around parts of the dynamical system which are "harder" by sampling a time-point from the solution time points.

3.1. Unbiased Sampling

When training a Neural ODE, the integration timespan is fixed. Training any deep learning model involves several thousand steps. We compute the total local error estimate over the entire timespan when performing global regularization. For unbiased sampling, we hypothesize that if we regularize at random uniformly sampled time points in the timespan, the learned dynamical system will demonstrate similar properties in terms of NFE compared to global regularization. Our new regularization term becomes $(\mathcal{R}_E)_{\mathtt{unbiased}}$ as:

$$(\mathcal{R}_E)_{\mathtt{unbiased}} = (\mathrm{E}_{\mathtt{Est}})_{\mathtt{t_{reg}}} \cdot |dt_{\mathtt{t_{reg}}}| \quad \mathtt{t_{reg}} \sim \mathbb{U}[\mathtt{tspan}]$$

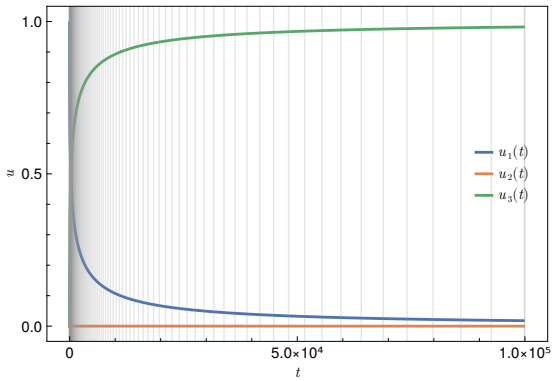

Figure 2: **Robertson Stiff ODE System**: Solving stiff systems like Robertson (Robertson, 1966) (using Rodas5 (Piché, 1995)) involves spending around 75% of the time in $t < 5000$ (i.e. 5% of the timespan). The vertical lines denote the timepoints at which the ODE System was solved.

3.2. Biased Sampling

Consider a simple scenario where the learned dynamics of the DE is harder to solve in $[0.25, 0.35]$, and we are solving the DE from $t_0 = 0$ to $t_1 = 1$. Our primary aim is to modify the learned system s.t. it becomes simpler to solve in $[0.25, 0.35]$. If we use unbiased sampling, the probability that we regularize at $\mathtt{t_{reg}} \in [0.25, 0.35]$ is 0.1 (which is low). The problem gets even more severe if the range is lowered. An extreme version of this problem is observed for stiff systems like Robertson's Equations (See Figure 2) where 75% of the time is spent in solving 5% of the problem. We note that these extreme scenarios rarely occur for traditional deep learning tasks since Pal et al. (2021) observed

minor speedups using stiffness regularization. However, the problem that some parts of the dynamical system are harder to integrate persists, and designing a regularization scheme targeting those parts is highly desirable.

We considered a simple scenario where the learned dynamical system was fixed. However, while training NDEs, this system evolves with training, and apriori predicting the more difficult portions to integrate is not feasible. Adaptive solvers take more frequent steps in the parts of the DE where it is harder to integrate. Anantharaman et al. (2020) leveraged this property of adaptive solvers to learn surrogates for stiff systems. Since these solvers adapt to concentrate around the most numerically difficult time points, we automatically obtain the time points where we want to regularize the model. Hence, for our biased sampling regularize, we uniformly sample the regularization timepoint $\texttt{t}_{\texttt{reg}}$ from the time points at which the solver solved the differential equation.

4. Experiments

In this section, we compare the effectiveness of unbiased and biased local regularization's effectiveness on the training and prediction timings of NDEs. We choose image classification and time series prediction problems in line with prior works on accelerating NDEs. We consider the following baselines:

1. Vanilla Neural ODE with Continuous Interpolating Adjoint.
2. Vanilla Neural SDE with discrete sensitivities.
3. Global Regularization of Neural Differential Equations using discrete sensitivity analysis (Pal et al., 2021).
4. TayNODE (Kelly et al., 2020) and STEER (Behl et al., 2020) for models reported in Pal et al. (2021).

