Analysis on Solving Differential Equations using Neural Networks

Naitik Agrawal
Department of Mathematical Sciences, Indian Institute of Technology (BHU), Varanasi, India
naitik.agrawal.mat23@iitbhu.ac.in

***Abstract*—Neural networks have been recognized to be a powerful tool for solving differential equations, which opened the way of the approach to complicated mathematical problems. In this Project under the guidance of Dr. Santwana Mukhopadhyay, we have discussed neural network based approaches, including Lagaris method and PINNs. We have revisited here results on the Lagaris approach, which generally holds good only for cases at a simpler level but is restricted when more complex systems are taken into consideration. Adding a hybrid mixture of experts and examining the activation function in several ways, we were able to push these capabilities even further into nonlinear problem-solving, such as the Van der Pol and Duffing equations. Although attempts to implement Hermite Neural Networks for more advanced systems fell short, PINNs proved effective in tackling higher-complexity equations.This study provides practical insights into the implementation of neural network methods, contributing to their advancement in scientific computing.**

***Index Terms*—Neural Networks, Differential Equations, Physics-Informed Neural Networks, Universal Approximation Theorem, Activation Functions. ANN. Lagaris**

I. Introduction

Differential equations are fundamental for modeling much of the physical, biological, and engineering experience. They play important roles in simulating heat transfer, as well as understanding what happens in oscillatory systems, to name but two, in a wide range of scientific investigation through to industrial applications. Traditional numerical methods, such as Runge-Kutta algorithms, have long been the standard for solving these equations. However, the complexity of modern problems now demands fresh approaches that can deal with complex equation and large computational domains.

This project extends the Lagaris method by introducing architectural advancements such as a mixture of experts and incorporation of a wide variety of activation functions. Such features bring even improved stability and efficiency in training, making the method efficient to solve problems with nonlinear equations, like systems like Van der Pol and Duffing. PINNs are utilized to solve equations with even greater complexity by imposing physical constraints directly within the learning process. By integrating these advancements, this work aims to demonstrate how neural networks can offer a robust, flexible, and efficient alternative for solving differential equations.

II. Lagaris

The Lagaris method leverages neural networks to approximate solutions to ordinary differential equations (ODEs). It constructs a trial solution $\hat{u}(x;\theta)$ that satisfies the boundary or initial conditions and minimizes a loss function to align with the governing differential equation.

Mathematical Formulation

Consider an ODE of the form:

$$\frac{d^n u}{dx^n} = f(x, u, u', \ldots, u^{(n-1)}),$$

where x represent the independent variable and $u(x)$ the solution to the differential equation. The trial solution $\hat{u}(x;\theta)$, parameterized by the neural network weights θ, is expressed as:

$$\hat{u}(x;\theta) = A(x) + B(x) \cdot N(x;\theta),$$

where:

- $A(x)$, $B(x)$ are function to ensure trial solution satisfies the boundary conditions .
- $N(x;\theta)$ is the output of the neural network, which is adjusted to satisfy the differential equation.

Loss Function

The loss function $\mathcal{L}$ measures the residuals of the ODE at collocation points $\{x_i\}_{i=1}^{M}$:

$$\mathcal{L}(\theta) = \sum_{i=1}^{M} \left(\frac{d^n \hat{u}(x_i;\theta)}{dx^n} - f(x_i, \hat{u}(x_i;\theta), \hat{u}'(x_i;\theta), \ldots) \right)^2 .$$

Minimizing $\mathcal{L}(\theta)$ ensures that $\hat{u}(x;\theta)$ approximates $u(x)$ while satisfying the differential equation constraints.

III. Lagaris Examples

In this section, we provide an overview of the few examples from Lagaris et al.'s work, demonstrating the application of artificial neural networks (ANNs) to solve various differential equations.

A. Example 1: First-Order ODE

The differential equation under consideration is:

$$\frac{d\Psi}{dx} + \left(x + \frac{1+3x^2}{1+x+x^3} \right) \Psi = x^3 + 2x + x^2 \frac{1+3x^2}{1+x+x^3},$$

$$\Psi(0) = 1, \quad x \in [0,1].$$

Analytical Solution:

$$\Psi_a(x) = \frac{e^{-x^2/2}}{1 + x + x^3} + x^2$$

Trial Solution:

$$\Psi_t(x) = 1 + xN(x; \mathbf{p})$$

Results: The performance of the trial solution was evaluated with different activation functions. Below, we compare the approximated solution with the analytical solution and analyze the residuals.

