Stability-Informed Initialization of Neural Ordinary Differential Equations

Theodor Westny [1] **Arman Mohammadi** [1] **Daniel Jung** [1] **Erik Frisk** [1]

Abstract

This paper addresses the training of Neural Ordinary Differential Equations (neural ODEs), and in particular explores the interplay between numerical integration techniques, stability regions, step size, and initialization techniques. It is shown how the choice of integration technique implicitly regularizes the learned model, and how the solver's corresponding stability region affects training and prediction performance. From this analysis, a stability-informed parameter initialization technique is introduced. The effectiveness of the initialization method is displayed across several learning benchmarks and industrial applications.

1. Introduction

Learning dynamic systems from observations has a long history of applications, particularly in time-series analysis and prediction. This research domain is intrinsically connected to classical system identification and statistical model estimation methods (Ljung, 1999). While there are many early studies on using machine learning to adapt parameters of continuous functions from observations (Cohen & Grossberg, 1983; Rico-Martinez & Kevrekidis, 1993; Anderson et al., 1996; González-García et al., 1998), using deep architectures to learn from sequential data has largely been dominated by Recurrent Neural Networks (RNNs) (LeCun et al., 2015), with notable exceptions (Oord et al., 2016; Bai et al., 2018; Vaswani et al., 2017; Gu et al., 2022). However, with the introduction of Neural Ordinary Differential Equations (neural ODEs) (Chen et al., 2018), the idea of learning continuous functions using Neural Networks (NNs) has attracted the attention of several researchers. While neural ODEs have been successfully applied in image classification (Gholami et al., 2019; Zhang et al., 2019; Zhuang et al., 2020; Gusak et al., 2020), motivated by their syntactic similarities to that of Residual Networks (ResNets) (Chen et al., 2018; Gholami et al., 2019; Zhang et al., 2019; Queiruga et al., 2020; Ott et al., 2021), their strong inductive bias towards sequential modeling makes them attractive for applications that deal with sequential data (Rubanova et al., 2019; Yildiz et al., 2019; Kidger et al., 2020; Huang et al., 2020; Chen et al., 2021; Westny et al., 2023; Lipman et al., 2023; Verma et al., 2024).

This paper examines the impact of numerical solver choice on the performance and parameterization of models. We demonstrate that by careful consideration of the stability regions of the dynamic system with those of the selected solver, and by analyzing the eigenvalues from linearizing the models, the integration method and step size can significantly limit the space of learnable parameter values of NNs trained via stochastic gradient descent methods. Based on these observations, a stability-informed initialization technique is proposed for neural ODEs, yielding notable enhancements in training efficiency and prediction accuracy.

1.1. Contributions

The primary contributions of this paper are:

- An investigation into the effects of the numerical integration method on the parameters of neural ODEs and its connection to solver stability regions.
- Proposal of an Stability-Informed Initialization (SII) technique for neural ODEs.

Implementations are made publicly available.[2]

1.2. Related Work

The interest in neural ODEs has grown considerably since they were first introduced in (Chen et al., 2018). In light of their successful application, numerous research for a more comprehensive understanding of neural ODEs (Yan et al., 2019; Massaroli et al., 2020; Zhang et al., 2020; Gusak et al., 2020; Zhu et al., 2022) has been proposed with further improvements in (Gholami et al., 2019; Finlay et al., 2020; Zhuang et al., 2020; Xia et al., 2021; Krishnapriyan et al., 2022). The investigations in this paper target the effects of the numerical integration method and step size on the

[1]Department of Electrical Engineering, Linköping University, Linköping, Sweden. Correspondence to: Theodor Westny <theodor.westny@liu.se>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

[2]https://github.com/westny/neural-stability

learned model. A majority of related works that consider these questions mainly focus on the effect on prediction performance. However, an important aspect to consider when it comes to the effect of solver selection is not only their discretization error but also their stability regions (Ascher & Petzold, 1998), something that could be important when alternating solvers. Motivated by both the theoretical and practical benefits in a deep learning context, our analysis is mainly focused on fixed-step Runge-Kutta methods. In fact, the limitations of variable-step solvers when training neural ODEs was noted already in (Chen et al., 2018), some of which have been the target of later research (Gholami et al., 2019; Zhang et al., 2019; Zhuang et al., 2020).

It was observed in (Zhuang et al., 2020; Gusak et al., 2020; Mohammadi et al., 2023) that exchanging the numerical solver during testing from the one used during training considerably influenced performance. In (Zhuang et al., 2020), they demonstrated that training a neural ODE with an adaptive step solver and then utilizing fixed-step solvers of first, second, and fourth-order during inference led to a rise in error rates, albeit with a smaller increase as the order increased. In (Queiruga et al., 2020) and (Ott et al., 2021), these observations were explored in depth. In (Ott et al., 2021), it was observed that employing coarse discretization during training, followed by testing with an equally or less precise solver led to a considerable drop in performance.

Neural ODEs share several similarities with Deep State Space Models (SSMs), notably by using a continuous-time formulation. In the study by (Gu et al., 2020), an initialization technique for SSMs that enhances their ability to capture long-range dependencies was introduced—effectively integrated into the S4 model (Gu et al., 2022). However, S4 was observed to suffer from numerical instability during autoregressive generation, which was subsequently addressed in (Goel et al., 2022) by ensuring that the state matrix is constrained to be Hurwitz. Similarly, one can also draw connections between this work and methods in the reservoir computing literature (Lukoševičius & Jaeger, 2009) as well as the choice of stable dynamical systems in Random Projection Filter Banks (Farahmand et al., 2017). While these approaches share a thematic resemblance with our emphasis on stability, a key distinction of our work is the additional focus on the numerical solvers and their stability regions—an essential aspect when solving differential equations numerically and therefore also for the training and prediction processes.

2. Stability of Dynamic Systems and Solvers

A continuous-time dynamic model is described in ODE form by a state-transition function f as

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}; \theta) \tag{1}$$

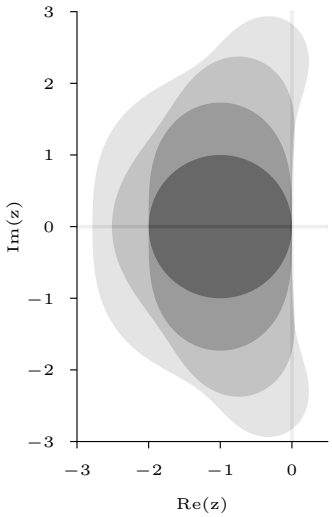

Figure 1: Stability regions for $p \in \{1, 2, 3, 4\}$-stage explicit RK methods of order p where $z = h\lambda$. The innermost circle represents the region of stability for the EF method, where $p = 1$. As p increases, so does the stability region.

where $\boldsymbol{x} \in \mathbb{R}^{d_x}$ is the dynamic state, and $\boldsymbol{u} \in \mathbb{R}^{d_u}$ an external (control) input, and θ the parameters of the model. Two important stability properties that are beneficial to consider when learning the function f, in particular when f is represented by a deep NN, are 1) the stability of the continuous-time system (1), and 2) the stability region of the numerical solver used to integrate (1).

