How Does Noise Help Robustness? Explanation and Exploration under the Neural SDE Framework

Xuanqing Liu
UCLA
xqliu@cs.ucla.edu

Tesi Xiao
UC Davis
texiao@ucdavis.edu

Si Si
Google
sisidaisy@google.com

Qin Cao
Google
qincao@google.com

Sanjiv Kumar
Google
sanjivk@google.com

Cho-Jui Hsieh
UCLA, Google
chohsieh@cs.ucla.edu

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), which naturally incorporates various commonly used regularization mechanisms based on random noise injection. For regularization purposes, our framework includes multiple types of noise patterns, such as dropout, additive, and multiplicative noise, which are common in plain neural networks. We provide some theoretical analyses explaining the improved robustness of our 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

Despite the superhuman performance in many computer vision tasks, recent findings [2, 8, 28] demonstrate that deep neural networks remain to be more fragile than human or even shallow models. Existing work support such phenomenon from different perspectives; for instance, on CIFAR-10 and ImageNet, [25] shows that the test accuracy drops by $5\% - 15\%$ if we replace the original test set by a new one. This experiment casts doubts on the brittleness of generalization and implies that current classifiers are very sensitive to minutiae of data cleaning process. Even on the same test set, unnoticeable adversarial perturbations crafted by specific algorithms [19] can make the test accuracy close to zero. In the less challenging non-adversarial case, [11] collects tens of different types of perturbations and corruptions, including motion blur or frog, to large scale image dataset; they found that test accuracy drops considerably on corrupted images. We summarize three settings above as generalization, adversarial robustness, and non-adversarial robustness. Ideally, we desire a model not only to be distributionally robust but also be resistant against adversarial and non-adversarial perturbations. Unfortunately, previous works only focus on one of these problems – *e.g.* they may apply adversarial training technique [8, 19] to tackle adversarial examples but fall short of clean accuracy when no perturbation exists. It is thus very interesting to see whether there is a unified way to mitigate all the problems, and whether we can find a theoretical explanation for it.

In this paper, we study the role of randomness for training a robust neural network. There are abundant sources of randomness: 1) dropout layer [26] randomly disable some neural connections and set the corresponding hidden features to zero; 2) similarly, drop block [7] selects a dense, rectangular region to zero; 3) stochastic depth network [12] removes some residual blocks as a whole during the training stage; and 4) random smoothing [4] adds *i.i.d.* Gaussian noise to input images to be resistant to adversarial perturbations. As we will see later, all of these ideas are mostly the same, which are composed of the deterministic part – a neural network, and the stochastic part – Bernoulli or Gaussian random variables.

To study and understand how randomness stabilizes neural networks, we propose a new continuous neural network framework called Neural Stochastic Differential Equation (Neural SDE), which models the continuous limits of ResNet based on the recent proposed Neural ODE model [3] and adds stochastic diffusion and jump terms to cover various commonly used regularization mechanisms based on random noise, including Dropout, stochastic depth and Gaussian smoothing. Inside the Neural SDE model,

there are drift term – the deterministic part of our network; the diffusion term – the stochastic part driven by multi-dimensional Brownian motions; and the jump term – the stochastic part driven by Poisson process. Diffusion term is best suited for **small but persistent** changes to hidden features, such as Gaussian smoothing after each residual block. Jump term works better for **strong and sparse** strikes to hidden features, examples include dropout/drop block layer and stochastic depth network. Based on the formulation, we draw a theoretical connection between the robustness of neural networks to the stability of the solution. Furthermore, we manage to get the stability condition in the scope of neural network.

In addition to the theoretical contribution, we also show that it's possible to directly train the Neural SDE model efficiently by numerically solving the SDE system. With the added stochastic regularizations, the proposed continuous neural network outperforms the Neural ODE network in terms of generalizability, adversarial robustness, and non-adversarial robustness on CIFAR10 (or CIFAR10.1 [24]), STL10, as well as Tiny-ImageNet datasets.

2. Related work

Our work is inspired by the success of the Neural ODE network, and we seek to improve the generalization and robustness of Neural ODE by adding noise in the dynamic system. Regularization mechanisms such as dropout cannot be easily incorporated in the original Neural ODE due to its deterministic nature.

Neural ODE The idea of formulating ResNet as a dynamic system was discussed in [5]. A framework was proposed to link existing deep architectures with discretized numerical ODE solvers [18], 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 [3] 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 gradient by adjoint methods was developed, making it possible to train large, multi-scale generative networks [1, 9]. 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 (SDE) in the context of neural network has been studied recently, focusing either on understanding how dropout shapes the loss landscape [27], or on using SDE as a universal function approximation tool to learn the solution of high dimensional PDEs [23]. Instead, we aim to explain why adding random noise boosts the stability of deep 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 [26] randomly disables some neurons to avoid overfitting, which can be viewed as multiplying hidden states with Bernoulli random variables. Stochastic depth neural network [12] randomly drops some residual blocks of residual neural network during training time. Another successful regularization for ResNet is Shake-Shake regularization [6], which sets a binary random variable to randomly switch between two residual blocks during training. More recently, dropblock [7] 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 [17, 16] 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 work on stochastic version of Neural ODE [13, 29, 30]. However compared with [13] and [29], our paper addresses a very different problem, i.e. the robustness of deep random neural network. And while [30] 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 how and why random noise in training and testing phases improves robustness.

