Stiff Neural Ordinary Differential Equations

Suyong Kim [1] **Weiqi Ji** [1] **Sili Deng** [1] **Christopher Rackauckas** [1 2 3]

Abstract

Neural Ordinary Differential Equations (ODE) are a promising approach to learn dynamic models from time-series data in science and engineering applications. This work aims at learning Neural ODE for stiff systems, which are usually raised from chemical kinetic modeling in chemical and biological systems. We first show the challenges of learning neural ODE in the classical stiff ODE systems of Robertson's problem and propose techniques to mitigate the challenges associated with scale separations in stiff systems. We then present successful demonstrations in stiff systems of Robertson's problem and an air pollution problem. The demonstrations show that the usage of deep networks with rectified activations, proper scaling of the network outputs as well as loss functions, and stabilized gradient calculations are the key techniques enabling the learning of stiff neural ODE. The success of learning stiff neural ODE opens up possibilities of using neural ODEs in applications with widely varying time-scales, like chemical dynamics in energy conversion, environmental engineering, and the life sciences.

1. Introduction

Neural Ordinary Differential Equations (ODEs) are elegant approaches for data-driven modeling dynamic systems from time-series data in science and engineering applications. It offers various advantages over physics-based modeling and traditional neural network modeling. Compared to physics-based modeling where a presumed functional form is required, the neural ODE approach precludes the need for expert knowledge in identifying such prior information (Rackauckas et al., 2020; Ji & Deng, 2020). Compared to traditional neural network modeling, such as recurrent neural networks, neural ODEs enables flexibility in the modeling of irregularly and incomplete sampled time series (Rubanova et al., 2019), and can be more efficient by taking the advantages of modern ODE solvers and adjoint methods (Chen et al., 2018; Rackauckas et al., 2019).

Despite the success of neural ODEs (Chen et al., 2018) in many sciences and engineering problems (Bills et al., 2020; Portwood et al., 2019; Maulik et al., 2020; Owoyele & Pal, 2020), it has been recognized that it is very challenging to learn neural ODEs for stiff dynamics (Ghosh et al., 2020), characterized by dynamics with widely separated time scales. Part of the challenge is due to the high computational cost of solving stiff ODEs as well as handling the ill-conditioned gradients of loss functions with respect to neural network weights (Anantharaman et al., 2020). In addition, it has been shown that stiffness could lead to failures in many data-driven modeling approaches, such as reduced order of modeling (Huang et al., 2020) and physics-informed neural networks (Ji et al., 2020). It was suggested that the stiffness could lead to gradient pathologies and ill-conditioned optimization problems, which leads to the failure of stochastic gradient descent based optimization (Wang et al., 2020).

This work is motivated to elucidate the challenges of learning neural ODEs for stiff systems and proposes strategies to overcome these obstacles. We study learning neural ODEs on data generated from two classical stiff systems, ROBER (Robertson, 1966) and POLLU (Verwer, 1994), which are extensively used for benchmarking stiff ODE integrators. Both ROBER and POLLU describe the dynamics of species concentrations in stiff chemical reaction systems. We propose strategies to mitigate the challenges associated with the scale separations in time as well as the magnitudes of states. We are thus able to successfully learn stiff neural ODEs for those two benchmark problems. We demonstrate how the application of adjoint methods to stiff systems leads to numerical issues and changes in the computational complexity that are overcome by proposing new adjoint techniques. Those strategies not only demonstrate successful numerical methods and architectures for handling time scale separations in neural ODEs but also elucidate general theoretical issues of data-driven modeling of stiff dynamic systems which can be transferred to other techniques.

[1]Massachusetts Institute of Technology [2]Pumas AI [3]University of Maryland Baltimore. Correspondence to: Suyong Kim <suyong@mit.edu>, Weiqi Ji <weiqiji@mit.edu>, Sili Deng <silideng@mit.edu>, Christopher Rackauckas <crackauc@mit.edu>.

Preprint ,

2. Background

2.1. Neural Ordinary Differential Equations

Many science and engineering problems can be formulated as ODEs, that is,

$$\frac{dy\left(t\right)}{dt} = f\left(y\left(t\right), \theta, t\right) \tag{1}$$

where t is time, $y(t)$ is the state variables, and the function f models the dynamics. Identifying the model f is one of the central tasks in many scientific discoveries, which usually involves proposing functional forms and then performing parameter inference against observation data. Proposing functional forms of f is a very challenging task for many complex systems, such as modeling the gene regulatory networks and cell signaling networks in life science (Hoffmann et al., 2019; Yuan et al., 2020). Discovering the functional forms of f could require decades of effort with expert knowledge and it is often the case that a lot of unknown dynamics are yet to discover in those complex systems (Brunton et al., 2016; Mangan et al., 2016). With the help of the universal approximation theorem, one can approximate the model f using a neural network without worrying about missing important dynamics as in physics-based modeling, i.e.,

$$\frac{dy\left(t\right)}{dt} = NN_{\theta}\left(y\left(t\right), t\right) \tag{2}$$

If one can measure the data pair of $\left(y\left(t\right), \frac{dy(t)}{dt}\right)$, one can train the neural network straightforwardly as a normal supervised learning problem. However, we usually only have access to the time-series data of $y\left(t\right)$. Instead, Neural ODEs train the NN by integrating the ODEs,

$$\begin{aligned} y\left(t_1\right) &= y\left(t_0\right) + \int_{t_0}^{t_1} NN\left(y\left(t\right), t\right) dt \\ &= ODESolve\left(y\left(t_0\right), NN, t_0, t_1, \theta\right) \end{aligned} \tag{3}$$

in which $y(t_0)$ are the initial conditions of the integration, $t \in (t_0, t_1)$ is the range of the integration. We can define the loss functions as the difference between the observed state variables and the predicted state variables,

$$L(\theta) = MAE\left(y\left(t\right)^{model}, y\left(t\right)^{obs}\right) \tag{4}$$

where the mean absolute error (MAE) as an example metric. Using modern (anonymized) differentiable ODE solvers , one can compute the gradient of the loss function to the model parameters efficiently. This work employs (anonymized) to integrate the neural ODEs and do the backpropagation. We can thus use stochastic gradient descent algorithms to optimize the neural network.

2.2. Stiff ODE Systems

Ernst Hairer's classic working definition for the field of numerical ODE solving is "stiff equations are problems for which explicit methods don't work" (Wanner & Hairer, 1996; Shampine & Gear, 1979). In other words, it is the case where numerical ODE solvers like the `dopri` method, chosen in the original neural ODE paper (Chen et al., 2018), fail to adequately solve the differential equation. Many definitions have been proposed to explain this phenomena. A commonly used definition is the stiffness index:

$$S = \frac{Re(\lambda_{max})}{Re(\lambda_{min})}(t_1 - t_0) \tag{5}$$

where λ_i are the eigenvalues of the Jacobian. While these types of stiffness can be helpful in identifying essential analytical properties for numerical methods to capture, every stiffness index does not capture all of the numerically difficult problems. Shampine famously stated "Indeed, the classic Robertson chemical kinetics problem typically used to illustrate stiffness has two zero eigenvalues. Any definition of stiffness that does not apply to this standard test problem is not very useful." (Shampine & Thompson, 2007). In additional, the role of the time span is often overlooked. Shampine notes that a system may not seem stiff if one is only solving on the timescale of the fast process: stiffness requires that one is investigating the effects on the time scale of the slow process.

