

# Analysis on Differential Equaiton Solver using Neural Network

Rasesh Shetty  
23124040

# **Solving Differential Equations with Neural Networks**

## **An Exploratory Approach using Neural Network Methods**

Presented By: Rasesh Udayakumar Shetty

Date: 30 October 2024

Objective: A summary of techniques, experiments, and findings from solving differential equations with neural networks, including insights on activation functions, optimizers, and parameter tuning.

# Motivation and Objectives

- Why Neural Networks for Differential Equations?
  - Neural networks, especially Physics-Informed Neural Networks (PINNs), offer an alternative, flexible approach for approximating solutions.
- Project Goals:
  - Explore and implement different neural network approaches (Lagarias method, Runge-Kutta, and PINNs) for solving differential equations.
  - Experiment with various activation functions, optimizers, and hyperparameters to improve solution accuracy and convergence speed.

# Methodology - Overview of Approaches

- Key Techniques Explored:
  - Lagaris Approach: First step in using neural networks to solve ordinary differential equations (ODEs).
  - Runge-Kutta Method: Implemented as a benchmark for comparison with neural network solutions.
  - Physics-Informed Neural Networks (PINNs): Applied to solve complex PDEs, including the 1D Heat Equation, 1D Burgers Equation, and Harmonic Oscillator Equation.
- Why PINNs?
  - PINNs integrate physical laws directly into the loss function, allowing the network to learn both the solution and physics constraints, improving accuracy and stability in solutions.

# Methodology - Overview of Approaches

- Methods Implemented:
  - Lagaris Method: Applied to ordinary differential equations (ODEs) as an entry point for neural network-based solutions.
  - Runge-Kutta (RK) Method: Used as a conventional numerical method for comparison, known for stability but computationally heavy for complex systems.
  - Physics-Informed Neural Networks (PINNs): Leveraged for more complex partial differential equations (PDEs), including:
    - 1D Heat Equation: Demonstrating PINNs' ability to handle time-dependent equations.
    - 1D Burgers Equation: Used to capture nonlinear dynamics, showcasing PINNs' flexibility.
    - Harmonic Oscillator Equation: Highlighting PINNs' capacity to incorporate boundary conditions directly into the model.

# Key Experiments with Activation Functions

- Activation Functions Tested:
  - Tanh: Offers fast convergence but struggles with complex differential equations, as it tends to flatten out, leading to potential loss of detail in the solution.
  - Softplus & Gaussian: These functions help preserve the critical features of differential equations, handling non-linearities well without flattening out, which improves solution accuracy and stability.
  - Some other honorable mentions: SiLU, Swish (Both are basically  $\text{Relu} * \text{Sigmoid}$  with some weight difference)
- Key Insight:
  - While tanh is effective for quick convergence, Softplus and Gaussian are better suited for complex differential equations, maintaining feature fidelity and providing more consistent results.

# Optimization - Adam and LBFGS

- Optimiser Choices:
  - We Experimented on various types of Optimisers:
    - Adagrad
    - RMSprop
    - Adam
    - AdaMax(Adam optimizer with EMA to infinite power)
    - NAdam( Nesterov Accelerated Gradient + Adam)
  - We finally concluded the following combination to be working the best, i.e Adams everywhere, and L-BFGS sparingly wherever complex equations are involved to improve the results.
  - Adam: Most advanced efficient Optimization algorithm. Used for initial training due to its efficiency in handling large datasets and adaptability in early epochs.
  - L-BFGS: Quasi newton BFGS optimisation (first mentioned in Lagaris paper). Applied in later stages for fine-tuning; this optimizer provides better convergence for differential equations but is computationally heavy, so it's only used sparingly.

# Optimization - Adam and LBFGS

- Learning Rate Sensitivity:
  - Lagaris Approach: Found to be highly sensitive to the learning rate, requiring careful adjustment to prevent unstable training.
  - PINNs: The learning rate stabilizes, but the need to tune the physics-based weight introduces additional sensitivity in this approach.

## Loss Functions

- We tried various types of losses, but the best ones that seem to work are **MSE** and **MAE** , so we stopped search for better loss functions since they satisfied our need adequately



# Universal Approximation Theorem - Practical Insights

- After the failure with Hermite Neural network implementation, we experimented with various factors like number of neurons per layer, number of layers, number of epochs and finally concluded that neural networks can approximate any function, regardless of complexity, if provided with enough neurons and sufficient training epochs.
- Evidence from Experiments:
  - Through various tests, it was confirmed that increasing neuron count and training duration allows the model to approximate even highly complex functions with high accuracy.
- Conclusion:
  - This supports the Universal Approximation Theorem, demonstrating neural networks' potential to handle intricate solutions in differential equations when appropriately scaled.

