Machine Learning and Partial Differential Equations

Project Report

Sayantan Sarkar

Mathematics Department
State University of New York at Buffalo

Supervisor: Dr. Gino Biondini

December 13, 2023

1 Introduction

In the vast expanse of human knowledge, where science meets philosophy, there lies a profound truth: our understanding of the universe is perpetually evolving. Just as the ancient philosophers pondered the mysteries of existence, today we stand at the cusp of a new era of discovery, powered by the mighty artificial intelligence. AI and related technologies are not just reshaping our world; they are fundamentally redefining the boundaries of what is possible.

At the heart of this transformative wave is a subdiscipline of AI called, machine learning, a discipline that has emerged as a cornerstone of the modern technological renaissance. Machine learning, in its essence, is the art and science of teaching computers to learn from data, to identify patterns, and make decisions with minimal human intervention. It's a field that harmonizes statistical theory, computer science, and, most importantly, a deep understanding of the problem at hand.

As our society becomes increasingly data-driven, the relevance and impact of machine learning grow exponentially. From healthcare, where predictive models save lives, to environmental science, where they help us understand and combat climate change, the applications are as diverse as they are profound. In every dataset, there lies a story waiting to be told, a hidden pattern waiting to be discovered.

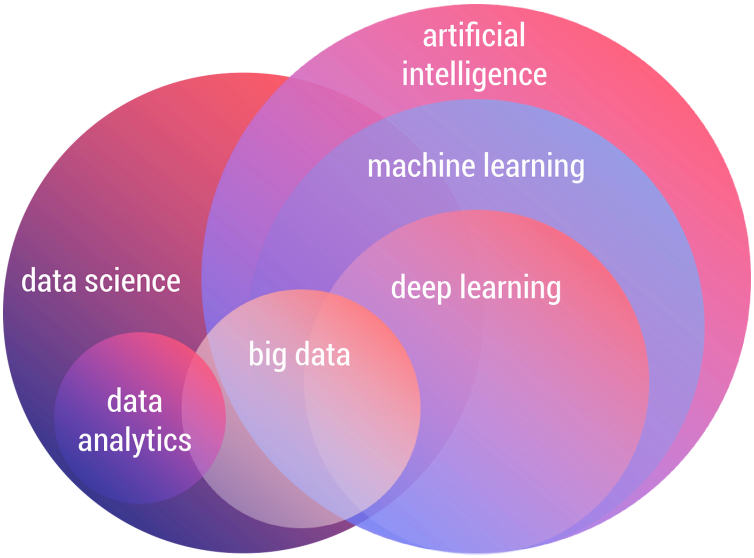
One of the most fascinating intersections in this journey is the realm of partial differential equations (PDEs). These equations are more than just mathematical constructs; they are the language of the physical universe, describing everything from the flow of air over a wing to the ripples in a pond. Traditionally, solving these equations has been a daunting task, requiring sophisticated numerical methods and an in-depth understanding of very advanced mathematics.

Enter the Physics-Informed Neural Networks (PINNs), a novel and exciting frontier in machine learning. PINNs represent a harmonious blend of physics and artificial intelligence, where neural networks are not only informed by data but also guided by the laws of physics. This approach holds the promise of revolutionizing our ability to solve and understand complex PDEs, making it an indispensable tool for scientists and engineers.

The importance of mastering these topics cannot be overstated. As we stand on the shoulders of giants, looking out at a world brimming with data and possibilities, the need to understand and leverage these tools becomes paramount. We are not just passive observers of this revolution; we are active participants, shaping the future with every line of code, every model we build, and every equation we solve.

In conclusion, before we get started on the details of the theory and computation, let us remember the words of the great philosopher Socrates: "*The only true wisdom is in knowing you know nothing.*" It's this humble pursuit of knowledge, this relentless quest to understand, that drives us forward. In learning about machine learning, PDEs, and PINNs, we

are not just acquiring skills; we are partaking in the grand human tradition of pushing the boundaries of what we know, ever striving to unveil the mysteries of the universe.