Due to the ambiguity of the definition of stiff systems, we will choose two differential equations studied extensively by the stiff ODE solver literature as representative highly stiff systems (Wanner & Hairer, 1996): ROBER (Robertson, 1966) and POLLU (Verwer, 1994). Specifically, the canonical ROBER problem consists of three species and five reactions, and the POLLU problem consists of 20 species and 25 reactions describing the air pollution formation in atmospheric chemistry. The formula of the ROBER and POLLU problem is presented in Section 3. We will investigate the difficulty of fitting data sampled from these systems, thus illuminating the unique features of the training process which arise in the discovery of stiff systems and the numerical methods for handling them.

3. Method

3.1. Stabiliy of Adjoints on Stiff ODEs

The core problem of training neural ODEs is calculating the gradient of the ODE's solution. In the method proposed by (Chen et al., 2018), the ODE is reversed and augmented with the adjoint equations as:

$$\frac{d}{dt}\begin{bmatrix} z \\ \omega \\ \frac{dL}{d\theta} \end{bmatrix} = -\begin{bmatrix} 1 & z^T & \omega^T \end{bmatrix}\begin{bmatrix} f & \frac{\partial f}{\partial z} & \frac{\partial f}{\partial \theta} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{6}$$

However, previous work has called into question the stability of reversing the ODE equation, i.e.

$$y' = -f(y, \theta, -t) \tag{7}$$

when solving from t_1 to t_0. While theoretically this reverses the ODE exactly, in practice numerical issues can cause errors which propagate into the gradient. (Gholami et al., 2019) noted that if the Lipschitz constant is too large in absolute value, then either the forward pass or reverse solve is numerically unstable. This can be seen by using a standard ODE solver on the linear ODE $\frac{dy}{dt} = -\lambda y$ on $t \in (0, 1)$ with $y(0) = 1$, $\lambda = 100$ imparts about 1% error while $\lambda = 10000$ cannot be reversed numerically in double precision.

To show how this result extends to stiff ODEs, recall that the Lipschitz constant of a nonlinear function is the infimum of values Λ such that $\left\| \frac{f(x)-f(y)}{x-y} \right\| \leq \Lambda$ for some domain $x, y \in \Omega$, which directly implies that the global Lipschitz constant satisfies $\|\frac{df}{dy}\|_\infty \leq \Lambda$ over the whole domain. Thus because the maximum eigenvalues of the Jacobian give a lower bound on the Lipschitz constant, stiff ODEs fall into the class of problems which are numerically difficult to reverse. Figure 1 demonstrates this on the ROBER stiff ODE test problem and demonstrates that even with imperceptible errors in the forward pass we can receive noticeable errors in the reverse solve even when plotted in log scale. Using the well-known `CVODE` BDF integrator from Sundials (Hindmarsh et al., 2005), we see that with $tol = abstol = reltol = 10^{-6}$ we receive 72% error, $tol = 10^{-7}$ gives 38% error, and $tol = 10^{-8}$ gives 5% error, with 0.1% error only reached at $tol < 10^{-10}$, i.e. below the threshold of single precision arithmetic.

The adjoint calculation only amplifies this error. With a test loss function $L(\theta) = \sum y_1(t_i)$ at evenly spaced points in log-space, using the adjoint as described caused `CVODE` to exit with a Newton divergence near the start of the reverse pass for all $tol = 10^{-i}$ for integers $i = 6, 7, 8, 9, 10, 11, 12, 13, 14$. Manually inspecting the solution process reveals the problem. While the reverse solution only has an approximately 10^{-10} error in y_2 at $t = 99999.98999$ as shown in Figure 2, the large k_2 amplifies this term by 6×10^7 imparting approximately $\mathcal{O}(10^{-3})$ error into $\frac{d}{dt}\frac{dL}{d\theta}$. This is because while in the original ODE the middle term is of the form $k_2 y_2^2$, in the Jacobian the term is $2k_2 y_2$ and without the squaring the small error is not diminished. This sends $\frac{dL}{d\theta}$ negative in a way that linearly explodes.

Avoiding this issue requires one does not attempt to reverse the ODE. To this effect we tested the interpolated checkpointing method of `CVODES` (Hindmarsh et al., 2005) and discrete adjoint sensitivities via automatic differentiation of the solver which both successfully solved the equation and agreed on all derivatives to at least three decimal places

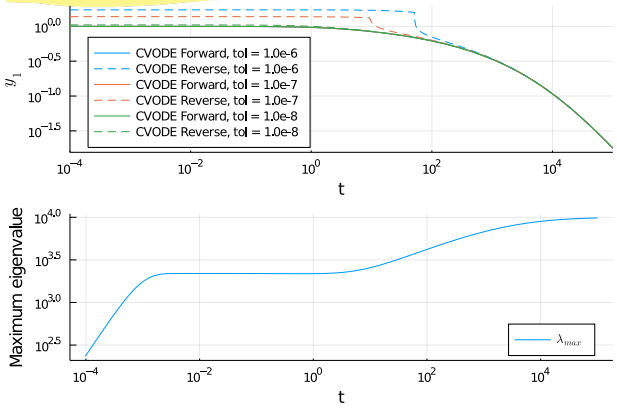


Figure 1. **Reversability of the ROBER Equation**. **Top:** Solution forward and backwards of y_1 in the ROBER test problem with the CVODE integrator at the shown tolerance using the BDF tableau. **Bottom:** Maximum eigenvalue of the Jacobian over the solution trajectory.

when solved with a $tol = 10^{-6}$. We note that one can prove the consistency of the adjoint discretiation in the case of discrete adjoint sensitivities ensuring the stability (Zhang & Sandu, 2014).

3.2. Linear Scaling Adjoints

Being limited to non-reversing adjoints, we investigate the computational cost of derivative calculations on stiff ODEs. To illustrate the computational cost and motivate new formulations, consider the implicit Euler method:

$$y_{n+1} = y_n + hf(y_{n+1}, t + h) \tag{8}$$

which is solved via the root finding problem $G(y_{n+1}) = y_{n+1} - y_n + hf(y_{n+1}, t + h) = 0$ for the next step. The instability of fixed point solvers (Wanner & Hairer, 1996) makes it necessary to solve the root finding problem by Newton's method, i.e. $y_{i+1} = y_i - \frac{dG(y_i)}{dy_i}^{-1} G(y_i)$ so that $y_i \to y_{n+1}$. The core computational cost is thus the solving of the linear system $\frac{dG(y_i)}{dy_i} x = G(y_i)$. The condition number of the matrix $\frac{dG(y_i)}{dy_i}$ is precisely the ratios of the maximum and minimum eigenvalues of the system's Jacobian, meaning the linear system is ill-conditioned for stiff equations. Krylov subspace methods like GMRES have slow convergence when the condition number of the matrix is high (Liesen & Tichý, 2004) and thus using these methods can require problem-specific preconditioners to make convergence computationally acceptable. For this reason, methods for solving stiff ODEs generally attempt direct methods, either dense or sparse, on the Jacobian. The general cost of performing such a linear solve via LU-factorization is $\mathcal{O}(k^3)$ where k is the number of columns in the Jacobian which is equivalent to the number of states in the ODE.

