Neural Ordinary Differential Equations

Weijie Gan [*1] **Zayid Oyelami** [*1]

Abstract

A new family of deep-learning, especially in recurrent neutral network (RNN), is introduced using differential equations. Instead of simply passing a layer to the next phase, the value of a layer is computed by solving a differential equations via a black-box-based algorithm. Effectively, the use of ODEs in neural networks is known as NODE. The utility of this method was tested in several experiments to exhibit constant memory cost, adaptive computation, and better performance than standard RNNs while accounting for accuracy and computational costs.

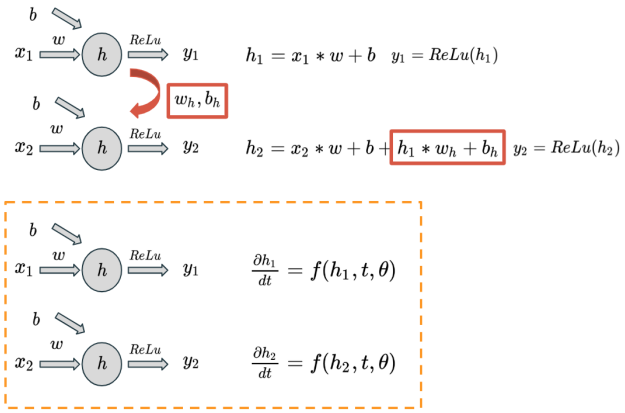

Figure 1. Idea of RNN and NODE. $ReLu$ is activation function in neural network and $x_1, x_2...$ are input batch training data. The above figure is general RNN and the below(with orange rectangle) is NODE.

1. Introduction (by Weijie)

Recurrent Neutral Network(RNN) is a class of artificial neutral network and deep learning technology, where the value of layers in neutral network is computed not only from the input, but also value of same layers in last phase. More concretely, "the last phase" means when we use to the last batch of data train our neutral network. This structure builds a connection between nodes to form a directed graph along a temporal sequence.

Mathematically, the idea of RNN can be shown as:

$$h_{t+1} = h_t * w_h + b_h + x * w + b \tag{1}$$

where h_{t+1} means value of layers in current $t+1$ phase while h_t indicates value of layers in the last t phase, x is input and w, w_h are trainable coefficients and b, b_h are trainable biases.

Inspired by the fact that RNN need lots of memory storage, which is because we need to restore value of layer from last phase(especially, in some case, many phases in the past), novel ways to achieve RNN is introduced by Chen, called Neural Ordinary Differential Equations(NODE)(Chen et al., 2018).

In this cases, the connection along temporal sequence is built by a differential equation, instead of a linear function shown as above equation (1):

$$\frac{\partial h_t}{\partial t} = f(h_t, t, \theta) \tag{2}$$

where θ and f are given user-determined parameters and function. Starting from first phase h_0, we can define the value of layer in phase t, h_t, to be the solution of the equation. Figure (1) contrasts two approaches of both general RNN and NODE.

Defining and evaluating a model using NODE method have several benefits:

- **Memory efficiency** In the third section, we will show that we don't need back-propagating through the operations of the solvers. That means it's unnecessary to store any intermediate data from forward pass.
- **Adaptive computation** Euler's method would be one of the simplest method to solve Differential Equations.

[*]Equal contribution [1]Washington Univ. in St.Louis. Correspondence to: Weijie <weijie.gan@wustl.edu>, Zayid <zzoyelam@gmail.com>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

And many other methods for solving the problem are introduced recently(Runge, 1895; Shampine, 1987). That means we have many flexible choices to solve our NODE in neutral network.

- **Parameter efficiency** When the hidden unit dynamics are parameterized as a function in equation 2, layers from different phases are tied together. Thus, parameters are reduced in neutral network.

- **Continuous time-series models** In usual recurrent neutral, the input data are discrete observation while the proposed method can incorporate data obtained from at arbitrary times.

