

Exploratory Porject

Ode Solver

By :- Naitik Agrawal
23124030

Solving Differential Equations with Neural Networks

An Exploratory Approach using Neural Network Methods

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?

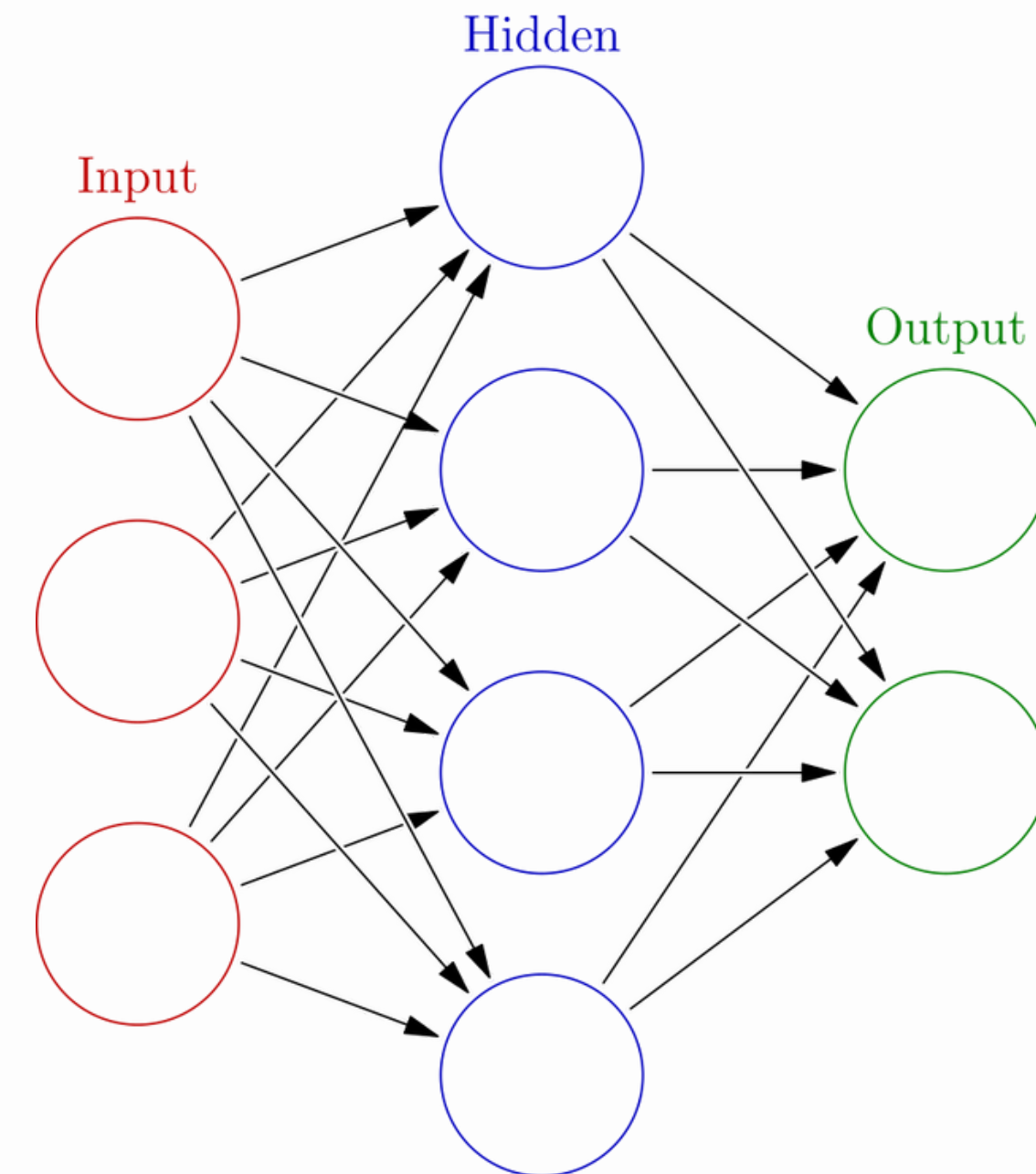
Neural networks, especially Physics-Informed Neural Networks (PINNs), offer an alternative, flexible approach for approximating solutions.

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.

Universal Approximation Theorem

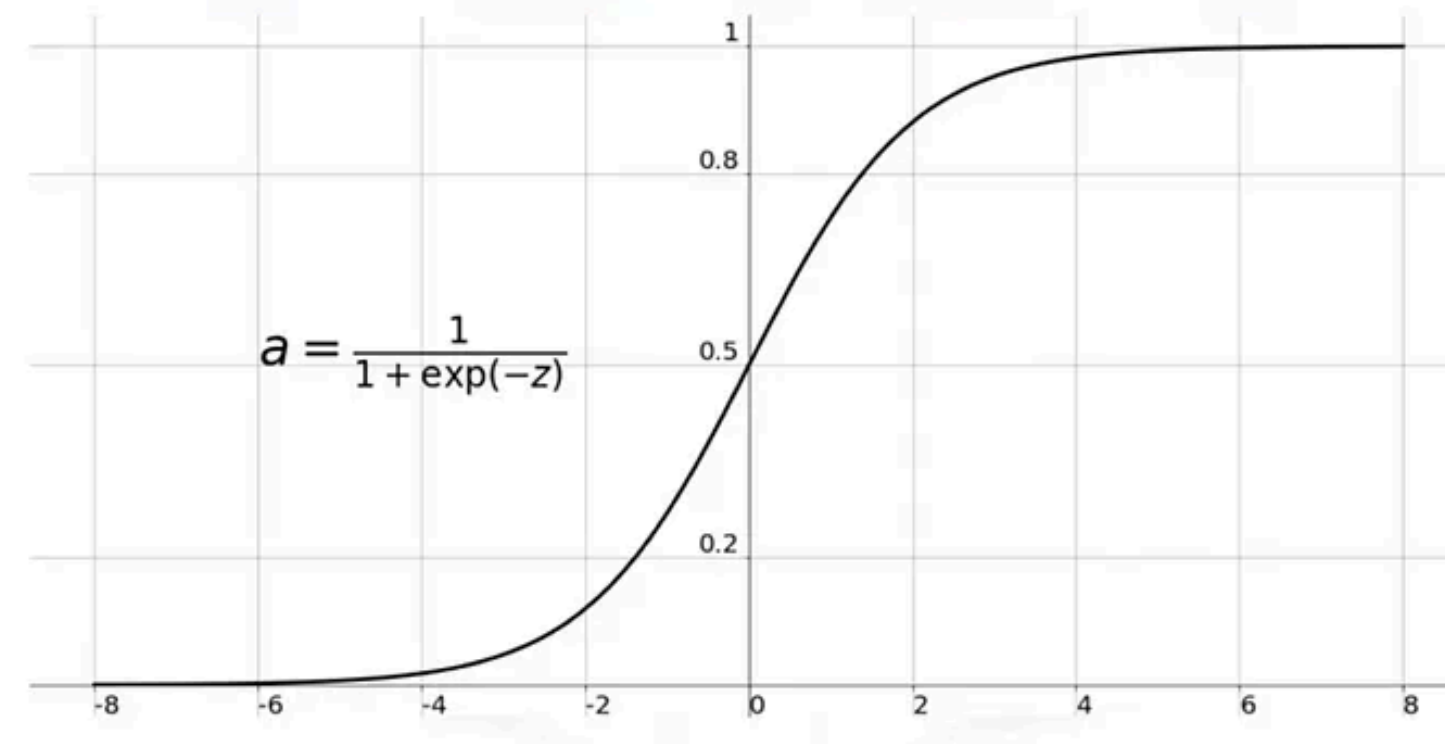
Given a family of neural networks, for each function f from a certain function space, there exists a sequence of neural networks ϕ_1, ϕ_2, \dots from the family, such that $\phi_n \rightarrow f$ according to some criterion. That is, the family of neural networks is dense in the function space.



Activation Functions

Activation functions play a crucial role in neural networks by introducing non-linearity, allowing the model to learn complex patterns in the data. Without activation functions, a neural network would behave like a linear regression model, rendering it incapable of handling intricate relationships.

