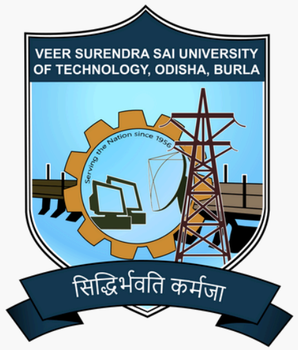
Veer Surendra Sai University Of Technology, Burla

Dept. of Computer Science and Engineering



Solving Burgers' Equation Using Hyperparameter-Optimized Physics-Informed Neural Networks

Presented by

Jyoti Prakash Panda (2102040004)

ShivbamSrinibas Bhoi (2102041042)

Guided by- Mr. Atul Vikas Lakra

Contents

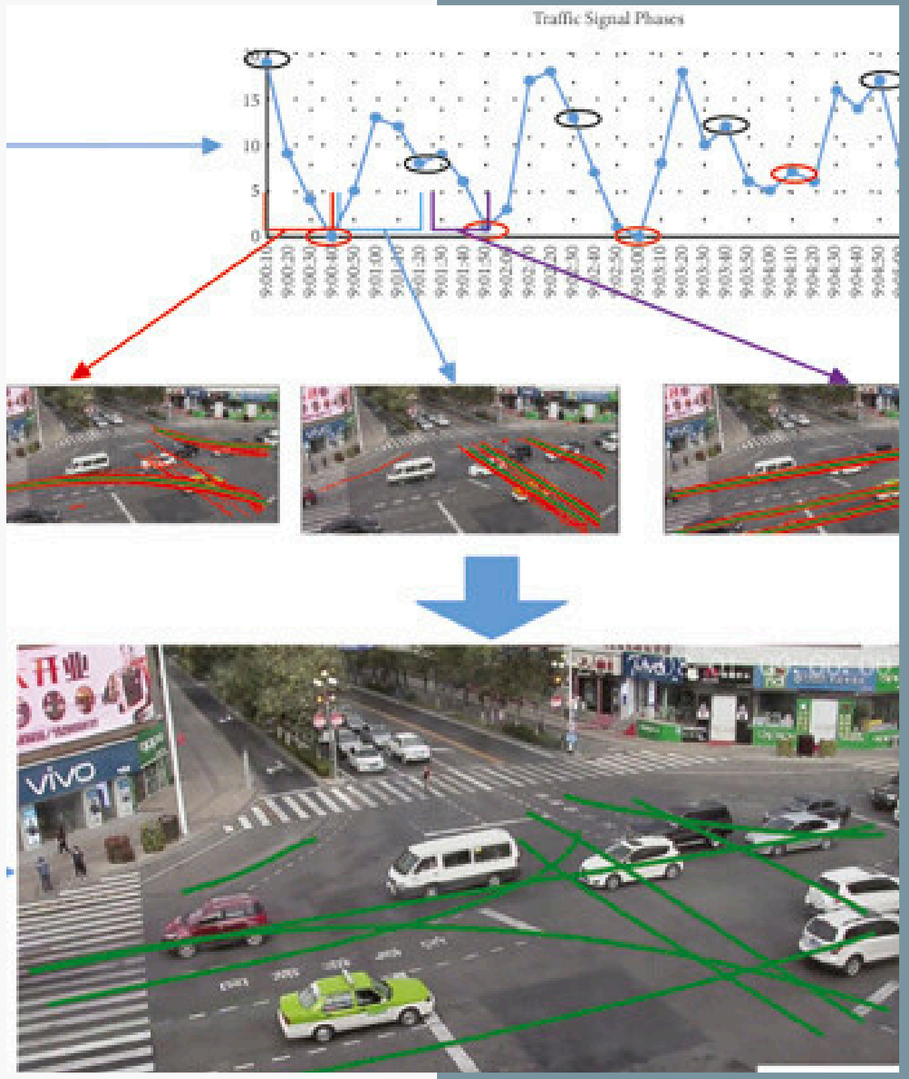
Introduction

- **Importance of fluid dynamics in various fields (engineering, meteorology, etc.).**

- **Complex systems modeled by equations.**

Example Scenarios:

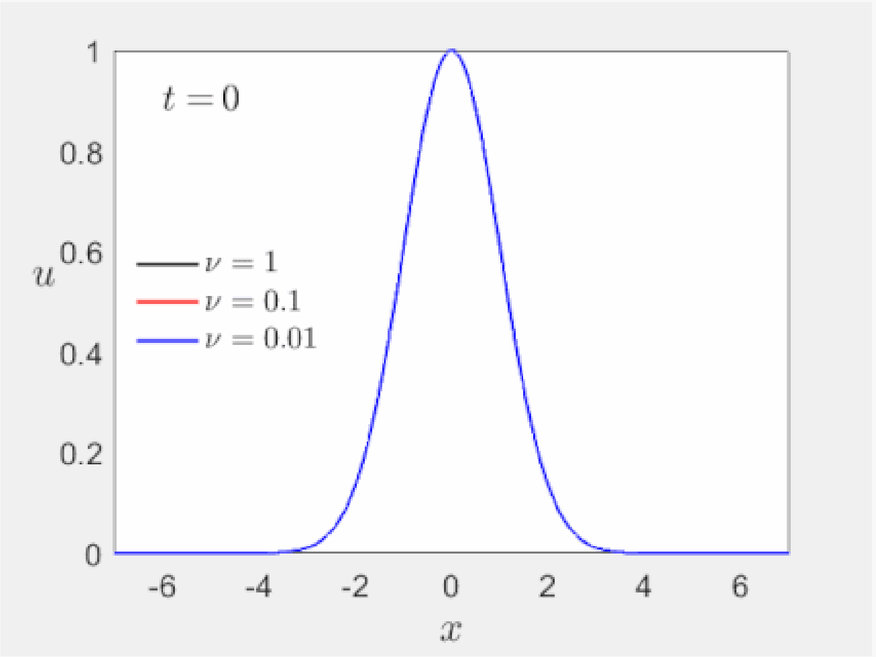
- **Wave Behavior Prediction**: shockwaves and their impact on physical systems.
- **Traffic Flow Modeling**: mathematical equations to optimize traffic management systems and reduce congestion.
- **Environmental Applications**: Modeling pollutant dispersion in air and water to design effective control measures.