We use the DifferentialEquations.jl (Rackauckas et al., 2019) and Lux.jl (Pal, 2022) software stack written in the Julia Programming Language (Bezanson et al., 2017) for all our experiments.

Some details about the data presented in the tables:

- All experimental results in the tables marked with ¶ were taken directly from Pal et al. (2021).
- We have tried to match the hardware details presented in the paper and the corresponding GitHub repository for Pal et al. (2021), but we note that differences in wall clock timings can be partially attributed to hardware.
- TayNODE (Kelly et al., 2020) uses a different ODE integrator. Hence the NFEs are not directly comparable.

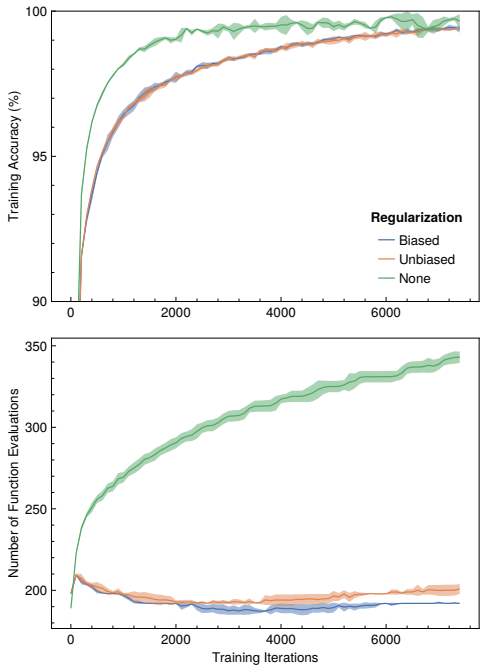

Figure 3: **MNIST Classification using Neural ODE**

4.1. MNIST Image Classification

We train a neural differential equation classifier to map flattened MNIST (LeCun et al., 1998) images to their corresponding labels.

4.1.1. Neural Ordinary Differential Equation

Training Details: We use the same model architecture as described in Kelly et al. (2020). Our model comprises of single hidden layered explicit model f_θ modeling the ODE dynamics followed by a linear classifier g_ϕ. The hidden layer is 100-dimensional. We train with a batch size of 512 for a total of 7500 steps. We use Adam (Kingma & Ba, 2014) with a constant learning rate of 0.001. For error estimate regularization, we exponentially decrease the regularization coefficient from 2.5 to 1.0. We use Tsit5 (Tsitouras, 2011) as the ODE integrator with an absolute and relative tolerance of 10^{-8}.[2]

Baselines: We consider a Vanilla Neural ODE trained with the exact aforementioned specifications. All other baselines are directly taken from Pal et al. (2021).

Results: We summarize the results in Table 2 and Figure 3. Using local regularization speeds up prediction in all cases while it leads to a minor slowdown during training for biased sampling.

[2]We note that this is not a realistic tolerance at which image classification models are trained. We use this tolerance to allow a direct comparison to prior works.