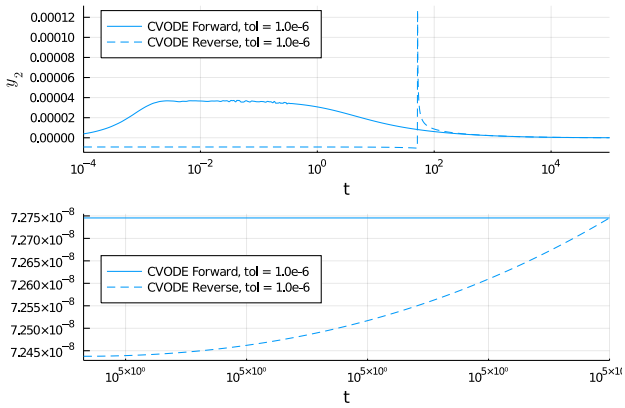


Figure 2. **Reversable instability of the ROBER Equation**. **Top:** Solution forward and backwards of y_2 in the ROBER test problem with the CVODE integrator at the shown tolerance using the BDF tableau. **Bottom:** Zoomed in solution on $t \in (99999.98999, 10^5)$.

The super-linear cost growth of solving a stiff system changes the efficiency calculus of methods for computing the derivative of the ODE solution and thus for calculating the gradient of the neural ODE's loss function. In the traditional argument, forward-mode automatic differentiation of an ODE is equivalent to solving the forward sensitivity equations:

$$z' = f(z, p, t) \tag{9}$$

$$\frac{d}{dt}\frac{dz}{dp} = \frac{\partial f}{\partial z}\frac{dz}{dp} + \frac{\partial f}{\partial p} \tag{10}$$

where there exists one sensitivity $\frac{dz}{dp}$ for each parameter. Thus given k state equations and m parameters, the total size of the ODE system is $\mathcal{O}(km)$. In contrast, the adjoint method works by only solving the original ODE forwards, and solves a system of size $\mathcal{O}(m+k)$ in reverse, greatly reducing the computational cost when the size of the ODE and the parameters are sufficiently large.

However, the total computational cost for implicit methods is based on the cube of the system size, meaning standard forward-mode sensitivities scale as $\mathcal{O}(k^3m^3)$ while the adjoint method of (Chen et al., 2018) or (Serban & Hindmarsh, 2003) scales as $\mathcal{O}((k+m)^3)$. Given the large number of parameters in a neural ODE, the cubic scaling adds a tremendous computational cost. To mitigate this issue, we developed 3 separate strategies. For the first, notice that if one uses the interpolating adjoint but only solves ω in reverse and saves a continuous interpolation $\omega(t)$, then one can solve $\frac{dL}{d\theta} = \int_{t_0}^{t_1} \omega^T(t)\frac{\partial f}{\partial z}dt$ independently. Using quadrature techniques like Clenshaw-Curtis can converge exponentially fast, using less steps than the ODE solver. But most importantly, this operation is $\mathcal{O}(m)$, trading computation for memory and bringing the cost down to $\mathcal{O}(k^3+m)$. We termed this the `QuadratureAdjoint` method.

Alternatively, we note that one could remove the implicit handling of the $\frac{dL}{d\theta}$ term by using an implicit-explicit (IMEX) solver (Kennedy & Carpenter, 2003) on the adjoint equation. The split can be achieved by handling all terms implicitly while enforcing explicit handling of $-\omega^T\frac{\partial f}{\partial \omega}$, in which case the order is returned to $\mathcal{O}(k^3+m)$.

Lastly, one can simply calculate the derivative of $l \ll m$ parameters at a time, performing $\frac{m}{l}$ derivative calculations but bringing the complexity down to $\mathcal{O}(mk^3l^2)$ for forward and $\mathcal{O}(\frac{m}{l}(k+l)^3)$ for reverse. Note that this technique also applies to forward and reverse automatic differentiation of the solver, and the separate differentiations can be solved in parallel. This is the technique that we used to make the experiments of Section 4 computationally viable.

3.3. Equation Scaling

Armed with computationally stable and efficient gradient calculations, we noticed that these techniques were still not enough to stabilize the training process. To further tackle the difficulties in training stiff neural ODEs introduced by scale separation, we propose to normalize the neural network outputs and the loss components for different species. Specifically, we learn the following normalized form of neural ODEs:

$$\begin{aligned} \frac{dy(t)}{dt} &= NN(y(t), t)\frac{y_{scale}}{t_{scale}} \\ y_{scale} &= y_{max} - y_{min} \\ t_{scale} &= t_1 - t_0 \end{aligned} \tag{11}$$

Where y_{scale} is a vector which consists of the characteristic scales for each species and t_{scale} refers to the characteristic time scale. We further re-balance the loss components by normalizing the loss functions as

$$L(\theta) = MAE\left(\frac{y(t)^{model}}{y_{scale}}, \frac{y(t)^{obs}}{y_{scale}}\right) \tag{12}$$

4. Experiments

4.1. ROBER Problem

Robertson's equations, denoted as ROBER, are one of the prominent stiff ODEs which model a reaction network with three chemicals (Robertson, 1966). The species concentrations $[y_1, y_2, y_3]$ are governed by Equation 13 with reaction rate constants of $k_1 = 0.04, k_2 = 3 \cdot 10^7, k_3 = 10^4$.

$$\begin{aligned}\frac{dy_1}{dt} &= -k_1 y_1 + k_3 y_2 y_3 \\ \frac{dy_2}{dt} &= k_1 y_1 - k_2 y_2^2 - k_3 y_2 y_3 \\ \frac{dy_3}{dt} &= k_2 y_2^2\end{aligned} \tag{13}$$

4.1.1. Baseline Model

We first study baseline neural ODEs models to elucidate the challenges of training stiff neural ODEs. The baseline model consists of one hidden layer with 50 hidden nodes and a hyperbolic tangent activation function, which is similar to the demo code in DiffEqFlux.jl and torchdiffeq. The stiff ODE integrator `Rosenbrock23`, an Order 2/3 L-Stable Rosenbrock-W method, is employed, and we hybrid Rosenbrock23 with Tsit5 via auto-switching to accelerate the integration and training. ForwardDiff.jl is adopted for the backpropagation, which we found is more efficient than adjoint methods in the current case due to the small number of states. The training data is generated by integrating Equation 13 with the initial conditions of $[y_1, y_2, y_3] = [1, 0, 0]$, time span of $\left[10^{-5}, 10^{5}\right]$. The 50 sample data points are uniformly spaced in a logarithmic time scale. The neural ODE is trained with the ADAM (Kingma & Ba, 2014) optimizer with a learning rate of 0.005. The training ended at 10,000 epochs where the loss function flatlines, as shown in Figure 3.

Figure 4 shows the predictions of the learned baseline model. The baseline model failed to learn the actual dynamics and consequently failed to reproduce the species profiles. In addition, the gradient norm highly fluctuates and training losses do not decrease which implies training instability (Figure 3). We hypothesize that the failure is attributed to the time scale separations and stiffness induced scale separations in the neural network outputs as well as the imbalance of different loss function components. Specifically, after the initial short induction period, the changes of the y_1 and y_3 are in the order of unity while y_2 in the order of 10^{-5}, which implies that one has to approximate such widely separated scales in a single neural network. In addition, since the magnitude of y_2 is 5 orders of magnitude smaller than y_1 and y_3, such imbalance in the three loss components could lead to gradient pathologies, which was anticipated that stochastic gradient descent is unstable in the presence of such gradient pathologies (Wang et al., 2020).