2 Numerical Approximation of PDE: Finite Difference Method for Burger's Equation

The Burgers' equation is a fundamental partial differential equation in fluid dynamics. For the one-dimensional non linear case with viscosity, it is expressed as:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2} \tag{1}$$

where u represents the velocity field, and ν is the kinematic viscosity. The equation combines effects of advection and diffusion, making it a suitable model for various physical phenomena.

2.1 Finite Difference Method (FDM)

To numerically solve the Burgers' equation, we employ the Finite Difference Method (FDM), discretizing the equation in both space and time.

2.1.1 Discretization

Consider a uniform spatial grid with points x_i where $i = 0, 1, \ldots, N-1$, and $x_i = -1 + i\Delta x$. The time is discretized with steps $t^n = n\Delta t$. The finite difference approximations are:

$$\frac{\partial u}{\partial t} \approx \frac{u_i^{n+1} - u_i^n}{\Delta t} \tag{2}$$

$$\frac{\partial u}{\partial x} \approx \frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x} \tag{3}$$

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} \tag{4}$$

2.1.2 Finite Difference Equation

Substituting these into the Burgers' equation gives the update formula for the velocity field:

$$u_i^{n+1} = u_i^n - u_i^n\frac{\Delta t}{2\Delta x}(u_{i+1}^n - u_{i-1}^n) + \nu\frac{\Delta t}{\Delta x^2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n) \tag{5}$$

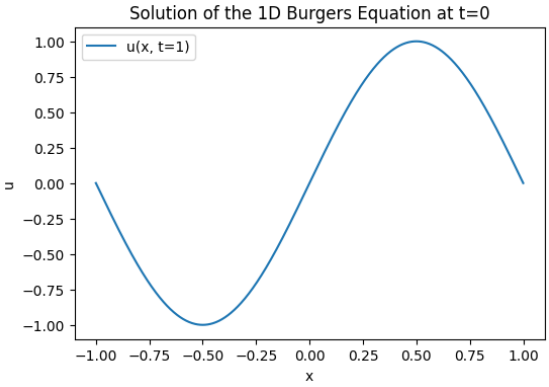
2.1.3 Boundary and Initial Conditions

The boundary conditions are $u(-1, t) = u(1, t) = 0$ and the initial condition is $u(x, 0) = \sin(\pi x)$.

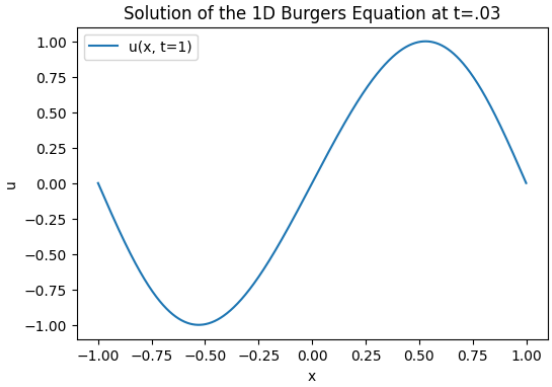
2.1.4 Stability and Accuracy

The stability of the FDM scheme is crucial and depends on the choices of Δx and Δt. A common criterion to ensure stability is the Courant–Friedrichs–Lewy (CFL) condition.