Method	Train Accuracy (%)	Test Accuracy (%)	Training Time (hr)	Prediction Time (s / batch)	Testing NFE
Vanilla NODE	99.898 ± 0.066	97.612 ± 0.163	0.54 ± 0.001	0.088 ± 0.020	303.559 ± 3.194
STEER ¶	100.00 ± 0.000	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	80.3 ± 0.43
ERNODE ¶	99.71 ± 0.28	97.32 ± 0.06	0.82 ± 0.02	0.060 ± 0.001	177.0 ± 0.00
SRNODE ¶	100.00 ± 0.000	98.08 ± 0.15	1.24 ± 0.06	0.094 ± 0.003	259.0 ± 3.46
Local Unbiased ERNODE	99.447 ± 0.039	97.526 ± 0.131	0.49 ± 0.002	0.046 ± 0.002	187.961 ± 1.812
Local Biased ERNODE	99.477 ± 0.051	97.488 ± 0.016	1.12 ± 0.065	0.044 ± 0.002	182.849 ± 1.578
Vanilla NSDE	98.27 ± 0.11	96.66 ± 0.16	2.70 ± 0.00	0.51 ± 0.07	313.86 ± 2.94
ERNSDE ¶	98.16 ± 0.11	96.27 ± 0.35	4.19 ± 0.04	7.23 ± 0.14	184.67 ± 2.31
SRNSDE ¶	98.79 ± 0.12	96.80 ± 0.07	8.54 ± 0.37	14.50 ± 0.40	382.00 ± 4.00
Local Unbiased ERNSDE	98.05 ± 0.09	96.57 ± 0.13	2.10 ± 0.01	0.39 ± 0.10	228.93 ± 1.77
Local Biased ERNSDE	98.02 ± 0.07	96.44 ± 0.16	1.90 ± 0.00	0.36 ± 0.03	230.10 ± 0.71

Table 2: **MNIST Image Classification using Neural DE**: Using local unbiased regularization on neural ODE speeds up training by $1.1\times$ and predictions by $1.9\times$ while reducing the total NFEs to $0.619\times$. Local Biased Regularization tends to slow down training for smaller models on GPU while it further reduces the NFEs by $0.602\times$. For Neural SDE, we observe a similar reduction of NFEs by $0.729 \times -0.733\times$ and a training time improvement of $1.28 \times -1.42\times$. The best global regularization method gets lower NFEs but overall takes more wall clock than the best performing local regularization method.

Method	Test Loss ($\times 10^{-3}$)	Training Time (hr)	Prediction Time (s / batch)	Testing NFE
Vanilla NODE	3.41 ± 0.10	2.48 ± 0.22	0.16 ± 0.01	758.0 ± 25.87
STEER ¶	3.48 ± 0.01	1.62 ± 0.26	0.54 ± 0.06	699.0 ± 141.1
TayNODE ¶	4.21 ± 0.01	12.3 ± 0.32	0.22 ± 0.02	167.3 ± 11.93
ERNODE ¶	3.57 ± 0.00	0.94 ± 0.13	0.21 ± 0.02	287.0 ± 17.32
SRNODE ¶	3.58 ± 0.05	0.87 ± 0.09	0.20 ± 0.01	273.0 ± 0.000
Local Unbiased ERNODE	3.64 ± 0.07	2.31 ± 0.02	0.09 ± 0.00	422.0 ± 4.580
Local Biased ERNODE	3.63 ± 0.08	2.12 ± 0.24	0.10 ± 0.01	463.0 ± 63.02

Table 3: **Physionet Time Series Interpolation**: Local Regularization reduces NFEs by $0.556 \times -0.610\times$ reducing the prediction timings by $1.6 \times -1.78\times$. Our methods additionally improve training timings by $1.073 \times -1.167\times$. We note that the difference in training time compared to (E/S)RNODE methods is due to change in the sensitivity algorithm.

4.1.2. Neural Stochastic Differential Equation

Training Details: We downsample the flattened images to a 32-dimensional vector before feeding it into the Neural SDE which uses a diffusion model (f_θ) having a 64-dimensional hidden layer and a linear drift model (g_ϕ). Finally, a linear classifier (h_γ) predicts the label. We train our models on CPU with a batch size of 512 for a total of 4000 steps. We optimize the weights using Adam (Kingma & Ba, 2014) with a constant learning rate of 0.01. We use SOSRI2 SDE solver (Rackauckas & Nie, 2017) with a tolerance of 0.14. We fix our regularization coefficient to be 10^3. For this experiment, we rely on discrete sensitivity analysis.

Baselines: ERNSDE and SRNSDE results were taken from Pal et al. (2021). These were trained for 40 epochs, nearly equivalent to training for 4000 iterations.

