Harnessing Machine Learning for Physics: A Synergistic Approach with Physics Informed Neural Networks and Kolmogorov Arnold Networks in Solving Complex Partial Differential Equations

Shusrith S
Department of AIML
PES University
pes1202201377@pesu.pes.edu

Namita Achyuthan
Department of AIML
PES University
pes1202201119@pesu.pes.edu

Siddhi Zanwar
Department of AIML
PES University
pes1202201256@pesu.pes.edu

***Abstract*—This work demonstrates a comprehensive pipeline for data generation, noise incorporation, and machine learning-based noise removal for partial differential equations. We leverage Physics-Informed Neural Networks (PINNs) to solve the partial differential equation (PDE) with noise, and Kolmogorov-Arnold Networks (KANs) for efficient denoising. The report outlines methods for dataset generation, training, and comparative analysis of noisy versus clean data performance. Results show that the KAN achieves robust noise reduction while preserving the PDE structure.**

***Index Terms*—Burgers' equation, Physics-Informed Neural Networks, Kolmogorov-Arnold Networks, noise removal, partial differential equations.**

I. Introduction

A. Partial Differential Equations

Partial Differential Equations (PDEs) are equations that involve unknown multivariable functions and their partial derivatives. They are used to describe a wide range of physical phenomena, such as heat conduction, fluid flow, and wave propagation. The general form of a PDE in two variables (for simplicity) is:

$$F(x, y, u, u_x, u_y, u_{xx}, u_{xy}, u_{yy}) = 0$$

where $u(x, y)$ is an unknown function of the variables x and y, and u_x, u_y, u_{xx}, etc., represent its partial derivatives with respect to x and y. The solution of a PDE typically requires boundary and initial conditions that specify the value of the function at the boundaries of the domain and at an initial time.

PDEs can be classified into three categories based on their characteristics:

- **Elliptic** (e.g., Laplace's equation),
- **Parabolic** (e.g., heat equation),
- **Hyperbolic** (e.g., wave equation).

Burgers' equation, which is a nonlinear PDE, falls under the category of **hyperbolic equations** because it models wave propagation.

B. Burgers' Equation

Burgers' equation is a fundamental PDE in fluid dynamics and traffic flow, and it is often used to model shock waves and turbulence. The equation can be written as:

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

where:

- $u(x, t)$ is the unknown function (representing the velocity or the flow),
- x is the spatial variable,
- t is the time variable,
- ν is the kinematic viscosity (a constant).

The left-hand side represents advection (convective transport) of the quantity u, while the right-hand side represents diffusion due to viscosity. The equation can model shock formation (wave steepening) and turbulence, making it a useful tool in understanding how waves evolve in fluids and traffic systems.

- **Shock waves:** When the nonlinearity of the equation causes the solution to steepen over time, leading to discontinuities (shock waves).
- **Turbulence:** When small disturbances lead to chaotic, irregular behavior in the fluid flow.

Burgers' equation has wide applications in fluid dynamics, traffic flow, acoustics, and other areas, and its solutions can often reveal insights into the behavior of complex systems.

C. Noise in PDEs

In real-world systems, data and models are often subject to **noise**, which can arise from various sources such as measurement errors, environmental fluctuations, or computational inaccuracies. This noise introduces significant challenges when solving PDEs, as traditional numerical methods typically assume clean, noise-free data. The presence of noise in the initial or boundary conditions, or even in the physical properties (like viscosity), can distort the solution, making it harder to accurately predict the system's behavior.

In the case of Burgers' equation:

- **Measurement noise:** In fluid dynamics, sensors may introduce errors in measuring velocity or pressure.
- **Turbulent flow:** In real-world scenarios, the flow is often turbulent, which introduces complex, high-frequency variations in the data, contributing to noise.
- **Numerical noise:** Even when using computational methods like finite difference or finite element methods, numerical approximations introduce errors, which can become significant in large-scale or long-time simulations.

Noise-aware techniques, which are an integral part of modern machine learning approaches, aim to deal with such noisy data by either directly modeling the noise or by regularizing the learning process to prevent the model from overfitting to noise.