Sigmoid Function



Activation Function

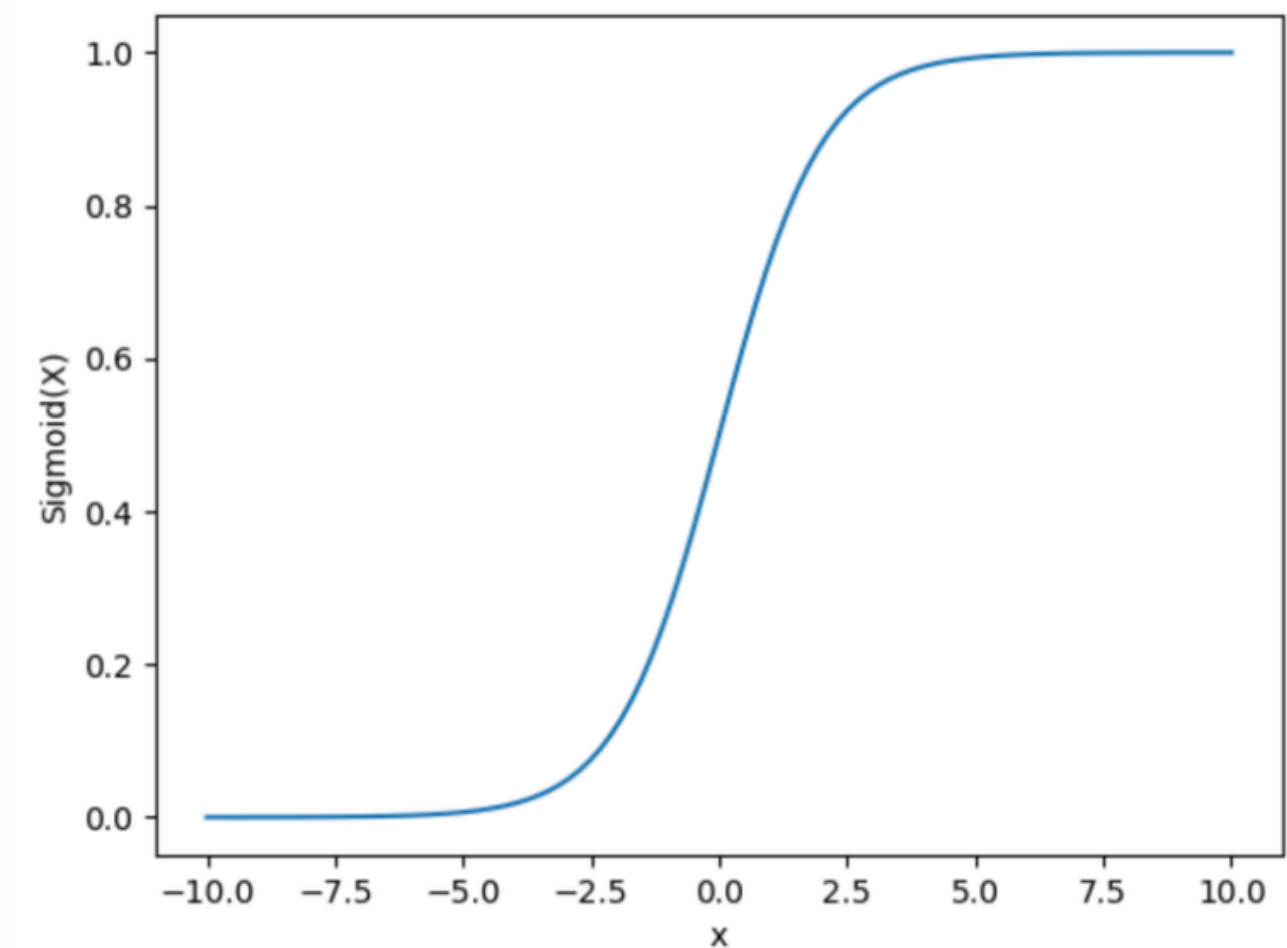
Properties we want in a ideal activation function

- Non-Linearity
- Differentiability
- Output Range
- Computational Efficiency
- Avoiding Vanishing/Exploding Gradients

Sigmoid

- Differentiable
- Non Linear
- Not Zero Centered
- Saturating Function
- Max derivative is 0.25 at 0

Saturating activation functions can cause the vanishing gradient problem, where gradients become very small during backpropagation, slowing down or even halting the learning process in deep neural network

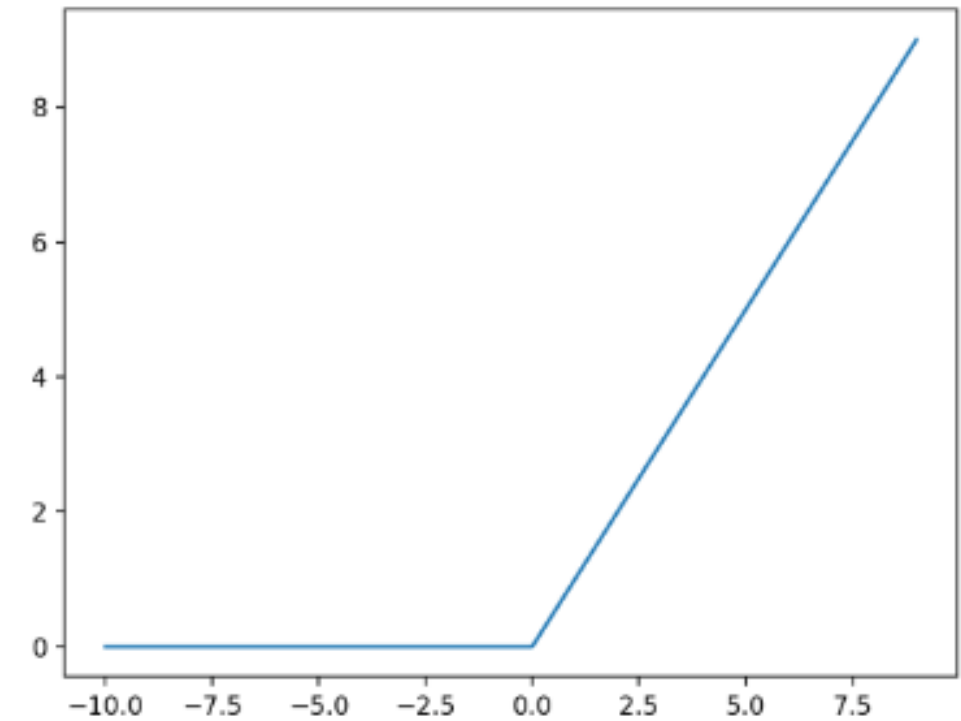


Relu

- Piece Wise Differentiable
- Non Linear
- Not Zero Centered
- Non Saturating Function
- Max derivative is 1

Generally Batch normalization is performed on output from relu activation function to make output zero-centered

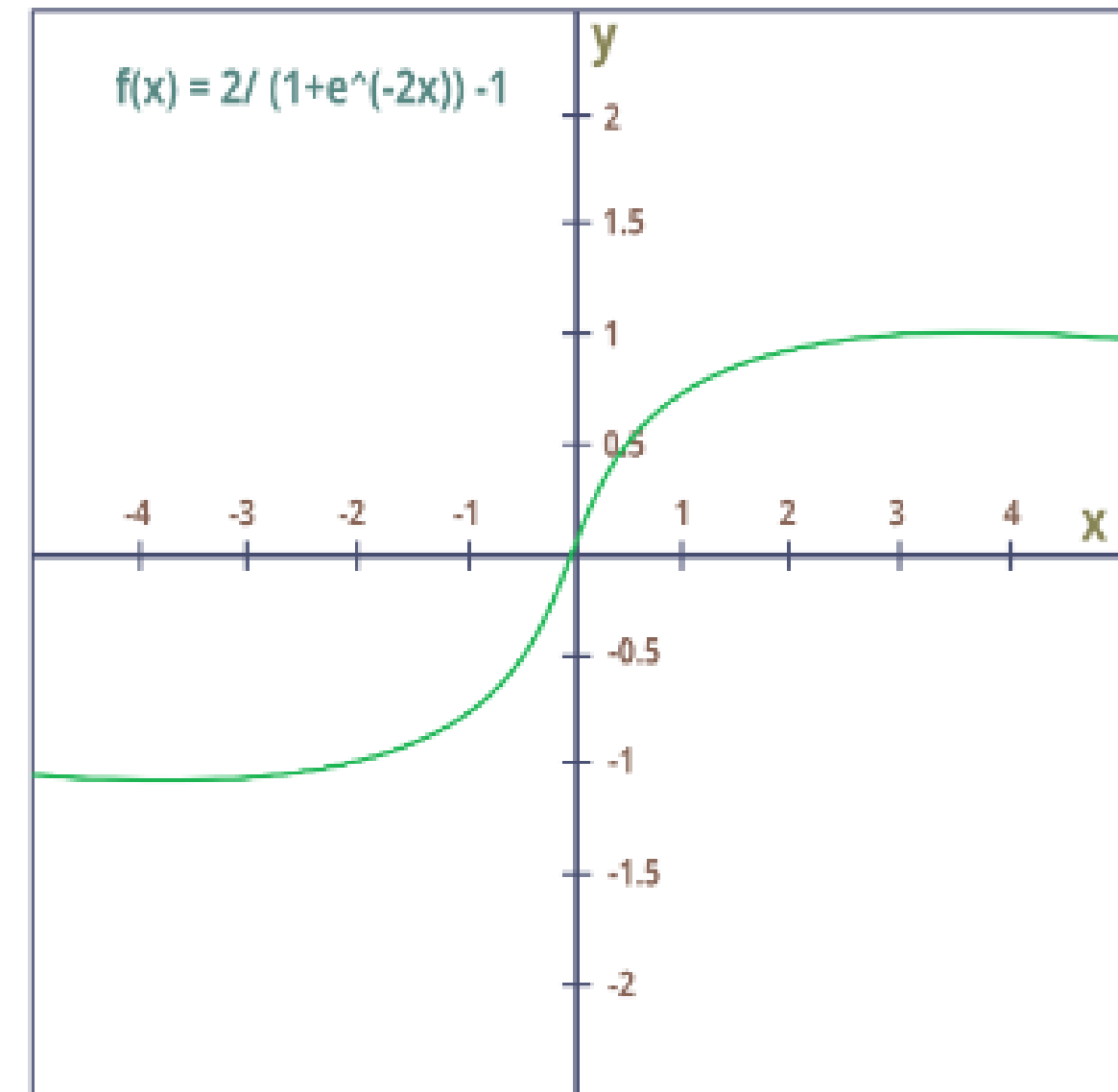
Variants of relu functions are released to solve the problem of dying relu problem and making data zero-centered



Tanh

- Differentiable
- Non Linear
- Zero Centered
- Non Saturating Function

Problem with tanh is vanishing gradient problem



Loss function

A loss function is a mathematical function used in machine learning and deep learning to measure the difference between the predicted output of a model and the actual target value. The goal during training is to minimize this loss function.

