Solving PDEs with Neural Networks

Colin Scherer

April 2025

1 Introduction

For my final project, I have chosen to use Neural Networks to solve PDEs. Specifically, given an initial condition (a starting configuration), determine the ending configuration at a later time.

One PDE I will be trying to solve includes the one-dimensional Burger's equation, which takes the form given by Equation 1. The initial conditions will be Gaussian Random Field initial conditions, with $u_0 = u(x, 0)$ for $x \in [0, 2\pi]$, and a viscosity of $\nu = 0.1$ (constant). All initial condition samples and expected solutions were taken from:
https://www.kaggle.com/datasets/scaomath/pde-dataset.

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2} \tag{1}$$

Additionally, I attempt to tackle the two-dimensional Navier-Stokes equation for an incompressible fluid, with Reynold's number 500. The data used for training and testing was taken from https://zenodo.org/records/12825163.

My main motivation for this project comes from the numerous ways machine learning and computational methods were applied to physical problems this semester, particularly on PDEs. I wanted to explore this portion of the class due to my particular interest in both machine learning and physics, and wanted to see how they could be combined more.

Through this write-up, I hope to prove the feasibility of neural networks in solving PDEs.

2 Model and Experiments

My model is a simple feed-forward neural network, with a hidden layer dimension $d_h = 64$ and number of layers $L = 4$. Each hidden layer is followed by the ReLU activation function, which is followed by Batch Normalization to avoid overfitting. In the case of two-dimensional equations (such as the Navier-Stokes equation), inputs are flattened beforehand. The model was implemented in PyTorch and trained for 150-500 epochs, with a learning rate of 0.001. The AdamW optimizer was used, along with mean squared error as my loss function.

In the case of the 1-D Burger's equation, I compare selected solutions with solutions obtained computationally (using the Lax-Friedrichs method). This method of solving was based on [1].

For the 2-D Navier-Stokes equation, I tried incorporating convolutional and pooling layers, as discussed in class, but found them to be too computationally expensive with a relatively low error decrease, so I decided to forego them for now. I also would have liked to incorporate traditional solvers here as well, but was not sure how to go about doing that.

3 Results

Shown below are sample results for both classes of PDEs.

3.1 One-Dimensional Burgers Solutions

Shown in figure 1 are sample solutions taken from the test set. Generally, the neural network predictions seem a lot closer to the expected solution, and handle edge cases more robustly. This is reflected by the mean squared error – the best MSE obtained by the neural network was 0.008.

3.2 Two-Dimensional Navier-Stokes Solutions

Shown in figure 2 are sample solutions found by my network, where the leftmost iamges are the initial states, the center images are the expected final states, and the rightmost images are the predicted final states by my network.

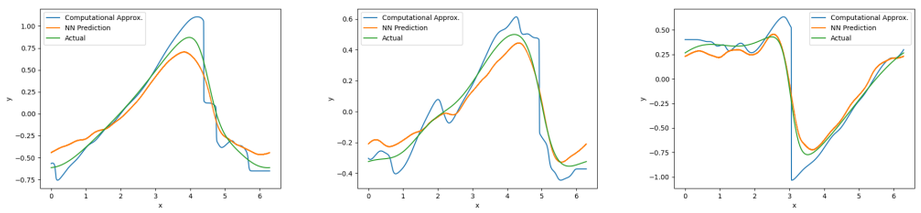

Figure 1: Sample predictions of the 1-D Burgers Equation, along with computational (traditional) approximation. Green lines are the actual (expected) solution, orange lines are the neural network prediction, and blue lines are the computational approximation.

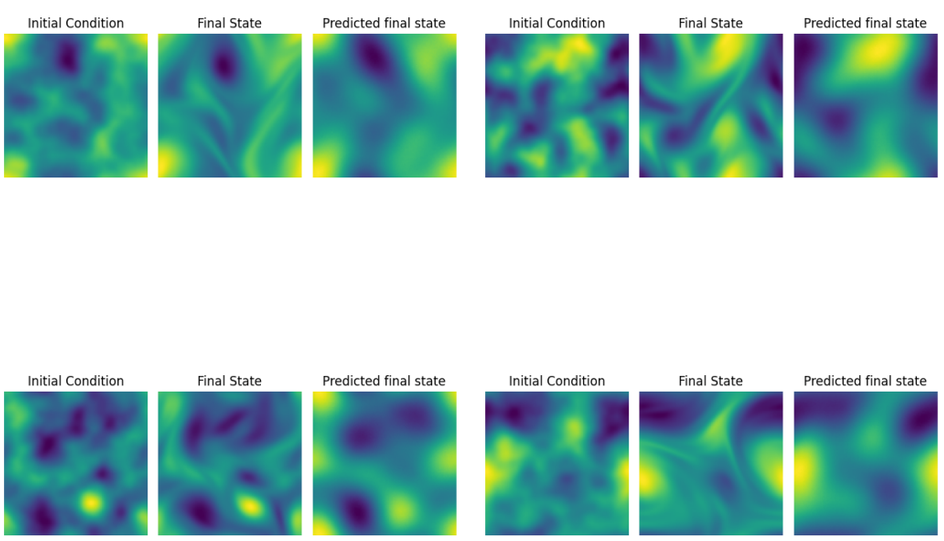

Figure 2: Sample predictions of my neural network from the test set.

4 Discussion and Conclusions

Although the computational estimates shown in section 3.1 seem inaccurate, I believe there may have been issues with my implementation, which led to more exaggerated prediction error. Indeed, the original expected solutions were obtained using traditional solvers.

Even with this accounted for, it can be argued that neural networks have more potential in problems like these. Whereas traditional solvers require huge amounts of time to solve only one equation (one set of ICs) in a given class of PDEs, neural networks can be trained once, and then generalize to any equation in the given class of PDEs.

This time save combined with relatively low error certainly makes the case for neural network integration into PDE solvers.

Despite the potential when evaluated on the one-dimensional Burgers Equation, my network struggles with the Navier-Stokes equation, particularly with resolution. Although a lot of the trends with flows seem accurate, the final prediction seems very blurry and hence unsure of itself. I tried incorporating convolutional layers in order to gather more meaningful structure relationships, but it proved very expensive to train, without decreasing error significantly.

If these neural networks were optimized further, they could be used to model various fluids such as gases, liquids, and even traffic flow. With extremely accurate results, weather prediction and mass transport can be optimized to achieve superior results.

During this project, I have used various techniques covered in class, such as neural networks (feed-forward and convolutional) and PDEs. I also use many data visualization tools we were taught throughout the semester.

The datasets used in this project were obtained from [2], which also goes more in-depth into how neural networks can be optimized using physics-based information. I suggest

referring to it for more ideas on how neural networks can be optimized for PDE solving.

All code can be found on my github.

References

[1] John Butler. https://john-s-butler-dit.github.io/NumericalAnalysisBook/Chapter%2010%20-%20Hyperbolic%20Equations/1004_Burger%20Equation.html, 2021.

[2] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations, 2021.