D. Physics-Informed Neural Networks (PINNs)

Physics-Informed Neural Networks (PINNs) are a novel approach to solving PDEs by integrating the governing physical laws (represented as PDEs) directly into the training process of neural networks. Instead of relying purely on data to train a model, PINNs leverage the known physical laws that govern the system.

In the case of solving PDEs, PINNs work as follows:

- **Network architecture:** A neural network is used to approximate the solution of the PDE. The network takes spatial and temporal coordinates as inputs and produces the approximate solution $u(x,t)$ as output.
- **Loss function:** The loss function of the neural network is modified to not only penalize the error between the predicted solution and the observed data (e.g., boundary and initial conditions) but also penalize the violation of the governing PDE. For example, for Burgers' equation, the loss would include a term that measures how much the network's output deviates from the true solution of the PDE.

The general structure of the PINN loss function for a PDE like Burgers' equation might look like:

$$\mathcal{L} = \mathcal{L}_{\text{data}} + \mathcal{L}_{\text{PDE}} + \mathcal{L}_{\text{BCs}}$$

where:

- $\mathcal{L}_{\text{data}}$ is the loss associated with the data (e.g., initial and boundary conditions),
- $\mathcal{L}_{\text{PDE}}$ is the loss associated with how well the network satisfies the PDE (i.e., residual of the PDE),
- $\mathcal{L}_{\text{BCs}}$ ensures the boundary conditions are satisfied.

PINNs allow for solving PDEs in complex geometries and with limited data, providing a powerful tool for simulations where traditional methods may struggle, especially when dealing with noisy or incomplete data.

E. Using PINNs to Solve PDEs

Using PINNs to solve a PDE like Burgers' equation involves the following steps:

1) **Design the Neural Network:** A neural network is designed with inputs corresponding to spatial and temporal variables and outputs corresponding to the solution of the PDE (in this case, the velocity field).
2) **Define the Loss Function:** The loss function incorporates the residual of the PDE, the initial and boundary conditions, and possibly noisy data.
3) **Training:** The network is trained to minimize the loss function, which ensures that the solution satisfies both the PDE and the physical constraints (e.g., boundary conditions).
4) **Prediction:** Once the network is trained, it can be used to predict the solution of the PDE for any given set of spatial and temporal inputs.

One of the advantages of PINNs is their ability to work with limited or noisy data. The network can learn to satisfy the governing PDE even when data points are sparse or contaminated by noise, which is often the case in real-world applications like traffic flow and fluid dynamics.

F. Kolmogorov-Arnold Networks (KANs)

Kolmogorov-Arnold Networks (KANs) are neural networks inspired by Kolmogorov's Superposition Theorem, which states that any continuous function can be approximated by a finite sum of nonlinear functions of the input variables. KANs are designed to represent such functions more efficiently by breaking down complex, high-dimensional functions into simpler components.

In the context of solving PDEs like Burgers' equation, **KANs** offer several advantages:

- **Efficient Approximation:** KANs can efficiently approximate complex, high-dimensional solutions by decomposing the problem into simpler components. This is particularly useful when the solution to a PDE involves intricate patterns like turbulence or shock formation.
- **Noise Robustness:** Since KANs are designed to handle nonlinearities and complex structures, they are robust to noisy data and can capture the underlying patterns even in the presence of noise.
- **Generalization:** KANs allow for better generalization to different conditions, which is important when solving PDEs under varying physical parameters (e.g., viscosity, initial conditions, or boundary conditions).

By integrating **KANs** with **PINNs**, we can combine the power of physical modeling with efficient and robust function approximation. This hybrid approach can be particularly useful for simulating PDEs in environments where the data is noisy or incomplete, while still respecting the underlying physical laws.

II. Dataset Generation

The dataset generation code is adapted from https://github.com/pdebench/PDEBench, with modifications to introduce noise into the data. Burgers' equation is used as a proof of concept, with noise types randomly chosen between skewed normal and exponential distributions.