3. Neural SDE model

Traditional neural networks are usually stacked with multiple layers; recent work [3] shows that we can model it in the continuous limit. This means that there is no notion of discrete layers, and hidden features are changed smoothly. Mathematically it has the following form

$$\boldsymbol{h}_t = \boldsymbol{h}_s + \int_s^t \boldsymbol{f}(\boldsymbol{h}_\tau, \tau; \boldsymbol{w})\, \mathrm{d}\tau, \tag{1}$$

where $t > s$ are two different "depths"; $\boldsymbol{h}_t$ is the hidden features at depth t; $\boldsymbol{f}$ is the residual block parameterized by $\boldsymbol{w}$. This formula is exactly the continuous limit of the original ResNet [10] structure

$$\boldsymbol{h}_{n+1} = \boldsymbol{h}_n + \boldsymbol{f}(\boldsymbol{h}_n; \boldsymbol{w}_n), \tag{2}$$

here the layer index $n = 1, 2, \ldots, N$ is discrete. Notice that the original Neural ODE model does not contain any randomness in hidden features $\boldsymbol{h}_t$. Thus it is not ready to model

a variety of random neural networks (such as dropout). To address this limitation, we augment the original Neural ODE model (1) with two kinds of stochastic terms: one is the diffusion term (to model Gaussian noise), and the other is jump term (to model Bernoulli noise), formally

$$\boldsymbol{h}_t = \boldsymbol{h}_s + \underbrace{\int_s^t \boldsymbol{f}(\boldsymbol{h}_\tau, \tau; \boldsymbol{w})\,\mathrm{d}\tau}_{\text{drift term}} + \underbrace{\int_s^t \boldsymbol{G}(\boldsymbol{h}_\tau, \tau)\,\mathrm{d}\mathbf{B}_\tau}_{\text{diffusion term}} + \underbrace{\int_s^t \boldsymbol{J}(\boldsymbol{h}_\tau, \tau) \odot \mathbf{Z}_{N_\tau}\,\mathrm{d}N_\tau}_{\text{jump term}}. \tag{3}$$

Compared with Neural ODE model in (1) that only contains deterministic component (drift term), we add two extra terms in (3) to model different nature of randomness: diffusion term and jump term. The diffusion term consists of Brownian motion $\mathbf{B}_t$ and its coefficient $\boldsymbol{G}$ (optionally) parameterized by unknown variables $\boldsymbol{v}$. Inside the jump term, deterministic function $\boldsymbol{J}(\boldsymbol{h}_\tau, \tau)$ controls the jump size; Random variables $\mathbf{Z}_{N_\tau} \sim \text{Bernoulli}(\pm 1, p)$ controls the direction; and $N_\tau \sim \text{Poisson}(\lambda\tau)$ is a Poisson counting process controlling the "frequency" of jumps. For completeness, we include some key properties of Brownian motion to appendix, and for more systematic discussion we refer readers to related sections in [22, 14], informally we can regard the random variable $\mathrm{d}\mathbf{B}_t$ as i.i.d. Gaussian random variables with distribution $\mathcal{N}(0, \mathrm{d}t)$. Next we will explain these two terms in details.

Diffusion term. This part is an Itô integral and we know it follows Gaussian distribution. To see it more clearly, we can simply set $\boldsymbol{G}(\boldsymbol{h}_\tau, \tau) = \sigma$ and so the result of integration will be $\mathbf{B}_t - \mathbf{B}_s \sim \mathcal{N}(0, (t-s)\sigma^2)$, which is consistent with adding Gaussian noise to each residual block. For general $\boldsymbol{G}$, it does allow closed-form solution but the result is still Gaussian, only the variance is now dependent on hidden features $\boldsymbol{h}_\tau$.

Jump term. The key feature of jump term is that the integral is calculated over Poisson process N_t, that is, the total number of jumps in interval $[s, t]$ follows Poisson distribution

$$P(N_{s\to t} = n) = \frac{[\lambda(t-s)]^n}{n!} e^{-\lambda(t-s)}, \quad n \in \mathbb{Z}_+.$$

We can imagine that by inserting the jump term to our hidden state transition formula (3), we are effectively adding n dropout layers to the network, where n is drawn from some Poisson distribution; and for each dropout layer, it is randomly placed to any network depth (see Fig. 1 to get a better picture). Additionally, for each dropout layer, the drop probability is determined by Bernoulli random variables $\mathbf{Z}$.

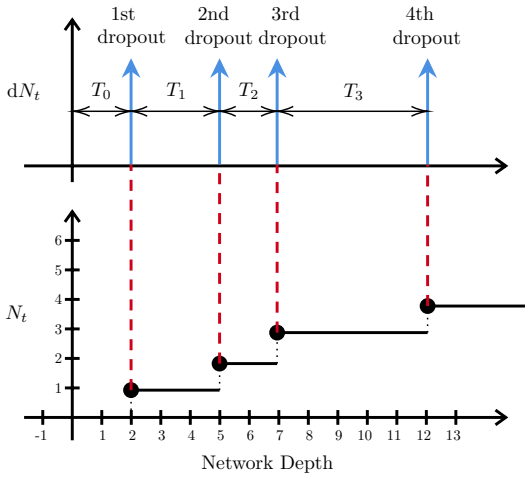

Figure 1. Illustrateion of dropout layers under (3).

