Influence of activation functions on the convergence of Physics-Informed Neural Networks for 1D wave equation

Paweł Maczuga[1] and Maciej Paszyński[1]

[1]AGH University of Science and Technology, Kraków, Poland

Abstract. In this paper, we consider a model wave equation. We perform a sequence of numerical experiments with Physics Informed Neural Network, considering different activation functions, and different ways of enforcing the initial and boundary conditions. We show the convergence of the method and the resulting numerical accuracy for different setups. We show that, indeed, the PINN methodology can solve the problem efficiently and accurately the wave-equations without actually solving a system of linear equations as it happens in traditional numerical methods like, e.g., finite element or finite difference method. In particular, we compare the influence of selected activation functions on the convergence of the PINN method. Our PINN code is available on github: https://github.com/pmaczuga/pinn-comparison/tree/iccs.

Keywords: PINN · wave-equations · activation functions · initial conditions · boundary conditions · deep neural network

1 Introduction

Physics Informed Neural Networks (PINN) was introduced by George Karniadakis in 2019 [8]. In PINN, the neural networks represent the solution of PDE

$$u(x,t) = \text{PINN}(x,t) = A_1\sigma(A_2\sigma(A_3 \dots \sigma(A_n \begin{bmatrix} x \\ t \end{bmatrix} + y_n) + \dots + y_3) + y_2) + y_1 \quad (1)$$

where A_k is the weight matrix of layer k and y_k are the biases of this layer. In the training process, PINN learns the residual of the PDE and the residual of boundary and initial conditions by probing the residuals using sample points. The PINN can learn the solution of PDE without actually forming and solving a system of linear equations. The training process is usually performed until we reach a solution with a loss value less than the prescribed accuracy [4], or for the fixed number of epochs [6, 8]

The PINN method can also actually fail. For example, the PINN method, when applied to a problem of advection-diffusion, can only find a solution for small values of convection coefficients, and it fails when the problem is convection dominated. In that case, it is necessary to train the PINN first with the solutions for small values of convection and increase it gradually during the training

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_6

process [6]. Difficult problems require hard-coding of the boundary conditions into the computational model [2].

Using some references in the PINN literature, we can find some guidelines about the number of layers, the number of neurons per layer, the training rate, the number of epochs during the training, and the number of points used during the training. For example, in [1], the authors use a fully-connected neural network with eight hidden layers and 200 neurons per layer. They also use the training rate of 0.0001 followed by 0.0005 and 0.001, each for 50 epochs. Low number of epochs (for PINNs) comes from the fact, that the authors used huge number of points divided into mini-batches. Usually, increasing the number of points during the training process improves the convergence of the training, but it is also necessary to consider the distribution of points, especially for problems with singularities. From that paper, in different experiment, we can also read an example the number of collocation points for the training of PDE (7000 for the residual of the main PDE and 3000 for the zero divergence equation) as well as for the training of initial and boundary conditions (between 30 to 800 depending on the kind).

Incorporating initial and boundary conditions into PINN also requires some special considerations, often leading to difficulties [11]. The initial conditions and Dirichlet boundary conditions are often enforced in a hard way [10], while the Neumann boundary conditions are usually enforced in a weak way. It is also necessary to weigh the loss functions related to the PDE and to the boundary or initial conditions or use a larger number of samples on the boundary and initial condition residual terms [12].

As observed in [10], the convergence rate of the training process depends strongly on the activation function used. The authors observed that the training rate for the adaptive tanh activation function [3] was much faster than for the Switch activation function [9]. In the end, the accuracy of the obtained solution was the same, but the convergence rates were different.

In this paper, we focus on wave equations, and we employ the manufactured solution technique to check the convergence of the PINN with different activation functions. We refer to the best habits on how to set up the PINN parameters as described in the mentioned literature review.

