Inverse Modeling on Lorenz-63 with Physics-Constrained Residual Neural Networks

Jostein Barry-Straume
Department of Computer Science
Virginia Tech
Blacksburg, VA 24060
jostein@vt.edu

Abstract

This paper seeks to find an optimal set of hyperparameters for residual networks (ResNets) of varying architectures such that the causal factors that produce the chaotic dynamical system of a given Lorenz-1963 attractor are modeled effectively. DeepXDE is used to construct the ordinary differential equations of the Lorenz system, generate training data, and define model architecture. Tensorflow is used for hyperparameter tuning, and Google Colaboratory Pro is leveraged for GPU training.

1. Introduction

Scientific Machine Learning (SciML) has arisen as a replacement to traditional numerical discretization methods. The main driving force behind this replace is neural networks (NNs). As a vehicle to approximate the solution to a given partial differential equation (PDE) or ordinary differential equation (ODE), NNs offer a mesh-free approach via auto differentiation, and break curse of dimensionality [3].

The Lorenz-1963 system is used to understand the Earth's climate and help provide insight for numerical weather and climate prediction.[1] "Inverse modeling uses the output of a system to infer the intrinsic physical parameters. Inverse problems often stand out as important in physics-based modeling communities because they can potentially shed light on valuable information that cannot be observed directly." [4] "The solution of an inverse problem can be computationally expensive... due to the potentially millions of forward model evaluations needed for estimator evaluation or characterization of posterior distributions of physical parameters. ML-based surrogate models are becoming a realistic choice since they can model high-dimensional phenomena with lots of data and execute must faster than most physical simulations." [4]

DeepXDE is a deep learning library built on top of TensorFlow, and serves as a multi-purpose tool relating to pedagogical and research problems in computational science and engineering (CSE) [3]. DeepXDE offers a scaffolding for constructing and analyzing CSE experiments. Moreover, the library contains example datasets to work on. In particular, data for a given Lorenz system is provided as a NumPy dataset. DeepXDE is used to train and test Residual neural networks on said dataset. Tensorflow's Tensorboard is used to conduct hyperparameter tuning. Neural density, the number of residual blocks, epoch times, as well as learning rate are investigated to see the impacts they have on inferring the intrinsic physical parameters of the Lorenz system.

The paper is organized as follows. Methodology behind inverse problems on Lorenz systems, residual networks, hyperparameters, physics-informed neural networks, and DeepXDE library usage is found in section 2. The experimental results are in section 3, with discussion of said results following in section 4. Although this report was conducted by a lone individual, a contribution section is found in section 5 for posterity's sake.

2. Methodology

2.1. Lorenz System

In [3], Lu et al. investigate the coefficient identification problem of the Lorenz system using a Fully Connected Neural Network (FNN). This report builds upon Lu et al.'s work in Section 4.3 of [3] by conducting the investigation using Residual networks to discover the coefficients of the Lorenz system. The inverse modeling problem for the 1963 Lorenz system involves identifying the coefficients of the following system:

$$\frac{\delta x}{\delta t} = \rho(y - x), \frac{\delta y}{\delta t} = x(\alpha - z), \frac{\delta z}{\delta t} = xy - \beta z \qquad (1)$$

With the initial conditions for the system are set as x(0), y(0), z(0) = (-8, 7, 27), and where ρ, α, and β are the three

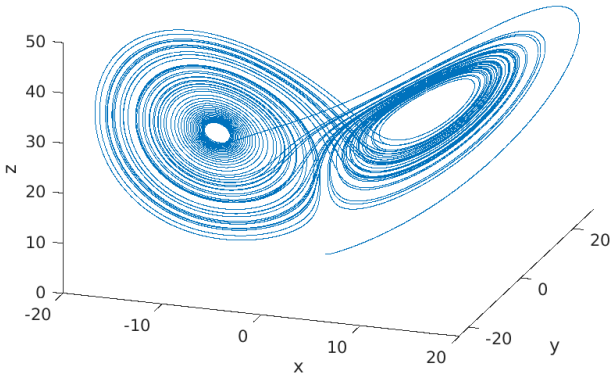

Figure 1. Example of a Lorenz-63 System

coefficients to be gleaned from the observations at given times [3]. The initial conditions for this report's experiments follow those of Lu et al. The training data resulting from these initial conditions will serve as a foundation with which to compare the results of FNNs and ResNets in future work.

