Neural Ordinary Differential Equations: From Image Classification to Generative Modeling

Prateek Garg
20D070060

Sameep Chattopadhyay
20D070067

Vedang Gupta
200100166

***Abstract*—This report explores the theory and versatile application of Neural Ordinary Differential Equations [1] (Neural ODEs), a significantly under-explored architecture for data classification and generative modeling. We have leveraged the continuous-depth models and investigated their efficacy in classifying images while showcasing the remarkable performance comparable to traditional deep networks such as ResNets. Furthermore, we explore different ODE solvers to improve the model performance. Lastly, this research extends Neural ODEs to function as Variational Autoencoders (VAEs), demonstrating their capacity to learn rich latent representations and act as synthetic data generators. By harnessing continuous-depth models' inherent adaptability and expressive power, this study illuminates their potential for advancing both supervised and unsupervised learning methods.**

I. Introduction

Differential equations serve as a cornerstone for understanding continuous and sequential data in various domains like physics, healthcare, and finance. The dynamic nature of such data, constantly evolving, presents a challenge in effectively modeling its behavior. Conversely, discrete sequential data, as observed in Natural Language Processing (NLP), transitions discretely between states, emphasizing a different nature of information processing.

Presently, mainstream methodologies like transformers or recurrent neural networks dominate the landscape for handling these distinct data types. However, these methods treat continuous and discrete sequential data similarly despite their fundamental differences, raising questions about their optimal treatment.

In a pivotal paper [1] introduced in 2018, an innovative solution emerged to bridge this gap – the concept of Neural Ordinary Differential Equations (Neural ODEs). These Neural ODEs present a promising avenue for handling continuous sequential data by viewing it through the lens of differential equations, potentially revolutionizing the way we approach modeling dynamic systems and discrete sequential data in neural networks. Thus we move ahead and see how these concepts pertaining to ODEs are utilized for classification and generative modeling.

II. Problem Statement

We start our discussion by looking at a setup consisting of a first-order ODE and some (noisy) observations along its trajectory-

$$\frac{dy}{dt} = f(y(t), t) - \text{observations}$$

$$\{(y_0, t_0), (y_1, t_1), ..., (y_M, t_M)\} - \text{observations}$$

A general solution to the above (ODE, observations) pair can be represented by a function $z : I \times \mathbb{R}^n \to \mathbb{R}^n$. The existence and uniqueness of solutions to an IVP is ensured, provided the RHS of the ODE is *Lipschitz continuous*, i.e.,

$$||f(x_1) - f(x_2)|| \leq \lambda ||x_1 - x_2|| \quad \forall x_1, x_2 \in X.$$

where the function f is of the form $f : X \subset \mathbb{R}^n \to \mathbb{R}^n$ A variant of the above problem is known as the initial value problem where only a single observation (y_0, t_0) is known, and now we attempt to find an approximation $\widehat{f}(y, t, \theta)$ of dynamics function $f(y, t)$, and thus obtain the value of $y(t_1)$ given $y(t_0)$.

For the above problem, one starts the system's evolution from y_0, t_0 for time $t_1 - t_0$ with some parameterized dynamics function using any ODE initial value solver such as Euler [2] & Runge-Kutta (RK) method [3]. After that, one ends up at some new state $\hat{y_1}, t_1$, compares it with the observation y_1, and tries to minimize the difference by varying the parameters θ. More formally, it can be stated as the minimization of the following loss function-

$$L(y(t_1)) = L\Big(\int_{t_0}^{t_1} f(y(t), t, \theta) dt\Big) = L\big(\text{ODESolve}(y(t_0), f, t_0, t_1, \theta)\big)$$

To optimize L, one needs to compute the gradients wrt. its parameters: $y(t_0), t_0, t_1, \theta$. Hence, we first determine a quantity $a(t)$, known as the adjoint and representing the dependence between the loss and state at every time instance:

$$a(t) = -\frac{\partial L}{\partial y(t)}$$

$$\frac{da(t)}{dt} = -a(t)\frac{\partial f(y(t), t, \theta)}{\partial y}$$

The adjoint can be thought of as an instantaneous analog of the chain rule. As derived in the original Neural ODE paper [1], the adjoint and its derivative can be used to obtain the following quantities-

$$\frac{\partial L}{\partial y(t_0)} = \int_{t_1}^{t_0} a(t)\frac{\partial f(y(t), t, \theta)}{\partial y} dt$$

$$\frac{\partial L}{\partial \theta} = \int_{t_1}^{t_0} a(t)\frac{\partial f(y(t), t, \theta)}{\partial \theta} dt$$

$$\frac{\partial L}{\partial t_0} = \int_{t_1}^{t_0} a(t) \frac{\partial f(y(t), t, \theta)}{\partial t} dt$$

The above trick lies in the center of Neural ODE implementation as it corresponds to the backpropagation of the loss up to the initial state and allows us to efficiently calculate $\mathrm{d}L(\mathbf{x}(t_1))/\mathrm{d}\theta$ without storing all the function activations from the forward pass.