4.1.2. Mitigation of Gradient Pathologies via Scaling

Figure 5 shows the results of a successful learning of stiff ROBER problems with equation scaling. The neural networks consist of 6 hidden layers and 5 nodes per layer,

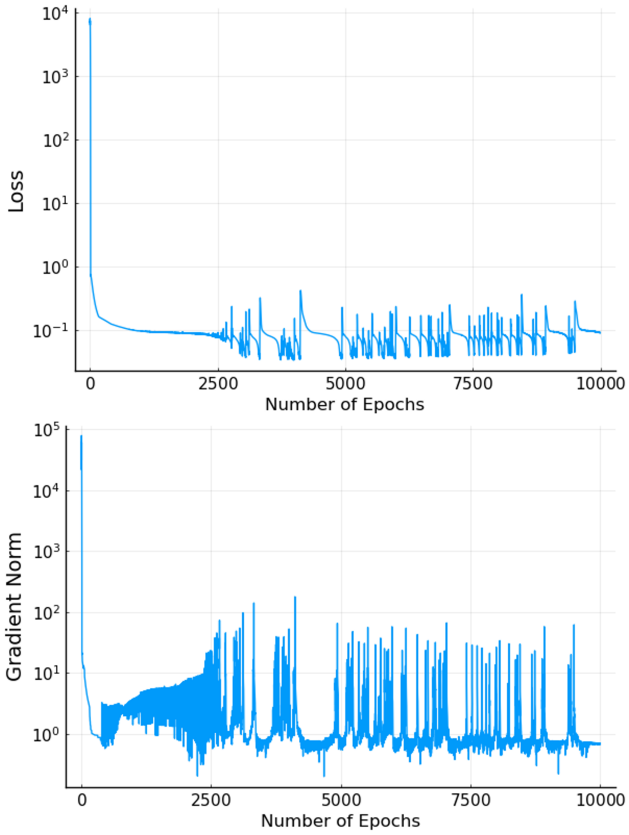


Figure 3. **Training of the baseline model. Top:** Loss profile along epochs **Bottom:** gradient norm along epochs

with an activation function of GELU (Hendrycks & Gimpel, 2016). GELU was chosen to mitigate potential issues with vanishing gradients typically seen in recurrant neural networks with saturating activation functions (Pascanu et al., 2013). Figure 6 shows the history of loss functions and the L_2 norm of the gradient. The loss functions steadily decrease, and the gradient norm is stable during the training. The training in a workstation using a single CPU processor took 2 hours.

We carried out sensitivity analyses to better understand the importance of proposed techniques, i.e., the equation scaling and the deep networks, on the training of stiff ODEs. Our test shows that scaling is essential for training stiff neural ODEs, as the training with the same neural network structure but without equation scaling failed to learn the ROBER problem. Regarding the neural network structure, we found that shallow neural networks with the hyperbolic tangent activation function (tanh) can also successfully learn the ROBER problem but takes a longer training time, which could be attributed to the potential gradient issues. The results of the training without equation scaling and the training with the shallow network are provided in the supplemental material.

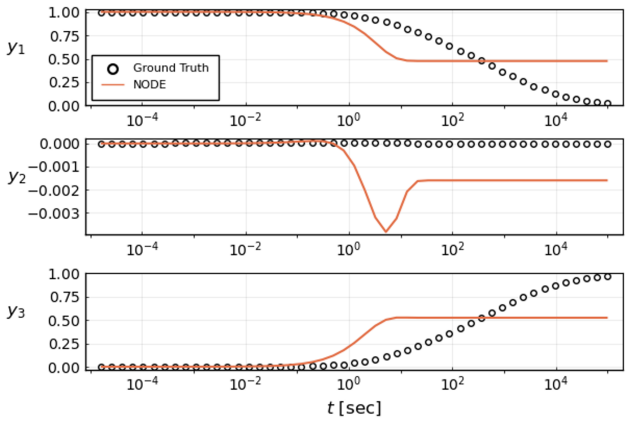


Figure 4. **The ground truth data (symbols) generated using the ROBER model (solid line) and the predictions of the learned baseline model**

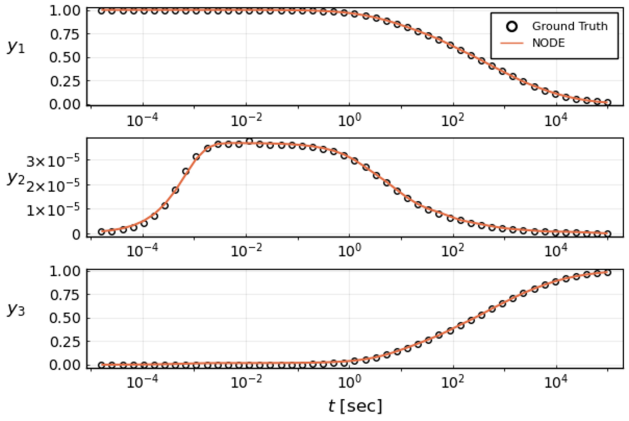
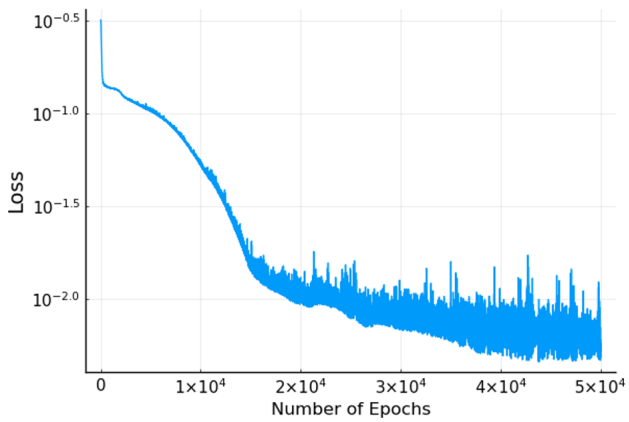


Figure 5. **Ground truth of Robertson equations (circle mark) and results of the scaled neural ordinary differential equation with six hidden layers of 5 nodes and GELU activation function (orange line)**

We can also accelerate the training with temporal regularization (Ghosh et al., 2020) and annealing of learning rates although such techniques were not required for successful learning of stiff neural ODEs. The temporal regularization, STEER (Ghosh et al., 2020), proceeds as randomly truncate the training to an integration time ahead of the entire time-space, and in such a way introduce stochastic into the optimization to accelerate the training. The annealing of a learning rate is a widely adopted technique for accelerating the training of neural networks at later stages.

