Delay Differential Neural Networks

Srinivas Anumasa
Research Scholar, Indian Institute of Technology Hyderabad

Srijith P.K
Assistant Professor, Indian Institute of Technology Hyderabad

ABSTRACT

Neural ordinary differential equations (NODEs) treat computation of intermediate feature vectors as trajectories of ordinary differential equation parameterized by a neural network. In this paper, we propose a novel model, delay differential neural networks (DDNN), inspired by delay differential equations (DDEs). The proposed model considers the derivative of the hidden feature vector as a function of the current feature vector and past feature vectors (history) unlike only the current feature vector in the case of NODE. The function is modelled as a neural network and consequently, it leads to continuous depth alternatives to recent ResNet variants. For training DDNNs, we discuss a memory-efficient adjoint method for computing gradients and back-propagate through the network. DDNN improves the data efficiency of NODE by further reducing the number of parameters without affecting the generalization performance. Experiments conducted on real-world image classification datasets such as cifar10 and cifar100 to show the effectiveness of the proposed model.

CCS CONCEPTS

• **Computing methodologies**; • **Machine learning**; • **Machine learning approaches**;

KEYWORDS

Delay differential equations, Deep learning, Adjoint method

ACM Reference Format:
Srinivas Anumasa and Srijith P.K. 2021. Delay Differential Neural Networks. In *2021 6th International Conference on Machine Learning Technologies (ICMLT 2021), April 23–25, 2021, Jeju Island, Republic of Korea.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3468891.3468908

1 INTRODUCTION

The ability of deep learning models to capture rich representations of high dimensional data has led to its successful application in computer vision problems like image classification [1–3] and image captioning [4]. The backbone of many such tasks is a deep learning model called Residual Networks [1]. They allowed deep learning to solve complex computer vision tasks by training deep neural networks with more than 100 layers without suffering from vanishing gradient problem. ResNets achieve this by using skip connections where input at any layer is added to the output of that layer modelling the identity mapping. The feature mappings in ResNet can be considered as discretization of a continuous solution modelled using ordinary differential equations (ODE) [5]. Based on this idea, a generalization of the ResNet was introduced, called neural ordinary differential equations (NODE) [6]. NODE is an ordinary differential equation parameterized by a neural network and can be seen to generalize ResNets to arbitrarily many layers. Neural ODE has been shown to achieve a performance close to ResNet but with a reduced number of parameters. Like in ResNets, the representations learned through NODE blocks are finally mapped to the output through a fully connected neural network (FCNN). Neural ODEs are also shown to be more robust than convolutional neural networks (CNN) [7]. In this paper, we propose a novel continuous depth neural network model, delay differential neural networks (DDNN), inspired by delay differential equations (DDE). The proposed model assumes the derivative of the feature vector depends not only on the current feature vector but also on the feature vectors computed in the past. DDNN can be considered as a generalization to NODE models that utilize past feature vectors in addition to the present feature vector for modelling the feature mappings. Similar to NODE, the depth is not fixed but the number of feature vectors required from the past is fixed. The proposed DDNN model will improve upon the modelling capabilities of NODE, and further reduces the number of parameters in a NODE model. DDNN can be seen as the continuous depth generalization to deep neural network models like DenseNet [2] and MixNet [3]. They showed that designing a neural network architecture by utilizing the previously computed feature vectors can improve generalization performance of ResNets. Inspired by these recent advances in ResNet variants, we design a DDNN model, which concatenates the historical feature vector with current feature feature vector and demonstrate its usefulness on image classification data sets.

Our contributions can be summarized as follows:

- We propose delay differential neural networks (DDNN), a continuous depth deep learning model based on delay differential equations parameterized by neural networks.
- We provide a memory-efficient adjoint method for updating the parameters of DDNN during backpropagation.
- We conduct experiments on image classification datasets such as Cifar10 and Cifar100(fine) datasets demonstrating the effectiveness of DDNN.