The training data is generated by solving eq. (1) from the time-span of $t = [0, 3]$ using a Runge-Kutta method with ground truth coefficients of $[\rho, \alpha, \beta] = [10, 15, 8/3]$. Additionally, the training data is comprised of 400 uniformly distributed random points, with 25 equally spaced points serving as the residual points for metric evaluation [3].

An example of a Lorenz-63 system plotted in three dimensional space is shown in fig. 1.

2.2. Residual Networks

The goal of regular neural networks is to model the behavior of a given function, say, $h(x)$. Networks that use residual learning differ in that they incorporate the input x to the output of the network [2]. In doing so, residual networks then have to learn the model $f(x) = h(x) - x$ [2]. A visual explanation of this situation is illustrated in figure fig. 2.

The skip connection enables the network to advance towards the ground truth even if certain layers have not been engaged yet [2]. These skip connections, or residual units, when combined and stack together numerous times form the basis for deep residual learning. As illustrated in fig. 3, the residual unit itself can be viewed as its own neural network within the overall network, except with a skip connection [2].

2.3. Hyperparameters

One of the motivating factors of this report is to evaluate the performance of ResNets with different network configurations. In particular, the implementation and evaluation of ResNets with the following parameters were investigated:

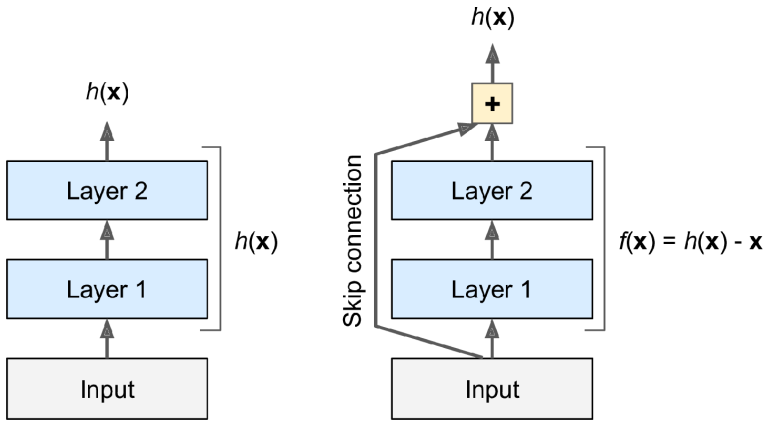

Figure 2. Standard Neural Network Learning (left) and Residual Learning (right) [2].

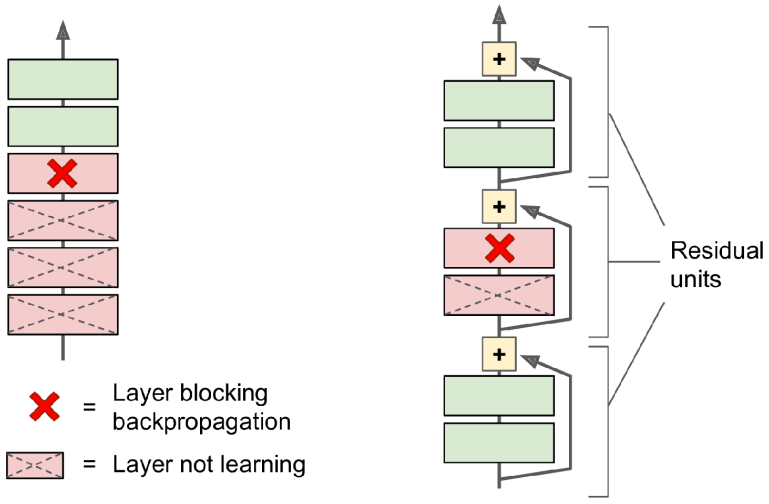

Figure 3. Standard Deep Learning (left) and Residual Deep Learning (right) [2]

ResNet Blocks	Neuron Size	Epoch	Learning Rate
3	32	10000	0.001
5	64	30000	0.002
7	128	50000	

The total number of models that result from training and evaluating all combinations of the hyperparameters is fifty-four. Tensorflow's Tensorboard is leveraged in order to efficiently and effectively grid search each combination of hyperparameters. The overall relative L2 error and loss history of each model is saved. These metrics are aggregated into a Panda's Dataframe, which is then exported to a Comma Separated File. A snapshot of this information is shown in tabular form in fig. 6. Moreover, DeepXDE enables one to visualize performance of said models via several plots.

