Artificial Intelligence Research and Development
A. Cortés et al. (Eds.)

doi:10.3233/FAIA220337

On Flows of Neural Ordinary Differential Equations That Are Solutions of Lotka-Volterra Dynamical Systems

Argimiro ARRATIA [a,1], Carlos ORTIZ [b] and Marcel ROMANÍ [a]
[a] *Soft Computing Research Group (SOCO)*
at Intelligent Data Science and Artificial Intelligence Research Center
Department of Computer Sciences,
Polytechnical University of Catalonia, Barcelona, Spain.
argimiro@cs.upc.edu, marcel.romani@estudiantat.upc.edu
[b] *Dept. of Computer Science and Mathematics,*
Arcadia University, Glenside, PA, U.S.A.
ortiz@arcadia.edu

Abstract. Neural Ordinary Differential Equations (NODE) have emerged as a novel approach to deep learning, where instead of specifying a discrete sequence of hidden layers, it parameterizes the derivative of the hidden state using a neural network [1]. The solution to the underlying dynamical system is a flow, and various works have explored the universality of flows, in the sense of being able to approximate any analytical function. In this paper we present preliminary work aimed at identifying families of systems of ordinary differential equations (SODE) that are universal, in the sense that they encompass most of the systems of differential equations that appear in practice. Once one of these (candidate) universal SODEs is found, we define a process that generates a family of NODEs whose flows are precisely the solutions of the universal SODEs found above. The candidate universal SODE family that we present here is the generalized Lotka-Volterra (LV) families of differential equations. We present the NODE models built upon this LV systems and a description of their appropriate flows and some preliminary implementations of this process.

Keywords. Neural Ordinary Differential Equations, Lotka-Volterra system, neural networks, flows.

1. Introduction

Models from the class of neural networks with an infinite impulse response such as recurrent neural networks, and more in particular Residual Neural Networks (ResNet), are the inspiration for the Neural Ordinary Differential Equations (NODE) network model [1]. In a residual network each layer can be defined as a finite transformation of the previous layer:

[1]Corresponding Author: Argimiro Arratia, e-mail: argimiro@cs.upc.edu

$$h_{t+1} = h_t + g(h_t, \theta_t) \tag{1}$$

where h_t is the hidden state at layer t, g is a dimension preserving function and θ is a vector of parameters. These iterative updates can also be interpreted as an Euler discretization of a continuous transformation [3]. By augmenting the number of layers and taking smaller steps Δt, the ResNet dynamics expressed by (1) becomes in the limit an ordinary differential equation (ODE) specified by a neural network

$$\frac{dy(t)}{dt} = f(y(t), \theta(t), t) \tag{2}$$

Here f represents the Euler's discretization method to approximate a continuous function, and at each discrete time t, $y(t) = h_t$, $\theta(t) = \theta_t$ and $\Delta t f(h_t, \theta_t, t) = g(h_t, \theta_t)$. Thus, a neural ODE extends the traditional residual network model in that it has infinitely many layers at different point in time, and instead of fitting different weights and biases at each layer, it needs to fit a set of parameters θ that minimizes some cost function that depends on the initial values of $y(t)$, say $y(0) = h_0$, and the output of the system, say at some time T, $y(T) = h_T$ (w.l.o.g. we will consider $T = 1$). In other words, given the task of modeling a mapping $F : \mathbb{R}^n \to \mathbb{R}^n$, we are interested in optimizing the parameters θ such that the solution to the initial value problem

$$\frac{dy}{dt} = f(y, \theta, t), \quad y(0) = x \tag{3}$$

will accurately predict F at time $T = 1$. This value can be computed by an ODE solver, which evaluates the hidden unit dynamics of f wherever necessary from input layer $y(0) = h_0$ to output layer $y(1) = h_1$. There are many powerful and accurate optimizing algorithms, for instance, the family of algorithms based on the adjoint method [4].

Because we are interested in how the values of $y(1)$ change given the initial values $y(0)$, we study the *flow* of the NODEs. The NODE approach promises to yields benefits in terms of memory management as well as harnessing the theoretical wealth of minimization techniques, and stability results, in dynamical systems.