2 RELATED WORK

NODE [6] generalizes ResNet [1] as continuous time representation of discrete feature vectors obtained from a Residual block transformations. In [6], a memory efficient adjoint method was proposed for learning parameters in a NODE model. Following this work, a series of approaches were proposed [8–11] to address the learning issues in NODE. It was observed that generalization performance of NODE

ICMLT 2021, April 23–25, 2021, Jeju Island, Republic of Korea

ACM ISBN 978-1-4503-8940-2/21/04…$15.00
https://doi.org/10.1145/3468891.3468908

got degraded when it was made more deeper by concatenating multiple NODE blocks. The issue was caused due to inconsistency among the feature vectors computed during forward propagation and backpropagation which lead to incorrect computation of gradients. Adaptive checkpoint and modified adjoint approaches [8, 11] were proposed to address the computation of accurate gradients. On the other hand, Augmented NODE [9] improves the function modelling capability of the NODE by augmenting the space (adding additional dimensions) in which NODE is solved. In contrast, we propose to consider feature representations from previous time steps to improve the function modelling capability in NODE.

3 BACKGROUND

Let $\mathcal{D} = \{X, \boldsymbol{y}\} = \{(\mathrm{x_i}, y_i)\}_{i=1}^{N}$ be set of training datapoints with $\mathrm{x_i} \in \mathcal{R}^{\mathcal{D}}$ and $\mathrm{y_i} \in \{1, \ldots, \mathrm{C}\}$. We denote the test data point as $(\mathrm{x}_*, \mathrm{y}_*)$. The aim is to learn a function which maps from input $\boldsymbol{x}$ to a class label y from the data so that it will have good generalization performance. We assume the function learnt using a neural network model to be denoted by f. The hidden layers in a neural network are denoted as $\boldsymbol{z}$. In this section, we will provide background information required to understand the proposed model.

3.1 Neural Ordinary Differential Equations (ODE)

Deep learning models learn a sequence of transformation through different spatial domain to map input $\boldsymbol{x}_i$ to output y_i. In a ResNet block computation of a hidden layer representation can be expressed using the following transformation.

$$\boldsymbol{z}(t+1) = \boldsymbol{z}(t) + f(t, \boldsymbol{z}(t), \theta(t)) \tag{1}$$

Where, $\boldsymbol{z}(t)$, $\boldsymbol{z}(t+1)$ are feature vectors with $t \in \{0 \ldots T\}$ and f is neural network parameterized by parameters $\theta(t)$. If we use the same transformation at every step, the expression in Equation 1 is equivalent to computing the trajectory of an ordinary differential equation (ODE) using Euler method with step size one. This forms the basis of neural ODEs (NODE) [6] and it can be seen as solving an ordinary differential equation.

$$\frac{d\boldsymbol{z}(t)}{dt} = f(t, \boldsymbol{z}(t), \theta) \tag{2}$$

Given initial feature vector $\boldsymbol{z}(0)$, the final feature vector $\boldsymbol{z}(T)$ can be computed by solving the ODE. On the final feature vector $\boldsymbol{z}(T)$, necessary transformations are applied using a fully connected neural network (FCNN), involving multiple linear mapping and activation to predict class probabilities. Parameters of the NODE model are learnt by computing gradients of the parameters with respect to loss function using memory-efficient adjoint approach.

3.2 Delay Differential Equations (DDE)

In ODEs, the derivative of the state vector with respect to time is modelled as a function of the current state vector at that time.

$$\frac{d\boldsymbol{z}(t)}{dt} = g(t, \boldsymbol{z}(t)) \tag{3}$$

where $\boldsymbol{z}(t) \in R^d$, $g : R^{d+1} \rightarrow R^d$. The solution to the ODE, i.e., the state vector value at time t is obtained as a solution to the initial value problem (IVP) (with initial state $\boldsymbol{z}(0)$) using numerical techniques like Euler or Runge-kutta method. In delay differential equations (DDE), the derivative of the state vector with respect to time can be seen as a function of the current state vector and previously computed state vectors. DDEs fall under the class of infinite-dimensional dynamical systems, where the function evolves in time. A typical delay differential equation as shown in Equation 4,

where, $\tau_1, \tau_2 \ldots \tau_n$ are constant delay values and $\phi(t)$ is the history function.