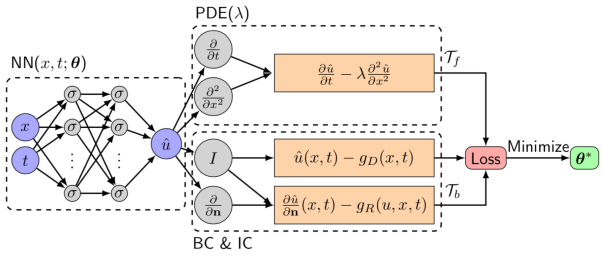

Figure 4. Example of a PINN schematic to solve a toy diffusion problem [3]

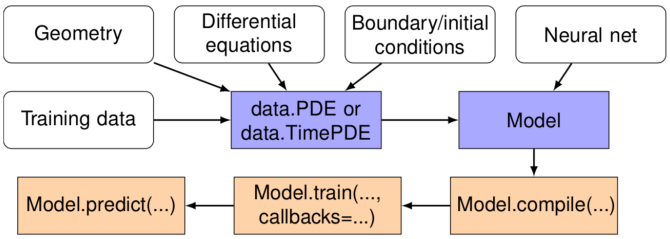

Figure 5. Flowchart of DeepXDE corresponding to algorithm 2. The white boxes define the PDE problem and the training hyperparameters. The blue boxes combine the PDE problem and training hyperparameters in the white boxes. The orange boxes are the three steps (from right to left) to solve the PDE [3].

These plots will be illustrated and discussed in section 3 and section 4, respectively.

2.4. Physics-Informed Neural Network

The general algorithm for a Physics-Informed Neural Network (PINN) is as follows below in algorithm 1. The Residual networks (ResNets) investigated in this report follow the general algorithm in algorithm 1. A visual explanation of this algorithm is represented in fig. 4.

Algorithm 1: The PINN algorithm for solving differential equations [3]

Result: Minimize loss function $L(\Theta; T)$

1. Construct neural network $\hat{u}(x;\theta)$ with parameters θ.
2. Specify two training sets T_f and T_b for the equation and boundary/initial conditions.
3. Specify a loss function by summing the weighted L^2 norm of both the PDE equation and boundary condition residuals.
4. Train the neural network to find the best parameters θ^* by minimizing the loss function $L(\Theta; T)$.

2.5. DeepXDE Procedure

The procedure detailed in algorithm 2 outlines the usage of DeepXDE to solve differential equations. A visual explanation of this heuristic is represented in fig. 5. This usage methodology was combined with Tensorflow's Tensorboard to conduct hyperparameter tuning.

Algorithm 2: The procedure to use DeepXDE to solve differential equations [3]

Result: Training and evaluation of a model on a given PDE

1. Specify the computational domain using the **geometry** module
2. Specify the PDE using the grammar of **TensorFlow**
3. Specify the boundary and initial conditions
4. Combine the geometry, PDE, and boundary/initial conditions together into **data.PDE** for a time-independent problem. When specifying training data, either the specific point locations can be set or DeepXDE can sample the required number of points from a set on a grid randomly
5. Construct a neural network using the **maps** module
6. Define a **Model** by combining the PDE problem in Step 4 and the neural net in Step 5
7. Call **Model.compile** to set the optimization hyperparameters, such as optimizer and learning rate
8. Call **Model.train** to train the network from random initialization. The training behavior is monitored by using **callbacks**
9. Call **Model.predict** to predict the PDE solution at different locations

3. Experimental Results

3.1. Best Models

Figure 6 reflects the top five performing model configurations among fifty-four different residual networks (ResNets). Of the five best models, none use seven residual network blocks in their model configuration. Likewise, the highest neural density is not represented in these five configurations. Moreover, only one of the five models used the higher learning rate hyperparameter of 0.002.

