Solving the 1D Advection PDE with a Physics Informed Neural Network

Chayton Bouwmeester

March 2025

1 Abstract

In this project I have developed a physics informed neural network (PINN) which is able to solve the 1D advection equation on a domain with periodic boundary conditions. Testing during the development of the PINN found best results with a sine activation function and 32 nodes per hidden layer. After finding a well-performing structure, the PINN was evaluated against the upwind and Lax-Wendroff numerical methods as well as the analytical solution of the advection equation. It was found that the PINN, once trained, was able to accurately capture the behaviour of the system and provided better results when compared to the traditional numerical methods. Furthermore, PINNs were shown to overcome the issue instability for certain grid parameters, as well as the effect of numerical diffusion. This project overall illustrates a simple example of how PINNs can be implemented and their ability to solve PDEs.

2 Introduction

Solving partial differential equations numerically is a common task in many engineering and physics applications. However, in problems with high dimensionality, i.e. simulations with a high resolution or with many variables, numerical methods can take prohibitively long. Physics informed neural networks (PINNs) offer an approach to solving these equations in situations where traditional numerical methods are not computationally feasible.

Physics informed neural networks operate using the universal approximation theorem, which states that given a sufficient number of hidden nodes and a non-linear activation function, a neural network can approximate any continuous function to an arbitrary precision [3] [6]. What makes these neural networks 'physics informed' is the direct incorporation of physical laws into the loss functions used to train these neural networks. This guides the training processes and ensures certain physical constraints are adhered to by the solutions provided by the PINN.

Although training can be costly, the advantage of neural networks is that once trained, solutions can be produced for very little computational cost. Furthermore, unlike numerical methods, neural networks appear to be much less sensitive to the curse of dimensionality [4] [2]. PINNs can also be useful in situations where data is sparse, in which case one can incorporate known physics associated with a problem directly into the loss function to better allow the neural network to converge to a reasonable solution even when data is sparse or incomplete [1].

In this project I will develop a PINN for solving the 1D advection equation. I will explore the performance of the neural networks for different neural network hyperparameters. Once the network has been created, I will compare the performance of the PINN to traditional numerical methods as well as the analytical solution of the advection equation.

3 Methods

3.1 Advection Equation

The advection equation in one dimension is a hyperbolic partial differential equation which describes the motion of a scalar u as it is advected by a velocity field. It can be written as follows:

$$\frac{\partial u}{\partial t} = a\frac{\partial u}{\partial x} \tag{1}$$

Where a is the advection velocity. To solve this problem, we wish to find $u(x,t)$ which satisfies equation 1. We can impose the following initial condition (IC):

$$u(x, t = 0) = u_0(x) \tag{2}$$

Where L is the length of the domain in x. The analytical solution of the advection equation for an infinite domain and a constant velocity a is then found through the method of characteristics to be:

$$u = u_0(x - at) \tag{3}$$

Where u_0 is an arbitrary function defined on $\mathbb{R}$. We can see here that the function maintains its shape but is simply shifted in phase proportionally to the size of a. This analytical solution will be used to evaluate the performance of the PINNs and the numerical methods.

3.2 Problem Setup

The domain in which the 1D advection equation will be solved is set up by creating a discrete grid in x and t according to the parameters specified in Table 1. For the purposes of testing the numerical methods and the neural network, these parameters will be used for the grid except when otherwise stated.

In order to solve the advection equation, one must provide an initial condition to act as an initial distribution which will be advected across x over time. For this project, a Gaussian pulse will be used:

$$u_0 = \exp\left(-(\frac{(x - \frac{L}{4})}{0.1})^2\right) \tag{4}$$

The boundary condition (BC) will be set as periodic i.e.

$$u(x = 0, t) = u(L, t) \tag{5}$$

Table 1: Problem Parameters, their symbols, and corresponding values

Parameter	**Symbol**	**Value**
Advection Velocity	c	1.0
Spatial Grid Points	nt	300
Time Step	Δt	0.01
Spatial Domain Length	L	5.0
Simulation Time	T	5.0