4.2. POLLU Problem

To show the generality of the technique, we demonstrate the effectiveness of the proposed methods in the POLLU problem, which is more complex than the ROBER problem and consists of 20 species and 25 reactions. The POLLU



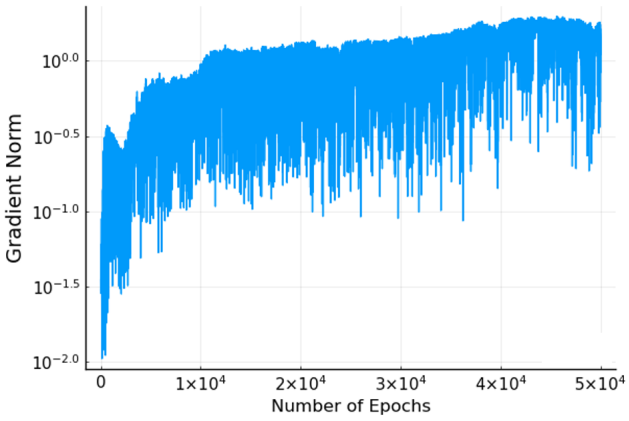


Figure 6. **Training of a scaled neural ODE. Top:** Loss profile along epochs **Bottom:** gradient norm along epochs

is an air pollution model developed at The Dutch National Institute of Public Health and Environmental Protection. It can be described mathematically by the following 20 nonlinear ODEs shown in Equation 14, and full details about the model can be found in (Verwer, 1994) and supplemental materials.

$$\frac{dy\left(t\right)}{dt} = f\left(y\left(t\right)\right)$$
$$y\left(0\right) = y_0, y \in \mathbb{R}^{20}, 0 \leq t \leq 60s \tag{14}$$

The neural ODEs consist of 3 hidden layers and 10 nodes per hidden layer. The label data is sampled uniformly in $[0, 60]$ on a linear scale. The rest of the hyper-parameter settings are the same as the ones for the ROBER problem. Figure 7 presents the label data generated with Equation 14 and the predictions of the learned neural ODEs. Note that the baseline model again failed for the POLLU problem similarly to the ROBER problem, and it is presented in the supplemental material. The trained neural ODEs can capture both the dynamics during the induction period and the near-equilibrium states. It also well captures both the dynamics of the major species, such as y_1, y_2, with relatively

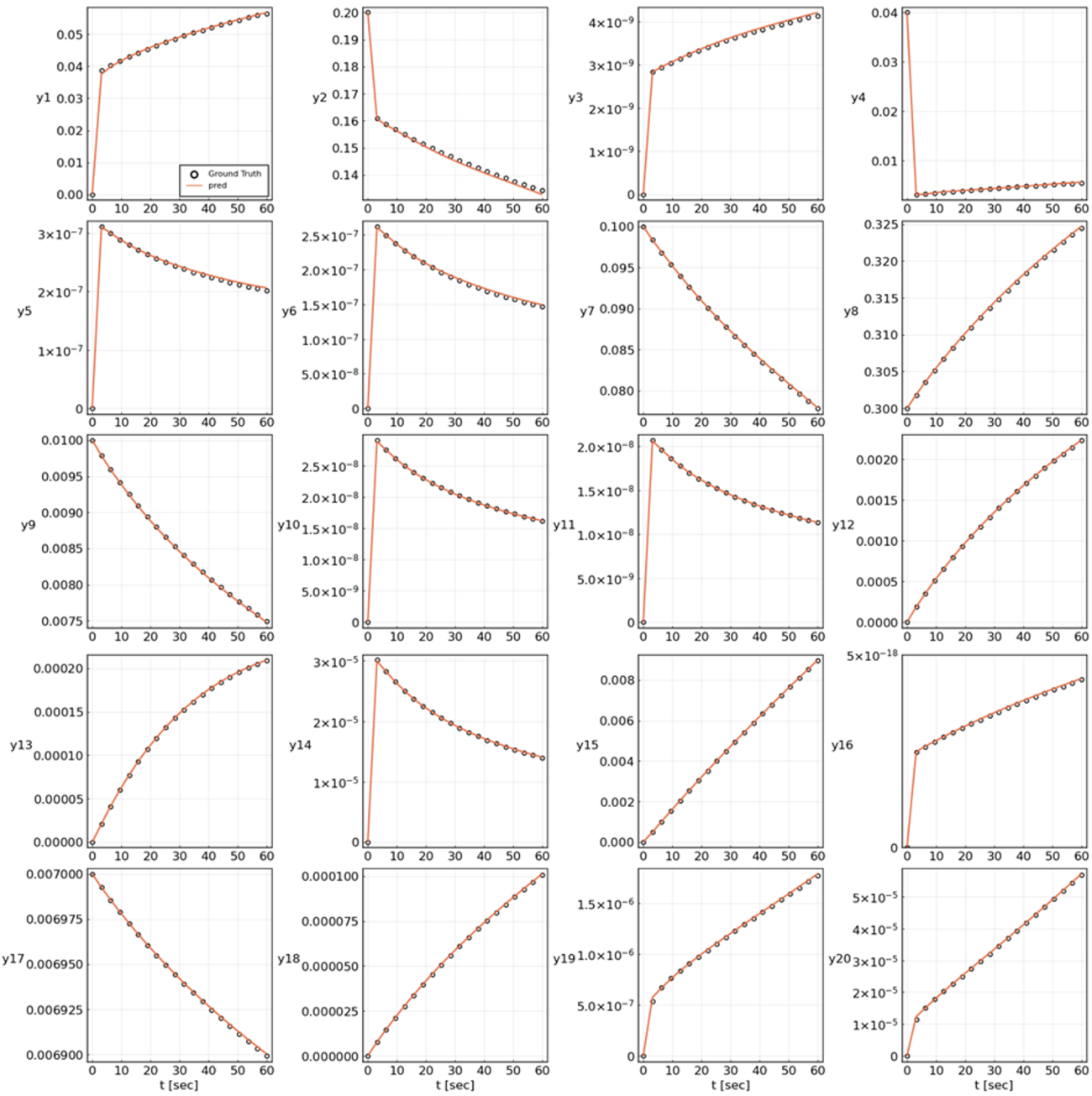


Figure 7. **Ground truth of an air pollution model POLLU (symbols) and results of the scaled neural ordinary differential equation with 3 hidden layers of 10 nodes and GELU activation function (solid line)**

large concentration and slow time scales, and the dynamics of transient intermediate species, such as y_3, y_{10}, with relatively low concentrations and fast time scales. Therefore, the demonstration of the POLLU problem further shows the robustness of proposed approaches in learning stiff neural ODEs where previous techniques had failed.

5. Conclusion and Discussion

This work proposed new derivative calculation techniques for the stability of stiff ODE systems while reducing the computational complexity. It is coupled with equation scaling and deeper neural networks with rectified activation functions to learn neural ODEs in stiff systems. The effectiveness of the proposed approaches is demonstrated in two classical stiff systems: the ROBER and the POLLU. While the baseline models even failed to learn the overall trend of the dynamics, the proposed approaches accurately reproduced all of the species profiles including both species with slow and fast time scales.

While we have shown the effectiveness of the proposed techniques, there is a demand for further theoretical analysis of the challenges of learning stiff neural ODE, such as how stiffness affects the gradient dynamics during training and leads to training failures. In addition, more theoretical analysis on equation scaling relations to the gradient flow of neural ODEs would better elucidate why it was important in the training process. Given the first order optimizers are solving an ordinary differential equation on the gradient flow themselves, one likely could define the stiffness index on the Jacobian of the gradient of the optimization, which is the Hessian with respect to the parameters. Handling the optimization itself with stiffly-aware methods could be beneficial in cases where gradient pathologies are not mitigated.