It is important to note that the top performing model is identified with a session ID of 7. Its model configuration is as follows: 3 residual unit blocks, a neural density of 64, a

learning rate of 0.001, and an epoch time of 30,000 iterations. This model configuration is placed third from the top in fig. 6. The dataframe was first sorted by L2 relative error, and then subsequently by average validation loss of the Lorenz system coefficients. The model configuration corresponding to session ID 8 technically has the lowest L2 relative error, but its average loss is very high. This indicates that such a model configuration was prone to overfitting. In contrast, the model configuration associated with session 7 has the lowest average loss among the top five models, as well as a L2 relative error that is nearly the same as the second lowest L2 relative error. With all this in mind, it is clear that the session 7 model made for the best performing model.

Figure 7 illustrates the top performing model learning the true coefficients of the Lorenz-63 system over time. This graph is important in understanding the time frame in which the model learns and infers the intrinsic physical parameters of the Lorenz-63 system. At around the 10,000 epoch level the model begins to have some semblance of understanding. By the 30,000 epoch iteration the model has essentially discovered the ground truth of all coefficients of the Lorenz system.

Figure 8 reflects the true versus predicted coefficients over the given dynamical time-span of $t = [0, 3]$ for the top performing model. In this graph, the predictions are represented by dotted lines. The fact that these lines overlap almost exactly with the ground truth is very promising. In other poor performing models, one would be able to distinguish the true values of x, y, and z (solid lines) from their predicted counterparts (dotted lines).

Figure 9 plots the prediction of the Lorenz-63 point in 3D space over the given time-span for the top performing model. This three dimensional grid should be compared to the example Lorenz-63 plot as shown in fig. 1. Comparing fig. 9 and fig. 1 explains why it is important to have extremely large number of epochs (compared to traditional machine learning endeavours). The models need more training time in a computational science and engineering setting to learn chaotic and dynamical systems. The fact that fig. 9 can only replicate the beginnings of the Lorenz system is telling. In fact, Lu et al. use an even higher epoch iteration time for their FNN implementation in [3]. To avoid overfitting issues, lower epoch times were explored in this report.

Figure 10 projects the three dimensional parameters onto two dimensional space via a contour plot for the top performing model. The segmentation of contour lines are color coded based on how close the predicted z-value is to the true coefficient value of 8/3. Inaccurate values are represented by dark blue shades. Middling accurate values are represented by tan/orange. The true value of the z-axis coefficient is approximately 2.67, and is represented as the darkest shade of red on contour the contour plot. It is interesting to see how the ground truth of the z coefficient is discovered half way through the x coefficient discovery. Perhaps this indicates that the model was able to learn certain spatial directions of the Lorenz system more easily than others. Why this would be the case is unclear at this time.

3.2. Metric Analysis

Figure 11 illustrates the average loss by number of blocks per learning rate. Of note is the smaller range of average loss produced by the smallest number of residual unit blocks (3). This is true for learning rates, as well. This might indicate that inferring the dynamical system of Lorenz-63 is carried out to a more efficient degree with a smaller number of skip connections. However, this is in direct contrast with the how and why the ResNet model architecture came to be known so well in computer vision performance.

Figure 12 shows the average loss by neural density per learning rate. The highest neural density option, combined with the smallest learning rate, resulted in distribution of average loss (among x, y, and z variables) that had the smallest range. Moreover, it is relatively normal looking all things considered.

Figure 13 reflects the average loss by epoch time per learning rate. This violin plot is somewhat misleading, because at first glance an epoch choice of 10,000 iterations appears to be the best. However, no model configuration using 10,000 epochs achieved top 5 performance. If average validation data loss was the only important metric, then perhaps the smallest epoch choice would be best. However, L2 relative error should be taken into account, and 10,000 epochs does not appear to be enough training time for the model to achieve high precision.

Figure 14 illustrates the L2 relative error by number of ResNet blocks per learning rate. Similarly to fig. 11, fig. 14 produced the lowest mean L2 relative error with the smallest number of residual units. 7 ResNet blocks with a learning rate of 0.002 made for an extremely right tail skewed distribution, favoring large L2 relative error values.

Figure 15 shows the L2 relative error by neural density per learning rate. The results of this violin plot make it difficult to glean anything useful. However, it seems as though the highest neural density resulted in a greater proportion of L2 relative errors approaching values of 0.8. The distribution of L2 relative errors with respect to the middling neural (64) and higher learning rate (0.002) density is unusually large and uniform.

Figure 16 reflects the L2 relative error by epoch per learning rate. There appears to be a loose trend of L2 relative error decreasing as the number of epochs increases. However, while the mean of the distribution goes lower as epochs increase, the overall variance increases. It is surpris-

ing that the higher learning rate (0.002) produced a distribution of L2 relative errors with a lower mean than the smaller step size choice (0.001). In general, a learning rate of 0.001 appears to result in lower L2 relative error.

