Time Dependence in Non-Autonomous Neural ODEs

Anonymous Authors[1]

Abstract

Neural Ordinary Differential Equations (ODEs) are elegant reinterpretations of deep networks where continuous time can replace the discrete notion of depth, ODE solvers perform forward propagation, and the adjoint method enables efficient, constant memory backpropagation. Neural ODEs are universal approximators only when they are non-autonomous, that is, the dynamics depends explicitly on time. We propose a novel family of Neural ODEs with time-varying weights, where time-dependence is non-parametric, and the smoothness of weight trajectories can be explicitly controlled to allow a tradeoff between expressiveness and efficiency. Using this enhanced expressiveness, we outperform previous Neural ODE variants in both speed and representational capacity, ultimately outperforming standard ResNet and CNN models on select image classification and video prediction tasks.

1. Introduction & Related Work

The most general Neural ODEs are nonlinear dynamical systems of the form,

$$\dot{x} = f(x, t, \theta) \tag{1}$$

parameterized by $\theta \in \mathbb{R}^k$ and evolving over an input space $x \in \mathbb{R}^n$. The observation that Euler integration of this ODE,

$$x_{t+dt} = x_t + f(x_t, t, \theta)dt$$

resembles residual blocks in ResNets establishes a simple but profound connection between the worlds of deep learning and differential equations (Chen et al., 2018; Haber & Ruthotto, 2017). The evolution of an initial condition $x_0 \in \mathbb{R}^n$ from t_0 to t is given by the integral expression,

$$x_t(\theta) = x_0 + \int_{t_0}^{t} f(x(s), s, \theta)ds.$$

The corresponding flow operator defined by,

$$\phi_t(x_0; \theta) = x_t(\theta),$$

is a parametric map from $\mathbb{R}^n \mapsto \mathbb{R}^n$. As such, it provides a hypothesis space for function estimation in machine learning, and may be viewed as the continuous limit of ResNet-like architectures (He et al., 2016).

Reversible deep architectures enable a layer's activations to be re-derived from the next layer's activations, eliminating the need to store them in memory (Gomez et al., 2017). For a Neural ODE, by construction a reversible map, loss function gradients can be computed via the adjoint sensitivity method with constant memory cost independent of depth. This decoupling of depth and memory has major implications for applications involving large video and 3D datasets.

When time dependence is dropped from Eqn 1, the system becomes *autonomous* (Khalil & Grizzle, 2002). Irrespective of number of parameters, an autonomous Neural ODE cannot be a universal approximator since two trajectories cannot intersect, a consequence of each x being uniquely associated to a $\dot{x}$ with no time-dependence. As a result, simple continuous, differentiable and invertible maps such as $h(x) = -x, x \in \mathbb{R}$ cannot be represented by the flow operators of autonomous systems (Dupont et al., 2019). Note that this is a price of continuity: residual blocks which are discrete dynamical systems can generate discrete points at unit-time intervals side-stepping trajectory crossing.

For continuous systems, it is easy to see that allowing flows to be time-varying is sufficient to resolve this issue (Zhang et al., 2019). Such *non-autonomous* systems turn out to be universal and can equivalently be expressed as autonomous systems evolving on an extended input space with dimensionality increased by one. This idea of augmenting the dimensionality of the input space of an autonomous system was explored in (Dupont et al., 2019), which further highlighted the representational capacity limitations of purely autonomous systems. Despite the crucial role of time in Neural ODE approximation capabilities, the dominant approach in the literature is simply to append time to other inputs, giving it no special status. Instead, in this work, we:

1. Introduce new, ***explicit*** constructions of non-autonomous Neural ODEs (NANODEs) of the form

$$\dot{x} = f(x, \theta(t; \alpha)), \tag{2}$$

where hidden units are rich functions of time with their own parameters, α. This non-autonomous treatment frees the weights for each hidden layer to vary in complex ways with time t, allowing trajectories to cross. (Sec. 2.3). We explore a flexible mechanism for varying expressiveness while adhering to a given memory limit. (Sec. 3.1).

2. We then use the above framework to outperform previous Neural ODE variants and standard ResNet and CNN baselines on CIFAR classification and video prediction tasks.