$$\frac{d\boldsymbol{z}(t)}{dt} = g(t, \boldsymbol{z}(t), \boldsymbol{z}(t-\tau_1), \ldots, \boldsymbol{z}(t-\tau_n)), \quad t > t_0$$
$$\boldsymbol{z}(t) = \phi(t), \quad t \leq t_0 \tag{4}$$

In this case, the change in state vector not only depends on the current state vector but also on the previous n state vectors $\boldsymbol{z}(t-\tau_1), \ldots, \boldsymbol{z}(t-\tau_n)$. For a DDE, due to this dependence, initial value of the state $\boldsymbol{z}(t_0)$ alone is not enough to compute the trajectory. It also requires a history function which allows it to compute the state values at time $t \leq t_0$. In general, the delays can be a function of state vectors or can be any function of time. However, we restrict to positive valued delays. DDEs can be solved as a sequence of IVPs for ODEs (considering the history function) and can be shown to have a unique solution in the interval of interest. DDE with constant delays have a solution which behave differently from the ODEs. In contrast to ODEs which produce smoothly varying trajectories, DDE trajectory can exhibit abrupt changes and jumps. In fact, one can show that DDEs exhibit a first order discontinuity at the initial point and higher order discontinuities in the intervals which are multiple of time delays. Hence, the abrupt changes in the DDE trajectory may get smoothen out eventually. This behavior could be quite useful in modelling the changes in feature representations in neural networks, and motivates us to develop delay differential neural networks which we explain in detail in the following section.

3.3 Numerical Method for DDE

The numerical methods to solve DDE can be obtained by extending the numerical methods to solve an ODE such as, Euler method and Runge-Kutta (RK) methods [12] by considering the delay term. For instance, a q^{th} order, fixed step size, explicit RK method to solve a DDE [13] is shown as in Equation 5. Where, g is a DDE, h is the step size and values of the parameters $\{b_i, c_i\}_{i=1}^{q}$, $\{a_{ij}\}_{i=1,j=1}^{q,\ i-1}$ depend on the order q of RK method.

$$\boldsymbol{z}(t+h) = \boldsymbol{z}(t) + h\sum_{i=1}^{q} b_i k_i$$
$$k_i = g\left(t + c_i h, \boldsymbol{z}(t) + h\sum_{j=1}^{i-1} a_{ij} k_j, \boldsymbol{z}(t + c_i h - \tau)\right) \tag{5}$$

In this work, to solve a DDNN model we opted for RK12 adaptive numerical method, which is a 2 stage numerical method as shown in Equation 6. With values, $b_1 = 0.5$,$b_2 = 0.5$, $a_{21} = 1$, $c_1 = 0$, $c_2 = 1$. Where, h_t is the step size accepted by the solver, which depends on the computed error. For computing history term $z(t + h_t - \tau)$ at any time in the past, we apply linearly interpolation over the computed feature vectors in the past and their accepted time values.

$$\boldsymbol{z}(t + h_t) = \boldsymbol{z}(t) + h_t(0.5k_1 + 0.5k_2)$$

$$k_1 = f(t, z(t), z(t-\tau)),\ k_2 = g(t + h_t, z(t) + h_t k_1, z(t + h_t - \tau))$$
$$error = -h_t 0.5 k_1 + h_t 0.5 k_2 \quad (6)$$

4 DELAY DIFFERENTIAL NEURAL NETWORKS

Neural ODEs have been shown to be quite useful in many tasks which were earlier solved using ResNets. They allowed the neural networks to grow arbitrary deeper without increasing the number of parameters. However, their generalization performance was not close to ResNets, and several approaches were proposed to improve upon it. In this section, we propose an approach, delay differential neural networks (DDNN), which would improve the function modelling capability of NODE and further reduces the number of parameters in the model. The proposed approach utilizes the past feature vectors to compute the next feature vector, through the framework of delay differential equations. The proposed delay differential neural networks (DDNN) consider the following dynamics to model the feature representations $z(t)$.

$$\frac{dz(t)}{dt} = f(t, z(t), z(t-\tau), \theta), \quad t > t_0,\ \tau > 0$$
$$z(t) = \phi(t), \quad t \leq t_0 \quad (7)$$