Types of loss function

- MSE :- Mean Squared Loss
- MAE :- Mean Absolute Error
- BSE :- Binary Cross Entropy Loss

Optimizer

Optimizers play a critical role in training machine learning models by updating weights to minimize the loss function

Mathematical Formula: .

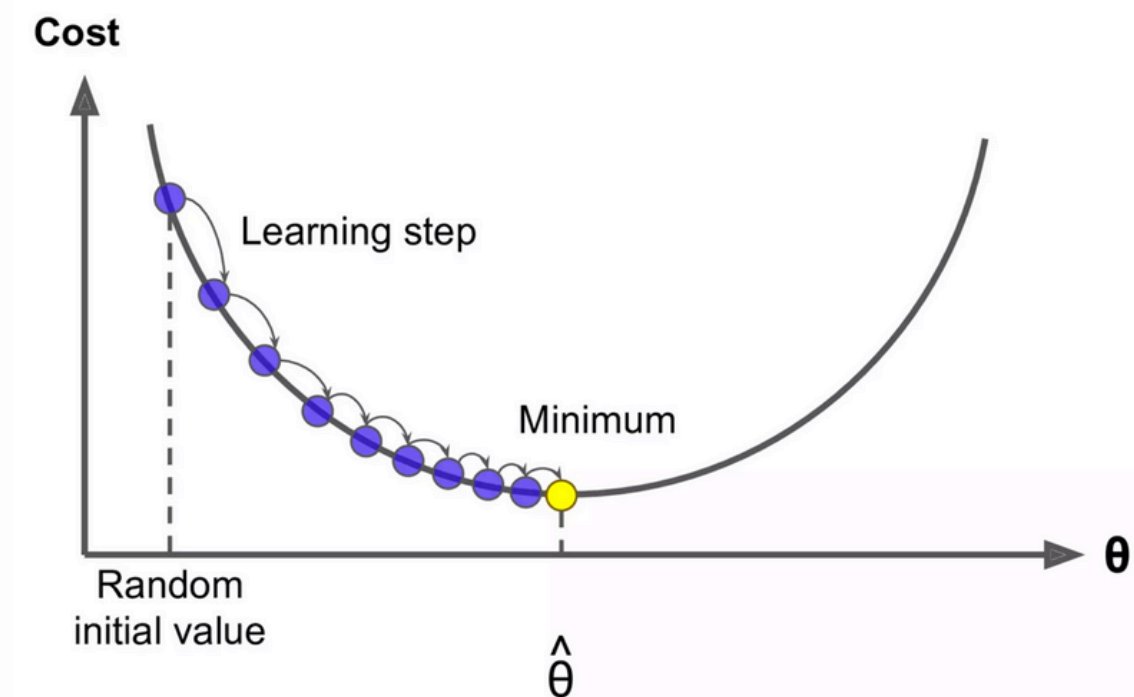
$$\theta = \theta - \alpha \nabla J(\theta)$$

where :

θ represents the weights,

α is the learning rate,

$\nabla(\theta)$ is the gradient of the loss function with respect to the weights.



Problems with High and Low Learning Rates

High Learning Rate :-

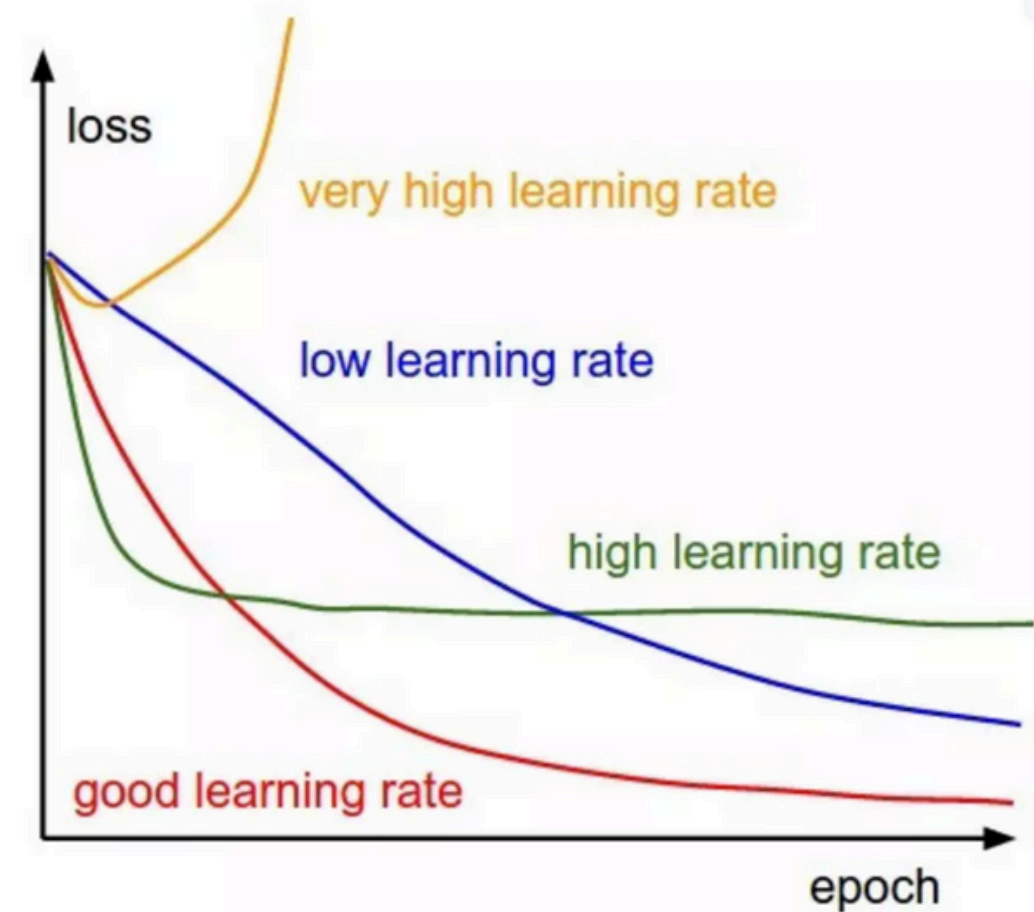
- Can cause the optimization process to overshoot the minimum, leading to divergence or oscillation around the optimal value.
- Results in instability in training and can prevent the model from converging.

Low Learning Rate:-

- Leads to slow convergence and may result in getting stuck in local minima or saddle points.
- Can cause unnecessarily long training times and increased computational cost without significant improvements in performance.

How to decide the learning rate??

As we can see from the results, a good learning rate should decay with number of epochs, to reach closer to the minima.



ODE SOLVER

An Ordinary Differential Equation (ODE) Solver is a numerical method used to find approximate solutions to ordinary differential equations, which are equations involving functions of one variable and their derivatives.

ODEs are essential in various fields, including physics, engineering, biology, and finance, as they model a wide range of phenomena, such as motion, growth, and decay.

RKM

The Runge-Kutta Method is a family of iterative methods for solving ODEs. The most common form is the fourth-order Runge-Kutta method (RK4).

Mathematical Formulation:

with an initial condition $y(t_0) = y_0$

RK4 approximates the solution by iterating:

$$\frac{dy}{dt} = f(t, y)$$

$$k_1 = hf(t_n, y_n)$$

$$k_2 = hf\left(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right)$$

$$k_3 = hf\left(t_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right)$$

$$k_4 = hf(t_n + h, y_n + k_3) \quad y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

Limitations:

Fixed step size can lead to inaccuracies for stiff equations or rapidly changing solutions.

ANN Lagaris

Traditional numerical methods for solving differential equations, such as finite difference methods and Runge-Kutta schemes, face challenges with scalability and efficiency, particularly in high-dimensional spaces. To address these limitations, Lagaris et al. propose leveraging neural networks as universal function approximators to directly approximate solutions to differential equations.

Methodology

The trial solution $u(x)$ for a differential equation

$$\mathcal{D}(u(x)) = f(x)$$

$$u(x) = A(x) + B(x) \cdot N(x, \theta)$$

Where :-

- $N(x, \theta)$ is the neural network that approximates the solution.
- $A(x)$ is a function specifically designed to satisfy the boundary conditions.
- $B(x)$ ensures that the neural network output adheres to the boundary constraints.

Training

The training process involves several steps:

- Sampling Points: A set of points $\{ x_1, x_2, \dots, x_N \}$ is sampled from the domain where the differential equation is defined.
- Forward Pass: For each sampled point x_i , the neural network $N(x_i, \theta)$ computes the trial solution $u(x_i)$. The differential operator \mathbf{D} is then applied to $u(x_i)$ to compute the residual $R(x_i)$.
- Loss Calculation: The residuals are used to compute the loss using the previously defined loss function:

$$\text{Loss} = \frac{1}{N} \sum_{i=1}^N |R(x_i)|^2$$

Training

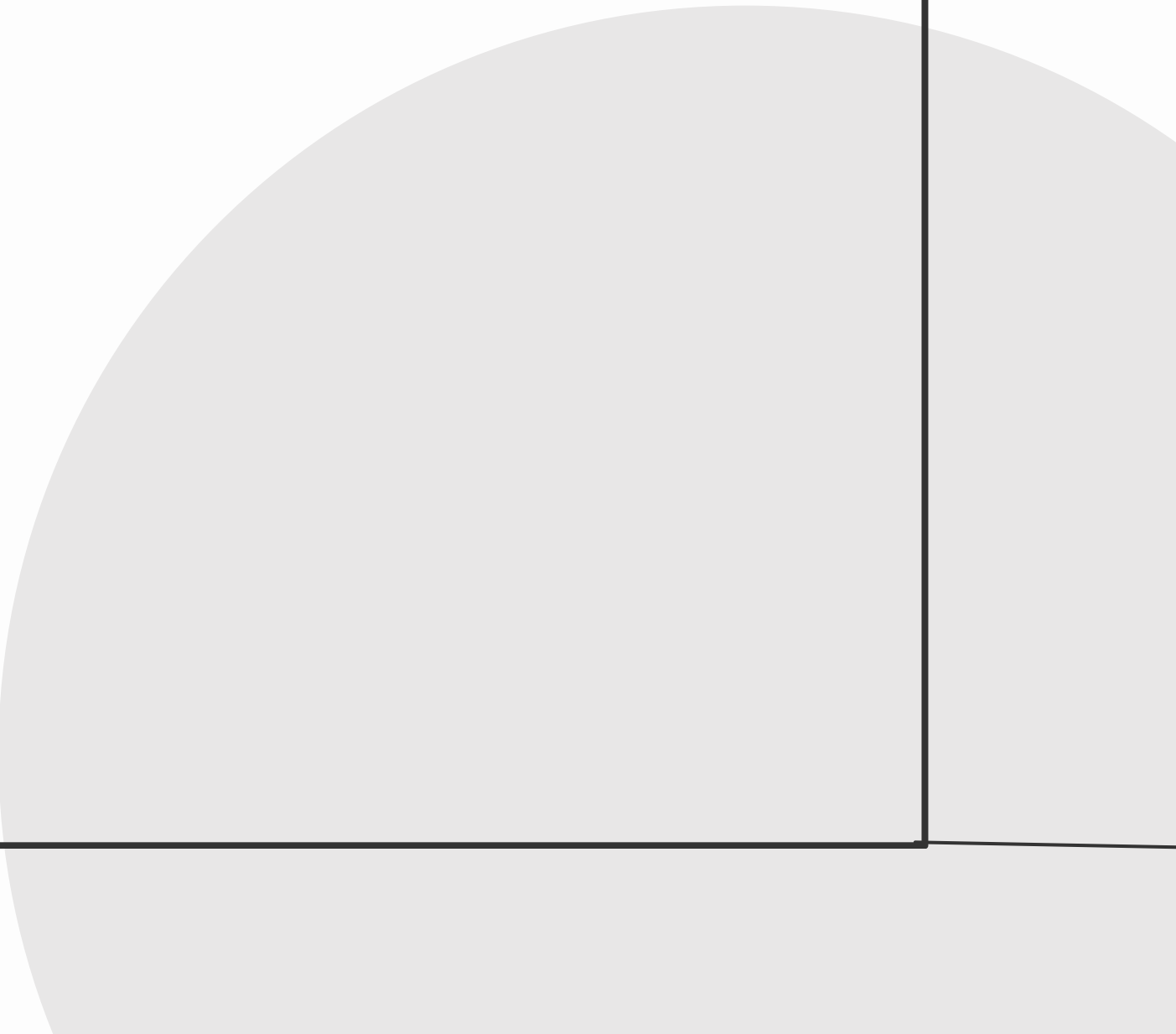
- Backpropagation: The loss is minimized using backpropagation. The gradients of the loss with respect to the neural network parameters **theta** are computed, allowing the model to update its weights accordingly.
- Iteration: Steps 1-4 are repeated for a specified number of epochs or until the loss converges to an acceptable level.

$$\theta \leftarrow \theta - \eta \nabla \text{Loss},$$

where η is the learning rate.

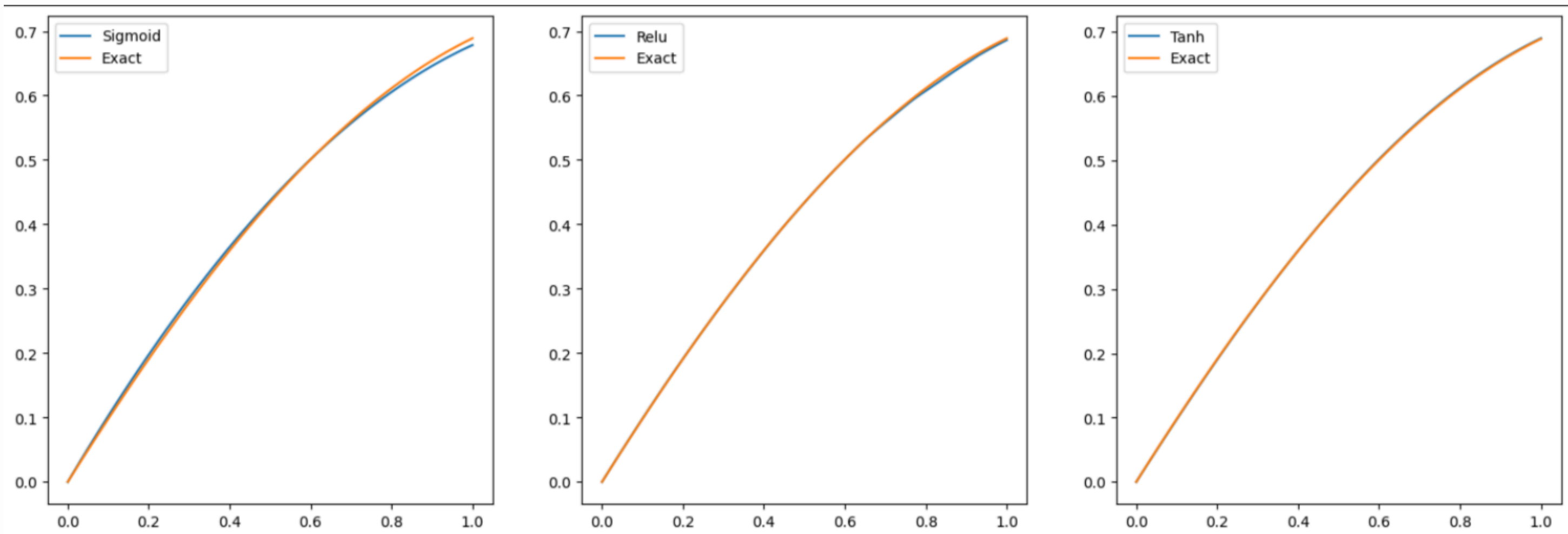
Types of Differential Equations Addressed

The framework is applicable to a variety of differential equations, including:

- Ordinary Differential Equations (ODEs)
 - Partial Differential Equations (PDEs)
- 
- A decorative light gray curved shape, resembling a large quarter-circle or a stylized wave, is positioned in the bottom right corner of the slide, partially overlapping the white content area.

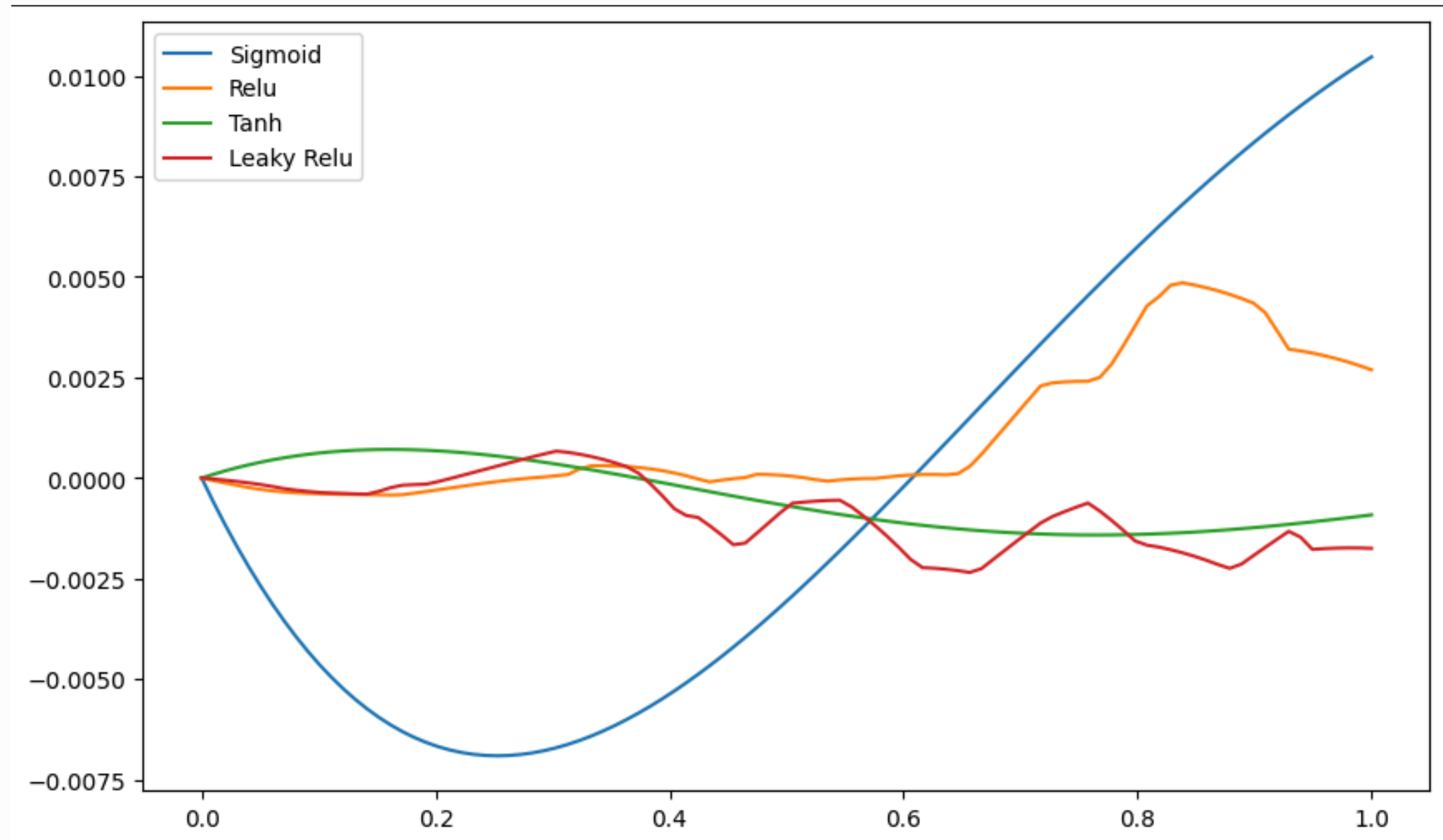
Example

$$\frac{d}{dx}\Psi + \frac{1}{5}\Psi = e^{-\frac{x}{5}} \cos(x), \quad \Psi(0) = 0 \quad \text{and} \quad x \in [0, 10]$$