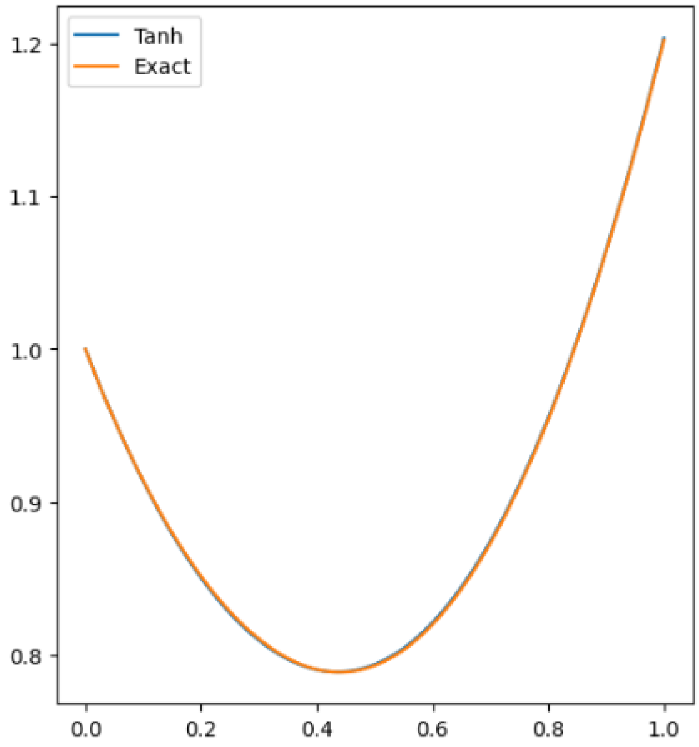

Fig. 1: Approximation vs. exact solution for Example 1.

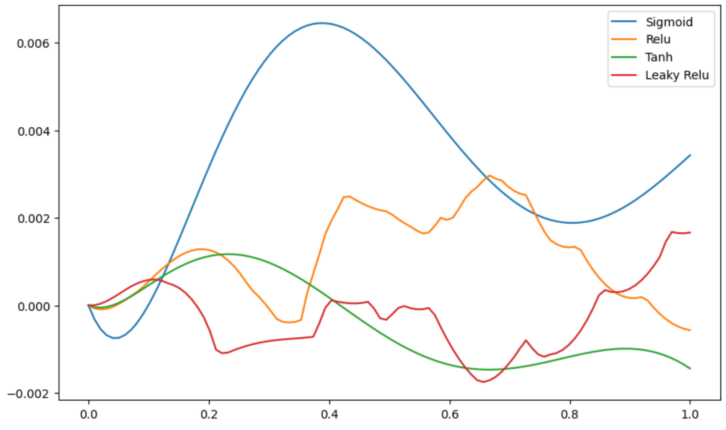

Fig. 2: Residual plot for different activation functions.

Activation Function	Average Error
Sigmoid	0.006
ReLU	0.002
Tanh	**0.001**
Leaky ReLU	0.0015

TABLE I: Comparison of average errors for different activation functions. The Tanh activation function achieved the lowest error, demonstrating its superior performance for this case.

B. Example 2: Second-Order ODE

The differential equation under consideration is:

$$\frac{d^2\Psi}{dx^2} + \frac{1}{5}\frac{d\Psi}{dx} + \Psi = -\frac{1}{5}e^{-(x/5)}\cos x$$

$$\Psi(0) = 0, \frac{d\Psi}{dx} = 1, \quad x \in [0, 2].$$

Analytical Solution:

$$\Psi(x) = e^{-(x/5)}\sin(x)$$

Trial Solution:

$$\Psi_t(x) = x + x^2 N(x, \mathbf{p})$$

Results: For this equation i have Tanh activation function

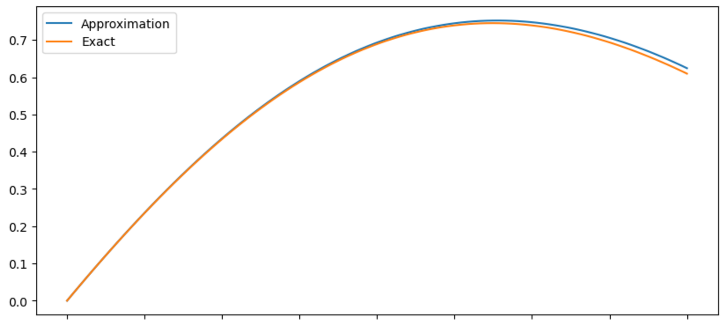

Fig. 3: Approximation vs. exact solution

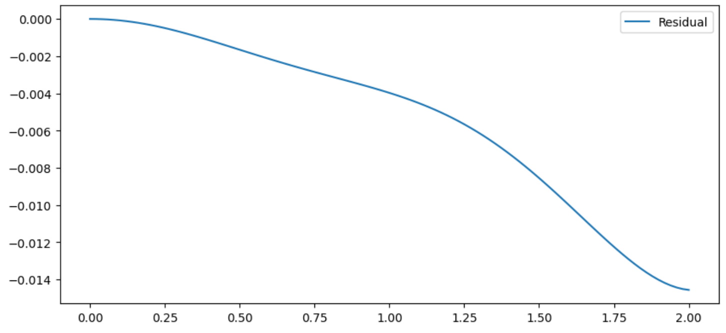

Fig. 4: Residual plot

C. Example 3: Coupled First-Order ODE

The differential equation under consideration is:

$$\frac{d\Psi_1}{dx} = \cos(x) + \Psi_1^2 + \Psi_2 - \left(1 + x^2 + \sin^2(x)\right)$$

$$\frac{d\Psi_2}{dx} = 2x - \left(1 + x^2\right)\sin(x) + \Psi_1\Psi_2$$

$$\Psi_1(0) = 0, \quad , \Psi_2(0) = 1, \quad x \in [0, 3].$$

Analytical Solution:

$$\Psi_1(x) = \sin(x), \quad \Psi_2(x) = 1 + x^2$$

Trial Solution:

$$\Psi_{t_1}(x) = xN_1(x; \mathbf{p_1}), \quad \Psi_{t_2}(x) = 1 + xN_2(x; \mathbf{p_2})$$

Results: The performance of the trial solution was evaluated with different activation functions. Tanh found to be best among them