Results: We summarize the results in Table 2 and Figure 4. Local regularization improves training and prediction performance while keeping the test accuracy nearly constant.

4.2. Physionet Time Series Interpolation

Training Details: We use the experimental setup for Physionet 2012 Challenge Dataset (Citi & Barbieri, 2012) from Kelly et al. (2020). We use a Latent Neural ODE (Rubanova et al., 2019) to perform time series interpolation on the dataset. We use the preprocessed dataset from Kelly et al. (2020) to ensure a fair comparison and independent runs are performed using an 80:20 split of the dataset.

For specific model architecture details, we refer the readers to Pal et al. (2021). We train the model for a total of 3000 iterations using Adamax (Kingma & Ba, 2014) with a learning rate of 0.01 with 10^{-5} inverse decay per step. We use a batch size of 512. We diverge from Pal et al. (2021), in using

Configuration	Method	Train Accuracy (%)	Test Accuracy (%)	Training Time (s / batch)	Prediction Time (s / batch)	Testing NFE
Standard	Vanilla	83.683 ± 1.450	67.394 ± 0.849	0.457 ± 0.018	0.130 ± 0.013	115.315 ± 12.136
	Local Unbiased ER	83.665 ± 0.805	67.678 ± 0.874	0.399 ± 0.014	0.096 ± 0.007	89.048 ± 7.335
	Local Biased ER	83.958 ± 1.032	67.745 ± 0.824	0.555 ± 0.008	0.088 ± 0.003	81.301 ± 1.255
Multi-Scale	Vanilla	92.807 ± 12.458	80.048 ± 6.740	0.572 ± 0.012	0.170 ± 0.005	27.616 ± 0.905
	Local Unbiased ER	94.159 ± 9.694	80.432 ± 5.548	0.641 ± 0.025	0.175 ± 0.019	27.760 ± 0.177
	Local Biased ER	99.987 ± 0.023	83.460 ± 0.727	0.774 ± 0.293	0.163 ± 0.015	26.334 ± 0.992

Table 4: **CIFAR10 Image Classification using Neural DE**: For the standard Neural ODE, local regularization reduces the NFE by $0.705 \times -0.772\times$, thereby improving prediction timings by $1.35 \times -1.477\times$. However, unregularized model training takes $0.823\times$ the time for the biased model. For multi-scale models, the NFE and prediction time improvements are marginal and come at the cost of higher training time.