Rationale of (3). It might not be straightforward to see why our model (3) is a proper replacement for the discrete version of a residual neural network equipped with Gaussian smoothing and Dropout layers. Here we make more justifications about it. We first notice that the noise pattern coming from dropout layers is very different from the noise generated by Gaussian smoothing. By definition, Dropout randomly sets some features to zero, so the noise here is inherently Bernoulli distributed. On the other hand, the diffusion term is a Gaussian process (Itô integral). Therefore, it is not reasonable to model a dropout layer with diffusion term, nor is it suitable to model Gaussian noise with jump term. That is why we use two separate terms in our continuous framework.

3.1. Some concrete examples

We proposed a new framework in (3) for encoding the randomness into Neural ODE using diffusion and jump terms. Next we will give some concrete examples for these two terms.

Dropout. Dropout layer randomly disables some connections in neural network, here we consider a common situation where dropout layer is placed after convolutional block and before residual connection (Fig. 2). Mathematically we

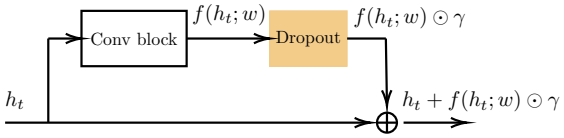

Figure 2. Our dropout layer configuration.

can formulate it as $\boldsymbol{h}_{t+1} = \boldsymbol{h}_t + \boldsymbol{f}(\boldsymbol{h}_t; w) \odot \boldsymbol{\gamma}$, where $\boldsymbol{h}_t$

is input features at depth τ, and $P(\boldsymbol{\gamma}_i = 0) = p_{\text{drop}}$ determines the drop rate. To be compatible with (3), we rewrite it as

$$\begin{aligned}\boldsymbol{h}_{t+1} &= \boldsymbol{h}_t + \boldsymbol{f}(\boldsymbol{h}_t; w) \odot \boldsymbol{\gamma} \\ &= \boldsymbol{h}_t + \frac{1}{2}\boldsymbol{f}(\boldsymbol{h}_t; w) + 2\boldsymbol{f}(\boldsymbol{h}_t; w) \odot \frac{\boldsymbol{\gamma} - 0.5}{2}, \\ &= \boldsymbol{h}_t + \frac{1}{2}\boldsymbol{f}(\boldsymbol{h}_t; w) + 2\boldsymbol{f}(\boldsymbol{h}_t; w) \odot \mathbf{Z}\end{aligned} \tag{4}$$

where we are essentially shifting Bernoulli random variables from $\boldsymbol{\gamma}$ (with values $\{0, 1\}$) to $\mathbf{Z}$ (with values $\{-1, +1\}$). Comparing (4) with (3), we arrive at the following continuous version of (4)

$$\boldsymbol{h}_t = \boldsymbol{h}_s + \int_s^t \frac{1}{2}\boldsymbol{f}(\boldsymbol{h}_\tau; w)\,\mathrm{d}\tau + \int_s^t 2\boldsymbol{f}(\boldsymbol{h}_\tau; w) \odot \mathbf{Z}_{N_\tau}\,\mathrm{d}N_\tau.$$

Stochastic depth network. This is very similar to the previous dropout setting, the only difference is that for stochastic depth network, random vector $\boldsymbol{\gamma}$ is no longer i.i.d. Bernoulli distributed but "bonded" together. More formally, $\boldsymbol{\gamma} = \gamma \cdot \boldsymbol{J}$ where $\boldsymbol{J}_{i,j} = 1$ is an all-ones matrix and γ is a (scalar) Bernoulli random variable (of values $\{0, 1\}$). Beyond that, there is no difference with previous dropout layer, and the continuous form of it is

$$\boldsymbol{h}_t = \boldsymbol{h}_s + \int_s^t \frac{1}{2}\boldsymbol{f}(\boldsymbol{h}_\tau; w)\,\mathrm{d}\tau + \int_s^t 2\boldsymbol{f}(\boldsymbol{h}_\tau; w) Z_{N_\tau}\,\mathrm{d}N_\tau, \tag{5}$$

here Z_{N_τ} is just a scalar random variable.

Gaussian-dropout. We can create another kind of dropout noise that does not involve Bernoulli random variables. We first scale the original dropout (4) by $1 - p_{\text{drop}}$, which becomes

$$\boldsymbol{h}_{t+1} = \boldsymbol{h}_t + \boldsymbol{f}(\boldsymbol{h}_t; w) \odot \frac{\boldsymbol{\gamma}}{1 - p_{\text{drop}}}. \tag{6}$$

