STABILIZING NEURAL ODE NETWORKS WITH STOCHASTICITY

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural Ordinary Differential Equation (Neural ODE) has been proposed as a continuous approximation to the ResNet architecture. Some commonly used regularization mechanisms in discrete neural networks (e.g. dropout, Gaussian noise) are missing in current Neural ODE networks. In this paper, we propose a new continuous neural network framework called Neural Stochastic Differential Equation (Neural SDE) network, which naturally incorporates various commonly used regularization mechanisms based on random noise injection. Our framework can model various types of noise injection frequently used in discrete networks for regularization purpose, such as dropout and additive/multiplicative noise in each block. We provide theoretical analysis explaining the improved robustness of Neural SDE models against input perturbations. Furthermore, we demonstrate that the Neural SDE network can achieve better generalization than the Neural ODE and is more resistant to adversarial and non-adversarial input perturbations.

1 INTRODUCTION

Residual neural networks (ResNet) (He et al., 2016) are composed of multiple residual blocks transforming the hidden states according to:

$$\boldsymbol{h}_{n+1} = \boldsymbol{h}_n + \boldsymbol{f}(\boldsymbol{h}_n; \boldsymbol{w}_n), \tag{1}$$

where $\boldsymbol{h}_n$ is the input to the n-th layer and $\boldsymbol{f}(\boldsymbol{h}_n; \boldsymbol{w}_n)$ is a non-linear function parameterized by $\boldsymbol{w}_n$. Recently, a continuous approximation to the ResNet architecture has been proposed (Chen et al., 2018), where the evolution of the hidden state $\boldsymbol{h}_t$ can be described as a dynamic system obeying the equation:

$$\boldsymbol{h}_t = \boldsymbol{h}_s + \int_s^t \boldsymbol{f}(\boldsymbol{h}_\tau, \tau; \boldsymbol{w})\, \mathrm{d}\tau, \tag{2}$$

where $\boldsymbol{f}(\boldsymbol{h}_\tau, \tau; \boldsymbol{w})$ is the continuous form of the nonlinear function $\boldsymbol{f}(\boldsymbol{h}_n; \boldsymbol{w}_n)$; $\boldsymbol{h}_s$ and $\boldsymbol{h}_t$ are hidden states at two different time $s \neq t$. A standard ODE solver can be used to solve all the hidden states and final states (output from the neural network), starting from an initial state (input to the neural network). The continuous neural network described in (2) exhibits several advantages over its discrete counterpart described in (1), in terms of parameter efficiency, memory efficiency, explicit control of the numerical error of final output, *etc.*

One missing component in the current Neural ODE network is the various regularization mechanisms commonly employed in discrete neural networks. These regularization techniques have been demonstrated to be crucial in reducing generalization errors, and in improving the robustness of neural networks to adversarial attacks. Many of these regularization techniques are based on stochastic noise injection. For instance, dropout (Srivastava et al., 2014) is widely adopted to prevent overfitting; injecting Gaussian random noise during the forward propagation is effective in improving generalization (Bishop, 1995; An, 1996) as well as robustness to adversarial attacks (Liu et al., 2018; Lecuyer et al., 2018). However, these regularization methods in discrete neural networks are not directly applicable to Neural ODE network, because Neural ODE network is a deterministic system.

Our work attempts to incorporate the above-mentioned stochastic noise injection based regularization mechanisms to the current Neural ODE network, to improve the generalization ability and the robustness of the network. We propose a new continuous neural network framework called **Neural**

Stochastic Differential Equation (Neural SDE) network, which models stochastic noise injection by stochastic differential equations (SDE). In this new framework, we can employ existing techniques from the stability theory of SDE to study the robustness of neural networks. Our results provide theoretical insights to understanding why introducing stochasticity during neural network training and testing leads to improved robustness against adversarial attacks. Furthermore, we demonstrate that, by incorporating the noise injection regularization mechanism to the continuous neural network, we can reduce overfitting and achieve lower generalization error. For instance, on the CIFAR-10 dataset, we observe that the new Neural SDE can improve the test accuracy of the Neural ODE from 81.63% to 84.55%, with other factors unchanged. Our contributions can be summarized as follows:

- We propose a new Stochastic Differential Equation (SDE) framework to incorporate randomness in continuous neural networks. The proposed random noise injection can be used as a drop-in component in any continuous neural networks. Our Neural SDE framework can model various types of noises widely used for regularization purpose in discrete networks, such as dropout (Bernoulli type) and Gaussian noise.
- We carry out a theoretical analysis of the stability conditions of the Neural SDE network, to prove that the randomness introduced in the Neural SDE network can stabilize the dynamical system, which helps improve the robustness and generalization ability of the neural network.
- We verify by numerical experiments that stochastic noise injection in the SDE network can successfully regularize the continuous neural network models, and the proposed Neural SDE network achieves better robustness and improves generalization performance.

Notations: Throughout this paper, we use $\boldsymbol{h} \in \mathbb{R}^n$ to denote the hidden states in a neural network, where $\boldsymbol{h}_0 = \boldsymbol{x}$ is the input (also called initial condition) and y is the label. The residual block with parameters $\boldsymbol{w} \in \mathbb{R}^d$ can be written as a nonlinear transform $\boldsymbol{f}(\boldsymbol{h}_\tau, \tau; \boldsymbol{w})$. We assume the integration is always taken from 0 to T. $\mathbf{B}_t \in \mathbb{R}^m$ is m-dimensional Brownian motion. $\boldsymbol{G}(\boldsymbol{h}_\tau, \tau; \boldsymbol{v}) \in \mathbb{R}^{n\times m}$ is the diffusion matrix parameterized by $\boldsymbol{v}$. Unless stated explicitly, we use $\|\cdot\|$ to represent ℓ_2-norm for vector and Frobenius norm for matrix.

2 Related work

Our work is inspired by the success of the recent Neural ODE network, and we seek to improve the generalization and robustness of Neural ODE, by adding regularization mechanisms crucial to the success of discrete networks. Regularization mechanisms such as dropout cannot be easily incorporated in the Neural ODE due to its deterministic nature.

Neural ODE The basic idea of Neural ODE is discussed in the previous section, here we briefly review relevant literature. The idea of formulating ResNet as a dynamic system was discussed in (E, 2017). A framework was proposed to link existing deep architectures with discretized numerical ODE solvers (Lu et al., 2018), and was shown to be parameter efficient. These networks adopt layer-wise architecture – each layer is parameterized by different independent weights. The Neural ODE model (Chen et al., 2018) computes hidden states in a different way: it directly models the dynamics of hidden states by an ODE solver, with the dynamics parameterized by a shared model. A memory efficient approach to compute the gradients by adjoint methods was developed, making it possible to train large, multi-scale generative networks (Ardizzone et al., 2018; Grathwohl et al., 2018). Our work can be regarded as an extension of this framework, with the purpose of incorporating a variety of noise-injection based regularization mechanisms. Stochastic differential equation in the context of neural network has been studied before, focusing either on understanding how dropout shapes the loss landscape (Sun et al., 2018), or on using stochastic differential equation as a universal function approximation tool to learn the solution of high dimensional PDEs (Raissi, 2018). Instead, our work tries to explain why adding random noise boosts the stability of deep neural networks, and demonstrates the improved generalization and robustness.

Noisy Neural Networks Adding random noise to different layers is a technique commonly employed in training neural networks. Dropout (Srivastava et al., 2014) randomly disables some neurons to avoid overfitting, which can be viewed as multiplying hidden states with Bernoulli random variables. Stochastic depth neural network (Huang et al., 2016) randomly drops some residual blocks of residual neural network during training time. Another successful regularization for ResNet is Shake-Shake regularization (Gastaldi, 2017), which sets a binary random variable to randomly switch between two residual blocks during training. More recently, dropblock (Ghiasi et al., 2018) was designed specifically for convolutional layers: unlike dropout, it drops some continuous regions rather than sparse points to hidden states. All of the above regularization techniques are proposed

to improve generalization performance. One common characteristic of them is that *they fix the network during testing time*. There is another line of research that focuses on improving robustness to perturbations/adversarial attacks by noise injection. Among them, random self-ensemble (Liu et al., 2018; Lecuyer et al., 2018) adds Gaussian noise to hidden states during both training and testing time. In training time, it works as a regularizer to prevent overfitting; in testing time, the random noise is also helpful, which will be explained in this paper. Very recently, there are several concurrent works on Neural SDE (Jia & Benson, 2019; Tzen & Raginsky, 2019; Wang et al., 2019). However compared with Jia & Benson (2019) and Tzen & Raginsky (2019), our paper addresses a very different problem, i.e. the robustness of deep random neural network. And while Wang et al. (2019) deals with adversarial robustness (which is also in our scope, but we extend it to *non-adversarial* robustness), their method is still based on adversarial training. In contrast, we are interested in the robustness rooted in the randomness of training and testing.

