A Survey of Physics-informed Neural Networks and Non-differentiable Optimization

Yang Junyan

October 1, 2021

1 Introduction

Physics-informed Neural Networks (PINN) are neural networks that are trained to solve supervised learning tasks while respecting any given law of physics described by general nonlinear partial differential equations. It forms a new class of data-efficient universal function approximators that naturally encode any underlying physical laws as prior information.

2 Physics-informed Neural Networks

Here is the general form of the parameterized and nonlinear partial differential equations.

$$u_t + \mathcal{N}[u; \lambda] = 0, \tag{1}$$

where u(t, x) denotes the latent (hidden) solution and $\mathcal{N}[;\lambda]$ is a nonlinear differential operator parameterized by λ. one dimensional Burgers' equation as demo
two main classes of problem:
1. Given fixed model parameters λ what can be said about the unknown hidden state u(t, x) of the system?
2. Given the the observed data what are the parameters λ that best describe it?

2.1 Data-driven Solution

A major function of PINN is to obtain data driven solutions of partial differential equations, the general form of which is shown in equation 2.

$$u_t + \mathcal{N}[u] = 0, x \in \Omega, t \in [0, T], \tag{2}$$

where u(t, x) denotes the latent (hidden) solution, N [·] is a nonlinear differential operator, and Ω is a subset of $\mathbf{R}^D$. Considering the different nature of continuous-time and discrete-time problems, two corresponding sets of model were proposed.

2.1.1 Continuous Time Models

f(t,x) is defined as

$$f := u_t + \mathcal{N}[u], \tag{3}$$

then u(t,x) is approximated by a deep neural network. Such action result in a PINN f(t,x)

- Example: Burgers' Equation and Shrödinger Equation

2.1.2 Discrete Time Models

The general form of Runge-Kutta methods with q stages is expressed as

$$\begin{aligned} u^{n+c_i} &= u^n - \Delta t \sum_{j=1}^{q} a_{ij} \mathcal{N}[u^{n+c_j}], i = 1, ..., q, \\ u^{n+1} &= u^n - \Delta t \sum_{j=1}^{q} b_j \mathcal{N}[u^{n+c_j}], \end{aligned} \tag{4}$$

where $u^{n+c_j}(x) = u(t^n + c_j \Delta t, x)\ for\ j = 1, ..., q\ and\ a_{ij}, b_j, c_j$ are parameters. The equations above are equal to

$$\begin{aligned} u^n &= u_i^n,\ i = 1, ...q, \\ u^n &= u_{q+1}^n \end{aligned} \tag{5}$$

where

$$\begin{aligned} u_i^n &:= u^{n+c_i} + \Delta t \sum_{j=1}^{q} a_{ij} \mathcal{N}[u^{n+c_j}], i = 1, ..., q, \\ u_{q+1}^n &:= u^{n+1} + \Delta t \sum_{j=1}^{q} b_j \mathcal{N}[u^{n+c_j}] \end{aligned} \tag{6}$$

A PINN that takes x as an input and $[u_1^n(x), ..., u_q^n(x), u_{q+1}^n(x)]$ as output can be constructed when neural network is placed prior on $[u^{n+c_1}(x), ..., u^{n+c_q}(x), u^{n+1}(x)]$.

- Example: Allen-Cahn Equation

2.2 Data-driven Discovery

Another major function of PINN is to perform data-driven discovery of partial differential equations.For example, in regard of equation 2, what are the parameters λ that best describe the observed data, which is a small set of scattered and potentially noisy observations of the hidden state u(t,x) of a system?

2.2.1 Continuous Time Models

The PINN f(t,x) is defined as

$$f := u_t + \mathcal{N}[u; \lambda], \tag{7}$$

where u(t,x) is approximated by a deep neural network.

- Example: Navier-Stokes Equation

2.2.2 Discrete Time Models

The general form of Runge-Kutta method is shown

$$\begin{aligned} u^{n+c_i} &= u^n - \Delta t \sum_{j=1}^{q} a_{ij} \mathcal{N}[u^{n+c_j}; \lambda], i = 1, ..., q, \\ u^{n+1} &= u^n - \Delta t \sum_{j=1}^{q} b_j \mathcal{N}[u^{n+c_j}; \lambda], \end{aligned} \tag{8}$$

where $u^{n+c_j}(x) = u(t^n + c_j, x)\ for\ j = 1, ..., q\ and\ a_{ij}, b_j, c_j$ are parameters. The equation 8 can also be expressed as

$$\begin{aligned} u^n &= u_i^n,\ i = 1, ...q, \\ u^n &= u_i^{n+1},\ i = 1, ..., q. \end{aligned} \tag{9}$$

where

$$
\begin{aligned}
u_i^n &:= u^{n+c_i} + \Delta t \sum_{j=1}^{q} a_{ij} \mathcal{N}[u^{n+c_j}; \lambda], i = 1, ..., q, \\
u_i^{n+1} &:= u^{n+c_i} + \Delta t \sum_{j=1}^{q} (a_{ij} - b_j) \mathcal{N}[u^{n+c_j}; \lambda], i = 1, ..., q.
\end{aligned}
\tag{10}
$$