Example

$$\frac{d}{dx} \Psi + \frac{1}{5} \Psi = e^{-\frac{x}{5}} \cos(x),$$
$$\Psi(0) = 0 \quad \text{and} \quad x \in [0, 10]$$



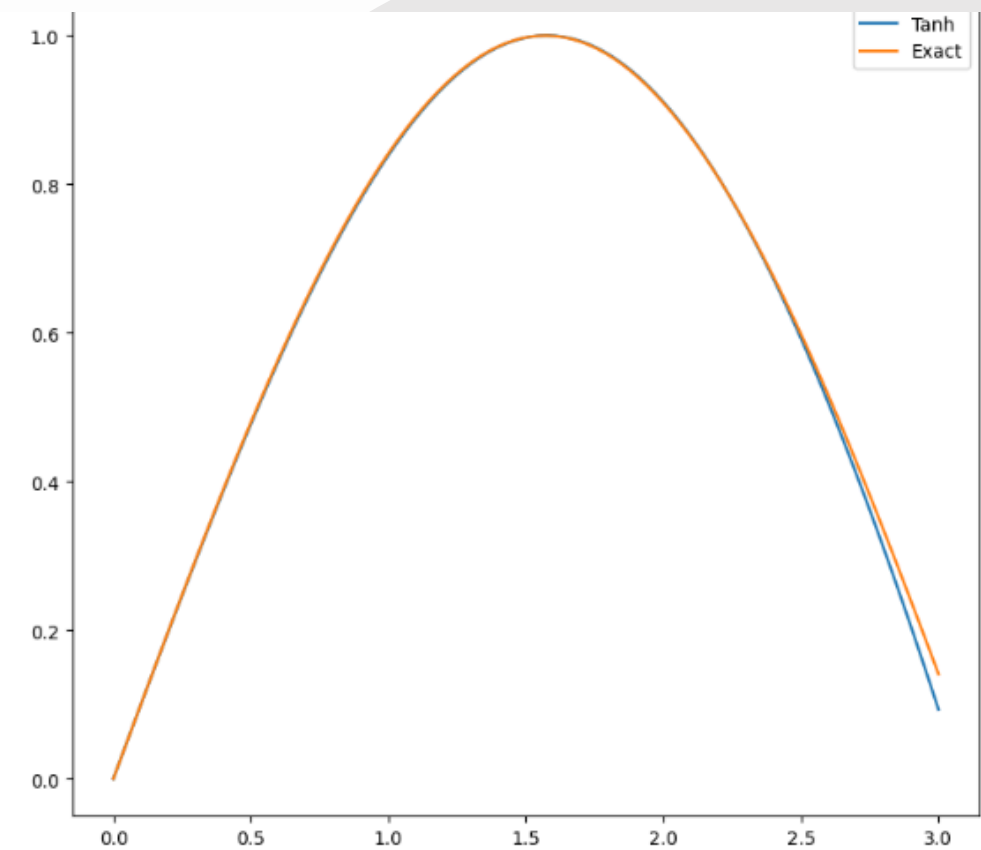
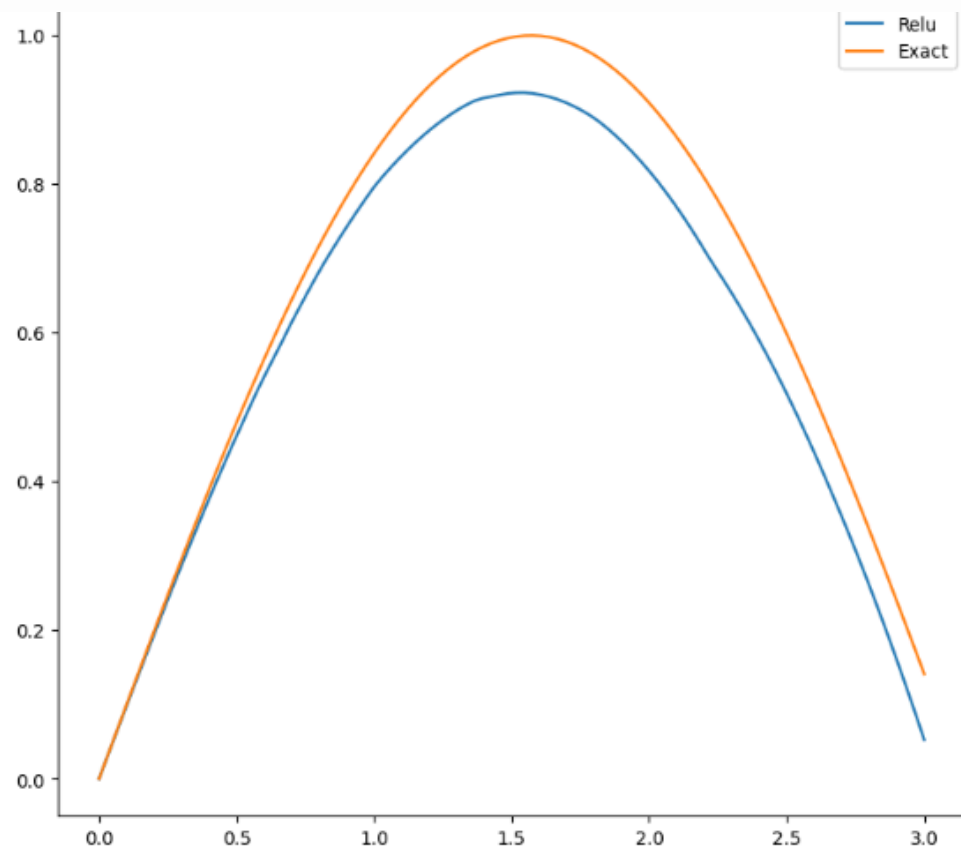
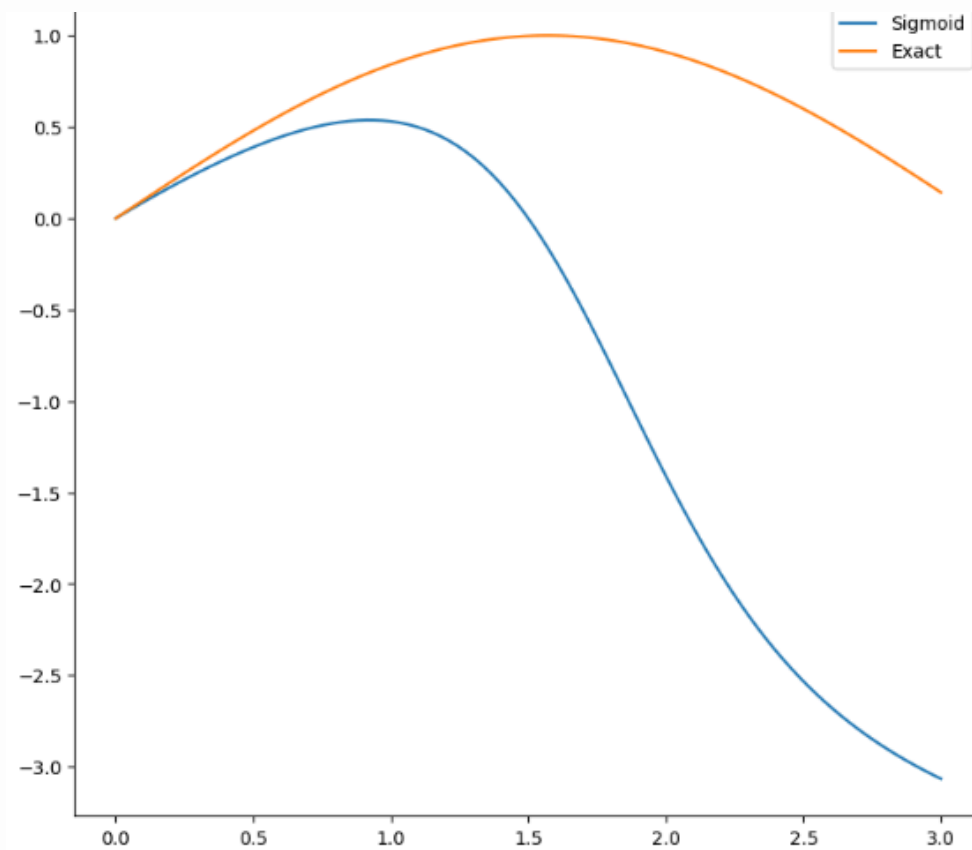
Residual

Example

$$\frac{d\Psi_1}{dx} = \cos(x) + \Psi_1^2 + \Psi_2 - (1 + x^2 + \sin^2(x)) \quad (30)$$

$$\frac{d\Psi_2}{dx} = 2x - (1 + x^2) \sin(x) + \Psi_1 \Psi_2 \quad (31)$$

with $x \in [0, 3]$ and $\Psi_1(0) = 0$ and $\Psi_2(0) = 1$. The analytic solutions are $\Psi_{a1}(x) = \sin(x)$ and $\Psi_{a2}(x) = 1 + x^2$ and are



Observation

- Tanh works better in comparison to “Relu” and “Sigmoid” for Ordinary Differential Equation Solver
- Tanh leads to faster convergence

In the Lagaris paper, the neural network model is limited to a single layer with a small number of neurons, suitable for simpler equations. To address more complex equations, I extended this approach by experimenting with deeper networks, increasing both the number of layers and the number of neurons per layer.