2. Methods: Neural ODEs & Time

2.1. Resnets & Autonomous Neural ODEs

Consider the case of a standard Neural Network with hidden states given by $h_{t+1} := \sigma(W_t h_t)$. The linear transformation of each layer, $W_t h_t$, is a matrix multiplication between a weight matrix $W_t \in \mathbb{R}^{N\times N}$ and a vector $h_t \in \mathbb{R}^N$.

In a Deep Neural Network, the weight matrices W_t are composed of $N \times N$ scalar weights, $w_{ij} \in \mathbb{R}$. The hidden dynamics can be rewritten as $\mathbf{h}_{t+1} := f(\mathbf{h}_t, \theta_t)$, where θ_t are learned parameters encoding the weight matrices.

Unconstrained ResNet (Uncon-Resnet): Residual Neural Networks (ResNets) iteratively apply additive non-linear residuals to a hidden state:

$$\mathbf{h}_{t+1} = \mathbf{h}_t + f(\mathbf{h}_t, \theta_t), \tag{3}$$

where $t \in \{0...T\}$ and θ_t are the parameters of each Residual Block. This can be viewed as a discretization of the Initial Value Problem (IVP):

$$\mathbf{h}_T = \mathbf{h}_0 + \int_0^T f(\mathbf{h}_s, \theta_s) ds \tag{4}$$

Constrained ResNet (Con-Resnet): In the Neural ODE used in the classification experiments in (Chen et al., 2018), a function $\dot{\mathbf{h}}_t = f(\mathbf{h}_t; \theta)$ specifies the derivative and is approximated by a given Neural Network block. This block is defined independent of time, so weights are shared across steps. Through a dynamical systems lens, this Neural ODE approximates an autonomous nonlinear dynamical system. This Neural ODE is analogous to a Constrained ResNet with shared weights between blocks.

2.2. Non-Autonomous Neural ODEs - Time Appended State

By contrast to an autonomous system, consider the general non-autonomous system of the form, $\dot{x} = f(x, t, \theta)$, where θ are the parameters. For simplicity, let us discuss the case where f is specified by a single linear neural net layer with an activation function σ.

Recall that in an *autonomous system* there is no time dependence, so at each time t, $\dot{x} = \sigma(Wx)$, and $\theta := W \in \mathbb{R}^{N\times N}$ for $x \in \mathbb{R}^N$.

Time Appended (AppNODE): In works by Chen et al. (2018) and Dupont et al. (2019), we see a limited variant of a *non-autonomous system* that one might call *semi-autonomous* as time t is simply added in: $\dot{x} = \sigma(W[x, t])$. In this case, $W \in \mathbb{R}^{N\times(N+1)}$. Each layer can take the node corresponding to t and decide to use it to adjust other weights, but there is no explicit requirement or regularization to force or encourage the network to do this.

Let us consider an alternative: making the weights themselves **explicit** functions of time. For clarity, everything hereon is our novel contribution unless otherwise noted.

2.3. Non-Autonomous ODEs - Weights as a Function of Time

In a Neural ODE, the discretization of the ODE solver is roughly analogous to depth in a standard Neural Network. This connection is most intuitive in the discrete, as opposed to adaptive, ODE solver case, where the integral in Equation 4 is approximated by a discretization:

$$\int_0^T f(\mathbf{h}_s, \theta_s) ds \approx \sum_{t=0}^{T/\Delta t} f(\mathbf{h}_t, \theta_t)\Delta t. \tag{5}$$

For Non-Autonomous Neural ODE (NANODE) where weights θ_t are themselves functions of time, $\theta(t; \alpha)$, with parameters α, the question arises of what kinds of functions to use to specify $\theta(t; \alpha)$. We consider the following framings to be natural to explore.

2.3.1. Bases for Time Varying Dynamics

Framed in terms of a dense network block $\sigma(Wh)$, we can make the weight matrix a function of time, $W \to W_t$ by associating each $W_{t,ij}$ element in the time-dependent weight matrix with a function $W_{t,ij} = \phi(t, \alpha)$. This function, ϕ, can be defined by numerous bases.