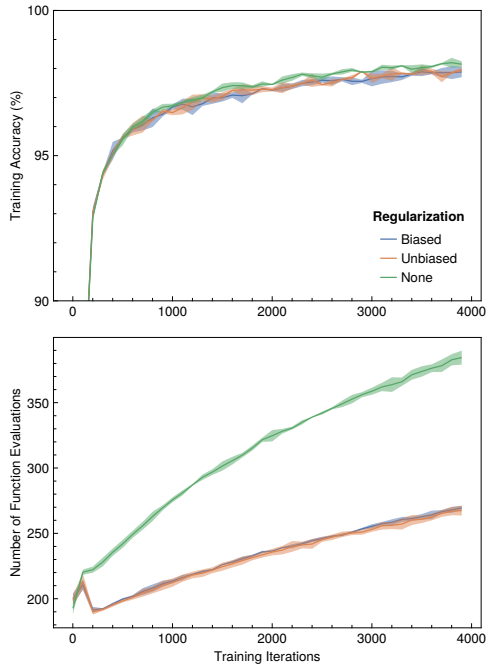

Figure 4: **MNIST Classification using Neural SDE**

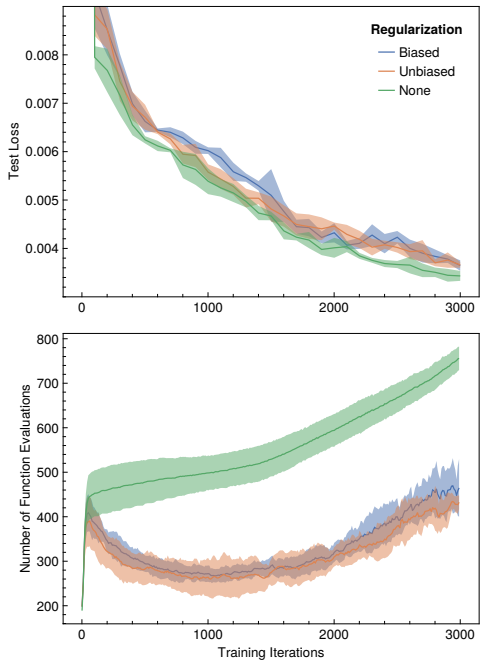

Figure 5: **Physionet Time Series Interpolation using Latent ODE**

the regularization term as $(\mathtt{E_{Est}})_{\mathtt{t_{reg}}} \cdot |dt|_{\mathtt{t_{reg}}}$ instead of the squared regularization term $\sum_j (\mathtt{E_{Est}})_j^2$. Additionally, we decay the regularization coefficient exponentially from 100 to 10 over the 3000 training iterations.

Baselines: Vanilla NODE was trained with the exact aforementioned configuration. All the other baselines were trained using discrete sensitivity analysis, and the exact details are present in Pal et al. (2021).

Results: We summarize the results in Figure 5 and Table 3.

4.3. CIFAR10 Image Classification

4.3.1. Neural Ordinary Differential Equation

Training Details: We use the CNN architecture for CIFAR10 as described in Poli et al. (2020). We train the models for 31250 steps with Adam (Kingma & Ba, 2014) using a cosine-annealing learning rate scheduler from 0.003 to 0.0001. We train the models with a batch size of 32 and keep the regularization coefficient fixed at 2.5. We use Tsit5 (Tsitouras, 2011) with a tolerance of 10^{-4}.

Results: We summarize the results in Figure 6 and Table 4.