4. Discussion

4.1. DeepXDE

The deep learning library DeepXDE won best paper award at this year's Society of Industrial and Applied Mathematics' (SIAM) Conference on Computational Science and Engineering (CSE21). The tool has been transformative for pedagogical and research purposes with regards to computational science and engineering (CSE). That being said, the library is severely lacking a coherent and comprehensive tutorial. As a result, a significant allocation of time was spent on combing through both open and closed GitHub issues to discern how to accomplish certain tasks.

Compounding this issue is the library's use of a Tensorflow version that is backward compatible with Tensorflow version 1.0. There is debate between the author of DeepXDE and members of the Github community whether or not DeepXDE should be fully updated. Tensorflow's Tensorboard application does not work properly as DeepXDE is currently implemented. As a result, the full usefulness of the Tensorboard application was out of reach, along with the insight provoking parallel coordinate view it provides.

Although DeepXDE is built upon Tensorflow, it has fundamentally changed the model object. In a normal Tensorflow environment, one could import and invoke the print model function to visually see how a given model was constructed. DeepXDE does something similar, but less useful from a high level perspective. DeepXDE will show the exact parameters that are available for tuning, but not in a graphical manner.

4.2. Optimization Algorithms

The original intent at the beginning of the experimental design was to investigate the impact that different optimization algorithms had on inverse modeling the Lorenz-63 system. In particular, stochastic gradient descent (SGD) and limited-memory Broyden-Fletcher-Goldfarb-Shanno box constraint (L-BFGS-B) were to be used as hyperparameters.

Unfortunately, over the course of hyperparameter tuning it became clear that using SGD as an optimizer resulted in a significant number of Not a Number (NaN) values. This left all model configurations leveraging SGD optimization as unusable.

In contrast, L-BFGS-B converged so quickly that testing different epoch levels was not possible. Additionally, the loss history was not large enough to perform metric evaluation, nor produce meaningful visual diagrams.

As a result, the Adaptive Moment Estimation (Adam) optimization algorithm was chosen as a constant among model configuration possibilities. It is suspected that L-BFGS-B holds the potential to be the better optimizer, but the steep learning curve of DeepXDE and its stark lack of documentation leaves this for a future issue to investigate.

4.3. Future Work

The results of training and testing fifty-four residual network (ResNet) model configurations provides a good foundation for continued future work. The original project proposal of this report included bench-marking performances of Lu et al.'s fully connected neural network (FNNs) in [3] with that of the ResNets investigated here. The limited runway for project implementation made this infeasible in the current scope of the project. Future work will evaluate average loss and L2 relative error metrics between FNNs and ResNets.

Generalization of physics-informed neural networks (PINNs), trained on a given set of boundary/initial conditions, and subsequently tested on differing conditions pertaining to other dynamical systems, is an open topic in the field of Scientific Machine Learning. The knowledge gained from conducting the experiments found in this report will be put to work tackling this open problem in future work.

5. Contribution

This report and its corresponding experiments were realized by a singular contributor. To foster future collaboration and promote reproducible research, fig. 17 is a hyperlink to the Google Colab notebook used to carry out the experimental results found in this report. Please note that this notebook can only be accessed by individuals belonging to Virginia Tech. If you are outside this organization, please contact the author for access.

Figure 17. Google Colab Notebook Hyperlink

References

[1] J. D. Daron and D. A. Stainforth. On quantifying the climate of the nonautonomous lorenz-63 model. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 25(4):043103, 2015.

[2] A. Géron. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, aug 2019.

