Activation function trade-offs for training efficiency of Physics-Informed Neural Networks used in solving 1D Burgers' Equation

Analyzing the impact of the choice of adaptive activation function on the speed and accuracy of generating PDE solutions using PINNs

Rareș Mihail

Supervisors: Dr. Jing Sun, Dr. Alexander Heinlein, Dr. Tiexing Wang.

EEMCS, Delft University of Technology, The Netherlands
Shearwater GeoServices, UK

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
January 26, 2025

Name of the student: Rareș Mihail
Final project course: CSE3000 Research Project
Thesis committee: Dr. Jing Sun, Dr. Alexander Heinlein, Dr. Tiexing Wang, Dr. Hayley Hung

Abstract

Physics-Informed Neural Networks(PINNs) have emerged as a potent, versatile solution to solving both forward and inverse problems regarding partial differential equations(PDEs), accomplished through integrating laws of physics into the learning process. The applications of this new approach are endless, as these types of equations appear across numerous fields: fluids mechanics, quantum mechanics, electrochemistry and many others. Ever since their conception, researchers have continuously improved the flexibility and performance of PINNs through advancements in the architecture of neural networks, optimization algorithms, creative sampling methods and many more. As computational power and the interest of researchers grow, the revolutionary potential of PINNs is closer to fulfillment than ever. This paper aims to examine a small part of this evolutionary process, specifically the performance and flexibility of different activation functions used in the training of the PINN, as well as potential problems this approach could solve.

1 Introduction

Physics-Informed Neural Networks (PINNs) (Raissi, Babaee, and Givi 2019) are a class of machine learning models that integrate physical laws expressed as partial differential equations (PDEs) into the neural network training process. PINNs have gained significant attention due to their ability to solve forward (estimating the solution to a governing mathematical model) and inverse (learning the mathematical model's parameters from observed data) problems in scientific computing, where traditional numerical methods often struggle with high-dimensional spaces or ill-posed problems. This new tool has been extensively used in solving problems across the scientific spectrum, ranging from fluid mechanics (Raissi, Yazdani, and Karniadakis 2020, Sun and J.-X. Wang 2020), quantum mechanics (Jin, Mattheakis, and Protopapas 2022) and electromagnetism (Piao et al. 2024), to molecular dynamics (Oca Zapiain, Venkatraman, and M. Wilson 2023) and electrochemistry (X. Huang et al. 2025).

One of the challenges in training PINNs lies in achieving high accuracy while maintaining training efficiency. Activation functions, as a fundamental component of neural networks, play a crucial role in shaping the network's expressivity and optimization dynamics. Research on activation functions has demonstrated their impact on convergence rates, gradient stability, and solution accuracy, especially for nonlinear and stiff PDEs. Studies such as (Jagtap, Kawaguchi, and Karniadakis 2020) and (Kawaguchi, Jagtap, and Karniadakis 2020) have introduced novel activation functions such as adaptive and locally adaptive activation functions, providing insights into their limitations and benefits. (H. Wang, Song, and G. Huang 2024) extended this work by introducing adaptive sigmoidal activation functions tailored to PDE settings, reporting promising results on specific problem classes.

1.1 Structure of the Paper

The paper is organized as follows:

- **Section 1:** Introduces the research problem, key questions, and the motivation for exploring activation functions in PINNs.
- **Section 2:** Provides essential information about the general problem formulation that PINNs aim to solve, as well as insight into the inner workings and architecture of PINNs.
- **Section 3:** Details the methodology for training PINNs.
- **Section 4:** Explains the setup of the experiments and motivates the design choices made in order to generate meaningful results.
- **Section 5:** Presents the results and discusses performance comparisons among the tested activation functions.
- **Section 6:** Tackles the issue of the ethical aspects of the research conducted, as well as its reproducibility.
- **Section 7:** Provides an interpretation of the results with regards to the research questions.
- **Section 8:** Highlights the contributions of this paper.

1.2 Contribution

This work investigates how different activation functions impact the accuracy and training efficiency of PINNs when solving the 1D Burgers' Equation. The main research question is:

Which activation function optimizes the accuracy and training efficiency of Physics-Informed Neural Networks for solving the 1D Burgers' Equation?

To address this overarching question, the study explores the following sub-questions:

1. How do common activation functions (e.g., ReLU, tanh, sigmoid) perform in terms of accuracy and training time?
2. Can adaptive or learnable activation functions improve training dynamics compared to static ones?
3. How do improved activation functions compare to one another in terms of training speed and accuracy?

Preliminary findings suggest that adaptive and learnable activation functions outperform traditional static functions, both in terms of error reduction and convergence speed. What this research will focus on is the extent of the performance improvement that these adaptive activation functions bring, as well as whether there are any significant trade-offs.

2 Background

In this section we will take a close look at the most important components of the task at hand: the mathematical postulation of the general problem PINNs aim to solve, as well as the actual, explicit problem the PINN is trained to solve and the internal mechanisms of this type of neural networks.

2.1 Problem formulation

Consider a general PDE in some domain $\Omega \in R^d$ (where d is the dimension of the problem) possibly with boundary conditions, initial conditions and a governing equation:

$$\mathcal{N}\left(u(\mathbf{x},t);\theta\right) = 0, \quad \mathbf{x} \in \Omega,\, t \in \mathcal{T}$$

where $\mathcal{N}$ is the operator that represents the PDE, u(x, t) is the unknown solution to the PDE, which depends on the spatial and temporal coordinates x and t, and θ represents the parameters of the PINN.