By placing a neural network prior on $[u^{n+c_1}(x), ..., u^{n+c_q}(x)]$, we obtain two physics informed neural networks $[u_1^n(x), ..., u_q^n(x), u_{q+1}^n(x)]$ and $[u_1^{n+1}(x), ..., u_q^{n+1}(x), u_{q+1}^{n+1}(x)]$. Please be noted these two PINN share the same set of parameter as the causality between the input and output is identical. Therefore, with two sets of snapshots at t^n and t^{n+1}: x^n, u^n and x^{n+1}, u^{n+1}, we can train the PINN by minimizing the sum of squared errors

$$
SSE = SSE_n + SSE_{n+1} \tag{11}
$$

where

$$
\begin{aligned}
SSE_n &:= \sum_{j=1}^{q} \sum_{i=1}^{N_n} |u_j^n(x^{n,i}) - u^{n,i}|^2, \\
SSE_{n+1} &:= \sum_{j=1}^{q} \sum_{i=1}^{N_{n+1}} |u_j^{n+1}(x^{n+1,i}) - u^{n+1,i}|^2.
\end{aligned}
\tag{12}
$$

where $x^n = {x^{n,i}}_{i=1}^{N_n}$, $u^n = {u^{n,i}}_{i=1}^{N_n}$, $x^{n+1} = {x^{n+1,i}}_{i=1}^{N_{n+1}}$, and $u^{n+1} = {u^{n+1,i}}_{i=1}^{N_{n+1}}$

- Example: Korteweg–de Vries Equation

3 Non-differentiable Optimization

Without traditional nondifferentiable optimization:

- Differentiable Approximation:
  use NN to substitute loss function.

- Utilize Automatic Differentiation:
  Some logic losses are almost everywhere differentiable function, obtaining the derivative of which by auto differentiation has been proven partially correct.
  A almost everywhere differentiable function means the function is differentiable everywhere except on a set of measure 0.

- Derivative-free Optimization :
  The algorithm finds high quality solutions by randomly sampling candidate solutions from a parameterized distribution model over the solution space.
  Instead of optimizing the original loss function H(x), a parameterized family of probability density function on X with Θ being a parameter space $\{f(x;\theta)|\theta \in \Theta \subseteq R^d\}$ is introduced and the loss function is defined as:

  $$
  H(x)f(x:\theta) \tag{13}
  $$

  which is continuous on the parameter space and usually differentiable with respect to θ. Then candidate solution from $f(x;\theta)$ on the solution space X is generated, and a gradient-base method is implemented to update parameter θ.

- Isotonic Modeling:
  Fitting isotonic models under **convex** non-differentiable loss functions. However, this may not fit into our problem.

Nondifferentiable optimization:

- convex world:
  Traditional convex analysis has revealed many solutions including but not limited to subgradient and ϵ- subgradient method

  - subgradient method
    Generating a sequence $\{x_k\}_{k=0}^{\infty}$ according to formula

    $$x_{k+1} = x_k - h_k(x_k)g_f(x_k) \tag{14}$$

    where x_0 is a given starting point, $g_f(x_k)$ is an arbitrary subgradient from subgradient set $G_f(x_k)$. Please be noted the the stepsize h_k is not constant and such method is only capable for continuously differentiable function.

  - ϵ-subgradient method
    ϵ-subgradient method is like steepest decent method but the direction is separately determined.
    Let f(x) be a convex function defined on E^n, and $\epsilon \geq 0$:

    $$G_f^{\epsilon}\overline{x} = g \in E^n : f(x) - f(\overline{x}) \geq (g, x - \overline{x}) - \epsilon \; for \; all \; x \tag{15}$$

    The direction of ϵ - steepest descent direction is determined by the flowing fomular

    $$d_{\epsilon}(\overline{x}) = -\frac{g^*}{||g^*||}, where \; g^* = argmin_{g G_f^{\epsilon}(\overline{x})}||g|| \tag{16}$$

    where $\overline{x}$ is nonminimum point in the neighborhood of this point.
    If $\epsilon = 0$, $d_0(\overline{x})$ is considered with the direction of the deepest descent in the point $\overline{x}$.

  Some alterations has been made to the over-mentioned methods such as Fejer-type subgradient methods, stochastic subgradient method, and other subgradient-type methods with space dilation.
  There are also methods based on non-smooth optimization that could be looked into in the future.