Where f is a neural network parameterized by θ which consider not only the current feature vector but also the previously computed feature vector at time $t - \tau$. Here we restrict ourselves to a single constant delay, while the proposed framework can be generalized to multiple delays as well. The initial representation $z(0)$ is obtained through some basic operation (e.g. downsampling) of the input $\boldsymbol{x}$. This is transformed through DDNN defined in Equation 7 to obtain the final representation $z(T)$ at some time T. The final representation $z(T)$ is transformed using a fully connected neural network to obtain the output y. Figure 1 provides a pictorial representation of the transformations in a DDNN model. The final representation $z(T)$ is obtained by solving the initial value problem defined by the DDNN as shown in Equation 7. The solution to the DDNN model can be obtained using numerical techniques such as the Runge-Kutta method discussed in Section 3.2. Solving DDNN requires one to know the values of $z(t)$ for $t \leq t_0$, and is provided by the history function $\phi(t)$. We consider the history function $\phi(t) = z(0)$ for $t \leq t_0$. The key component in the proposed DDNN model is the function f, which consider the current and past feature vectors. In our model, we consider concatenation of the feature vectors.

4.1 Concatenation of Feature Vectors

In our DDNN model, feature vector $z(t-\tau)$, current feature vector $z(t)$ are concatenated, and this concatenated feature vector is provided as an input to the function f. This is loosely inspired from the DenseNet architecture which does the concatenation of feature vectors. For the DDNN model, the size of the input and output feature vectors fed to the neural network transformation are not the same. The neural network is designed in such a way, that the incoming higher dimensional feature vector is transformed to a lower dimensional feature vector. This process is continued until the time component t reaches T. In our experiments for image classification, a DDNN block is constructed based on convolutional

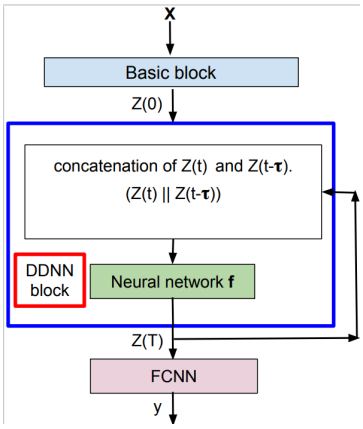

Figure 1: Feature transformations in a DDNN model considering concatenation "||" operation over past and present feature vectors

neural networks (CNN) which deal with tensor valued features with multiple channels. These features are concatenated across channel dimension and the CNN computes a tensor of reduced channels. An advantage of the DDNN is that it can lead to a reduced number of parameters with a competitive generalization performance compared to NODE and ResNet. With DDNN, one can work with a half the output size compared to NODE but without affecting the performance due to its consideration of previous feature vectors. The dynamics of the DDNN can be written as,

$$\frac{dz(t)}{dt} = f(t, (z(t) || z(t-\tau)), \theta), \quad t > t_0$$
$$z(t) = \phi(t), \quad t \leq t_0 \quad (8)$$

Where, "||" stands for concatenation operator. The parameter τ here is a hyperparameter which is determined through the validation data.

5 BACKPROPAGATION USING ADJOINT METHOD

We discuss an approach to learn the parameters in the DDNN model. The parameters associated with the neural networks of various DDNN blocks can be learnt through back-propagation. To apply back-propagation, standard loss functions for regression and classification can be used. The output $\hat{y}$ obtained from the FCNN block is used to compute the loss function L. The loss function can be taken as the least square's loss for regression problems and cross-entropy loss for classification problems, for e.g., $L(y, \hat{y}) = -y \log p(\hat{y}) - (1-y) \log 1 - p(\hat{y})$. However, applying standard back propagation to learn the DDNN parameters, requires computing and storing the gradients with respect to every state and is not memory efficient. We use the adjoint method, a memory-efficient way of computing the gradients for learning the parameters in the DDNN model. This is similar to the adjoint method used in the NODE [6, 8, 11] models, but we derive it to the case with delay differential equations. For the DDNN model, we extend the adjoint method in [14] for learning the parameters of a DDE. Algorithm [1] provides a snippet of adjoint method used for the DDNN model.

