Machine Learning and Partial Differential Equations

Sayantan Sarkar

State University of New York at Buffalo

December 13, 2023

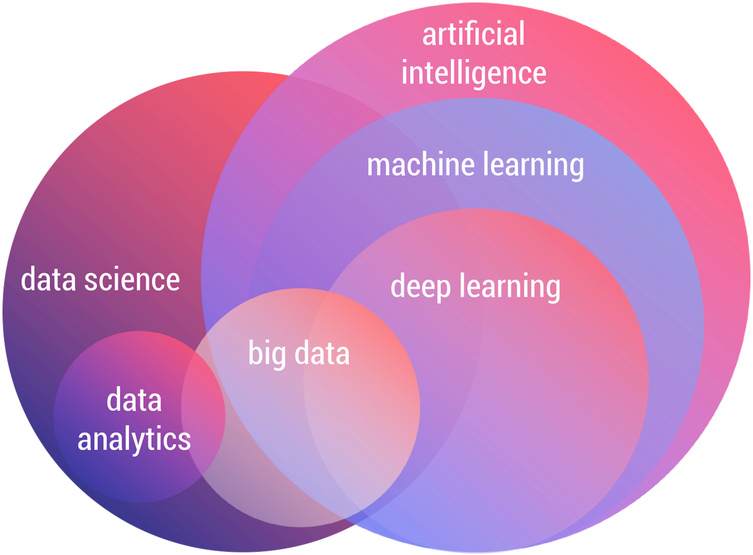
Outline

1. Introduction
2. Numerical Methods (FDM)
3. ML and Neural Network
4. Mathematical framework for PINNs
5. Case-study:Burger's Equation:

Introduction

- 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.
- Machine learning, at the heart of AI 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.
- One of the most fascinating application of ML is in the realm of partial differential equations (PDEs).
- Physics-Informed Neural Networks (PINNs), is 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.

Some common terms



Ways to solve a PDE

- Analytical
- Approximate
  - Numerical Methods (like, FDM, FVM, FEM and spectral method)
  - Data- driven (Physics Informed Neural Network)

Numerical Methods

We have taken Burger's equation

$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}=\frac{\partial^2 u}{\partial x^2} \text{ with } u(-1,t)=u(1,t)=0 \text{ and } u(x,0)=\sin(\pi x). \quad (1)$$

FDM

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:

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

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

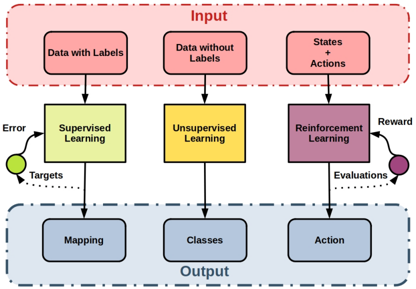
$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} \quad (4)$$

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

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

Machine Learning

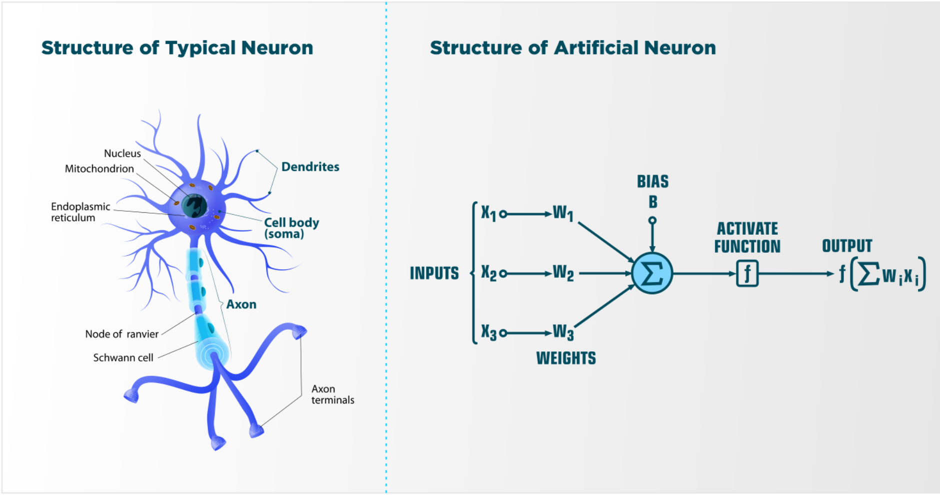
Machine learning is a branch of artificial intelligence that focuses on building applications that can learn from and make decisions based on data. Unlike traditional algorithms, machine learning models adjust their parameters automatically to find patterns in data, a process known as training.



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.

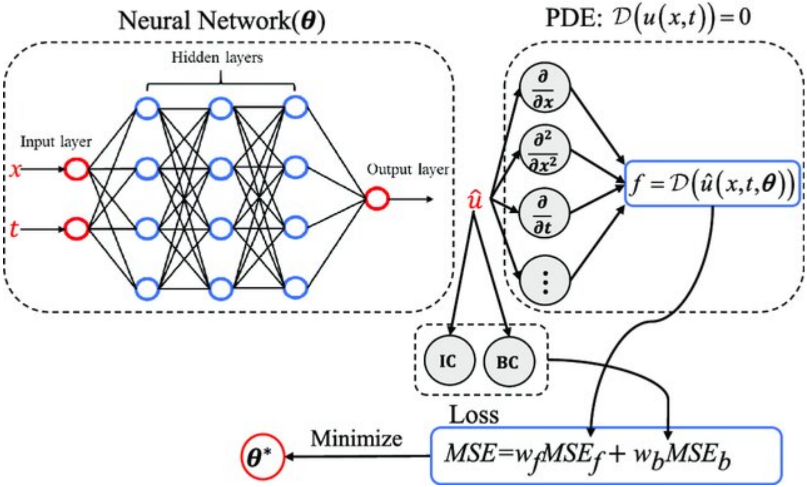
Neural Networks



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.

PINNs


Neural Network($\boldsymbol{\theta}$)
Hidden layers
Input layer
x
t
Output layer
$\hat{u}$
PDE: $\mathcal{D}(u(x,t))=0$
$\frac{\partial}{\partial x}$
$\frac{\partial^2}{\partial x^2}$
$\frac{\partial}{\partial t}$
$\vdots$
$f=\mathcal{D}(\hat{u}(x,t,\boldsymbol{\theta}))$
IC
BC
Loss
$MSE=w_f MSE_f + w_b MSE_b$
Minimize
$\boldsymbol{\theta}^*$

Why PINNs?

- 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.

Approximator

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 **x** is the input, **W** and **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.

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.

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

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.

Mean Squared Error for the Field (MSE$_f$)

$$MSE_f = \frac{1}{|\mathcal{T}_f|} \sum_{\boldsymbol{x} \in \mathcal{T}_f \subset \Omega} \|\mathcal{D}(\hat{u}(\boldsymbol{x}, \boldsymbol{\theta}))\|_2^2 \tag{9}$$

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

Mean Squared Error for the Boundary (MSE$_b$)

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

$$MSE_b = \frac{1}{|\mathcal{T}_b|} \sum_{\boldsymbol{x} \in \mathcal{T}_b \subset \partial\Omega} \|\mathcal{B}(\hat{u}(\boldsymbol{x}, \boldsymbol{\theta}))\|_2^2 \tag{10}$$

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

MSE

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

$$MSE = w_f MSE_f + w_b MSE_b \tag{11}$$

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

PINN for Burger's Equation

- This process consists of a multi-layer architecture with Tanh activation functions. Key pa- rameters 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.

Simulation: google colab.