# Results

## Lagaris Implementation

Core idea revolved around making an equation with two terms in it (one for constraint other for neural network), and loss function, then trying to minimize the same.

$$\Psi_t(\vec{x}) = A(\vec{x}) + F(\vec{x}, N(\vec{x}, \vec{p}))$$

$$E[\vec{p}] = \sum_i \left\{ \frac{d\Psi_t(x_i)}{dx} - f(x_i, \Psi_t(x_i)) \right\}^2$$

We implemented all the examples given in the paper and got considerably good results, thanks to clarity of the Research paper

# Results

## Hermite Polynomial

- This paper was pretty straightforward at first, but on implementation it did not work as intended, we tried various methods to tweak, but then came to conclusion that the number of epochs on which the paper had trained on, was not something we could have done on our systems.
- This was a failure for us in terms of implementation, but in turn I learnt how Runge-Kutta-Method works and used its implementation in later experiments.
- So I would like to assume this paper as a helping stone in my path of this experiment

# Physics-Informed Neural Networks (PINNs)

- What are PINNs?: PINNs embed physical laws (in the form of differential equations) directly into the neural network's loss function, ensuring the model respects these laws during training.
- How PINNs Work: Instead of using only data points, PINNs also include the residuals of differential equations in the loss function. This approach allows the neural network to approximate solutions that are consistent with both the data and the governing equations.
- Advantages:
  - Solves differential equations without requiring labeled data.
  - Can incorporate boundary conditions, initial conditions, and complex physical constraints directly in the learning process.

# Physics-Informed Neural Networks (PINNs)

## Customized Loss Function

Loss includes terms for both the physical equation's residuals and boundary/initial conditions.

Mathematically express the loss function for an example equation, such

$$\mathcal{L} = \mathcal{L}_{PDE}^{\text{as}} + \mathcal{L}_{BC}$$

where:

$\mathcal{L}_{BC}$  : Loss term that penalizes deviations from the differential equation.

$\mathcal{L}_{PDE}$  : Loss term enforcing boundary conditions.