In case the problem exhibits boundary and initial conditions, they need to be enforced through the following equations :

$$u(\mathbf{x},t) = g(\mathbf{x},t), \quad \mathbf{x} \in \partial\Omega,\, t \in \mathcal{T}$$

$$u(\mathbf{x},0) = h(\mathbf{x}), \quad \mathbf{x} \in \Omega$$

where g stands for the equation representing the *boundary conditions* (which is why x belongs to the points on the boundary of the domain) and h represents the equation depicting the *initial conditions* (which is why t = 0 on LHS and not taken as input on RHS).

The solution to these equations is approximated by a neural network $\hat{u}(\mathbf{x},t;\theta)$ with parameters θ:

$$u(\mathbf{x},t) \approx \hat{u}(\mathbf{x},t;\theta)$$

Following this conceptual framework for solving a PDE using PINNs, the next step is to apply it to our own equation. The PINN is trained to come up with a solution for the forward problem of 1D viscous Burgers' Equation.

The 1D Burgers' equation is a fundamental partial differential equation in fluid dynamics, nonlinear acoustics, and traffic flow. It describes the evolution of a scalar field, often representing velocity or a concentration, under the influence of both convective and diffusive processes. It is a simplification of the Navier-Stokes equations in the case of one spatial dimension and no external forces.

The general form of the 1D Burgers' equation is:

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

where *u(x,t)* is the unknown function (e.g., velocity) that depends on both space *x* and time *t* and *v* represents the kinematic viscosity (low kinematic vicosity means the fluid flows easily, while high kinematic viscosity means that the fluid flows more sluggishly). The first term on the left side represents advection (convective term) and the second term on the left side represents the nonlinearity. The right-hand side represents diffusion, more explicitly the dissipation of the field due to viscosity: if *v* is small, the system exhibits more pronounced non-linear effects, as opposed to when *v* is large, when the effects are more diffusive, smoothing.

Therefore, we will define the operator representing the PDE, $\mathcal{N}$, as:

$$\mathcal{N}(u) = \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2}$$

Now that we have a clear idea of the mathematical formulation of the problem at hand, it's time to take a look at the characteristics that allow a PINN to solve it.

2.2 PINN inner workings

We consider a neural network with D layers, out of which D-1 hidden layers and 1 output layer. Each i-th layer contains n_i neurons, and receives from the previous layer an output $x_i - 1 \in R^{n_{i-1}}$ which is transformed as such:

$$l_i := w_i * x_{i-1} + b_i$$

The w_i term represents the trainable weight matrix of the i-th layer, while the b_i term represents the trainable bias of the i-th layer. Both terms are sampled from independent and identically distributed samplings. Before the result of the equation is passed on to the next layer, an activation function is applied:

$$L_i := \sigma\big(l_i(x_{i-1})\big)$$

Finally, if we were to express the output of a neural network using these two equations, it would be a chain of compositions of alternating functions: the output of layer l_i serves as input for the global activation function, whose output serves as input for layer $l_i + 1$ and so on. The equation looks like this:

$$u_\Theta\big(n_0\big) = (l_D \circ \sigma \circ l_{D-1} \circ \sigma \circ ... \circ \sigma \circ l_1)(n_0)$$

2.3 Activation functions

The last crucial design choice when constructing the PINN is the activation function, the main topic of this paper. We will first take a look at some traditional activation functions that are widely used in related research. These will serve as benchmarks for the next topic of this section and the most important part of our research, the adaptive activation functions.

1. $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

   The *hyperbolic tangent* activation function is a smooth and continuous function that maps the input to values between -1 and 1. Since many PDE solutions, including ones for the 1D Burgers' Equation, have smooth and symmetric characteristics, *tanh* is a natural choice that helps PINNs to converge more efficiently during training. Additionally, this activation function is very stable for both low and high inputs, providing non-zero gradients across a broad input range.

2. $\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$

   Being one of the first activation function used in neural networks, thanks to its role and relevance in the field of computational neuroscience (H. R. Wilson and Cowan 1972), the *sigmoid* activation function serves as a foundational reference point for evaluating the performance of PINNs. Additionally, in the context of the

1D Burgers' Equation, the main drawback of this activation function, the *vanishing gradient problem* (Hochreiter 1998), is minimized by the limited range of the domain from which the data is sampled. Another characteristic of our PDE that makes this AF suitable is the low-dimensionality of the input data.

3. $\text{ReLU}(x) = \max(0, x)$

   As for the *rectified linear unit* activation function, its widespread use, driven by its simplicity and efficiency, make it a strong candidate as a benchmark activation function. However, its characteristic *dying neuron problem* is especially impactful when it comes to PDEs, even ones as simple as the 1D Burgers' Equation. The zeroing of negative values is also problematic, considering the nature of the x-input, which often takes values smaller than zero. Thanks to these well-known, well-documented problems of the *ReLU* activation function, it serves as an excellent indicator of the improvements that the other activation functions bring.

.