2.1. Stability of Dynamic Systems

Stability in dynamic systems refers to the property of a system to return to its equilibrium state or to maintain its state of equilibrium after being subjected to disturbances (Ljung, 1999). In the context of control systems, stability is crucial to ensure the desired response and performance. The stability of a general dynamic system like (1) is complex (Khalil, 2002). However, for our purposes, it will prove sufficient to consider the stability of

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u}, \tag{2}$$

where the stability is directly related to the eigenvalues λ of the matrix $\boldsymbol{A}$ and exponential stability is ensured if all eigenvalues lie in the complex left half-plane $\lambda < 0$.

2.2. Stability Regions of Numerical Solvers

Differential equations like (1) do not, in general, have explicit analytical solutions and numerical integration techniques have to be used (Ascher & Petzold, 1998). Similarly to the stability of the system itself, it is important to consider the stability of the solver as this has a profound impact on the accuracy and stability of the solution trajectory. There

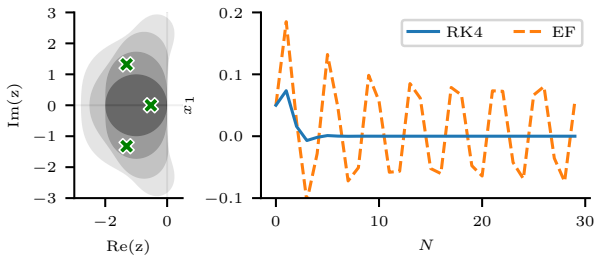

Figure 2: The figure depicts the scaled poles (green crosses) of a synthetic dynamic system f and the simulated response to a small perturbation using a 4th-order Runge-Kutta (RK4) and EF method.

are strong relationships between the properties of the system, the chosen solver, the step size, and the accuracy and stability of obtained numerical solutions.

Figure 1 illustrates the regions of absolute stability of explicit pth-order, $s = p \leq 4$ stage Runge-Kutta (RK) methods, where the 1-stage RK method is known as Euler forward (EF). The stability regions are given by the expression

$$\left| 1 + h\lambda + \frac{(h\lambda)^2}{2} + \cdots + \frac{(h\lambda)^p}{p!} \right| \leq 1, \tag{3}$$

where h is the step size and λ is the eigenvalue (Ascher & Petzold, 1998, pp. 87–89). From Figure 1 it can, e.g., be deduced which maximum step size h is possible given the dynamics of a system. Here, only fixed-step explicit methods are considered, but the approach can be directly extended to other integration methods. For example, several implicit solvers have well-defined stability regions (Ascher & Petzold, 1998, p. 143).

3. Implications for Neural ODEs

The concept of stability regions is an important property of numerical ODE solvers (Ascher & Petzold, 1998). However, these stability regions and their implications on the learning and prediction performance of neural ODEs have yet to be investigated in detail.

Consider a dynamic system $\dot{\boldsymbol{x}} = f(\boldsymbol{x})$, modeled using a feedforward NN with nonlinear activations. In Figure 2, we illustrate the poles of the linearized system within the complex plane, i.e., the eigenvalues of the Jacobian matrix $\boldsymbol{A} = \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}}|_{\boldsymbol{x}=\boldsymbol{0}}$, scaled by the step size h. Clearly, the eigenvalues are well within the stability region of RK4 and simulating it using that particular solver yields a stable response. However, for the EF method, the two complex eigenvalues are outside of its stability region, and the simulated response exhibits oscillatory behavior resulting in large prediction errors. This is an important property to highlight in the context of neural ODEs, since training a model using a higher-order method might yield a model with eigenmodes that are problematic for a lower-order method. For example, in (Queiruga et al., 2020; Ott et al., 2021) it was observed that changing the numerical solver during inference from the solver employed during training can have a profound impact on the overall performance. While they concluded that the behavior was due to the discretization error, this could also be attributed to the solvers' stability regions (Mohammadi et al., 2023).

3.1. Learned Model Dynamics

The choice of numerical solver affects the placement of the learned model dynamics. To illustrate, Figure 3 shows the kernel density estimate of the eigenvalues, for a linearization around a zero reference of 400 different neural ODEs, depicted in the complex plane. The models are parametrized using feedforward NNs and trained using the EF, Midpoint (MP), and 3rd-order Runge-Kutta (RK3) methods, respectively. They are all tasked with learning the dynamics of a linear system (one unique reference system for each model) with 3 dynamic states that are unattainable for each solver since the eigenvalues of the state-transition jacobian of the reference systems, shown with green crosses, are manually placed outside of the respective solvers' stability region. With some slight abuse of notation, the eigenvalues of the state-transition jacobian will be referred to as the system poles.

The training and test data consist of $N = 100$ long sequences, generated by simulating the reference system using RK4. The poles of the learned models used for the illustrations are based on the version of the model that achieves minimum test loss during training. Several important conclusions can be drawn from the observed results.

Regardless of the nature of the system that the model should learn, the poles are contained within the stability region of the solver. This follows intuition since poles outside of the respective stability regions would yield poor prediction performance (cf. Section 3).

It is evident from the illustrations and observations made during training that the models display a limited tendency to shift poles to the left within the complex plane. Note that poles with large negative real parts give rise to states with fast dynamics, and quick transient responses, something the models struggle to learn. We hypothesize that this phenomenon is related to the spectral bias of NNs (Rahaman et al., 2019; Basri et al., 2020; Tancik et al., 2020), as it has been observed that NNs encounter difficulties when learning higher frequencies. Lack of excitation of high-frequency dynamics could also be a contributing factor (Ljung, 1999).

Another important observation to note is the high density of

poles around the origin. The cause of this can be traced back to the initialization technique employed here — also the default in PyTorch (Paszke et al., 2019). In this method, parameter values are initialized from $\mathcal{U}(-1/\sqrt{d_{\text{in},i}}, 1/\sqrt{d_{\text{in},i}})$, where $d_{\text{in},i}$ is the input dimension of the i-th layer. When initializing using this method, the linearized system poles are contained within a ball around the origin with a radius dependent on the number of parameters. This makes learning dynamic systems more difficult since they all have to be moved from the origin during training. Further, with such an initialization strategy some poles lie in the right half-plane, thus rendering the system unstable which makes training more difficult. An important consideration here is also the effect of using regularization strategies for neural ODEs since penalties on the learnable parameters would have the effect of pushing the poles closer to the origin.