[3] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis. DeepXDE: A deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, 2021.

session	num_blocks	num_neurons	learning_rate	epoch	average_loss	x_loss	y_loss	z_loss	l2_relative_error
8	3	64	0.001	50000	2.315219	3.455063	3.066611	0.423982	0.000591
20	5	32	0.001	50000	0.205576	0.063713	0.257074	0.295942	0.000784
7	3	64	0.001	30000	0.081646	0.014852	0.217615	0.012471	0.000796
2	3	32	0.001	50000	1.041203	0.483846	0.054153	2.585610	0.000935
4	3	32	0.002	30000	2.213078	0.425077	0.403318	5.810840	0.001040

Figure 6. Snapshot of Top Performing Models

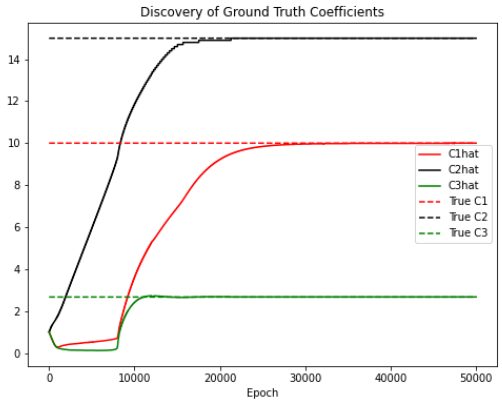

Figure 7. Lorenz System Parameter Learning Over Time

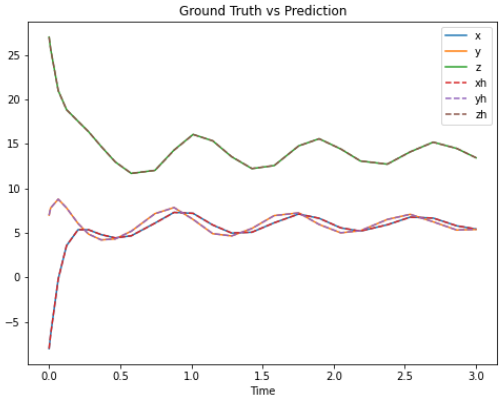

Figure 8. Lorenz System Parameter Prediction Vs Training

[4] J. Willard, X. Jia, S. Xu, M. Steinbach, and V. Kumar. Integrating physics-based modeling with machine learning: A survey, 2020.