Limitations

The Lagaris method for solving differential equations using artificial neural networks performs effectively on simpler ordinary differential equations (ODEs) and low-order partial differential equations (PDEs).

However, when applied to more complex differential equations with the same model architecture, the network becomes highly sensitive to the learning rate.

Even slight adjustments in the learning rate can significantly impact convergence

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

Hermite Polynomial

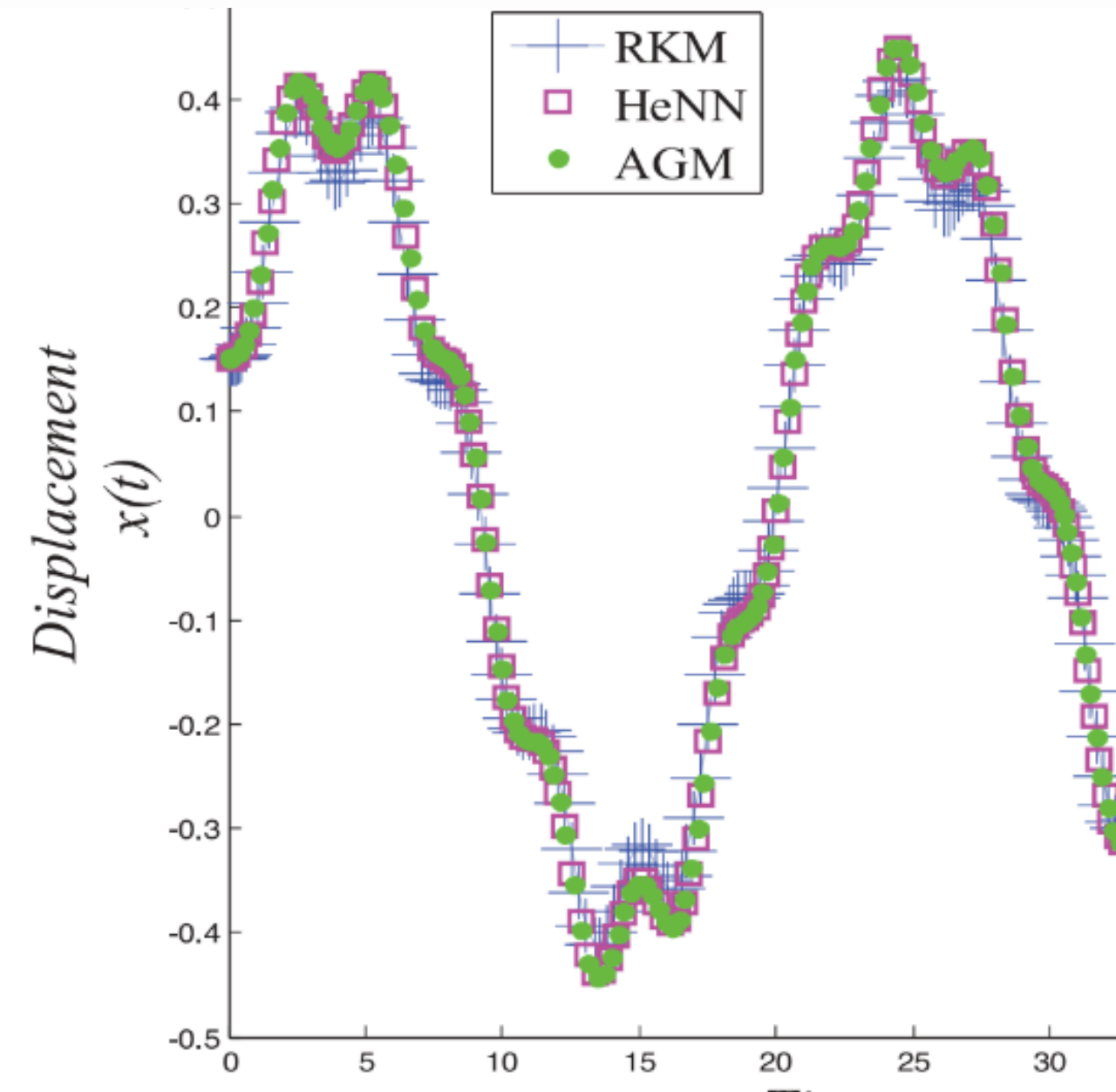
This paper was pretty straightforward at first, but on implementation it did not work as intended, we tried various methods to tweak, but then came to conclusion that the number of epochs on which the paper had trained on, was not something we could have done on our systems.

This was a failure for us in terms of implementation

Example

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

with initial conditions $x(0) = 0.15, x'(0) = 0$.