Importantly, this manuscript demonstrates the impact of errors in the continuous adjoint handling on the training of a neural ODE. Such an analysis could be required in other "implicit layer" methods as well. For example, we referenced how a nonlinear solver destabilizes when its Jacobian exhibits ill-conditioning, equivalent to stiffness in the ODE system when viewing it as a solver to steady state (Wanner & Hairer, 1996). This would suggest that on highly stiff systems, methods like the DEQ (Bai et al., 2019) similar behavior would apply to the forward pass. But importantly, similar instabilities in the adjoint problem likely occur due to the reliance on the adjoint's assumption that $G(x) = 0$ exactly, which is more difficult to satisfy as the condition number rises. Likely similar stabilization may be required for this case. Similarly, differentiable optimization as a neural network layer (Amos & Kolter, 2017; Amos, 2019) requires satisfaction of the KKT equations for its adjoint, which are likely to be less satisfied when the Hessian of the optimization problem is ill-conditioned, therefore likely requires a similar stability analysis.

We end by noting that such developments in handling highly stiff equations can allow for machine learning architectures that satisfy arbitrary constraint equations during their evolution. Rackauckas et al. (2020) proposed using differential-algebraic equations (DAE) in mass-matrix form for this purpose, i.e.

$$Mz' = f(z, p, t) \tag{15}$$

where M is singular. For example, it's well-known that the Rosenbrock equation can be written in its DAE form:

$$\frac{dy_1}{dt} = -k_1 y_1 + k_3 y_2 y_3 \tag{16}$$

$$\frac{dy_2}{dt} = k_1 y_1 - k_2 y_2^2 - k_3 y_2 y_3 \tag{17}$$

$$0 = y_1 + y_2 + y_3 - 1 \tag{18}$$

where the mass matrix is of the $M = [100; 010; 000]$. Similarly, an arbitrary system constrained to have dynamical states sum to 1 could be imposed by:

$$\frac{d[y_1, y_2]}{dt} = NN(y) \tag{19}$$

$$0 = y_1 + y_2 + y_3 - 1 \tag{20}$$

and more generally

$$\frac{dy}{dt} = NN([y, z]) \tag{21}$$

$$0 = g(y, z, t) \tag{22}$$

constraints equation can be set or fit using domain information as necessary. While currently able to be specified and trained in differentiable DAE solver systems, we noticed fitting instabilities similar to stiff neural ODEs arise in such cases. Indeed, Wanner & Hairer (1996) notes that DAEs are a form of infinite stiffness, being posed as the limit of singularly perturbed stiff ODEs. Petzold (1982) famously showcased how "DAEs are not ODEs", showing how they exhibit additional behavior that must be appropriately treated, and thus an analysis of such systems will be required for fully stabilizing neural network DAE formulations including the relationship of training techniques to differential index.

References

Amos, B. *Differentiable optimization-based modeling for machine learning*. PhD thesis, PhD thesis. Carnegie Mellon University, 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.

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.

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

Bills, A., Sripad, S., Fredericks, W. L., Guttenberg, M., Charles, D., Frank, E., and Viswanathan, V. Universal battery performance and degradation model for electric aircraft. *arXiv preprint arXiv:2008.01527*, 2020.

Brunton, S. L., Proctor, J. L., and Kutz, J. N. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences*, 113(15):3932–3937, 2016.

Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. Neural ordinary differential equations. *arXiv preprint arXiv:1806.07366*, 2018.

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

Ghosh, A., Behl, H. S., Dupont, E., Torr, P. H., and Namboodiri, V. Steer: Simple temporal regularization for neural odes. *arXiv preprint arXiv:2006.10711*, 2020.

Hendrycks, D. and Gimpel, K. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

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 Transactions on Mathematical Software (TOMS)*, 31(3):363–396, 2005.

Hoffmann, M., Fröhner, C., and Noé, F. Reactive sindy: Discovering governing reactions from concentration data. *The Journal of chemical physics*, 150(2):025101, 2019.

Huang, C., Wentland, C. R., Duraisamy, K., and Merkle, C. Model reduction for multi-scale transport problems using structure-preserving least-squares projections with variable transformation. *arXiv preprint arXiv:2011.02072*, 2020.

Ji, W. and Deng, S. Autonomous discovery of unknown reaction pathways from data by chemical reaction neural network. *arXiv preprint arXiv:2002.09062*, 2020.

Ji, W., Qiu, W., Shi, Z., Pan, S., and Deng, S. Stiff-pinn: Physics-informed neural network for stiff chemical kinetics. *arXiv preprint arXiv:2011.04520*, 2020.

Kennedy, C. A. and Carpenter, M. H. Additive runge–kutta schemes for convection–diffusion–reaction equations. *Applied numerical mathematics*, 44(1-2):139–181, 2003.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Liesen, J. and Tichỳ, P. Convergence analysis of krylov subspace methods. *GAMM-Mitteilungen*, 27(2):153–173, 2004.

Mangan, N. M., Brunton, S. L., Proctor, J. L., and Kutz, J. N. Inferring biological networks by sparse identification of nonlinear dynamics. *IEEE Transactions on Molecular, Biological and Multi-Scale Communications*, 2(1):52–63, 2016.

Maulik, R., Mohan, A., Lusch, B., Madireddy, S., Balaprakash, P., and Livescu, D. Time-series learning of latent-space dynamics for reduced-order model closure. *Physica D: Nonlinear Phenomena*, 405:132368, 2020.

Owoyele, O. and Pal, P. A neural ordinary differential equations approach for chemical kinetics solvers. *arXiv preprint arXiv:2101.04749*, 2020.

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

Petzold, L. Differential/algebraic equations are not ode's. *SIAM Journal on Scientific and Statistical Computing*, 3 (3):367–384, 1982.

Portwood, G. D., Mitra, P. P., Ribeiro, M. D., Nguyen, T. M., Nadiga, B. T., Saenz, J. A., Chertkov, M., Garg, A., Anandkumar, A., Dengel, A., et al. Turbulence forecasting via neural ode. *arXiv preprint arXiv:1911.05180*, 2019.

Rackauckas, C., Innes, M., Ma, Y., Bettencourt, J., White, L., and Dixit, V. Diffeqflux. jl-a julia library for neural differential equations. *arXiv preprint arXiv:1902.02376*, 2019.

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. *arXiv preprint arXiv:2001.04385*, 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 ordinary differential equations for irregularly-sampled time series. *33rd Conference on Neural Information Processing Systems*, 2019.

Serban, R. and Hindmarsh, A. C. Cvodes: An ode solver with sensitivity analysis capabilities. Technical report, Technical Report UCRL-JP-200039, Lawrence Livermore National Laboratory, 2003.

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.

Verwer, J. G. Gauss–seidel iteration for stiff odes from chemical kinetics. *SIAM Journal on Scientific Computing*, 15(5):1243–1250, 1994.

Wang, S., Teng, Y., and Perdikaris, P. Understanding and mitigating gradient pathologies in physics-informed neural networks. *arXiv preprint arXiv:2001.04536*, 2020.

Wanner, G. and Hairer, E. *Solving ordinary differential equations II*, volume 375. Springer Berlin Heidelberg, 1996.

Yuan, B., Shen, C., Luna, A., Korkut, A., Marks, D. S., Ingraham, J., and Sander, C. Cellbox: Interpretable machine learning for perturbation biology with application to the design of cancer combination therapy. *Cell Systems*, 2020.

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.