What is Burgers' Equation?

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

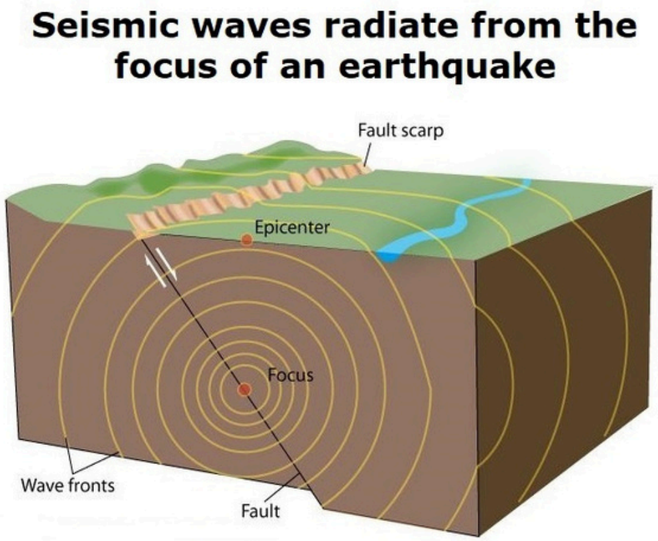
- Fundamental partial differential equation and convection–diffusion equation occurring in various areas of applied mathematics, such as fluid mechanics, nonlinear acoustics, gas dynamics, and traffic flow.
- For a given field u(x,t) and diffusion coefficient (or kinematic viscosity, as in the original fluid mechanical context).



Solutions of the Burgers equation starting from a Gaussian initial condition $u(x,0)=e^{-x^2/2}$.

The Importance of Understanding PDEs

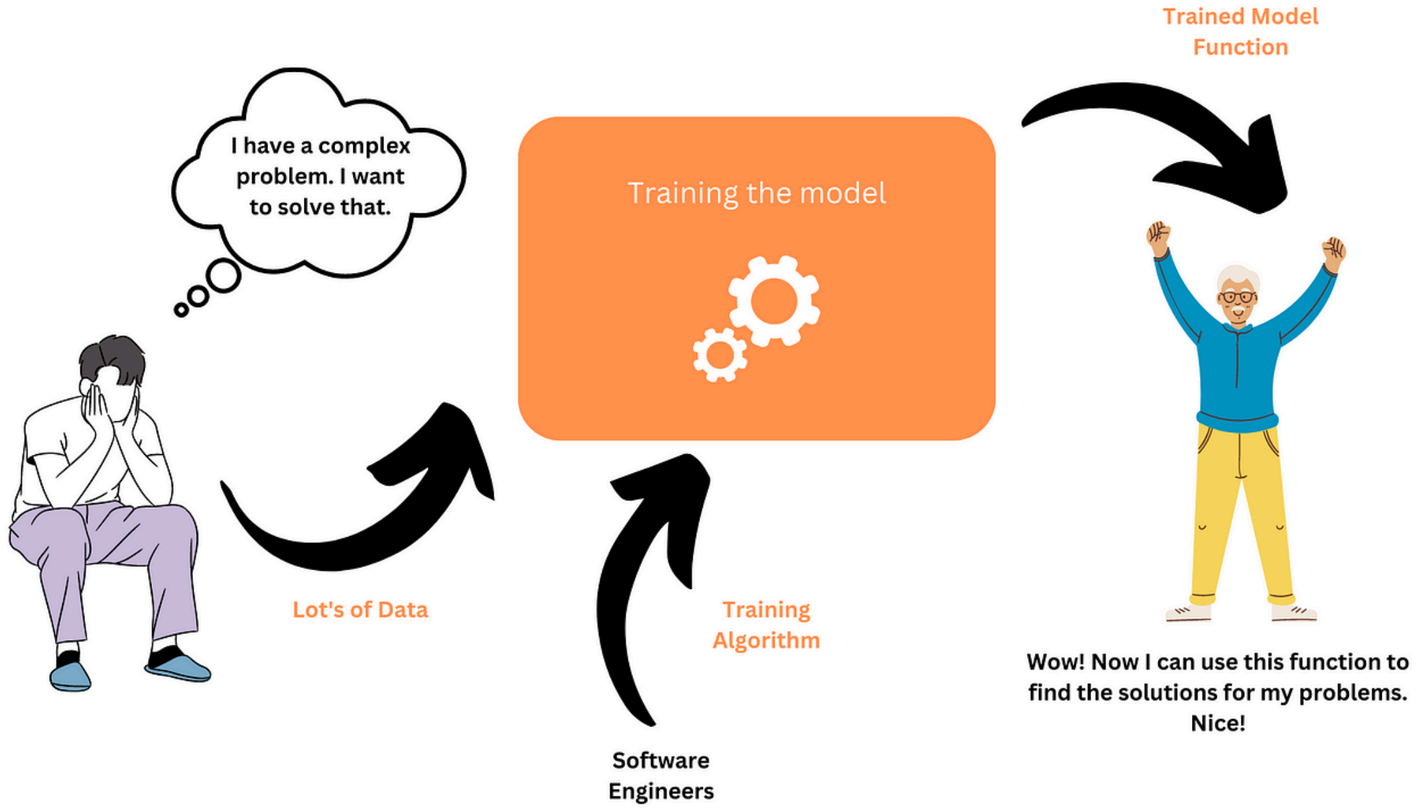
- **PDEs are mathematical equations that involve multiple independent variables, their partial derivatives, and a dependent variable.**
- **They are used to describe the behavior of physical systems where change occurs across both space and time.**