A. Initial Conditions

The initial conditions of the PDE are defined based on the specified mode:

- *sin*: Sine wave
- *gaussian*: Gaussian distribution
- *step*: Step function
- *possin*: Positive sine wave
- *sinsin*: Double sine wave with varying frequencies

B. Noise Generation

Here's a refined version of your text:

Our initial experiments revealed that Physics-Informed Neural Networks (PINNs) performed well when the input noise followed a zero-centered white Gaussian distribution, as the model seemed to implicitly assume this noise type. However, real-world data typically involves noise that deviates from this idealized assumption. Further experimentation showed that PINNs struggled significantly when the noise deviated from a Gaussian distribution. To address this, we introduced noise into the data by randomly selecting between skew-normal or exponential distributions. The noise is then scaled by the noise_level parameter, introducing variability in the dataset to enhance the model's robustness to a wider range of noise types.

C. Boundary Conditions

The boundary conditions are implemented for a 1D array with the option to add noise to the boundaries.

- **Periodic**: The boundary values at the two ends of the array are set to match the corresponding values at the opposite ends, effectively creating a continuous, looped domain.
- **Noise**: Noise is added to the boundary cells, with the noise scaled by the specified noise_level parameter. This helps simulate more realistic conditions by introducing variability at the boundaries.

Additionally, the function has an option to return both the noisy boundary conditions and the noise-free version for comparison, controlled by the retNoise flag.

D. Simulation Parameters

The simulation is governed by key parameters, including the time intervals for saving results, the start and end times, the number of grid points, and the frequency of output generation.

The spatial grid is determined by the grid spacing and the coordinates at the edges and centers of the grid cells. The temporal grid is defined by the total number of time steps and the corresponding time intervals, ensuring consistency in time-stepping and output saving.

E. Data Storage

Simulation data is efficiently stored in HDF5 format, providing hierarchical organization for scalable storage. The spatial and temporal coordinates are saved for reproducibility and future analysis.

F. Sample Generation Process

The process of solving a partial differential equation (PDE) involves evolving the system over time using a time-stepping method that ensures numerical stability and accuracy. At each time step, the solution is updated by calculating fluxes and applying boundary conditions, with careful attention to maintaining consistency in the solution's behavior.

Further noise was also incorporated into the system, particularly at the flux calculation stages. This noise changes the waveform of the solution by some degree, ensuring further robustness while simulating physical dynamics. The iterative approach maintains the accuracy of the solution while accommodating both the underlying physics and noise.

G. Visualization

Visualization of the simulation results is achieved by generating animated GIFs, which capture the evolution of the solution over time and present the results in an accessible, visual format.

This work presents an efficient approach for generating noisy Burgers' equation datasets, addressing the challenges of real-world data variability. The integration of noise and boundary conditions enhances the model's robustness for machine learning and numerical simulation applications.

III. Physics-Informed Neural Networks (PINNs)

Physics-Informed Neural Networks (PINNs) integrate the governing equations of the system directly into the loss function, enabling the model to learn physical laws while minimizing the discrepancy between the predicted and true solutions. In this study, the PINN framework was applied to solve the Burgers' equation, incorporating its constraints into the loss function. The loss function consisted of three key components: the residual of the partial differential equation (PDE), the loss due to initial conditions, and the loss associated with the periodic boundary conditions. These components were simultaneously minimized during training to guide the network towards learning the correct solution.

A. Model Architecture

The model used a fully connected neural network (FCNN) with three hidden layers, each containing 50 neurons. The network was implemented using TensorFlow and trained to predict the solution of the Burgers' equation. The neural network architecture was chosen to balance the computational complexity and the capacity to capture the non-linear relationships present in the solution. The loss function used in training had three main components:

- **PDE Residual Loss**: The difference between the network's prediction and the actual PDE residual was calculated, penalizing deviations from the governing equation.
- **Initial Condition Loss**: This term ensured that the predicted solution matched the given initial condition at the starting time.

- **Boundary Condition Loss**: This term ensured the solution adhered to the specified boundary conditions, typically periodic, at the spatial domain boundaries.

By including these three components, the network was simultaneously constrained by the PDE, initial conditions, and boundary conditions, ensuring that the model's predictions adhered to the physical laws governing the problem.