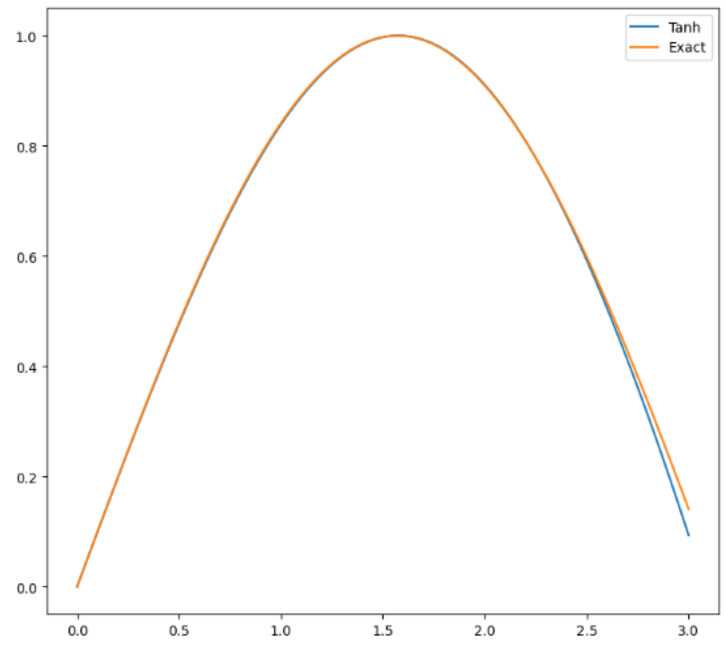

Fig. 5: Approximation vs. exact solution for Ψ_{t_1}

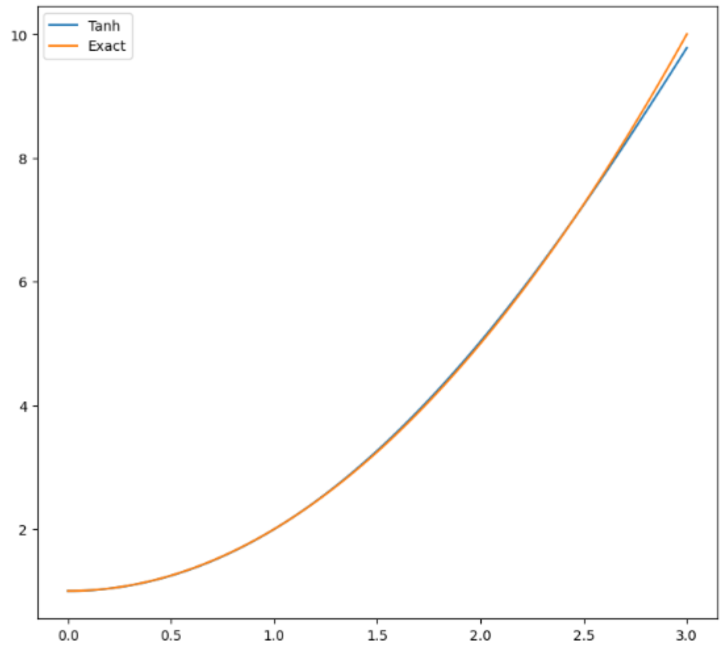

Fig. 6: Approximation vs. exact solution for Ψ_{t_2}

D. Example 4: PDE

The differential equation under consideration is:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = e^{-x}(x - 2 + y^3 + 6y)$$

$$\frac{d\Psi_2}{dx} = 2x - \left(1 + x^2\right)\sin(x) + \Psi_1\Psi_2$$

$$\Psi(1, y) = (1 + y^3)e^{-1}, \quad \Psi(x, 0) = xe^{-x}, \quad \Psi(0, y) = y^3$$

$$\Psi(x, 1) = e^{-x}(x + 1), \quad x, y \in [0, 3].$$

Analytical Solution:

$$\Psi_a(x, y) = e^{-x}(x + y^3)$$

Trial Solution:

$$\Psi_t(x, y) = A(x, y) + x(1 - x)y(1 - y)N(x, y, \vec{P})$$

Results: The performance of the trial solution was evaluated with different activation functions. Relu found to be best among them

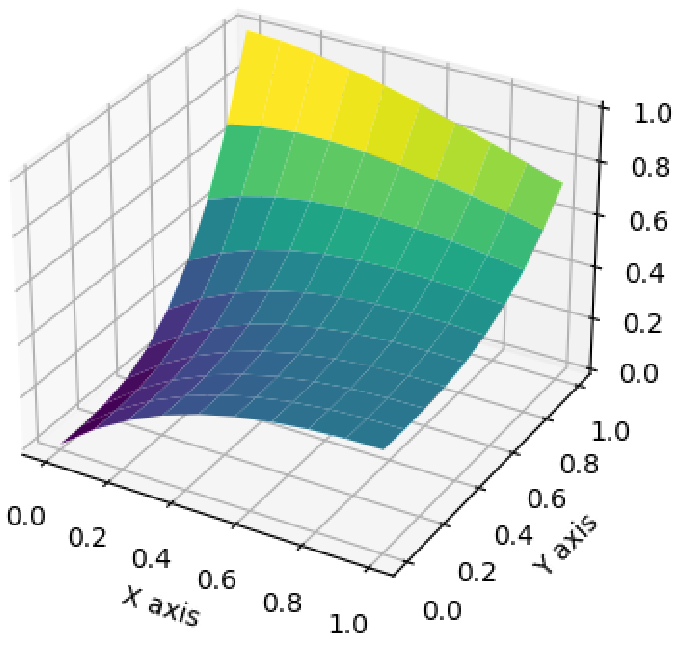

Fig. 7: Approximation vs. exact solution

IV. Runge-Kutta 4th Order Method (RK4)

The Runge-Kutta 4th order method (RK4) is a numerical technique for approximating solutions to ordinary differential equations (ODEs) of the form:

$$\frac{dy}{dx} = f(x, y), \quad y(x_0) = y_0$$

where $y(x)$ is the function to be approximated. RK4 is especially popular because it offers a good balance between accuracy and computational efficiency.