These findings motivate the potential need to develop initialization strategies more suitable for neural ODEs used to learn dynamic systems. Since the poles of these models are approximately contained within the stability region of the employed solver, it makes intuitive sense to initialize the model within that same region, while also adhering to the stability requirements of dynamic systems. Furthermore, since the model is stable from the start of training, this allows for training with longer prediction horizons without risking divergence. In addition, motivated by the increased difficulty of learning fast dynamics, we hypothesize that by initializing the model with poles further into the left-half plane the model will experience a smoother learning process.

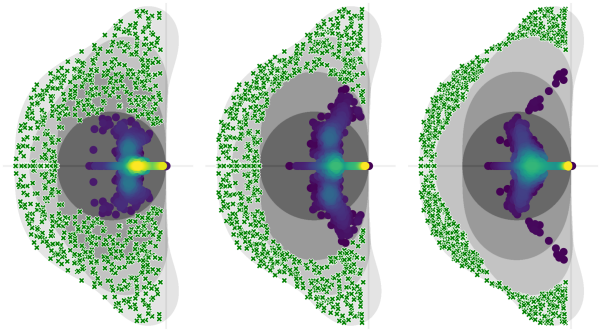

Figure 3: Kernel density estimate of learned model poles based on the approximate linearized system when using the EF, MP, and an RK3 method (from left to right). The kernel density estimates are based on a total of $3 \cdot 400$ poles (no. of states $\times$ no. of models). The references are various linear systems with 3 states. The combined poles of all linear systems are illustrated with green crosses.

4. Stability-Informed Initialization

Consider a dynamic system in the form (1) where f is assumed to be a feedforward NN with depth n in the form

$$f(\mathbf{x}) = \boldsymbol{W}_n \sigma(\cdots \sigma(\boldsymbol{W}_1 \mathbf{x} + \boldsymbol{b}_1) \cdots) + \boldsymbol{b}_n, \tag{4}$$

where σ is any nonlinearity (activation function), $\boldsymbol{W}_i \in \mathbb{R}^{d_{h_i} \times d_{h_{i-1}}}$ and $\boldsymbol{b}_i \in \mathbb{R}^{d_{h_i}}$ are the learnable weights and biases of the model, and d_{h_i} is the hidden dimension of layer $i \in \{1, \ldots, n\}$. It is assumed that no layer has a dimension less than the state dimension d_x. Finally, the NN input is $\mathbf{x} = \boldsymbol{x} \oplus \boldsymbol{u}$, where $\oplus$ is the concatenation operation.

Uninformed random initialization of the weights $\boldsymbol{W}_i$ and biases $\boldsymbol{b}_i$ will with high probability give an initial model that violates the stability constraints highlighted in Section 2. As detailed in Section 3, this has a considerable impact on training and will be further discussed in the remainder of the paper. Now, a rejection sampling-based initialization procedure of the parameters in (4) is described that respects the stability region of the chosen solver.

4.1. Parameter Initialization

The key idea of the approach is to consider a linearization of (4) for sufficiently small $\boldsymbol{b}_i$ and assume an activation function σ with a linear region. This assumption holds for several widely used activation functions such as the ReLU and its variants. For activation functions with variable slopes, like sigmoid or tanh, the linear coefficient can be conservatively replaced by their maximum slope at the origin. This scales the eigenvalues towards the origin while still maintaining the stability properties of the system.

Note that the stability of the linearized system does not necessarily imply the stability of the nonlinear system, or even of a linear time-varying system, as discussed in (Khalil, 2002). However, analyzing the stability of the linearized system can provide a useful starting point for initialization.

Let κ be the slope of the activation function around the origin, then it holds that

$$\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} \approx \kappa^n \, \boldsymbol{W}_n \boldsymbol{W}_{n-1} \cdots \boldsymbol{W}_1 = \boldsymbol{A}, \tag{5}$$

which is exactly an n-layer *linear* feedforward network. This expression is central to the initialization technique—outlined next. Without loss of generality and for simplicity of the presentation, κ is from now on assumed 1. Note that, $\boldsymbol{W}_i$ should be scaled by $1/\kappa_i$ if $\kappa_i \neq 1$.

Weight matrices To illustrate the procedure, first, consider the simple case where the desired eigenvalues of $\boldsymbol{A}$ in (5) are $\lambda_j \in \mathbb{R}$ and let $\boldsymbol{W}_i \in \mathbb{R}^{d_x \times d_x}$ be all identical

diagonal square matrices

$$\boldsymbol{W}_1 = \cdots = \boldsymbol{W}_n = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_{d_x})^{1/n}. \quad (6)$$

Then, clearly, $\boldsymbol{A}$ in (5) has the desired eigenvalues λ_j. The nth root is used to allocate information about each eigenvalue throughout the whole feedforward network. This constitutes one alternative to initializing the weights $\boldsymbol{W}_i$ but the approach has several restrictions, described next.

First, consider the case when one wants the eigenvalues to be complex. With real weights $\boldsymbol{W}_i \in \mathbb{R}^{d_{h_i} \times d_{h_{i-1}}}$ in (4), this requires that complex eigenvalues come in conjugate pairs. Consequently, when d_x is odd, the system must have a least one real eigenvalue. Regardless, the construction is still straightforward. Let $\lambda_k \in \mathbb{C}$ and, as before, compute the nth (principal) root as $\lambda_k^{1/n} = \mu_k + j\omega_k$ where j is the imaginary unit. First, note that the matrix

$$J_{\lambda_k} = \begin{bmatrix} \mu_k & \omega_k \\ -\omega_k & \mu_k \end{bmatrix} \quad (7)$$

has exactly the eigenvalues $\mu_k \pm j\omega_k$. Next, block-diagonal $\boldsymbol{W}_i$:s are constructed where the kth block J_{λ_k} from above yields

$$\boldsymbol{W}_1 = \cdots = \boldsymbol{W}_n = J_{\lambda_1} \oplus J_{\lambda_2} \oplus \cdots \oplus J_{\lambda_{d_x/2}} = \\ = \begin{bmatrix} \mu_1 & \omega_1 & 0 & \cdots & 0 \\ -\omega_1 & \mu_1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \mu_{d_x/2} & \omega_{d_x/2} \\ 0 & 0 & 0 & -\omega_{d_x/2} & \mu_{d_x/2} \end{bmatrix} \quad (8)$$

where $\boldsymbol{W}_i$ has all real entries and the compound $\boldsymbol{A}$ has the desired eigenvalues.