3.3 Numerical Methods

A simple first order numerical method for solving the advection equation is the upwind scheme. This method uses the information from 'upwind' to solve the PDE for a positive a i.e. it is forward in time, backwards in space method. Formally, it can be written as:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + a\frac{u_i^n - u_{i-1}^n}{\Delta x} = 0 \quad \text{for} \quad a > 0$$

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + a\frac{u_{i+1}^n - u_i^n}{\Delta x} = 0 \quad \text{for} \quad a < 0$$

This method is stable as long as the Courant-Friedrichs-Lewy (CFL) condition is satisfied:

$$C = |\frac{a\Delta t}{\Delta x}| \leq 1 \quad \text{and} \quad 0 \leq a \tag{6}$$

Where C is the Courant number. Additionally the Lax-Wendroff scheme will also be used as a point of comparison. The Lax-Wendroff scheme is a second order accurate method of solving hyperbolic PDEs. For a linear hyperbolic PDE, such as the 1D advection equation, it can be formally written as:

$$u_i^{n+1} = u_i^n - \frac{a\Delta t}{2\Delta x}(u_{i+1}^n - u_{i-1}^n) + \frac{a^2\Delta t^2}{2\Delta x^2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n),$$

Similar to the upwind scheme, the Lax-Wendroff scheme can be shown to be stable for Courant numbers less than or equal to 1.

3.4 Physics Informed Neural Network

In this project I aim to build a neural network which approximates a function $u(x,t)$ which satisfies equation 1 as well as the specified initial and boundary conditions. The process for training the neural network is summarised as follows:

- Randomly sample 200 values within the range $[0, L]$ for x and $[0, T]$ for t.
- Compute the corresponding values of u from the neural network given each pair of x and t values.
- Using the neural network prediction, calculate the derivatives in x and t from equation 1 using automatic differentiation. Equation 1 can then be evaluated. As the result should be equal to zero, the square of the residual can be used as part of the loss of the neural network, enforcing the PDE.
- Evaluate the IC loss by comparing the neural network's predictions for u for each sampled value of x at $t = 0$ against the previously specified initial condition. The square difference is then the IC loss.
- Evaluate the BC loss by comparing the neural network's prediction for u at $x = 0$ against the prediction for $x = L$ for the specified time step. The square difference is then the BC loss.
- The three loss functions are then added together with specified weights. This overall loss is then used to optimise the neural network.
- Repeat for each training epoch.

When calculating the overall loss, the PDE loss is multiplied by 2 whereas the BC and IC losses were not weighted. This was purely obtained through trial and error and was found to give the best results.

Different activation functions and the number of nodes per layer will be explicitly explored in this project, but the other hyperparameters will remain fixed for the purposes of testing. The parameters which are not investigated explicitly were also simply arrived upon through trial and error. Table 2 summarises the values for these hyperparameters which are used for training the neural networks unless otherwise stated. The neural network code is written in Python using the PyTorch library [5].

4 Results and Discussion

4.1 Analytical Solution

Firstly, the analytical of the solution can be plotted as a point of comparison to the other methods (Figure 1). As expected, the solution simply shows the initial Gaussian pulse shifted in phase when stepping forward in time. The pulse maintains its shape as it travels in space and time.

Table 2: Neural Network/Training Parameters and Corresponding Values

Hyperparameter	**Value**
Number of Epochs	14000
Number of Samples	200
Number of Hidden Layers	4
Number of Nodes per Layer	32
Optimiser	Adam
Learning Rate	0.01
Activation Function	Sin