Given the above intriguing properties of ODE solvers, it seems fascinating to combine them with neural networks, i.e. try to model the transition function f with a neural network while utilizing pre-existing ODE solvers to generate its evolution. This combination could yield a powerful modeling tool since neural networks are universal function approximations, and they can, in theory, approximate any differentiable function to an arbitrary precision.

III. Experiments

A. Numerical ODE Solvers

Solving differential equations analytically is not an option for complicated fs, given, for example, by neural networks. We need numerical solvers. Runge-Kutta methods are a family of iterative methods that find approximate solutions to IVPs. We will start with the most straightforward and most intuitive Runge-Kutta method, the Euler method [2].

Consider the IVP

$$\dot{y} = f(y, t), \quad y(t_0) = y_0.$$

where $y(t_0)$, and f are given.

Pick a step-size $h > 0$, a number of steps N, and define

$$y_{n+1} = y_n + hf(y_n, t_n)$$
$$y_{n+1} = y_n + h.$$

This is the most basic numerical integrator. One intuition behind the Euler method is that we are evolving the trajectories by iteratively taking small steps toward the slope.

Hence, we use this Euler method and compare its performance with RK4, an alternate ODE solver, which has a lower order of loss w.r.t. step size h, but with a much higher computational steps than Euler. The RK4 equations [3] are given by-

$$y_{n+1} = y_n + \frac{1}{6} h (k_1 + 2k_2 + 2k_3 + k_4)$$
$$t_{n+1} = t_n + h$$

for $n = 0, 1, 2, 3, \dots, N$ with

$$k_1 = f(y_n, t_n)$$
$$k_2 = f\left(y_n + h\frac{k_1}{2}, t_n + \frac{h}{2}\right)$$
$$k_3 = f\left(y_n + h\frac{k_2}{2}, t_n + \frac{h}{2}\right)$$
$$k_4 = f(y_n + hk_3, t_n + h).$$

Note: The Euler method is just RK4 but considering only k_1. We showcase the application of the ODE solvers in solving IVP for Lotka Volterra equations & equations of a harmonic pendulum, with the earlier being given by-

$$\dot{x} = x - xy$$
$$\dot{y} = xy - y.$$

We start with initial value $(x_0, y_0) = (1, 2)$. Note that there is no closed-form solution to this system of ODEs.

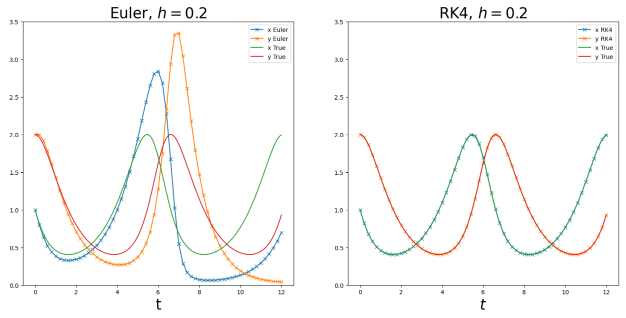

Fig. 1. Learning Lotka-Volterra trajectories: Euler vs RK4

Next, we use ODE solvers for the equations representing the motion of a simple harmonic pendulum-

$$\dot{\theta} = -p_\theta$$
$$\dot{p_\theta} = \sin(\theta).$$

To demonstrate the effectiveness of these ODE solver methods, we plot the obtained trajectories (phase spaces) for different initial values in the form of phase plots.

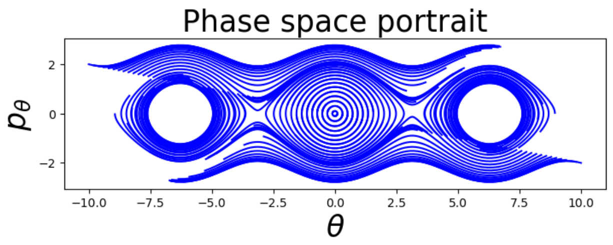

Fig. 2. Phase Space Plot for Pendulum Equations

B. Proof of Concept:

After demonstrating the effectiveness of ODE solvers for the above non-linear systems, we move towards the Neural ODEs. We tested a Neural ODE for restoring true dynamics function from sampled data as an initial proof-of-concept. It was done for the following linear DE-