Table 1: Validation accuracy of DDNN models, under different delay values. All the models are trained for 90 epochs

Dataset	Delay									
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Cifar10	92.98	93.44	92.96	93.02	93.22	92.88	92.94	92.88	93.18	93.24
Cifar100	72.20	72.14	71.92	72.18	72.84	73.26	71.32	72.52	72.58	72.94

Algorithm 1 Adjoint method during backpropagation for DDNN model

Initialize $\alpha^T(T)$, $\frac{dL}{d\theta} = 0$
For N to 1:
(1) Compute the value of the function f at t_i and $(t_i + \tau)$ for computing $f_{\boldsymbol{z}}(t_i)$, $f_\theta(t)$ and $f_{\boldsymbol{v}}(t_i + \tau)$.
(2) local backward, update $\alpha(t_i)$ and $\frac{dL}{d\theta}$ according to discretization as shown in Equations 9 and 10.

The adjoint $\alpha^T(t) = -\frac{\partial L(y,\hat{y})}{\partial \boldsymbol{z}(t)}$ models the derivative of the loss function with respect to a feature vector $\boldsymbol{z}(t)$ at time t. The adjoint can be used to compute the gradient of parameters θ with respect to loss L. We can model the dynamics of the adjoint $\alpha^T(t)$ over time and is given by the following delay differential equation,

$$\frac{d\alpha^T(t)}{dt} = -\alpha^T(t) f_{\boldsymbol{z}}(t) - \alpha^T(t+\tau) f_{\boldsymbol{v}}(t+\tau) \tag{9}$$

where $f_{\boldsymbol{v}}(t)$ is $\frac{\partial f(t,\boldsymbol{z}(t),\boldsymbol{z}(t-\tau),\theta)}{\partial \boldsymbol{z}(t-\tau)}$, $f_{\boldsymbol{z}}(t)$ is $\frac{\partial f(t,\boldsymbol{z}(t),\boldsymbol{z}(t-\tau),\theta)}{\partial \boldsymbol{z}(t)}$. The value of the adjoint $\alpha^T(t)$ at time t can be found by solving the DDE in Equation 9 backwards in time. The numerical methods for DDE based on Runge-Kutta methods discussed in Section 3.3 can be used to solve Equation 9. Computing the gradients of the parameters θ with respect to loss L requires evaluating an integral, which depends on $\boldsymbol{z}(t)$ and $\alpha^T(t)$.

$$\frac{dL}{d\theta} = \int_0^T \alpha^T(t) f_\theta(t)\, dt - \int_{-\tau}^0 \alpha^T(t+\tau) f_{\boldsymbol{v}}(t+\tau) z_\theta(t)\, dt \tag{10}$$

Where $f_\theta(t)$ is $\frac{\partial f(t,\boldsymbol{z}(t),\boldsymbol{z}(t-\tau),\theta)}{\partial \theta}$ and $z_\theta(t)$ is $\frac{\partial \boldsymbol{z}(t)}{\partial \theta}$. Our approach uses adjoint method for computing gradients during backpropagation and uses adaptive checkpoint [11] framework for computing accurate gradients. The adaptive checkpoint approach requires storing the feature vectors computed during forward propagation and is then utilized during backpropagation. This is shown to be numerically more accurate than naive adjoint method [6]. Assuming N to be the number of states evaluated, the approach requires a memory complexity of $O(N)$ for each DDNN block.

Table 2: Comparing test accuracy of ACA_NODE, ResNet18 models against DDNN model. For the column DDNN, along with the test accuracy of the best model based on best validation accuracy, delay values are also given.

Dataset	ACA_NODE	ResNet18	DDNN
Cifar10	94.16	94.28	94.30(0.2)
Cifar100	74.60	76.08	74.34(0.6)

6 EXPERIMENTS

We conduct experiments for image classification on the real-world datasets such as Cifar10 and Cifar100, and compare against ACA_NODE which is the state-of-the-art NODE model [11] and ResNet18 models.