2 Wave Equation

The main goal of this paper is to demonstrate the influence of different activation functions on the convergence of Physics Informed Neural Networks designed to solve the wave equation, mainly:

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

for $x \in [0,1]$ and $t \in [0,1]$. With the definition (1) in mind, relating $u(x,t) = \text{PINN}(x,t)$ with a neural network, we define the loss function based on the

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_6

residual of the PDE.

$$\mathrm{LOSS}_{\mathrm{PDE}}(x,t) = \left(\frac{\partial \mathrm{PINN}^2(x,t)}{\partial^2 x} - c^2 \frac{\partial \mathrm{PINN}^2(x,t)}{\partial^2 t} \right)^2, \tag{3}$$

where we differentiate the neural network, representing the solution, with respect to x and t. Now, we will discuss different possible initial and boundary conditions for the wave equation.

Specifically, we choose one of the following setups:

1) Zero boundary conditions and initial conditions that satisfies the boundary conditions

$$\begin{cases} u(x,0) & = A\sin(\varphi\pi x) \\ \frac{\partial u(x,0)}{\partial t} & = 0 \\ u(0,t) = u(1,t) & = 0 \end{cases} \tag{4}$$

These boundary and initial conditions result in the following family of exact solutions

$$u_{exact}(x,t) = A\sin(\varphi\pi x)\cos(c\ \varphi\pi t), \tag{5}$$

For these boundary and initial conditions, we define the following loss functions

$$\begin{aligned} \mathrm{LOSS}_{\mathrm{IC}}(x,0) &= \left(\frac{\partial \mathrm{PINN}(x,0)}{\partial t} - A\sin(\varphi\pi x) \right)^2, \\ \mathrm{LOSS}_{\mathrm{ICdt}}(x,0) &= \left(\frac{\partial \mathrm{PINN}(0,t)}{\partial t} - 0.0 \right)^2, \\ \mathrm{LOSS}_{\mathrm{BC0}}(0,t) &= \left(u(0,t) - 0.0 \right)^2, \\ \mathrm{LOSS}_{\mathrm{BC1}}(1,t) &= \left(u(1,t) - 0.0 \right)^2. \end{aligned} \tag{6}$$

2) Reflective boundary condition and corresponding initial solution

$$\begin{cases} u(x,0) & = Acos(\varphi\pi x) \\ \frac{\partial u(x,0)}{\partial t} & = 0 \\ \frac{\partial u(0,t)}{\partial x} = \frac{\partial u(1,t)}{\partial x} & = 0 \end{cases} \tag{7}$$

which results in the following family of exact solutions:

$$u_{exact} = A\cos(\varphi\pi x)\cos(c\varphi\pi t) \tag{8}$$

With this alternative boundary and initial conditions, we define the following loss functions

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_6

$$\begin{aligned}
\mathrm{LOSS}_{\mathrm{IC}}(x,0) &= \left(\frac{\partial \mathrm{PINN}(x,0)}{\partial t} - A\cdot\cos(\varphi\pi x)\right)^2, \\
\mathrm{LOSS}_{\mathrm{ICdt}}(x,0) &= \left(\frac{\partial \mathrm{PINN}(0,t)}{\partial t} - 0.0\right)^2, \\
\mathrm{LOSS}_{\mathrm{BC0}}(0,t) &= \left(\frac{\partial u(0,t)}{\partial t} - 0.0\right)^2, \\
\mathrm{LOSS}_{\mathrm{BC1}}(1,t) &= \left(\frac{\partial u(1,t)}{\partial t} - 0.0\right)^2. \qquad (9)
\end{aligned}$$

The total loss is

$$\mathrm{LOSS}(x,t) = \mathrm{LOSS}_{\mathrm{PDE}}(x,t)+\mathrm{LOSS}_{\mathrm{IC}}(x,t)+\mathrm{LOSS}_{\mathrm{ICdt}}(x,t)+\mathrm{LOSS}_{\mathrm{BC0}}(x,t)+\mathrm{LOSS}_{\mathrm{BC1}}(x,t) \qquad (10)$$

Now we can choose one of the above setups of initial and boundary conditions and select values of the following parameters:

- c - equation constant.
- A - initial amplitude.
- φ - number of "hills" and "valleys" in the initial condition.