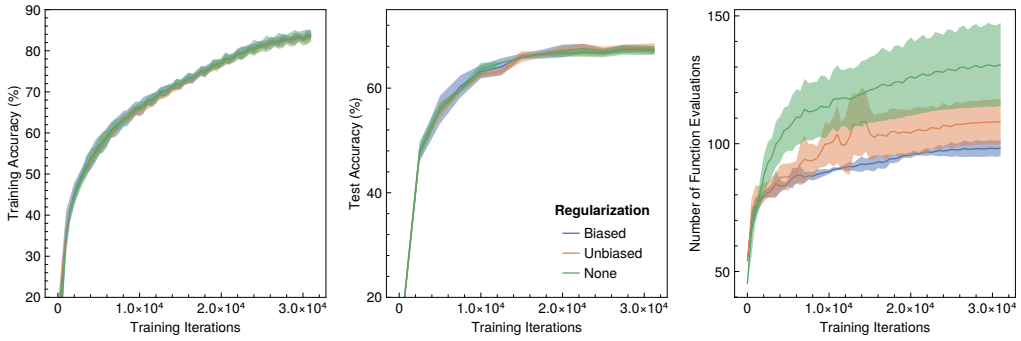

Figure 6: **CIFAR10 Image Classification using Standard Neural ODE**

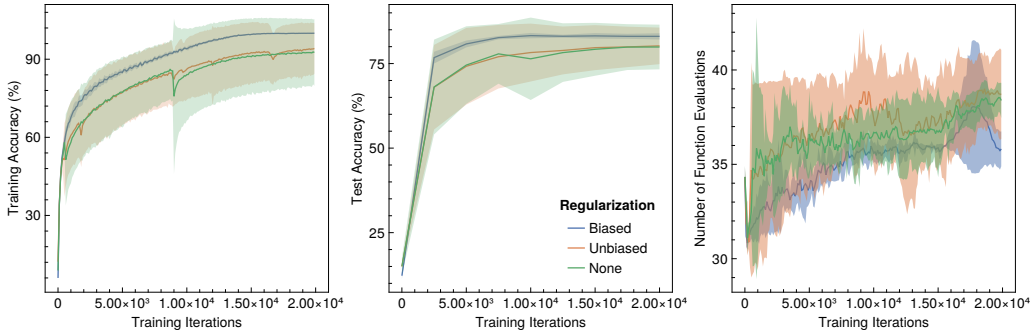

Figure 7: **CIFAR10 Image Classification using Multi-Scale Neural ODE**

4.3.2. Multiscale Neural ODE

Training Details: We modify the Tiny Multiscale DEQ architecture for CIFAR10 from Bai et al. (2020) as Multiscale Neural ODE with Input Injection (See Section 2.5). To stabilize the training for larger models, we exponentially increase the regularization coefficient from 0.1 to 5.0. We train with a batch size of 128 using VCAB3 (Wanner & Hairer, 1996) with a tolerance of 0.05.

Results: We summarize the results in Figure 7 and Table 4. The benefits from regularization for NFEs and prediction timings seem marginal. However, regularization using biased sampling makes the training dynamics stable as observed in Figure 7.

5. Related Works

Neural Differential Equations (and by extension most Implicit Neural Networks) are mostly constrained by expensive training and prediction timings (Dupont et al., 2019; Kelly et al., 2020; Finlay et al., 2020; Behl et al., 2020; Pal et al., 2021). Traditionally accelerating these models have relied on using higher-order regularization terms to constrain the space of learnable dynamics (Finlay et al., 2020; Kelly et al., 2020). These models speed up predictions, but their benefits are often overshadowed by a massive training slowdown (Pal et al., 2021).

More recently, quite a few first-order schemes have been proposed. Behl et al. (2020) randomized the endpoint of Neural ODEs to incentivize simpler dynamics. However, Pal et al. (2021) didn't find significant benefits of using STEER in their experiments. Pal et al. (2021) used internal solver heuristics – local error and stiffness estimates – to control the learned dynamics in a way that decreased both prediction and training time. We discuss their work in detail in Section 2.4. Xia et al. (2021) rewrite Neural ODEs as heavy ball ODEs to accelerate both forward and backward passes. Djeumou et al. (2022) replace ODE solvers in the forward with a Taylor-Lagrange expansion and report significantly better training and prediction times.

6. Discussion

In this manuscript, we have shown that we can obtain similar properties to global regularization by regularizing dynamical systems at randomly sampled time points. Additionally, this comes with the benefit of not being forced into a specific sensitivity analysis method. We have taken every experiment in Pal et al. (2021) and empirically showed that our local regularization works at par with global regularization.

However, our experiments using stiffness estimate for local regularization did not yield positive results and were not presented in this manuscript. Thus, we have demonstrated that we can "close the blackbox" and still leverage all the benefits of internal solver heuristics to improve training and predictions of neural differential equations.

6.1. Limitations

We note the following limitations of our work:

- Similar to Pal et al. (2021), if the objective is to learn the actual dynamical system, our method will not yield proper results. Our method is applicable only when the final state is relevant, i.e., in most classical deep learning tasks.
- Regularization introduces a new regularization coefficient hyperparameter which, if not tuned correctly, can lead to unstable dynamics or might negate the scheme's usefulness.

7. Acknowledgement

The authors acknowledge the MIT SuperCloud and Lincoln Laboratory Supercomputing Center for providing HPC resources that have contributed to the research results reported within this paper. This material is based upon work supported by the National Science Foundation under grant no. OAC-1835443, grant no. SII-2029670, grant no. ECCS-2029670, grant no. OAC-2103804, and grant no. PHY-2021825. We also gratefully acknowledge the U.S. Agency for International Development through Penn State for grant no. S002283-USAID. The information, data, or work presented herein was funded in part by the Advanced Research Projects Agency-Energy (ARPA-E), U.S. Department of Energy, under Award Number DE-AR0001211 and DE-AR0001222. We also gratefully acknowledge the U.S. Agency for International Development through Penn State for grant no. S002283-USAID. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof. This material was supported by The Research Council of Norway and Equinor ASA through Research Council project "308817 - Digital wells for optimal production and drainage". Research was sponsored by the United States Air Force Research Laboratory and the United States Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

