University of Oslo

FYS-STK3155

Solving the Wave Equation with Neural Networks

Brage Wiseth bragewi@ifi.uio.no
Felix Cameren Heyerdahl felixch@ifi.uio.no
Eirik Bjørnson Jahr eiribja@ifi.uio.no

December 18, 2023

https://github.com/bragewiseth/MachineLearningProjects

Contents

Introduction 3
1 Wave Equation 4
2 Data 4
3 Methods 5
3.1 Finite Difference Scheme 5
3.2 Physics Informed Neural Networks (PINNs) 6
3.3 Hybrid Solver 7
3.4 Recurrent Neural Networks (RNNs) 7
4 Results and Analysis 8
4.1 Hyperparameters and Architecture 9
4.2 Comparisons 11
5 Conclusion 13
Bibliography 14

Abstract

This study explores the application of neural networks, specifically Physics Informed Neural Networks (PINNs) and Recurrent Neural Networks (RNNs), in solving the wave equation, contrasting their effectiveness with classical methods like finite difference and analytical solutions. It demonstrates the feasibility of employing neural networks for solving partial differential equations (PDEs), offering a proof of concept for future research in this domain. We found that neural networks are indeed capable of approximating the wave equation, although the accuracy of the models is not as high as we would have hoped. For our simple case where both analytical and finite difference solutions are available, the neural network models are not able to achieve the same accuracy. The PINN manages to capture the physical laws embedded in its loss function, but might fail to generalize to other initial conditions. The RNN is able to capture the temporal dynamics of the wave equation and this approach it is more generalizable. Although the neural nets indeed learn the dynamics of waves, they also suffer from artifacts, such as violating energy conservation. A hybrid approach, combining the finite difference scheme with the PINN, is able to mitigate these artifacts and is the most promising approach for cases with fixed initial conditions. Despite this, our study demonstrates the potential of neural networks in solving more complex PDEs, paving the way for future research in this field.

Keywords: PINNs, RNNs, PDEs, Wave Equation, Neural Networks

Introduction

Partial differential equations (PDEs) are ubiquitous in the field of physics and engineering, describing a wide range of phenomena, from fluid dynamics to quantum mechanics. PDEs are in a sense the language of physics, and as such, solving them yields valuable insights into the physical world. While analytical solutions are often preferred, they are not always feasible, they are often limited to simple cases with idealized boundary and initial conditions. Numerical methods, although versatile, can be computationally demanding and less accurate for long simulations. In recent years, the advent of deep learning has opened new avenues for addressing complex computational problems. This paper aims to showcase neural networks as a promising tool for solving PDEs. As neural networks are universal function approximators, they should be capable of approximating the solutions to PDEs. A naive approach to employing neural networks for solving PDEs can be to simply train a dense neural network with simulated or recorded data in hopes that the neural network will learn the physical laws embedded in the data. This might require a lot of data, and without any constraints on what functions the neural network can learn, the search space is huge. By giving the neural network some help and insight about the specific problem, we hope to drastically limit the search space of the neural network, and the hunger for data as well as increase the accuracy of the model. We explore two distinct methodologies: Physics Informed Neural Networks (PINNs) and Recurrent Neural Networks (RNNs). PINNs embed physical laws in their structure, encouraging the learning of solutions that respect these laws. While RNNs are adept at processing sequences, allowing them to capture temporal dynamics effectively. Furthermore, we investigate a hybrid approach combining a traditional numerical methods like finite differnce schemes for labeled data generation combined with physical laws as constraints in the loss function. This investigation is primarily a proof of concept, highlighting neural networks' potential in solving PDEs, rather than an exhaus-

tive exploration of their full capabilities in this field, which we reserve for future research. We will delve into:

- **Wave Equation**: Discussing the wave equation's fundamentals.
- **Data**: Describing the data used for training the neural networks.
- **Methods**: Outlining our approaches to solve the wave equation using neural networks.
- **Results and Analysis**: Presenting and analyzing the outcomes of our experiments.