Based on previous research (Kawaguchi, Jagtap, and Karniadakis 2020), the AFs with the most potential to bring an improvement to PINN performance, and the ones that will be compared to the benchmark activation functions mentioned earlier, are:

1. layer-wise locally adaptive AF (LAAF) : $\sigma\left(w_j^i L_i\left(n^{i-1}\right)\right), \quad i = 1, 2, \ldots, D-1, \quad j = 1, 2, \ldots, N_i.$
   With the addition of an adaptable parameter to each layer, there's an extra D-1 parameters to be optimized during the training process. Therefore, every hidden layer has its own slope for the activation function (w_j^i).
2. neuron-wise locally adaptive AF (N-LAAF) : $\sigma\left(ma_j^i\left(L_i\left(n^{i-1}\right)\right)_j\right), \quad i = 1, 2, \ldots, D-1, \quad j = 1, 2, \ldots, N_i.$
   With the addition of an adaptable parameter to each neuron within each layer, there's an extra $\sum_{i=1}^{D-1} N_k$ parameters to be trained each iteration of the training process. As with the layer-wise one, each neuron has its own slope for the activation function (ma_j^i) (Kawaguchi, Jagtap, and Karniadakis 2020).

As you can see in the formulas for both locally adaptive activation functions, a *traditional, underlying activation function* (σ) still is part of the concept, but accompanied by slope terms present at 2 different levels of the neural network: at layer level and at neuron level. The exact choice of this underlying activation function is not trivial, therefore is discussed later in Section 5, where the choice made in this research paper is accompanied by proof and explanation.

2.4 Physics-informed loss function

In a normal neural network, an error metric such as cross entropy loss or mean squared error would be calculated with regards to this output and the ground truth, which would further be used in calibrating the model through backpropagation. When it comes to PINNs however, the loss is comprised of 3 separate terms, whose role is to embed the governing physical laws into the equation:

1. $L_{\text{BC}}(\theta)$ is the boundary condition loss, ensuring the solution matches the boundary data:

   $$L_{\text{BC}}(\theta) = \frac{1}{N_{\text{BC}}} \sum_{i=1}^{N_{\text{BC}}} \left|u(\mathbf{x}_i, t_i) - g(\mathbf{x}_i, t_i)\right|^2$$

   where N_{BC} represents the number of points used to enforce the boundary condition (they are sampled on the boundaries of the defined domain Ω), $u(\mathbf{x}_i, t_i)$ represents the output of the neural network for the i-th point and $g(\mathbf{x}_i, t_i)$ represents the boundary condition computed for the i-th point.

2. $L_{\text{IC}}(\theta)$ is the initial condition loss, ensuring the solution matches the initial data:

   $$L_{\text{IC}}(\theta) = \frac{1}{N_{\text{IC}}} \sum_{i=1}^{N_{\text{IC}}} \left|u(\mathbf{x}_i, 0) - h(\mathbf{x}_i)\right|^2$$

   where N_{IC} represents the number of points used to enforce the initial condition (they are sampled at the initial time *t = 0*), $u(\mathbf{x}_i, 0)$ represents the output of the neural network for the i-th point at time 0 and $h(\mathbf{x}_i)$ represents the initial condition computed for the i-th point at time 0.

3. $L_{\text{PDE}}(\theta)$ is the residual of the PDE evaluated at points in the domain Ω:

   $$L_{\text{PDE}}(\theta) = \frac{1}{N_{\text{PDE}}} \sum_{i=1}^{N_{\text{PDE}}} \left|\mathcal{N}(u(\mathbf{x}_i, t_i); \theta)\right|^2$$

   where N_{PDE} represents the number of collocation points sampled within the domain Ω and $\mathcal{N}$ represents the partial differential equation operator that is applied to the predicted solution generated by the neural network $u(\mathbf{x}_i, t_i); \theta)$, as described in section 2.1.

These 3 loss terms are combined as such in the total loss of the PINN:

$$L(\theta) = \lambda_1 \cdot L_{\text{PDE}}(\theta) + \lambda_2 \cdot L_{\text{BC}}(\theta) + \lambda_3 \cdot L_{\text{IC}}(\theta)$$

where λ_1, λ_2 and λ_3 are the weights that balance the importance of each loss term. This loss function is used to perform backpropagation and tune the parameters of the neural network through the use of auto-differentiation (Baydin et al. 2018).

3 Methodology

This chapter describes the methodology used to research the impact of different activation functions on the performance of PINNs for the 1D Burgers' Equation.

3.1 Training process

The training process of a PINN is mostly similar to that of a normal neural network, with two small but crucial differences: the use of autodifferentiation (Baydin et al. 2018), as an efficient and accurate way to evaluate function derivatives, and the loss function comprised of three separate weighted terms, as explained in section 2.2.

Therefore, the training process of the underlying PINN is comprised of the following steps:

1. **Problem formulation:** As explained in section 2.1, we are aiming to generate solutions for the forward problem of the 1D Burgers' PDE, using each activation function detailed in section 2.3.
2. **Designing the PINN:** The next crucial step is to layout the architecture of the PINN, as the choices made in this step have a crucial influence over the results. Therefore, the scheme of the PINN is detailed thoroughly in section 4.2.
3. **Physics-informed loss function:** The inner mechanics of the physics loss have already been highlighted in detail in section 2.4. What is however worth mentioning is the weighting of the 3 terms in the loss function : $\lambda_1 = 0.4$, $\lambda_2 = 0.3$, $\lambda_3 = 0.3$. This weight distribution was chosen with the two aspects in mind. Firstly, the PDE to be solved has relatively simple Dirichlet initial and boundary conditions, therefore the PINN has no issues in coming up with solutions that satisfy them. Secondly, as a consequence of the first point, assigning high importance to these conditions would dilute the results, making them harder to interpret. Hence, a bigger weight was assigned to the PDE loss, as this has the highest variance.
4. **Sampling the domain:** *Collocation points* are sampled, where the PDE is evaluated and the physics loss is computed. Despite a multitude of proven performance-enhancing sampling techniques (Wu et al. 2023), a uniformly random sampling distribution was chosen. The techniques mentioned earlier are particularly useful when dealing with high-dimensional PDEs, which is not our case. They also entail a higher computational cost to the already limited available hardware.
5. **Training the PINN:** First, hidden layer weights and biases are initialized. Then, the predicted solution $u_\Theta(x,t)$ is computed for a batch of 4096 collocation points. Based on these solutions, automatic differentiation is used to compute the derivatives found in the governing PDE, and the physics loss is computed. Finally, the gradients of the loss with respect to the network's parameters are calculated using automatic differentiation and the NN is optimized using Adam, as explained in section 4.2.

The visual representation of this process can be found in figure 1.

4 Experimental Setup

In this section we will discuss the practical steps and measures employed in order to generate clear, meaningful results that would later allow for insightful conclusions into the performance of the proposed activation functions. First, the training setup will be explained, then the design choices made during designing the Physics-Informed Neural Network will be clarified. Finally, a thorough inspection of the evaluation metrics used to quantify the results will be conducted.

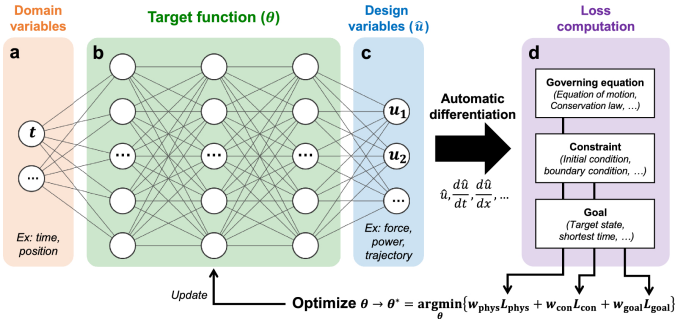


Figure 1: Training process of a PINN.

4.1 Training experiments

In order to efficiently and effectively train the model, the technique of Mini-Batch Gradient Descent (Ruder 2016) was used, each batch having a size of 64 training points. This choice came as a natural decision considering the vast amount of research demonstrating its superior performance (Masters and Luschi 2018) to its counterparts, Stochastic and Batch Gradient Descent.

Additionally, the model was trained for up to 30000 epochs. This particular number was chosen with two aspects in mind : the model needed time to get over the high initial errors caused by random initialization of weights, as well as giving enough time to the activation functions to differentiate amongst themselves based on their advantages and disadvantages discussed later in section 5.

In order to ensure the reproducibility and validity of the results, data about each training session is saved locally, allowing ease of access to the values of the evaluation metrics discussed in section 4.3. Additional scripts are provided that take as input the data generated during training and produce clear plots and figures demonstrating the results.

4.2 PINN architecture

While designing the PINN architecture, I made the following choices:

- **Network structure:** Fully connected feedforward neural network with 4 hidden layers and 256 neurons per layer. The choice to employ a relatively small FCNN stems from the simplicity of the Burgers' Equation, despite the drawbacks researched in the (S. Wang, Yu, and Perdikaris 2022) paper. The main disadvantage of this type of NN is an inherent spectral bias, more specifically a difficulty of the model to learn high-frequency data. It is safe to say that this characteristic of the FCNN does not negatively influence the validity of the results discussed in section 5, but rather the opposite.

- **Optimizer:** Adam (Kingma and Ba 2014) optimizer, supplemented with learning rate scheduling. The computationally efficient algorithm, tailored specifically for Stochastic Gradient Descent problems, was a sensible choice considering its ease of use and adaptability to both high and low-dimensional parameter spaces.
- **Loss function:** typical PINN loss function, made up of 3 separate terms: physics loss, initial conditions loss and boundary conditions loss, as detailed in section 2.2.
- **Hyperparameters:** I performed a grid search to identify optimal hyperparameters, including the learning rate and the initialization parameters for the activation functions that are adaptive.

4.3 Evaluation metrics

In this study, several evaluation metrics were employed to assess the impact of adaptive activation functions on the performance of PINNs in solving the 1D Burgers' equation.

The total loss was plotted against the number of epochs to monitor the network's convergence behavior during training, providing insight into the effectiveness of different activation functions in minimizing error over time. The number of epochs needed to reach a specific threshold error was recorded to evaluate the efficiency of each activation function, with a shorter training time indicating faster convergence. The lowest error achieved after 30,000 epochs was considered to compare the overall performance of the models and their ability to approximate the solution to the Burgers' equation. Additionally, the number of trainable parameters was tracked to assess the complexity of each network and determine if adaptive activation functions lead to more compact or efficient models. The time elapsed during training and the memory usage were also monitored, as these factors are crucial for practical implementation, especially for larger-scale problems. These metrics provide a comprehensive understanding of how adaptive activation functions, such as LAAF and N-LAAF, influence both the learning dynamics and the computational efficiency of PINNs.