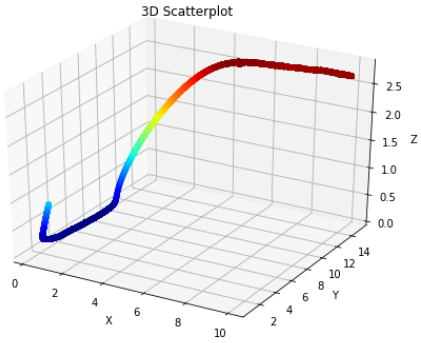

Figure 9. Lorenz System Scatter Plot

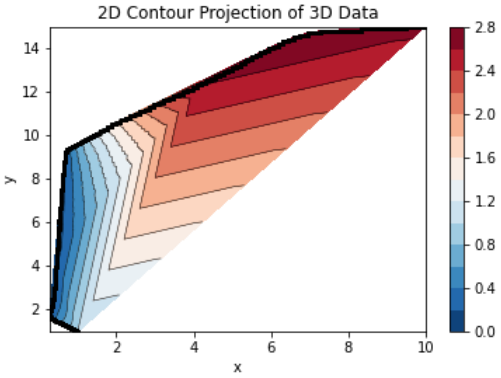

Figure 10. Contour Plot of Lorenz System Parameters

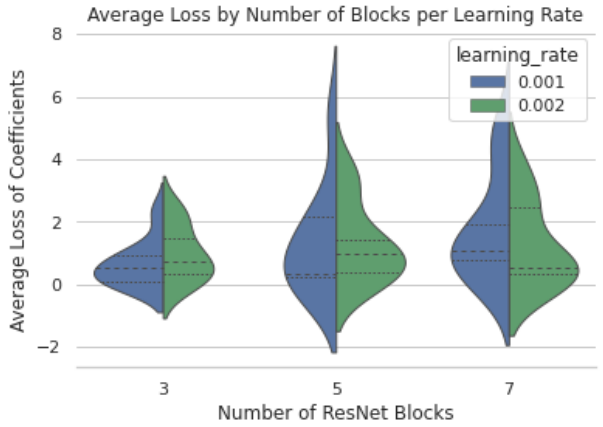

Figure 11. Average Loss by Number of Blocks per Learning Rate

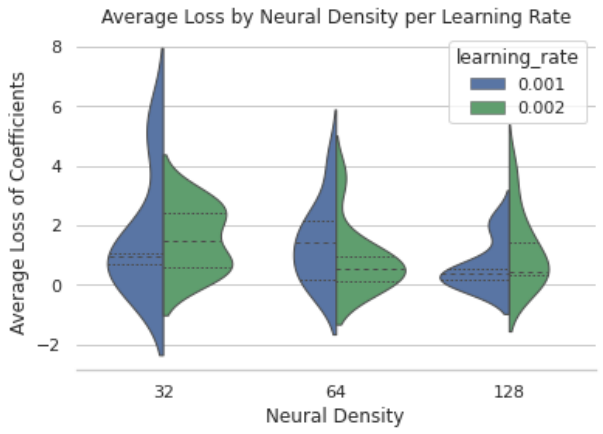

Figure 12. Average Loss by Neural Density per Learning Rate

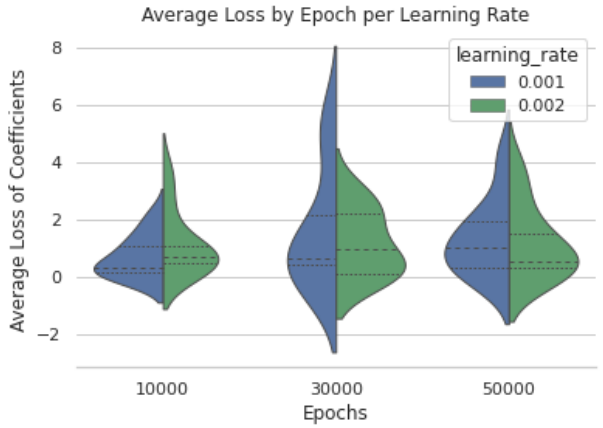

Figure 13. Average Loss by Epoch per Learning Rate

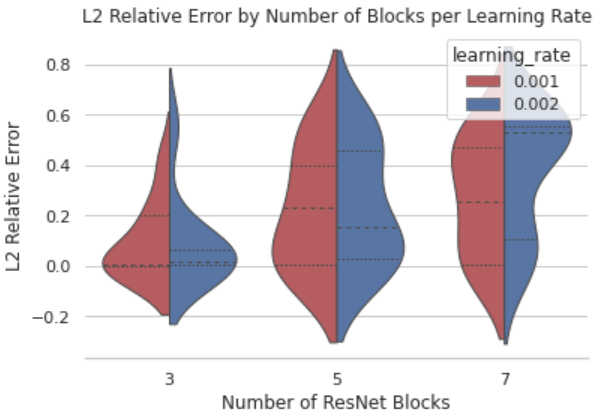

Figure 14. L2 Relative Error by Number of Blocks per Learning Rate

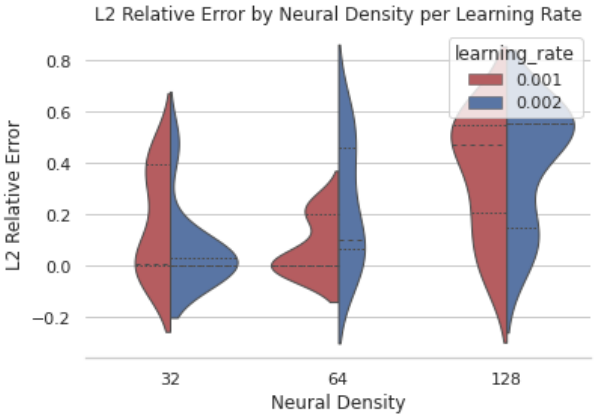

Figure 15. L2 Relative Error by Neural Density per Learning Rate

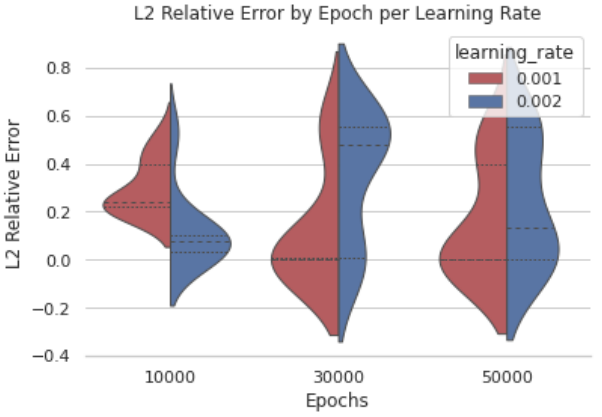

Figure 16. L2 Relative Error by Epoch per Learning Rate