Reimplementation of the Classical Neural Ordinary Differential Equation Using Modern Computational Tools

Wasif Ul Islam [1]

1. Introduction

Residual Neural Networks (RNN) provided a significant improvement on performance regarding deep layered neural networks. The performance gains were obtained from introducing residual elements applied to the ReLu operations within the hidden layer. This introduced robustness against the vanishing/exploding gradient problem that most classical neural networks face (Shorten, 2019).

However, there are some application of RNNs that still did not provide the ideal flexibility, in terms of training models for data involving time-series data. Since RNNs have discrete hidden layers and residual components to the ReLu operations performed to the hidden layers, it can interfere with learning dependencies which are sensitive to time (Walther et al., 2023).

A different class of neural networks look to satisfy the drawbacks that the previously mentioned network contain: Neural Ordinary Differential Equations (NeuralODE). The prime benefit that NeuralODEs provide is it is naturally suited for modelling continuos time data. In addition, it can provide computational flexibility in terms of configurability in forward-pass and backpropagation.

This paper's objective is to provide a brief literature review into works that have enabled the use of NeuralODEs and discuss the advantages and disadvantages related to the use of NeuralODEs. As many pieces of work focus on performance of NeuralODEs on time-series datasets, a short experiment involving the classification of the MNIST dataset using a proof-of-concept NeuralODE model is presented along with comparisons of performance with a typical proof-of-concept implementation of a residual neural network.

[1]School of Electrical and Computer Engineering, Purdue University, West Lafayette, Indiana USA.

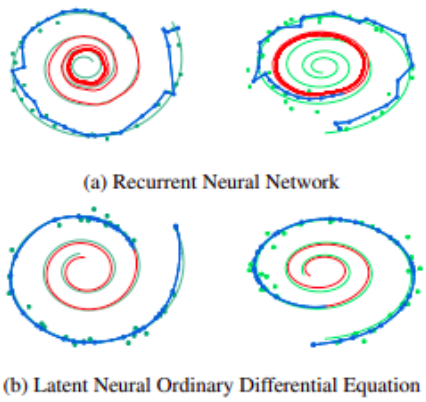
Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

2. Related Works

2.1. Article: Neural Ordinary Differential Equations

The concept of NeuralODEs have been discussed before the prescribed implementation by the paper *Neural Ordinary Differential Equations*. However, the primitive approach to the implementation of NeuralODEs had very large computational resource consumption, to the point where training the model on a substantial scale was not feasable. Since the NeuralODE consist of one continous layer with specific time steps for numerical integration (Derivation of Neural ODE vs. Classical NN is explained further below), performing backpropagation on NeuralODEs would be taking the gradient across each of the time steps. Since solving ODEs require small timesteps for accurate representation of the dynamics that is being learned, ordinary back-propagation become as resource dependant as a ResNet with hundreds of layers.

The major contribution for the respective paper is the adjoint sensitvity method, which allows for the back-propagation of NeuralODE block at near constant space-complexity (He et al., 2015). A comparison between a Recurrent Neural Network with 25 hidden-layers and the proposed NeuralODE architecture was presented with time-series data.



(a) Recurrent Neural Network

(b) Latent Neural Ordinary Differential Equation

Figure 1. Comparison Between Recurrent Neural Network and NeuralODE on Time-Series Data (He et al., 2015)

2.2. Article: Augmented Neural ODE

The family of NeuralODEs have been extended from the previous contribution to make it more generalizable for existing framworks. The paper *Augmented Neural ODEs* proposes several postulates. The first assertion is that the classical implementation of NeuralODEs with the use of adjoint sensitivity method has limitations regarding representation of functions containing intersecting vector flows. The second assertion involves the existance of output space being rigid to the input space. The proposed solution to the problem was to introduce higher-dimensionality to the output space. This allows the ODE learning algorithm to lift points into additional dimension without having differential collisions (Dupont et al., 2019). The higher dimensionality mapping allowed for functions that were not representable by the classical NeuralODEs, available for training.