B. Training Process

The PINN model was trained on a noisy dataset over 200 epochs using the Adam optimizer, a widely-used optimization algorithm known for its efficiency in training neural networks. During training, the loss function guided the model to reduce the residual errors of the PDE, as well as the errors from the initial and boundary conditions. The learning rate was dynamically adjusted based on the validation loss to prevent overfitting and ensure steady convergence.

In the prediction process, the trained PINN model was used to predict the solution at future time steps. The model relied on the initial and boundary conditions provided as input, along with the viscosity parameter, to predict the solution iteratively over a specified number of time steps. The initial condition provided the starting point of the simulation, while the boundary conditions maintained the periodic nature of the solution at the domain's edges. The model predicted the solution at each subsequent time step by using the previous time step's solution as input, effectively simulating the evolution of the system in time. This iterative process produced a real-world-like simulation of the Burgers' equation, where the network integrated the effect of viscosity and boundary conditions into the predicted output at each step.

C. Results

However, during the training process, it was observed that incorporating noise into the dataset had a detrimental effect on the performance of the PINN. The noise, introduced via skew-normal and exponential distributions, led to a significant increase in the loss function values, particularly in the PDE residuals. While the model was able to handle the clean data reasonably well, the noisy input caused the model to fail in learning the correct mapping between inputs and outputs. This resulted in poor prediction accuracy, as the network became confused by the noise and could not effectively generalize the physical principles governing the Burgers' equation.

IV. Kolmogorov-Arnold Networks (KANs)

Kolmogorov-Arnold Networks (KANs) are a powerful tool for approximating high-dimensional functions. In this work, KANs were used for denoising a noisy dataset, specifically removing noise from predictions made by PINNs. By leveraging the decomposition of high-dimensional functions into one-dimensional mappings, KANs utilize B-spline interpolation to efficiently approximate complex functions. This modular and flexible approach ensures that the model can capture intricate patterns in the data while handling noise and other irregularities.

A. Model Description

KANs decompose high-dimensional functions into simpler one-dimensional components, utilizing B-splines as the basis functions for interpolation. This structure is consistent with the Kolmogorov-Arnold theorem, which ensures that the network can efficiently approximate any continuous function. The key advantage of KANs is their ability to approximate complex functions with minimal computational overhead. B-splines allow the network to interpolate over the input space, making KANs particularly well-suited for noise removal and function approximation tasks. By modeling the input-output relationship through a sequence of linear transformations and spline interpolations, KANs can reconstruct a denoised version of the predicted solution, improving the accuracy of the output.

B. Model Components

The KAN architecture consists of several key components:

- **Initialization and Grid Setup**: The input domain is represented by a grid, extended by the spline order to ensure smooth transitions at the boundaries. This grid setup facilitates the interpolation process by providing a regular, structured input space. Base weights (`base_weight`) are used for linear transformations, while spline weights (`spline_weight`) represent the B-spline coefficients responsible for interpolating the input data.
- **B-Spline Basis Calculation**: The B-spline basis functions are computed for each input. These functions allow the network to approximate the relationship between input and output in a smooth, continuous manner, making the model highly adaptable to various data distributions.
- **Spline Coefficient Computation**: The spline coefficients are computed using a least-squares fitting process. This ensures that the spline interpolation accurately reflects the data's underlying patterns, enabling the model to make precise predictions even in the presence of noise.
- **Forward Pass**: The forward pass combines a linear transformation of the input, using the base weights, with spline interpolation, using the spline weights, to generate the final output. This combination allows the model to process both the global and local patterns in the data efficiently.
- **Grid Adaptation**: The `update_grid` function allows the network to adapt the spline grid based on the input distribution. This flexibility ensures that the model can dynamically adjust to new data, making it well-suited for tasks involving evolving or noisy datasets.
- **Regularization**: Regularization is applied through a custom loss term that penalizes overly complex spline weights, ensuring that the model does not overfit to the noise in the data. This regularization includes terms for weight sparsity and entropy, promoting smooth approximations that generalize well to unseen data.

To further enhance generalization, dropout layers are integrated into the KAN architecture. These layers are applied between each spline interpolation layer to introduce stochasticity and prevent overfitting. The inclusion of batch normalization and SiLU activation functions further helps to stabilize training and improve convergence.