3 Neural Stochastic Differential Equation

In this section, we first introduce our proposed Neural SDE to improve the robustness of Neural ODE. Informally speaking, Neural SDE can be viewed as using randomness as a drop-in augmentation for Neural ODE, and it can include some widely used randomization layers such as dropout and Gaussian noise layer (Liu et al., 2018). However, solving Neural SDE is non-trivial, we derive the gradients of loss over model weights. Finally we theoretically analyze the stability conditions of Neural SDE.

Before delving into the multi-dimensional SDE, let's first look at a 1-d toy example to see how SDE can solve the instability issue of ODE. Suppose we have a simple SDE, $\mathrm{d}x_t = x_t\,\mathrm{d}t + \sigma x_t\,\mathrm{d}B_t$ with B_t be the standard Brownian motion. We provide a numerical simulation in Figure 1 for x_t with different σ.

When we set $\sigma = 0$, SDE becomes ODE $\mathrm{d}x_t = x_t\,\mathrm{d}t$ and $x_t = c_0 e^t$ where c_0 is an integration constant. If $c_0 \neq 0$ we can see that $x_t \to \pm\infty$. Furthermore, a small perturbation in x_t will be amplified through t. This clearly shows instability of ODE. On the other hand, if we instead make $\sigma > \sqrt{2}$ (the system is SDE), we have $x_t = c_0 \exp\left((1-\sigma^2/2)t + \sigma B_t\right) \overset{\text{a.s.}}{\to} 0$.

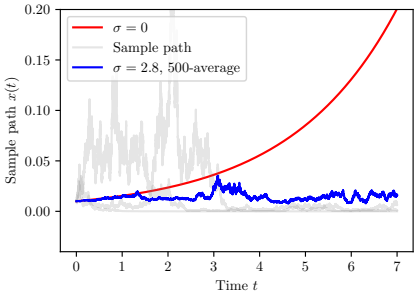

Figure 1: Toy example. By comparing the simulations under $\sigma = 0$ and $\sigma = 2.8$, we see adding noise to the system can be an effective way to control x_t. Average over multiple runs is used to cancel out the volatility during the early stage. It is noteworthy that here we employ the multiplicative noise, where the deviation term scales proportional to x_t.

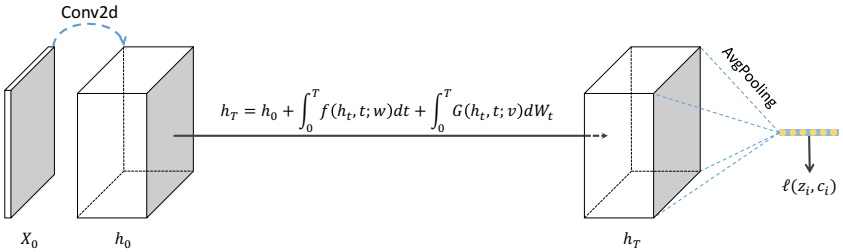

Figure 2: Our model architecture (more details can be found in appendix). The initial value of SDE is the output of a convolutional layer, and the value at time T is passed to a linear classifier after average pooling.

The toy example in Figure 1 reveals that the behavior of solution paths can change significantly after adding a stochastic term. This example is inspiring because we can control the impact of perturbations on the output by adding a stochastic term to neural networks.

Figure 2 shows a sample Neural SDE model architecture, and it is the one used in the experiment. It consists of three parts, the first part is a single convolution block, followed by a Neural SDE network (we will explain the detail of Neural SDE in Section 3.1) and lastly the linear classifier. We put most of the trainable parameters into the second part (Neural SDE), whereas the first/third parts are

mainly for increasing/reducing the dimension as desired. Recall that both Neural ODE and SDE are dimension preserving.

3.1 MODELING RANDOMNESS IN NEURAL NETWORKS

In the Neural ODE system (2), a slightly perturbed input state will be amplified in deep layers (as shown in Figure 1) which makes the system unstable to input perturbation and prone to overfitting. Randomness is an important component in discrete networks (e.g., dropout for regularization) to tackle this issue, however to our knowledge, there is no existing work concerning adding randomness in the continuous neural networks for regularization purpose. And it is non-trivial to encode randomness in continuous neural networks, such as Neural ODE, as we need to consider how to add randomness so that to guarantee the robustness, and how to solve the continuous system efficiently. To solve these challenges, motivated by (Lu et al., 2018; Sun et al., 2018), we propose to add a single diffusion term into Neural ODE as:

$$\mathrm{d}\boldsymbol{h}_t = \boldsymbol{f}(\boldsymbol{h}_t, t; \boldsymbol{w})\,\mathrm{d}t + \boldsymbol{G}(\boldsymbol{h}_t, t; \boldsymbol{v})\,\mathrm{d}\mathbf{B}_t, \tag{3}$$

where $\mathbf{B}_t$ is the standard Brownian motion (Øksendal, 2003), which is a continuous time stochastic process such that $\mathbf{B}_{t+s} - \mathbf{B}_s$ follows Gaussian with mean 0 and variance t; $\boldsymbol{G}(\boldsymbol{h}_t, t; \boldsymbol{v})$ is a transformation parameterized by $\boldsymbol{v}$. This formula is quite general, and can include many existing randomness injection models with residual connections under different forms of $\boldsymbol{G}(\boldsymbol{h}_t, t; \boldsymbol{v})$. As examples, we briefly list some of them below.

Gaussian noise injection: Consider a simple example in (25) when $\boldsymbol{G}(\boldsymbol{h}_t, t; \boldsymbol{v})$ is a diagonal matrix, and we can model both additive and multiplicative noise as

$$\begin{aligned} \text{additive: } \mathrm{d}\boldsymbol{h}_t &= \boldsymbol{f}(\boldsymbol{h}_t, t; \boldsymbol{w})\,\mathrm{d}t + \boldsymbol{\Sigma}(t)\,\mathrm{d}\mathbf{B}_t \\ \text{multiplicative: } \mathrm{d}\boldsymbol{h}_t &= \boldsymbol{f}(\boldsymbol{h}_t, t; \boldsymbol{w})\,\mathrm{d}t + \boldsymbol{\Sigma}(\boldsymbol{h}_t, t)\,\mathrm{d}\mathbf{B}_t, \end{aligned} \tag{4}$$

where $\boldsymbol{\Sigma}(t)$ is a diagonal matrix and its diagonal elements control the variance of the noise added to hidden states. This can be viewed as a continuous approximation of noise injection techniques in discrete neural network. For example, the discrete version of the additive noise can be written as

$$\boldsymbol{h}_{n+1} = \boldsymbol{h}_n + \boldsymbol{f}(\boldsymbol{h}_n; \boldsymbol{w}_n) + \boldsymbol{\Sigma}_n \boldsymbol{z}_n, \quad \text{with } \boldsymbol{\Sigma}_n = \sigma_n \boldsymbol{I},\ \boldsymbol{z}_n \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, 1), \tag{5}$$

which injects Gaussian noise after each residual block. It has been shown that injecting small Gaussian noise can be viewed as a regularization in neural networks (Bishop, 1995; An, 1996). Furthermore, (Liu et al., 2018; Lecuyer et al., 2018) recently showed that adding a slightly larger noise in one or all residual blocks can improve the adversarial robustness of neural networks. We will provide the stability analysis of (25) in Section 3.3, which provides a theoretical explanation towards the robustness of Neural SDE.

Dropout: Our framework can also model the dropout layer which randomly disables some neurons in the residual blocks. Let us see how to unify dropout under our Neural SDE framework. First we notice that in the discrete case

$$\boldsymbol{h}_{n+1} = \boldsymbol{h}_n + \boldsymbol{f}(\boldsymbol{h}_n; \boldsymbol{w}_n) \odot \frac{\boldsymbol{\gamma}_n}{p} = \boldsymbol{h}_n + \boldsymbol{f}(\boldsymbol{h}_n; \boldsymbol{w}_n) + \boldsymbol{f}(\boldsymbol{h}_n; \boldsymbol{w}_n) \odot (\frac{\boldsymbol{\gamma}_n}{p} - \boldsymbol{I}), \tag{6}$$

where $\boldsymbol{\gamma}_n \overset{\text{i.i.d.}}{\sim} \mathcal{B}(1, p)$ and $\odot$ indicates the Hadamard product. Note that we divide γ_n by p in (6) to maintain the same expectation. Furthermore, we have

$$\frac{\boldsymbol{\gamma}_n}{p} - I = \sqrt{\frac{1-p}{p}} \cdot \boxed{\sqrt{\frac{p}{1-p}}\Big(\frac{\gamma_n}{p} - \boldsymbol{I}\Big)} \approx \sqrt{\frac{1-p}{p}}\boldsymbol{z}_n, \quad \boldsymbol{z}_n \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, 1). \tag{7}$$

The boxed part above is approximated by standard normal distribution (by matching the first and second order moment). The final SDE with dropout can be obtained by combining (6) with (7)

$$\mathrm{d}\boldsymbol{h}_t = \boldsymbol{f}(\boldsymbol{h}_t, t; \boldsymbol{w})\,\mathrm{d}t + \sqrt{\frac{1-p}{p}}\boldsymbol{f}(\boldsymbol{h}_t, t; \boldsymbol{w}) \odot \mathrm{d}\mathbf{B}_t. \tag{8}$$

Others: Lu et al. (2018) includes some other stochastic layers that can be formulated under Neural SDE framework, including shake-shake regularization (Gastaldi, 2017) and stochastic depth (Huang et al., 2016). Both of them are used as regularization techniques that work very similar to dropout.

