Physics-informed machine learning

Emmanuel Skoufris

University of Queensland

28 June 2024

Introduction

- Partial differential equations (PDEs) play an important role in physics and engineering as they represent physical laws that systems must obey as they evolve over time
- A PDE relates a function $u : \mathbb{R}^n \to \mathbb{R}$ to its partial derivatives via some equation.
  - Example: $u_x + uu_t + t^2 = 0$.
- There are traditionally two approaches to solving for u :
  1. Classical methods that seek exact solutions.
     - Separation of variables, method of characteristics, Fourier series, Green's functions.

     However, many famous PDEs in maths and physics do not admit any known exact solutions.
  2. Numerical methods that discretize the domain and then use analytical techniques (like Taylor's theorem) to approximate the solution at finitely many points.
     - Finite differences, finite element method etc.

A New Paradigm

- Deep neural networks are known for their ability to approximate continuous functions, as a result of numerous Universal Approximation Theorems.
- Automatic differentiation also provides an efficient way to differentiate neural networks with respect to either the parameters of the network or its inputs.
- With these two ingredients, we can formulate a new, data-efficient way to approach solving PDEs, by obtaining actual functions that (hopefully) approximate the solution.
- The following is based on the work of (Raissi, Perdikaris, and Karniadakis, 2017).

A New Paradigm

- Consider the general problem

$$\begin{aligned} u_t(x,t) + \mathcal{N}[u](x,t) = 0,\ (x,t) \in \mathcal{D} \times (0,T) \\ u(x,t) = g(x,t),\ (x,t) \in \partial\mathcal{D} \times (0,T) \\ u(x,0) = f(x),\ x \in \mathcal{D} \cup \partial\mathcal{D}, \end{aligned}$$

  where $\mathcal{N}$ is an operator (linear or non-linear), and $\mathcal{D} \subseteq \mathbb{R}^n$ is some domain.
- Let u_Θ be a multi-layer perceptron (MLP), with parameters given by Θ.

A New Paradigm

- Let $\{(x_b^i, t_b^i, u_b^i)\}_{i=1}^{N_b}$, $\{(x_0^i, t_0^i, u_0^i)\}_{i=1}^{N_0}$ be the known boundary and initial state data, respectively. Let $P = \{(x_p^i, t_p^i)\}_{i=1}^{N_p}$ be a set of *collocation points*.
- We define the *residual loss* of the network at (x, t) by

$$r_\Theta(x, t) = \frac{\partial u_\Theta(x, t)}{\partial t} + \mathcal{N}[u_\Theta](x, t).$$

- The total loss of the network u_Θ is given by

$$L_\Theta = \frac{1}{N_b}\sum_{i=1}^{N_b}\left[u_\Theta(x_b^i, t_b^i) - u_b^i\right]^2 + \frac{1}{N_0}\sum_{i=1}^{N_0}\left[u_\Theta(x_0^i, t_0^i) - u_0^i\right]^2 + \frac{1}{N_p}\sum_{i=1}^{N_p} r_\Theta(x_p^i, t_p^i)^2.$$

- Adding the 'physics loss' can be seen as regularising the solution so that it obeys known physical laws.

Example: One-dimensional heat equation

- Consider the one-dimensional heat equation given by

$$\begin{aligned} u_t(x,t) - u_{xx}(x,t) = 0,\ (x,t) \in (0,2\pi) \times (0,5] \\ u(0,t) = 0 = u(2\pi,t),\ t \in (0,5] \\ u(x,0) = \sin(x),\ x \in [0,2\pi]. \end{aligned}$$

- The analytical solution can be found using separation of variables and is given by

$$u(x,t) = \sum_{n=1}^{\infty} A_n \sin\left(\frac{n}{2}x\right) \exp\left(-\frac{n^2}{4}t\right),$$

where $\{A_n\}_{n=1}^{\infty}$ are the Fourier coefficients of sin.

Findings

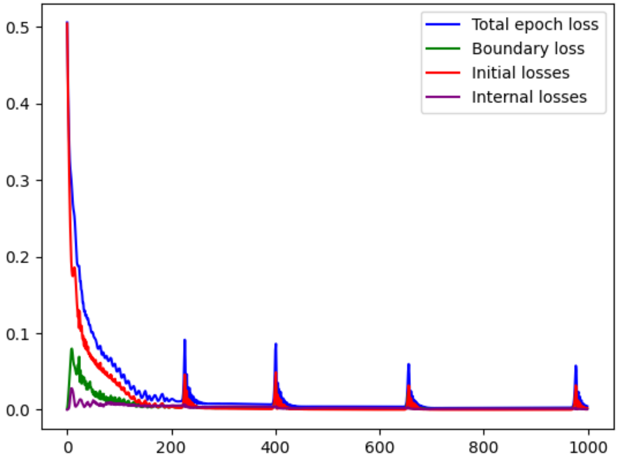
- We implemented a basic MLP to solve this equation:

```
pinn1D(
  (pinn): Sequential(
    (0): Linear(in_features=2, out_features=100, bias=True)
    (1): Tanh()
    (2): Linear(in_features=100, out_features=100, bias=True)
    (3): Tanh()
    (4): Linear(in_features=100, out_features=100, bias=True)
    (5): Tanh()
    (6): Linear(in_features=100, out_features=100, bias=True)
    (7): Tanh()
    (8): Linear(in_features=100, out_features=1, bias=True)
  )
)
```

- Code:
  https://github.com/ESkoufris/PhysicsInformedMachineLearning-WinterResearchProject.git

Extensions

- The loss can be a bit unstable:



Extensions

- Refactor the code so that it works with 64 bit floating point numbers, for higher precision.
- Alter the sampling method.
- Experiment more with hyperparameters.
- Extend the current code so that it works for higher dimensional PDEs.
- Experiment with different architectures.
  - A GAN can make use of any data that are available representing either the exact solution at certain points or physically observed data.
- Examine more the ideas of Neural Operators.

References

Raissi, M., P. Perdikaris, and G. E. Karniadakis (2017). *Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations*. arXiv: 1711.10561 [cs.AI]. URL: https://arxiv.org/abs/1711.10561.