A first step to explore this approach is to study the universality of the NODEs in terms of their flows being able to approximate any function $F : \mathbb{R}^n \to \mathbb{R}^n$. Results have been obtained that give topological and analytical conditions on collections of functions such that their flows are universal for different measures of approximations [6]. Most of these approaches, however, study NODEs where the function $f(y, \theta, t)$ itself is a neural network, e.g. [5]. We note that these characterizations do not generate in themselves interesting families of SODEs.

We are interested in a more direct approach. We want to identify first families $\mathscr{G}$ of systems of differential equations that are universal, in the sense that they encompass most of the systems of differential equations that appear in practice. Then we want to find a family $\mathscr{F}$ of functions f for which the flows of the differential equation in (1) is exactly the family $\mathscr{F}$. Thirdly, we want to explore the process of optimizing the parameters of a differential equation (1) for functions f in $\mathscr{F}$.

In the following sections, we first present our general scheme for defining flows of universal SODEs and then the Lotka-Volterra system as our first candidate of universal SODE for building Neural ODE.

2. Computing the flow

Definition 2.1 *Let* $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ *be a time independent vector field and let* $\mathbf{y}(t)$ *be the solution of the Initial Value Problem (IVP)*

$$\frac{d\mathbf{y}}{dt} = \mathbf{f}(\mathbf{y}(t), \theta) \text{ with } \mathbf{y}(0) = \mathbf{x}.$$

Then the function

$$\varphi : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n,$$

defined as $\varphi(\mathbf{x}, t) = \mathbf{y}(t)$ *is called the flow of the vector field* $\mathbf{f}$.

We can define the optimization process the neural ODE appearing in (3) in terms of the flow as follows:

Given the IVP in (3), the optimization process (by the gradient method based on the adjoint approach) obtains the values of the parameters θ that minimize a total error (or cost function) constructed from comparing a collection of training data

$$\hat{T} = \{(\mathbf{x}_i, \hat{\mathbf{y}}_i)\}_{i \in I}$$

with a collection of calculated outputs of the IVP in (3):

$$T = \{(\mathbf{x}_i, \varphi(\mathbf{x}_i, 1))\}_{i \in I}.$$

Once the optimal values of θ are obtained, the solution to the optimization problem is given by the function $\varphi(\mathbf{x}, 1)$. More specifically, for any input $\mathbf{x}$, the value of $\varphi(\mathbf{x}, 1)$ is the output of neural SODE solution of the IVP (3) at $t = 1$, with the optimal parameters and with initial value $\mathbf{y}(0) = \mathbf{x}$.

Observe that for the optimization process and for the generation of the approximation functions, the key role is played by the flow value $\varphi(\mathbf{x}, 1)$ and not by the solution of the IVP in (3) in the interval $(0, 1)$ of t.

In order for the Neural SODEs approach to be useful, one needs to ensure that the flows of the SODEs involved are **universal**, in the sense that there exists a family $\mathscr{S}$ of SODEs such that any function $F : \mathbb{R}^n \to \mathbb{R}^n$ can be approximated by the flow of an SODE $S \in \mathscr{S}$. Here the definition of "approximate" can take many different meanings, as discussed elsewhere, but essentially means that for any F, there exists an $S \in \mathscr{S}$ with associated flow φ_S such that for any $\mathbf{x} \in \mathbb{R}^n$, $F(\mathbf{x})$ is close to $\varphi_S(\mathbf{x}, 1)$.

Here is how we propose to find a collection $\mathscr{S}$ of SODEs that is universal. First, we simplify the situation above as follows. Observe that for every natural n there exists (computationally) easy bijections $\mathbb{R}$ into $\mathbb{R}^n$. Let us call one such (computable) bijection $\mathbf{h}$. Thus we can recast approximating a function $\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^n$ into approximating an equivalent function $\mathbf{F}^* : \mathbb{R} \to \mathbb{R}^n$ defined as $\mathbf{F}^*(t) = \mathbf{F}(\mathbf{h}(t))$. Thus we can recast the following concepts:

1. The IVP (3) is now: $\frac{d\mathbf{y}}{dt} = \mathbf{f}(\mathbf{y}, \theta)$ with $\mathbf{y}(0) = (x, x, \ldots, x)$.