1. Wave Equation

The wave equation, a fundamental partial differential equation, models wave propagation through various media. It is mathematically represented as:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}, \tag{1}$$

where $u(x,t)$ denotes the displacement of the wave at position x and time t, and c signifies the wave speed. Typically, x is considered within a 3-dimensional or 2-dimensional space ($\mathbb{R}^3$ or $\mathbb{R}^2$) and t within the temporal dimension ($\mathbb{R}$). This paper, however, will focus on the one-dimensional case ($x \in \mathbb{R}$). The equation illustrates that the wave's acceleration at a point in time is directly proportional to its spatial curvature at that point. Solving the wave equation requires understanding the complex interaction between its spatial and temporal aspects. While analytical solutions exist for certain initial conditions, they are often impractical for more complex scenarios, necessitating the use of numerical methods or neural networks.

2. Data

This study utilizes specific initial and boundary conditions to model the wave equation. The initial conditions are set as follows:

$$u(x,0) = e^{-0.25x^2}, \quad \frac{\partial u}{\partial t}(x,0) = 0, \tag{2}$$

representing a Gaussian pulse with zero initial momentum at $t = 0$. The boundary conditions are:

$$u(-5,t) = u(5,t) = 0. \tag{3}$$

The analytical solution, derived from d'Alembert's formula, serves as an accuracy benchmark for our methods:

$$u(x,t) = \frac{1}{2}(e^{-0.25(x+ct)^2} + e^{-0.25(x-ct)^2}). \tag{4}$$

As we utilize different methods with slightly different goals we need to align the data accordingly.

Unsupervised Data

For the pure PINN with only the wave equation as a constraint, we do not require any labeled data. The input to the model is simply the coordinates (x, t), and the output is the displacement $u(x, t)$. We use a uniform distribution for x and t within the bounds.

Supervised Data

For the hybrid and RNN models, we need labeled data. We input coordinates (x, t), and use the solution from the finite difference scheme at that point as the label.

Sequential Supervised Data (RNN)

For the RNN model we input a sequence of spatial points at each time slice $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n$ where each $\mathbf{u}_i$ is the solution $\mathbf{u}(\mathbf{x}_i, t_i)$ over the entrie bounded spatial domain $\mathbf{x}_i$ at time t_i. These solutions can be sampled from the analytical solution, or from the finite difference solver. For a given sequence $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n$ of inputs the label is the next point in the sequence $\mathbf{u}_{n+1}$. It is important that the data is not shuffled, as the network needs to process the data in order to learn the telmporal dynamics of the wave equation.

3. Methods

3.1 Finite Difference Scheme

A finite difference scheme is a numerical method for approximating the solutions to differential equations. The idea is to approximate the derivatives in the differential equation with finite differences. The simplest way to do this is to use the forward difference approximation:

$$\frac{\partial u}{\partial x} \approx \frac{u(x+h) - u(x)}{h} \tag{5}$$

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} \tag{6}$$

where h is a small number. The smaller h is, the more accurate the approximation will be. However, if h is too small, the approximation will be unstable. The optimal value of h depends on the problem. In our experiments we have used $h = \Delta t = 0.02$ and $h = \Delta x = 0.1$. We can use the forward difference approximation to approximate the derivatives in the wave equation:

$$\frac{\partial^2 u}{\partial t^2} \approx \frac{u(x, t+h) - 2u(x, t) + u(x, t-h)}{h^2} \tag{7}$$

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u(x+h, t) - 2u(x, t) + u(x-h, t)}{h^2} \tag{8}$$

We can then use these approximations to approximate the wave equation:

$$\frac{u(x, t+h) - 2u(x, t) + u(x, t-h)}{h^2} = c^2 \frac{u(x+h, t) - 2u(x, t) + u(x-h, t)}{h^2} \tag{9}$$

We can then solve for $u(x, t+h)$:

$$u(x, t+h) = 2u(x, t) - u(x, t-h) + c^2(u(x+h, t) - 2u(x, t) + u(x-h, t)) \tag{10}$$

This is expression translates directly into code. We can use this to get a numerical solution to the wave equation. For each time step, the approximation gets more and more inaccurate. This is because the approximation is based on the previous time step, which is based on the time step before that, and so on. This way the error accumulates over time. For simulations that run for a long time, the error can become very large. One way to address this is to use a smaller time step h.