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix}\mathbf{h}(t)\\ \mathbf{a}(t)\end{bmatrix} = \mathbf{f}(\begin{bmatrix}\mathbf{h}(t)\\ \mathbf{a}(t)\end{bmatrix}, t), \qquad \begin{bmatrix}\mathbf{h}(0)\\ \mathbf{a}(0)\end{bmatrix} = \begin{bmatrix}\mathbf{x}\\ \mathbf{0}\end{bmatrix}$$

Figure 2. Activation Function Augmented with Higher Dimensionality (Dupont et al., 2019)

2.3. Article: How to Train Your Neural ODE

Beside the adjoint sensitivity method, developed by the authors of the paper *Neural Ordinary Differential Equations*, parallel efforts were also taken to address concerns of high resource usage of back-propagation of NeuralODEs. The paper *How To Train Your Neural ODE* proposes a method to reduce the computational complexity of back-propagation in time-series data predictions, by introducing regularization terms to the loss-function (Finlay et al., 2020). This allows for simpler solutions that can be sufficient while limiting NFEs (Number of Function Evaluation) to a reasonable number. The first regularization term penalty enourages particles to travel with one-dimensional translation by using optimal transport mapping, while the second regularization term enforces limitations of the vector-field Jacobian to provide force experienced to be as constant as possible. Such constraint of dynamics allow for simpler solutions with less computational complexities (Finlay et al., 2020).

2.4. A Comparative Derivation of Neural ODE

Conceptially, NeuralODEs are an extension of Residual Neural Networks. As mentioned previously, the difference between the respective family of neural networks is the continuos nature of NeuralODEs vs. the discrete nature of Residual Neural Networks. To show the difference, we will go from one forward-pass calculation of a Residual Neural Network to a NeuralODE.

Residual Neural Network

$$h_{t+1} = ReLu(W_t h_t + b_t) + h_t$$

The additional hidden layer term added after the activation function is the residual component of the Residual Neural Network.

Containing the activation function with standard notations, we can rewrite the equation as:

$$h_t = f(h_t, \theta_t) + h_t$$

We can convert the following equation to its differential form by expressing it in terms of its limits

$$h_{t+1} - h_t = f(h_t, \theta_t)$$

If we reduce the change between one hidden layer to the next to be infinitesimally small, we can express the equation as:

$$\frac{h_{t+\delta} - h_t}{\delta} = f(h_t, \theta_t)$$

Taking the limit of δ and setting it to zero provides us with the differential form of the equation:

$$\lim_{\delta \to 0} \frac{h_{t+\delta} - h_t}{\delta} = f(h_t, \theta_t)$$

$$\frac{\mathbf{dh_t}}{\mathbf{dt}} = \mathbf{f}(\mathbf{h_t}, \theta_\mathbf{t})$$

- Legend:
  - h_t: Hidden Layer t
  - W_t: Weight Matrix t
  - b_t: Bias Vector t
  - θ_t: Layer Parameters t
  - δ: Change in Hidden Layer (Infinitesimally Small)

3. Experiment

3.1. Defining the Experiment

An implementation of the Proof-of-Concept NeuralODE is developed. The workflow is modified from a typical NeuralODE implementation to accomodate visual classification of the MNIST dataset. The same image classification workflow is performed with a pre-built modified ResNet-18 to classify the MNIST dataset. Classification accuracy is then compared between the two models from the same testing dataset.

3.2. Defining Models and Assumptions

Neural ODE

The NeuralODE implementation is a modified adaptation of the implementation provided by Mikhail Surtsukov. Variations were made to the convolutional layer, along with changing the the backpropagation method to use the adjoint sensitivity method from **PyTorch** using the Runge-Kutta fixed step solver (Surtsukov, 2019; AI, 2023). The model architecture is as follows:

- **Convolutional Layer (2D)**
  - Input: 1 Channel, 28x28 Image
  - Convolve: 64 Filters, 3x3 Kernel, Stride 1
  - Norm: Normalize Based on Transport Object [Mean, St. Dev]
  - ReLU: Activation Function
  - Convolve: 64 Filters, 3x3 Kernel, Stride 1
- **ODE Layer**
  - Input: 64 Filters, 64*12*12 Tensor
  - Time Update
  - ODE: ODEFunc
    - Convolutional Layer (2D)
    - ReLU: Activation Function
    - Statistical Norm [Mean, St. Dev]
    - Convolutional Layer (2D)
    - ReLU: Activation Function
    - Statistical Norm [Mean, St. Dev]
- **Average Pool Layer**
- **Resize: Flatten to 64 Parameters**
- **Fully Connected Layer: 10 Labels**

ResNet-18

The ResNet-18 model is a pretrained and pre-built model that is used from the **PyTorch** Library. The model architecture generally uses 18 residual layers. ResNet-18 is an adaptation of the ImageNet architecture used to classify images with 3 channels and 224x224 pixels (AI, 2023). The model architecture had to be slightly modified to accomodate the MNIST dataset, which has 1 channel and 28x28 pixels. The first convolutional layer was edited to have 1 channel (Shorten, 2019).

3.3. Training Method

Hyperparameter Selection

Hyperparameters [Learning Rate, Epochs, Batch Size] were selected based on standards used for prototyping convolutional models. The batch size was set to 64, the learning rate was set to 0.0001/.001 for NeuralODE and ResNet-18 respectively, and the number of epochs was set to 10 (Stoyanov, 2019).

Initially, the learning rate for NeuralODE was set to 0.001, but the model was not able to converge to a solution as it was overshooting the argmin of the loss function. The learning rate was reduced to 0.0001, and the model was able to converge.

As mentioned previously, the **PyTorch** library was used to train the NeuralODE and ResNet-18 models. The training method is as follows:

- Define Data-Transform Object for ResNet Dimension and Normalization [Mean, St. Dev]
- Download MNIST Dataset using PyTorch and save it to directory (Train, Test)
- Define Data-Loader Objects for Train and Test Datasets with batch size, and whether to shuffle
- Model is defined (For ResNet, modification made to first convolutional layer)
- Optimizer [**Adam**] and Loss Function [**Cross-Entropy**] is defined

3.4. Results

Both models were trained for 10 epochs. Both models performed well with accuracies in the high 90th percentile. However, the ResNet performed better across all epochs, the errors were much more consistent, and the accuracies were higher and consistent. This could be due to the fact that ResNets are inherently better at image classification than NeuralODEs, In addition, the NeuralODE did not undergo any hyper-parameter tuning, which could have improved the accuracy of the model. The simpler convolutional layers, along with the lack of pooling layers could have also contributed to the lower accuracy of the NeuralODE model.