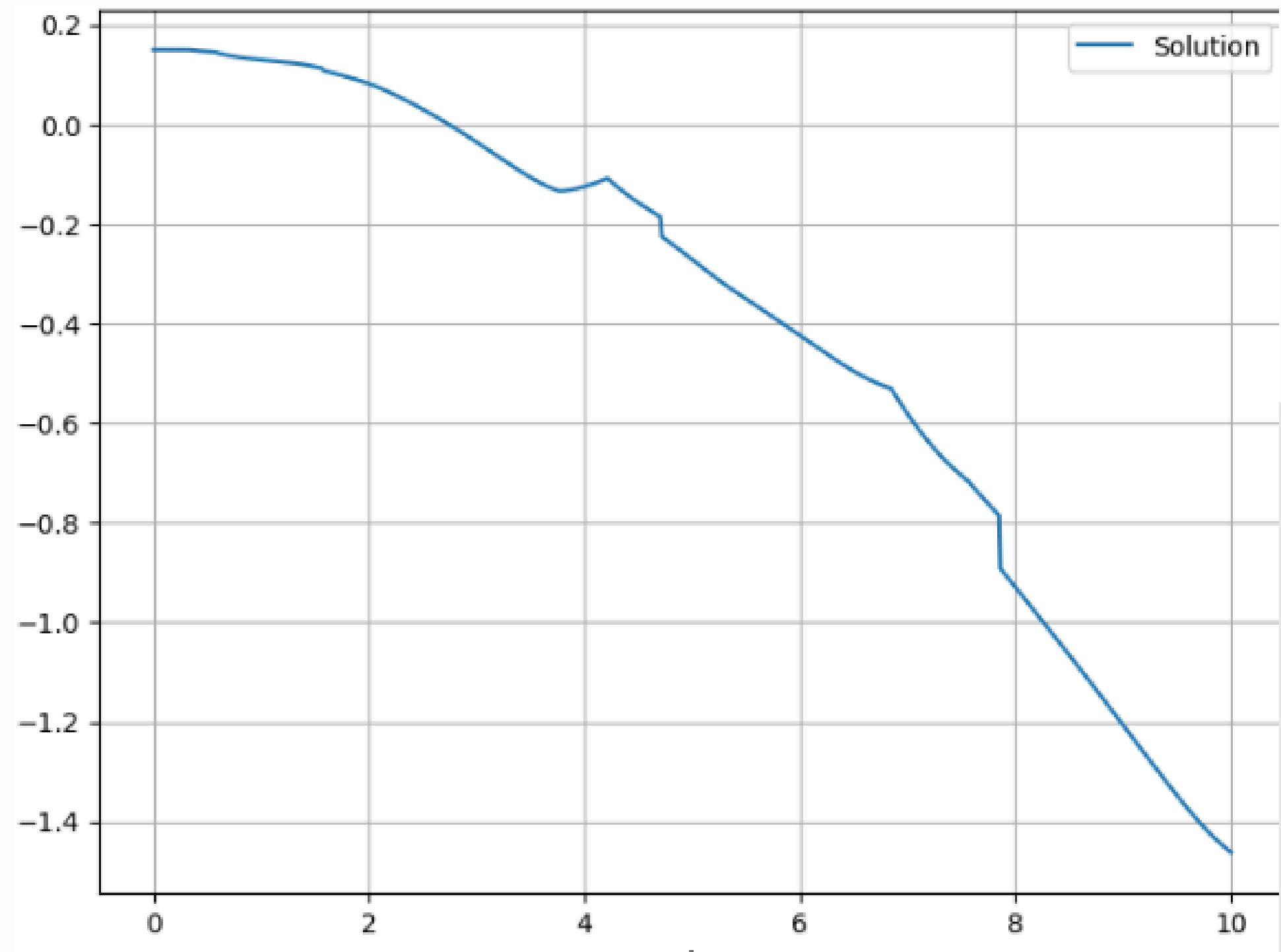


Actual Solution

Example

ANN lagaris result after
10,000 epochs

Hyper Parameters as
explained in original paper



Intuition Behind *Mixture of Experts* in ODE Solvers

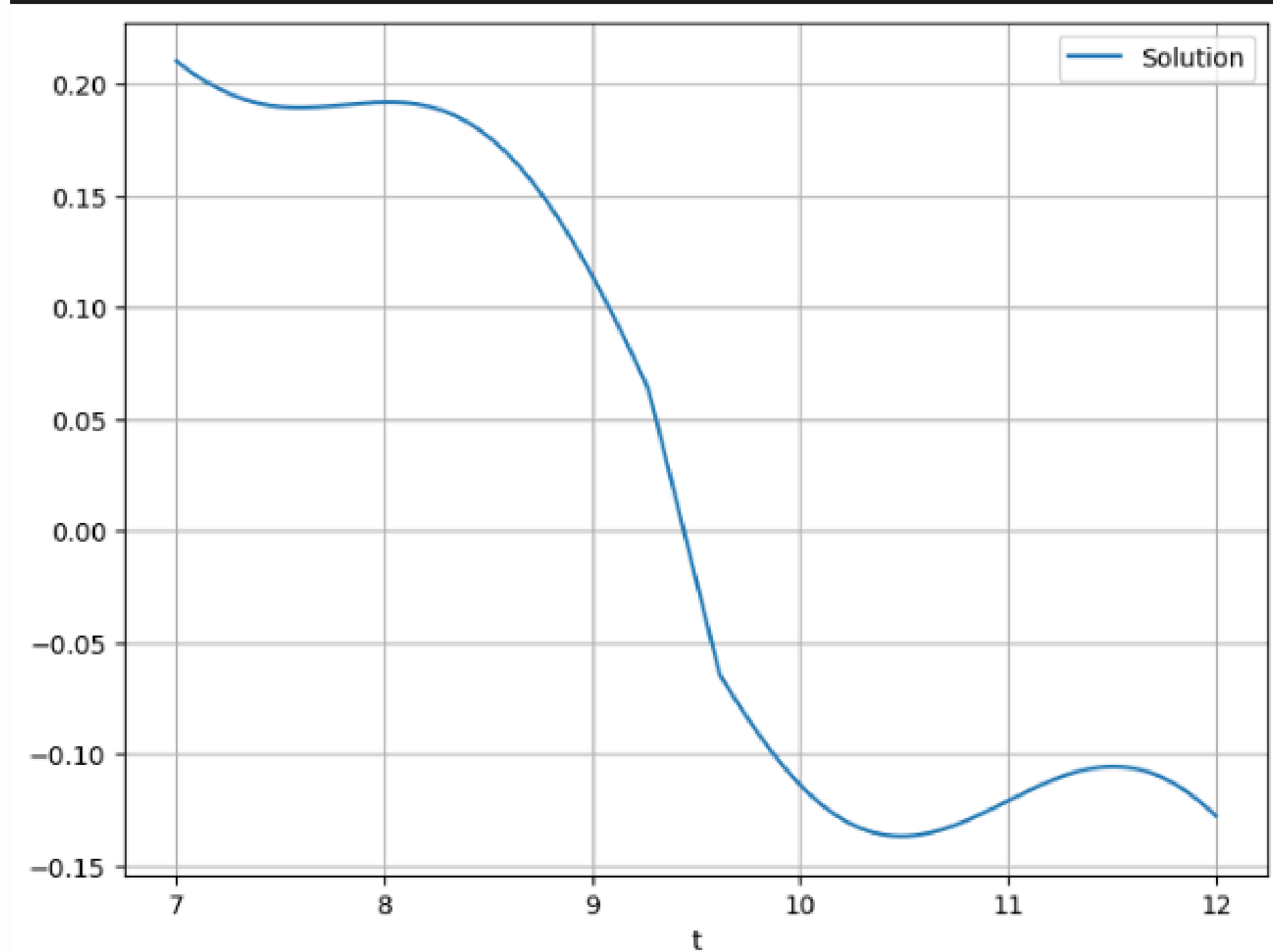
Differential equations often exhibit complex behaviors, and fitting a single neural network to capture all such dynamics can lead to suboptimal results. The intuition behind using a mixture of experts in ODE solvers is to leverage specialized networks, each trained to handle specific types or ranges of dynamics. Much like *oscillatory equations* where the output isn't linear, training a single model on a broad range of dynamics can lead to instability and hinder learning. By splitting the task across expert networks, each can focus on learning stable solutions within its own specialized range, leading to a more robust overall solution.

MOE

Number of Experts :- 3

Number Of Epochs :- 10,000

Activation Function :- Relu



MOE

Now model is performing better than previous but still output are not good. Model has started to learn something. I have observed the whole training process and observed model is not learning further , or learning very slowly

Although the Mixture of Experts model showed some effectiveness, I had concerns about its training performance. To address this, I attempted to partition the entire domain into smaller regions and train the model on each part separately. However, this approach did not yield the expected improvement in performance.

MOE

I observed that switching from the Mean Squared Error (MSE) loss function to the Mean Absolute Error (MAE) loss function improved the stability of the model's training. Although the overall performance did not improve significantly, training became more stable.

Since our main objective is to achieve a model that can solve differential equations with stable training dynamics, this stability is a promising step toward producing accurate outputs.