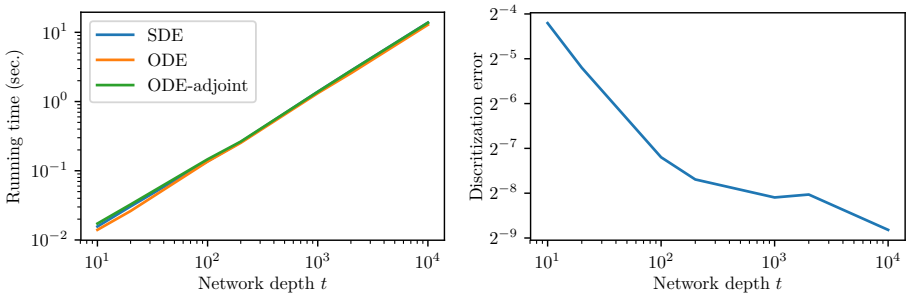

Figure 3: *Left*: we compare the propagation time between Neural ODE, ODE with adjoint, and SDE. We can see that the running time increases proportionally with network depth and there is no significant overhead in neural SDE. *Right*: We compute the error of SDE solver caused by discretization in Euler schemes, measured by the relative error in $\boldsymbol{h}_t$, i.e. $\varepsilon = \frac{\|\boldsymbol{h}_T - \hat{\boldsymbol{h}}_T\|}{\|\boldsymbol{h}_T\|}$ and $\boldsymbol{h}_T$ is the ground-truth (computed with a very fine grid), $\hat{\boldsymbol{h}}_T$ is computed with coarse grid $\Delta t \in [1.0 \times 10^{-4}, 1.0 \times 10^{-1}]$ (note that network depth $t = T/\Delta T$).

3.2 Propagation though SDE Solvers

The overall implementation of training neural network containing SDE solver is similar to Neural ODE (Chen et al., 2018) and it is stated in Algorithm 1. We can see from the algorithm that for forward propagation we apply some standard SDE solvers such as Euler-Maruyama (Kloeden & Platen, 2013), Milstein (Milshtein, 1975) or higher order Runge-Kutta method, but for backward propagation we can simply rely on the automatic gradient. In practice, we find that the most straightforward autograd

Algorithm 1 Forward and backward propagation of Neural SDE

1: **procedure** TRAINING-PROCESS ▷ Do forward & backward propagation
2: Given initial state $\boldsymbol{h}_0$, integral range $[0, T]$.
3: $\boldsymbol{h}_T = \text{SDE\_Solve}\Big(\boldsymbol{f}(\boldsymbol{h}_t, t; \boldsymbol{w}), \boldsymbol{G}(\boldsymbol{h}_t, t; \boldsymbol{v}), [0, T]\Big)$. ▷ Call a black-box SDE solver
4: Calculate loss $L = \ell(\boldsymbol{h}_T)$.
5: Calculate gradient $\frac{\partial L}{\partial \boldsymbol{w}}$ and $\frac{\partial L}{\partial \boldsymbol{v}}$ with autograd.
6: Update network parameters $\boldsymbol{w}$ and $\boldsymbol{v}$.

method works efficiently in our experiments, see Figure 3(left). Furthermore, we find that the discretization error is small enough to perform classification task even for larger grid size in SDE solver (Line 3 in Algorithm 1) as shown in Figure 3(right). More details can be found in Appendix C.

Note that instead of using automatic gradient computation, we can also calculates the gradients by adjoint method, which first transforms the SDE system in (25) into a deterministic PDE through Feynman-Kac formula (Cattiaux & Mesnager, 2002). Due to the space limit we defer it to appendix E. Since a simple implementation with auto-differentiation is already achieving competitive run time and approximation with Neural ODE as shown in Figure 3, we will use that for the experiments throughout this paper.

3.3 Robustness of Neural SDE

In this section, we theoretically analyze the stability of Neural SDE, showing that the randomness term can indeed improve the robustness of the model against small input perturbation. This also explains why noise injection can improve the robustness in discrete networks, which has been observed in literature (Liu et al., 2018; Lecuyer et al., 2018). First we need to show the existence and uniqueness of solution to (25), we pose following assumptions on drift $\boldsymbol{f}$ and diffusion $\boldsymbol{G}$.

Assumption 1. *$\boldsymbol{f}$ and $\boldsymbol{G}$ are at most linear, i.e. $\|\boldsymbol{f}(\boldsymbol{x}, t)\| + \|\boldsymbol{G}(\boldsymbol{x}, t)\| \leq c_1(1 + \|\boldsymbol{x}\|)$ for $c_1 > 0$, $\forall \boldsymbol{x} \in \mathbb{R}^n$ and $t \in \mathbb{R}^+$.*

Assumption 2. *$\boldsymbol{f}$ and $\boldsymbol{G}$ are c_2-Lipschitz: $\|\boldsymbol{f}(\boldsymbol{x}, t) - \boldsymbol{f}(\boldsymbol{y}, t)\| + \|\boldsymbol{G}(\boldsymbol{x}, t) - \boldsymbol{G}(\boldsymbol{y}, t)\| \leq c_2\|\boldsymbol{x} - \boldsymbol{y}\|$ for $c_2 > 0$, $\forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ and $t \in \mathbb{R}^+$.*

Based on the above assumptions, we can show that the SDE (25) has a unique solution (Øksendal, 2003). We remark that assumption on $\boldsymbol{f}$ is quite natural and is also enforced on the original Neural ODE model (Chen et al., 2018); as to diffusion matrix $\boldsymbol{G}$, we have seen that for dropout, Gaussian noise injection and other random models, both assumptions are automatically satisfied as long as $\boldsymbol{f}$ possesses the same regularities.

We analyze the dynamics of perturbation. Our analysis applies not only to the Neural SDE model but also to Neural ODE model, by setting the diffusion term $\boldsymbol{G}$ to zero. First of all, we consider initializing our Neural SDE (25) at two slightly different values $\boldsymbol{h}_0$ and $\boldsymbol{h}_0^e = \boldsymbol{h}_0 + \boldsymbol{\varepsilon}_0$, where $\boldsymbol{\varepsilon}_0$ is the perturbation for $\boldsymbol{h}_0$ with $\|\boldsymbol{\varepsilon}_0\| \leq \delta$. So, under the new perturbed initialization $\boldsymbol{h}_0^e$, the hidden state at time t follows the same SDE in (25),

$$\mathrm{d}\boldsymbol{h}_t^e = \boldsymbol{f}(\boldsymbol{h}_t^e, t; \boldsymbol{w})\,\mathrm{d}t + \boldsymbol{G}(\boldsymbol{h}_t^e, t; \boldsymbol{v})\,\mathrm{d}\mathbf{B}_t', \quad \text{with } \boldsymbol{h}_0^e = \boldsymbol{h}_0 + \boldsymbol{\varepsilon}_0, \tag{9}$$

where $\mathbf{B}_t'$ is Brownian motions for the SDE associated with initialization $\boldsymbol{h}_0^e$. Then it is natural to analyze how the perturbation $\boldsymbol{\varepsilon}_t = \boldsymbol{h}_t^e - \boldsymbol{h}_t$ evolves in the long run. Subtracting (25) from (9)

$$\begin{aligned}\mathrm{d}\boldsymbol{\varepsilon}_t &= \big[\boldsymbol{f}(\boldsymbol{h}_t^e, t; \boldsymbol{w}) - \boldsymbol{f}(\boldsymbol{h}_t, t; \boldsymbol{w})\big]\,\mathrm{d}t + \big[\boldsymbol{G}(\boldsymbol{h}_t^e, t; \boldsymbol{v}) - \boldsymbol{G}(\boldsymbol{h}_t, t; \boldsymbol{v})\big]\,\mathrm{d}\mathbf{B}_t \\ &= \boldsymbol{f}_\Delta(\boldsymbol{\varepsilon}_t, t; \boldsymbol{w})\,\mathrm{d}t + \boldsymbol{G}_\Delta(\boldsymbol{\varepsilon}_t, t; \boldsymbol{v})\,\mathrm{d}\mathbf{B}_t.\end{aligned} \tag{10}$$

Here we made an implicit assumption that the Brownian motions $\mathbf{B}_t$ and $\mathbf{B}_t'$ have the *same* sample path for both initialization $\boldsymbol{h}_0$ and $\boldsymbol{h}_0^e$, i.e. $\mathbf{B}_t = \mathbf{B}_t'$ w.p.1. In other words, we focus on the difference of two random processes $\boldsymbol{h}_t$ and $\boldsymbol{h}_t^e$ driven by the same underlying Brownian motion. So it is valid to subtract the diffusion terms.