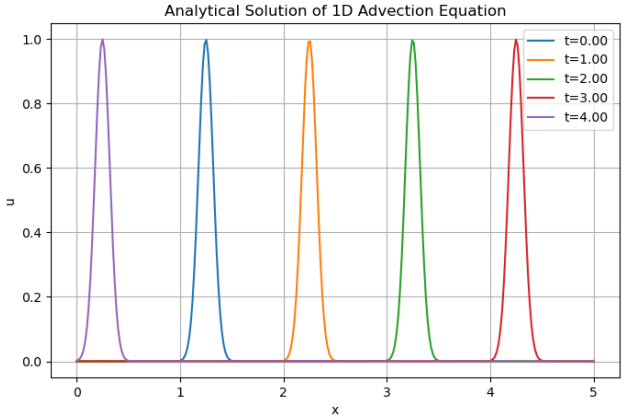
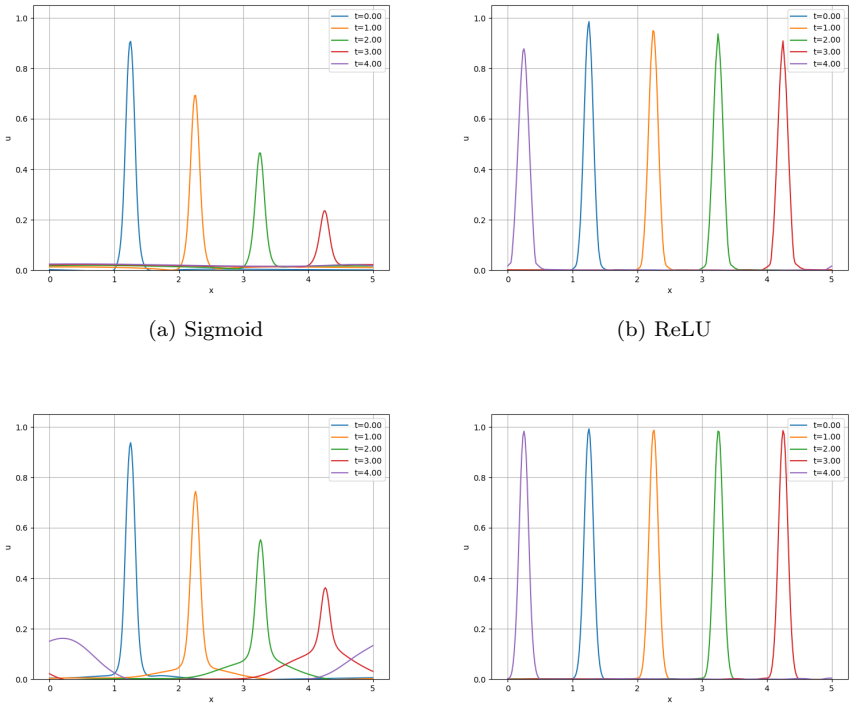


Figure 1: Analytical solution for the 1D advection equation with parameters in Table 1.

4.2 PINN - Activation Functions

In Figure 2, results from the neural network for the hyperparameters specified in Table 2 are shown for four different time steps and with varying activation functions. For each plot, the specified activation function is used in all layers of the network. By simple visual inspection it is clear that the sine activation function provides the best results when compared to the analytical solution. The hyperbolic tangent and sigmoid activation functions both appear to show significant diffusion over time and both appear to struggle with values of u close to zero. ReLU appears to perform much better for the values of u close to zero, but still suffers from a degree of diffusion. The sine activation function on the other hand seems be able to handle values of u close to zero will showing only a minor degree of diffusion. This could be due to fact that the advetion equation models wave-like propagation, and in this problem I have also imposed periodic boundary conditions. As the sine function inherently encodes periodic

behaviour, perhaps this allows it to more naturally represent this system. For the following sections, I therefore use the sine activation function for all the layers in the network.



(a) Sigmoid

(b) ReLU

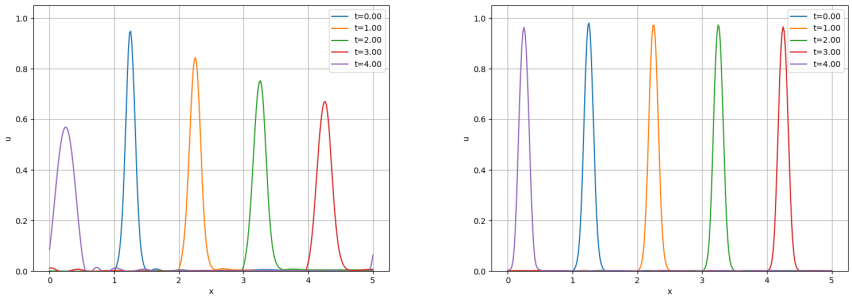
(c) Tanh

(d) Sin

Figure 2: Neural network calculations for the advection equations using four different activation functions: Sigmoid (a), ReLU (b), Tanh (c), and Sin (d).