2. The training data set $\hat{T} = \{(\mathbf{x}_i, \hat{\mathbf{y}}_i)\}_{i\in I}$ is now of the form $\hat{T}_1 = \{(x_i, \hat{\mathbf{y}}_i)\}_{i\in I}$.
3. The set of inputs-outputs of the IVP is now $T_1 = \{(x_i, \varphi(x_i, 1))\}_{i\in I}$, where $\varphi(x,1)$ is the solution of the IVP with initial condition $\mathbf{y}(0) = (x, x, \ldots, x)$, at $t = 1$.

Note that $\varphi(x,1)$ is a function from $\mathbb{R}$ to $\mathbb{R}^n$. Hence our optimization process for neural SODE becomes the following.

Definition 2.2 (The optimization process for a Neural SODEs) *Given an IVP:*

$$\frac{d\mathbf{y}}{dt} = \mathbf{f}(\mathbf{y}, \theta) \text{ with } \mathbf{y}(0) = (x, x, \ldots, x) \tag{4}$$

The optimization process (by the gradient method based on the adjoint approach) obtains the values of the parameters θ that minimize a total error (or cost function) constructed from comparing a collection of training data

$$\hat{T} = \{(x_i, \hat{\mathbf{y}}_i)\}_{i\in I}$$

with a collection of calculated outputs of the IVP in (4):

$$T = \{(x_i, \varphi(x_i, 1))\}_{i\in I}.$$

Once the optimal values of θ are obtained, the solution to the optimization problem is given by the function $\varphi(x,1)$. More specifically, for any input x, the value of $\varphi(x,1)$ is the output of our optimized Neural SODE.

3. The Lotka-Volterra system

Now we consider the Lotka-Volterra n-dimensional SODE. We begin by observing, as done elsewhere, that there is heuristic evidence that the families of n-dimensional Lotka-Volterra are universal in the sense that most of the types of systems of differential equations used in practice can be reduced to SODE's that are n-dimensional Lotka-Volterra [2]. Thus, a vast collection of functions F, the solutions of a very general collection of SODEs, can be seen as solutions of a Lotka Volterra n-dimensional SODE.

We propose then to study the flows $\varphi(x,t)$ that are associated with n-dimensional Lotka-Volterra SODEs. More specifically, given an n-dimensional Lotka Volterra Initial Value Problem:

$$LV(n, \theta, \mathbf{z}_0) : \frac{d\mathbf{z}}{dx} = \mathbf{L}(\mathbf{z}, \theta) \text{ with } \mathbf{z}(0) = \mathbf{z}_0, \tag{5}$$

where the n-dimensional Lotka-Volterra equations $LV(n, \theta, \mathbf{z}_0)$ with parameters $\theta = (\lambda_1, \ldots, \lambda_n; A_{ij} : 1 \leq i \leq n, 1 \leq j \leq m)$ are:

$$z_i' = \lambda_i z_i + z_i \sum_{j=1}^{m} A_{ij} z_j, \quad i = 1, \ldots, n$$

we will associate to it a (trivial) SODE with flow $\varphi_{n,\theta,\mathbf{z}_0}(t,x)$ such that $\varphi_{n,\theta,\mathbf{z}_0}(x,1) = \mathbf{z}(x)$.

Since the Lotka Volterra SODE's capture all the useful SODEs, we can hope that the functions $\varphi_{n,\theta,\mathbf{z}_0}(\_,1)$ will capture all solutions of the useful SODEs.

Our proposed process to use Neural SODE's on a training problem defined by a collection of points $\hat{T} = \{(x_i, \hat{\mathbf{y}}_i)\}_{i\in I}$, with, for every i, $\hat{\mathbf{y}}_i \in \mathbb{R}^n$ is explained in the following section.