References

Anantharaman, R., Ma, Y., Gowda, S., Laughman, C., Shah, V., Edelman, A., and Rackauckas, C. Accelerating simulation of stiff nonlinear systems using continuous-time echo state networks. *arXiv preprint arXiv:2010.04004*, 2020.

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.

Bai, S., Koltun, V., and Kolter, J. Z. Multiscale Deep Equilibrium Models. *arXiv:2006.08656 [cs, stat]*, November 2020. URL `http://arxiv.org/abs/2006.08656`. arXiv: 2006.08656.

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.

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.

Burt, P. J. and Adelson, E. H. The laplacian pyramid as a compact image code. In *Readings in computer vision*, pp. 671–679. Elsevier, 1987.

Chen, L.-C., Yang, Y., Wang, J., Xu, W., and Yuille, A. L. Attention to scale: Scale-aware semantic image segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3640–3649, 2016.

Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

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.

Citi, L. and Barbieri, R. Physionet 2012 challenge: Predicting mortality of icu patients using a cascaded svm-glm paradigm. In *2012 Computing in Cardiology*, pp. 257–260. IEEE, 2012.

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.

Djeumou, F., Neary, C., Goubault, E., Putot, S., and Topcu, U. Taylor-lagrange neural ordinary differential equations: Toward fast training and evaluation of neural odes, 2022.

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. *Advances in Neural Information Processing Systems*, 32, 2019.

Farabet, C., Couprie, C., Najman, L., and LeCun, Y. Learning hierarchical features for scene labeling. *IEEE transactions on pattern analysis and machine intelligence*, 35 (8):1915–1929, 2012.

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.

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.

Hindmarsh, A. C., Brown, P. N., Grant, K. E., Lee, S. L., Serban, R., Shumaker, D. E., and Woodward, C. S. Sundials: Suite of nonlinear and differential/algebraic equation solvers. *ACM Trans. Math. Softw.*, 31(3): 363–396, sep 2005. ISSN 0098-3500. doi: 10.1145/1089014.1089020. URL https://doi.org/10.1145/1089014.1089020.

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

Kim, S., Ji, W., Deng, S., Ma, Y., and Rackauckas, C. Stiff neural ordinary differential equations. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 31(9):093122, 2021.

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.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

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.

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

Pal, A. Lux: Explicit parameterization of deep neural networks in julia. https://github.com/avik-pal/Lux.jl/, 2022.

Pal, A., Ma, Y., Shah, V., and Rackauckas, C. V. Opening the blackbox: Accelerating neural differential equations by regularizing internal solver heuristics. In *International Conference on Machine Learning*, pp. 8325–8335. PMLR, 2021.

Pal, A., Edelman, A., and Rackauckas, C. Mixing implicit and explicit deep learning with skip deqs and infinite time neural odes (continuous deqs). *arXiv preprint arXiv:2201.12240*, 2022.

Piché, R. An l-stable rosenbrock method for step-by-step time integration in structural dynamics. *Computer Methods in Applied Mechanics and Engineering*, 126(3-4): 343–354, 1995.

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

Rackauckas, C. Engineering trade-offs in automatic differentiation: From tensorflow and pytorch to jax and julia, Jan 2022. URL https://tinyurl.com/2j94bhc9.

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. 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., 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.

Robertson, H. The solution of a set of reaction rate equations. *Numerical analysis: an introduction*, 178182, 1966.

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.

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.

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

Xia, H., Suliafu, V., Ji, H., Nguyen, T., Bertozzi, A., Osher, S., and Wang, B. Heavy ball neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 34, 2021.

Yu, F. and Koltun, V. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

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.