$$\frac{dy}{dt} = \begin{bmatrix} -0.1 & -1.0 \\ 1.0 & -0.1 \end{bmatrix} y$$

We try to fit,

$$\frac{dy}{dt} = \theta \cdot y, \theta \in \mathbb{R}^{2\times 2}$$

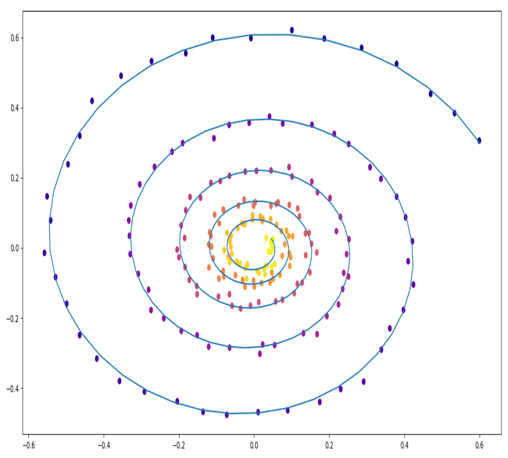

Fig. 3. Learning the Dynamics from sampled points using Neural ODEs

Note that this is a very simple linear network without any non-linearity. It works well since it is very similar to original dynamical equation. In further sections, we model a similar but a more complex spiral dataset with a Variational Autoencoder which using Neural-ODE without making strong assumptions on the process.

C. Neural ODEs as Classifier

Now that we have shown that Neural ODEs are capable of learning any function, we decided to leverage it for classification tasks. As discussed earlier, Neural ODEs model the transformation between input and output as a continuous process governed by differential equations. Thus, Unlike conventional neural networks, which have fixed layer architectures for function approximation, Neural ODEs can leverage something along the lines of continuous-depth models.

For data classification tasks, Neural ODEs allow these continuous-depth models to evolve their complexity dynamically based on the input data, enabling adaptive and flexible representations. They achieve this by learning the continuous transformations between data points, providing a more nuanced understanding of the underlying structure within the dataset. By training the model's parameters through techniques like backpropagation through ODE solvers, the network learns to transform input data into meaningful representations, enabling accurate classification.

To demonstrate the classification capabilities of Neural ODEs we use a synthetic 2D Half Moons Dataset [4], which is a non-linearly separable, binary classification dataset.

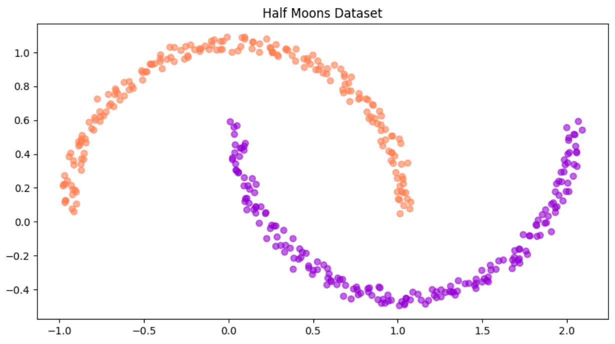

Fig. 4. Half Moons Synthetic Dataset

D. Neural ODE inspired by ResNets

We know that in residual networks [5], the hidden state changes according to the formula

$$y_{t+1} = y_t + f(y_t, \theta_t)$$

where $t \in \{0...T\}$ is residual block number and f is a function learned by layers inside the block.

If one takes a limit of an infinite number of residual blocks with smaller steps, one can get the continuous dynamics of hidden units to be an ordinary differential equation just as above.

$$\frac{dy(t)}{dt} = f(y(t), t, \theta)$$

Starting from the input layer $h(0)$, one can define the output layer $h(T)$ to be the solution to this ODE initial value problem at some time T.

Now one can treat θ as parameters shared among all infinitesimally small residual blocks, which would circle us back towards Neural ODE. Thus, Neural ODEs can be a more flexible alternative to the Resnet for diverse deep learning tasks. To show the effectiveness of Neural ODEs and benchmark it against Resnet, we choose the MNIST classification task. We are using the new RK4 solver-based Neural ODE for our task and analyzing its performance.

E. Neural ODEs As Variational Autoencoders

As the abovementioned experiments show, Neural ODEs can be utilized for continuous sequential data even when the continuous trajectory is in some unknown latent space.

Thus, we move to our next experiment, where we try generating continuous sequential data using Neural ODE. Our training data here consists of random spirals, one half-clockwise and another counter-clockwise. Then, random timespans of size 100 are sampled from these spirals, and trajectories are generated through an RNN encoder. This latent trajectory is then mapped onto the data space trajectory and compared with the actual observations. Thus, the model learns to generate data-alike trajectories.