3.1. Proposed optimization Process for Neural SODEs

We consider the Lotka-Volterra n-dimensional IVP defined in (5):

$$LV(n,\theta,\mathbf{z}_0) : \frac{d\mathbf{z}}{dx} = \mathbf{L}(\mathbf{z},\theta) \text{ with } \mathbf{z}(0) = \mathbf{z}_0$$

We define an associated IVP for a SODE in the independent variable t and with parameters θ, t as follows

$$\frac{d\mathbf{y}}{dt} = \begin{bmatrix} -x \\ -x \\ \cdots \\ -x \end{bmatrix} + \mathbf{z}(x) \text{ with } \mathbf{y}(0) = \begin{bmatrix} x \\ x \\ \cdots \\ x \end{bmatrix}, \tag{6}$$

where $\mathbf{z}(x)$ is the solution of the $LV(n,\theta,\mathbf{z}_0)$ defined in (5), evaluated at x. Since the solution of the IVP (6) is simply

$$\mathbf{y}(t,x) = (1-t)\begin{bmatrix} x \\ x \\ \cdots \\ x \end{bmatrix} + (t)\mathbf{z}(x).$$

We see that $\mathbf{y}(1,x) = \mathbf{z}(x)$. Since $\varphi(x,1)$, the flow of the IVP (6) at $t = 1$, is $\mathbf{y}(1,x)$, we have that $\varphi(x,1) = \mathbf{z}(x)$.

We apply the optimization process (by the gradient method based on the adjoint approach) to $LV(n,\theta,\mathbf{z}_0)$ to obtain the values of the parameters $(\theta,\mathbf{z}_0)$ that minimize a total error (or cost function) constructed from comparing the collection of training data

$$\hat{T}_1 = \{(x_i, \hat{\mathbf{y}}_i)\}\, i \in I$$

with a collection of calculated outputs of the $LV(n,\theta,\mathbf{z}_0)$, and defined as

$$T = \{(x_i, \varphi(x_i,1))\}_{i\in I} = \{(x_i, \mathbf{z}(x_i))\}_{i\in I},$$

(since $\varphi(x,1) = \mathbf{z}(x)$).

Once the optimal values of $\theta, \mathbf{z}_0$ are obtained, the solution to the optimization problem is given by the function $\mathbf{z}(x)$. More specifically, for any input x, the value of $\varphi(x,1) = \mathbf{z}(x)$ is the output of our approximation function.

We remark that in this approach, we are optimizing the parameters $\theta, \mathbf{z}_0$ of the initial value problem $LV(n,\theta,\mathbf{z}_0)$ using the adjoint method, as in the classical Neural SODE approach. The advantages of this approach are: the process is easily adapted to other universal families of SODEs, such as Riccati equations; the process is easily scalable to any dimension n; as with the canonical Neural SODE approach, the use of memory may be limited in this approach.

4. Experiments

4.1. Reproducing the evolution of Lotka-Volterra populations

We initialized a Neural ODE whose differential equation corresponds to an n-dimensional Lotka-Volterra system with random parameters $\theta \in [-0.05, 0.05]^{n(n+1)}$ and initial conditions $\mathbf{z}_0 \in [0,1]^n$. The goal was to find such optimal values that minimize a loss function given by

$$\text{Loss}(\hat{\mathbf{Y}}, \mathbf{Y}) = \sum_i |\hat{\mathbf{y}}_i - \mathbf{y}_i|^2. \tag{7}$$

The first experiment was performed on an easy data set produced by solving a Lotka-Volterra system of equations. Populations following n-dimensional Lotka-Volterra systems for $n = 2, 5, 10$ were generated by (randomly) choosing values of parameters $\hat{\theta} \in [-0.25, 0.25]^{n(n+1)}$ and initial conditions $\hat{\mathbf{z}}_0 \in [0,1]^n$ and letting the system evolve for a limited time span of $[0,2]$. Population values $\hat{\mathbf{y}}_i$ at each time step x_i were recorded every 0.05 time units to generate the data sets $\hat{T}_n = \{(x_i, \hat{\mathbf{y}}_i)\}_i$. Random noise was added to $\hat{\mathbf{y}}_i$.

Results are shown in Figures 1 and 2. As we can see, our Neural ODE model parameters have been optimized correctly using the adjoint method.