6.1 Image Classification

Under image classification, we conducted experiments on Cifar10 and Cifar100 datasets. Cross-entropy loss is used to compute gradients with respect to the parameters. We split the dataset of 60000 samples into 50000 as training, 5000 as validation data, and 5000 as test data. For DDNN models, validation data will be used to select the best delay hyperparameter. For training the models we follow a similar setup as discussed in [11], where all the models are trained for 90 epochs on both Cifar10 and Cifar100. Scheduled decay learning rate is used with an initial learning rate as 0.1 with decay as 0.1. The learning rate is rescheduled at epoch 30 and 60.

The architecture design of DDNN is different when compared with ACA_NODE. In ACA_NODE model, dimension of the expected input feature vector (Tensor) to ACA_NODE block is same as the computed feature vector (Tensor) dimension. But for DDNN, the

input tensor and output tensor differ in terms of number of channels. Let C be expected number of channels by both the models ACA_NODE and DDNN. But the computed tensor is different in terms of number of channels. DDNN produces a tensor with $C/2$ number of channels unlike C for ACA_NODE. This results in a reduced number of parameters for DDNN (7 Million) compared to ACA_NODE model (11 Million).

We trained DDNN models with different delay values and compared their validation performance in Table 1. Test accuracy of the best performing DDNN model trained on Cifar10 and Cifar100 datasets are given in Table 2 and compared against ACA_NODE and ResNet18. We found that the DDNN model performed better than other models on the Cifar10 dataset, achieving a test accuracy of 94.30 for a delay value of 0.2.

7 CONCLUSION

In this work we proposed a novel continuous depth deep learning model, delay differential neural networks (DDNN), based on the principles of delay differential equations. We provided an architecture, DDNN which concatenates considering the present and past feature vectors. We also discussed an adjoint method which provides a memory efficient way to learn parameters in DDNN. We discussed the performance of DDNN models on image classification datasets such as Cifar10 and Cifar100. We showed that under concatenation operation, DDNN model with reduced number of parameters, performs well without affecting the generalization performance. As a future work, we develop more general DDNN models and apply to other problems in computer vision.

REFERENCES

[1] 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, pages 770–778, 2016.

[2] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Wein-berger. Densely connected convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 4700–4708,2017

[3] W. Wang, X. Li, T. Lu, and J. Yang. Mixed link networks. In Proceedings of the 27th International Joint Conference on Artificial Intelligence, pages 2819– 2825, 2018.

[4] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Showand tell: A neural image caption generator. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Jun 2015.

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

[6] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. In Advances in neural information processing systems, pages 6571–6583, 2018.

[7] Y. Hanshu, D. Jiawei, T. Vincent, and F. Jiashi. On robustness of neural ordinary differential equations. In International Conference on Learning Representations, 2019.

[8] A. Gholami, K. Keutzer, and G. Biros. Anode: unconditionally accurate memory-efficient gradients for neural odes. In Proceedings of the 28th International Joint Conference on Artificial Intelligence, pages 730–736. AAAI Press, 2019.

[9] E. Dupont, A. Doucet, and Y. W. Teh. Augmented neural odes. In Advances in Neural Information Processing Systems, pages 3140–3150, 2019.

[10] C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios. Neural spline flows. In Advances in Neural Information Processing Systems, pages 7511–7522, 2019.

[11] J. Zhuang, N. Dvornek, X. Li, S. Tatikonda, X. Papademetris, and J. Duncan. Adaptive checkpoint adjoint method for gradient estimation in neural ODE. In Proceedings of the 37th International Conference on Machine Learning, Proceedings of Machine Learning Research, pages 11639–11649. PMLR, 2020.

[12] L. F. Shampine. Numerical solution of ordinary differential equations. CRC Press, 1994.

[13] F. Ismail, R. A. Al-Khasawneh, M. Suleiman, *et al.* Numerical treatment of delay differential equations by runge-kutta method using hermite interpolation. MATEMATIKA: Malaysian Journal of Industrial and Applied Mathematics, 18:79–90, 2002.

[14] J. Calver and W. Enright. Numerical methods for computing sensitivities for odes and ddes. Numerical Algorithms, 74(4):1101–1117, 2017.