Captured by the Wave Equation, used in acoustics, seismology

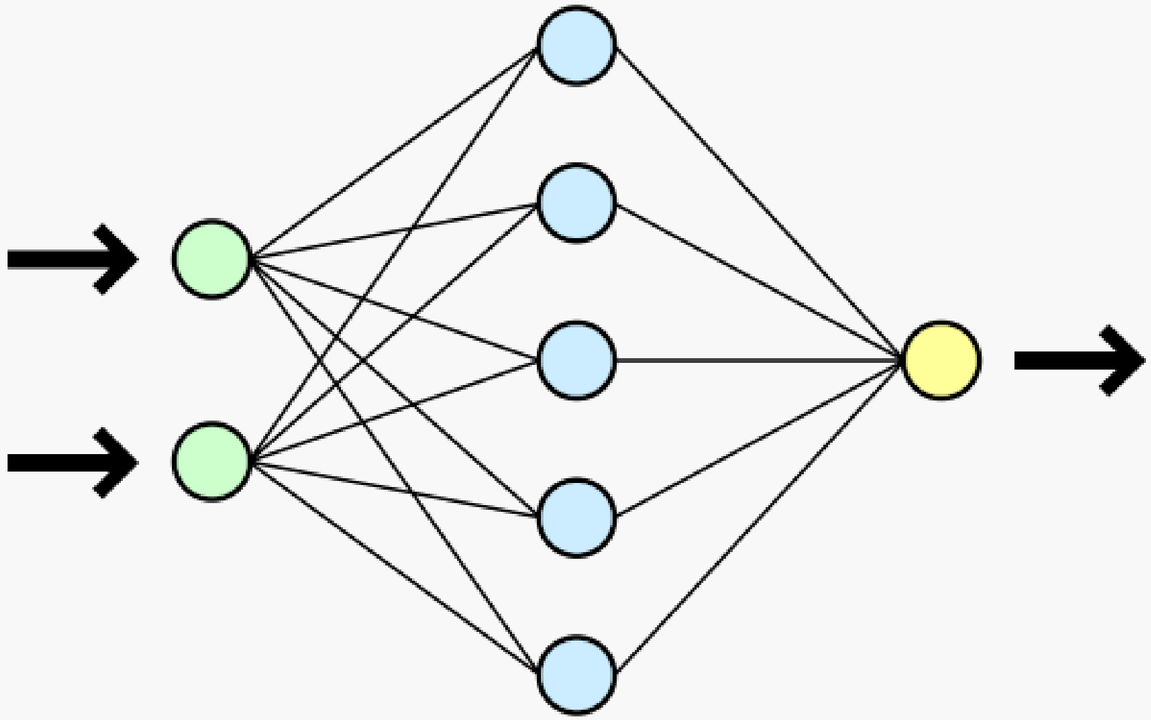
Challenges in Traditional Methods

- Numerical instability.
- Difficulty in capturing shock waves.
- High computational costs.



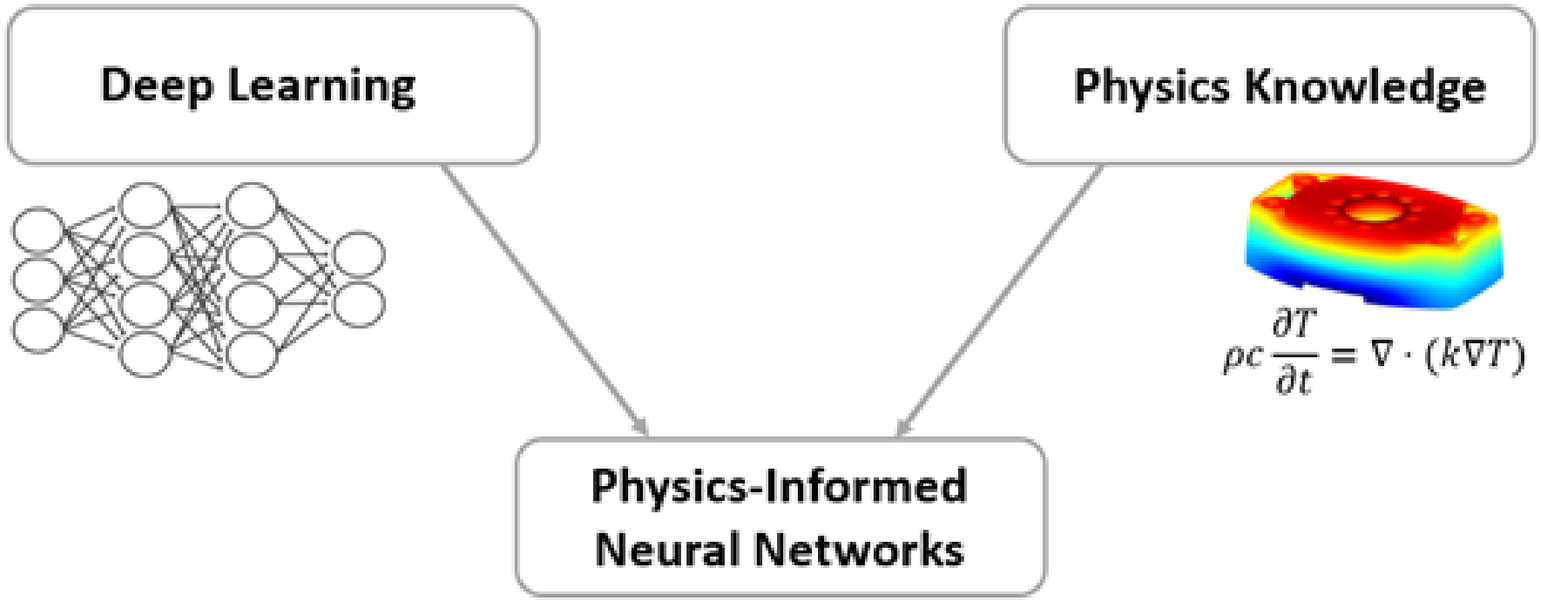
Enter Neural Networks

- Computational models inspired by the human brain.
- Structure: Layers of interconnected nodes (neurons).