An important property of (10) is that it admits a trivial solution $\boldsymbol{\varepsilon}_t \equiv \mathbf{0}$, $\forall t \in \mathbb{R}^+$ and $\boldsymbol{w} \in \mathbb{R}^d$. We show that both the drift ($\boldsymbol{f}$) and diffusion ($\boldsymbol{G}$) are zero under this solution:

$$\begin{aligned}\boldsymbol{f}_\Delta(\mathbf{0}, t; \boldsymbol{w}) &= \boldsymbol{f}(\boldsymbol{h}_t + \mathbf{0}, t; \boldsymbol{w}) - \boldsymbol{f}(\boldsymbol{h}_t, t; \boldsymbol{w}) = 0, \\ \boldsymbol{G}_\Delta(\mathbf{0}, t; \boldsymbol{v}) &= \boldsymbol{G}(\boldsymbol{h}_t + \mathbf{0}, t; \boldsymbol{w}) - \boldsymbol{G}(\boldsymbol{h}_t, t; \boldsymbol{w}) = 0.\end{aligned} \tag{11}$$

The implication of zero solution is clear: for a neural network, if we do not perturb the input data, then the output will never change. However, the solution $\boldsymbol{\varepsilon}_t = \mathbf{0}$ can be *highly unstable*, in the sense that for an arbitrarily small perturbation $\boldsymbol{\varepsilon}_0 \neq \mathbf{0}$ at initialization, the change of output $\boldsymbol{\varepsilon}_T$ can be arbitrarily bad. On the other hand, as shown below, by choosing the diffusion term $\boldsymbol{G}$ properly, we can always control $\boldsymbol{\varepsilon}_t$ within a small range.

In general, we cannot get the closed form solution to a multidimensional SDE but we can still analyze the asymptotic stability through the dynamics $\boldsymbol{f}$ and $\boldsymbol{G}$. This is an extension of Lyapunov stability theory to a stochastic system. First we define the notion of stability in the stochastic case. Let $(\Omega, \mathcal{F}, P)$ be a complete probability space with filtration $\{\mathcal{F}_t\}_{t\geq 0}$ and $\mathbf{B}_t$ be an m-dimensional Brownian motion defined in the probability space, we consider the SDE in (10) with initial value $\boldsymbol{\varepsilon}_0$

$$\mathrm{d}\boldsymbol{\varepsilon}_t = \boldsymbol{f}_\Delta(\boldsymbol{\varepsilon}_t, t)\,\mathrm{d}t + \boldsymbol{G}_\Delta(\boldsymbol{\varepsilon}_t, t)\,\mathrm{d}\mathbf{B}_t, \tag{12}$$

For simplicity we dropped the dependency on parameters $\boldsymbol{w}$ and $\boldsymbol{v}$. We further assume $\boldsymbol{f}_\Delta : \mathbb{R}^n \times \mathbb{R}_+ \mapsto \mathbb{R}^n$ and $\boldsymbol{G}_\Delta : \mathbb{R}^n \times \mathbb{R}_+ \mapsto \mathbb{R}^{n\times m}$ are both Borel measurable. We can show that if assumptions (1) and (2) hold for $\boldsymbol{f}$ and $\boldsymbol{G}$, then they hold for $\boldsymbol{f}_\Delta$ and $\boldsymbol{G}_\Delta$ as well (see Lemma A.2 in Appendix), and we know the SDE (12) allows a unique solution $\boldsymbol{\varepsilon}_t$. We have the following Lynapunov stability results from (Mao, 2007).

Definition 3.1 (Lyapunov stability of SDE)**.** *The solution $\boldsymbol{\varepsilon}_t = \mathbf{0}$ of (12):*

A. is stochastically stable if for any $\alpha \in (0,1)$ and $r > 0$, there exists a $\delta = \delta(\alpha, r) > 0$ such that $\Pr\{\|\boldsymbol{\varepsilon}_t\| < r \text{ for all } t \geq 0\} \geq 1 - \alpha$ whenever $\|\boldsymbol{\varepsilon}_0\| \leq \delta$. Moreover, if for any $\alpha \in (0,1)$, there exists a $\delta = \delta(\alpha) > 0$ such that $\Pr\{\lim_{t\to\infty} \|\boldsymbol{\varepsilon}_t\| = 0\} \geq 1 - \alpha$ whenever $\|\boldsymbol{\varepsilon}_0\| \leq \delta$, it is said to be stochastically asymptotically stable;

B. is almost surely exponentially stable if $\limsup_{t\to\infty} \frac{1}{t} \log \|\boldsymbol{\varepsilon}_t\| < 0$ a.s.[1] for all $\boldsymbol{\varepsilon}_0 \in \mathbb{R}^n$.

Note that for part A in Definition 3.1, it is hard to quantify how well the stability is and how fast the solution reaches equilibrium. In addition, under assumptions (1, 2), we have a straightforward

[1] "a.s." is the abbreviation for "almost surely".

result $\Pr\{\boldsymbol{\varepsilon}_t \neq \mathbf{0} \text{ for all } t \geq 0\} = 1$ whenever $\boldsymbol{\varepsilon}_0 \neq \mathbf{0}$ as shown in Appendix (see Lemma A.3). That is, almost all the sample paths starting from a non-zero initialization can never reach zero due to Brownian motion. On the contrary, the almost sure exponentially stability result implies that almost all the sample paths of the solution will be close to zero exponentially fast. One important result regarding to stability of this system is Mao (2007), deferred to Theorem A.1 in Appendix. We now consider a special case, when the noise is multiplicative $\boldsymbol{G}(\boldsymbol{h}_t, t) = \sigma \cdot \boldsymbol{h}_t$ and $m = 1$. The corresponding SDE of perturbation $\boldsymbol{\varepsilon}_t = \boldsymbol{h}_t^e - \boldsymbol{h}_t$ has the following form

$$\mathrm{d}\boldsymbol{\varepsilon}_t = \boldsymbol{f}_\Delta(\boldsymbol{\varepsilon}_t, t; \boldsymbol{w})\,\mathrm{d}t + \sigma \cdot \boldsymbol{\varepsilon}_t\,\mathrm{d}\mathbf{B}_t. \tag{13}$$

Note that for the deterministic case of (13) by setting $\sigma \equiv 0$, the solution may not be stable in certain cases (see Figure 1). Whereas for general cases when $\sigma > 0$, following corollary claims that by setting σ properly, we will achieve an (almost surely) exponentially stable system.

Corollary 3.0.1. *For* (13)*, if $\boldsymbol{f}(\boldsymbol{h}_t, t; \boldsymbol{w})$ is L-Lipschtiz continuous w.r.t. $\boldsymbol{h}_t$, then* (13) *has a unique solution with the property $\limsup_{t\to\infty} \frac{1}{t}\log\|\boldsymbol{\varepsilon}_t\| \leq -(\frac{\sigma^2}{2} - L)$ almost surely for any $\boldsymbol{\varepsilon}_0 \in \mathbb{R}^n$. In particular, if $\sigma^2 > 2L$, the solution $\boldsymbol{\varepsilon}_t = \mathbf{0}$ is almost surely exponentially stable.*

4 Experimental Results

In this section we show the effectiveness of our Neural SDE framework in terms of generalization, non-adversarial robustness and adversarial robustness. We use the SDE model architecture illustrated in Figure 2 during the experiment. Throughout our experiments, we set $\boldsymbol{f}(\cdot)$ to be a neural network with several convolution blocks. As to $\boldsymbol{G}(\cdot)$ we have the following choices:

- **Neural ODE**, this can be done by dropping the diffusion term $\boldsymbol{G}(\boldsymbol{h}_t, t; \boldsymbol{v}) = \mathbf{0}$.
- **Additive noise**, when the diffusion term is independent of $\boldsymbol{h}_t$, here we simply set it to be diagonal $\boldsymbol{G}(\boldsymbol{h}_t, t; \boldsymbol{v}) = \sigma_t \boldsymbol{I}$.
- **Multiplicative noise**, when the diffusion term is proportional to $\boldsymbol{h}_t$, or $\boldsymbol{G}(\boldsymbol{h}_t, t; \boldsymbol{w}) = \sigma_t \boldsymbol{h}_t$.
- **Dropout noise**, when the diffusion term is proportional to the drift term $\boldsymbol{f}(\boldsymbol{h}_t, t; \boldsymbol{w})$, i.e. $\boldsymbol{G}(\boldsymbol{h}_t, t; \boldsymbol{v}) = \sigma_t \mathrm{diag}\{\boldsymbol{f}(\boldsymbol{h}_t, t; \boldsymbol{w})\}$.