This report basically introduces this novel recurrent neutral network method. In section (2), related works about different solver and learning strategy of differential equations will be introduced. The section (3) will discuss NODE solvers. Experiment summary for Unsupervised Learning, Continuous Normalizing Flows and Generative Latent Function are shown in section (4). Finally, evaluation of experiements and limitations of NODE will be discussed in section (5).

2. Related Works (by Zayid)

The characteristic features of this paper relied on past work to built upon in the areas of adjusting computation time, use of reversibility to reduce storage, learning differential equations, and differentiating ODE solvers.

2.1. Adaptive Computation

Adaptive computation time was an algorithm that was discovered for the purposes of flexibly determining the amount of computational steps needed for going from input to output (Graves, 2016; Jernite et al., 2016; Figurnov et al., 2017; Chang et al., 2018). This method served as a means of improving performance of recurrent neural networks that would compute at a constant rate no matter the input.

2.2. Reversible Architecture

The modification of ResNets to have reversible architecture was a push to avoid the bottleneck of memory capacity that occurs when networks get deeper and wider (Gomez et al., 2017; Ruthotto & Haber, 2018; Chang et al., 2017). The reversible architecture allowed for more efficient design by being able to compute activations in a current layer from the next layer hence limiting the amount of storage need. This did in fact increase some parameters that were not reversible. Ultimately, the authors of this paper were definitely inspired by this, however, they decided to lift restrictions placed on the architecture of prior reversible ResNets that resulted in partitioned hidden units.

2.3. Learning Differential Equations

As a major proponent of learning differential equations from data, Raissi (2018) has connected components of machine learning to creating high complexity models. Others like Raissi (Long et al., 2017) have used neural networks with insufficient amount of data simply to build data-driven models outright. Furthermore, more research has been building to adding Gaussian Processes to fit differential equations (Raissi et al., 2018).

2.4. Differentiation through ODE Solvers

The adjoint method for differentiation through ODE solvers in not original to the authors of this paper. Existing libraries such as the dolfin library (Farrell et al., 2013) and Stan library (Carpenter et al., 2015) implement some version of the adjoint method for solving ODEs and PDEs. Still, the authors decided to expand upon these methods by using a generic vector-Jacobian product for the use of general integration of black-box ODE solvers.

3. Methods (by Weijie)

When training neutral network, we need to compute gradient for each layer by backpropagation, which means gradient passes from the last layer to the first layer. The main difficulty in the ODE method is how to perform backpropagation through ODE layers and solver.

NODE approaches this by treating the solver as a black box (i.e. we do not need figure out how to actually solve the equation (2). Then, we compute compute gradients using the adjoint sensitivity method(Chambers et al., 2006).

In adjoint sensitivity method, we need to formulate scalar-valued loss function $L()$ needed to be optimized as:

$$L(z(t_1)) = L\left(z(t_0) + \int_{t_0}^{t_1} f(z(t), t, \theta)dt\right)$$

To optimize L by gradient method(like gradient descent), we need to compute gradient with respect to θ. In first step, we should determine how value in state t, $z(t)$ affects value of loss function. It's called adjoint: $a(t) = \frac{\partial L}{\partial z(t)}$.

The paper(Chen et al., 2018) proves in the Appendix that (1) its dynamics are computed by another ODE. This ODE can be thought of as the instantaneous analog of the chain rule:

$$\frac{\partial a(t)}{\partial t} = -a(t)^T \frac{\partial f(z(t), t, \theta)}{\partial z}$$

and (2) the gradient of loss function is the integral of $\frac{\partial a(t)}{\partial t}$:

$$\frac{\partial L}{\partial \theta} = -\int_{t_1}^{t_0} a(t)^T \frac{\partial f(z(t), t, \theta)}{\partial \theta} dt \tag{3}$$

where the integral is computed starting from its final value $z(t_1)$ to initial value $z(t_0)$.