Bucketed Time (B-NANODE): Here, we consider piecewise constant weights. We initialize a vector $\vec{b} \in \mathbb{R}^d$ to represent each W_{ij} over t (i.e., fixing i, j). In the simplest case, if our discretization (depth) L and d are the same, then we can map each time t to an index in $\vec{b}$ to select distinct parameters over time. For $d < L$, we can group parameters between successive times to have partial weight-sharing.

Polynomial (Poly-NANODE): We define $\phi(t, \alpha)$ as the output of a $(d-1)$-degree polynomial with learned coefficients $\alpha \in \mathbb{R}^d$, i.e.

$$W_{t,ij} = \phi(t, \alpha) = \alpha^T z(t), \tag{6}$$

where z is the monomial basis or a better conditioned basis like Chebyshev or Legendre polynomials. For the monomial basis, we have:

$$W_{t,ij} = \phi(t, \alpha) = \sum_{n=0}^{d-1} \alpha_n t^n. \tag{7}$$

As we increase d, each $W_{t,ij}$ function's expressiveness increases, allowing W to vary in complex ways over time.

Note that using 1-degree polynomials is analogous to augmenting the state with scalar value t. Augmenting the state in this way is therefore strictly less general than the above non-autonomous construction. If we were to reframe our non-autonomous system to be autonomous, where $x \in \mathbb{R}^{n+1}$ instead of $x \in \mathbb{R}^n$, by simply letting $x_{n+1} = t$ and $x'_{n+1} = 1$, we arrive at the augmented case (Dupont et al., 2019). We note that trajectories can now cross over each other.

While this standard polynomial construction is intuitive, we found it difficult to train. The output magnitude of a standard polynomial can vary greatly and higher order polynomials are very sensitive to small changes in t. This motivates the trigonometric construction below.

Trigonometric Polynomials (T-NANODE): We define W_t as a finite linear combination of basis functions $\sin(nt)$ and $\cos(nt)$, with where $n \in \mathbb{N}_+$:

$$W_{t,ij} = \phi(t, \alpha) = a_0 + \sum_{n=1}^{d} a_n \cos(nt) + \sum_{n=1}^{d} b_n \sin(nt) \tag{8}$$

with per W_{ij} learnable coefficients:

$$\alpha = [a_0, a_1, ..., a_n, b_0, b_1, ...b_n].$$

These polynomials are widely used, for example, in the interpolation of periodic functions and in discrete Fourier transforms. Trigonometric polynomials have bounded magnitude, so can parameterize kernels that are less susceptible to inciting vanishing/exploding gradient issues that hinder optimization (Bengio et al., 1994).

3. Experiments

We conduct a suite of experiments to demonstrate the effectiveness of our NANODE approach.

3.1. Image Classification

We first consider the task of image classification using residual flow architectures defined by successive bottleneck residual blocks. As baselines, we trained two ResNet variants: 1) *Uncon-ResNet* and 2) *Con-ResNet* (described in Section 2.1). *Uncon-ResNet* is a standard ResNet architecture where the weight of each ResNet block are not tied to the weights of other ResNet blocks. *Con-ResNet* is a ResNet architecture where the weights of each ResNet block are constrained to all utilize the same set of parameters, resembling an autonomous Neural ODE, where the weight at each step are fixed.

Table 1. Comparison of various architectures for CIFAR-10 and CIFAR-100 image classification tasks. Trigonometric NANODE (T-NANODE-10) outperforms an Autonomous NODE (Auto), as well as the largest Unconstrained ResNet we could train on a single GPU. All the NANODE architectures have a significantly smaller activation memory footprint (ACT. MEM) than the equivalent Unconstrained Resnet. Bucket NANODE (B-NANODE) tended to perform worse than T-NANODE for order < depth. Results averaged across 3 runs, distribution info in supplementary materials.