Physics-Informed Neural Networks (PINNs)

- PINNs ensure the neural network satisfies the equation's dynamics (e.g., advection-diffusion balance)



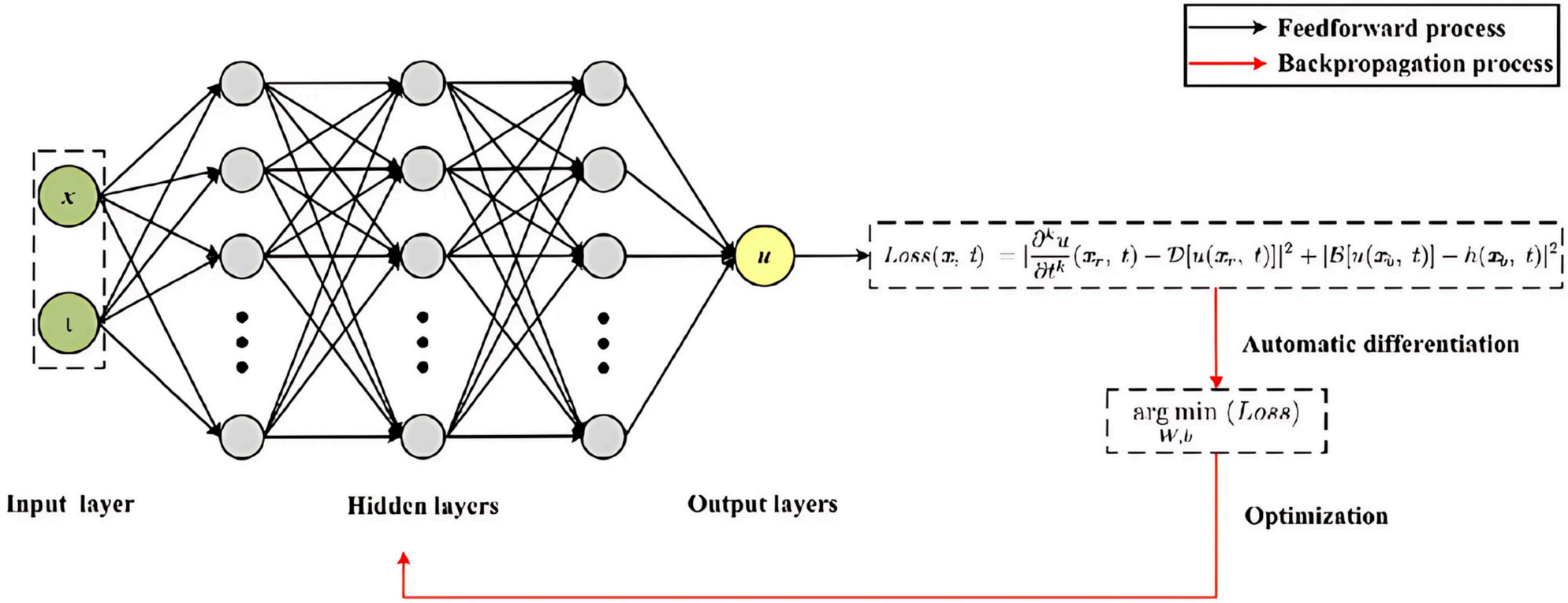
How PINNs Work?

Overview

- **Data Generation from PDEs:**
    - PDEs define the relationship between physical variables.
    - PINNs use the governing equations to generate synthetic data points (e.g., spatial and temporal domains).
- **Loss Function Formulation:**
    - **Residual Los**s: Ensures predictions satisfy the PDE (enforcing physics constraints).
    - **Data Loss**: Matches predictions with available observational data.
- **Training Process:**
    - A neural network is trained to minimize the total loss function.
    - Optimization ensures the solution satisfies the PDE while aligning with any real-world data.

The Structure of a PINN

- Architecture overview:



Defining the Problem

Burgers' Equation:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2} \qquad x \in [-1, 1] \qquad t \in [0, 1]$$

Let,

$$u_t = \frac{\partial u}{\partial t}$$

$$u_x = \frac{\partial u}{\partial x}$$

$$u_{xx} = \frac{\partial^2 u}{\partial x^2}$$

So,

$$u_t + u(u_x) = \nu u_{xx}$$

After rearranging,

$$u_t + u(u_x) - \nu u_{xx} = 0$$

Constructing the Neural Network

Steps to Design the Neural Network:

- Choose the number of layers (depth).
- Determine the number of neurons per layer (width).
- Select Activation Functions (ReLU, Sigmoid/Tanh)
- Properly initialize weights to improve convergence speed.

A Neural Network is a function:

$$NN(X) = W_n\sigma_{n-1}(W_{n-1}\sigma_{n-2}(\dots(W_2(W_1X + b_1) + b_2)+..) + b_{n-1}) + b_n$$

Note: We usually train our NN by iteratively minimizing a loss function (MSE:mean squared error) in the training dataset(known data)

Constructing the Neural Network

We can use a neural network to approximate any function (Universal Aproximation Theorem)

$$NN(x,t) \approx u(x,t)$$

Since NN is a function, we can obtain its derivatives: $\frac{\partial NN}{\partial t}$, $\frac{\partial^2 NN}{\partial x^2}$.

Assume $$NN(t,x) \approx u(t,x)$$

Then $$\frac{\partial NN}{\partial t} + N(\frac{\partial NN}{\partial x}) - \nu \frac{\partial^2 NN}{\partial x^2} \approx u_t + u(u_x) - \nu u_{xx} = 0$$

And $$\frac{\partial NN}{\partial t} + N(\frac{\partial NN}{\partial x}) - \nu \frac{\partial^2 NN}{\partial x^2} \approx 0$$

We define this function as f: $$f(t,x) = \frac{\partial NN}{\partial t} + N(\frac{\partial NN}{\partial x}) - \nu \frac{\partial^2 NN}{\partial x^2}$$

PINNs' Loss function

We evaluate our PDE in a certain number of "collocation points" (Nf) inside our domain (x,t). Then we iteratively minimize a loss function related to f:

$$MSE_f = \frac{1}{N_f} \sum_{i=1}^{N_f} |f(t_f^i, x_f^i)|^2$$

Usually, the training data set is a set of points from which we know the answer. In our case, we will use our boundary(BC) and initial conditions(IC).
Since we know the outcome, we select Nu points from our BC and IC and used them to train our network.

$$MSE_u = \frac{1}{N_u} \sum_{i=1}^{N_u} |y(t_u^i, x_u^i) - NN(t_u^i, x_u^i)|^2$$

$$MSE = MSE_u + MSE_f$$

Training the PINN

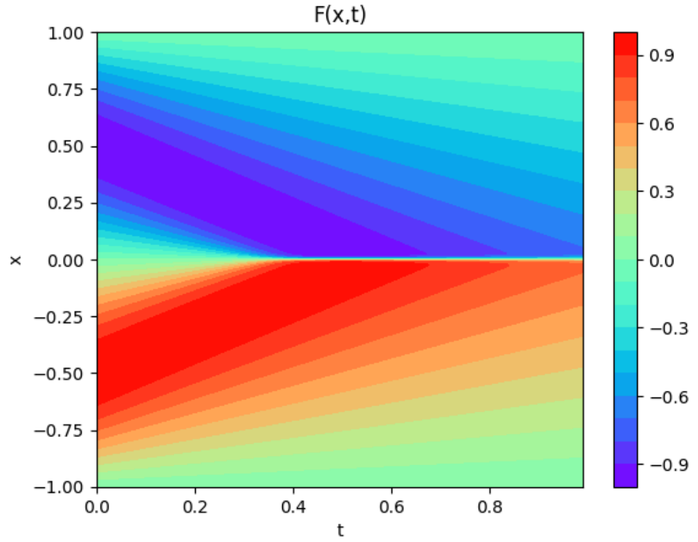
Method Used:

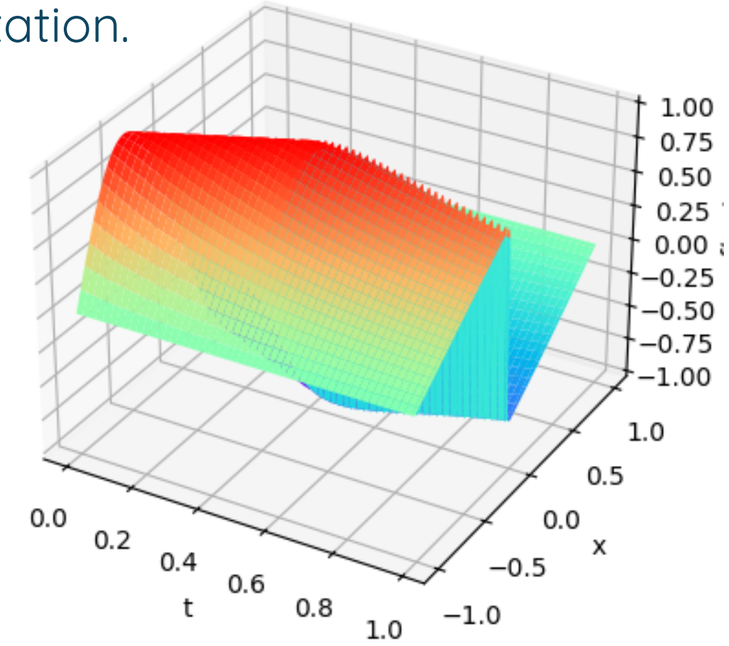
- **L-BFGS** (Limited-memory Broyden-Fletcher-Goldfarb-Shanno)

Benefits:

- Faster convergence for smooth and differentiable loss functions.
- Efficient for problems with relatively small datasets, such as PINNs.
- Utilizes second-order derivative approximations for precise optimization.

3d Representation of Original Data

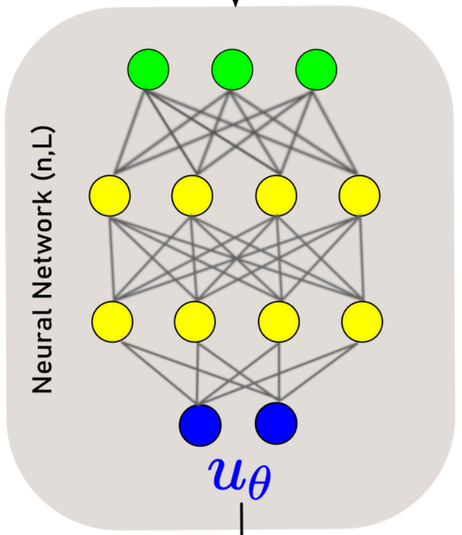




Recap & Next Step:

Input Data

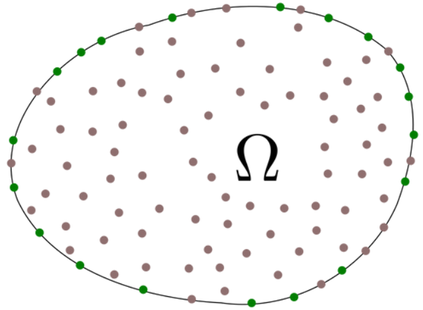
PDE + Boundary + (Experiment)



Non-Linear PDE

$$\partial_t u(x,t) + \mathcal{N}[u(t,x)] = 0$$
$$u(x,0) = u_0(x), \quad x \in \Omega$$
$$u(x,t) = g(x,t) \quad x \in \partial\Omega \times [0,T]$$

- Boundary Collocation Point
- PDE Collocation Point



Boundary Loss

$$\hat{J}_{BC}(\boldsymbol{\theta}) := \frac{1}{N_u}\sum_{i=1}^{N_u}\left[\left(\hat{\mathbf{u}}(\mathbf{x_i},\mathbf{0};\boldsymbol{\theta}) - \mathbf{u_0}(\mathbf{x_i})\right)^2 + \left(\hat{\mathbf{u}}(\mathbf{x_i},\mathbf{t_i};\boldsymbol{\theta}) - \mathbf{g}(\mathbf{x_i},\mathbf{t_i})\right)^2\right]$$

PDE Loss

$$\hat{J}_{PDE}(\boldsymbol{\theta}) := \frac{1}{N_f}\sum_{i=1}^{N_f}\left[\left(\partial_t\hat{\mathbf{u}}(\mathbf{x_i},\mathbf{t_i};\boldsymbol{\theta}) + \mathcal{N}[\hat{\mathbf{u}}(\mathbf{x_i},\mathbf{t_i};\boldsymbol{\theta})]\right)^2\right]$$

[Automatic Differentiation]

Experiment Loss

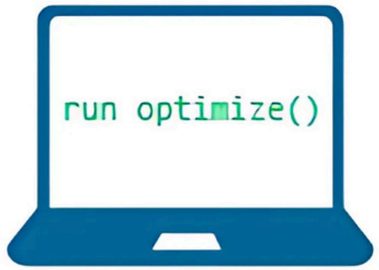
$$\hat{J}_{exp}(\boldsymbol{\theta}) := \frac{1}{N_e}\sum_{i=1}^{N_e}\left[\left(\hat{\mathbf{u}}(\mathbf{x_i},\mathbf{t_i};\boldsymbol{\theta}) - \mathbf{u}(\mathbf{x_i},\mathbf{t_i})\right)^2\right]$$

if experimental data is available*

$$\hat{J}_{PINN}(\boldsymbol{\theta}) := \hat{J}_{PDE}(\boldsymbol{\theta}) + \lambda_1\,\hat{J}_{BC}(\boldsymbol{\theta}) + (\lambda_2\,\hat{J}_{exp}(\boldsymbol{\theta}))$$

Hyperparameters → OPTIMIZER → Output

Hyperparameter Optimization



Hyperparameters | **Parameters** | **Score**

n_layers = 3
n_neurons = 512
learning_rate = 0.1

Weights optimization

85%

n_layers = 3
n_neurons = 1024
learning_rate = 0.01

Weights optimization

80%

n_layers = 5
n_neurons = 256
learning_rate = 0.1

Weights optimization

92%

Hyperparameter Optimization

Importance:

- Hyperparameters define how the neural network is structured and trained.