Next, we need to address the dimensionality and sparsity of the weight matrices. Certainly, $\dim(\boldsymbol{W}_i) \neq \dim(\boldsymbol{A})$ is most often the case when constructing a feedforward network like (4). Additionally, to avoid sparse weight matrices where there is a risk of slow learning due to the vanishing gradient problem (Basodi et al., 2020; Pascanu et al., 2013), we would like to avoid matrix elements with a value of zero. Finally, similar to established initialization techniques (Glorot & Bengio, 2010; Saxe et al., 2014; He et al., 2015), adding randomness to the weight initialization assures that diverse models can be generated, but also crucially breaks symmetry among neurons—enabling diverse feature extraction. To meet these requirements, the previous ideas presented above are generalized as

$$\boldsymbol{A} = \underbrace{\Lambda_n \Pi_{n-1}^{-1}}_{\boldsymbol{W}_n} \underbrace{\Pi_{n-1} \Lambda_{n-1} \Pi_{n-2}^{-1}}_{\boldsymbol{W}_{n-1}} \cdots \underbrace{\Pi_2 \Lambda_2 \Pi_1^{-1}}_{\boldsymbol{W}_2} \underbrace{\Pi_1 \Lambda_1}_{\boldsymbol{W}_1}, \quad (9)$$

where Λ_i is used to address the dimension property and Π_i the denseness and randomness property.

The matrices $\Lambda_i \in \mathbb{R}^{d_{h_i} \times d_{h_{i-1}}}$ are constructed to inherit the properties of (6) and (8) for real and complex eigenvalues respectively such that they contain a factor of the system eigenvalues. To ensure that the dimensions of the respective matrix products adhere to the hidden dimensions of the layers in the network, the matrices are extended with blocks of zeros. Consider for example the simple case when $d_x = 2$ and $d_{h_1} = 4$, then

$$\Lambda_1 = \begin{bmatrix} \mu_1 & \omega_1 & 0 & 0 \\ -\omega_1 & \mu_1 & 0 & 0 \end{bmatrix}^{\mathrm{T}} \quad (10)$$

The matrices $\Pi_i \in \mathbb{R}^{d_{h_i} \times d_{h_i}}$ are random orthogonal matrices sampled from the Haar distribution (Stewart, 1980; Mezzadri, 2006). In practice, these are constructed by applying QR decomposition of a $d_{h_i} \times d_{h_i}$ matrix whose elements are independently normally distributed with zero mean and unit variance (Mezzadri, 2006). The choice of using orthogonal matrices is natural since their eigenvalues being 1 ensures that the complete model's eigenvalues are neither attenuated nor amplified along the computational graph and their property of having their transpose equal to their inverse offers numerical and implementational benefits. Note that introducing the random matrices Π_i does not change (9) since

$$\boldsymbol{A} = \Lambda_n \Pi_{n-1}^{-1} \cdots \Pi_2 \Lambda_2 \Pi_1^{-1} \Pi_1 \Lambda_1 = \Lambda_n \cdots \Lambda_2 \Lambda_1, \quad (11)$$

although their inclusion assures that $\forall \mathrm{w}_{j,k} \in \boldsymbol{W}_i, \mathrm{w}_{j,k} \neq 0$ with probability 1. Although not explored in this work, we note that it is possible to use other eigenvectors than the standard basis when constructing $\boldsymbol{A}$.

Dynamic inputs To include external inputs, the vector $\boldsymbol{u}$ is assumed to be concatenated with the state vector $\boldsymbol{x}$ prior to propagation. This is easily accounted for by modifying Λ_1 such that $\Lambda_1 \in \mathbb{R}^{d_{h_1} \times (d_x + d_u)}$. Continuing the example in (10), with $d_u = 1$, then

$$\Lambda_1 = \begin{bmatrix} \mu_1 & \omega_1 & 0 & 0 \\ -\omega_1 & \mu_1 & 0 & 0 \\ \nu_1 & \nu_2 & 0 & 0 \end{bmatrix}^{\mathrm{T}}, \quad (12)$$

where $\nu_k \sim \mathcal{U}(-\zeta, \zeta)$ and $\zeta = \frac{1}{n} \sum_{i=n} |\mu_i|$.

Bias terms Given that the linearization holds for $\boldsymbol{b}_i$ close to $\boldsymbol{0}$, it might seem appealing to initialize $\boldsymbol{b}_i = \boldsymbol{0}$ in order to satisfy the approximation. However, this could lead to ineffective learning of $\boldsymbol{b}_i$. To that end, initialization of the bias terms $\boldsymbol{b}_i$ is done according to

$$\boldsymbol{b}_i \sim \mathcal{U}(-\varepsilon \cdot \boldsymbol{1}_{d_{h_i}}, \varepsilon \cdot \boldsymbol{1}_{d_{h_i}}), \quad (13)$$

where $\boldsymbol{1}_{d_{h_i}}$ is a vector of ones with size d_{h_i} and ε is a tunable (small) value, here set to $\varepsilon = 10^{-4}$.

Algorithm 1 Stability Region Rejection Sampling

Input: Solver order p, step size h, state dimension d_x
Initialize: $\mathcal{E} \leftarrow \{\}$
repeat
  Sample: $\mu \sim \mathcal{U}(-3.0, -0.1)$
  if $|\mathcal{E}| < d_x - 2$ and `use_complex` **then**
    Sample: $\omega \sim \mathcal{U}(-3.0, 3.0)$
  else
    Set: $\omega = 0$
  end if
  Form: $\lambda = \mu + j\omega$
  if $\left|1 + \lambda + \frac{\lambda^2}{2} + \cdots + \frac{\lambda^p}{p!}\right| < 1 - \epsilon$ **then**
    $\mathcal{E} \leftarrow \mathcal{E} \cup \{\lambda/h\}$
    if $\omega \neq 0$ **then**
      $\mathcal{E} \leftarrow \mathcal{E} \cup \{\bar{\lambda}/h\}$
    end if
  else
    Reject and proceed to the next iteration
  end if
until $|\mathcal{E}| = d_x$
Return: $\mathcal{E}$

Summary Now that a method to randomly initialize the feedforward network (4) given a set of eigenvalues has been outlined, the procedure is completed using a simple rejection sampling algorithm. This involves randomizing real and/or complex-conjugate eigenvalue pairs within the stability region of the selected solver for a given step size based on condition (3). This procedure is outlined in Algorithm 1. The rejection-sampling algorithm produces a set of eigenvalues that are utilized to form Λ_i as described in (10) and (12). Subsequently, random orthogonal matrices Π_i are sampled. These matrices, together with Λ_i are used to construct the weights W_i as outlined in (9). The process concludes with the construction of the bias terms as in (13).

5. Teacher-Student Regression

Model We examine a teacher-student NN setting in which a student network is trained to replicate the output of a teacher network. The teacher is designed to simulate a nonlinear dynamic system and is adaptable to various dynamic states and inputs, allowing for a rich simulation study. Both the teacher and student models are outfitted with nonlinear ELU (Clevert et al., 2015) activation functions. The main considered input was in the form of pulse-width modulated waveforms w, meant to model a continuous-time input that excites model dynamics. The training and test set inputs $\boldsymbol{u}$ are constructed using different periodic functions with varying frequencies together with added noise

$$\boldsymbol{u} = [w_k + v_k \quad \dots \quad w_{k+N} + v_{k+N}], \tag{14}$$

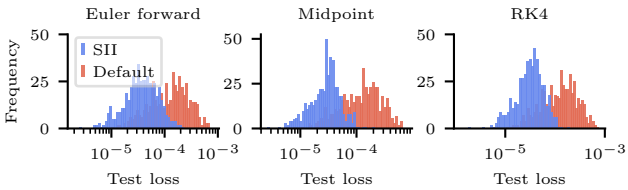

Figure 4: Histogram over minimum test loss when training on 500 different teachers, initialized with poles within the first-order region.