3 Training

4 Numerical results

In this section, we illustrate the best numerical result obtained from the training of the PINN solver for the 1D wave equation. Figure 3 compares the exact solution with the solution obtained after training of PINN. Figure 2 illustrates how PINN managed to learn the initial condition. Figure 1 shows how the loss function evolved during 60,000 epochs of training. The difference between the exact and PINN solution is computed point-wise. We conclude that the PINN can learn the solution of the wave equation with the accuracy of the order of 0.01. The question if this accuracy is satisfactory will depend on particular applications. The details of the model parameters, the architecture of the neural network, the learning parameters, the loss functions used, and the convergence of the training for different setups are summarized in the next section.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_6

Algorithm 1: PINN training

1: Randomly select $x \in (0,1)$
2: Randomly select $t \in (0,1)$
3: Compute all loss functions:

$$\mathrm{LOSS}_{\mathrm{PDE}}(x,t),\ \mathrm{LOSS}_{\mathrm{IC}}(x,t),\ \mathrm{LOSS}_{\mathrm{ICdt}}(x,t),\ \mathrm{LOSS}_{BC0}(x,t),\ \mathrm{LOSS}_{BC1}(x,t)$$

4: **for** Number of epochs **do**
5: **for** Coefficients of matrices from neural network layers$\{A^p_{qr}\}_{q=1,\ldots,n_p;r=1,\ldots,m_p;p=1,\ldots,n}$ **do**
6: Compute the derivatives

$$\frac{\partial \mathrm{LOSS}_{\mathrm{PDE}}(x,t)(x,t)}{\partial A^p_{qr}} \qquad \frac{\partial \mathrm{LOSS}_{\mathrm{IC}}(x,t)}{\partial A^p_{qr}}$$
$$\frac{\partial \mathrm{LOSS}_{\mathrm{BC0}}(x,t)}{\partial A^p_{qr}} \qquad \frac{\partial \mathrm{LOSS}_{\mathrm{BC1}}(x,t)}{\partial A^p_{qr}}$$

7: Correct the coefficients of matrices from neural network layers

$$A^p_{qr} = A^p_{qr} + \eta_{\mathrm{PDE}} \frac{\partial \mathrm{LOSS}_{\mathrm{PDE}}(x,t)}{\partial A^p_{qr}} \qquad A^p_{qr} = A^p_{qr} + \eta_{\mathrm{IC}} \frac{\partial \mathrm{LOSS}_{\mathrm{IC}}(x,t)}{\partial A^p_{qr}}$$
$$A^p_{qr} = A^p_{qr} + \eta_{\mathrm{BC0}} \frac{\partial \mathrm{LOSS}_{\mathrm{BC0}}(x,t)}{\partial A^p_{qr}} \qquad A^p_{qr} = A^p_{qr} + \eta_{\mathrm{BC1}} \frac{\partial \mathrm{LOSS}_{\mathrm{BC1}}(x,t)}{\partial A^p_{qr}}$$

8: **end for**
9: **for** Coefficients of bias vectors from neural network layers$\{b^p_q\}_{q=1,\ldots,n_p;p=1,\ldots,n}$ **do**
10: Compute the derivatives

$$\frac{\partial \mathrm{LOSS}_{\mathrm{PDE}}(x,t)}{\partial b^p_q} \qquad \frac{\partial \mathrm{LOSS}_{\mathrm{IC}}(x,t)}{\partial b^p_q}$$
$$\frac{\partial \mathrm{LOSS}_{\mathrm{BC0}}(x,t)}{\partial b^p_q} \qquad \frac{\partial \mathrm{LOSS}_{\mathrm{BC1}}(x,t)}{\partial b^p_q}$$

11: Correct the coefficients of matrices from neural network layers

$$y^p_q = y^p_q + \eta_{\mathrm{PDE}} \frac{\partial \mathrm{LOSS}_{\mathrm{PDE}}(x,t)}{\partial y^p_q} \qquad y^p_q = y^p_q + \eta_{\mathrm{IC}} \frac{\partial \mathrm{LOSS}_{\mathrm{IC}}(x,t)}{\partial y^p_q}$$
$$y^p_q = y^p_q + \eta_{\mathrm{BC0}} \frac{\partial \mathrm{LOSS}_{\mathrm{BC0}}(x,t)}{\partial y^p_q} \qquad y^p_q = y^p_q + \eta_{\mathrm{BC1}} \frac{\partial \mathrm{LOSS}_{\mathrm{BC1}}(x,t)}{\partial y^p_q}$$