C. Custom Loss Function

A custom loss function was used during training, combining Mean Squared Error (MSE) and L1 loss to balance precision and robustness. This loss function was designed to accommodate both small errors and outliers, making it well-suited for noisy datasets. The MSE term ensures that the model minimizes the overall error, while the L1 loss term promotes sparsity in the predicted outputs, helping to suppress noise. A weighting factor, denoted as α, was introduced to control the relative contributions of MSE and L1 losses, allowing flexibility in the optimization process depending on the nature of the data. The loss function effectively balances the tradeoff between achieving a low error rate and ensuring robustness to outliers or noise, which is crucial for tasks like denoising predicted solutions.

D. Training

The model was trained using a batch size of 2, with a learning rate scheduler that adjusted the learning rate based on validation performance. Regularization through dropout layers and weight sparsity ensured that the model did not overfit to the noisy data. Weights were saved at each epoch to facilitate subsequent predictions and model evaluation. The custom loss function played a key role in guiding the training process, allowing the model to learn an optimal solution that effectively removed noise while preserving the underlying signal.

In summary, KANs are highly effective for denoising tasks, particularly in the context of predicted solutions from PINNs. Their modular structure, which utilizes B-spline interpolation and a custom loss function, enables them to efficiently remove noise while maintaining the integrity of the underlying data. Regularization terms, dropout layers, and dynamic grid adaptation further enhance the model's ability to generalize and prevent overfitting, ensuring high-quality denoising performance.

V. Results

The PINN made poor approximations to the solutions to Burgers' equation under noisy conditions. The KAN demonstrated superior noise removal, recovering solutions that closely resembled the clean data.

Metric	**Score**
Mean Squared Error (MSE)	2793.0061
Root Mean Squared Error (RMSE)	52.8489
Mean Absolute Error (MAE)	5.2009
Relative Error	45.5094
L2 Norm	756974.6875
Max Absolute Error	4552.9487
R-squared	-7995.6167
Cosine Similarity	0.0745

TABLE I
PINN Performance Metrics on noisy data

Metric	**Score**
Mean Squared Error (MSE)	0.4748
Root Mean Squared Error (RMSE)	0.6890
Mean Absolute Error (MAE)	0.3468
Relative Error	2.4690
L2 Norm	697.2369
Max Absolute Error	15.0493
R-squared	0.3173
Cosine Similarity	0.8085

TABLE II
PINN Performance Metrics on clean data

Metric	**Score**
Mean Squared Error (MSE)	0.20299
Root Mean Squared Error (RMSE)	0.45055
Mean Absolute Error (MAE)	0.31751
Relative Error	0.8568
L2 Norm	647.9067
Max Absolute Error	10.7539
R-squared	0.4188
Cosine Similarity	0.7493

TABLE III
KAN Performance Metrics on denoising

VI. Conclusion and Future Work

This study presents a robust framework for addressing noisy partial differential equation (PDE) solutions using a combination of Physics-Informed Neural Networks (PINNs) and Kolmogorov-Arnold Networks (KANs). By systematically integrating noise into the data generation process and employing machine learning-based noise removal techniques, the proposed pipeline effectively bridges the gap between theoretical modeling and real-world scenarios where data is often imperfect.

We present our results demonstrating how this methodology stands out to be effective by showing significant improvements in denoising accuracy and computational efficiency over standard methods. The synergy of PINNs for solving noisy PDEs and KANs for denoising will lay the base for a solid and scalable approach to tackle complex systems like Burgers' Equation.

Future work could be on the extension of this methodology to higher-dimensional PDEs, more sophisticated models of noise, and applying the framework to other scientific and engineering domains. The promising results so far underpin its ability to enhance the efficiency of data-driven methods for solving noisy physical systems.

Acknowledgment

We acknowledge the creators of PDEBench and efficient-KAN repositories for their foundational work.

References

[1] PDEBench Repository. [Online]. Available: https://github.com/pdebench/PDEBench
[2] Efficient Kolmogorov-Arnold Networks. [Online]. Available: https://github.com/Blealtan/efficient-kan