5 Results and interpretation

The purpose of this section is to present and discuss the results of the experiments that were conducted as described in section 4, with data organized in plots and tables. In order to aid the reader in following the research questions and interpreting the results, we will first present a brief comparison of the traditional activation functions enumerated in section 2.3, which will be concluded with a definitive answer regarding the best performing of the three. Then, this activation function will be used both as a benchmark for comparing it to *LAAF* and *N-LAAF*, as well as the *underlying activation function*, mechanism explained in section 2.3.

5.1 Traditional activation functions

As mentioned before, we will first take a look at the performance of the three chosen traditional activation functions: *tanh*, *sigmoid* and *ReLU*.

Their performance is measured in Figure 2, which plots their accuracy as a function of number of epochs, more specifically their convergence speed. A quick look at the plot shows *tanh*'s superiority to its counterparts, *ReLU* performing the worst out of the three. Compared to *sigmoid*, the model trained using *tanh* also achieved a significant speed-up in convergence speed, reaching *sigmoid*'s minimum error after 30000 epochs after only 13000 epochs, a factor of improvement of **2.3** in speed. Additionally, this improved speed of convergence allowed the model trained using *tanh* to achieve a minimum error of over **100** times smaller compared to the one trained using *sigmoid*, and a minimum error of over **1600** times smaller compared to the one trained using *ReLU*.

This remarkable enhancement is explained by several factors:

- Solutions to the 1D Burgers' Equation are generally smooth in nature, making *tanh*, a smooth and continuous activation function, generally suited for approximating smooth functions.
- *ReLU* suffers from the *dying neuron problem*, which leads to gradients valued at zero for certain neurons, which is particularly problematic when solving PDEs, which often include complex relationships across the domain.
- *tanh* acts as a natural regularizer due to its saturation behaviour. This property contributes to the stabilization of training and prevention of overfitting.

Due to the clear improvement that the model trained using the *hyperbolic tangent* activation function brings compared to *ReLU* and *sigmoid*, *tanh* will be used in the next section both as a benchmark and as the base underlying function of *LAAF* and *N-LAAF*.

5.2 Adaptive activation functions

Now it's time to take a look at a detailed comparison of *tanh*, *LAAF* and *N-LAAF*, both adaptive activation functions being based on the *hyperbolic tangent* activation function.

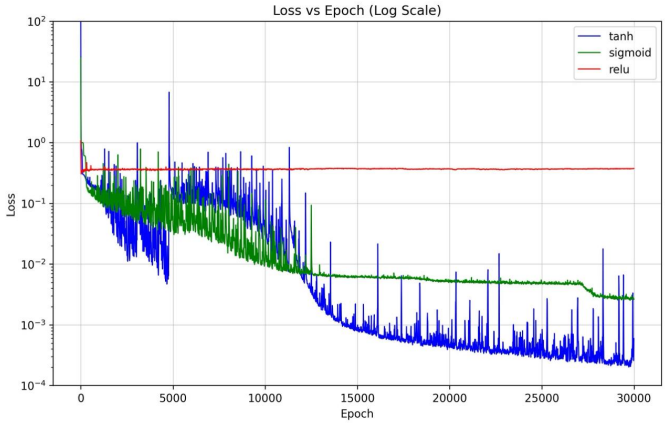


Figure 2: Aggregated loss history of the 3 proposed non-adaptive activation functions used in training the PINN for generating a solution for the 1D Burgers' Equation.

Method	lowest error after 30000 epochs
tanh	2.012e-04
sigmoid	2.531e-03
ReLU	3.3056e-01

Table 1: Epochs needed to reach a certain error threshold. Rows are the activation function used, columns are the error threshold.

Method	1e-2	1e-3	1e-4	1e-5
no-adaptive-tanh	3880	14180	31880	42120
LAAF	10360	13280	22240	33010
N-LAAF	2910	7570	9490	15480

Table 2: Epochs needed to reach a certain error threshold. Rows are the activation function used, columns are the error threshold.

Method	lowest error after 30000 epochs
no-adaptive-tanh	2.012e-04
LAAF	5.554e-05
N-LAAF	3.397e-06

Table 3: Epochs needed to reach a certain error threshold. Rows are the activation function used, columns are the error threshold.

Method	no. of trainable parameters
no-AF-tanh	198,401
LAAF	198,405
N-LAAF	199,425

Table 4: Different performance metrics for each activation function. The first column refers to the amount of time it took the PINN to train for 30000 epochs. The second column represents the total amount of computer memory used during training. The third column represents the amount of trainable parameters included in the PINN

Method	time elapsed	RAM used
no-AF-tanh	1h48m02s	20,79GB
LAAF	2h2m9s	22,85GB
N-LAAF	2h11m47s	25,20GB

Table 5: Different performance metrics for each activation function. The first column refers to the amount of time it took the PINN to train for 30000 epochs. The second column represents the total amount of computer memory used during training. The third column represents the amount of trainable parameters included in the PINN

The results displayed in Figure 3 are consistent with the findings in the preceding research papers on the topic of adaptive activation functions (Kawaguchi, Jagtap, and Karniadakis 2020), showing a clear superiority in accuracy for the adaptive AFs compared to their non-adaptive counterparts. The