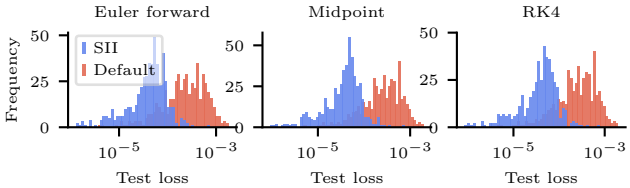

Figure 5: Histogram over minimum test loss when training on 500 different teachers, initialized with poles outside of all stability regions.

where $v \sim \mathcal{N}(0, 0.1)$ and N is the number of samples. Data generation then proceeds by solving the Initial Value Problem (IVP)

$$\boldsymbol{x} = \boldsymbol{x}_0 + \int_0^t \mathcal{T}(\tau, \boldsymbol{x}(\tau), \boldsymbol{u}(\tau))\mathrm{d}\tau, \quad t = N \cdot \Delta t \tag{15}$$

where $\mathcal{T}$ is the teacher model, $\boldsymbol{x}_0 \sim \mathcal{U}[0, 1)$ the initial state, and $\boldsymbol{x}$ the corresponding solution, recovered using a numerical solver, in this case, the variable-step Dormand-Prince method (Dormand & Prince, 1980). The state trajectory $\boldsymbol{x}$ is determined by the integration length t, where Δt is the sampling period. When solving the IVP, the teacher utilizes linear interpolation to compute intermediate input values.

Training We conduct a study, using 500 random seeds (and teachers), training various student models using different hyperparameters, solvers, and initialization techniques.

Results The first investigation is done by initializing the teacher model with 4 dynamic states using the proposed technique such that the poles are inside of the first-order stability region. Each student model has the same hyperparameter configuration as the teachers, in this case, 3 layers with 128 nodes in each layer. Despite the randomness in the example, there is a clear tendency toward lower loss values using the proposed technique, with orders of magnitude smaller losses (see Figure 4).

In general, the eigenvalues of the true system to be learned from training data are rarely known. To simulate the case when neither the solver nor the step size is chosen optimally, the second investigation is conducted to study how well the

nonlinear students can approximate the data regardless. This is done by initializing the teacher model with 2 dynamic states using the proposed technique such that the poles are outside of the stability region of the employed solver. To add to the difficulty of the task, each student model is under-parameterized relative to the teacher. While the teacher has 3 layers with 128 nodes in each layer, the student has 2 layers with 32 nodes in each layer.

In Figure 5, the histograms of the minimum test loss are shown for the different solvers and initializations. Similar to the first investigation, the results show a clear improvement.

6. Experiments

To assess the performance and usability of the proposed technique, several experiments are conducted across a diverse set of problem domains. Our choice of experimental cases was guided by three principal criteria: (i) prevalence in the literature, (ii) complexity and diversity of the tasks, and (iii) real-world applicability and relevance. For each experimental scenario, multiple instances of the models are trained under different random seeds to ensure the reproducibility and statistical significance of the results. Importantly, the configuration of hyperparameters, e.g., batch size and learning rate may differ with the task but is always the same for all methods. Unless stated otherwise, the models are trained and tested using the Midpoint method.

6.1. Pixel-Level Image Classification

Drawing inspiration from (Le et al., 2015; Gu et al., 2022), the original MNIST (LeCun et al., 1998) and CIFAR-10 (Krizhevsky et al., 2009) datasets are modified for a sequential classification task. This modification involves presenting images to the model one pixel at a time, requiring the model to classify the image only after all pixels have been observed. These tasks have been popularized as benchmarks for long-range dependency problems.

Model We utilize an encoder-decoder-like architecture, embedding each pixel $\boldsymbol{u}_k \in [0, 1]$ into a high-dimensional vector $\tilde{\boldsymbol{u}}_k \in \mathbb{R}^{d_{\tilde{u}}}$ through a linear-affine transformation. These vectors are sequentially fed into a neural ODE, which are used to compute the time derivative of the latent state $\boldsymbol{z}_k \in \mathbb{R}^{d_z}$, expressed as $\dot{\boldsymbol{z}} = f(\boldsymbol{z}, \tilde{\boldsymbol{u}})$, where f is a neural network. Through numerical integration, the latent state is updated for each new pixel. After processing all pixels, the sequence's final state is input into a two-layer feedforward network to predict the class probabilities $\hat{\boldsymbol{y}}_N = h(\boldsymbol{z}_N)$, where N is the number of pixels.

Training and testing We incorporate principles from early-exit networks (Teerapittayanon et al., 2016) to enhance feature propagation during the encoding stage by training the model to predict the correct class at all intermediate steps. The loss function is formulated as

$$\mathcal{L}_{\text{CE}} = \sum_{k=1}^{N} \ell_{CE}(\hat{\boldsymbol{y}}_k, \boldsymbol{y}) \cdot \boldsymbol{w}_k, \tag{16}$$

where ℓ_{CE} is the standard Cross-Entropy (CE) criterion, and $\boldsymbol{w}_k$ is an exponentially increasing term that assigns more importance to later predictions. During testing, the model output is taken as the final prediction $\hat{\boldsymbol{y}}_N$ in the sequence.

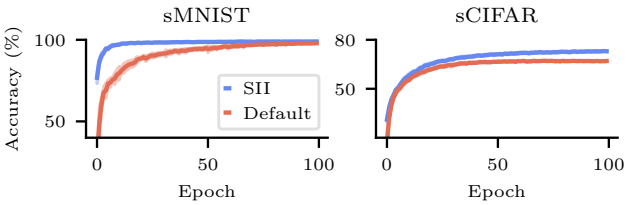

Figure 6: Test accuracy over training epochs on sequential MNIST and CIFAR-10. Solid lines represent the mean test accuracy, and the shaded regions illustrate the approximate 99% confidence interval around the mean.

Results For each dataset, 10 model instances for each initializing technique were trained. The resulting accuracy is summarized in Table 1, reporting the mean and standard deviation. Here, pMNIST refers to the permuted MNIST variant (Goodfellow et al., 2013). It is clear from the results that using the proposed initializing technique provides positive improvements in test accuracy. Additionally, employing SII shows clear enhancement in training efficiency, achieving convergence at greater speed. This effect is depicted in Figure 6, where the test acccuracy curves are illustrated across the training epochs.