The reason we add an $1 - p_{\text{drop}}$ scaling factor is that now the output expectation $\mathbb{E}[\boldsymbol{h}_{t+1}] = \boldsymbol{h}_t + \boldsymbol{f}(\boldsymbol{h}_t; w)$ looks as if no dropout is used, due to the fact that $\mathbb{E}[\gamma] = 1 - p_{\text{drop}}$. Disentangling the mean from variance, we have

$$\begin{aligned}\boldsymbol{h}_{t+1} &= \boldsymbol{h}_t + \boldsymbol{f}(\boldsymbol{h}_t; w) + \boldsymbol{f}(\boldsymbol{h}_t; w) \odot \left(\frac{\boldsymbol{\gamma}}{1 - p_{\text{drop}}} - \boldsymbol{I}\right) \\ &= \boldsymbol{h}_t + \boldsymbol{f}(\boldsymbol{h}_t; w) + \sqrt{\frac{p_{\text{drop}}}{1 - p_{\text{drop}}}}\boldsymbol{f}(\boldsymbol{h}_t; w) \odot \boldsymbol{z}_t,\end{aligned} \tag{7}$$

where $\boldsymbol{I}$ is the identity matrix, and $\boldsymbol{z}_t \triangleq \sqrt{\frac{1-p_{\text{drop}}}{p_{\text{drop}}}}\left(\frac{\boldsymbol{\gamma}}{1-p_{\text{drop}}} - \boldsymbol{I}\right)$. We can verify that $\boldsymbol{z}_t$ as a two-point distribution, has the same mean and variance as standard Gaussian distribution. So as an approximation, we directly replace $\boldsymbol{z}_t$ with $\mathcal{N}(0, 1)$. After that, the continuous version becomes

$$\boldsymbol{h}_t = \boldsymbol{h}_s + \int_s^t \boldsymbol{f}(\boldsymbol{h}_\tau; w)\,\mathrm{d}\tau + \int_s^t \sqrt{\frac{p_{\text{drop}}}{1 - p_{\text{drop}}}}\boldsymbol{f}(\boldsymbol{h}_t; w) \odot \mathrm{d}\mathbf{B}_\tau.$$

Gaussian smoothing. As we have mentioned before, Gaussian noise is better modeled by diffusion term. The traditional way of applying Gaussian smoothing is adding small, uncorrelated noise to each hidden layer [17] (or just the input layer [16, 4]), mathematically

$$\boldsymbol{h}_{t+1} = \boldsymbol{h}_t + \boldsymbol{f}(\boldsymbol{h}_t; w) + \mathbf{W}_t, \quad \mathbf{W}_t \sim \mathcal{N}(\mathbf{0}, \sigma^2\boldsymbol{I}). \tag{8}$$

But through experiments we found that multiplicative noise of following form also works

$$\boldsymbol{h}_{t+1} = \boldsymbol{h}_t + \boldsymbol{f}(\boldsymbol{h}_t; w) + \boldsymbol{f}(\boldsymbol{h}_t; w)\mathbf{W}_t, \quad \mathbf{W}_t \sim \mathcal{N}(\mathbf{0}, \sigma^2\boldsymbol{I}). \tag{9}$$

Unlike (8), the noise scale in (9) grows with the scale of output from convolution block $\boldsymbol{f}(\boldsymbol{h}_t; w)$, thus the noise variance is self-adjustable and sometimes it can be advantageous. For both additive and multiplicative Gaussian noise, the integral form is as straightforward as follows

$$\boldsymbol{h}_t = \boldsymbol{h}_s + \int_s^t \boldsymbol{f}(\boldsymbol{h}_\tau; w)\,\mathrm{d}\tau + \begin{cases}\int_s^t \boldsymbol{G}(\tau)\,\mathrm{d}\mathbf{B}_\tau, \\ \int_s^t \boldsymbol{G}(\boldsymbol{h}_t, \tau)\,\mathrm{d}\mathbf{B}_\tau.\end{cases} \tag{10}$$

As we can see, for additive noise, diffusion coefficient $\boldsymbol{G}$ is independent on $\boldsymbol{h}_t$; while for multiplicative noise, $\boldsymbol{G}$ can change along with hidden features $\boldsymbol{h}_t$.

3.2. Training algorithm and complexity

The implementation of the stochastic, continuous neural network training algorithm is similar to Neural ODE [3] and it is stated in Algorithm 1. In fact, we can view (3) as a SDE problem, and standard SDE solvers can be applied here for training. We can see from the algorithm that for forward propagation we pick a standard SDE solvers such as Euler-Maruyama [15], Milstein [21] or higher order Runge-Kutta method, but for backward propagation we simply rely on the automatic gradient provided by major deep learning frameworks. Although it is possible to derive

Algorithm 1 Forward and backward propagation

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

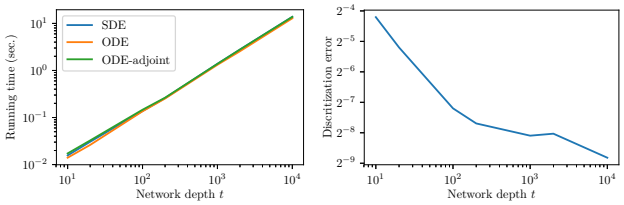

Figure 3. *Left*: we compare the propagation time between Neural ODE, ODE with adjoint, and SDE. We can see that the running time increases proportionally with network depth and there is no significant overhead in our model. *Right*: We compute the error of SDE solver caused by discretization in Euler 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 $\Delta t \in [1.0\times10^{-4}, 1.0\times10^{-1}]$ (note that network depth $t = T/\Delta T$).

an adjoint algorithm as in [3] to save memory consumption, in practice, we find that the most straightforward autograd method works efficiently in all our experiments, see Fig. 3(left). Furthermore, we observe the error caused by discretization is small enough for end tasks even when using a large grid size in SDE solver (Line 3 in Algorithm 1) as shown in Fig 3(right). More details are in the appendix.