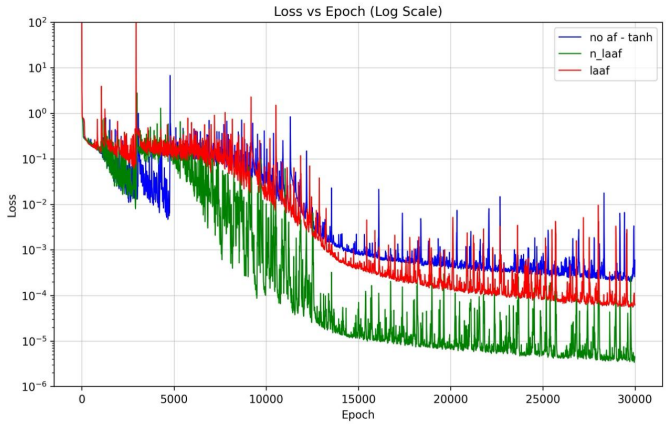


Figure 3: Aggregated loss history of the 3 candidate activation functions used in training the PINN for generating a solution for the 1D Burgers' Equation.

data in Table 2 provides further insight into the magnitude of the improvements, N-LAAF achieving an error of *1e-5* almost three times faster than non-adaptive tanh. This significant improvement is also displayed visually in the plots in Appendix A, where the point-wise error of the *non-adaptive tanh* is visibly higher (therefore more points that are more pronounced). What is most surprising however, is the meaningful difference in con-vergence speed between LAAF and N-LAAF, the latter being twice as fast as the former. The reason for this discrepancy is also the added specificity of N-LAAF, quantified in table 4, where it is clear that N-LAAF has 1000 more trainable pa-rameters. As for exactly how this added specificity explains this behaviour, an in-depth discussion is provided in section 7.

As a result of the improved convergence speed of the adaptible AFs, the minimum error achieved after 30000 epochs for each AF also varies greatly. As presented in table 3, *N-LAAF* is more than 100 times more accurate than *tanh*, while, compared to *LAAF*, it manages to be 61 times more accurate. On the other hand, this enhancement comes at a couple of trade-offs, as illustrated in table 5. In terms of time allocated for training, the model using *N-LAAF* took 23 minutes more than the one using *tanh*, a 21% increase, and 9 minutes more than the one using *LAAF*, a 7% increase in running time. Additionally, the *N-LAAF* model was also the most expensive memory-wise, using 10% more memory than the *LAAF* one and 20% more than the *tanh* one.

6 Responsible Research

In this section I will touch upon the ethical aspects of this research paper, as well as the reproducibility of the research methods employed.

6.1 Ethical Issues

Considering the fact that this research employs no usage of any kind of user data, and the target problem targets only the mathematical tool that PDEs are, and not the context they are used in, no ethical issues should arise from the work conducted and presented in this paper.

6.2 Reproducibility

The reproducibility of the conducted research is made as simple as possible, ensuring the integrity and validity of the findings. A ll a lgorithms e xplained a nd u sed a re implemented in the 3.8 version of Python, which is compatible with the most crucial frameworks and libraries used : Tensorflow, N umpy a nd M athplotlib. Additionally, all code is made publicly available in a GitHub repository (https://github.com/raresmihail123/rp-pinn-activation-functions), along with instructions for installation and usage.

7 Discussion

In this section we will thoroughly examine the possible underlying reasons behind the results discussed in section 5. All discussion will be centered around the motives behind the enhanced performance of the neuron-wise locally adaptive activation function.

7.1 Adaptability

The most obvious motivation for the results is the fine-grained adaptability that the neuron-wise parameters introduce. Thanks to this, the network is better capable of capturing localized features, such as sharp gradients or subtle variations in smooth regions, which are both present in typical solution to the 1D Burgers' Equation. These areas are clear also in the figures in appendix A, around x = 0 and the diagonal stripes exhibited at the start of the simulation. Therein also lies the reason for *layer-wise locally adaptive activation function's* and *tanh's* reduced performance, considering the deficiency in trainable parameters visible in table 4.

7.2 Spectral bias

In order to aid the reader in fully appreciating the extent of the results, this section will be split into two parts: first, a background on *spectral bias* and its emergence in neural networks will be provided, after which we will take a look at the implications observable on our results.

Explanation of the concept

Introduced first b y R ahaman e t a l. 2 019, a nd l ater proven mathematically by Cao et al. 2019, *spectral bias* is defined as a learning bias towards low complexity functions, exhibited by neural networks. More specifically, during training, neural networks have a tendency to first fit the low complexity components, placing priority on first learning the simple patterns that generalize across the entire data set. It is true that this behavior is not always present, especially in the cases when the original data is in a higher-dimension manifold, but it actually lies on a lower-dimensional manifold of complex shape embedded in the input space of the model. This, however, is not the case, considering the two dimensional character of our data.

In the context of partial differential equations, these "low complexity functions" translate directly to the areas of the domain where the function is *low frequency*. This *low frequency* property is characterized by smoothness and the absence of strong local fluctuations. This is extremely relevant to our interpretation of the results, as discussed in the next section.

The impact of spectral bias on our results

In light of the discussion in the previous section, we need to take a closer look at the 1D Burgers' Equation in relation to PINNs. The *low-frequency areas* are easily observable in the figures i n Appendix A, e specially t he g raphs detailing point-wise errors. Here we can observe that there are no errors around the low-frequency data points, where $u(x, t)$ changes gradually, or where the viscosity is high and the diffusion term dominates. On the other hand, areas containing high-frequency data points, in the vicinity of steep gradients around the value x = 0, are where the majority of the error builds up. The additional trainable parameters are proven to be quite effective in overcoming this difficulty, hence the significant improvement in performance for N-LAAF.