Table 1: Pixel-Level Classification Accuracy (mean $\pm$ std.)

	sMNIST	spMNIST	sCIFAR
Default	97.9 ± 0.20 %	90.4 ± 0.41 %	67.6 ± 0.15 %
SII	$\mathbf{99.0} \pm 0.09$ %	$\mathbf{94.4} \pm 0.26$ %	$\mathbf{73.3} \pm 0.32$ %

6.2. Latent Dynamics

This evaluation centers on benchmarks introduced in (Botev et al., 2021). These benchmarks encompass three core tasks: (1) embed a sequence of images $(\boldsymbol{x}_0, \ldots, \boldsymbol{x}_t)$ to a lower-dimensional abstract state $\boldsymbol{z}_t$, (2) simulating the system in the latent space using a continuous dynamic model, and (3) mapping the predicted sequence $(\boldsymbol{z}_{t+1}, \ldots, \boldsymbol{z}_{t+N})$ back into the image domain $(\boldsymbol{x}_{t+1}, \ldots, \boldsymbol{x}_{t+N})$.

Model While our approach is largely inspired by the proposal in (Toth et al., 2020), the model structure used is based

on the U-Net architecture (Ronneberger et al., 2015). Given a sequence of images $[\boldsymbol{x}_0, \dots, \boldsymbol{x}_t] \in \mathbb{R}^{t\times 3\times 32\times 32}$, the first step is to transpose the channel and temporal dimension. By treating the temporal dimension as an additional feature alongside the image height and width, 3D convolution and pooling operations are utilized for input embedding. The encoder (contracting path), compresses the image sequence into the abstract state $\boldsymbol{z}_t \in \mathbb{R}^{d_z\times 4\times 4}$ while simultaneously increasing the channel dimension ($d_z \gg 3$). After forward integration of the initial abstract state, the decoder (expansive path) reconstructs the predicted states into a sequence of images $[\hat{\boldsymbol{x}}_{t+1}, \dots, \hat{\boldsymbol{x}}_{t+N}] \in \mathbb{R}^{N\times 3\times 32\times 32}$.

Training and testing For this study, a simple summed Mean-Squared Error (MSE) loss is used for training

$$\mathcal{L}_{\text{MSE}} = \sum_{k=1}^{N} \left(\hat{\boldsymbol{x}}_{t+k} - \boldsymbol{x}_{t+k}\right)^2 \tag{17}$$

where $\hat{\boldsymbol{x}}$ is the prediction and $\boldsymbol{x}$ is the ground truth image. During training, we set $N = 60$ and $t = 10$.

During testing, the prediction horizon is increased in order to assess the long-term prediction and extrapolation capabilities of the respective models. For a more fair comparison across datasets, performance is reported using the normalized MSE (Zhong et al., 2021):

$$\text{MSE}_n = \frac{1}{N} \sum_{k=1}^{N} \frac{\left(\hat{\boldsymbol{x}}_{t+k} - \boldsymbol{x}_{t+k}\right)^2}{\boldsymbol{x}_{t+k}^2} \tag{18}$$

Results Investigations are conducted on four out of the 17 datasets in (Botev et al., 2021): the Spring-mass system (Mass Spring), the Double Pendulum, Molecular Dynamics (16 particles), and the 3D Room (consisting of MuJoCo (Todorov et al., 2012) scenes). Ten model instances were trained for each dataset under each initialization technique. The resulting normalized MSE is summarized in Table 2, illustrating clear improvements across all datasets when employing SII. Notably, the experiments revealed that the benefits of SII became increasingly evident as the prediction horizon was extended, a trend that can be hypothesized to stem from its stability properties.

Table 2: Latent Dynamics MSE_n (mean $\pm$ std.)

	Default	SII
Mass Spring	0.102 ± 0.015	$\mathbf{0.066} \pm 0.008$
Double Pend.	0.084 ± 0.017	$\mathbf{0.061} \pm 0.002$
Molecular Dyn.	0.151 ± 0.036	$\mathbf{0.106} \pm 0.011$
3D Room	0.588 ± 0.102	$\mathbf{0.176} \pm 0.055$

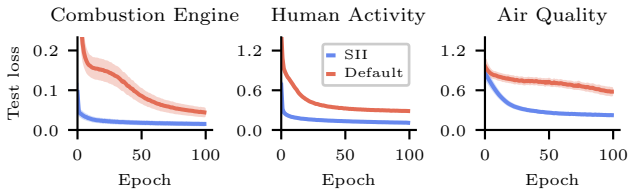

Figure 7: Test loss over the total number of training epochs on the time-series forecasting datasets. Solid lines represent the mean test loss and the shaded regions illustrate the approximate 99% confidence interval around the mean.

6.3. Multivariate Time-Series Forecasting

This study aims to develop a model capable of simulating the dynamics of an unknown system. The model is tasked with forecasting specific sensor reference values based on signals received from other components within the system:

$$\dot{\boldsymbol{z}} = f_\gamma(\boldsymbol{z}, \boldsymbol{u}) \tag{19a}$$

$$\hat{\boldsymbol{y}} = h_\theta(\boldsymbol{z}, \boldsymbol{u}) \tag{19b}$$

where $\hat{\boldsymbol{y}} \in \mathbb{R}^{d_y}$ is the measurement prediction, and $\boldsymbol{u} \in \mathbb{R}^{d_u}$ is a vector of input measurement signals. Importantly, the system is partly described by a state-transition function (19a), with unknown states and dynamics. In order to compute predictions according to (19b), the latent states $\boldsymbol{z} \in \mathbb{R}^{d_z}$ must be solved for, which requires that $\boldsymbol{z}_0$ is known. Inspired by Variational Autoencoders (VAEs) (Kingma & Welling, 2014; Rezende et al., 2014), we employ an encoder network g_ϕ that outputs the variational posterior over the initial state $\boldsymbol{z}_0 \sim q_\phi(\boldsymbol{z}_0|\boldsymbol{u}_0)$ and parametrize it as a diagonal Gaussian with learnable parameters.

The study utilizes three distinct datasets. The first dataset features standard signals (commonly found in commercial vehicles) from an internal combustion engine, recorded over multiple driving cycles (Jung, 2022). The second dataset, referred to as the Human Activity dataset (Vidulin et al., 2010), features data from five individuals, each wearing four localization tags (left ankle, right ankle, belt, chest). The third dataset, referenced as the Air Quality dataset (Vito, 2016) encompasses a set of air quality measurements from an urban monitoring station in Italy. It provides measurements of various air pollutants and particulate matter, along with environmental factors such as temperature and humidity.