3.2 Physics Informed Neural Networks (PINNs)

Physics Informed Neural Networks (PINNs) leverage the Universal Approximation Theorem, which posits that a neural network with even a single hidden layer can approximate any function to a desired degree of precision. This foundational principle enables neural networks to approximate complex functions, including solutions to differential equations. PINNs, in particular, incorporate physical laws into their structure, allowing them to learn the underlying dynamics of a system. For our case we wish to embed the wave equation into the loss function of the neural network.

Generally, partial differential equations (PDEs) can be expressed in the form:

$$f\left(x_1, \ldots, x_N, \frac{\partial g(x_1,\ldots,x_N)}{\partial x_1}, \ldots, \frac{\partial g(x_1,\ldots,x_N)}{\partial x_N}, \frac{\partial g(x_1,\ldots,x_N)}{\partial x_1 \partial x_2}, \ldots, \frac{\partial^n g(x_1,\ldots,x_N)}{\partial x_N^n}\right) = 0 \tag{11}$$

where $g(x_1, \ldots, x_N)$ is the solution to the PDE. By expressing the PDE in this form, we can see that the PDE is satisfied when $f = 0$. We can then use this to define a loss function for the neural network:

$$\min_P \mathcal{L} = \min_P \sum_{i=1}^{N} f(x_i, P) \tag{12}$$

where P are the parameters of the neural network, and x_i are the data points. The specific form of f for our case is:

$$f(x, t, P) = \frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} \tag{13}$$

where $u(x, t, P)$ is the solution to the wave equation. In addition to minimizing eq. 13, we also want to ensure that the neural network satisfies the initial and boundary conditions. We can propose a trial solution that ensures this:

$$u(x, t) = h_1(x, t) + h_2(x, t, N(x, t, P)), \tag{14}$$

Here, $h_1(x, t)$ represents the initial and boundary conditions, while $h_2(x, t, N(x, t, P))$ ensures that the the output of the neural network respects the initial and boundary conditions. The network minimizes the wave equation's residual, aligning the trial solution with the wave's true dynamics. In our specific implementation, we simply add the pure wave equation loss with the boundary and initial conditions loss. where the wave equation loss is the mean of eq. 13 over the data points, the initial conditions and boundary loss is defined as

$$u(x, 0) - \Omega + \frac{\partial u}{\partial t}(x, 0) - \Gamma \tag{15}$$

where Ω and Γ are the initial conditions giving us the final loss function

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} f(x_i, t_i, P) + u(x_i, 0) - \Omega + \frac{\partial u}{\partial t}(x_i, 0) - \Gamma \tag{16}$$

Extracting the initial condotions generated by the neural network is trivial, as we can simply evaluate the network at $t = 0$, and $t = dt$ to get the initial displacement and momentum respectively. The output of the neural network $u(x, t)$ is a function of x and t, and we can exctract the second order derivatives needed to evaluate the wave equation, this can be computed by differentiating the output of the neural network twice with respect to x and t. This can be done analytically, or by using automatic differentiation. The architecture of the neural network is a fully connected feed forward neural network, although other architectures may be used.
[2] [5]

3.3 Hybrid Solver

In some cases, a finite difference scheme may be quick to compute, and provide a decent approximation to the solution. We propose a hybrid solver, where we use a finite difference scheme to generate data with labels, and use this data to define an error between the output of the neural network and the finite difference scheme. We can choose to only use this error as the loss function, however this will only give us a solution that is as good as the finite difference scheme. To improve on this, we can also add the wave equation as a constraint to the loss function. It is also possible to weight the components of the loss function differently, to give more or less importance to the different components. We present an example of this in our experiments.

$$\mathcal{L} = \alpha\mathcal{L}_{\text{FD}} + \mathcal{L}_{\text{PINN}} \tag{17}$$

where $\mathcal{L}_{\text{FD}}$ is MSE of the output of the neural network with the finite difference scheme as the target, (or your favorite solver), and $\mathcal{L}_{\text{PINN}}$ is the physics informed neural network loss. α is a hyperparameter that determines the relative importance of the two components of the loss function.
[5]