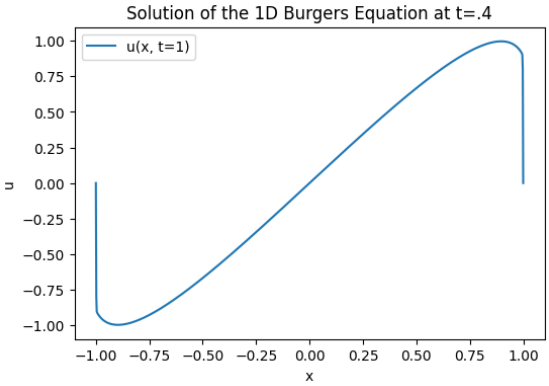
2.2 Simulation



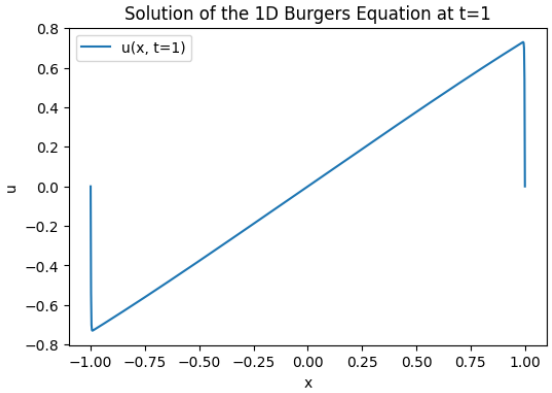
(a) t=0



(b) t=0.03



(a) t=0.4



(b) t=1

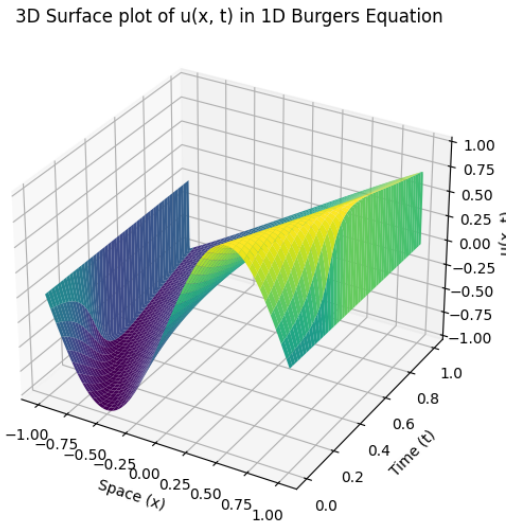
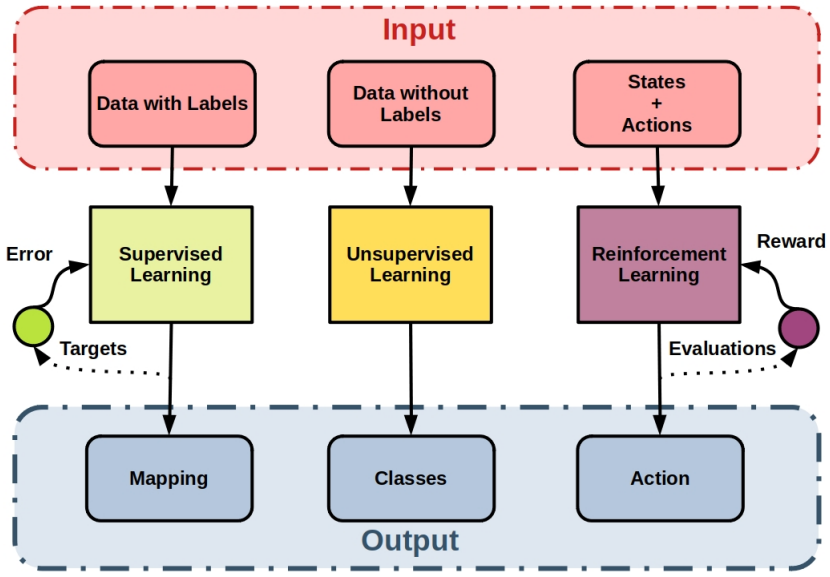