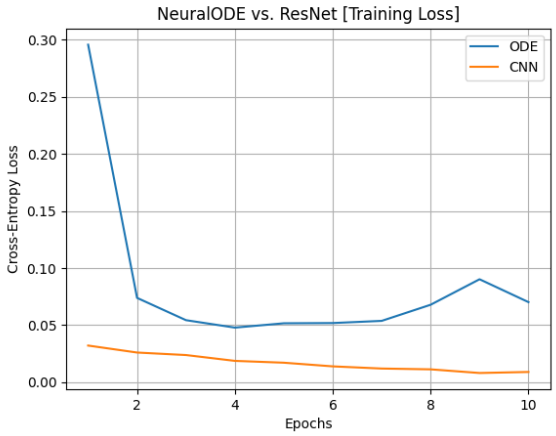


Figure 3. NeuralODE vs. ResNet-18 Training Loss

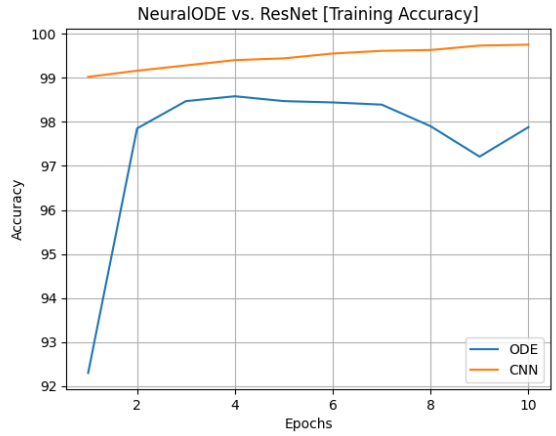


Figure 4. NeuralODE vs. ResNet-18 Training Accuracy

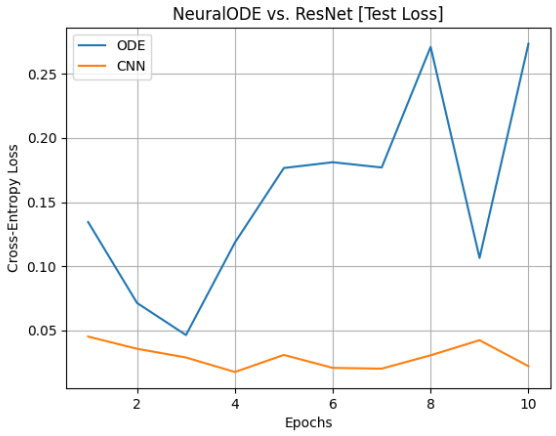


Figure 5. NeuralODE vs. ResNet-18 Testing Loss

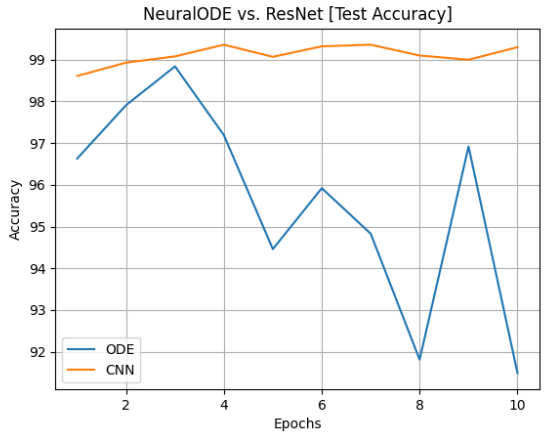


Figure 6. NeuralODE vs. ResNet-18 Testing Accuracy

4. Conclusion

For the source code that was developed and used in this project, please refer to the following link:

Source Code: https://github.com/Fallensegal/NeuralODE

A short literature review along with a simple implementation of NeuralODEs was presented using available modern tools. The NeuralODE model was compared to a ResNet-18 model over the MNIST dataset. Even though both models performed well, the ResNet-18 model performed better across all epochs, in addition to it being more consistent.

To build on top of this work. One simple approach would be to understand the effect of hyperparameter tuning on the NeuralODE model. It could be the fact that NeuralODEs behave differently than ResNets, which could require different standard hyperparameters for optimal learning. Time series experiments could also be performed to highlight its strenght vs. a convolutional model.

For further expanding on related works, an application of regularization of function dynamics along with the use of the adjoint sensitivity method for backpropagation could be explored. Such efforts could be fruitful in introducing new variations that are comparitvely resource conservative.

References

AI, M. Pytorch documentation, 2023. URL https://pytorch.org/docs/stable/index.html. Accessed: April 20th, 2023.

Dupont, E., Doucet, A., and Teh, W. Y. Augmented neural odes. Number 32, 2019.

Finlay, C., Jacobsen, J.-H., Nurbekyan, L., and Oberman, A. M. How to train your neural ode. 2020.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. pp. 770–778. IEEE, 2015.

Shorten, C. Introduction to resnets, 2019. URL https://towardsdatascience.com/introduction-to-resnets-c0a830a288a4. Accessed: April 10th, 2023.

Stoyanov, M. Optimizing hyperparameters for mnist dataset in javascript, 2019. URL https://shorturl.at/ctJX0. Accessed: April 20th, 2023.

Surtsukov, M. Neural ordinary differential equations, 2019. URL https://msurtsukov.github.io/Neural-ODE/. Accessed: April 15th, 2023.

Walther, D., Viehweg, J., Haueisen, J., and Mader, P. A systematic comparison of deep learning methods for eeg time series analysis. *Frontiers in Neuroinformatics*, 17, 2023.