3.4 Recurrent Neural Networks (RNNs)

The problem with our current approch with PINNs is that we have to hard code the wave equation and the initial conditions into the loss function. This means that we have to train a new model for each set of initial conditions. In addressing the challenges posed by this lack of generalization, we can use a Recurrent Neural Network (RNN). RNNs are a class of neural networks that are well suited for processing sequential data. RNNs are able to process sequences of data, and use the information from previous data points to process the current data point. This way the RNN can learn the temporal dynamics of the wave equation, and thus generalize to other initial conditions. We can train the RNN on a small time frame, and then use the RNN to predict the evolution of the wave for the rest of the time frame. This way we can train a single model that can be used to predict the evolution of the wave for any sequence wheter it be a small time frame simulation or

analytical solution. An RNN is a neural network is on the form

$$\mathbf{h}_t = \sigma(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t + \mathbf{b}_h) \tag{18}$$

Where $\mathbf{h}_t$ is the hidden state at time t, $\mathbf{x}_t$ is the input at time t, $\mathbf{W}_{hh}$ is the weight matrix for the hidden state, $\mathbf{W}_{xh}$ is the weight matrix for the input, $\mathbf{b}_h$ is the bias, and σ is the activation function. The output of the RNN is then

$$\mathbf{y}_t = \mathbf{W}_{hy}\mathbf{h}_t + \mathbf{b}_y \tag{19}$$

where $\mathbf{W}_{hy}$ is the weight matrix for the output, and $\mathbf{b}_y$ is the bias. The RNN is trained by feeding it a sequence of inputs $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$, and the label is the next point in the sequence $\mathbf{x}_{n+1}$. The loss function is then

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{y}_i - \mathbf{x}_{i+1})^2 \tag{20}$$

where N is the number of data points in the sequence. to capture the temporal dynamics of the wave equation. This way we can train a single model that can be used to predict the evolution of the wave for any set of initial conditions. RNNs are distinguished by their ability to process sequences and their inherent memory, making them well-suited for time-dependent problems. Our RNN model utilizes a series of LSTM (Long Short-Term Memory)[3] layers, chosen for their effectiveness in capturing long-range dependencies and mitigating the vanishing gradient problem. For the RNN model we input a sequence of spatial points at each time slice $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n$ where each $\mathbf{u}_i$ is the solution $\mathbf{u}(\mathbf{x}_i, t_i)$ over the entrie bounded spatial domain $\mathbf{x}_i$ at time t_i. These solutions can be sampled from the analytical solution, or from the finite difference solver. For a given sequence $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n$ of inputs the label is the next point in the sequence $\mathbf{u}_{n+1}$. It is important that the data is not shuffled, as the network needs to process the data in order to learn the telmporal dynamics of the wave equation. [4]

4. Results and Analysis

Table 1: Mean squared error between analytical solution and different solvers. Lower is better.

Finite difference	4.9722476e-05
PINN	2.7890243e-05
Hybrid	0.00002

Our experiments yielded insightful results regarding the performance of different solvers. We found that the PINN and hybrid outperformed other methods in terms of accuracy. The RNN did not show promising results, likely due to the simplicity of our example. Although the PINN and hybrid solver peroformed well in terms of accuracy, they both suffer from extensive training times.

Limitations and Potential of PINNs:

The implementation of PINNs in our study was tailored specifically for the wave equation, with both the equation and initial conditions hardcoded into the loss function. This approach limits the generalizability of our PINNs model to other problems. Nevertheless, by adopting a more generalized loss function, it might be feasible to extend the application of PINNs to a broader range of problems.

Overfitting Considerations in Different Solvers:

In the context of solving the wave equation with PINNs, traditional overfitting concerns are less relevant since the network seeks an exact solution rather than a generalized one. However, for the hybrid solver and RNN, overfitting remains a significant challenge. We addressed this in the hybrid solver by adjusting the weight of different loss function components. Additionally, classical regularization techniques were employed as preventive measures against overfitting.