Basically, we can consider the adjoint as a "state" or "value" in each ODE layers. We accumulate information from all layers using integral operation, to estimate gradient of parameters. The figure (2) shows the whole idea discussed in this part.

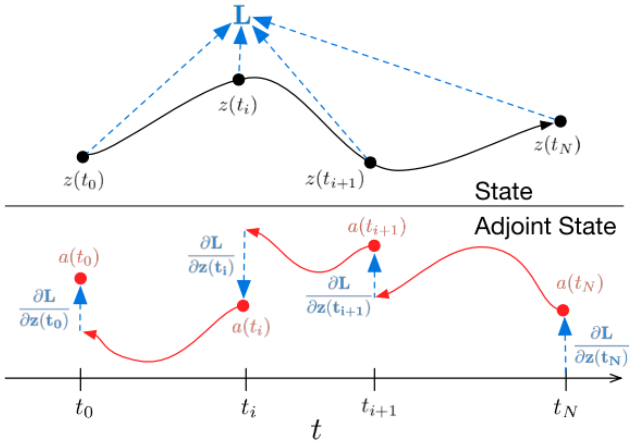

Figure 2. The UP figure shows each state in NODE affect the value of loss function and the DOWN figure indicates that how we can solve gradient of loss function by equation (3).

4. Experiments(by Zayid and Weijie)

NODE was used on three tasks to compare to other residual neural networks: 1) Supervised Learning , 2) Normalizing Flows, and 3) Generative Latent Function Time Series Model.

4.1. Supervised Learning

Supervised learning was done on the MNIST dataset using several other neural network architectures: 1-Layer MLP, ResNet, RK-Net, and ODE-Net. Performance between the models were compared on the measures of test error, number of parameters used, memory storage, and computation time. The ability to control error via tuning the tolerance of the true solution was also assessed.

4.2. Normalizing Flows

A normalizing flow is transformation of initial density into a more complex one by applying a sequence of invertible transformations until a desired level of complexity is attained. Discrete (NF) and continuous normalizing flows (CNF) were computed an compared for their ability to transform noise to data. The number of hidden units, M, for the CNF were varied and compared to an equivalent number of hidden layers, K, used for the NF.

4.3. Generative Latent Function Time Series Model

It's difficult to apply neutral network to irregularly-sampled data, such as medical records, network traffic or neural spiking data. In these cases, we can obtain only certain part of data along time series.

Our approach can be also used to address missing data using a generative time-series model. We can formulate each observed data in phase t as state x_t. All z_t connected with a differential equations with is similar with NODE, are time-continues data we need to solve. On the other word, we only obtain some data x_t from z.

The observed data is related with state h by likelihoods: $x_{t_i} = p(x|z_t, \theta)$.

5. Discussion(by Zayid)

Overall, the authors evaluated the use of Black-Box ODE solvers for time-series modeling, supervised learning, and density estimation with flexible and fast computation. Also, a scaleable version of continuous-time normalizing flows was developed.

It is known that the key benefits of using ODE solvers is memory efficiency, adaptive computation, and parameter efficiency. One of the ways of achieving such a feat stems from the adjoint sensitivity method that computes gradients by solving a second, augmented ODE backwards in time, and is applicable to all ODE solvers. This approach scales linearly with problem size, has low memory cost, and explicitly controls numerical error.

In the supervised learning setting, ODE-Nets and RK-Nets achieved similar performance as the ResNet, while using fewer parameters which is critical in demonstrating how useful ODE-Nets can potentially be. Then, when experimenting with normalizing flows the authors observed that despite increasing the depth,K (layers) of the NF and the width, M (hidden units) of the CNF result in better normalizing flows. This is notable because increasing the width of of a neural network is easier than increasing the depth. Additionally, CNFs are reversible, so training can be done on a density estimation task and still be able to sample from the learned density efficiently. Lastly, in these classification and density estimation experiments, we were able to reduce the tolerance to 1e-3 and 1e-5, respectively, without degrading performance.