{Where $\eta_{\mathrm{PDE}}, \eta_{\mathrm{IC}}, \eta_{\mathrm{BC0}}, \eta_{\mathrm{BC1}}$ are training rates for different loss functions.}
12: **end for**
13: **end for**

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_6

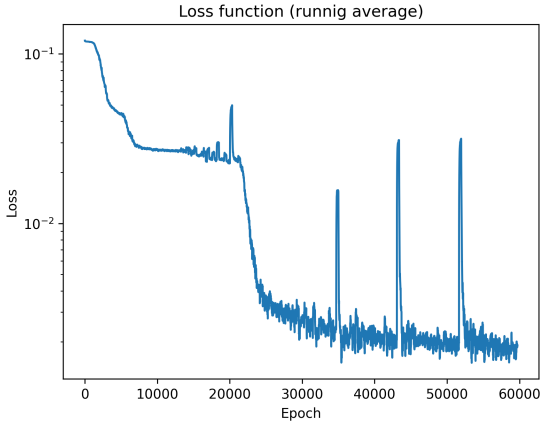


Fig. 1: Convergence of loss function during PINN training.

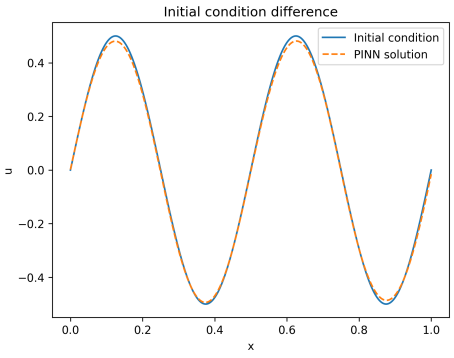
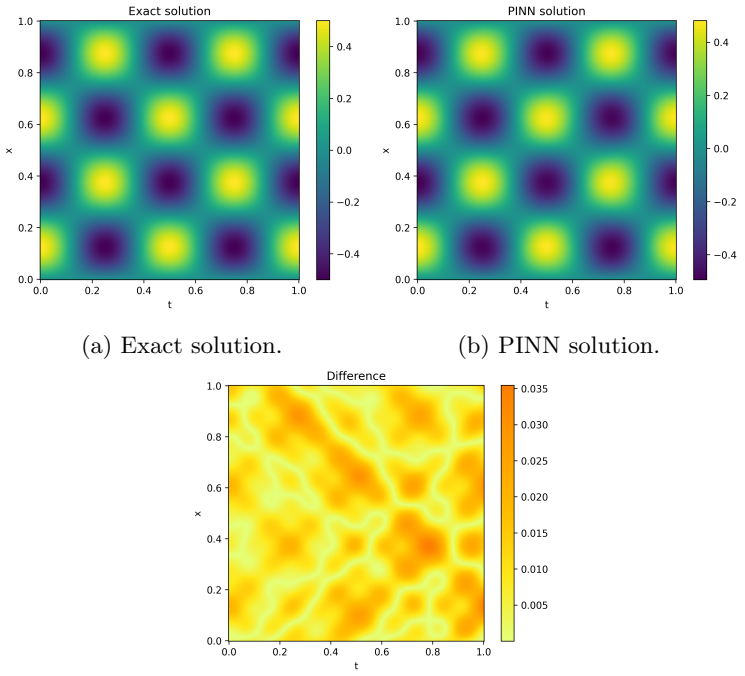


Fig. 2: Learning the initial condition by PINN.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_6



(a) Exact solution.

(b) PINN solution.

(c) Difference between solutions.

Fig. 3: Comparison of exact and PINN solution to the wave equation.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_6

5 Experiments

The experiments are divided into two parts. In the first set of numerical experiments, we created the PINN architecture and trained it to solve the wave equation with various boundary conditions and parameters. The goal of these experiments was to find three sets of parameters where the PINN is converging well to the exact solution and use these setups to test the convergence rate of PINN with different activation functions.

5.1 Parameters tuning

In our numerical experiments, we referred to setups introduced by authors of six research papers [1, 2, 6, 8, 10, 11]. Table 1 presents the literature review (the number of layers, the number of neurons per layer, different activation functions, the learning rate (LR), the number of epochs for training (unspecified in some papers) as well as the number of collocation points used for probing the loss functions of the PDE, the initial and boundary conditions).

