Using Neural Networks to Solve Forward and Inverse Problems of Second-Order Ordinary Differential Equations

Jacob T. Pennington[1,], and Junshan Lin[2]*

[1] Undergraduate Student, Department of Mathematics and Statistics, Auburn University
[2] C Harry Knowles Endowed Professor, Department of Mathematics and Statistics, Auburn University

Differential equation models are ubiquitous in physics and engineering, and solving them efficiently and accurately is an important task in modern applied mathematics. With the growing success of AI and machine learning in recent years, building deep-learning solvers that could give solutions to these problems in real-time is plausible and exciting. The goal of this project was to develop efficient machine learning algorithms that could accurately predict the solution to second-order ordinary differential equations given the coefficients, the forward problem, and predict the coefficient given the solutions, the inverse problem. The differential equations that this project aimed to solve have the following form:

$$\frac{d}{dx}\left(a(x) \cdot \frac{du}{dx}\right) = f(x), \quad x \in (0,1) \tag{1}$$

To clarify, for some predetermined $f(x)$, the forward problem is to solve for $u(x)$ given coefficient $a(x)$, and the inverse problem is to solve for $a(x)$ given $u(x)$. For simplicity, we assumed the boundary condition $u(0)=0=u(1)$.

In order to develop machine learning algorithms to solve this problem, we first needed to generate data to train them on. To do this, we developed a finite difference solver in MATLAB to compute the solutions of the differential equations numerically. This solver was built using the centered difference approximation, allowing for efficient solving of a tridiagonal linear system [2].

Next, we generated many random coefficients $a(x)$ to run through the difference solver to find the corresponding $u(x)$ for each one. We found that the coefficient needed to be continuous, smooth, and positive, so we used the Karhunen-Loeve (K-L) Expansion [3] to generate many different pseudo-random coefficients.

The K-L Expansion we used is given by:

$$a(x) = \mu + \sum_{k=1}^{100} \lambda_k \cdot \xi_k \cdot \cos(k\pi x), \tag{2}$$

where, $\lambda_k = 1/k^2\pi^2$ and ξ_k is a random variable. By testing with trial and error, we found that using a mean $\mu=4$ and pulling each ξ_k from the normal distribution, N(0,50), gave the best results for our purposes. We also assume the following source function:

$$f(x) = 200 \cdot (x^2 - x) \cdot \sin(10x). \tag{3}$$

With this all coded in MATLAB and decided, we were able to efficiently generate data to train our machine learning algorithm consisting of 5,000 pairs of coefficients $a(x)$ and their corresponding solutions $u(x)$, each with 100 steps from 0 to 1, or 99 data points in between.

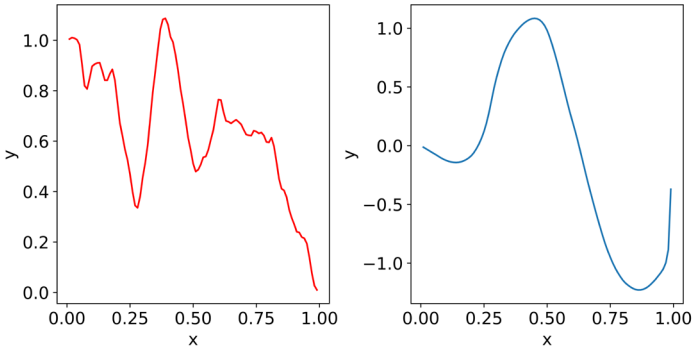

Fig. 1 An example pair of coefficient $a(x)$ (left) and solution $u(x)$ (right)

Once the data was generated in pairs as shown in Fig-

*Corresponding author:jtp0064@auburn.edu

ure 1, work began on constructing neural networks in Python using the PyTorch library to find the map from *a(x)* to *u(x)*.

The first task was to develop autoencoders to reduce the dimensions of both *a(x)* and *u(x)*, so we could find the map between them in the new low dimension. Autoencoders are special kinds of neural networks designed to reduce the dimensions of some set of data. They consist of two parts: the encoder to get to the low dimension, and the decoder to get back to the high dimension. These are adjusted simultaneously by running an input through both and using back-propagation with a cost function defined as the difference between the input and the output of the autoencoder [1].

We found that the most effective autoencoders for both *a* and *u* did not use a traditional activation function but one simple linear transformation to the low dimension. The a autoencoder could reduce the coefficient from 99 to 15 data points with 3 digits of accuracy, and the *u* autoencoder could reduce from 99 to 25 with the same accuracy.

Once these autoencoders were developed, we could work on the network to map the coefficient to solution (the forward map). Although we pursued training all autoencoders and this map together, we found that we had more accurate results when training both autoencoders and the map in the low dimension separately. We also found through testing on another generated dataset that we needed more data to train on, so we generated 100,000 coefficient-solution pairs.

Ultimately, by training on this new data set, we were able to train a network (as shown in Figure 2) so that it had an average mean squared error of 0.01 between the low-dimension solution and the network's predicted solution, and an average error of 0.02 between the true high-dimension solution and predicted solution of the test data. The network consisted of 5 layers of neurons, first with 30, then 50, 100, 50, and 30 neurons in the layers and the rectified linear unit activation function between them.