1) Mathematical Formulation: The RK4 method approximates $y(x)$ at successive points $x_0, x_1, \ldots, x_n$ by calculating intermediate slopes that better estimate the behavior of y over each interval. For a step size h, the update rule for moving from x_n to $x_{n+1} = x_n + h$ is given by:

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

where:

$$\begin{aligned} k_1 &= hf(x_n, y_n) \\ k_2 &= hf\left(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right) \\ k_3 &= hf\left(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right) \\ k_4 &= hf(x_n + h, y_n + k_3) \end{aligned}$$

Each k_i term represents a slope estimate:

- k_1: The slope at the beginning of the interval.
- k_2: The slope at the midpoint of the interval, using k_1 to adjust y_n.
- k_3: Another slope at the midpoint, but using k_2 to refine y_n.
- k_4: The slope at the end of the interval, using k_3 for adjustment.

The final update combines these slopes with specific weights, giving more weight to the midpoints (k_2 and k_3), which improves the accuracy of the estimate.

V. Van der Pol-Duffing Oscillator Equation

The Van der Pol-Duffing Oscillator is a nonlinear differential equation that combines the dynamics of the Van der Pol oscillator and the Duffing oscillator into a single equation. This model is widely used in physics and engineering to study nonlinear oscillatory systems, which often exhibit complex behavior such as limit cycles, bifurcations, and chaos.

The combined equation is expressed as:

$$\frac{d^2\Psi}{dt^2} - \mu\left(1 - \Psi^2\right)\frac{d\Psi}{dt} + \alpha\Psi + \beta\Psi^3 = F\cos(\omega t)$$

where:

- Ψ: The state variable of the system,
- μ: The nonlinear damping coefficient (from the Van der Pol component),
- α: The linear stiffness coefficient,
- β: The nonlinear stiffness coefficient (from the Duffing component),
- $F\cos(\omega t)$: The external driving force, with F being the amplitude and ω the angular frequency of the force.

The Van der Pol-Duffing oscillator exhibits rich and complex dynamics, including periodic, quasi-periodic, and chaotic solutions, depending on the values of the parameters.

We have used Runge-Kutta method to compare Neural Network approximate solution.

VI. ANN Lagaris Method for Solving the Oscillator

The ANN-based Lagaris method was initially applied to solve the Van der Pol-Duffing oscillator. However, significant difficulties were encountered:

- The ANN failed to capture the intricate oscillatory patterns of the solution accurately.
- Increasing the network depth and complexity did not significantly improve the results, as the model struggled to converge, especially for chaotic regimes.

VII. Mixture of Experts

The Mixture of Experts is a powerful machine learning framework designed to address the limitations of monolithic models in solving complex, high-dimensional problems. Unlike traditional approaches that rely on a single model to approximate the entire solution space, the MoE framework employs a collection of specialized models, referred to as *experts*, each trained to handle a specific region or subdomain of the problem space. A gating network is used to dynamically combine the outputs of these experts, ensuring that the contributions of the models are weighted appropriately based on the input data.

A. Overview of the Mixture of Experts Framework

The MoE framework is built on the principle of dividing and conquering, enabling the system to break down complex problems into smaller, manageable subdomains. The key components of the framework are as follows:

- **Experts:** These are specialized models (e.g., neural networks) trained to focus on specific subdomains of the problem. Each expert learns to approximate the behavior of the system within its assigned region, thereby reducing the overall complexity of the learning task.
- **Gating Network:** The gating network determines the relevance of each expert for a given input. It assigns weights to the outputs of the experts based on their applicability to the input domain, ensuring smooth transitions between subdomains.
- **Aggregation Mechanism:** The weighted outputs of the experts are combined to produce the final solution, allowing the framework to capture the global behavior of the system by leveraging the strengths of individual experts.

B. Motivation for Using MoE in Solving Differential Equations

The MoE framework is particularly well-suited for solving differential equations due to the following advantages:

- **Nonlinearity and Complexity:** Many differential equations exhibit highly nonlinear and oscillatory behavior that is difficult for a single model to approximate accurately. By dividing the solution space into subdomains, the MoE approach allows each expert to specialize in learning specific patterns or behaviors.
- **Scalability:** As the problem complexity increases, training a single model becomes computationally expensive. The modular nature of MoE ensures that the framework remains scalable by training smaller, focused experts.
- **Improved Generalization:** The gating network ensures that only the most relevant experts contribute to the solution, reducing the risk of overfitting and enhancing the model's ability to generalize across different domains.

