Physics Informed Neural Networks: A New Approach to Scientific Computing

Hasif Ahmed

October 11, 2023

Abstract

Physics Informed Neural Networks (PINNs) are a novel approach that combines the power of deep learning and physical models to solve problems in science and engineering that are challenging for traditional solvers. This report provides a comprehensive overview of the mathematical foundations of neural networks and demonstrates how PINNs can be applied to a variety of problems in science. We provide a detailed analysis of the strengths and limitations of PINNs and compare them with traditional solvers.

1 Introduction

The advancement in artificial intelligence and machine learning has opened up new avenues for solving complex problems in science and engineering. Traditional solvers, based on numerical methods and physical models, have limitations in terms of computational efficiency and accuracy, especially for high-dimensional and nonlinear problems. Physics Informed Neural Networks (PINNs) are a novel approach that leverages the power of deep learning to improve the solution of such problems. Recent breakthroughs in machine learning, spurred by vast data and computational power, have impacted various fields like image recognition, natural language processing, cognitive science, and genomics. However, many analysis tasks, especially in physical, biological, or engineering systems, grapple with limited data, making data acquisition costly. In such scenarios, conventional machine learning models, like

deep or convolutional neural networks, often lack robustness and don't guarantee convergence. Training such models to understand high-dimensional input-output pairs with limited data seems impractical. However, when analyzing physical or biological systems, there's significant prior knowledge not utilized in today's machine learning. Whether it's consistent physical laws, empirical rules, or domain expertise, this knowledge can regularize and guide models, e.g., by eliminating non-realistic solutions in fluid dynamics that defy mass conservation. Integrating this structured knowledge into algorithms enhances the data's informational value, allowing models to hone in on correct solutions even with minimal training data. This report aims to provide a comprehensive overview of the mathematical foundations of neural networks, explain the concept of PINNs, and demonstrate their applicability to a variety of scientific problems.

2 Mathematical Foundations of Neural Networks

Neural networks are a type of machine learning model inspired by the human brain. A neural network consists of layers of interconnected nodes or neurons. Each connection between neurons has an associated weight, and each neuron has an associated bias. The output of each neuron is computed as a weighted sum of its inputs, passed through a non-linear activation function.

2.1 Forward Propagation

Forward propagation is the process of passing the input data through the network, layer by layer, until the output layer is reached. The output of each neuron in a layer is computed as follows:

$$z_i = \sum_{j=1}^{n} w_{ij} x_j + b_i \tag{1}$$

$$a_i = f(z_i) \tag{2}$$

where x_j is the input to the neuron, w_{ij} is the weight of the connection between the i-th neuron and the j-th input, b_i is the bias of the i-th neuron, z_i is the weighted sum of the inputs, a_i is the output of the neuron, and f is the activation function.

2.2 Backward Propagation

Backward propagation is the process of updating the weights and biases of the network to minimize the difference between the predicted output and the actual target values. The error for a single output neuron is given by:

$$E = \frac{1}{2}(t - a)^2 \tag{3}$$

where t is the target value and a is the output of the neuron.

3 Physics Informed Neural Networks (PINNs)

Physics Informed Neural Networks (PINNs) are a type of neural network that leverages the power of deep learning and the knowledge of the underlying physics of a problem to obtain more accurate and physically consistent solutions. PINNs are designed to satisfy the governing differential equations of a physical system, in addition to fitting the available data.

3.1 Formulation of PINNs

For a given physical system, let's consider it to be governed by a PDE of the form:

$$\mathcal{L}(u; x) = 0, \quad x \in \Omega \tag{4}$$

Here, $\mathcal{L}$ denotes the differential operator, u represents the unknown function we aim to solve for, and Ω stands for the domain in which the PDE is defined.

In the realm of PINNs, rather than seeking an analytical solution for u, we aim to approximate this solution using a neural network, which we'll denote as u_{NN} with associated parameters θ. The architecture of the neural network can be chosen based on the problem's complexity, with deeper networks being used for more intricate problems.