Figure 3: Burger's equation simulation

3 Physics Informed Neural Networks

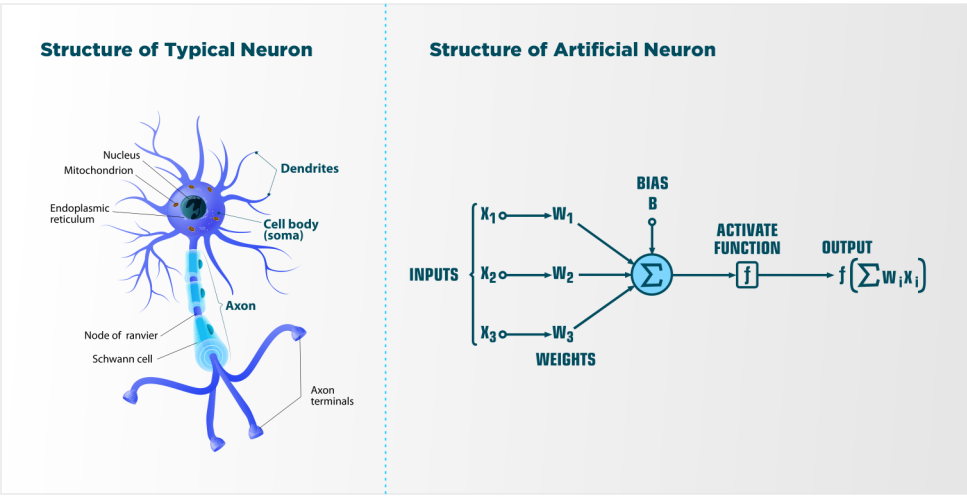
3.1 Machine Learning

Machine learning is a branch of artificial intelligence that focuses on building applications that can learn from and make decisions based on data. Unlike traditional algorithms, machine learning models adjust their parameters automatically to find patterns in data, a process known as training. The two main types of machine learning are supervised learning, where the model learns from labeled data, and unsupervised learning, where the model identifies patterns in unlabeled data.



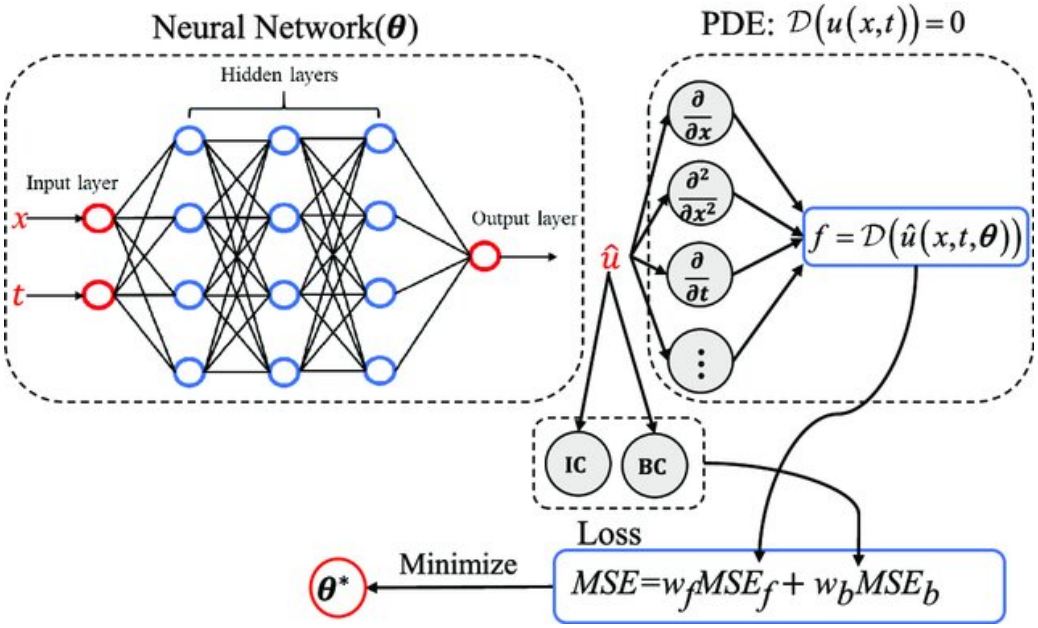
3.2 Fundamentals of Neural Networks

At the heart of many machine learning applications are neural networks, which are computing systems vaguely inspired by the biological neural networks in our brains. A neural network consists of layers of interconnected nodes (neurons), where each connection (synapse) can transmit a signal from one neuron to another. The receiving neuron processes the signal and signals downstream neurons connected to it. The 'learning' in a neural network occurs through the adjustment of synapses based on the data it processes.



3.3 Physics-Informed Neural Networks (PINNs)

Physics-Informed Neural Networks (PINNs) are a novel type of neural network that incorporate physical laws into the learning process. This is achieved by embedding the governing physical equations (like differential equations) into the structure of the neural network. The idea is to leverage the power of neural networks in approximating complex functions while ensuring that the predictions adhere to established physical principles.



3.4 Why PINNs?

The integration of physical laws in neural networks offers several advantages:

- Data Efficiency: PINNs can make accurate predictions even with limited data, as they leverage known physics laws.