Note the last three are our proposed Neural SDE with different types of randomness as explained in Section 3.1. For more experimental details, such as neural network architecture of $\boldsymbol{f}(\cdot)$ and the solver for Neural SDE, please refer to Appendix B. Note that we use the same architecture for both ODE and SDE, so the comparisons are fair.

4.1 Generalization Performance

In the first experiment, we show small noise helps generalization. However, note that our noise injection is different from randomness layer in the discrete case, for instance, dropout layer adds Bernoulli noise at training time but not testing time; whereas our Neural SDE model keeps randomness at testing time and takes the average prediction of multiple forward propagations.

As for datasets, we choose CIFAR-10, STL-10 and Tiny-ImageNet[2] to include various sizes and number of classes. The experimental results are shown in Table 1. We see that for all datasets, Neural SDE consistently outperforms ODE, and the reason is that adding moderate noise to the models at training time can act as a regularizer and thus improves testing accuracy. Based upon that, if we further keep testing time noise and ensemble the outputs, we will obtain even better results.

Table 1: Evaluating the model generalization under different choices of diffusion matrix $\boldsymbol{G}(\boldsymbol{h}_t, t; \boldsymbol{v})$ introduced above. For the last three noise types, we search a suitable parameter σ_t for each so that the diffusion matrix $\boldsymbol{G}$ properly regularizes the model. TTN means testing time noise. Model size is counted by #parameters.

Data	Model size	Accuracy@1 — w/o TTN				Accuracy@1 — w/ TTN			
		ODE	Additive	Multiplicative	Dropout	ODE	Additive	Multiplicative	Dropout
CIFAR-10	115 K	81.63	83.65	83.26	83.60	–	83.89	83.76	**84.55**
STL-10	2.44 M	58.03	61.23	60.54	61.26	–	62.11	**62.58**	62.13
Tiny-ImageNet	2.49 M	45.19	45.25	46.94	47.04	–	45.39	46.65	**47.81**

[2]Downloaded from `https://tiny-imagenet.herokuapp.com/`

4.2 IMPROVED NON-ADVERSARIAL ROBUSTNESS

In this experiment, we aim at evaluating the robustness of models under non-adversarial corruptions following the idea of (Hendrycks & Dietterich, 2019). The corrupted datasets contain tens of defects in photography including motion blur, Gaussian noise, fog etc. For each noise type, we run Neural ODE and Neural SDE with dropout noise, and gather the testing accuracy. The final results are reported by mean accuracy (mAcc) in Table 2 by changing the level of corruption. Both models are trained on completely clean data, which means the corrupted images are not visible to them during the training stage, nor could they augment the training set with the same types of corruptions. From the table, we can see that Neural SDE performs better than Neural ODE in 8 out of 10 cases. For the rest two, both ODE and SDE are performing very close. This shows that our proposed Neural SDE can improve the robustness of Neural ODE under non-adversarial corrupted data.

Table 2: Testing accuracy results under different levels of non-adversarial perturbations.

Data	Noise type	mild corrupt ← Accuracy → severe corrupt				
		Level 1	Level 2	Level 3	Level 4	Level 5
CIFAR10-C†	ODE	75.89	70.59	66.52	60.91	53.02
	Dropout	77.02	71.58	67.21	61.61	53.81
	Dropout+TTN	**79.07**	**73.98**	**69.74**	**64.19**	**55.99**
TinyImageNet-C†	ODE	23.01	19.18	15.20	**12.20**	**9.88**
	Dropout	22.85	18.94	14.64	11.54	9.09
	Dropout+TTN	**23.84**	**19.89**	**15.28**	12.08	9.44

† Downloaded from `https://github.com/hendrycks/robustness`

4.3 IMPROVED ADVERSARIAL ROBUSTNESS

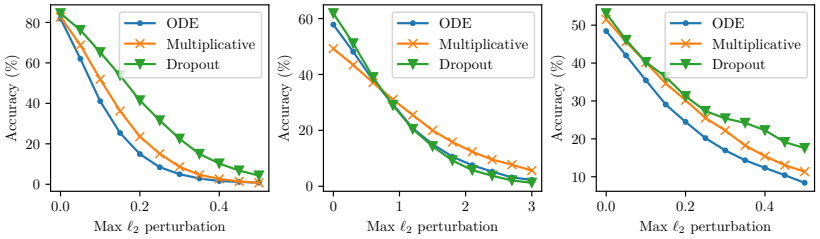

Figure 4: Comparing the robustness against ℓ_2-norm constrained adversarial perturbations, on CIFAR-10 (left), STL-10 (middle) and Tiny-ImageNet (right) data. We evaluate testing accuracy with three models, namely Neural ODE, Neural SDE with multiplicative noise and dropout noise.

Next, we consider the performance of Neural SDE models under adversarial perturbation. Clearly, this scenario is strictly harder than the previous case: by design, the adversarial perturbations are guaranteed to be the worst case within a small neighborhood (ignoring the suboptimality of optimization algorithms) crafted through constrained loss maximization procedure, so it represents the worst case performance. In our experiment, we adopt multi-step ℓ_∞-PGD attack (Madry et al., 2017), although other strong white-box attacks such as C&W (Carlini & Wagner, 2017) are also suitable. The experimental results are shown in Figure 4. As we can see Neural SDE with either multiplicative noise or dropout noise are more resistant to adversarial attack than Neural ODE, and dropout noise outperforms multiplicative noise.

4.4 VISUALIZING THE PERTURBATIONS OF HIDDEN STATES

In this experiment, we take a look at the perturbation $\boldsymbol{\varepsilon}_t = \boldsymbol{h}_t^e - \boldsymbol{h}_t$ at any time t. Recall the 1-d toy example in Figure 1, we observe that the perturbation at time t can be well suppressed by adding a strong diffusion term, which is also confirmed by theorem. However, it is still questionable whether the same phenomenon also exists in deep neural network since we cannot add very large noise to the network during training or testing time. If the noise is too large, it will also remove all useful features. Thus it becomes important to make sure that this will not happen to our models. To this end, we first sample an input $\boldsymbol{x}$ from CIFAR-10 and gather all the hidden states $\boldsymbol{h}_t$ at time $t = [0, \Delta t, 2\Delta t, \ldots, N\Delta t]$. Then we perform regular PGD attack (Madry et al., 2017) to find the

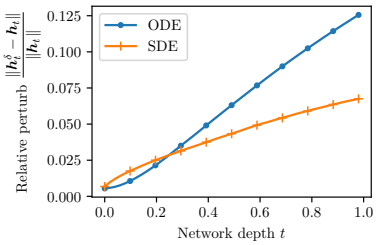

Figure 5: Comparing the perturbations of hidden states, $\boldsymbol{\varepsilon}_t$, on both ODE and SDE (we choose dropout-style noise).

perturbation $\boldsymbol{\delta}_x$ such that $\boldsymbol{x}_{\text{adv}} = \boldsymbol{x} + \boldsymbol{\delta}_x$ is an adversarial image, and feed the new data $\boldsymbol{x}_{\text{adv}}$ into network again so we get $\boldsymbol{h}_t^e$ at the same time stamps as $\boldsymbol{h}_t$. Finally we plot the error $\boldsymbol{\varepsilon}_t = \boldsymbol{h}_t^e - \boldsymbol{h}_t$ *w.r.t.* time t (also called "network depth"), shown in Figure 5. We can observe that by adding a diffusion term (dropout-style noise), the error accumulates much slower than the ordinary Neural ODE model.

5 Conclusion

To conclude, we introduce the Neural SDE model which can stabilize the prediction of Neural ODE by injecting stochastic noise. Our model can achieve better generalization and improve the robustness to both adversarial and non-adversarial noises.

References

Guozhong An. The effects of adding noise during backpropagation training on a generalization performance. *Neural computation*, 8(3):643–674, 1996.

Lynton Ardizzone, Jakob Kruse, Sebastian Wirkert, Daniel Rahner, Eric W Pellegrini, Ralf S Klessen, Lena Maier-Hein, Carsten Rother, and Ullrich Köthe. Analyzing inverse problems with invertible neural networks. *arXiv preprint arXiv:1808.04730*, 2018.

Chris M Bishop. Training with noise is equivalent to tikhonov regularization. *Neural computation*, 7 (1):108–116, 1995.

Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 39–57. IEEE, 2017.

Patrick Cattiaux and Laurent Mesnager. Hypoelliptic non-homogeneous diffusions. *Probability Theory and Related Fields*, 123(4):453–483, 2002.

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pp. 6572–6583, 2018.

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

Xavier Gastaldi. Shake-shake regularization. *arXiv preprint arXiv:1705.07485*, 2017.

Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Dropblock: A regularization method for convolutional networks. In *Advances in Neural Information Processing Systems*, pp. 10727–10737, 2018.

Will Grathwohl, Ricky TQ Chen, Jesse Betterncourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*, 2019.