Table 1: Review of PINN architectures. LR - learning rate, N_r, N_i, N_b are number of collocation points for residual, initial condition and boundary condition respectively. Dash represents data that was not specified in the referenced article.

Reference	Layers	Neurons	Activation	LR	epochs	Collocation points
[10]	3	20	Swish ($\beta = 1$)	-	-	-
[2]	5	100	tanh	10^{-3}	-	$N_r = 20,000$ $N_i = 50$ $N_b = 50$
[6]	4	50	tanh	10^{-4}	-	-
[8]	5	100	tanh	10^{-3}	-	$N_r = 20,000$ $N_i = 50$ $N_b = 50$
[1]	8	200	sin	1×10^{-3} 5×10^{-4} 1×10^{-4}	150	$N_r = 3 \times 10_6$
[1]	6	60	tanh	6×10^{-4}	3×10^5	$N_r = 7,000$
[11]	4	50	tanh	10^{-3} adaptive	40,000	-

We chose the PINN architecture, and we solved the wave equation with PINN. The architecture stays the same for all the experiments; only the parameters of the equation change. Namely, following Table 1, we selected the following PINN architecture and the training parameters:

- Feed-Forward fully connected network.
- 4 layers.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_6

- 80 neurons per layer.
- learning rate = 0.002.
- 50 000 epochs.
- Equally spaced collocation points.
  - For residual $N_r = 222\,500 = 150 \cdot 150$.
  - For initial conditions $N_i = 150$.
  - For boundary conditions $N_b = 150$.
- tanh activation function.

We train the above PINN with the sets of parameters:

- $c \in \{0.3, 0.5, 1.0, 2.0, 3.0\}$.
- $A \in \{0.5, 1.0, 2.0\}$.
- $\varphi \in \{2, 4, 6\}$.

We employed boundary conditions - zero and reflective.

5.2 Activation functions

The second set of numerical experiments concerns the investigation of the influence of different loss functions on the convergence of the training. We chose a couple of sets of parameters that yielded interesting results and trained the network again using the different activation functions. The rest of the network architecture stays exactly the same.

For activation functions we choose

- tanh, the most commonly used function in PINNs [2, 6, 8, 11] and indeed it yields very good results.
- sin in [8].
- sigmoid, rarely seen along PINNs.
- swish is adaptive activation function with the following formula: $swish = x \cdot sigmoid(\beta\, x)$ introduced in [9]. β is an adaptive parameter, that changes during training, similar to weights. In the context of PINNs it was used in [10], but with constant β.
- adaptive tanh with the formula $\text{atanh} = \tanh(\alpha\, x)$, where α is trainable parameter. Used in [3].

A very popular function for other neural networks other than PINNs is ReLU, which is not present in the points above. That is because, in this (and many other) equations, we need to calculate the second derivative of the neural network. However, ReLU is basically two linear functions merged together, and so its second derivative is always 0, and that, in turn, prevents learning.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_6

6 Results

As seen in Figures 4, 5 and 6, PINN with that architecture is able to learn simple cases very well. However, it struggles with more difficult ones. The reflective boundary condition is usually harder to learn, but that is not always the case. There are also strange spikes for reflective BC, where increasing A or ϕ actually decreases loss. Figures 7-9 shows the influence of different activation functions. The main issue is usually a very large loss connected to the initial condition. This means that PINN is able to learn the equation itself quite well, but for a slightly different initial condition. It is illustrated in Figure 10.

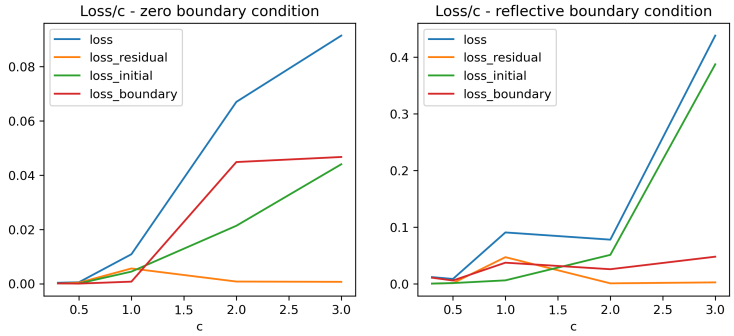


Fig. 4: Influence of equation parameter c on loss function. Other parameters are: $A = 1$, $phi = 4$.