7.3 Drawbacks

As for the drawbacks of using *N-LAAF* instead of its counterparts, the discussion is limited to computational complexity. Considering the limited hardware these experiments were run on, the relatively insignificant difference in time and memory and the much more significant discrepancy in performance, using *N-LAAF* in PINN architecture is certainly advantageous.

7.4 Further research

If we were to extrapolate the previous discussion, the enhanced model capacity and dynamic adjustment during training characteristic to *N-LAAF* would be effective in solving other PDEs as well, such as the Boltzmann or nonlinear Schrodinger equations. The extent of the improved performance when solving these complex PDEs using adaptive activation functions is certainly a worthwhile topic for further research.

8 Conclusions and Future Work

This study explores the performance of different activation functions used in PINNs for solving the 1D Burgers' Equation, aiming to determine, in order: how common activation functions perform on the task of solving forward problems for PDEs, whether adaptive activation functions can improve the training dynamic compared to traditional ones, and trade-offs between adaptive activation functions.

The research conducted provided a strong basis for favoring adaptive activation functions over common ones, specifically the *neuron-wise locally adaptive activation function*, which performed better than it's *layer-wise adaptive activation function* counterpart, with minimal computational trade-offs. It is also notable that the usage of adaptive activation functions is effective to some degree in combating the inherent problem of spectral bias (S. Wang, Yu, and Perdikaris 2022), thanks to the enhanced model capacity introduced by the additional trainable parameters.

A worthwhile subject of further research would be to analyze the extent of the superiority of *N-LAAF* when solving for other, more complex PDEs, such as the nonlinear Schrodinger or Boltzmann equations, as well as investigating the degree to which adaptive activation functions combat spectral bias.

References

Baydin, Atilim Gunes et al. (2018). "Automatic Differentiation in Machine Learning: A Survey". In: *Journal of Machine Learning Research* 18, pp. 1–43.

Cao, Yuan et al. (2019). "Towards Understanding the Spectral Bias of Deep Learning". In: *arXiv preprint arXiv:1912.01198*. URL: https://doi.org/10.48550/arXiv.1912.01198.

Hochreiter, Sepp (1998). "The vanishing gradient problem during learning recurrent neural nets and problem solutions". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02, pp. 107–116.

Huang, Xujia et al. (2025). "Enriched Physics-informed Neural Networks for Dynamic Poisson-Nernst-Planck Systems". In: *arXiv:2402.01768*. Accessed: 2025-01-24. URL: https://doi.org/10.48550/arXiv.2402.01768.

Jagtap, Ameya D., Kenji Kawaguchi, and George Em Karniadakis (2020). "Adaptive activation functions accelerate convergence in deep and physics-informed neural networks". In: *Journal of Computational Physics*.

Jin, H., M. Mattheakis, and P. Protopapas (2022). "Physics-Informed Neural Networks for Quantum Eigenvalue Problems". In: pp. 1–8. DOI: 10.1109/IJCNN55064.2022.9891944.

Kawaguchi, Kenji, Ameya D. Jagtap, and George Em Karniadakis (2020). "Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks". In: *Journal of Computational Physics*.

Kingma, Diederik P. and Jimmy Ba (2014). "Adam: A Method for Stochastic Optimization". In: *arXiv preprint arXiv:1412.6980*. Published as a conference paper at the 3rd International Conference for Learning Representations (ICLR), San Diego, 2015. DOI: 10.48550/arXiv.1412.6980. URL: https://arxiv.org/abs/1412.6980.

Masters, Dominic and Carlo Luschi (2018). "Revisiting Small Batch Training for Deep Neural Networks". In: *arXiv preprint arXiv:1804.07612*. URL: https://arxiv.org/abs/1804.07612.

Oca Zapiain, David Montes de, Aditya Venkatraman, and Mark Wilson (2023). "Accelerating Charge Estimation in Molecular Dynamics Simulations Using Physics-Informed Neural Networks: Corrosion Applications". In: DOI: 10.21203/rs.3.rs-4844493/v1. URL: https://doi.org/10.21203/rs.3.rs-4844493/v1.

Piao, S. et al. (Oct. 2024). "A Domain-Adaptive Physics-Informed Neural Network for Inverse Problems of Maxwell's Equations in Heterogeneous Media". In: *IEEE Antennas and Wireless Propagation Letters* 23.10, pp. 2905–2909. DOI: 10.1109/LAWP.2024.3413851.

Rahaman, Nasim et al. (2019). "On the Spectral Bias of Neural Networks". In: *Proceedings of the 36th International Conference on Machine Learning*. Proceedings of Machine Learning Research 97, pp. 5301–5310.

Raissi, Maziar, Hessam Babaee, and Peyman Givi (2019). "Deep learning of turbulent scalar mixing". In: *Physical Review Fluids* 4.12, p. 124501. DOI: 10.1103/PhysRevFluids.4.124501. URL: https://doi.org/10.1103/PhysRevFluids.4.124501.

Raissi, Maziar, Alireza Yazdani, and George Em Karniadakis (2020). "Hidden fluid mechanics: learning velocity and pressure fields from flow visualizations". In: *Science* 367.6481, pp. 1026–1030. DOI: 10.1126/science.aaw4749. URL: https://doi.org/10.1126/science.aaw4749.

Ruder, Sebastian (2016). "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747*. Added derivations of AdaMax and Nadam. URL: https://doi.org/10.48550/arXiv.1609.04747.