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European conference on computer vision*, pp. 646–661. Springer, 2016.

Junteng Jia and Austin R. Benson. Neural jump stochastic differential equations, 2019.

Peter E Kloeden and Eckhard Platen. *Numerical solution of stochastic differential equations*, volume 23. Springer Science & Business Media, 2013.

Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. Certified robustness to adversarial examples with differential privacy. *arXiv preprint arXiv:1802.03471*, 2018.

Xuanqing Liu, Minhao Cheng, Huan Zhang, and Cho-Jui Hsieh. Towards robust neural networks via random self-ensemble. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 369–385, 2018.

Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In *International Conference on Machine Learning*, pp. 3282–3291, 2018.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

Xuerong Mao. *Stochastic differential equations and applications*. Elsevier, 2007.

GN Milshtein. Approximate integration of stochastic differential equations. *Theory of Probability & Its Applications*, 19(3):557–562, 1975.

Bernt Øksendal. Stochastic differential equations. In *Stochastic differential equations*, pp. 65–84. Springer, 2003.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Maziar Raissi. Forward-backward stochastic neural networks: Deep learning of high-dimensional partial differential equations. *arXiv preprint arXiv:1804.07010*, 2018.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Qi Sun, Yunzhe Tao, and Qiang Du. Stochastic training of residual networks: a differential equation viewpoint. *arXiv preprint arXiv:1812.00174*, 2018.

Belinda Tzen and Maxim Raginsky. Neural stochastic differential equations: Deep latent gaussian models in the diffusion limit, 2019.

Bao Wang, Binjie Yuan, Zuoqiang Shi, and Stanley J Osher. Enresnet: Resnet ensemble via the feynman-kac formalism. In *Neural Information Processing Systems*, 2019.

Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2921–2929, 2016.

We include the omitted proofs, experiment details and performance analysis here.

APPENDIX A OMITTED THEOREMS AND PROOFS

Theorem A.1. (MAO, 2007) *If there exists a non-negative real valued function $V(\boldsymbol{\varepsilon}, t)$ defined on $\mathbb{R}^n \times \mathbb{R}_+$ that has continuous partial derivatives*

$$V_1(\boldsymbol{\varepsilon}, t) \coloneqq \frac{\partial V(\boldsymbol{\varepsilon}, t)}{\partial \boldsymbol{\varepsilon}}, V_2(\boldsymbol{\varepsilon}, t) \coloneqq \frac{\partial V(\boldsymbol{\varepsilon}, t)}{\partial t}, V_{1,1}(\boldsymbol{\varepsilon}, t) \coloneqq \frac{\partial^2 V(\boldsymbol{\varepsilon}, t)}{\partial \boldsymbol{\varepsilon} \partial \boldsymbol{\varepsilon}^\top}$$

and constants $p > 0, c_1 > 0, c_2 \in \mathbb{R}, c_3 \geq 0$ such that the following inequalities hold:

1. $c_1 \|\boldsymbol{\varepsilon}\|^p \leq V(\boldsymbol{\varepsilon}, t)$
2. $\mathcal{L}V(\boldsymbol{\varepsilon}, t) = V_2(\boldsymbol{\varepsilon}, t) + V_1(\boldsymbol{\varepsilon}, t)\boldsymbol{f}_\Delta(\boldsymbol{\varepsilon}, t) + \frac{1}{2}\mathsf{Tr}[\boldsymbol{G}_\Delta^\top(\boldsymbol{\varepsilon}, t) V_{1,1}(\boldsymbol{\varepsilon}, t)\boldsymbol{G}_\Delta(\boldsymbol{\varepsilon}, t)] \leq c_2 V(\boldsymbol{\varepsilon}, t)$
3. $\|V_1(\boldsymbol{\varepsilon}, t)\boldsymbol{G}_\Delta(\boldsymbol{\varepsilon}, t)\|^2 \geq c_3 V^2(\boldsymbol{\varepsilon}, t)$

for all $\boldsymbol{\varepsilon} \neq \mathbf{0}$ and $t > 0$. Then for all $\boldsymbol{\varepsilon}_0 \in \mathbb{R}^n$,

$$\limsup_{t \to \infty} \frac{1}{t} \log \|\boldsymbol{\varepsilon}_t\| \leq -\frac{c_3 - 2c_2}{2p} \quad a.s. \tag{14}$$

In particular, if $c_3 \geq 2c_2$, the solution $\boldsymbol{\varepsilon}_t \equiv \mathbf{0}$ is almost surely exponentially stable.

We present the proofs of theorems on stability of SDE. The proofs are adapted from (Mao, 2007). We start with two crucial lemmas.

Lemma A.2. *If $\boldsymbol{f}, \boldsymbol{G}$ satisfy Assumption (2), then $\boldsymbol{f}_\Delta, \boldsymbol{G}_\Delta$ satisfy Assumption (1,2).*

Proof. By Assumption (2) on $\boldsymbol{f}, \boldsymbol{G}$, we can obtain that for any $\boldsymbol{\varepsilon}, \tilde{\boldsymbol{\varepsilon}} \in \mathbb{R}^n, t \geq 0$

$$\|\boldsymbol{f}_\Delta(\boldsymbol{\varepsilon}, t)\| + \|\boldsymbol{G}_\Delta(\boldsymbol{\varepsilon}, t)\| \leq c_2 \|\boldsymbol{\varepsilon}\| \leq c_2(1 + \|\boldsymbol{\varepsilon}\|),$$
$$\|\boldsymbol{f}_\Delta(\boldsymbol{\varepsilon}, t) - \boldsymbol{f}_\Delta(\tilde{\boldsymbol{\varepsilon}}, t)\| + \|\boldsymbol{G}_\Delta(\boldsymbol{\varepsilon}, t) - \boldsymbol{G}_\Delta(\tilde{\boldsymbol{\varepsilon}}, t)\| \leq c_2 \|\boldsymbol{\varepsilon} - \tilde{\boldsymbol{\varepsilon}}\|.$$

This guarantees the uniqueness of the solution of (12). □

Lemma A.3. *For* (12)*, whenever $\boldsymbol{\varepsilon}_0 \neq \mathbf{0}$, $\Pr\{\boldsymbol{\varepsilon}_t \neq \mathbf{0}$ for all $t \geq 0\} = 1$.*

Proof. We prove it by contradiction. Let $\tau = \inf\{t \geq 0 : \boldsymbol{\varepsilon}_t = 0\}$. Then if it is not true, there exists some $\boldsymbol{\varepsilon}_0 \neq \mathbf{0}$ such that $\Pr\{\tau < \infty\} > 0$. Therefore, we can find sufficiently large constant $T > 0$ and $\theta > 1$ such that $\Pr(A) \coloneqq \Pr\{\tau < T$ and $|\boldsymbol{\varepsilon}_t| \leq \theta - 1, \ \forall\, 0 \leq t \leq \tau\} > 0$. By Assumption 2 on $\boldsymbol{f}$ and $\boldsymbol{G}$, there exists a positive constant K_θ such that

$$\|\boldsymbol{f}_\Delta(\boldsymbol{\varepsilon}, t)\| + \|\boldsymbol{G}_\Delta(\boldsymbol{\varepsilon}, t)\| \leq K_\theta \|\boldsymbol{\varepsilon}\|, \quad \text{for all } \|\boldsymbol{\varepsilon}\| \leq \theta \text{ and } 0 \leq t \leq T. \tag{15}$$

Let $V(\boldsymbol{\varepsilon}, t) = \|\epsilon\|^{-1}$. Then, for any $0 \leq \|\boldsymbol{\varepsilon}\| \leq \theta$ and $0 \leq t \leq T$, we have

$$\begin{aligned}
\mathcal{L}V(\boldsymbol{\varepsilon}, t) &= -\|\boldsymbol{\varepsilon}\|^{-3} \boldsymbol{\varepsilon}^\top \boldsymbol{f}_\Delta(\boldsymbol{\varepsilon}, t) + \frac{1}{2}\{-\|\boldsymbol{\varepsilon}\|^{-3}\|\boldsymbol{G}_\Delta(\boldsymbol{\varepsilon}, t)\|^2 + 3\|\boldsymbol{\varepsilon}\|^{-5}\|\boldsymbol{\varepsilon}^\top \boldsymbol{G}_\Delta(\boldsymbol{\varepsilon}, t)\|^2\} \\
&\leq \|\boldsymbol{\varepsilon}\|^{-2}\|\boldsymbol{f}_\Delta(\boldsymbol{\varepsilon}, t)\| + \|\boldsymbol{\varepsilon}\|^{-3}\|\boldsymbol{G}_\Delta(\boldsymbol{\varepsilon}, t)\|^2 \\
&\leq K_\theta \|\boldsymbol{\varepsilon}\|^{-1} + K_\theta^2 \|\boldsymbol{\varepsilon}\|^{-1} = K_\theta(1 + K_\theta) V(\boldsymbol{\varepsilon}, t),
\end{aligned} \tag{16}$$

where the first inequality comes from Cauchy-Schwartz and the last one comes from (15). For any $\delta \in (0, \|\boldsymbol{\varepsilon}_0\|)$, we define the stopping time $\tau_\delta \coloneqq \inf\{t \geq 0 : \|\boldsymbol{\varepsilon}_t\| \notin (\delta, \theta)\}$. Let $\nu_\delta = \min\{\tau_\delta, T\}$. By Itô's formula, $\mathbb{E}\left[e^{-K_\theta(1+K_\theta)\nu_\delta} V(\boldsymbol{\varepsilon}_{\nu_\delta}, \nu_\delta)\right]$

$$= V(\boldsymbol{\varepsilon}_0, 0) + \mathbb{E}\int_0^{\nu_\delta} e^{-K_\theta(1+K_\theta)s}\Big[-K_\theta(1+K_\theta)V(\boldsymbol{\varepsilon}_s, s) + \mathcal{L}V(\boldsymbol{\varepsilon}_s, s)\Big]\,\mathrm{d}s \leq \|\boldsymbol{\varepsilon}_0\|^{-1}. \tag{17}$$