A generative model through sampling procedure is given by:

$$z_{t_1}, z_{t_2}, ..., z_{t_M} = \text{ODESolve}(z_{t_0}, f, \theta_f, t_0, ..., t_M)$$

with $z_{t_0} \sim \mathcal{N}(0, I)$ and each $x_{t_i} \sim p(x \mid z_{t_i}; \theta_x)$

The above model can be trained using a variational autoencoder approach in the following manner:

1) Run the RNN encoder through the time series backward in time to infer the parameters $\mu_{z_{t_0}}$, $\sigma_{z_{t_0}}$ of variational posterior and sample from it

$$z_{t_0} \sim q\left(z_{t_0} \mid x_{t_0} ... x_{t_M}; t_0 ... t_M; \theta_q\right) = \mathcal{N}\left(z_{t_0} \mid \mu_{z_{t_0}} \sigma_{z_{t_0}}\right)$$

2) Obtain the latent trajectory

$$z_{t_1} ..., z_{t_N} = \text{ODESolve}(z_{t_0}, f, \theta_f, t_0, ..., t_N); \; \frac{dz}{dt} = f(z, t; \theta_f)$$

3) Map the latent trajectory onto the data space using another neural network: $\hat{x_{t_i}}(z_{t_i}, t_i; \theta_x)$
4) Maximize Evidence Lower Bound estimate for sampled trajectory

$$\text{ELBO} \approx N\Big(\sum_{i=0}^{M} \log p(x_{t_i} \mid z_{t_i}(z_{t_0}; \theta_f); \theta_x) +$$
$$KL\left(q(z_{t_0} \mid x_{t_0}, ..., x_{t_M}; t_0, ..., t_M; \theta_q) \parallel \mathcal{N}(0, I)\right)\Big)$$

And in case of Gaussian posterior $p(x \mid z_{t_i}; \theta_x)$ and known noise level σ_x

$$\text{ELBO} \approx -N\Big(\sum_{i=1}^{M} \frac{(x_i - \hat{x}_i)^2}{\sigma_x^2} - \log \sigma_{z_{t_0}}^2 + \mu_{z_{t_0}}^2 + \sigma_{z_{t_0}}^2\Big) + C$$

We now use the above setup to generate data from a model trained on a synthetic spiral dataset, which could represent some low-dimensional real-world time-varying data.

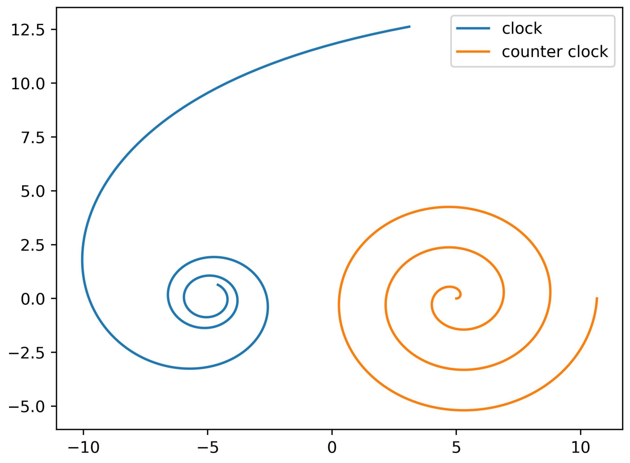

Fig. 5. Training Data Examples for VAE

IV. Results

A. Synthetic Data Classification

For the simple 2D dataset, our model is able to almost perfectly learn the underlying distribution and dynamics of the dataset and give a perfect test accuracy.

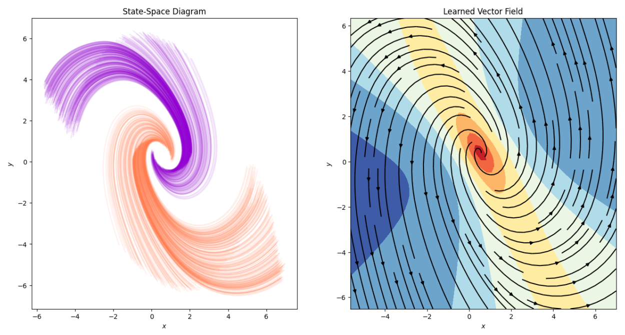

Fig. 6. Learning the dataset distribution and classifying through Neural ODE

B. ResNets vs. Neural ODE vs. Neural ODE-RK4

For the following comparison, we use Resnet-18, the ResNet model with the closest number of parameters to our model.