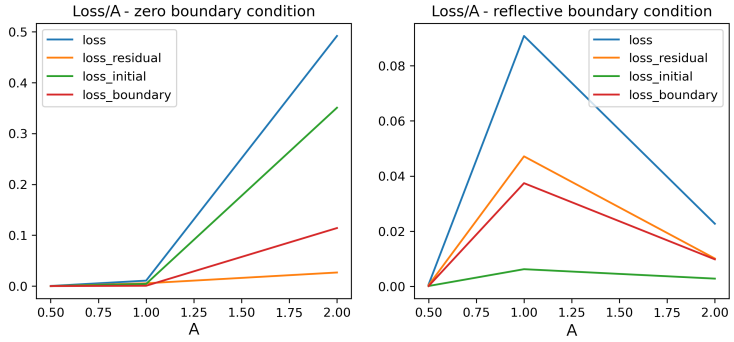


Fig. 5: Influence of initial condition amplitude on loss function. Other parameters are: $c = 1$, $phi = 4$. Strangely loss actually decreases as φ grows. Which means that somehow increasing it makes the equation easier to learn by the neural network. However more experiments are required in order to pin the exact reason, which is outside scope of this work.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_6

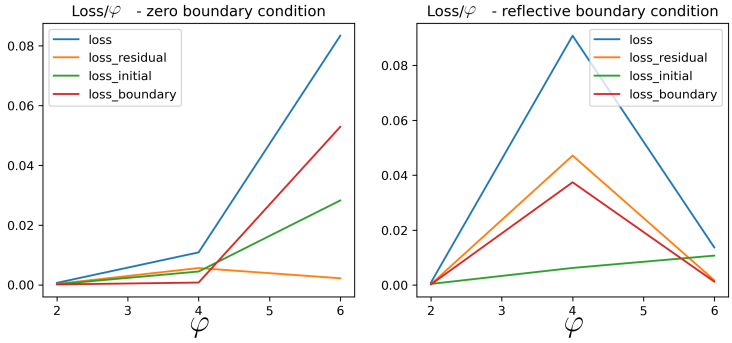
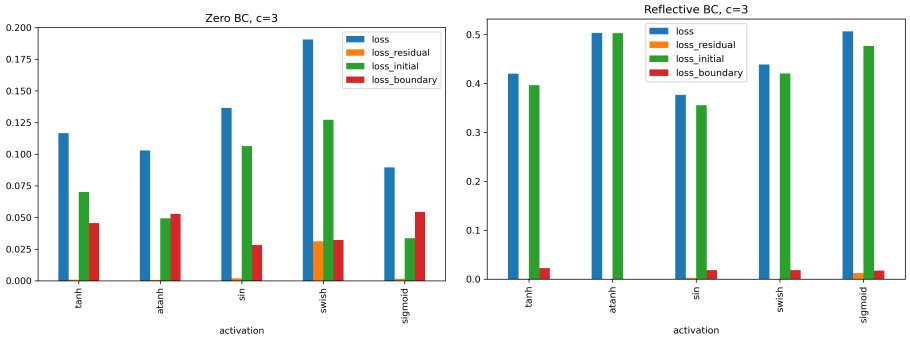


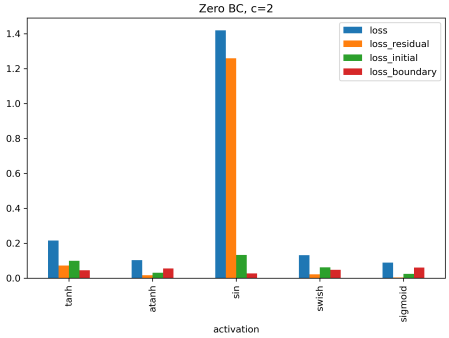
Fig. 6: Influence of *phi* parameter in initial condition on loss function. Other parameters are: $c = 1$, $A = 1$.