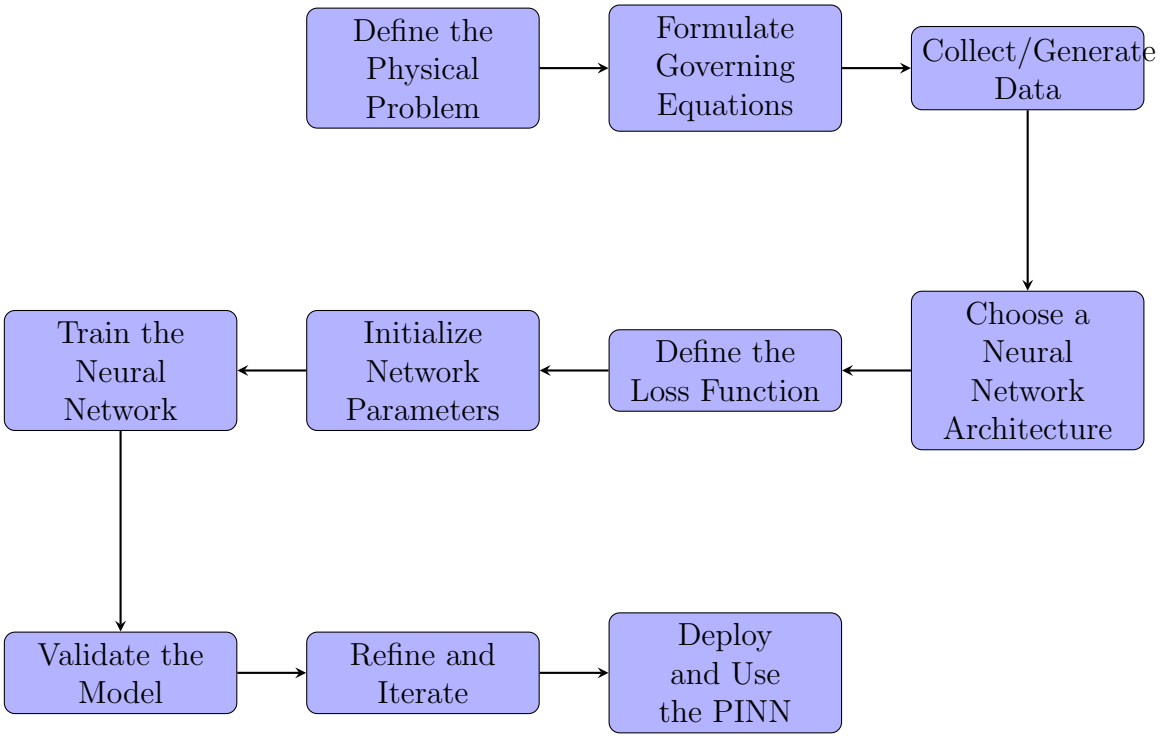
- Improved Accuracy: Incorporating physical laws can improve the accuracy and reliability of predictions, especially in scenarios where data is noisy or sparse.
- Interdisciplinary Applications: PINNs are particularly useful in scenarios where data-driven approaches need to be combined with physical modeling, like in fluid dynamics, material science, and climate modeling.

3.5 Mathematical Foundations of PINNs

A key aspect of PINNs is the incorporation of differential equations into the learning process. For example, consider the heat equation, a partial differential equation that describes the distribution of heat (or variation in temperature) in a given region over time. In a PINN, the neural network is trained to solve this equation by minimizing a loss function that encapsulates the difference between the network's predictions and the known solutions to the equation.

3.6 Implementing PINNs

In practice, implementing a PINN involves defining a neural network that takes inputs (like spatial coordinates and time) and outputs quantities of interest (like temperature in the case of the heat equation). The network is trained not just to fit the available data but also to satisfy the physical equations represented in its architecture. Here is a flowchart to simplify the implementation of PINNs-



3.7 Conclusion

Physics-Informed Neural Networks represent a significant step forward in the integration of machine learning with physical modeling. By embedding physical laws into the architecture of neural networks, PINNs offer a powerful tool for solving complex problems in science and engineering where data is limited or noisy.

4 Mathematical Framework for PINNs

Physics-Informed Neural Networks (PINNs) integrate the rigor of physical laws with the flexibility of machine learning. Now we delve into the mathematical framework underlying PINNs.