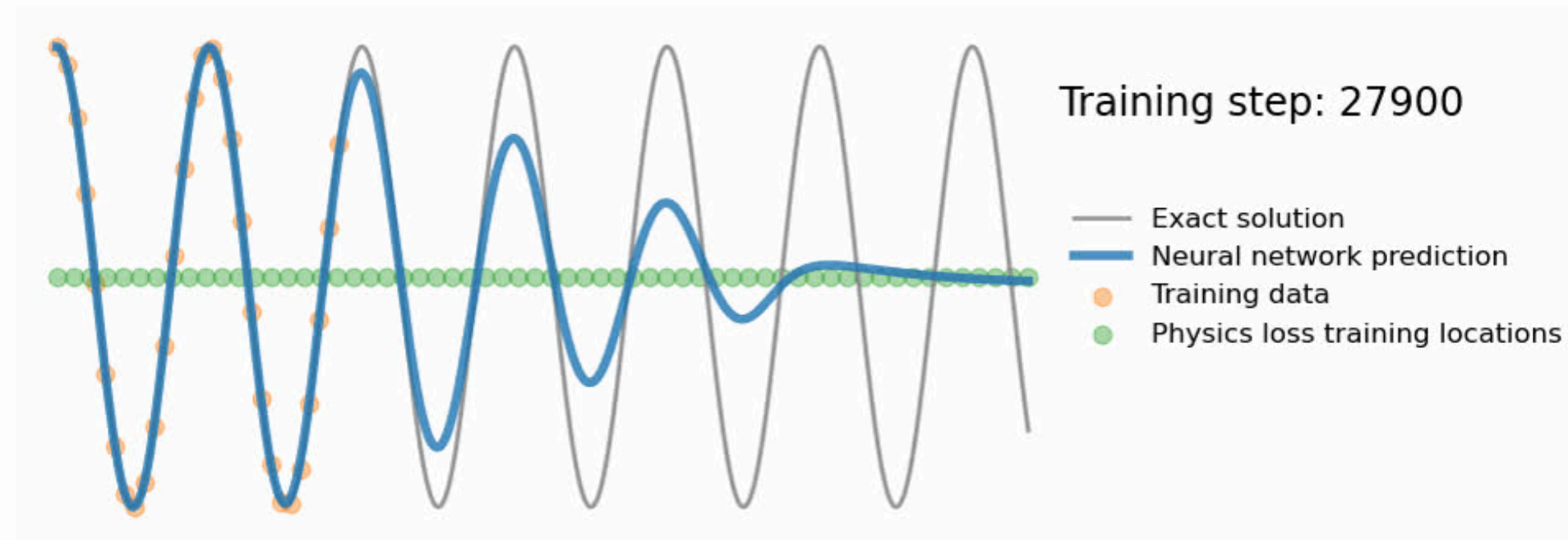
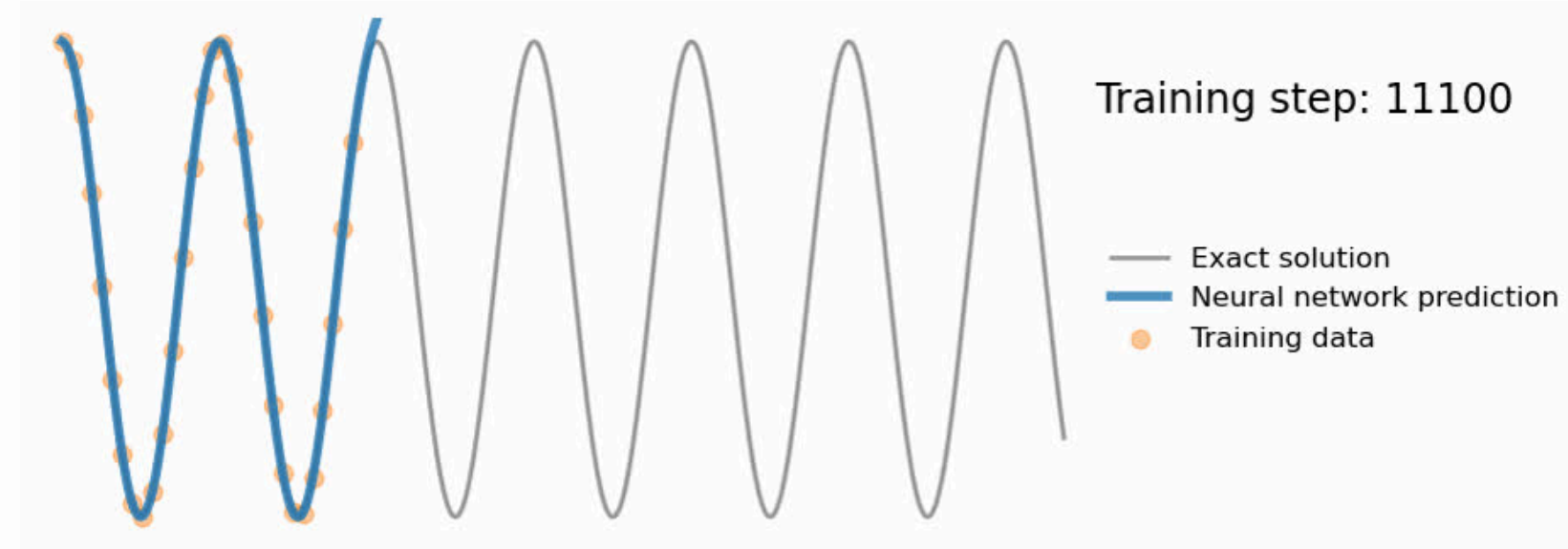
# Physics-Informed Neural Networks (PINNs)