Data Normalization and Scale Considerations:

In cases where input features exhibit vastly different scales, normalizing the data is generally advisable. Our experiments, however, did not involve such normalization as the features in our example had similar scales. This decision was based on the context of our specific use case, though it may vary in different scenarios.

Potential and Limitations of Neural Networks for PDEs:

We acknowledge that our examples might be too simplistic to fully demonstrate the capabilities of neural networks in solving Partial Differential Equations (PDEs). Our study was constrained to a one-dimensional problem with straightforward initial conditions and a brief simulation duration. Under these circumstances, traditional methods like finite difference schemes might be preferable due to their speed and accuracy. However, for more complex scenarios, the neural network-based methods we explored could offer advantages over conventional techniques. Furthermore, it's important to note that neural networks typically require substantial data to achieve optimal performance, a factor that should be considered in future applications.

4.1 Hyperparameters and Architecture

In our proof of concept, extensive hyperparameter tuning wasn't a primary focus. However, we conducted preliminary experiments to understand the impact of hyperparameters on model performance. Our efforts were primarily directed towards exploring how various components of the loss function influence the efficiency of the hybrid solver. Notably, we didn't investigate the hyperparameter effects on the RNN model's performance due to time and computational resource constraints. For our PINN models, we implemented two hidden layers, each comprising 50 nodes. The neural networks were developed using TensorFlow[1]. The Adam optimizer was employed with a learning rate of 0.001, which provided an effective balance between computational efficiency and learning accuracy. Additionally, we incorporated weight decay in our training process, applying a weight of 0.0001 to prevent overfitting

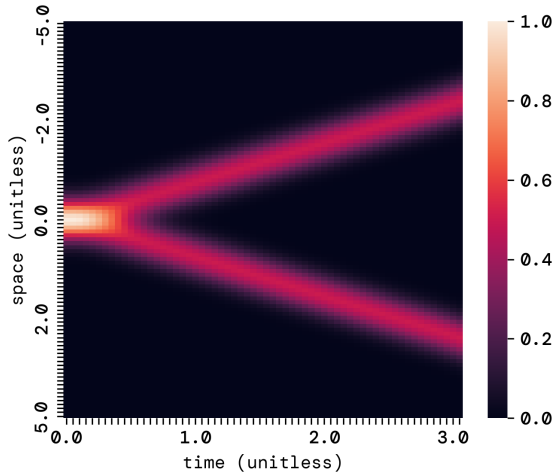

Figure 1: Analytical solution to the wave equation.

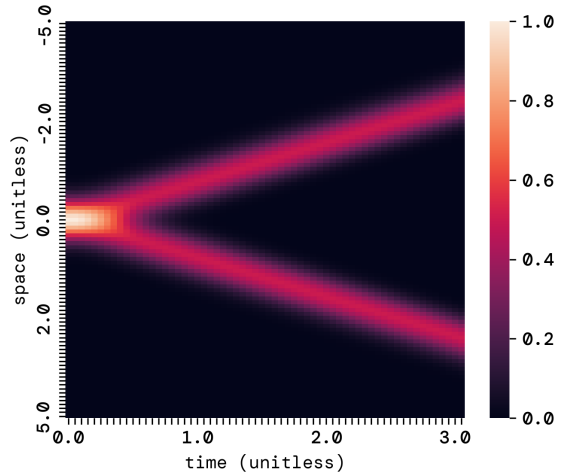

Figure 2: Finite difference solution to the wave equation.

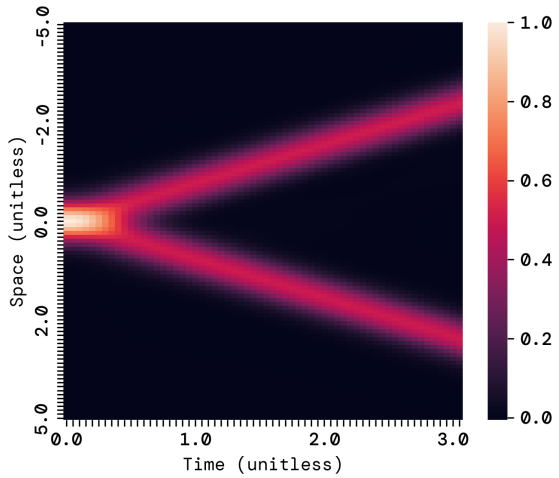

Figure 3: PINN solution to the wave equation.