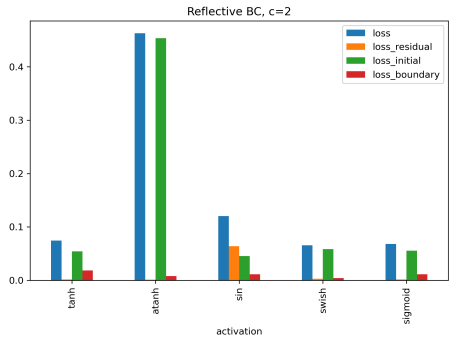
(a) Zero boundary condition. (b) Reflective boundary condition.

Fig. 7: Loss for different activation functions for zero boundary condition. Parameters are: $c = 3$, $A = 1$, $\phi = 4$.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_6

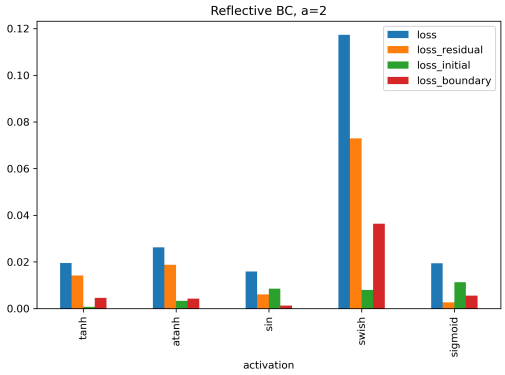


(a) Zero boundary condition with larger c.



(b) Reflective boundary condition with larger c.

Fig. 8: Loss for different activation functions, where one performs significantly worse than the others parameters: $c = 2$, $A = 1$, $\phi = 4$.



(a) Reflective boundary condition with larger amplitude - A.

Fig. 9: Loss for different activation functions, where one performs significantly worse than the others parameters: $c = 1$, $A = 2$, $\phi = 4$.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_6

7 Conclusions and future work

Surprisingly one of the best result is achieved by the sigmoid function, which is very rarely used for PINNs. As shown in Figures 8-9, there are cases where *atanh*, *sin*, and *swish* achieve much worse results than other activation functions. In the experiment, we performed, it was never the case for either *tanh* or sigmoid. Their effectiveness is as well bound to the specific problem, and there are cases where one or the other achieves better results. Interestingly the same can not be said about adaptive tanh, as it has huge downgrades in performance in some cases. In this case, the problem is connected to a very large loss for the initial condition, and the other two losses are rather low. Both sin and swish failed to learn the equation itself.

Although results in this paper are produced by solving a single PDE they can be in fact generalized to other PDEs as well. The approach is always the same regardless of the specific PDE that the network is trying to solve. That is the minimization of the equation's residual.

The PINNs themselves are likely the future of numerical simulations and worth exploring further. However, they do have some limitations at the moment. Specifically, PINNs are still achieving slightly worse results than state of the art finite element method. Furthermore, there still needs to be a complete theory concerning the Physics Informed Neural Networks. As seen in the results of this paper, there are aspects of PDE that are difficult to learn, most notably the initial condition and more difficult sets of equation parameters. There are ways to mitigate or even eliminate some of them. The most notable are:

- Enforcing the initial condition in a hard way [7].
- Using Variational PINNs (VPINNs), which vastly improves convergence [5].
- Pretraining the PINN for the easier set of parameters and then fine-tuning it using the proper ones [6].

Our future work can be divided in two parts. Firstly we will further explore the three aforementioned ideas and use that to solve 2D problems. Afterwards we plan to use PINNs to solve a real life problem, such as the simulation of tsunami wave caused by underwater earthquake.

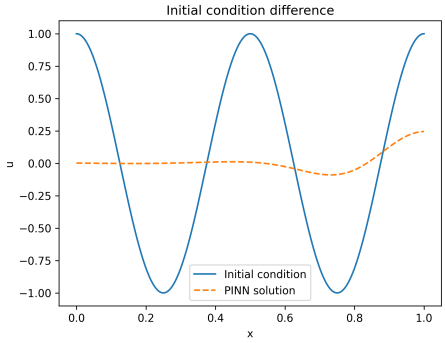
Acknowledgment

The Authors are thankful for support from the funds assigned to AGH University of Science and Technology by the Polish Ministry of Science and Higher Education. Research project supported by the program "Excellence initiative - research university" for the University of Science and Technology.