Sun, Luning and Jian-Xun Wang (2020). "Physics-constrained Bayesian neural network for fluid flow reconstruction with sparse and noisy data". In: *arXiv preprint arXiv:2001.05542*. eprint: 2001.05542. URL: https://arxiv.org/abs/2001.05542.

Wang, Honghui, Shiji Song, and Gao Huang (2024). "Learning Specialized Activation Functions for Physics-Informed Neural Networks". In: *Journal of Machine Learning Research*.

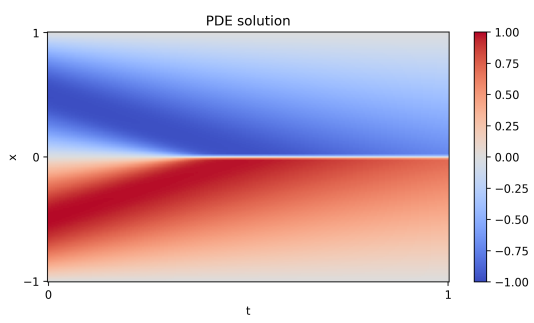
Wang, Sifan, Xinling Yu, and Paris Perdikaris (2022). "When and why PINNs fail to train: A neural tangent kernel perspective". In: *Journal of Computational Physics* 449, p. 110768. DOI: 10.1016/j.jcp.2021.110768. URL: https://www.sciencedirect.com/science/article/abs/pii/S0021999121008695.

Wilson, H. R. and J. D. Cowan (1972). "Excitatory and inhibitory interactions in localized populations of model neurons". In: *Biophysical Journal* 12, pp. 1–24. DOI: 10.1016/S0006-3495(72)86068-5.

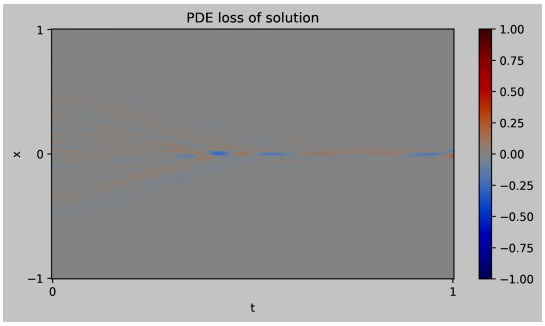
Wu, Chenxi et al. (2023). "A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks". In: *Computational Methods in Applied Mechanics and Engineering* 403, p. 115671. DOI: 10.1016/j.cma.2022.115671. URL: https://www.sciencedirect.com/science/article/abs/pii/S0045782522012020.

A Appendix

A.1 Visual representation of solutions and errors for 1D Burgers' Equation

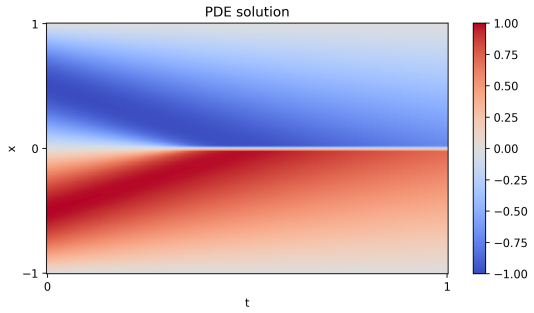


(a) tan-h, no adaptive

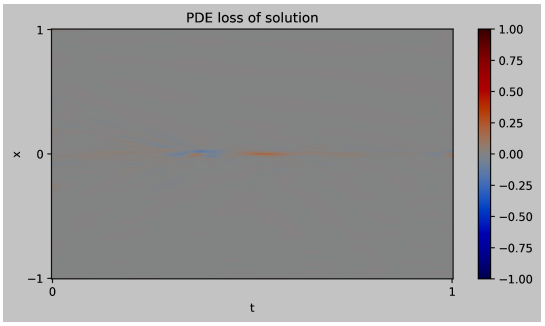


(b) Point-wise error.

Figure 4: Side-by-side visualization of PINN training and loss for tanh.

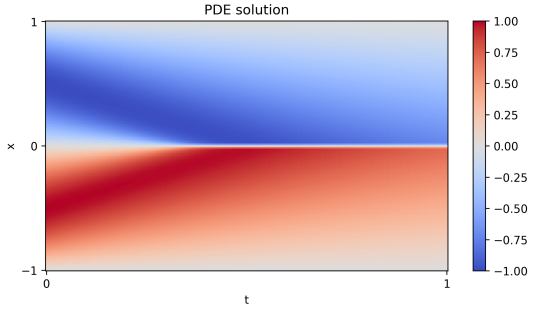


(a) LAAF.

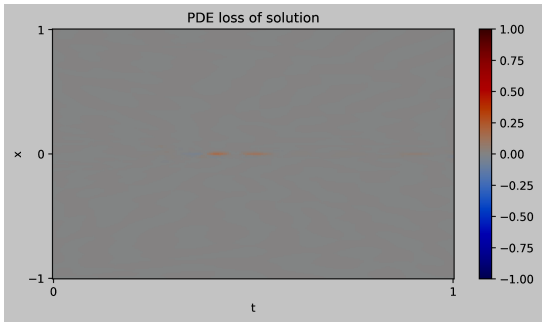


(b) Point-wise error.

Figure 5: Side-by-side visualization of PINN training and loss for LAAF.



(a) N-LAAF



(b) Point-wise error.

Figure 6: Side-by-side visualization of PINN training and loss for N-LAAF.