4. Connection between jump-diffusion and robustness

In this section, we build a new explanation to shed some light on answering how noise (inside jump-diffusion term (3)) helps training the robust neural network. It is worth noting that our analysis is very different from the traditional belief that noise acts as a regularizer during training. We focus on the role of randomness in testing time, and in this sense, our idea is complementary to the former. In the following parts, we first deliver a toy example to have a closer view of this phenomenon, and then we present some theories to understand the principles in behind.

4.1. A toy example

Let's look at a 1-dimensional toy example where randomness stabilizes the system. Suppose we have a simple SDE

$$\mathrm{d}x_t = x_t\,\mathrm{d}t + \sigma x_t\,\mathrm{d}\mathbf{B}_t, \tag{11}$$

with $\mathbf{B}_t$ being the standard Brownian motion. When we remove the diffusion term by setting $\sigma = 0$, (11) becomes an ODE: $\mathrm{d}x_t = x_t\,\mathrm{d}t$ with solution $x_t = x_0 e^t$, where x_0 is the initialization of x_t. If $x_0 \neq 0$, we can see that $x_t \to \pm\infty$ as $t \to \infty$. In other words, any small perturbation at initialization $x_0 = \epsilon$ will be amplified through the ODE-system at future time t. In contrast, if we add the diffusion back $\sigma \neq 0$, we then have the classic geometric Brownian motion with solution $x_t = x_0 \exp\left((1-\sigma^2/2)t + \sigma B_t\right)$. Once the variance of noise is large enough (e.g. $\sigma > \sqrt{2}$), then we know that $x_t \overset{\text{a.s.}}{\to} 0$.

To visualize the difference, we run several numerical simulations in Fig. 4 for x_t with different variances σ. The

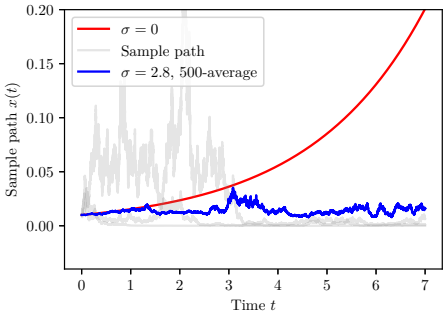

Figure 4. Toy example. By comparing the simulations under $\sigma = 0$ and $\sigma = 2.8$, we see adding noise to the system can be an effective way to control x_t. Average over multiple runs is used to cancel out the volatility during the early stage. It is noteworthy that here we employ the multiplicative noise, where the deviation term scales proportionally to x_t.

experiments in Fig. 4 clearly show that the behavior of solution paths can change significantly after adding a diffusion term. This example is inspiring because we can control the impact of perturbations on the output by adding a stochastic term to our networks.

4.2. Theoretical explanation

Inspired by the toy example above, we theoretically analyze the stability of Neural jump-diffusion equation (3). Our analytical results show that the jump-diffusion terms can indeed improve the robustness of the model against small, arbitrary input perturbations. This finding also explains why noise injection can improve both generalizability and robustness in discrete networks, which has been observed in current literature [17, 16]. To simplify our symbols, we ignore the jump term temporarily and focus on the diffusion term. The following assumptions on drift $\boldsymbol{f}$ and diffusion $\boldsymbol{G}$ guarantee the existence and uniqueness of solution.

Assumption 1 *$\boldsymbol{f}$ and $\boldsymbol{G}$ are at most linear, i.e. $\|\boldsymbol{f}(\boldsymbol{x},t)\| + \|\boldsymbol{G}(\boldsymbol{x},t)\| \le c_1(1+\|\boldsymbol{x}\|)$ 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: $\|\boldsymbol{f}(\boldsymbol{x},t) - \boldsymbol{f}(\boldsymbol{y},t)\| + \|\boldsymbol{G}(\boldsymbol{x},t) - \boldsymbol{G}(\boldsymbol{y},t)\| \le c_2\|\boldsymbol{x}-\boldsymbol{y}\|$ 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 (3) has a unique solution [22]. We remark that these assumptions on $\boldsymbol{f}$ are quite natural and are also enforced on the original Neural ODE model (see Sec. 6 of [3]). As to the diffusion matrix $\boldsymbol{G}$, we have seen that at least for additive Gaussian noise (where $\boldsymbol{G}$ is a constant matrix) and multiplicative Gaussian noise (where $\boldsymbol{G}$ is proportional to $\boldsymbol{f}$), 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 the Neural ODE model, by setting the diffusion term $\boldsymbol{G}$ and jump term $\boldsymbol{J}$ to zero. Our idea is illustrated in Fig. 5. First