After work on the forward problem, we did similar work on the inverse problem using the same generated data, just looking for the map from *u(x)* to *a(x)*. This problem proved more challenging, as the coefficient had more features that the model needed to learn from fewer features in the solution *u*.

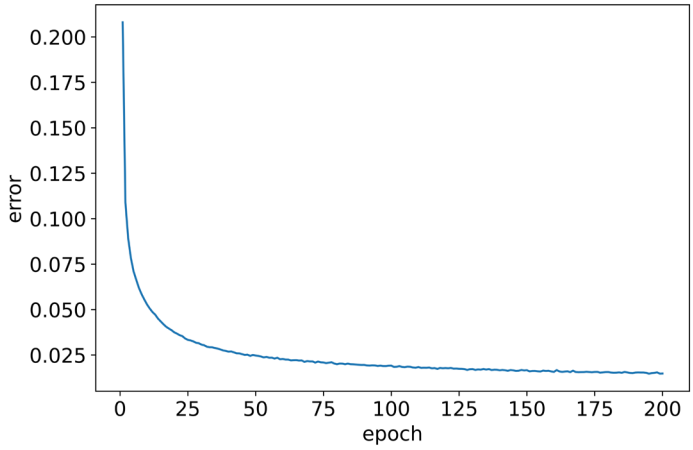

Fig. 2 A plot showing the error for each epoch of the forward network during training.

Because of this, we found that we needed a much larger network, consisting of 7 layers increasing to 1000 neurons, so that the model had the necessary degrees of freedom to accommodate such a map. We also found that, although it was about twice as expensive in terms of training time, training directly from *u(x)* to *a(x)* in the high dimension without using autoencoders proved more accurate than reducing each again. Ultimately, after completing the training (Figure 3), we were able to produce an average mean squared error of 0.009 between the network's predicted coefficient and the true coefficient in the inverse problem as well. The first layer was 99 neurons, then 200, 500, 1000, 500, 200, and 99 neurons, and we once again used the rectified linear unit activation function.

In conclusion, we were able to develop algorithms that could predict both forward and inverse problems with high accuracy. This work suggests that newly built neural network architecture can efficiently obtain solutions to a broad variety of differential equations. Further research must be conducted to test whether such neural networks can be used to solve the more complex partial differential equations; however, this project shed some light on attacking those problems.

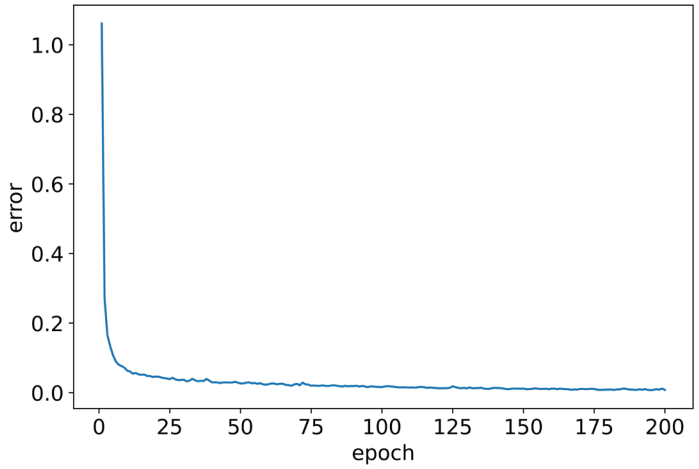

Fig. 3 A plot showing the error for each epoch of the inverse network during training.

Statement of Research Advisor

Jacob Pennington has developed a reduced-order based neural network for solving the forward and inverse problems in differential equations. He independently generated the training data using the finite difference method and trained the network using the stochastic gradient method in Pytorch. Through numerous numerical tests, it is shown that the new neural network is able to represent the forward and inverse maps for differential equations. The work sheds light for future investigation for the modeling of more complicated mathematic models using the reduced-order based networks.

- Dr. Junshan Lin, Department of Mathematics and Statistics, College of Sciences and Mathematics

References

[1] Bhattacharya, K., Hosseini, B., Kovachki, N.B., and Stuart, A.M., "Model Reduction and Neural Networks For Parametric PDEs", pp. 3-6, Journal of Computational Mathematics, (2021)

[2] LeVeque, R.J, Finite Difference Methods for Ordinary and Partial Differential Equations, Philadelphia: Society for Industrial and Applied Mathematics, pp.7-16, (2007)

[3] Lord, G.J., Powell, C.E., and Shardlow, T., An Introduction to Computational Stochastic PDEs, New York: Cambridge University Press, pp. 201-203, (2014)

Authors Biography

Jacob T. Pennington is a junior-year student pursuing a B.S. in Applied Mathematics at Auburn University. He plans to graduate in Spring 2024 and continue to work and learn in data science and machine learning while pursuing his M.S. in Data Science and Engineering the following year.

Junshan Lin is the C Harry Knowles Endowed Professor in Department of Mathematics and Statistics at Auburn University. He joined Auburn University in 2013 as an assistant professor, and was promoted to associate and full professor in 2018 and 2023. His research focuses on applied and numerical analysis, and scientific computation. He develops mathematical tools to solve scientific problems for waves in novel devices and materials, inverse wave scattering and imaging, computational inverse and optimal design problems. He has published over 40 papers in applied and computational mathematics journals and has delivered over 100 talks in a variety of international conferences and universities worldwide.