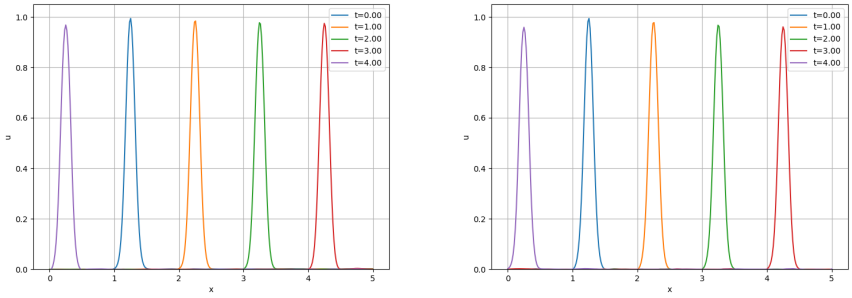
4.3 PINN - Nodes per Layer

Looking at Figure 3, it does indeed appear that a larger number of nodes per hidden layer allows the PINN to better capture the behaviour of the system up to a point. However, these improvements are quite incremental, with doubling of the number of nodes only providing marginally better results. This suggests that much of the complexity of this system can be sufficiently captured with a relatively small number of hidden nodes. When increasing up to 64 nodes per layer, the solutions do not show any significant improvement over 32 nodes per layer. Therefore, as 32 is shown to be capable of providing very good results while minimising training time, this will be used for the evaluation in section 4.5.



(a) 8 Nodes/Layer

(b) 16 Nodes/Layer



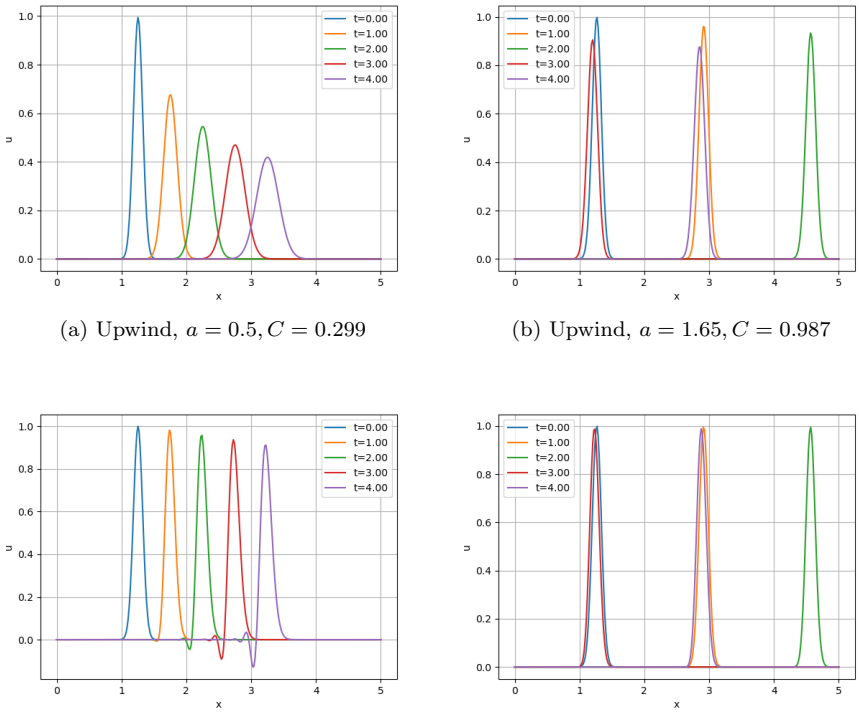
(c) 32 Nodes/Layer

(d) 64 Nodes/Layer

Figure 3: Neural network calculations for the advection equations using four different numbers of nodes per hidden layer: 8 (a), 16 (b), 32 (c), and 64 (d).

4.4 Numerical Schemes

Figure 4 shows the results of the two numerical schemes for different values of a, and therefore different Courant numbers (C). The stability of both the Upwind and Lax-Wendroff schemes for solving the 1D advection equation depends on the value of C. A value of C greater than 1 results in instability. When C is less than or equal to 1, the solutions are stable, however numerical diffusion becomes more significant the closer the C is to zero.