A neural network is a function approximator, mathematically expressed as:

$$F(\mathbf{x}; \mathbf{W}, \mathbf{b}) = \sigma(\mathbf{W} \cdot \mathbf{x} + \mathbf{b}) \tag{6}$$

where $\mathbf{x}$ is the input, $\mathbf{W}$ and $\mathbf{b}$ are the weights and biases of the network, and σ is a non-linear activation function. For multiple layers, this function is composed iteratively.

4.1 Embedding Physics in Neural Networks

In PINNs, the neural network learns to satisfy both the data and the physical laws. The physical laws are generally expressed in the form of differential equations, for example: we consider a differential equation defined on the region $\Omega = [L_1, L_2] \times [t_0, t_f]$ as:

$$\begin{aligned} \mathcal{D}(u(\boldsymbol{x})) &= \mathbf{0}, \quad \boldsymbol{x} \in \Omega, \\ \mathcal{B}(u(\boldsymbol{x})) &= \mathbf{0}, \quad \boldsymbol{x} \in \partial\Omega, \end{aligned} \tag{7}$$

where $\boldsymbol{x}$ is a vector of variables x and t, $u = u(\boldsymbol{x})$ is the exact solution, $\mathcal{D}$ is the differential operators extracted from the differential equation and $\mathcal{B}$ is a set of the differential operators extracted from the initial and boundary conditions and $\partial\Omega$ denotes the generalized boundary of Ω.
The goal is to approximate u using a neural network.

4.2 Neural Network Approximation

The neural network $\hat{u}(\boldsymbol{x}, \boldsymbol{\theta})$ approximates $u(x)$, structured as:

$$\begin{aligned} &\text{Input layer:} \quad \hat{\boldsymbol{u}}^{(0)} = \boldsymbol{x} \\ &\text{Hidden layer:} \quad \hat{\boldsymbol{u}}^{(\ell)} = \sigma\left(\boldsymbol{W}^{(\ell)}\boldsymbol{u}^{(\ell-1)} + \boldsymbol{b}^{(\ell)}\right), \quad \ell = 1, 2, \ldots, L, \\ &\text{Output layer:} \quad \hat{u}(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma\left(\boldsymbol{W}^{(L+1)}\boldsymbol{u}^{(L)} + \boldsymbol{b}^{(L+1)}\right), \end{aligned} \tag{8}$$

with $\boldsymbol{\theta} = \left\{\boldsymbol{W}^{(\ell)}, \boldsymbol{b}^{(\ell)}\right\}_{1 \le \ell \le L+1}$ and activation function σ.

4.2.1 Input Layer

The input layer receives the input vector $\boldsymbol{x}$ and passes it to the next layer:

$$\hat{\boldsymbol{u}}^{(0)} = \boldsymbol{x} \tag{9}$$

-this typically include spatial coordinates and time.

4.2.2 Hidden Layers

The network comprises one or more hidden layers. Each layer transforms its input through linear and non-linear operations:

$$\hat{\boldsymbol{u}}^{(\ell)} = \sigma\left(\boldsymbol{W}^{(\ell)}\boldsymbol{u}^{(\ell-1)} + \boldsymbol{b}^{(\ell)}\right), \quad \ell = 1, 2, \ldots, L \tag{10}$$

where L is the total number of hidden layers, $\boldsymbol{W}^{(\ell)}$ and $\boldsymbol{b}^{(\ell)}$ are the weights and biases of the ℓ-th layer, and σ is the activation function.

4.2.3 Output Layer

The final layer of the network produces the approximation of the solution $\hat{u}(\boldsymbol{x}, \boldsymbol{\theta})$:

$$\hat{u}(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma\left(\boldsymbol{W}^{(L+1)}\boldsymbol{u}^{(L)} + \boldsymbol{b}^{(L+1)}\right) \tag{11}$$

with $\boldsymbol{\theta}$ representing the collective set of all network parameters (weights and biases).

This structure allows the neural network to approximate complex functions, including solutions to differential equations, as required in PINNs.

4.3 Physics-Informed Loss Function

The physics-informed loss function in PINNs is essential for ensuring that the neural network adheres to physical laws. It is defined as a weighted sum of two components: MSE_f and MSE_b.