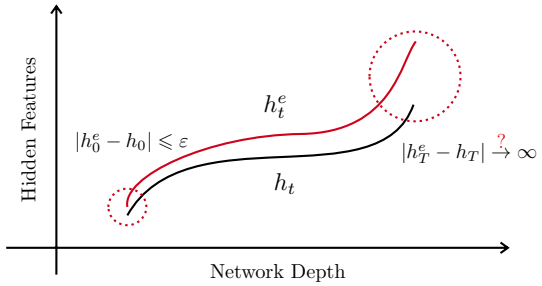

Figure 5. Illustration of our analysis. Given a smaller perturbation ε at the input, how does the error propagation through a deep neural network? If the error is controllable, then we can make sure that the final prediction result is also controllable. In our analysis, we do not need to care about how $\boldsymbol{h}_t$ or $\boldsymbol{h}_t^e$ evolves, only the difference $\boldsymbol{\varepsilon}_t = \boldsymbol{h}_t^e - \boldsymbol{h}_t$ matters; and this is depicted in (13).

of all, we consider initializing our differential equation (3) at two slightly different values $\boldsymbol{h}_0$ and $\boldsymbol{h}_0^e = \boldsymbol{h}_0 + \boldsymbol{\varepsilon}_0$, where $\boldsymbol{h}_0$ is the original (clean) input, $\boldsymbol{\varepsilon}_0$ is the perturbation (also called "error") on $\boldsymbol{h}_0$. In many real problems, the perturbation at input is bounded, i.e. $\|\boldsymbol{\varepsilon}_0\| \leq \delta$. Therefore, under the perturbed initialization $\boldsymbol{h}_0^e$, the hidden states at time t follow the same rule in (3), recalling the jump term is ignored for simplicity,

$$\mathrm{d}\boldsymbol{h}_t^e = \boldsymbol{f}(\boldsymbol{h}_t^e, t; \boldsymbol{w})\,\mathrm{d}t + \boldsymbol{G}(\boldsymbol{h}_t^e, t)\,\mathrm{d}\mathbf{B}_t', \text{with } \boldsymbol{h}_0^e = \boldsymbol{h}_0 + \boldsymbol{\varepsilon}_0, \tag{12}$$

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 (3) from (12), we have

$$\begin{aligned} \mathrm{d}\boldsymbol{\varepsilon}_t &= \big[\boldsymbol{f}(\boldsymbol{h}_t^e, t; \boldsymbol{w}) - \boldsymbol{f}(\boldsymbol{h}_t, t; \boldsymbol{w})\big]\,\mathrm{d}t \\ &\quad + \big[\boldsymbol{G}(\boldsymbol{h}_t^e, t) - \boldsymbol{G}(\boldsymbol{h}_t, t)\big]\,\mathrm{d}\mathbf{B}_t \\ &= \boldsymbol{f}_\Delta(\boldsymbol{\varepsilon}_t, t; \boldsymbol{w})\,\mathrm{d}t + \boldsymbol{G}_\Delta(\boldsymbol{\varepsilon}_t, t)\,\mathrm{d}\mathbf{B}_t. \end{aligned} \tag{13}$$

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 (13) is that it admits a trivial solution $\boldsymbol{\varepsilon}_t \equiv \mathbf{0}, \forall t \in \mathbb{R}^+$ and $\boldsymbol{w} \in \mathbb{R}^d$. To verify that, we only need to show that both the drift ($\boldsymbol{f}$) and diffusion ($\boldsymbol{G}$) are zero under $\boldsymbol{\varepsilon}_t = 0$:

$$\begin{aligned} \boldsymbol{f}_\Delta(\mathbf{0}, t; \boldsymbol{w}) &= \boldsymbol{f}(\boldsymbol{h}_t + \mathbf{0}, t; \boldsymbol{w}) - \boldsymbol{f}(\boldsymbol{h}_t, t; \boldsymbol{w}) = 0, \\ \boldsymbol{G}_\Delta(\mathbf{0}, t) &= \boldsymbol{G}(\boldsymbol{h}_t + \mathbf{0}, t) - \boldsymbol{G}(\boldsymbol{h}_t, t) = 0. \end{aligned} \tag{14}$$

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 large. Luckily, as we will show 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 the 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 (13) with initial value $\boldsymbol{\varepsilon}_0$

$$\mathrm{d}\boldsymbol{\varepsilon}_t = \boldsymbol{f}_\Delta(\boldsymbol{\varepsilon}_t, t)\,\mathrm{d}t + \boldsymbol{G}_\Delta(\boldsymbol{\varepsilon}_t, t)\,\mathrm{d}\mathbf{B}_t, \tag{15}$$

where 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 appendix), and we know the SDE (15) allows a unique solution $\boldsymbol{\varepsilon}_t$. We have the following Lynapunov stability results from [20].

Definition 4.1 (Lyapunov stability of SDE) *The solution $\boldsymbol{\varepsilon}_t = \mathbf{0}$ of (15):*

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 for all $\boldsymbol{\varepsilon}_0 \in \mathbb{R}^n$, $\limsup_{t\to\infty} \frac{1}{t} \log \|\boldsymbol{\varepsilon}_t\| < 0$ a.s.*[1]

Note that for part A in Definition 4.1, 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 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. 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 [20], deferred to appendix. We now consider a special case, when the noise is

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

Table 1. Evaluating the model generalization under different choices of diffusion matrix $\boldsymbol{G}(\boldsymbol{h}_t, t; \boldsymbol{v})$ introduced above. For the three noise types, we search a suitable parameter σ_t for each of them so that the diffusion matrix $\boldsymbol{G}$ properly regularizes the model. TTN means testing time noise. We observe adding noises can improve the test accuracy over Neural ODE, and furthermore, noise at testing time is beneficial.