(a) Upwind, $a = 0.5, C = 0.299$

(b) Upwind, $a = 1.65, C = 0.987$

(c) Lax-Wendroff, $a = 0.5, C = 0.299$

(d) Lax-Wendroff, $a = 1.65, C = 0.987$

Figure 4: Results of the numerical schemes for different values for the advection strength a and corresponding C values.

The upwind scheme shows the largest degree of numerical diffusion, as expected. The Lax-Wendroff scheme on the other hand is noticeably more robust against numerical diffusion due to the addition of the second order correction term. The Lax-Wendroff scheme does however begin to show some small oscillations behind the pulse over time (Figure 4c). This is because unlike the Upwind scheme, Lax-Wendroff is non-monotonic, which results in these spurious oscillations.

4.5 Comparison

For the comparison the PINN is trained according to the hyperparameters summarised in Table 2. The training loss over the epochs is shown in Figure 5. The loss values decrease fairly consistently until about 12000 epochs, at which point the losses begin to approximately stabilise.

A comparison of the different methods for different time steps is shown in Figure 6. Here it can be seen that the numerical methods diffuse over time whereas the neural network is able to very closely follow the analytical solution over the whole simulation time period. The PINN however shows the best results of the tested methods. The L2 norms of the difference vector of the methods as compared to the analytical solution are shown in Table 3 for different time steps. The norm of the PINN remains grows very little over the course of the simulation when compared to the numerical methods, although the solutions still do deviate from the analytical. It is also clear that the neural network was able to accurately model the behaviour of the periodic boundary conditions while maintaining an amplitude close to the analytical solution (Figure 6d).

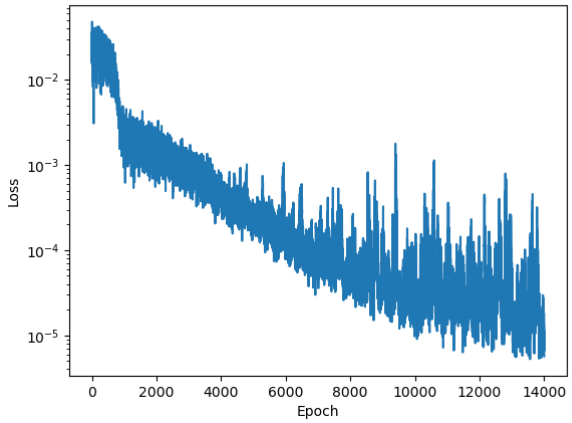
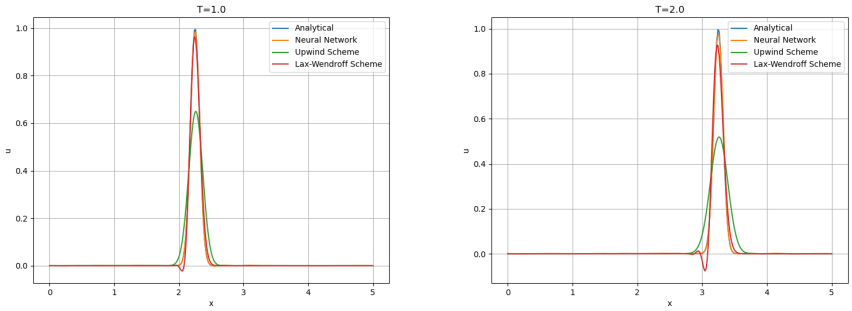


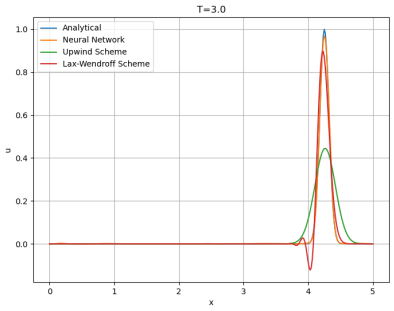
Figure 5: Training loss of PINN over training epochs.