Model	Accuracy	No. of Parameters
ResNet	99.12 %	180 k
Neural ODE	99.23 %	208 k

TABLE I
Performance Comparision Between Different Models

C. Generative Neural ODEs

The attempt to use Neural ODEs as the generative model was only a moderately successful experiment, as the iterative process of solving the ODEs consumed a lot of time and resources, which is also one of the reasons why Neural ODEs are not widely used for real-world tasks.

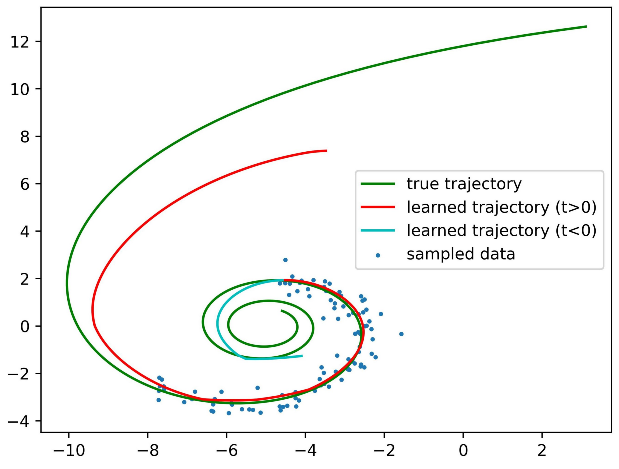

Fig. 7. Generating Spiral through NODE-VAE

V. Challenges Surmounted

The major challenges faced by us during the implementation of the project were:

1) **Theoretical Complexity:** a significant mathematical primer needs to be understood before implementing Neural ODEs; this includes a sound understanding of

vector calculus and differential equations theory. We had to spend a lot of time before being able to grasp the entire working here.

2) **Computational Intensity:** Implementing Neural ODEs involves iteratively solving the differential equations, which are often computationally intensive. This demands efficient solvers and computing resources, especially with large datasets or complex models. The inferencing time for the Neural ODEs is often higher than that of other models. Thus we have mainly dealt with simple models and datasets.
3) **Gradient Computation and Backpropagation:** Training Neural ODEs necessitates computing gradients through the ODE solvers, which is challenging due to the continuous-depth nature of these models. Ensuring stable and accurate gradient computation for effective backpropagation is crucial. We implemented our own code for efficient adjoint calculation to fasten the process.
4) **Limited Pre-existing Tools and Literature:** Unlike most conventional neural network architectures, Neural ODEs have fewer readily available tools and literature due to their under-explored nature. We had to spend a significant amount of time trying to search for good papers and implementations for Neural ODEs.
5) **Interpretebality:** Neural ODEs' continuous-depth nature might obscure interpretability, making understanding how information flows through the model is challenging. Thus it was significantly harder for us to debug the models.

VI. Conclusion

Our exploration of Neural Ordinary Differential Equations (Neural ODEs) showcased their remarkable versatility from image classification to generative modeling. These continuous-depth models offer an innovative approach to dynamic system modeling, demonstrating promising performance in tasks like image classification. Extending Neural ODEs to models like Variational Autoencoders (VAEs) highlights their ability to learn latent representations and generate high-quality data.

However, Neural ODEs have limitations hindering real-world applications. Their computational intensity and reliance on efficient solvers pose challenges, especially with larger datasets or complex architectures. The interpretability issue persists due to their continuous-depth nature, complicating the understanding of information flow. Additionally, the absence of dedicated tools and libraries restricts accessibility and ease of implementation.

Addressing Neural ODEs' computational demands, enhancing interpretability, and improving generalization capabilities are vital for realizing their full potential. Overcoming these limitations is crucial to making Neural ODEs more accessible and impactful across diverse AI applications. Continuous innovation will drive broader adoption and refinement, shaping Neural ODEs for future advancements.

Acknowledgement

The authors acknowledge the usage of Generative AI tools including Chat-GPT and Grammarly for the purpose of language refinement in this report

References

[1] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," *Advances in neural information processing systems*, vol. 31, 2018.

[2] "Euler method," *lamar Tutorial*. [Online]. Available: https://tutorial.math.lamar.edu/classes/de/eulersmethod.aspx

[3] "Runge kutta method," *Wolfram Alpha*. [Online]. Available: https://mathworld.wolfram.com/Runge-KuttaMethod.html

[4] tracy Renee, "How to create two moon dataset," *Medium*, vol. 2023. [Online]. Available: https://medium.com/mlearning-ai/how-to-create-a-two-moon-dataset-and-make-predictions-on-it-dcc090c829af

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.