Training and testing Training targets the optimization of the sequential Evidence Lower Bound (ELBO):

$$\begin{aligned}\mathcal{L}_{\text{ELBO}} = \mathbb{E}_{q_\phi(\boldsymbol{z}_0|\boldsymbol{u}_0)} \left[\textstyle\sum_{k=1}^{N} \log p_\theta(\boldsymbol{y}_k|\boldsymbol{z}_k, \boldsymbol{u}_k)\right] \\ - D_{\text{KL}}\left(q_\phi(\boldsymbol{z}_0|\boldsymbol{u}_0) \,||\, p(\boldsymbol{z}_0)\right),\end{aligned} \tag{20}$$

where $\boldsymbol{y}$ is the ground truth measurement and $p(\boldsymbol{z}_0) = \mathcal{N}(\mathbf{0}, I)$ is the prior over the initial latent states. The omis-

sion of $\frac{1}{N}$ in front of the reconstruction term is done intentionally to force the model to focus more on the prediction error, something that was found to improve validation performance. Test performance is reported using MSE.

Results A total of 3×10 different models per initialization method were trained, consisting of 3 random data splits (to mitigate some selection bias (Cawley & Talbot, 2010)) with 10 different model configurations each. Test performance is presented in Table 3. Using SII results in a significant improvement in test performance compared to the default initialization. This enhancement is further supported by the faster convergence rate, as depicted in Figure 7.

Table 3: Multivariate Time-Series MSE (mean $\pm$ std.)

	Combustion Engine	Human Activity	Air Quality
Default	0.04 ± 0.019	0.29 ± 0.006	0.57 ± 0.103
SII	$\mathbf{0.01} \pm 0.005$	$\mathbf{0.11} \pm 0.003$	$\mathbf{0.22} \pm 0.020$

7. Scope and limitations

Learnable model While the proposed technique is specific to a single class of feedforward NNs, the method could potentially be extended to any model class with linear operations, e.g., convolutional neural networks. Furthermore, the hidden dimension of each layer must be equal to or larger than the number of dynamic states to preserve the eigenvalues.

8. Conclusions

This paper analyses how the stability properties of the system and stability regions of the chosen numerical solver affect the training and prediction performance of neural ODEs. Further, it is illustrated how standard techniques for the initialization of the network, without considering the stability properties of the system and solver, may lead to slow training and suboptimal performance. Based on this, a general stability-informed initialization (SII) method is developed that adapts to the stability region of a chosen solver. The effectiveness of the approach is demonstrated in various machine-learning tasks, including successful applications to real-world measurement data. In all cases, increased efficiency of training, less susceptibility to random fluctuations, and improved model performance are demonstrated.

Acknowledgements

This research was supported by the Strategic Research Area at Linköping-Lund in Information Technology (ELLIIT) and the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. Computations were enabled by the Berzelius resource provided by the Knut and Alice Wallenberg Foundation at the National Supercomputer Centre. The authors would like to thank the reviewers for their insightful comments and suggestions, which have significantly improved the manuscript.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Anderson, J., Kevrekidis, I., and Rico-Martinez, R. A comparison of recurrent training algorithms for time series analysis and system identification. *Computers & Chemical Engineering*, 20:S751–S756, 1996. 1

Ascher, U. M. and Petzold, L. R. *Computer methods for ordinary differential equations and differential-algebraic equations*, volume 61. SIAM, 1998. 2, 3

Bai, S., Kolter, J. Z., and Koltun, V. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018. 1

Basodi, S., Ji, C., Zhang, H., and Pan, Y. Gradient amplification: An efficient way to train deep neural networks. *Big Data Mining and Analytics*, 3(3):196–207, 2020. 5

Basri, R., Galun, M., Geifman, A., Jacobs, D., Kasten, Y., and Kritchman, S. Frequency bias in neural networks for input of non-uniform density. In *International Conference on Machine Learning*, pp. 685–694. PMLR, 2020. 3

Botev, A., Jaegle, A., Wirnsberger, P., Hennes, D., and Higgins, I. Which priors matter? benchmarking models for learning latent dynamics. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021. 7, 8

Cawley, G. C. and Talbot, N. L. On over-fitting in model selection and subsequent selection bias in performance evaluation. *The Journal of Machine Learning Research*, 11:2079–2107, 2010. 9

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, volume 31, 2018. 1, 2

Chen, R. T. Q., Amos, B., and Nickel, M. Learning neural event functions for ordinary differential equations. In *International Conference on Learning Representations*, 2021. 1

Clevert, D.-A., Unterthiner, T., and Hochreiter, S. Fast and accurate deep network learning by exponential linear units (ELUs). *arXiv preprint arXiv:1511.07289*, 2015. 6

Cohen, M. A. and Grossberg, S. Absolute stability of global pattern formation and parallel memory storage by competitive neural networks. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):815–826, 1983. 1

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. 6

Farahmand, A.-m., Pourazarm, S., and Nikovski, D. Random projection filter bank for time series data. *Advances in Neural Information Processing Systems*, 30, 2017. 2

Finlay, C., Jacobsen, J.-H., Nurbekyan, L., and Oberman, A. How to train your neural ODE: the world of Jacobian and kinetic regularization. In *International Conference on Machine Learning*, pp. 3154–3164. PMLR, 2020. 1

Gholami, A., Keutzer, K., and Biros, G. ANODE: Unconditionally accurate memory-efficient gradients for neural ODEs. In *International Joint Conference on Artificial Intelligence*, 2019. 1, 2

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010. 5

Goel, K., Gu, A., Donahue, C., and Ré, C. It's raw! Audio generation with state-space models. In *International Conference on Machine Learning*, pp. 7616–7633. PMLR, 2022. 2

González-García, R., Rico-Martìnez, R., and Kevrekidis, I. G. Identification of distributed parameter systems: A neural net based approach. *Computers & chemical engineering*, 22:S965–S968, 1998. 1

Goodfellow, I. J., Mirza, M., Xiao, D., Courville, A., and Bengio, Y. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013. 7

Gu, A., Dao, T., Ermon, S., Rudra, A., and Ré, C. HiPPO: Recurrent memory with optimal polynomial projections. *Advances in Neural Information Processing Systems*, 33: 1474–1487, 2020. 2

Gu, A., Goel, K., and Re, C. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2022. 1, 2, 7

Gusak, J., Markeeva, L., Daulbaev, T., Katrutsa, A., Cichocki, A., and Oseledets, I. Towards understanding normalization in neural ODEs. In *International Conference on Learning Representations*, 2020. 1, 2

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *IEEE International Conference on Computer Vision*, pp. 1026–1034, 2015. 5

Huang, Z., Sun, Y., and Wang, W. Learning continuous system dynamics from irregularly-sampled partial observations. *Advances in Neural Information Processing Systems*, 33:16177–16187, 2020. 1

Jung, D. Automated design of grey-box recurrent neural networks for fault diagnosis using structural models and causal information. In *Learning for Dynamics and Control Conference*, pp. 8–20. PMLR, 2022. 8