Another interesting observation is that the PINN is able to reasonably model the advection equation for higher values of a, without showing the instability of the numerical methods (Figure 7a). As the PINN is not reliant on certain numerical constrains, it can learn the behaviour of the system for conditions that the numerical methods are unable to model. Another advantage of the PINN is that it does not suffer from the numerical diffusion at lower C values (Figure 7a) observed in both of the numerical methods (Figures 4a and c). This is because the PINN optimises globally over the entire problem domain, and can smooth out instabilities at all points. This means that the neural network is effectively decoupled from the grid dependant time stepping, and is unaffected

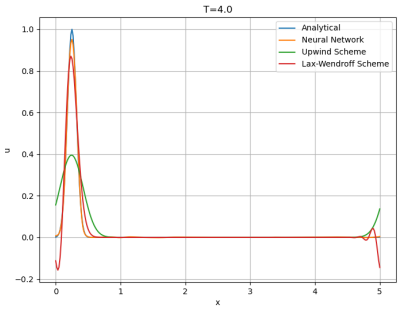


(a) t=1.0

(b) t=2.0



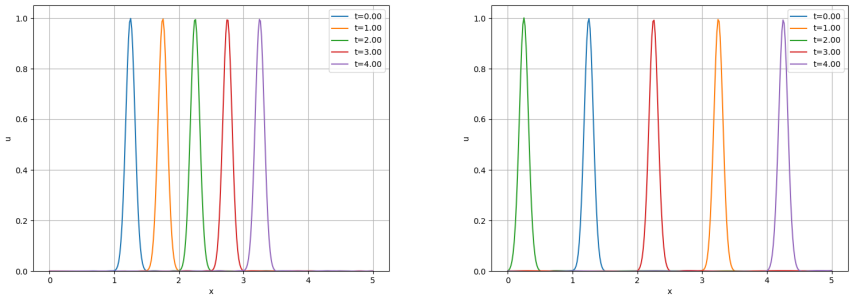
(c) t=3.0



(d) t=4.0

Figure 6: Comparison of different numerical methods and neural network solution for the default parameters ($C = 0.598$).

by these parameters. Numerical methods on the other hand must step through the problem sequentially, which leads these errors build over time as a result of the discretisation.



(a) $a = 0.5, C = 0.299$ (b) $a = 2.0, C = 1.196$

Figure 7: Neural network solutions with default parameters and two different values of a.

Table 3: L2 norm of differences between methods and analytical solution at different times.

Method	$t = 1.0$	$t = 2.0$	$t = 3.0$	$t = 4.0$
PINN	3.337e-2	5.779e-2	8.293e-2	9.469e-2
Lax-Wendroff	1.918e-1	3.746e-1	5.663e-1	6.121e-1
Upwind	9.214e-1	1.281	1.493	1.633

5 Future Work and Limitations

A limitation of this project is that all of the points within the domain are equally able to be sampled when training the PINN. In a real world scenario, it is unlikely that this would be possible, which could limit the ability of the PINN to capture a system in a realistic application. It could be interesting to explore the ability of the network to learn when datapoints are more sparse. Furthermore, it would be beneficial to more systematically test different combinations of parameters when training the neural network. In this project I simply chose two important parameters (the number of nodes and the activation function) and altered these while keeping the other parameters fixed. A more thorough investigation would involve using formal optimisation techniques such as a random search, or a grid search to evaluate different parameter combinations.

6 Conclusions

This project, although applied to a very simple case, illustrates both the process of building PINNs as well as their ability to solve differential equations. Once trained, the PINN is able to outperform the numerical methods over almost the entire problem domain. While both of the numerical methods tested showed numerical diffusion and eventual instability when stepped forward in time, the PINN managed to very closely follow the analytical solution. Furthermore, the PINN was seen to be able to handle conditions which the numerical schemes are unable to, whether it be due to numerical instability or diffusion.

References

[1] Amirhossein Arzani and Scott T. M. Dawson. Data-driven cardiovascular flow modelling: examples and opportunities. *Journal of The Royal Society Interface*, 18(175):rsif.2020.0802, 20200802, February 2021.

[2] Francis Bach. Breaking the Curse of Dimensionality with Convex Neural Networks. *Journal of Machine Learning Res*, 18(19):1–53, 2017.

[3] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

[4] Zheyuan Hu, Khemraj Shukla, George Em Karniadakis, and Kenji Kawaguchi. Tackling the curse of dimensionality with physics-informed neural networks. *Neural Networks*, 176:106369, August 2024.

[5] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library, December 2019. arXiv:1912.01703 [cs].

[6] M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, February 2019.