VIII. Physics-Informed Neural Networks (PINNs)

Physics-Informed Neural Networks (PINNs) leverage neural networks to solve partial differential equations (PDEs) by incorporating physical laws into the loss function. Unlike traditional neural networks, which rely solely on labeled data, PINNs use governing equations as constraints, ensuring that the solutions respect both data and physics.

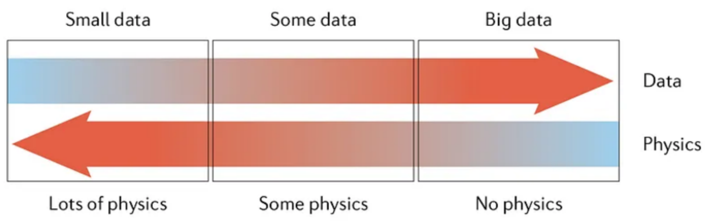

Fig. 8: Physics vs. Data

ANN Lagaris, relies entirely on the physics-based loss. When data points are incorporated into the network, it transitions into a Physics-Informed Neural Network (PINN), where the solution is guided not only by the physical laws but also by the available data, enhancing its accuracy and relevance to real-world scenarios.

A. Mathematical Formulation

A PINN approximates the solution $u(x, t; \theta)$ to a PDE:

$$\mathcal{N}[u(x,t)] = 0$$

where $\mathcal{N}$ is the differential operator. The network is trained to minimize a composite loss that incorporates the physical constraints and any initial or boundary conditions.

B. Loss Function

The total loss $\mathcal{L}$ is composed of three terms:

1) **PDE Residual Loss ($\mathcal{L}_{PDE}$)**:
$$\mathcal{L}_{PDE} = \frac{1}{N_r}\sum_{i=1}^{N_r}\left|\mathcal{N}[u(x_r^i, t_r^i; \theta)]\right|^2.$$
This ensures the solution satisfies the PDE over the domain.
2) **Initial Condition Loss ($\mathcal{L}_{IC}$)**:
$$\mathcal{L}_{IC} = \frac{1}{N_{IC}}\sum_{i=1}^{N_{IC}}\left|u(x_{IC}^i, t_{IC}^i; \theta) - u_{IC}(x_{IC}^i, t_{IC}^i)\right|^2.$$
This ensures the solution respects the initial conditions.
3) **Boundary Condition Loss ($\mathcal{L}_{BC}$)**:
$$\mathcal{L}_{BC} = \frac{1}{N_{BC}}\sum_{i=1}^{N_{BC}}\left|u(x_{BC}^i, t_{BC}^i; \theta) - u_{BC}(x_{BC}^i, t_{BC}^i)\right|^2.$$
This ensures the solution satisfies the boundary conditions.

The total loss is a weighted sum:

$$\mathcal{L} = \lambda_{PDE}\mathcal{L}_{PDE} + \lambda_{IC}\mathcal{L}_{IC} + \lambda_{BC}\mathcal{L}_{BC}.$$

where λ_{PDE}, λ_{IC}, and λ_{BC} control the contribution of each term.

IX. Hyper-Parameters

A. Activation Functions

Activation functions play a critical role in neural networks by introducing nonlinearity, which enables the network to approximate complex functions, including solutions to differential equations. Different activation functions were tested to assess their impact on the network's ability to approximate solutions to differential equations (DEs).

- **Tanh Activation Function**:
  - **Definition**: The tanh (hyperbolic tangent) function is defined as:
  $$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$
  - **Properties**: The tanh function maps input values to the range $(-1, 1)$, introducing smooth gradients that help with quick convergence. However, it suffers from the ”flattening out” effect where gradients become very small for large absolute values of x, potentially leading to vanishing gradients, especially in deep networks.
  - **Observations**: In our experiments, tanh led to faster convergence but was less effective for complex DEs, as it struggled to maintain solution fidelity in the presence of steep gradients.
- **Softplus Activation Function**:
  - **Definition**: The Softplus function is defined as:
  $$\text{Softplus}(x) = \ln(1 + e^x).$$
  - **Properties**: Softplus is a smooth approximation of the ReLU function, mapping input values to the range $(0, \infty)$. Unlike ReLU, Softplus has non-zero gradients for all input values, which prevents abrupt gradient changes and promotes stable convergence.
  - **Observations**: Softplus preserved the differential equation's features better, maintaining gradient stability and enabling the network to converge more effectively on nonlinear DEs.