Supplementary Material for "Stiff Neural Ordinary Differential Equations"

1. ROBER Problem

This section shows the sensitivity of neural networks and equation scaling to the learnability of the ROBER. Test 1, the successfully learned model in section 4.1.2 of the main manuscript, has 6 hidden layers and 5 nodes per layer with a GELU activation function. The other cases (Test 2-5) are summarized in Table 1.

1.1. Neural Networks

We first present the effect of activation functions on training. A GELU activation function (Test 1) is replaced with ReLU (Test 2) and tanh (Test 3) activation functions, respectively. We trained different neural networks with 50,000 epochs, and the species concentration profiles of y_1, y_2, y_3 are shown in Figure 1. Test 1 and Test 2 learn well while the tanh in Test 3 performs poorly. Test 3 with tanh activation functions fails to capture all the species y_1, y_2, y_3 after 50,000 epochs. Figure 2 shows the loss function along epochs. Test 1 with GELU performs the best in terms of speed and loss, and ReLU follows GELU. Test 3 with tanh shows poor performance, which is potentially due to the gradient vanishing of tanh.

We then present the effect of neural network size. Figure 3 presents a shallow neural network (Test 5) with one hidden layer, 50 nodes, and a GELU activation function. The results show that Test 5 learns slow species y_1 and y_3 well, but it is still under the training of fast species y_2 after 50,000 epochs.

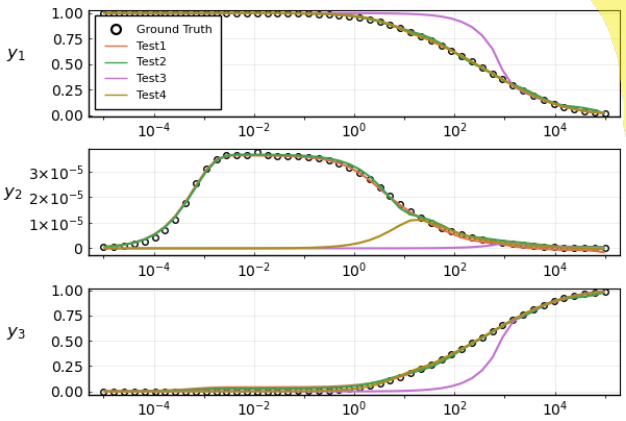


Figure 1. **ROBER species profiles** Neural ODEs with Equation Scaling (Test 1-4) after 50,000 epochs

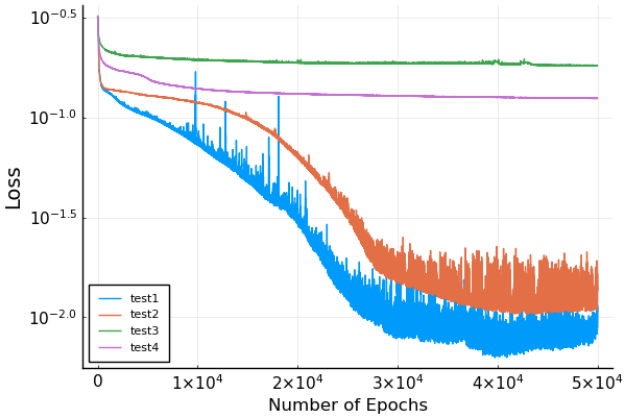


Figure 2. **ROBER loss profile along epochs** Neural ODEs with Equation Scaling (Test 1-4)

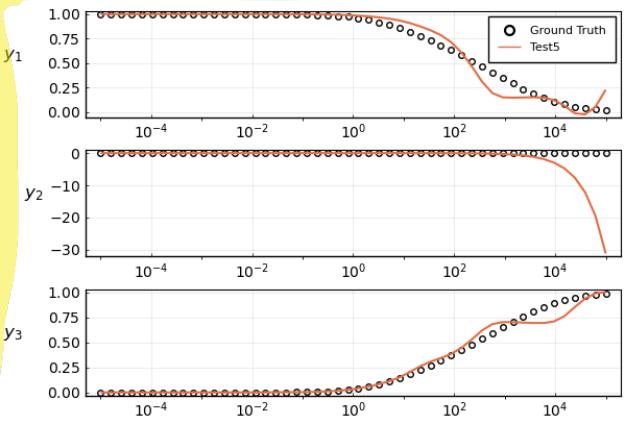


Figure 3. **ROBER species profiles** Neural ODEs without Equation Scaling (Test 5) after 50,000 epochs

1.2. Equation Scaling

Test 5 is the sensitivity test for equation scaling. Compared to Test 1, training neural ODEs without equation scaling could not capture even entire trend of species profiles (Figure 4), and failed to learn ROBER (Figure 3).

2. POLLU Problem

2.1. Full Model Description

POLLU has 20 species and 25 reactions. Given Equation 1, Table 2 and 3 give full equations with initial conditions and reaction rates (**?**).

Table 1. Sensitivity analysis for network structures and equation scaling

Test	# of Hidden Layers	# of Nodes per Layer	Activation Function	Equation Scaling
1	6	5	GELU	Yes
2	6	5	ReLU	Yes
3	6	5	tanh	Yes
4	1	50	GELU	Yes
5	6	5	GELU	No

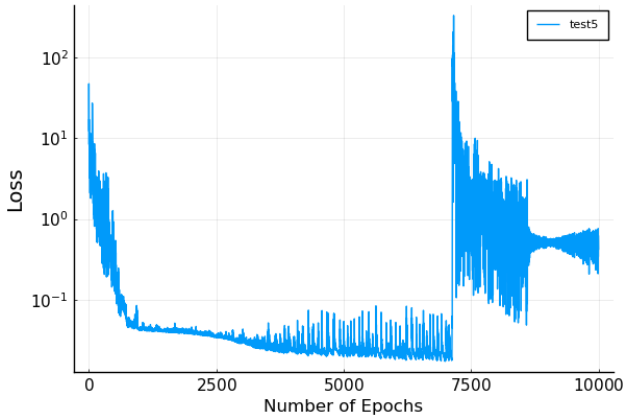


Figure 4. **ROBER loss profile along epochs** Neural ODEs without Equation Scaling (Test 5)

$$\begin{aligned} \frac{dy\left(t\right)}{dt} &= f\left(y\left(t\right)\right) \\ y\left(0\right) = y_0, y \in \mathbb{R}^{20}, 0 \leq t \leq 60s \end{aligned} \tag{1}$$

2.2. Baseline Model

We show the failure of training of a POLLU baseline model. A baseline model has a neural network of one hidden layer and 50 nodes with a hyperbolic tangent activation function (tanh). The other settings are identical to the successful case shown in the paper. Figure 5 shows a loss is flatlined from 10,000 epochs. Failure of reproducing all the species can be seen in Figure 6.