Since $\tau_\delta \leq T$ and $\|\boldsymbol{\varepsilon}_{\tau_\delta}\| = \delta$ for any $\omega \in A$, then (17) implies

$$\mathbb{E}\left[e^{-K_\theta(1+K_\theta)T}\delta^{-1}\mathbf{1}_A\right] = \delta^{-1}e^{-K_\theta(1+K_\theta)T}\Pr(A) \leq \|\boldsymbol{\varepsilon}_0\|^{-1}. \tag{18}$$

Thus, $\Pr(A) \leq \delta\|\boldsymbol{\varepsilon}_0\|^{-1}e^{K_\theta(1+K_\theta)T}$. Letting $\delta \to 0$, we obtain $\Pr(A) = 0$, which leads to a contradiction. □

PROOF OF THEOREM A.1

We then prove Theorem A.1. Clearly, (14) holds for $\varepsilon_0 = \mathbf{0}$ since $\varepsilon_t \equiv \mathbf{0}$. For any $\varepsilon_0 \neq \mathbf{0}$, we have $\varepsilon_t \neq \mathbf{0}$ for all $t \geq 0$ almost surely by Lemma A.3. Thus, by applying Itô's formula and condition (2), we can show that for $t \geq 0$,

$$\log V(\varepsilon_t, t) \leq \log V(\varepsilon_0, 0) + c_2 t + M(t) - \frac{1}{2}\int_0^t \frac{|V_1(\varepsilon_s, s)\boldsymbol{G}_\Delta(\varepsilon_s, s)|^2}{V^2(\varepsilon_s, s)}\,\mathrm{d}s. \tag{19}$$

where $M(t) = \int_0^t \frac{V_1(\varepsilon_s,s)\boldsymbol{G}_\Delta(\varepsilon_s,s)}{V(\varepsilon_s,s)}\,\mathrm{d}\mathbf{B}_s$ is a continuous martingale with initial value $M(0) = 0$. By the exponential martingale inequality, for any arbitrary $\alpha \in (0,1)$ and $n = 1, 2, \cdots$, we have

$$\Pr\left\{\sup_{0\leq t\leq n}\left[M(t) - \frac{\alpha}{2}\int_0^t \frac{|V_1(\varepsilon_s, s)\boldsymbol{G}_\Delta(\varepsilon_s, s)|^2}{V^2(\varepsilon_s, s)}\,\mathrm{d}s\right] > \frac{2}{\alpha}\log n\right\} \leq \frac{1}{n^2}. \tag{20}$$

Applying Borel-Cantelli lemma, we can get that for almost all $\omega \in \Omega$, there exists an integer $n_0 = n_0(\omega)$ such that if $n \geq n_0$,

$$M(t) \leq \frac{2}{\alpha}\log n + \frac{\alpha}{2}\int_0^t \frac{|V_1(\varepsilon_s, s)\boldsymbol{G}_\Delta(\varepsilon_s, s)|^2}{V^2(\varepsilon_s, s)}\,\mathrm{d}s, \quad \forall\, 0 \leq t \leq n. \tag{21}$$

Combining (19), (21) and condition (3), we can obtain that

$$\log V(\varepsilon_t, t) \leq \log V(\varepsilon_0, 0) - \frac{1}{2}[(1-\alpha)c_3 - 2c_2]t + \frac{2}{\alpha}\log n. \tag{22}$$

for all $0 \leq t \leq n$ and $n \geq n_0$ almost surely. Therefore, for almost all $\omega \in \Omega$, if $n - 1 \leq t \leq n$ and $n \geq n_0$, we have

$$\frac{1}{t}\log V(\varepsilon_t, t) \leq -\frac{1}{2}[(1-\alpha)c_3 - 2c_2] + \frac{\log V(\varepsilon_0, 0) + \frac{2}{\alpha}\log n}{n-1} \tag{23}$$

which consequently implies

$$\limsup_{t\to\infty}\frac{1}{t}\log V(\varepsilon_t, t) \leq -\frac{1}{2}[(1-\alpha)c_3 - 2c_2]) \quad a.s. \tag{24}$$

With condition (1) and arbitrary choice of $\alpha \in (0,1)$, we can obtain (14).

PROOF OF COROLLARY 3.0.1

We apply Theorem A.1 to establish the theories on stability of (13). Note that $\boldsymbol{f}(\boldsymbol{h}_t, t; \boldsymbol{w})$ is L-Lipschitz continuous w.r.t $\boldsymbol{h}_t$ and $\boldsymbol{G}(\boldsymbol{h}_t, t; \boldsymbol{v}) = \sigma\boldsymbol{h}_t, m = 1$. Then, (13) has a unique solution, with $\boldsymbol{f}_\Delta$ and $\boldsymbol{G}_\Delta$ satisfying Assumptions (1,2)

$$\|\boldsymbol{f}_\Delta(\varepsilon_t, t)\| + \|\boldsymbol{G}_\Delta(\varepsilon_t, t)\| \leq \max\{L, \sigma\}\|\varepsilon_t\| \leq \max\{L, \sigma\}(1 + \|\varepsilon_t\|),$$
$$\|\boldsymbol{f}_\Delta(\varepsilon_t, t) - \boldsymbol{f}_\Delta(\tilde{\varepsilon}_t, t)\| + \|\boldsymbol{G}_\Delta(\varepsilon_t, t) - \boldsymbol{G}_\Delta(\tilde{\varepsilon}_t, t)\| \leq \max\{L, \sigma\}\|\varepsilon_t - \tilde{\varepsilon}_t\|.$$

To apply Theorem A.1, let $V(\varepsilon, t) = \|\varepsilon\|^2$. Then,

$$\mathcal{L}V(\varepsilon, t) = 2\varepsilon^\top \boldsymbol{f}_\Delta(\varepsilon, t) + \sigma^2\|\varepsilon\|^2 \leq (2L + \sigma^2)\|\varepsilon\|^2 = (2L + \sigma^2)V(\varepsilon, t),$$
$$\|V_1(\varepsilon, t)\boldsymbol{G}_\Delta(\varepsilon, t)\|^2 = 4\sigma^2 V(\varepsilon, t)^2.$$

Let $c_1 = 1, p = 2, c_2 = 2L + \sigma^2, c_3 = 4\sigma^2$. By Theorem A.1, we finished the proof.

APPENDIX B EXPERIMENT SETTINGS

We have experimented several numerical solver for stochastic differential equations, and finally decided to adopt the most straightforward Euler scheme. Although higher order solvers would also work, we find low order solver is fast and precise enough. We follow the idea in Neural ODE (Chen et al., 2018) and divide the whole classifier into three parts, the first part is to increase the number of channels to a suitable value (which can also be regarded as feature extraction for neural SDE); the following part the the ODE/SDE solver, note that the shape of intermediate states are not changed throughout. The last layer is for classification.

The overview of our model architecture is described in Figure 2. Here we list some key hyper-parameters for each model in Table 3. We can see that the architectures are roughly the same, except that for Tiny-ImageNet, our model is significantly larger due to that fact that this data is significantly harder to train on.

