Physics-Informed Neural Networks for Solving Forward and Inverse Problems Involving Differential Equations

Qiuhong (Anna) Wei

October 25, 2020

1 Introduction

In recent years, with explosive growth of data and computational power, machine learning has yielded transformative results in various scientific disciplines, such as computer vision [1] and natural language processing [8]. When analyzing and solving complex physical systems, machine learning algorithms can typically extract information content from the data effectively. However, there often exists great prior information underlying scientific data—physical laws modelled by partial differential equations (PDEs)—that are commonly omitted by the modern data-driven machine learning approach.

To harness the power of both the machine learning practice and the underlying prior information, physics informed neural networks (PINNs) concurrently utilize physics and data to solve scientific problems, significantly decreasing the demand for data in the training process. Leading up to the development of PINNs, scientists took various different approaches to solve physical problems. A variety of traditional numerical methods are well established and studied for solving differential equations, such as the finite difference method, the finite element method, and spectral methods. Recently, scientists have attempted to construct machine learning algorithms to solve both linear [2, 4, 5] and nonlinear problems [3, 6]. However, those machine learning approaches were limited in applicability because of their use of assumptions, linearization, or local time-stepping methods. PINNs circumvent these limitations by leveraging deep neural networks. By encoding physical laws in the form of PDEs into deep neural networks, PINNs form a class of learning algorithms that is data-efficient and physics-informed, with powerful capacity for tackling problems in computational science and for modeling the world around us.

In order to apply the method of PINNs to a partial differential equation, we will first start with the forward and inverse problems of a simpler one-dimensional ordinary differential equation (ODE) to consolidate understanding and experiment with implementation of the neural network in code. We will validate the models we obtain for the ODE problems before extending the methodology to a more complex two-dimensional partial differential equation. Our ultimate goal is to understand through implementation how PINNs can be employed to solve forward and inverse problems involving differential equations.
2 Background

A partial differential equation is an equation involving derivatives of an unknown function \( u : \Omega \to \mathbb{R} \), where \( \Omega \) is an open subset of \( \mathbb{R}^d \) and \( d \geq 2 \). With more than one independent variables, \( u \) is either dependent on both temporal and spatial variables or independent of time but dependent on multiple spatial variables. Let us represent \( u \) by the general form \( u(t, x) \) where \( t \) represents time and \( x \) represents a spatial variable.

For a PDE to have a constrained solution space, both its spatial and temporal domains need to be constrained. Let us consider a complete PDE problem of the following general form

\[
\begin{align*}
  u_t + N[u; \lambda] &= 0, \quad t \in [0, T], \quad x \in \Omega \\
  u(t, x) &= \bar{u}, \quad t \in [0, T], \quad x \in \partial \Omega \\
  u(t, x) &= g(x), \quad t = 0, \quad x \in \Omega
\end{align*}
\]

where \( u(t, x) \) is the latent solution, \( N[\cdot; \lambda] \) is a nonlinear differential operator parametrized by \( \lambda \), and \( \partial \Omega \) is the boundary of \( \Omega \).

(1.1) is the general form of a parametrized and nonlinear partial differential equation. Boundary condition (1.2) imposes constraint on the spatial domain and initial condition (1.3) imposes constraint on the system state at \( t = 0 \). A complete PDE problem is defined with all three: the partial differential equation, the relevant boundary condition, and the relevant initial condition.

There are two types of problems related to such PDEs: forward problems and inverse problems. The forward problems ask given fixed model parameters \( \lambda \), what is the hidden solution of the system \( u(t, x) \)? The inverse problems ask given observed data of the solution \( u(t, x) \), what are the model parameters \( \lambda \)? To solve these two types of problems, we define residual \( f(t, x) \) to be the left-hand side of (1.1)

\[
f(t, x) := u_t + N[u; \lambda]
\]

A physics informed neural network has 2 components: the neural network component that approximates \( u \) from inputs \( (t, x) \) using a deep neural network, and the PDE that makes use of automatic differentiation to differentiate the neural network with respect to the input coordinates and model parameters to calculate the residual \( f \). Both the output from the neural network component \( u \) and the PDE component \( f \) are integrated into the loss function for the PINN

\[
MSE = MSE_u + MSE_f
\]

where \( MSE_u \) is computed by comparing \( u \) to the values specified by given conditions or data for particular input, and \( MSE_f \) is computed by making use of (1.1). More specifically, \( MSE_u \) enforces the initial and boundary conditions for input on the spatial boundary or at \( t = 0 \). For inverse problems, \( MSE_u \) also enforces the given observed data of \( u(t, x) \). On the
other hand, $MSE_f$ enforces the PDE for a set of collocation points.

With such a set up, if the given PDE is well-posed, the neural network architecture is sufficiently complex, and the number of collocation points is adequate, the PINN is capable of encoding any physical laws and achieving great prediction accuracy.

3 Proposed Methodology

We plan to first apply the physics informed neural network framework to solve the forward and inverse problems of a one-dimensional ordinary differential equation with analytical solution

$$u''(x) = f(x), \quad x \in [a, b]$$
$$u(a) = 0, \quad u'(b) = 1$$

where $f(x)$ is a fully known function for the forward problem and is unknown for the inverse problem for which it can be represented as $f(x, \lambda)$ with unknown $\lambda$. We will decide on the exact form of $f(x)$ later on. Some examples of $f(x)$ include $f(x) = 0, f(x) = x, f(x) = \sin(x)$.

To solve this ODE problem, we plan to implement the neural network in Python, most likely utilizing the Tensorflow framework. Throughout the implementation process, we will reference the problem set up and code in [7]. Afterwards, we would validate the models obtained by comparing their results against the analytical solution’s output to calculate the model’s predicative accuracy.

After we achieve reasonable accuracy with the solutions to the one-dimensional ordinary differential equation problems, we plan to extend the same methodology and implementation to a two-dimensional partial differential equation. Similarly, in order to effectively evaluate the models to be obtained, we would implement the PINN for a problem with known analytical solution or existing methods for generating benchmark test data. We aim to achieve satisfactory accuracy for our solution models for both the forward and inverse problems of the PDE.

References