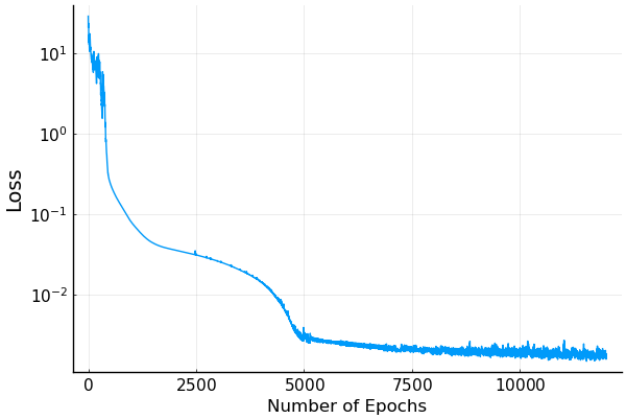


Figure 5. **Training of a POLLU baseline model: Loss profile along epochs**

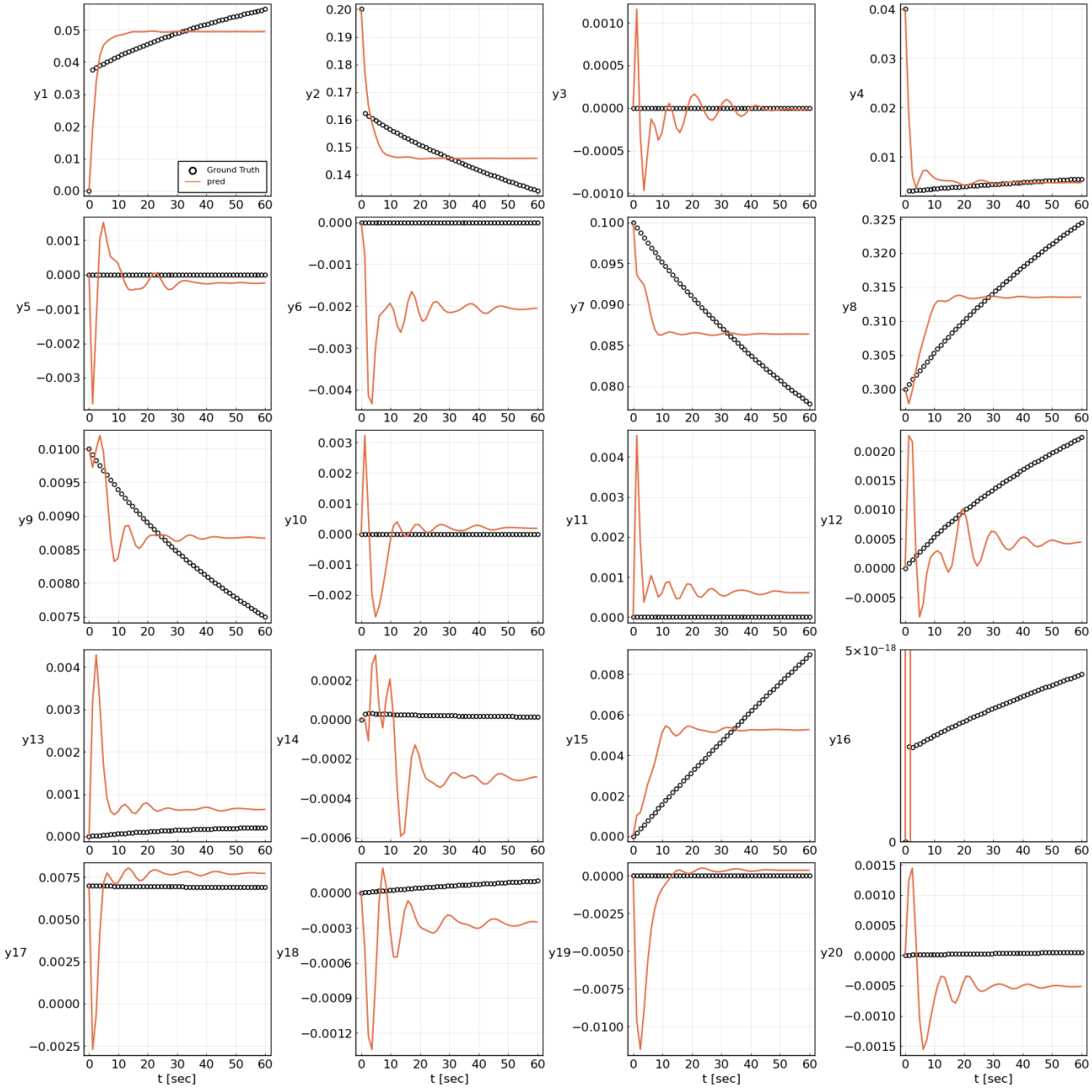


Figure 6. **Training of a POLLU baseline model: Species concentration profiles**

Table 2. Air pollution model: full equations and initial conditions

$\frac{d}{dt}$	$f\left(y\left(t\right)\right)$	$y\left(0\right)$
y_1	$-r_1 - r_{10} - r_{14} - r_{23} - r_{24} + r_2 + r_3 + r_9 + r_{11} + r_{12} + r_{22} + r_{25}$	0.0
y_2	$-r_2 - r_3 - r_9 - r_{12} + r_1 + r_{21}$	0.2
y_3	$-r_{15} + r_1 + r_{17} + r_{19} + r_{22}$	0.0
y_4	$-r_2 - r_{16} - r_{17} - r_{23} + r_{15}$	0.04
y_5	$-r_3 + 2r_4 + r_6 + r_7 + r_{13} + r_{20}$	0.0
y_6	$-r_6 - r_8 - r_{14} - r_{20} + r_3 + 2r_{18}$	0.0
y_7	$-r_4 - r_5 - r_6 + r_{13}$	0.1
y_8	$r_4 + r_5 + r_6 + r_7$	0.3
y_9	$-r_7 - r_8$	0.01
y_{10}	$-r_{12} + r_7 + r_9$	0.0
y_{11}	$-r_9 - r_{10} + r_8 + r_{11}$	0.0
y_{12}	r_9	0.0
y_{13}	$-r_{11} + r_{10}$	0.0
y_{14}	$-r_{13} + r_{12}$	0.0
y_{15}	r_{14}	0.0
y_{16}	$-r_{18} - r_{19} + r_{16}$	0.0
y_{17}	$-r_{20}$	0.007
y_{18}	r_{20}	0.0
y_{19}	$-r_{21} - r_{22} - r_{24} + r_{23} + r_{25}$	0.0
y_{20}	$-r_{25} + r_{24}$	0.0

Table 3. Reaction rates of the air pollution model

reaction	r	k	reaction	r	k
1	$k_1 y_1$	.350E+00	14	$k_{14} y_1 y_6$	.163E+05
2	$k_2 y_2 y_4$	.266E+02	15	$k_{15} y_3$	.480E+07
3	$k_3 y_5 y_2$	.120E+05	16	$k_{16} y_4$	.350E-03
4	$k_4 y_7$	.860E-03	17	$k_{17} y_4$	.175E-01
5	$k_5 y_7$	.820E-03	18	$k_{18} y_{16}$	.100E+09
6	$k_6 y_7 y_6$	.150E+05	19	$k_{19} y_{16}$	.444E+12
7	$k_7 y_9$	.130E-03	20	$k_{20} y_{17} y_6$	.124E+04
8	$k_8 y_9 y_6$	.240E+05	21	$k_{21} y_{19}$	.210E+01
9	$k_9 y_{11} y_2$	.165E+05	22	$k_{22} y_{19}$	.578E+01
10	$k_{10} y_{11} y_1$	.900E+04	23	$k_{23} y_1 y_4$	.474E-01
11	$k_{11} y_{13}$	.220E-01	24	$k_{24} y_{19} y_1$	.178E+04
12	$k_{12} y_{10} y_2$	.120E+05	25	$k_{25} y_{20}$	.312E+01
13	$k_{13} y_{14}$	.188E+01			