Finally, exploring time-series models, the authors found potential in NODE for applying neural networks to irregularly-sampled data such as medical records, network traffic, or neural spiking data that is rather difficult. Difficulties arise due missing data and ill-defined latent variables. However,

missing data can be addressed using generative time-series models with NODE.

Granted NODE reveals exciting features, there were some limitations the authors disclosed. Mini-batching was shown to be a less straightforward approach here than for standard neural networks. In some cases, controlling error on all batch elements together required evaluating the combined system K times more often than if each system was solved individually. When using ODEs it is also important to verify that a unique solution does in fact exist. The authors claim to verify this via Picard's Existence Theorem (Coddington & Levinson, 1955) only if the differential equation is uniformly Lipschitz continuous in z and continuous in t. This theorem holds for the authors' proposed model only if the neural network has finite weights and uses Lipshitz nonlinearities, such as tanh or relu. Moreover, the potential for introducing numerical error into the backward propagation. The authors recommended in the future that if the reconstructed trajectory diverges from the original, then **checkpointing**, the act of storing intermediate values of z on the forward pass, and reconstructing the exact forward trajectory by re-integrating from those points, can alleviate these problems.

References

Carpenter, B., Hoffman, M. D., Brubaker, M., Lee, D., Li, P., and Betancourt, M. The stan math library: Reverse-mode automatic differentiation in c++. *arXiv preprint arXiv:1509.07164*, 2015.

Chambers, M. L., Pontryagin, L. S., Boltyanskii, V. G., Gamkrelidze, R. V., Mishchenko, E. F., and Brown, D. E. The Mathematical Theory of Optimal Processes. *OR*, 2006. ISSN 14732858. doi: 10.2307/3006724.

Chang, B., Meng, L., Haber, E., Tung, F., and Begert, D. Multi-level residual networks from dynamical systems view. *arXiv preprint arXiv:1710.10348*, 2017.

Chang, B., Meng, L., Haber, E., Ruthotto, L., Begert, D., and Holtham, E. Reversible architectures for arbitrarily deep residual neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. Neural Ordinary Differential Equations. *Neural Information Processing Systems*, (NeurIPS), 2018. URL http://arxiv.org/abs/1806.07366.

Coddington, E. A. and Levinson, N. *Theory of ordinary differential equations*. Tata McGraw-Hill Education, 1955.

Farrell, P. E., Ham, D. A., Funke, S. W., and Rognes, M. E. Automated derivation of the adjoint of high-level transient finite element programs. *SIAM Journal on Scientific Computing*, 35(4):C369–C393, 2013.

Figurnov, M., Collins, M. D., Zhu, Y., Zhang, L., Huang, J., Vetrov, D., and Salakhutdinov, R. Spatially adaptive computation time for residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1039–1048, 2017.

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.

Graves, A. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Jernite, Y., Grave, E., Joulin, A., and Mikolov, T. Variable computation in recurrent neural networks. *arXiv preprint arXiv:1611.06188*, 2016.

Long, Z., Lu, Y., Ma, X., and Dong, B. Pde-net: Learning pdes from data. *arXiv preprint arXiv:1710.09668*, 2017.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. Multistep neural networks for data-driven discovery of nonlinear dynamical systems. *arXiv preprint arXiv:1801.01236*, 2018.

Runge, C. Ueber die numerische Auflösung von Differentialgleichungen. *Mathematische Annalen*, 1895. ISSN 00255831. doi: 10.1007/BF01446807.

Ruthotto, L. and Haber, E. Deep neural networks motivated by partial differential equations. *arXiv preprint arXiv:1804.04272*, 2018.

Shampine, L. Solving ordinary differential equations I. nonstiff problems. *Mathematics and Computers in Simulation*, 29(5):447, 10 1987. ISSN 03784754. doi: 10.1016/0378-4754(87)90083-8. URL https://linkinghub.elsevier.com/retrieve/pii/0378475487900838.