To train the neural network to be consistent with the physical system, a composite loss function is formulated as:

$$\mathcal{L}(\theta) = \mathcal{L}_{data}(\theta) + \lambda \mathcal{L}_{PDE}(\theta) \tag{5}$$

The term $\mathcal{L}_{data}$ denotes the data-driven loss[1], quantifying the discrepancies between the neural network's predictions, u_{NN}, and any available observational data. On the other hand, $\mathcal{L}_{PDE}$, often referred to as the 'Physics Loss'

in literature, encapsulates the error between the neural network's output when plugged into the PDE, i.e., $\mathcal{L}(u_{NN}; x)$, and the known ground truth (which is zero, based on the PDE definition). The hyperparameter λ plays a pivotal role in weighting the importance between data fidelity and physical consistency.

To begin, let's reiterate our objective: we are looking to train a neural network to minimize a loss function $\mathcal{L}(\theta)$, which encapsulates both data fidelity and physical consistency. The optimization of this loss function can be represented as:

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \mathcal{L}(\theta) \tag{6}$$

Where θ^* is the optimal set of parameters for our neural network.

Optimization: A plethora of optimization techniques can be employed, with gradient-based methods being particularly popular due to their efficiency in high-dimensional spaces. The backpropagation algorithm, facilitated by automatic differentiation, computes gradients of the loss function with respect to the neural network parameters. Common gradient-based optimizers include:

1. Stochastic Gradient Descent (SGD)
2. Adam
3. RMSprop

In the context of PINNs:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}(\theta_t) \tag{7}$$

Where η is the learning rate, and $\nabla_\theta \mathcal{L}(\theta_t)$ is the gradient of the loss with respect to parameters at iteration t.

The primary advantage of PINNs lies in their ability to generalize well even with sparse or noisy data, leveraging the embedded physical knowledge to guide the learning process. The method has gained traction due to its unique combination of data-driven learning with physics-based constraints.

4 Applications of PINNs

PINNs have been successfully applied to a wide range of problems in science and engineering, including fluid dynamics, heat transfer, and solid mechanics. In this section, we will present a few representative examples.

4.1 Fluid Dynamics

Consider the Navier-Stokes equations, which describe the motion of a fluid:

$$\rho\left(\frac{\partial u}{\partial t} + u \cdot \nabla u\right) = -\nabla p + \mu \nabla^2 u + f \tag{8}$$

$$\nabla \cdot u = 0 \tag{9}$$

where u is the velocity field, p is the pressure, ρ is the density, μ is the dynamic viscosity, and f is the body force.

We implement the PINN solution to the 2D Navier-Stokes problem using the code and plot the pressure field here.

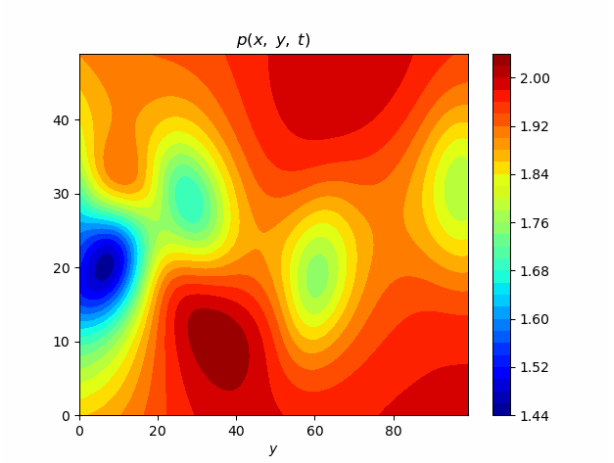

Figure 1: PINN approximation to the pressure field for Navier Stokes Equation

The neural network is trained to approximate the solution of these equa-

tions by using:

$$\psi(x, y, t) : \text{Stream function from the neural network.}$$
$$p(x, y, t) : \text{Pressure from the neural network.}$$

The velocity fields $\bar{u}$ and $\bar{v}$ are derived from the stream function:

$$u = \frac{\partial \psi}{\partial y}$$
$$v = -\frac{\partial \psi}{\partial x}$$

The residual (or physics-informed) loss components, represented as $\bar{f}$ and $\bar{g}$ in the code, ensure that the network's outputs satisfy the Navier-Stokes equations. They are given by:

$$f = u_t + uu_x + vu_y + p_x - \nu(u_{xx} + u_{yy})$$
$$g = v_t + uv_x + vv_y + p_y - \nu(v_{xx} + v_{yy})$$

Where subscripts denote partial derivatives. The total loss function that the neural network minimizes is a combination of the data mismatch (for $\bar{u}$ and $\bar{v}$) and the residuals (for $\bar{f}$ and $\bar{g}$).

$$\mathcal{L}(\theta) = \mathcal{L}_u(\theta) + \mathcal{L}_v(\theta) + \mathcal{L}_f(\theta) + \mathcal{L}_g(\theta)$$

Where:

$\mathcal{L}u$: MSE loss between predicted and true values of u.
$\mathcal{L}v$: MSE loss between predicted and true values of v.
$\mathcal{L}f$: MSE loss between f and 0 (since it should ideally be 0).
$\mathcal{L}g$: MSE loss between g and 0.