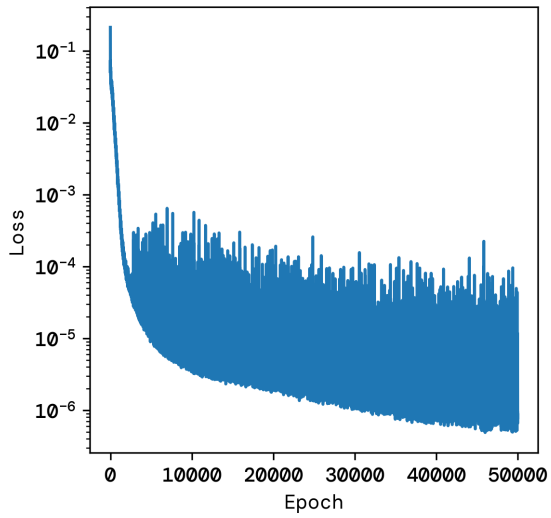

Figure 4: The loss curve for the PINN. The loss is very noisy, but it seems to not have converged after 50000 epochs.

and improve model generalization. The RNN model consisted of three LSTM layers, each comprising 50 nodes. The model was trained with the same hyperparameters as the PINN model, except for the sequence length which was set to 10.

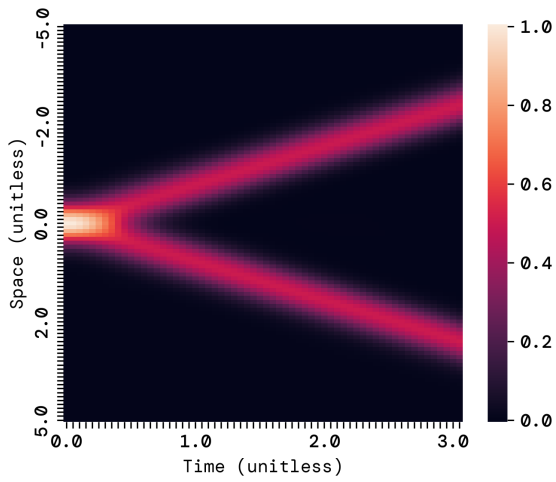

Figure 5: The hybrid solver is able to mitigate the artifacts of the pure PINN, and is the most accurate NN solver.

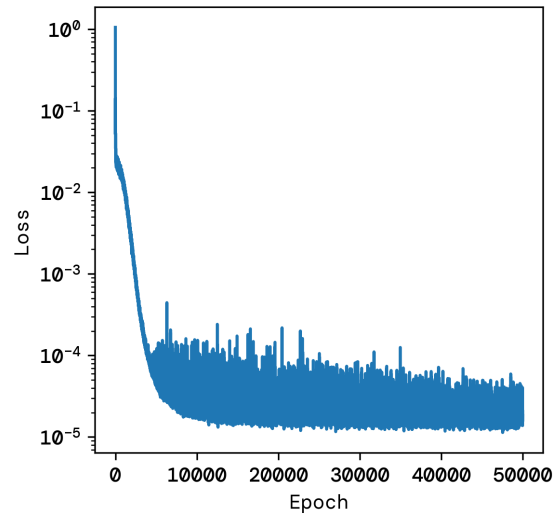

Figure 6: The loss curve for the hybrid solver. The loss seems to have converged after 10000 epochs.