Khalil, H. K. *Nonlinear Systems*. Prentice Hall, 3rd edition, 2002. 2, 4

Kidger, P., Morrill, J., Foster, J., and Lyons, T. Neural controlled differential equations for irregular time series. In *Advances in Neural Information Processing Systems*, volume 33, pp. 6696–6707, 2020. 1

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2014. 8

Krishnapriyan, A. S., Queiruga, A. F., Erichson, N. B., and Mahoney, M. W. Learning continuous models for continuous physics. *arXiv preprint arXiv:2202.08494*, 2022. 1

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. 7

Le, Q. V., Jaitly, N., and Hinton, G. E. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015. 7

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. 7

LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *Nature*, 521(7553):436–444, 2015. 1

Lipman, Y., Chen, R. T. Q., Ben-Hamu, H., Nickel, M., and Le, M. Flow matching for generative modeling. In *The Eleventh International Conference on Learning Representations*, 2023. 1

Ljung, L. *System identification: Theory for the User*. Prentice Hall, 2nd edition edition, 1999. 1, 2, 3

Lukoševičius, M. and Jaeger, H. Reservoir computing approaches to recurrent neural network training. *Computer science review*, 3(3):127–149, 2009. 2

Massaroli, S., Poli, M., Park, J., Yamashita, A., and Asama, H. Dissecting neural ODEs. In *Advances in Neural Information Processing Systems*, volume 33, pp. 3952–3963, 2020. 1

Mezzadri, F. How to generate random matrices from the classical compact groups. *arXiv preprint math-ph/0609050*, 2006. 5

Mohammadi, A., Westny, T., Jung, D., and Krysander, M. Analysis of numerical integration in RNN-based residuals for fault diagnosis of dynamic systems. *arXiv preprint arXiv:2305.04670*, 2023. 2, 3

Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016. 1

Ott, K., Katiyar, P., Hennig, P., and Tiemann, M. ResNet after all: Neural ODEs and their numerical solution. In *International Conference on Learning Representations*, 2021. 1, 2, 3

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. 5

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, 2019. 4

Queiruga, A. F., Erichson, N. B., Taylor, D., and Mahoney, M. W. Continuous-in-depth neural networks. *arXiv preprint arXiv:2008.02389*, 2020. 1, 2, 3

Rahaman, N., Baratin, A., Arpit, D., Draxler, F., Lin, M., Hamprecht, F., Bengio, Y., and Courville, A. On the spectral bias of neural networks. In *International Conference on Machine Learning*, pp. 5301–5310. PMLR, 2019. 3

Rezende, D. J., Mohamed, S., and Wierstra, D. Stochastic backpropagation and approximate inference in deep generative models. In *International Conference on Machine Learning*, pp. 1278–1286. PMLR, 2014. 8

Rico-Martinez, R. and Kevrekidis, I. G. Continuous time modeling of nonlinear systems: A neural network-based approach. In *IEEE International Conference on Neural Networks*, pp. 1522–1525. IEEE, 1993. 1

Ronneberger, O., Fischer, P., and Brox, T. U-Net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 234–241. Springer, 2015. 8

Rubanova, Y., Chen, R. T., and Duvenaud, D. K. Latent ordinary differential equations for irregularly-sampled time series. In *Advances in Neural Information Processing Systems*, volume 32, 2019. 1

Saxe, A. M., McClelland, J. L., and Ganguli, S. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *International Conference on Learning Representations*, 2014. 5

Stewart, G. W. The efficient generation of random orthogonal matrices with an application to condition estimators. *SIAM Journal on Numerical Analysis*, 17(3):403–409, 1980. 5

Tancik, M., Srinivasan, P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J., and Ng, R. Fourier features let networks learn high frequency functions in low dimensional domains. In *Advances in Neural Information Processing Systems*, volume 33, pp. 7537–7547, 2020. 3

Teerapittayanon, S., McDanel, B., and Kung, H.-T. Branchynet: Fast inference via early exiting from deep neural networks. In *IEEE International Conference on Pattern Recognition*, pp. 2464–2469, 2016. 7

Todorov, E., Erez, T., and Tassa, Y. MuJoCo: A physics engine for model-based control. In *IEEE/RSJ international Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012. 8

Toth, P., Rezende, D. J., Jaegle, A., Racanière, S., Botev, A., and Higgins, I. Hamiltonian generative networks. *International Conference on Learning Representations*, 2020. 7

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017. 1

Verma, Y., Heinonen, M., and Garg, V. ClimODE: Climate forecasting with physics-informed neural ODEs. In *International Conference on Learning Representations*, 2024. 1

Vidulin, V., Lustrek, M., Kaluza, B., Piltaver, R., and Krivec, J. Localization Data for Person Activity. UCI Machine Learning Repository, 2010. DOI: https://doi.org/10.24432/C57G8X. 8

Vito, S. Air Quality. UCI Machine Learning Repository, 2016. DOI: https://doi.org/10.24432/C59K5F. 8

Westny, T., Oskarsson, J., Olofsson, B., and Frisk, E. MTP-GO: Graph-based probabilistic multi-agent trajectory prediction with neural ODEs. *IEEE Transactions on Intelligent Vehicles*, 8(9):4223–4236, 2023. 1

Xia, H., Suliafu, V., Ji, H., Nguyen, T., Bertozzi, A., Osher, S., and Wang, B. Heavy ball neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, volume 34, pp. 18646–18659, 2021. 1

Yan, H., Du, J., Tan, V. Y., and Feng, J. On robustness of neural ordinary differential equations. In *International Conference on Learning Representations*, 2019. 1

Yildiz, C., Heinonen, M., and Lahdesmaki, H. ODE2VAE: Deep generative second order ODEs with Bayesian neural networks. In *Advances in Neural Information Processing Systems*, volume 32, 2019. 1

Zhang, H., Gao, X., Unterman, J., and Arodz, T. Approximation capabilities of neural ODEs and invertible residual networks. In *International Conference on Machine Learning*, pp. 11086–11095. PMLR, 2020. 1

Zhang, T., Yao, Z., Gholami, A., Gonzalez, J. E., Keutzer, K., Mahoney, M. W., and Biros, G. ANODEV2: A coupled neural ODE framework. In *Advances in Neural Information Processing Systems*, volume 32, 2019. 1, 2

Zhong, Y. D., Dey, B., and Chakraborty, A. Benchmarking energy-conserving neural networks for learning dynamics from data. In *Learning for dynamics and control*, pp. 1218–1229. PMLR, 2021. 8

Zhu, A., Jin, P., Zhu, B., and Tang, Y. On numerical integration in neural ordinary differential equations. In *International Conference on Machine Learning*, pp. 27527–27547. PMLR, 2022. 1

Zhuang, J., Dvornek, N., Li, X., Tatikonda, S., Papademetris, X., and Duncan, J. Adaptive checkpoint adjoint method for gradient estimation in neural ODE. In *International Conference on Machine Learning*, pp. 11639–11649. PMLR, 2020. 1, 2