4.3.1 Mean Squared Error for the Field ($\mathbf{MSE}_f$)

This component represents the mean squared error over the domain Ω, defined as:

$$MSE_f = \frac{1}{|\mathcal{T}_f|} \sum_{\boldsymbol{x} \in \mathcal{T}_f \subset \Omega} \|\mathcal{D}(\hat{u}(\boldsymbol{x}, \boldsymbol{\theta}))\|_2^2 \tag{12}$$

where $\mathcal{T}_f$ is a set of sample points within Ω, and $|\mathcal{T}_f|$ denotes the number of these points.

4.3.2 Mean Squared Error for the Boundary ($\mathbf{MSE}_b$)

This component calculates the mean squared error over the boundary $\partial\Omega$:

$$MSE_b = \frac{1}{|\mathcal{T}_b|} \sum_{\boldsymbol{x} \in \mathcal{T}_b \subset \partial\Omega} \|\mathcal{B}(\hat{u}(\boldsymbol{x}, \boldsymbol{\theta}))\|_2^2 \tag{13}$$

Here, $\mathcal{T}_b$ represents a set of sample points on $\partial\Omega$, with $|\mathcal{T}_b|$ being its size.

The total Mean Squared Error (MSE) combines these two components:

$$MSE = w_f MSE_f + w_b MSE_b \tag{14}$$

where w_f and w_b are the weights for MSE_f and MSE_b respectively.

4.4 Training PINNs/Optimization

The optimal neural network parameters $\boldsymbol{\theta}^*$ are obtained by minimizing the loss function:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} MSE, \tag{15}$$

resulting in the approximation $u(\boldsymbol{x}) \approx \hat{u}\left(\boldsymbol{x}, \boldsymbol{\theta}^*\right)$.

4.5 Hyperparameter Considerations

Hyperparameters such as the network's depth and width, learning rates, and iteration steps are crucial for the accuracy and efficiency of the learning process. They are typically determined through a combination of theoretical understanding and empirical testing.

4.6 Challenges and Solutions

- **Balancing Data and Physics:** Adjusting λ is critical for balancing the influence of data and physical laws.
- **Complex Differential Operators:** Advanced techniques like automatic differentiation can be used for complex $\mathcal{L}$.

4.7 Applications

PINNs have found applications in various fields where differential equations play a crucial role, such as fluid dynamics, material science, and climate modeling.

The integration of physical laws into neural networks through PINNs opens up new possibilities for solving complex problems in science and engineering. The mathematical framework of PINNs allows for the rigorous enforcement of physical principles while leveraging the power of machine learning for function approximation.

5 Case Study:

In this chapter we see the principle in action and look for different cases where we use PINN to solve differential equations.

5.1 Burger's equation with PINNs

This process consists of a multi-layer architecture with Tanh activation functions. Key parameters include spatial and temporal step sizes, the number of neurons in each layer, and the configuration of the optimizers. The training involves iterative optimization using both Adam and L-BFGS methods, guided by a custom loss function that ensures adherence to PDE constraints.

5.1.1 Neural Network Structure

- The network, class `NN`, includes multiple linear layers with Tanh activation functions.
- It expands its input from 2 features to 20 in the first layer, processes through several hidden layers, and outputs a single feature.

5.1.2 Net Class

- Encapsulates the `NN` model, training data, and optimization process.
- Manages computational domain setup, initial and boundary conditions.
- Tensors are configured for appropriate device (CPU or GPU) compatibility.