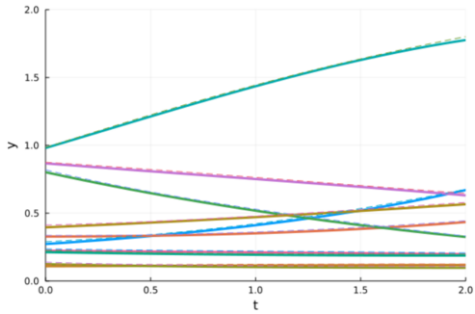

Figure 1. Plot of the estimated data (dashed lines) over the target data (thick lines, the noise added in the training set is not shown).

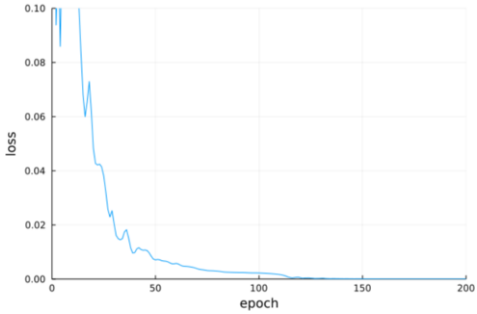

Figure 2. Loss of our Neural ODE model through the training phase.

4.2. *Approximation of* $1 + \sin(x)$

Now the goal is to approximate the function $1 + \sin(x)$ with a Neural ODE whose differential equation is a high dimensional ($n = 20$) Lotka-Volterra system. In this case, given that the target $\hat{y}_i$ is 1-dimensional, the loss function is defined as to take into account only the first dimension of the system, while the other 19 may take whatever values are best.

Figure 3 shows that for a small interval $[0, \pi]$ our algorithm has been able to approximate the target function pretty well.

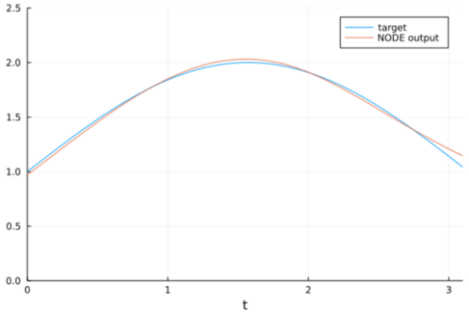

Figure 3. Plot of the 1st dimension of the Neural ODE output over the target $1 + \sin(x)$.

On the other hand, in Figure 4 we can see that even though the loss has not reached a minimum, numerical instabilities on the differential equation solvers complicate a further optimization.

References

[1] Chen, R. T., Rubanova, Y., Bettencourt, J., Duvenaud, D. K. (2018). Neural ordinary differential equations. Advances in Neural Information Processing Systems, 31, pp. 6571-6583.

[2] Hernndez-Bermejo, B., Fairén, V. (1997). Lotka-Volterra representation of general nonlinear systems. Mathematical Biosciences, 140 (1), 1-32.

[3] Lu, Y., Zhong, A., Li, Q., Dong, B. (2018). Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In: International Conference on Machine Learning (pp. 3276-3285). PMLR.

[4] Pontryagin, L. S. (1987). Mathematical theory of optimal processes. CRC press.

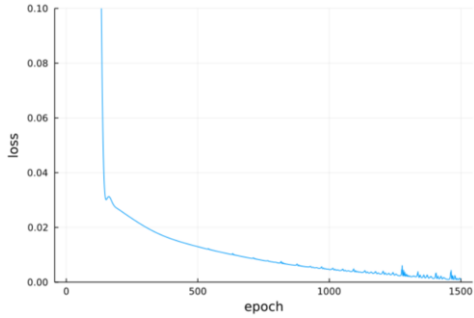

Figure 4. Loss of our Neural ODE model through the training phase.

[5] Rackauckas, C., et al (2020). Universal differential equations for scientific machine learning. arXiv preprint arXiv:2001.04385.

[6] Teshima, T., Tojo, K., Ikeda, M., Ishikawa, I., Oono, K. (2020). Universal approximation property of neural ordinary differential equations. arXiv preprint arXiv:2012.02414.