Summary: Activation functions such as Softplus proved advantageous in maintaining solution fidelity for nonlinear differential equations, where tanh was more susceptible to gradient vanishing issues.

B. Learning Rate Sensitivity

The learning rate (LR) significantly impacts the stability and convergence of neural network training. During our experiments, we observed the following sensitivities:

- **Lagaris Method**: This method required careful tuning of the learning rate, as too high an LR caused instability. For experiment i have used learning rate scheduler to mititgate the effect of unstability due to learning rate.
- **PINNs**: PINN also require careful adjustment to ensure that the network focused on learning the physics of the problem while also satisfying initial and boundary conditions.

X. Experiments and Results

In this section, we present the results of our experiments.

A. Van der Pol-Duffing Oscillator

The differential equation under consideration is:

$$\frac{d^2x}{dt^2} + 0.3^2\left(x + 0.2x^3\right) = 0.2\sin 2t,$$

with initial conditions:

$$x(0) = 0.15, \quad x'(0) = 0.$$

Analytical Solution: We used the Runge-Kutta Method to compute the exact solution.

Trial Solution:

$$x_{\mathrm{He}}(t,p) = 0.15 + t^2 N(t,p)$$

Results: For the trial solution, we experimented with various configurations, including the number of experts, layers, neurons per layer, and activation functions. The best configuration is as follows:

- **Number of Experts:** 2
- **Number of Layers:** 3
- **Neurons per Layer:** 32
- **Activation Function:** SoftPlus
- **Method:** ANN Lagaris

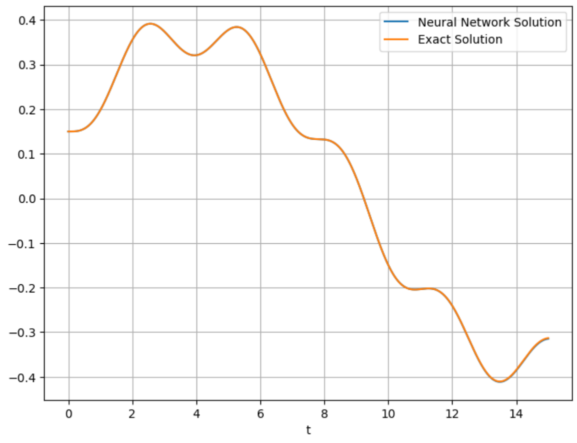

Fig. 9: Comparison of Approximation vs. Exact Solution

B. Harmonic Oscillator

The differential equation for the harmonic oscillator is:

$$\frac{d^2x}{dt^2} + \frac{\mu}{m}\frac{dx}{dt} + \omega_0^2 x = 0,$$

with parameters:

$$\frac{\mu}{m} = 2, \quad \omega_0 = 10, \quad x = [0,1]$$

and initial conditions:

$$x(0) = 1, \quad \frac{dx}{dt} = 0.$$

Analytical Solution:

$$x(t) = e^{-\delta t}(2A\cos(\phi + \omega t)), \quad \omega = \sqrt{\omega_0^2 - \delta^2}$$

Trial Solution:

$$x_{\mathrm{t}}(t,p) = 1 + t^2 N(t,p)$$

Results: For the trial solution, we experimented with various configurations, including the number of experts, layers, neurons per layer, and activation functions. The best configuration is as follows:

- **Number of Experts:** 2
- **Number of Layers:** 3
- **Neurons per Layer:** 32
- **Activation Function:** SoftPlus
- **Method:** ANN Lagaris

Upon experimenting with different numbers of points, we found that using a softmax activation function allowed for faster convergence, even with fewer points.