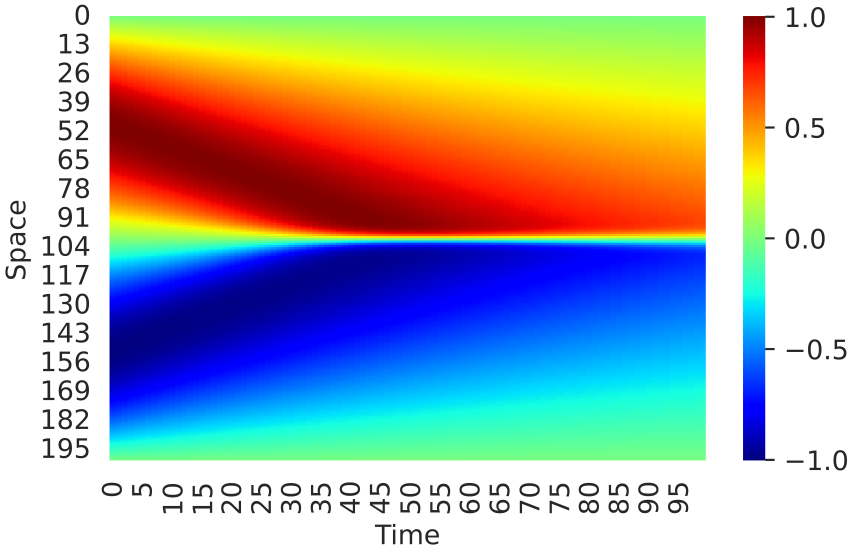
5.1.3 Parameters and Setup

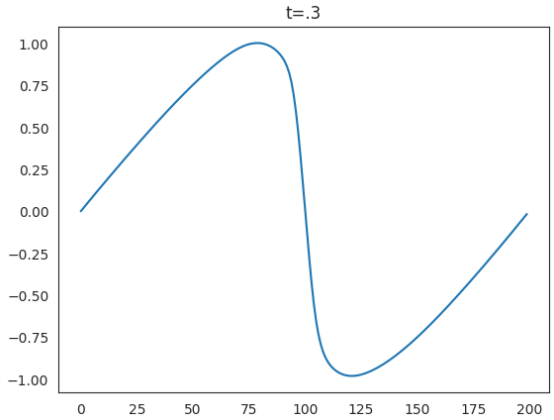
- Spatial and temporal step sizes are set, with a default of 0.1 (modifiable as per requirement).
- The spatial-temporal domain is defined, creating a computation grid.
- Boundary and initial conditions are specified and organized into tensors.
- The model is instantiated and configured for the computational device.
- Adam and L-BFGS optimizers are used for training.

5.1.4 Training and Evaluation

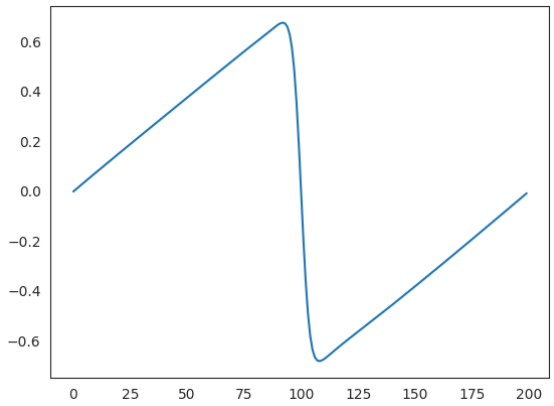
- The custom loss function combines data fidelity and adherence to PDEs.
- Training involves iterative optimization using Adam and L-BFGS methods.
- The model periodically prints loss for monitoring and is evaluated in the end for performance.

5.2 Simulation:





(a) t=0.4



(b) t=1

6 Conclusion

This investigation clearly demonstrates the significant potential and efficacy of neural networks within the realm of mathematical applications. As highlighted through our case study, neural networks not only enhance computational efficiency but also improve accuracy with fewer steps. This underscores their vital role as transformative tools in contemporary mathematical research and problem-solving.

Acknowledgements

I extend my heartfelt thanks to Dr. Gino Biondini for not only providing me with the opportunity to delve into this fascinating project but also for his invaluable guidance and support on countless matters. His mentorship has been instrumental both in my role as a student in his class and as an advisee.

I would also like to express my deep appreciation to my other advisor, Dr. Daozhi Han, for his assistance and unwavering support throughout this journey.

Additionally, my profound gratitude goes to my wife, parents and in-laws, whose care, love, and encouragement from near and from far have been the bedrock of my strength and perseverance during this endeavor.

References

[1] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis, *Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations*. arXiv:1711.10561v1 [cs.AI] 28 Nov 2017.

[2] Stefan Kollmannsberger, Davide D'Angella, Moritz Jokeit, Leon Herrmann *Deep Learning in Computational Mechanics An Introductory Course*. Studies in Computational Intelligence Volume 977, Springer.