## Some Equations on which i applied PINN

- 1-D heat equation
- Burger's Equation
- Oscillation Equation

# Physics-Informed Neural Networks (PINNs)

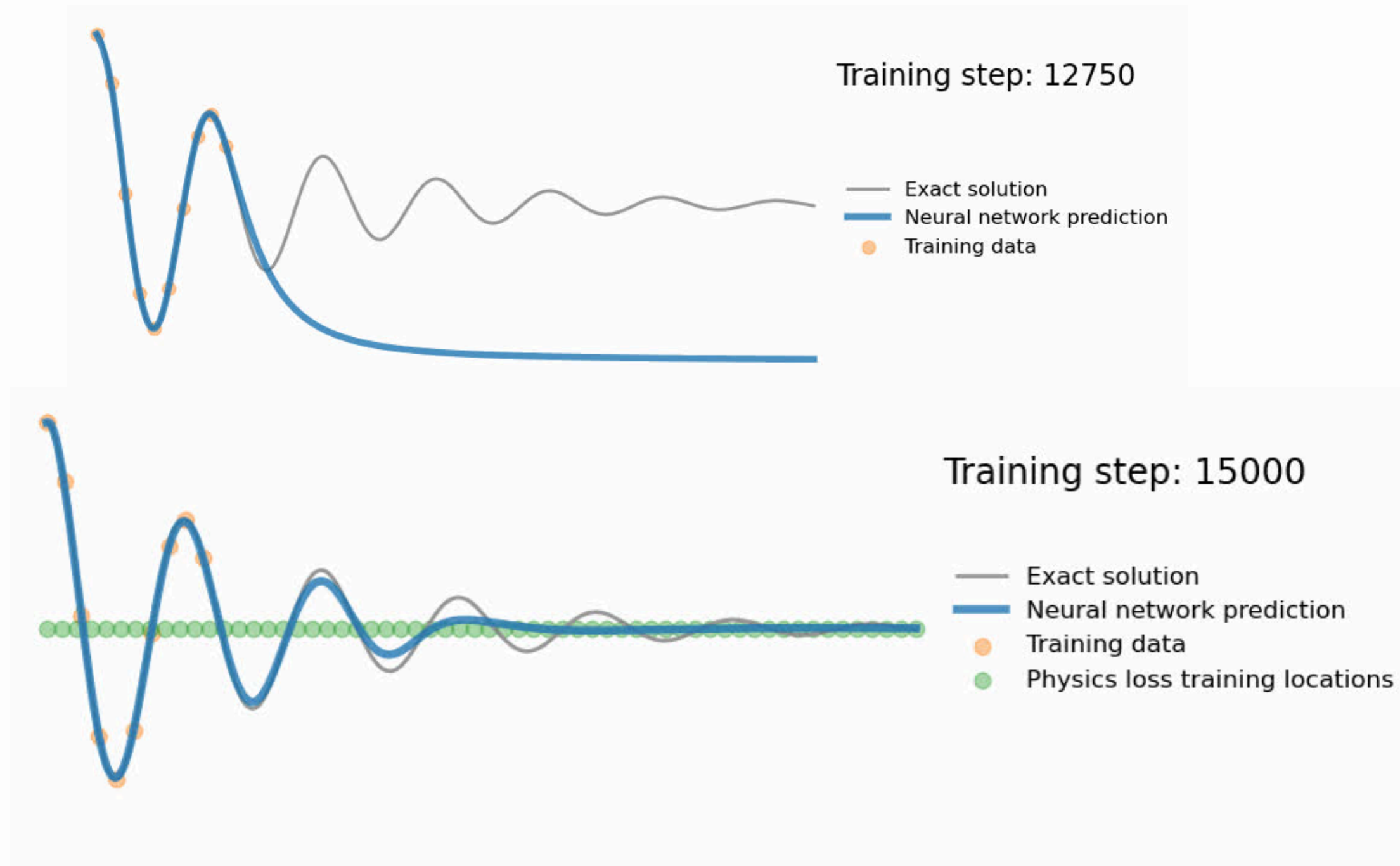
## Results on Oscillation Equation





# Physics-Informed Neural Networks (PINNs)

## Results on Oscillation Equation






# Key Learnings and Reflections

## Technical Insights

- Architecture Selection Matters: Choice of network architecture significantly impacts performance, especially for complex equations.
- Importance of Loss Function Design: For PINNs, integrating boundary conditions and equation residuals in the loss function was critical to model success.
- Runge-Kutta Learning: Insights gained on numerical methods and their importance in complex neural network training; this helped inform understanding of stability and convergence in network solutions.

## Challenges Encountered

- High Computational Cost: Implementing these neural networks for differential equations, especially with PINNs, was computationally intensive.
- Convergence Sensitivity: Many models were sensitive to hyperparameters (learning rate, network depth), impacting convergence and accuracy.
- Handling Boundary Conditions: Maintaining accurate boundary conditions across models was often challenging, particularly for non-linear equations.



**THANK YOU!!**