Data	Accuracy@1 — w/o TTN				Accuracy@1 — w/ TTN			
	ODE	Additive	Multiplicative	Dropout	ODE	Additive	Multiplicative	Dropout
CIFAR-10	87.95	88.69	89.06	88.23	–	88.73	**89.77**	88.44
CIFAR-10.1	70.00	70.80	71.50	71.85	–	71.70	72.05	**73.60**
STL-10	58.03	61.23	60.54	61.26	–	62.11	**62.58**	62.13
Tiny-ImageNet	45.19	45.25	46.94	47.04	–	45.39	46.65	**47.81**

multiplicative $\boldsymbol{G}(\boldsymbol{h}_t, t) = \sigma \cdot \boldsymbol{h}_t$ and $m = 1$. The corresponding SDE of perturbation $\boldsymbol{\varepsilon}_t = \boldsymbol{h}_t^e - \boldsymbol{h}_t$ becomes

$$\mathrm{d}\boldsymbol{\varepsilon}_t = \boldsymbol{f}_\Delta(\boldsymbol{\varepsilon}_t, t; \boldsymbol{w})\,\mathrm{d}t + \sigma \cdot \boldsymbol{\varepsilon}_t\,\mathrm{d}\mathbf{B}_t. \tag{16}$$

Note that for the deterministic case of (16) by setting $\sigma \equiv 0$, the solution may not be stable in certain cases (see Figure 4). 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 4.0.1 *For (16), if $\boldsymbol{f}(\boldsymbol{h}_t, t; \boldsymbol{w})$ is L-Lipschtiz continuous w.r.t. $\boldsymbol{h}_t$, then (16) 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.*

5. Experiment

In this section, we show the effectiveness of our framework in terms of generalization, non-adversarial robustness, and adversarial robustness. 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 $\boldsymbol{G}(\boldsymbol{h}_t, t) = \mathbf{0}$.
- **Additive noise**, when the diffusion term is independent of $\boldsymbol{h}_t$, here we simply set it to be diagonal $\boldsymbol{G}(\boldsymbol{h}_t, t) = \sigma_t \boldsymbol{I}$.
- **Multiplicative noise**, when the diffusion term is proportional to $\boldsymbol{h}_t$, or $\boldsymbol{G}(\boldsymbol{h}_t, t) = \sigma_t \boldsymbol{h}_t$.
- **Gaussian-dropout noise**, when the diffusion term is proportional to the drift term $\boldsymbol{f}(\boldsymbol{h}_t, t; \boldsymbol{w})$, i.e. $\boldsymbol{G}(\boldsymbol{h}_t, t) = \sigma_t \mathrm{diag}\{\boldsymbol{f}(\boldsymbol{h}_t, t; \boldsymbol{w})\}$.

Note the last three are our proposed model with different types of randomness, as explained in Section 3.1. For more experimental details, the architecture of $\boldsymbol{f}(\cdot)$ and the numerical solver for SDE, please refer to our appendix. Note that we use the same architecture for all the methods mentioned above, so the comparisons are fair.

5.1. Generalization Performance

In the first experiment, we show a small noise helps generalization. However, note that our noise injection is different from the randomness layer in the discrete case. For instance, dropout layers add Bernoulli noise at training time but not testing time, whereas our model keeps randomness at the testing time and takes the averaged 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 observe that for all datasets, noisy versions consistently outperform 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.

5.2. Improved non-adversarial robustness

We evaluate the robustness of models under non-adversarial corruption following the idea of [11]. 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 our model 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 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 corruption. From the table, we can see that our model 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 jump-diffusion improves the robustness of Neural ODE under non-adversarial corrupted data.

5.3. Improved adversarial robustness

Next, we consider the performance of our models under adversarial perturbation. This scenario is strictly harder than the previous case – perturbations are crafted through

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

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

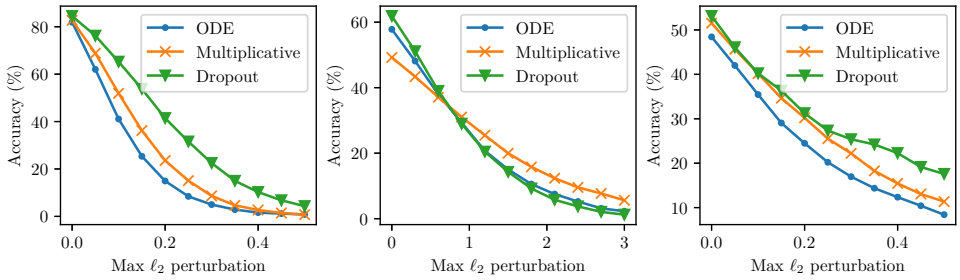

Figure 6. Comparing the robustness against ℓ_2-norm constrained adversarial perturbations, on CIFAR-10 (left), STL-10 (middle) and Tiny-ImageNet (right) data. We observe that jump-diffusion model with either multiplicative noise or dropout noise is more resistant to adversarial attack than Neural ODE.

constrained loss maximization procedure, so it represents the worst-case performance. In our experiment, we adopt ℓ_2-PGD attack with 20 steps [19]. The experimental results are shown in Figure 6. As we can see, jump-diffusion model with either multiplicative noise or dropout noise is more resistant to adversarial attack than Neural ODE. We also observe dropout noise outperforms multiplicative noise.