MODEL	CIFAR10 ACC (%)	CIFAR100 ACC (%)	ACT. MEM (GB)	PARAM MEM (GB)
AUTO	82.98	50.33	0.3	2.8E-4
APPNODE	83.20	60.68	0.3	2.8E-4
CON. RESNET	82.35	54.69	3.0	2.8E-4
UNCON. RESNET	86.72	60.91	3.0	2.8E-3
B-NANODE-10	84.38	51.66	0.3	2.8E-3
T-NANODE-10	**90.10**	**66.49**	**0.3**	5.6E-3
B-NANODE-100	**93.22**	**64.06**	**0.3**	2.8E-2

In addition to these baselines, we train several NANODE variants, shown in Figures 1 and 2. Each NANODE has bases ϕ of varying orders parameterizing their hidden unit dynamics. Our experiments demonstrate that by making the hidden unit dynamics non-autonomous, we can retain much of the memory benefit of an autonomous ODE (Auto) while achieving performance comparable to that of an Unconstrained ResNet. Furthermore, the memory efficiency benefits granted via the adjoint method allow us to train models significantly "deeper" than the Unconstrained ResNets and outperform them, as shown in Table 1. In Figures 1 and 2, we show how we can leverage the order d of $W_{t,ij} = \phi(t, \alpha)$ to vary the NANODE's representational capacity. This allows us to elegantly trade off between expressiveness and parameter-efficiency. It is worth noting that parameters typically require far less memory than activations in the CNN or ResNet context. For a reversible architecture such as our NANODEs, with activation memory complexity $\mathcal{O}(1)$, we only need to store our parameters, and the activations of a single Block. However, a standard ResNet with $\mathcal{O}(L)$ activation memory complexity must store activations for all L layers. As shown in Table 1, this means that, for a given memory budget, we can train much wider and deeper neural networks.

In Figure 2, we also compare Bucket and Trigonometric time treatments, the two best performing variants. We find that the trigonometric treatment outperforms the piece-wise, Bucket treatment for order less than the discretization, $d <$

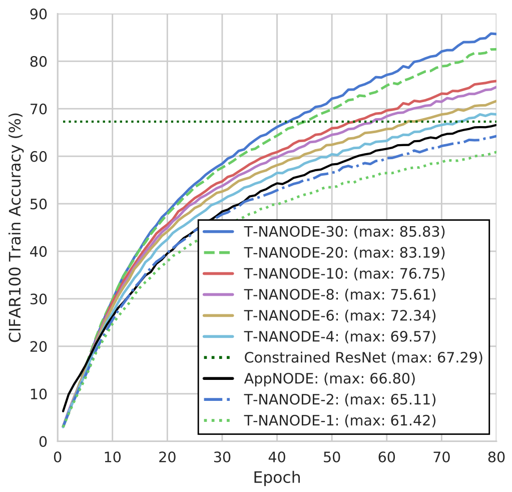

Figure 1. Increasing the order of the time-varying function defining a NANODEs dynamics enhances its expressiveness. For these NANODEs with discretization of 100 steps ($dt = 0.01$), as the degree of the trigonometric basis scales from 1 to 30, the representational capacity of the network increases, as shown by its ability to fit the training set. The threshold line represents the deepest constrained ResNet baseline that we could train on a single GPU.

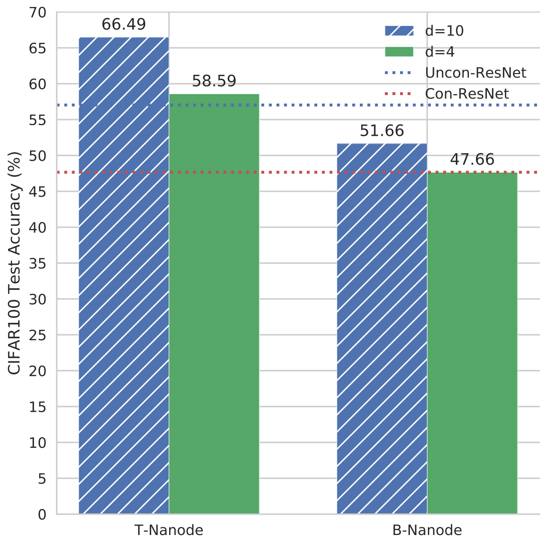

Figure 2. Mirroring results on the training set, as we scale the order of the Trigonometric Polynomial from 4 to 10, the expressiveness of the network increases as shown by its generalization performance on the test set. T-NANODE outperforms B-NANODE for order $d <$ discretization, suggesting the benefits of smoothness. Horizontal lines illustrate how unconstrained and constrained ResNet baselines, the largest we could train on a single GPU, compare.