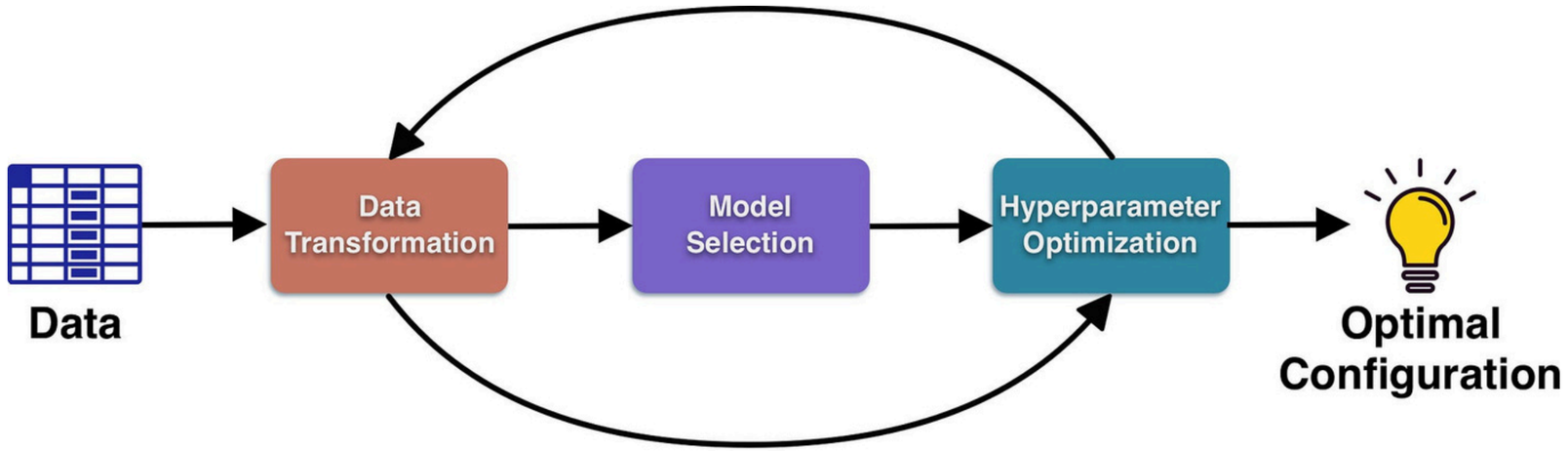
Why Optimize Hyperparameters?

- **Impact of Poor Choices:**
  - Wrong hyperparameters can lead to:
  - Poor convergence: The model fails to minimize the loss effectively.
  - Overfitting: The model performs well on training data but poorly on unseen data
- **Benefits of Optimization:**
  - Improves training efficiency.
  - Enhances accuracy of predictions.
  - Prevents underfitting and overfitting.

Key Hyperparameters in PINNs

Hyperparameters:

- **Learning Rate** – speed of updates during training.
- **Number of Layers** – depth affecting complexity capture.
- **Neurons per Layer** – width impacting representational power.
- **Activation Functions** – introduce nonlinearity into learning process.
- **Batch Size** – number of samples processed before updating weights.
- **Activation Function**



Fixed Parameters Training

Parameters we've taken:

- learning rate (lr)= 10^{-1}
- Hidden Layers = 8
- Layer Width (Neurons per layer)= 20
- Steps= 10000
- Activation Function = TanH

Result:

- Training Time: 99.94
- Test Error: 0.092899

Hyperparameter Optimization in PINN

Parameters we've taken:

- learning rate (lr)= [10^{-8}, 10^{-2}]
- Hidden Layers = [4, 6, 8, 10, 16, 32, 40, 46, 52, 64, 70]
- Layer Width(Neurons per layer) = [32, 64, 128, 256, 512]
- Steps= [5000, 10000, 20000, 30000, 50000]
- Activation Function = [TanH, SinH, CosH, CotH]
- Number of Iterations = 2k

Result Analysis

Fixed Parameters:

- learning rate (lr)= 0.1
- Hidden Layers = 8
- Layer Width (Neurons per layer)= 20
- Steps= 10000
- Activation Function = TanH

- Training Time: 99.94
- Test Error: 0.092899

Hyperparameters:

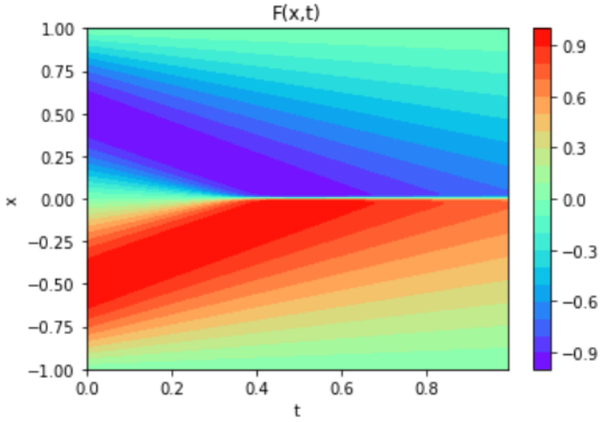
Best Parameters:

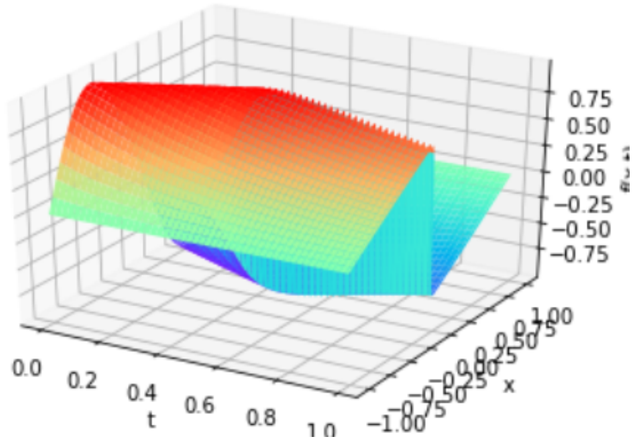
- learning rate (lr)= 0.00001857137188978888
- Hidden Layers = 8
- Layer Width = 128
- Steps= 20000
- Activation Function = TanH

- Training Time: 86.40
- Test Error: 0.00034103

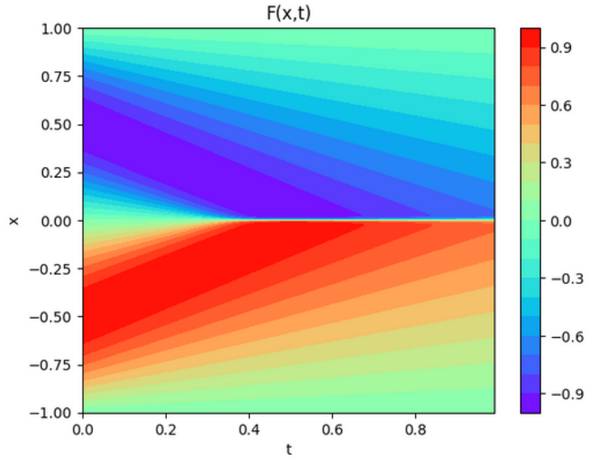
Graphical Comparison

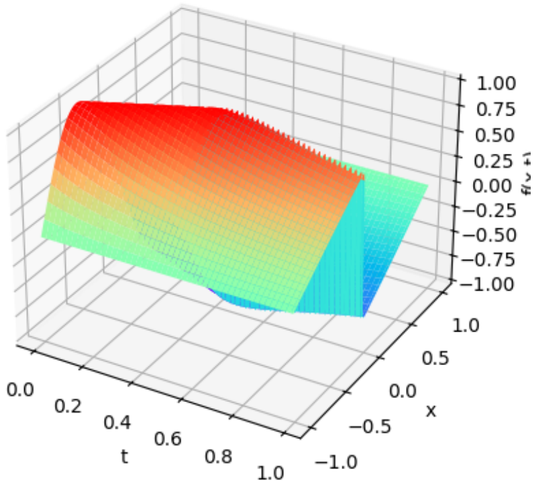
Fixed Parameters:





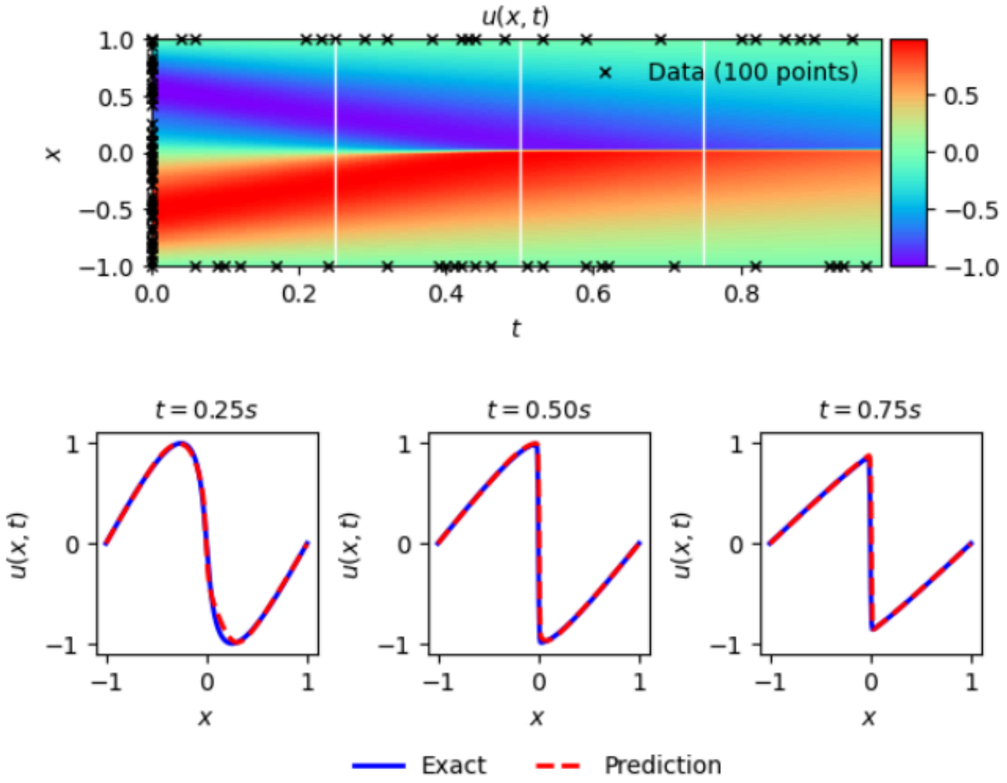
Hyperparameters:



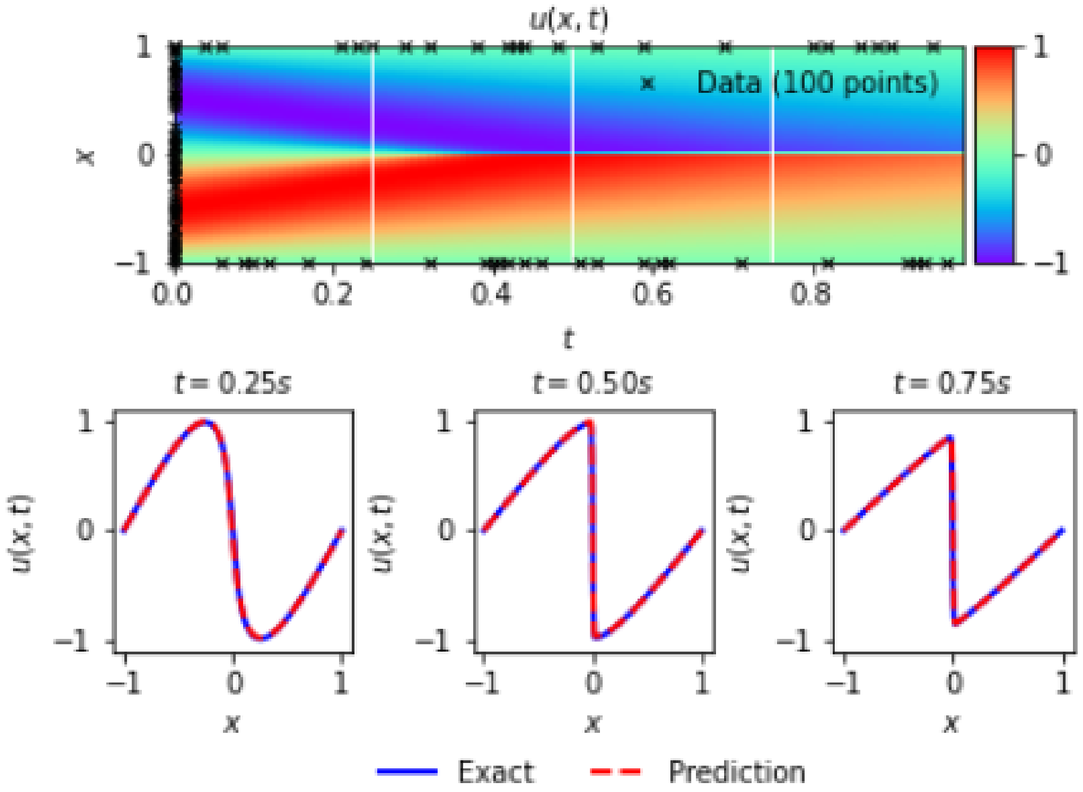


Graphical Comparison

Fixed Parameters:

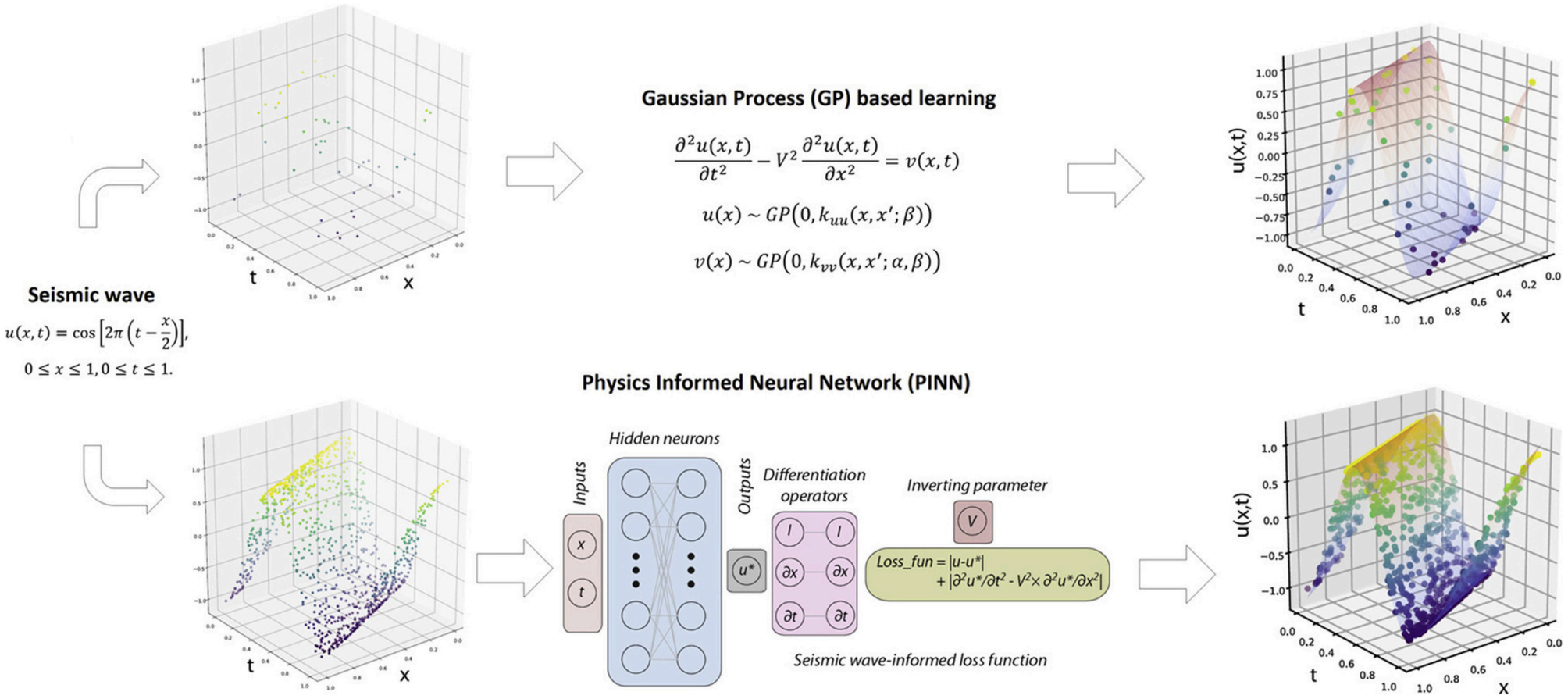


Hyperparameters:



Application of hyperparameter optimized PINN

Seismic wave (earthquake) demonstration:



References

- Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations." Journal of Computational Physics, 378, 686-707.
- Karniadakis, G. E., Kevrekidis, I. G., & Lu, L. (2021). "Physics-informed machine learning." Nature Reviews Physics, 3(6), 422-440.
- Long, Z., Lu, Y., Ma, X., & Dong, B. (2019). "PDE-Net: Learning PDEs from data." ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 1-10.
- Chen, J., & Zhang, Y. (2020). "Deep learning for solving partial differential equations: A review." Computational Mechanics, 66(2), 259-277.
- Optuna Documentation: "Optuna: A Next-generation Hyperparameter Optimization Framework." [Online]. Available: https://optuna.org/

Conclusion

Hyperparameters such as learning rate, number of layers, and neurons significantly impact the performance of PINNs. Proper optimization can lead to better convergence, accuracy, model generalization and cost optimization.

Thank you