PINNs can be used to solve these equations by minimizing a loss function that consists of three terms: the data loss, the PDE loss, and the divergence-free loss, which ensures that the predicted velocity field satisfies the continuity equation.

4.2 Heat Diffusion

We aim to solve the 1D heat equation using Physics-Informed Neural Networks. The mathematical model we are considering is:

The 1D heat equation is given by:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \tag{10}$$

Initial and Boundary Conditions

The initial condition is:

$$u(x, 0) = \sin(\pi x) \tag{11}$$

And the boundary conditions are:

$$u(0, t) = 0 \tag{12}$$

$$u(1, t) = 0 \tag{13}$$

Neural Network Architecture The neural network used is a simple feed-forward network with one hidden layer. The network takes two inputs (x and t) and outputs the predicted solution $u(x, t)$. Activation function used is the hyperbolic tangent (tanh).

Loss Function The loss function is constructed from three parts:

- The PDE loss: This is the residual of the heat equation, given by:

$$L_{\text{PDE}} = \left(\frac{\partial u}{\partial t} - \frac{\partial^2 u}{\partial x^2} \right)^2 \tag{14}$$

- The boundary conditions loss: Enforcing the values of u at the boundaries to be zero.

$$L_{\text{BC}} = u^2(0, t) + u^2(1, t) \tag{15}$$

- The initial condition loss: Enforcing the neural network to match the given initial condition.

$$L_{\text{IC}} = \left(u(x, 0) - \sin(\pi x) \right)^2 \tag{16}$$

The total loss function is the sum of the three losses:

$$L_{\text{total}} = L_{\text{PDE}} + L_{\text{BC}} + L_{\text{IC}} \tag{17}$$

Training Procedure Random samples from the domain are taken, and the loss function is minimized using the Adam optimizer. After training, the solution is compared to the analytical solution of the heat equation.

Results The final plot shows the comparison of the neural network prediction at various training steps against the analytical solution. The analytical solution at a given time t is:

$$u_{\text{true}}(x,t) = e^{-\pi^2 t}\sin(\pi x) \tag{18}$$

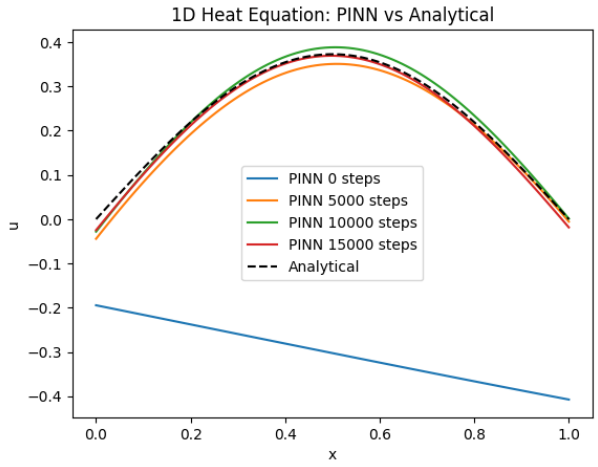

Figure 2: PINN approximation and exact solution to the 1d heat diffusion equation

The convergence of the neural network solution towards the analytical solution is evident with increasing training steps. The python script for this PINN can be found here.

5 Conclusion

Physics Informed Neural Networks (PINNs) provide a novel approach for solving complex problems in science and engineering that are challenging for traditional solvers. By incorporating the knowledge of the underlying physics into the neural network, PINNs can obtain more accurate and physically consistent solutions. This report provides a comprehensive overview of the mathematical foundations of neural networks, explains the concept of PINNs, and demonstrates their applicability to a variety of scientific problems.

6 References

1. 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.