L, suggesting the benefits of smoothness.

3.2. Video Prediction

Leveraging this memory scaling advantage, we consider the problem of video prediction, whereby a model is tasked with generating future frames conditioned upon some initial observation frames. In deterministic settings, e.g. an object sliding with a fixed velocity, a model has to infer the speed and direction from the prior frames to accurately extrapolate. Standard video prediction models can be memory intensive to train. Video tensor activations must contain an additional time dimension that scales linearly as the number of conditioning or generation frames increases. This makes it difficult to simultaneously parameterize powerful models with many filters, and learn long-horizon dynamics.

Experiments are conducted using the Moving MNIST (Srivastava et al., 2015) and BAIR Robot Pushing Small (Finn et al., 2016) video datasets. We used 2 conditioning frames to predict 10 future ahead in both tasks. We train an Encoder-Decoder Stochastic Video Generation (SVG) model with a learned prior, based on Denton & Fergus (2018). The baseline architecture contains VGG blocks (Simonyan & Zisserman, 2014) of several dimension preserving CNN blocks with 3×3 filters and 1×1 stride, followed by 2×2 max-pooling operations with stride 2×2. For our NANODE alternative, we replace the 2 out of every 3 CNN layers with a single NANODE block composed of trigonometric polynomial basis $\phi(t, \theta_{ij})$ of discretization $dt = 0.33$ and order 3. We train this reversible model on a single GPU, and are able to achieve faster convergence and lower final loss on both tasks compared to the more memory intensive baseline. Table 2 states the min loss achieved and memory usage of the respective architectures. There is much potential to further improve these video prediction architectures by pairing ideas regarding optimal width vs. depth scaling (Tan & Le, 2019) with the arbitrary depth scaling ability and expressiveness that NANODEs provide.

Table 2. Comparison in performance of different architectures on both the MNIST and BAIR Robot Pushing Small (BRP) video prediction datasets. The Evidence Lower Bound (ELBO) and the memory footprint of the network's activations and parameters are shown. The NANODE model is able to significantly outperform the SVG and NODE models on both tasks, while keeping a small activation memory (ACT. MEM) footprint similar to the Autonomous NODE (Auto) architecture.

MODEL	MNIST (-ELBO)	BRP (-ELBO)	ACT. MEM (GB)	PARAM MEM (GB)
AUTO	3.3E-5	9.0E-5	4.7	4.6E-3
SVG	2.2E-5	9.5E-5	4.9	4.6E-3
NANODE	**8.5E-6**	**7.6E-5**	**4.7**	1.9E-2

References

Bengio, Y., Simard, P., and Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

Chen, T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. In *Advances in neural information processing systems*, pp. 6571–6583, 2018.

Denton, E. and Fergus, R. Stochastic video generation with a learned prior. *arXiv preprint arXiv:1802.07687*, 2018.

Dupont, E., Doucet, A., and Teh, Y. W. Augmented neural odes. In *Advances in Neural Information Processing Systems*, pp. 3134–3144, 2019.

Finn, C., Goodfellow, I., and Levine, S. Unsupervised learning for physical interaction through video prediction. In *Advances in neural information processing systems*, pp. 64–72, 2016.

Gomez, A. N., Ren, M., Urtasun, R., and Grosse, R. B. The reversible residual network: Backpropagation without storing activations. In *Advances in neural information processing systems*, pp. 2214–2224, 2017.

Haber, E. and Ruthotto, L. Stable architectures for deep neural networks. *Inverse Problems*, 34(1):014004, 2017.

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

Khalil, H. K. and Grizzle, J. W. *Nonlinear systems*, volume 3. Prentice hall Upper Saddle River, NJ, 2002.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Srivastava, N., Mansimov, E., and Salakhudinov, R. Unsupervised learning of video representations using lstms. In *International conference on machine learning*, pp. 843–852, 2015.

Tan, M. and Le, Q. V. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

Zhang, H., Gao, X., Unterman, J., and Arodz, T. Approximation capabilities of neural ordinary differential equations. *arXiv preprint arXiv:1907.12998*, 2019.