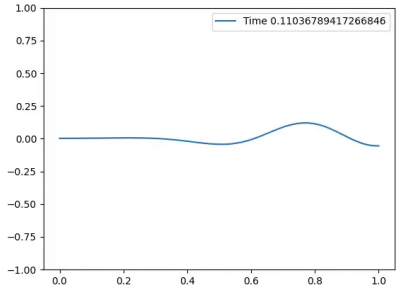
References

1. S. Cai, Z. Mao, Z. Wang, M. Yin, and G. E. Karniadakis. Physics-informed neural networks (PINNs) for fluid mechanics: a review. *Acta Mechanica Sinica*, 37(12):1727–1738, 2021.

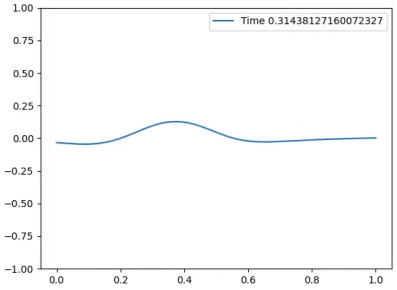
ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_6



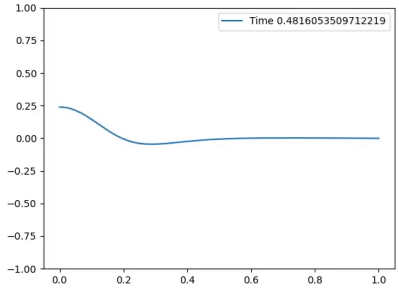
(a) Initial condition.



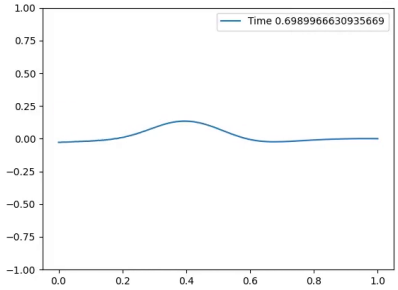
(b) $t = 0.11$.



(c) $t = 0.31$.



(d) $t = 0.48$.



(e) $t = 0.69$.

Fig. 10: Properly behaving wave, that failed to learned imposed initial condition.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_6

2. S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli. Scientific machine learning through physics–informed neural networks: Where we are and what's next. *Journal of Scientific Computing*, 92(3):88, 2022.
3. A. D. Jagtap, K. Kawaguchi, and G. E. Karniadakis. Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *Journal of Computational Physics*, 404:109136, 2020.
4. G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.
5. E. Kharazmi, Z. Zhang, and G. E. Karniadakis. Variational physics-informed neural networks for solving partial differential equations. *CoRR*, abs/1912.00873, 2019.
6. A. S. Krishnapriyan, A. Gholami, S. Zhe, R. M. Kirby, and M. W. Mahoney. Characterizing possible failure modes in physics-informed neural networks. *arXiv e-prints*, page arXiv:2109.01050, 2021.
7. L. Lu, R. Pestourie, W. Yao, Z. Wang, F. Verdugo, and S. Johnson. Physics-informed neural networks with hard constraints for inverse design. *SIAM Journal on Scientific Computing*, 43:B1105–B1132, 2021.
8. M. Raissi, P. Perdikaris, and G. 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, 2019.
9. P. Ramachandran, B. Zoph, and Q. V. Le. Searching for activation functions. *arXiv e-prints*, page arXiv:1710.05941, 2017.
10. L. Sun, H. Gao, S. Pan, and J.-X. Wang. Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data. *Computer Methods in Applied Mechanics and Engineering*, 361:112732, 2020.
11. S. Wang, Y. Teng, and P. Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43:A3055–A3081, 2021.
12. L. Yuan, Y.-Q. Ni, X.-Y. Deng, and S. Hao. A-pinn: Auxiliary physics informed neural networks for forward and inverse problems of nonlinear integro-differential equations. *Journal of Computational Physics*, 462:111260, 2022.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_6