MOE

Number of Experts :- 1

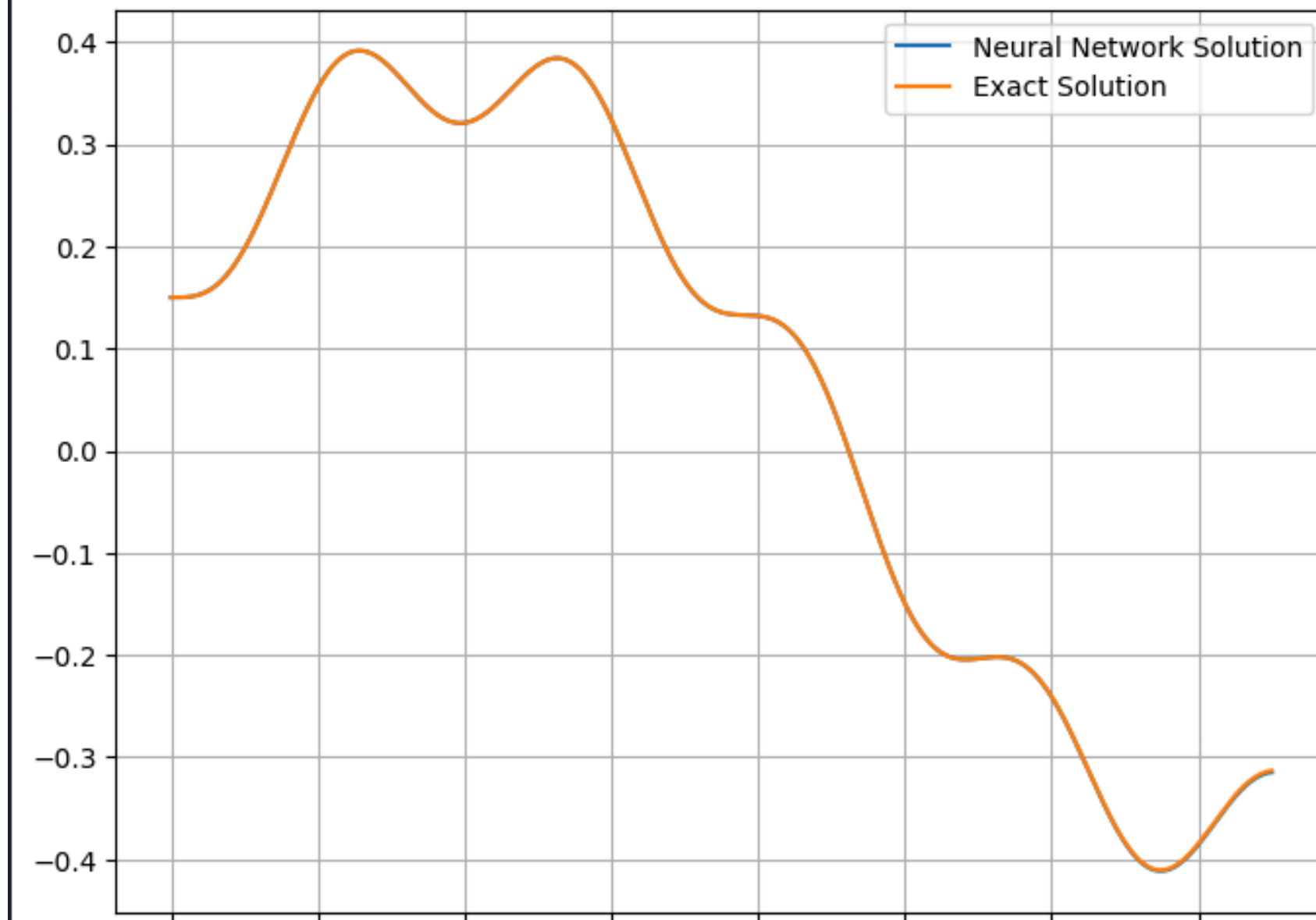
Number Of Epochs :- 40,000

Activation Function :- Tanh

Batch Size :- Domain Range

Number Of Layers :- 3

Number Of Neuron :- 32 per layer



MOE

Number of Experts :- 3

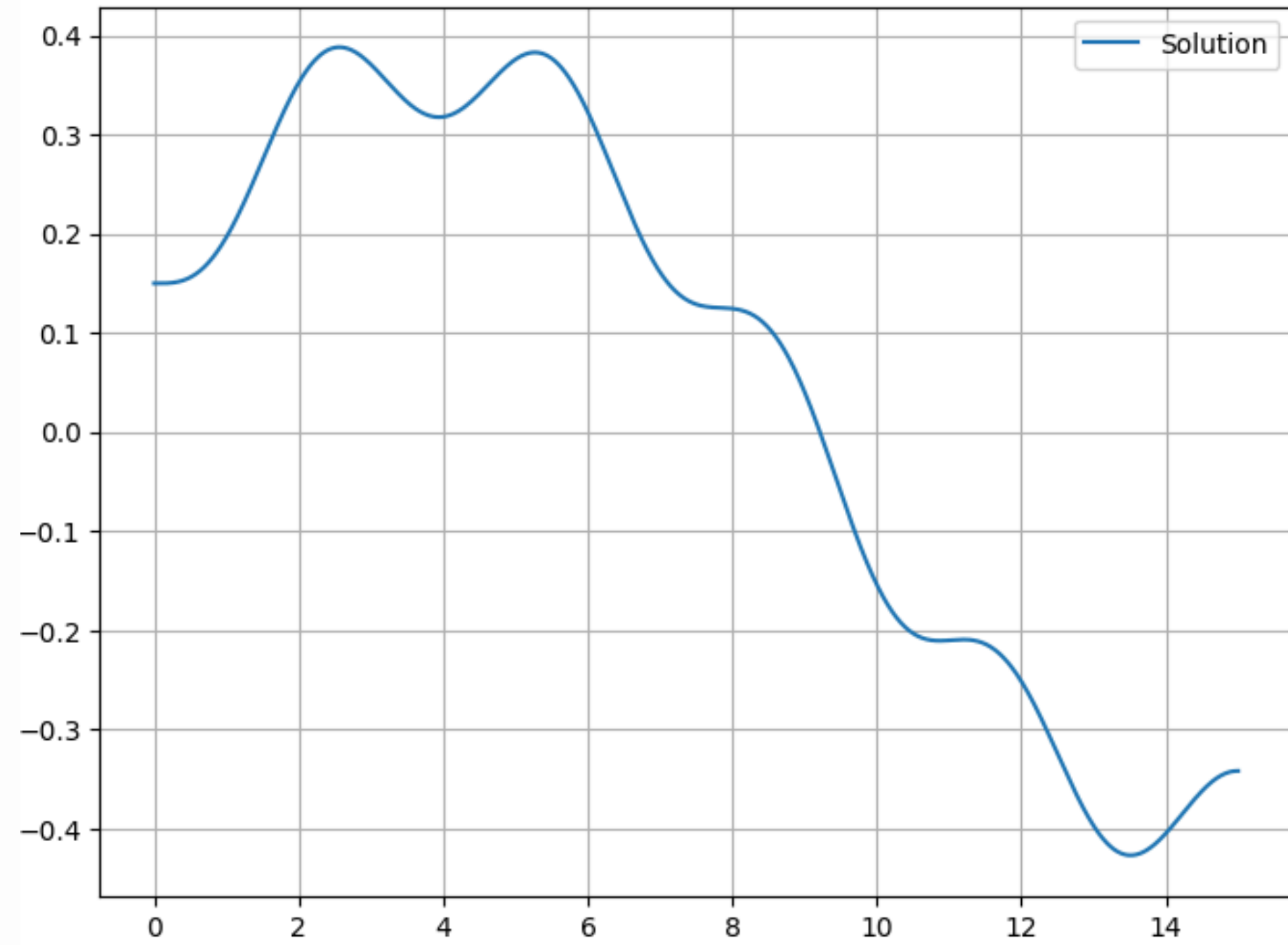
Number Of Epochs :- 40,000

Activation Function :- Tanh

Batch Size :- Domain Range

Number Of Layers :- 3

Number Of Neuron :- 32 per layer

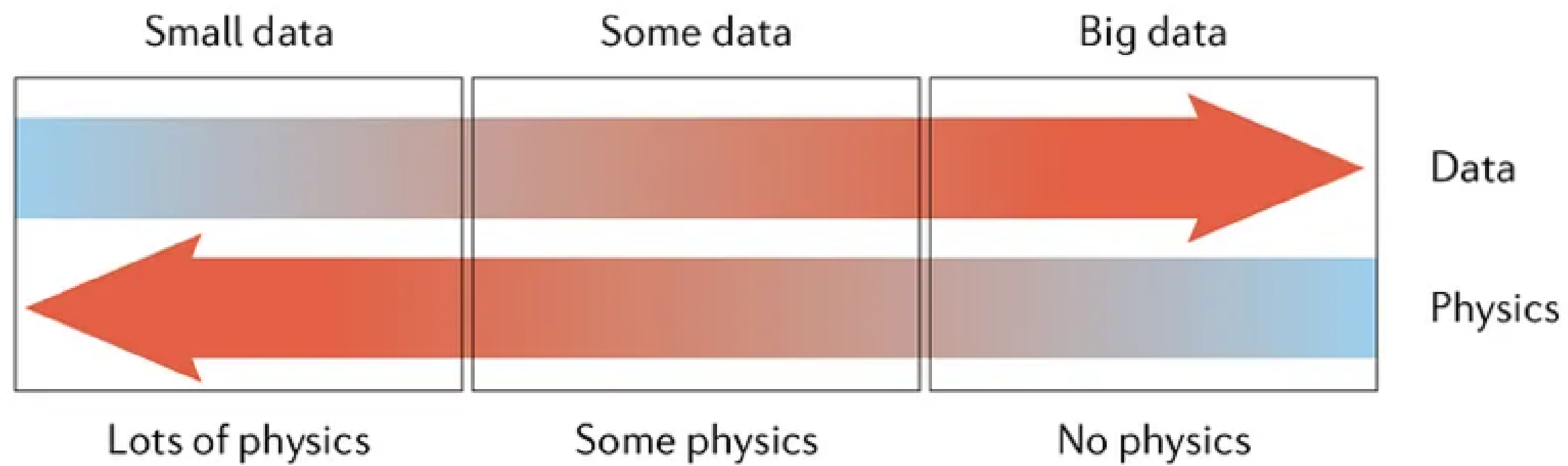


PINNS

Unlike traditional neural networks that rely purely on data, PINNs leverage differential equations to inform the model about the underlying physics of the system being modeled. This allows PINNs to learn solutions that respect physical constraints, making them particularly effective for solving problems governed by partial differential equations (PDEs) and ordinary differential equations (ODEs).

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

PINNS



Soft Plus Activation Function

The SoftPlus activation function is a smooth approximation of the ReLU activation function. It is defined to be continuously differentiable. Unlike the ReLU, which has a sharp nonlinearity at zero, the SoftPlus function gradually transitions from zero to positive values, making it more suited for applications that benefit from smooth gradients.

The equation for the SoftPlus activation function is:

$$f(x) = \ln(1 + e^x)$$

where x is the input to the function.

Properties of SoftPlus :-

- Range: The output of the SoftPlus function is always positive, with values ranging from 0 to + infinity.
- Smoothness : SoftPlus is continuously differentiable, making it more suitable for gradient-based optimization

Derivative of SoftPlus

- The derivative of the SoftPlus function, which is useful for backpropagation in neural networks, is given by:

$$f'(x) = \frac{1}{1 + e^{-x}} = \sigma(x)$$

✓ 1D Under-Damped Harmonic Oscillator

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

with the initial conditions

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

We will focus on solving the problem for the under-damped state, i.e. when

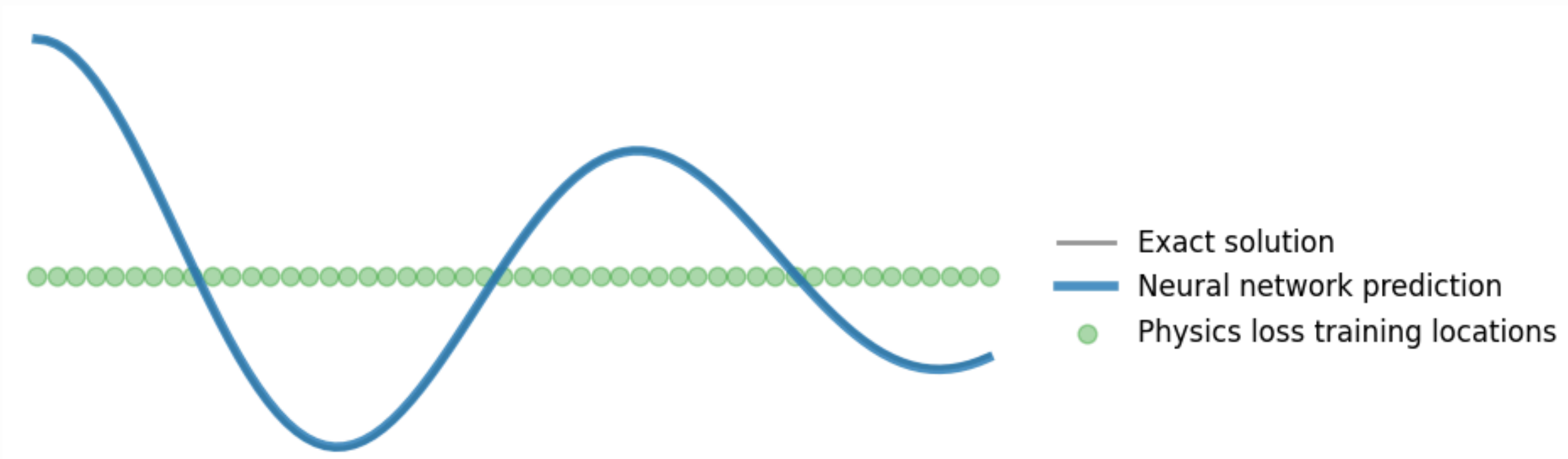
$$\delta < \omega_0 , \quad \text{with } \delta = \frac{\mu}{2m} , \quad \omega_0 = \sqrt{\frac{k}{m}} .$$

Exact Solution :

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

For this problem, we use $\delta = 1$, $\omega_0 = 10$, and try to learn the solution over the domain $x \in [0, 1]$.

Harmonic Oscillator



Harmonic Oscillator