Dataset	First block	SDE block	Last block
MNIST	$\begin{bmatrix}\text{Conv2d}(1,64,3,1)\end{bmatrix}\times 1$	$\begin{bmatrix}\text{GroupNorm}(32,64)\\ \text{Conv2d}(64,64,3,1,1)\\ \text{ReLU}\end{bmatrix}\times 3$	$\begin{bmatrix}\text{GroupNorm}(32,64)\\ \text{ReLU}\\ \text{GAP}\\ \text{Linear}(64,10)\end{bmatrix}\times 1$
CIFAR-10	$\begin{bmatrix}\text{Conv2d}(3,64,3,1)\end{bmatrix}\times 1$	$\begin{bmatrix}\text{GroupNorm}(32,64)\\ \text{Conv2d}(64,64,3,1,1)\\ \text{ReLU}\end{bmatrix}\times 3$	$\begin{bmatrix}\text{GroupNorm}(32,64)\\ \text{ReLU}\\ \text{GAP}\\ \text{Linear}(64,10)\end{bmatrix}\times 1$
Tiny-ImageNet	$\begin{bmatrix}\text{Conv2d}(3,64,3,1,1)\\ \text{GroupNorm}(32,64)\\ \text{ReLU}\\ \text{Conv2d}(64,128,4,2,1)\\ \text{GroupNorm}(32,128)\\ \text{ReLU}\\ \text{Conv2d}(128,256,4,2,1)\end{bmatrix}\times 1$	$\begin{bmatrix}\text{GroupNorm}(32,256)\\ \text{Conv2d}(256,256,3,1,1)\\ \text{ReLU}\end{bmatrix}\times 3$	$\begin{bmatrix}\text{GroupNorm}(32,256)\\ \text{ReLU}\\ \text{GAP}\\ \text{Linear}(256,200)\end{bmatrix}\times 1$

Table 3: Model hyper-parameters. We follow the parameter convention in PyTorch (Paszke et al., 2017). "GAP" means global average pooling Zhou et al. (2016).

APPENDIX C SOME EMPIRICAL ANALYSIS

We provide some extra experiments to examine the discretization error due to Euler scheme. Different from traditional weak and strong convergence analysis of SDE solver, here we only need to care about the error in mean values, i.e. $\|\mathbb{E}\boldsymbol{X}_t - \mathbb{E}\bar{\boldsymbol{X}}_t\|$, since in our case only the results will be first be averaged before linear classifier and the accuracy of classification should not be affected as long as the mean values are precise enough. To verify that, we run our neural SDE model under different discretization step, specifically $\Delta t = \{1.0\times 10^{-1}, 5.0\times 10^{-2}, 1.0\times 10^{-2}, 5.0\times 10^{-3}, 1.0\times 10^{-3}, 5.0\times 10^{-4}, 1.0\times 10^{-4}\}$ and because we cannot solve the equation in closed form, we choose the result by $\Delta t = 1.0\times 10^{-5}$ as the ground truth. For each step size, we solve the SDE 1000 times independently and average the resulting image embedding vectors. The discritization error is measured by the relative error in the sense of Euclidean norm: $\|\boldsymbol{a}-\boldsymbol{b}\|_2/\|\boldsymbol{b}\|_2$. The results are shown in Figure 6. We can observe that although finer step size leads to smaller discretization error, even a coarse step $\Delta t = 0.1$ with relative error $\sim 2^{-4}$ can hardly change the prediction results.

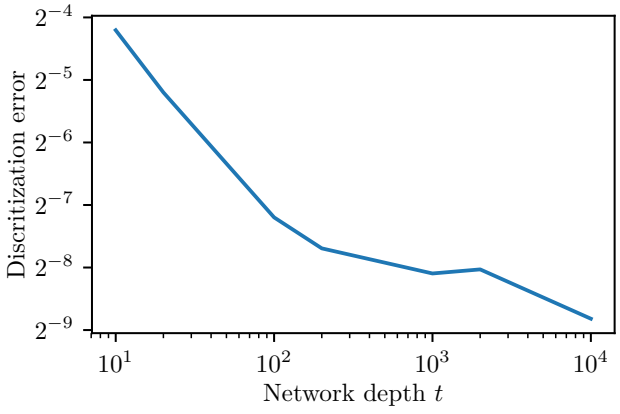

Figure 6: Discretization error under different step size in SDE solver.

APPENDIX D RUNNING TIME COMPARISON: NEURAL SDE, NEURAL ODE, AND NEURAL ODE-ADJOINT

See Figure 7.

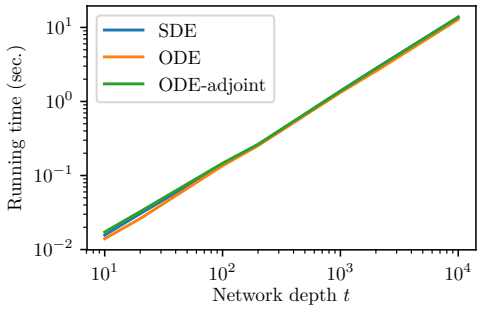

Figure 7: Running time comparison (forward propagation) between Neural SDE, Neural ODE and Neural ODE - adjoint. The curves are largely overlapped, meaning all methods have running time proportional to network depth.

APPENDIX E TRANSFORM STOCHASTIC ORDINARY DIFFERENTIAL EQUATION TO DETERMINISTIC PARTIAL DIFFERENTIAL EQUATION, AND ITS GRADIENT COMPUTATION THROUGH THE ADJOINT STATE METHOD

The d-dimensional stochastic differential equation (SDE) discussed in section 3.1 is as follows:

$$\mathrm{d}\boldsymbol{h}_t = \boldsymbol{f}(\boldsymbol{h}_t, t; \boldsymbol{w})\,\mathrm{d}t + \boldsymbol{G}(\boldsymbol{h}_t, t)\,\mathrm{d}\mathbf{B}_t, \tag{25}$$

Note that $\boldsymbol{w}$ are the parameters of our neural network. Our new loss function for the stochastic neural network is the expectation of the original loss function (e.g. softmax) L acting on the end time state $\boldsymbol{h}_T$ (counterpart to the last layer of discrete neural network), conditioned on initial state $\boldsymbol{h}_0$, or intermediate state $\boldsymbol{h}_t$. Let's denote this expectation as $u(\boldsymbol{h}_t, t)$

$$u(\boldsymbol{h}_t, t) = E(L(\boldsymbol{h}_T)|\boldsymbol{h}_t). \tag{26}$$

Under smoothness assumptions on $\boldsymbol{f}(\cdot)$ and $\boldsymbol{G}(\cdot)$ in equation (25), from Feynman–Kac formula (Cattiaux & Mesnager, 2002), $u(\boldsymbol{h}_t, t)$ satisfies:

$$\frac{\partial u}{\partial t} + \Sigma_{i=1}^{d} \boldsymbol{f}_i(\boldsymbol{h}, t; \boldsymbol{w}) \frac{\partial u}{\partial \boldsymbol{h}_i} + \frac{1}{2} \Sigma_{i,j=1}^{d} [\boldsymbol{G}\boldsymbol{G}^\top]_{i,j} \frac{\partial^2 u}{\partial \boldsymbol{h}_i \partial \boldsymbol{h}_j} = 0 \tag{27}$$

with the boundary condition (final condition):

$$u(\boldsymbol{h}, T) = L(\boldsymbol{h}_T) \tag{28}$$

Our objective is to find $\boldsymbol{w}$ that minimize

$$J(\boldsymbol{h}_0, \boldsymbol{w}) = E(L(\boldsymbol{h}_T)|\boldsymbol{h}_0; \boldsymbol{w}) = u(\boldsymbol{h}_0, 0; \boldsymbol{w}) \tag{29}$$

with $u(\boldsymbol{h}, t; \boldsymbol{w})$ satisfies the PDE system specified in equation(27).

This optimization problem can be solve with the adjoint state method, with the forward state solved from the PDE and boundary condition defined in equation(27). It can be derived that the adjoint state $a(\boldsymbol{h}, t)$ and the gradient $\frac{\partial J}{\partial \boldsymbol{w}}$ can be solved by the following equations:
The adjoint state satisfies the following system:

$$\frac{\partial a}{\partial t} + \Sigma_{i=1}^{d} \frac{\partial}{\partial \boldsymbol{h}_i}[f_i(\boldsymbol{h}, t; \boldsymbol{w})a] - \frac{1}{2} \Sigma_{i,j=1}^{d} \frac{\partial^2}{\partial \boldsymbol{h}_i \partial \boldsymbol{h}_j}[[\boldsymbol{G}\boldsymbol{G}^\top]_{i,j} a] = 0 \tag{30}$$

with the boundary condition (initial condition):

$$a(\boldsymbol{h}, 0) = \delta(\boldsymbol{h}_0) \tag{31}$$

$\delta(\boldsymbol{h}_0)$ is the Dirac delta function which concentrates at $\boldsymbol{h}_0$.

The gradient can be written as:

$$\frac{\partial J}{\partial \boldsymbol{w}} = \langle a, \frac{\partial f}{\partial \boldsymbol{w}} \frac{\partial u}{\partial \boldsymbol{h}} \rangle_{\boldsymbol{h},t} \tag{32}$$

$\langle \cdot, \cdot \rangle_{\boldsymbol{h},t}$ denotes integration over both $\boldsymbol{h}$ and t.

Equation(27) and equation(30) are actually the Kolmogorov backward equation and Kolmogorov forward equation respectively, which are well-known equations in the mathematical finance community. While the Kolmogorov backward equation describes the evolution of the expectation of loss with time, the Kolmogorov forward equation describes the evoluation of a probability distribution.