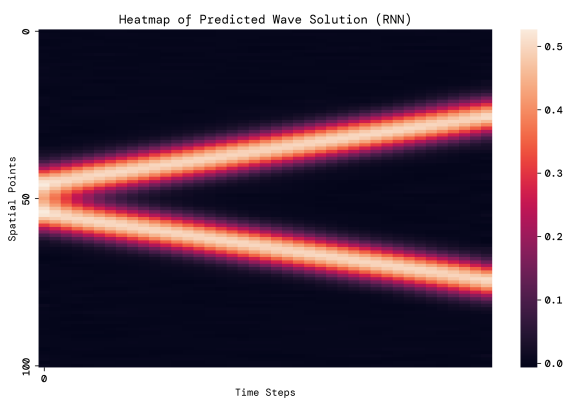

Figure 7: RNN solution to the wave equation on the same time frame as the training data. The first 10 points are not shown as they are used as input to the RNN.

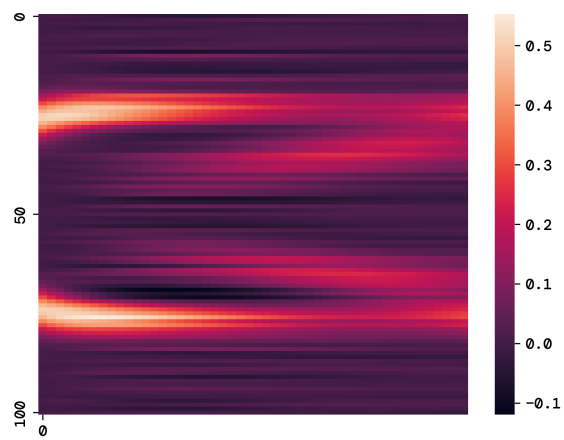

Figure 8: The RNN predicts the evolution of the wave after the initial time frame of the training data. We can see that the accuracy diminishes quickly.

4.2 Comparisons

With PINN and the hybrid we got decent results although it took a long time to train perhaps more sophisticated approaches could yield better results. The paper by Krishnapriyan et al. [5] shows more advanced approaches that could be used to improve the performance of the PINN and hybrid solver. The RNN model was not able to capture the temporal

dynamics of the wave equation, and the accuracy diminished quickly after the initial time frame of the training data. This is likely due to the simplicity of our example. more complex examples might yield better results. The paper by Hu et al. [4] shows promising results for the RNN model, although they use a more complex example.

5. Conclusion

We have explored the use of neural networks in solving the wave equation. Our findings indicate that neural networks are capable of approximating the wave equation, although the accuracy of the models is not as high as we would have hoped. For our simple case where both analytical and finite difference solutions are available, the neural network models are not able to achieve the same accuracy. However, as the partial differential equations become more complex, the analytical solutions may be impossible to find, and the finite difference solutions may become too computationally expensive or inaccurate. In these cases, neural networks may be a viable alternative. Future work could involve exploring more complex partial differential equations , such as the Navier-Stokes equations, with more complex bounds and initial conditions and comparing the accuracy of the neural network models to the analytical and finite difference solutions. Another interesting direction would be to explore the use of Nevertheless, our experiments shed light on how loss function components can be optimized to enhance the RNN model's effectiveness. This study demonstrates that a well-crafted loss function can significantly influence performance, especially when it incorporates techniques that leverage prior knowledge about the problem domain.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Morten Hjorth-Jensen. Fys-stk4155/3155 applied data analysis and machine learning. `https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/intro.html`, 2021. accessed 18.12.2023.

[3] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[4] Yihao Hu, Tong Zhao, Shixin Xu, Zhiliang Xu, and Lizhen Lin. Neural-pde: A rnn based neural network for solving time dependent pdes, 2022.

[5] Aditi S. Krishnapriyan, Amir Gholami, Shandian Zhe, Robert M. Kirby, and Michael W. Mahoney. Characterizing possible failure modes in physics-informed neural networks, 2021.

[6] OpenAI. Chat-gpt. `https://openai.com/chatgpt`, 2023. Some of the code and formulations are developed with the help from Chat-GPT.

[7] Brage Wiseth, Felix Cameren Heyerdahl, and Eirik Bjørnson Jahr. MachineLearning-Projects. `https://github.com/bragewiseth/MachineLearningProjects`, November 2023.