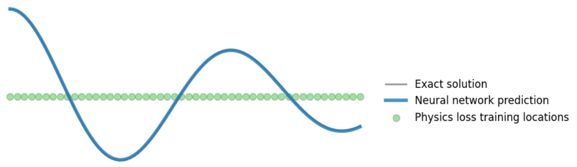

Fig. 10: Comparison of Approximation vs. Exact Solution

C. Alternative Harmonic Oscillator Configuration

For a different configuration of the harmonic oscillator:

$$\frac{\mu}{m} = 1, \quad \omega_0 = 29, \quad x = [0,2],$$

with initial conditions:

$$x(0) = 1, \quad \frac{dx}{dt} = 0.$$

Analytical Solution:

$$x(t) = e^{-\delta t}(2A\cos(\phi + \omega t)), \quad \omega = \sqrt{\omega_0^2 - \delta^2}$$

Trial Solution:

$$x_{\mathrm{t}}(t,p) = N(t,p)$$

Results: The best configuration is:

- **Number of Experts:** 2
- **Number of Layers:** 3
- **Neurons per Layer:** 32
- **Activation Function:** SoftPlus
- **Method:** PINN

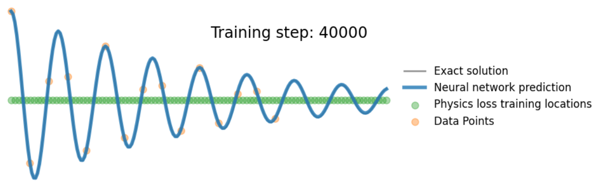

Fig. 11: Comparison of Approximation vs. Exact Solution

XI. Discussion

A. Learning Rate Adjustments

Initially, the **ANN Lagaris method** yielded satisfactory results, but as equation complexity increased, the ANN architecture struggled to effectively solve the equations. To address this, we transitioned to a **Mixture of Experts (MOE)** approach.

After multiple experiments, we achieved satisfactory results, but training remained unstable due to a high learning rate. Reducing the learning rate improved convergence but required more epochs. To optimize, we implemented a **Cosine Annealing Learning Rate Scheduler**, which gradually reduces the learning rate over time.

B. PINN and Physics Loss Weight

When using the **Physics-Informed Neural Network (PINN)** method, careful adjustment of the **physics loss weight** is essential. A value around **5e-5** was found to be optimal. The PINN method is highly sensitive to this weight, making careful tuning essential.

C. Mixture of Experts (MOE)

The **Mixture of Experts (MOE)** approach shows promise for solving complex differential equations in conjunction with PINN. However, challenges with the **Gating Network** remain, as ineffective gating can undermine the MOE approach. Further research into robust training methods for the gating network is recommended.

XII. Conclusion

This project demonstrated the effectiveness of neural networks for approximating solutions to differential equations, particularly with PINNs and MOE. Key findings include stable training with SoftPlus functions, advantages over tanh, and insights into optimizing sensitive parameters like learning rate. These results encourage further exploration of neural network methods for scientific computing.

Acknowledgment

This work was conducted under the guidance of Dr. Santwana Mukhopadhyay at the Department of Mathematical Sciences, Indian Institute of Technology (BHU), Varanasi. The author acknowledges the support and resources provided by the institute, which were essential for the successful completion of this project.

References

[1] I. E. Lagaris, A. Likas, and D. I. Fotiadis, ”Artificial Neural Networks for Solving Ordinary and Partial Differential Equations,”https://arxiv.org/pdf/physics/9705023

[2] S. Mall and S. Chakraverty, ”Hermite Polynomial-Based Functional Link Artificial Neural Network for Solving Differential Equations,” *Applied Soft Computing*, vol. 43, pp. 347–361, 2016. https://ieeexplore.ieee.org/document/7524041

[3] M. Raissi, P. Perdikaris, and G. E. Karniadakis, ”Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Partial Differential Equations,” https://arxiv.org/abs/1711.10561

[4] M. Raissi, P. Perdikaris, and G. E. Karniadakis, ”Hidden Physics Models: Machine Learning of Nonlinear Partial Differential Equations,”. https://arxiv.org/abs/1711.10566

[5] Nithin Chalapathi, Yiheng Du, Aditi Krishnapriyan”Scaling physics-informed hard constraints with mixture-of-experts”. https://arxiv.org/pdf/2402.13412