5.4. Visualizing the perturbations of hidden states

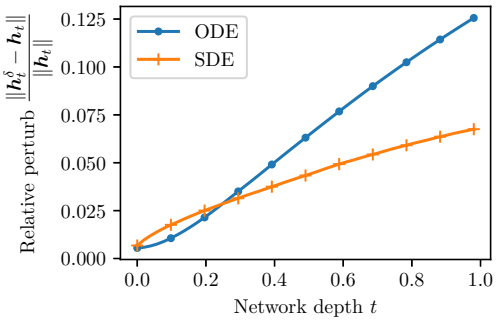

Figure 7. Comparing the perturbations of hidden states, ε_t, on both ODE and SDE (we choose dropout-style noise).

In this experiment, we take a look at the perturbation $\varepsilon_t = h_t^e - h_t$ at any time t. Recall that in the 1-d toy example in Figure 4, we observe that the perturbation at time t can be well suppressed by adding a strong diffusion term, which is also confirmed by our theorem. However, it is still questionable whether the same phenomenon also exists in deep neural networks 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 x from CIFAR-10 and gather all the hidden states h_t at time $t = [0, \Delta t, 2\Delta t, \ldots, N\Delta t]$. Then we perform regular PGD attack [19] to find the perturbation δ_x such that $x_{\text{adv}} = x + \delta_x$ is an adversarial image, and feed the new data x_{adv} into network again so we get h_t^e at the same time stamps as h_t. Finally we plot the error $\varepsilon_t = h_t^e - h_t$ *w.r.t.* time t (also called "network depth"), shown in Figure 7. We can observe that by adding a diffusion term (dropout-style noise), the error accumulates much slower than the ordinary Neural ODE model.

6. 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 under both adversarial and non-adversarial noises.

Acknowledgement This work is partially supported by NSF under IIS1719097.

References

[1] 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. 2

[2] Anurag Arnab, Ondrej Miksik, and Philip H. S. Torr. On the robustness of semantic segmentation models to adversarial attacks. In *CVPR*, 2018. 1

[3] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pages 6572–6583, 2018. 1, 2, 4, 5

[4] Jeremy M Cohen, Elan Rosenfeld, and J Zico Kolter. Certified adversarial robustness via randomized smoothing. *arXiv preprint arXiv:1902.02918*, 2019. 1, 4

[5] Weinan E. A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 5(1):1–11, 2017. 2

[6] Xavier Gastaldi. Shake-shake regularization. *arXiv preprint arXiv:1705.07485*, 2017. 2

[7] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Dropblock: A regularization method for convolutional networks. In *Advances in Neural Information Processing Systems*, pages 10727–10737, 2018. 1, 2

[8] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014. 1

[9] 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. 2

[10] 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*, pages 770–778, 2016. 2

[11] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*, 2019. 1, 7

[12] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European conference on computer vision*, pages 646–661. Springer, 2016. 1, 2

[13] Junteng Jia and Austin R. Benson. Neural jump stochastic differential equations, 2019. 2

[14] Ioannis Karatzas and Steven E Shreve. Brownian motion. In *Brownian Motion and Stochastic Calculus*, pages 47–127. Springer, 1998. 3

[15] Peter E Kloeden and Eckhard Platen. *Numerical solution of stochastic differential equations*, volume 23. Springer Science & Business Media, 2013. 4

[16] 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. 2, 4, 5

[17] 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)*, pages 369–385, 2018. 2, 4, 5

[18] 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*, pages 3282–3291, 2018. 2

[19] 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. 1, 8

[20] Xuerong Mao. *Stochastic differential equations and applications*. Elsevier, 2007. 6

[21] GN Mil'shtein. Approximate integration of stochastic differential equations. *Theory of Probability & Its Applications*, 19(3):557–562, 1975. 4

[22] Bernt Øksendal. Stochastic differential equations. In *Stochastic differential equations*, pages 65–84. Springer, 2003. 3, 5

[23] Maziar Raissi. Forward-backward stochastic neural networks: Deep learning of high-dimensional partial differential equations. *arXiv preprint arXiv:1804.07010*, 2018. 2

[24] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do cifar-10 classifiers generalize to cifar-10? 2018. https://arxiv.org/abs/1806.00451. 2

[25] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do imagenet classifiers generalize to imagenet? *arXiv preprint arXiv:1902.10811*, 2019. 1

[26] 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. 1, 2

[27] Qi Sun, Yunzhe Tao, and Qiang Du. Stochastic training of residual networks: a differential equation viewpoint. *arXiv preprint arXiv:1812.00174*, 2018. 2

[28] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013. 1

[29] Belinda Tzen and Maxim Raginsky. Neural stochastic differential equations: Deep latent gaussian models in the diffusion limit, 2019. 2

[30] Bao Wang, Binjie Yuan, Zuoqiang Shi, and Stanley J Osher. Enresnet: Resnet ensemble via the feynman-kac formalism. In *Neural Information Processing Systems*, 2019. 2