- non-convex world:

4 Correctness of Automatic Differentiation

The autodiff system is correct if the function f is differentiable almost everywhere. The justification of autodiff systems relies on two key concepts: piecewise analyticity under analytic partition, and intensional derivative.

- piecewise analyticity under analytic partition

  For X R^n and Y R^m, a set of pairs $\gamma = \{\langle A^i, f^i\rangle\}_{i\in[I]}$ is a piecewise representation of function from X to Y. For instance ReLU has representation: $\{\langle R_{<0}, x \to 0\rangle, \langle R_{\geq 0}, x \to x\rangle\}$.

  A set $A \subseteq R^n$ is analytic if and only if for some J,L $\in Z_{>0}$, there are analytic functions $g_j^+ : X_j^+ \to R$ and $g_l^- : X_l^- \to R$ over open domains X_j^+, $X_l^- \subseteq R^n (j \in [J], l \in [L])$ such that $A = \{x \in R^n | (\bigwedge_{j\in[J]} x \in X_j^+ \bigwedge g_j^+(x) > 0) \bigwedge (\bigwedge_{l\in[L]} x \in X_l^- \bigwedge g_t^-(x) \leq 0)\}$. A partition $\{A^i\}_{i\in[I]}$ of X is analytic if and only if all A^i are analytic.

  A representation $\gamma = \{\langle A^i, f^i\rangle\}_{i\in[I]}$ from X to Y is piecewise analytic under analytic partition (PAP in short), if and only if $\{A^i\}_{i\in[I]}$ is an analytic partition of X and f^i is analytic over its domain X^i for all $i \in [I]$.

A function f: X → Y is piecewise analytic under analytic partition if f = $\ll \gamma \gg$ for some PAP representation γ.

- Intensional derivative
  A intensional derivative of γ is the set $D_\gamma = \{\langle A^i, Df^i\rangle\}_{i\in[I]}$, where Df^i is the standard derivative of f^i viewed as a function from the domain X^i of f^i to $R^{m\times n}$ where $\gamma = \{\langle A^i, f^i\rangle\}$ is a PAP representation from X to Y for some $X \subseteq R^n$ and $Y \subseteq R^m$.
  Let f : X → Y be a PAP function. Define $\partial_* f$ to be the following set of functions: $\partial_* f = \{\ll D_\gamma \gg | \gamma \; is \; a \; PAP \; representation \; of \; f\}$. Each $df \in \partial_* f$ is called a first-order intensional derivative of f.
  The paper had proven for all $k \in Z_{\geq 0}$, the set $\partial_*^k f$ of the k-th order intensional derivatives is not empty.

For an autodiff system that satisfies the two requirements, there exist an intensinal derivative df of [e], the program.
Regarding the autodiff in practice, Pytorch and Tensorflow are not designed specifically to deal with nondifferentiable function the paper only mentioned that the derivative of ReLu at 0 is set as 0. Their correctness at certain points on other function is still partly questionable.
There are earlier studies in the field of autodiff and stochastic gradient descent.
https://papers.nips.cc/paper/2018/file/142c65e00f4f7cf2e6c4c996e34005df-Paper.pdf

5 Surrogate Loss

Substitute the non-differentiable function into a differential one.
Including but not limited to relu, sigmoid. Back in 2009, surrogate loss function was used for binary and multi-class classification.
In A Unified Framework of Surrogate Loss by Refactoring and Interpolation (2020), it proposed "UniLoss", which provided a unified way to generate a surrogate loss for deep network. Instead of minimizing the original loss e, where $e = (\delta(\phi(x; w), y))$, here $\phi(x; w)$ represent a neural network. The UniLoss method generate differential functions and the new loss is

$$\tilde{e} = \tilde{g}(\tilde{h}(f(\phi(x; w), y)) \tag{17}$$

These functions are generated by interpolation. Inverse distance weighting was used in the paper. In the paper On the Consistency of Output Code Based Learning Algorithms for Multiclass Learning Problems(2014), methods for binary and multi-class classifier were listed.
For binary classification, we would like to minimize the surrogate loss $[\eta\phi_1(v) + (1-\eta)\phi_{-1}(v)]$.

Loss	$\mathcal{V}$	$\phi_1(v)$	$\phi_{-1}(v)$	$\lambda(\eta)$	$\lambda^{-1}(v)$
Logistic	$\overline{\mathbb{R}}$	$\ln(1+e^{-v})$	$\ln(1+e^{v})$	$\ln\left(\frac{\eta}{1-\eta}\right)$	$\frac{1}{1+e^{-v}}$
Exponential	$\overline{\mathbb{R}}$	e^{-v}	e^{v}	$\frac{1}{2}\ln\left(\frac{\eta}{1-\eta}\right)$	$\frac{1}{1+e^{-2v}}$
Least-squares	$[-1,1]$	$(v-1)^2$	$(v+1)^2$	$2\eta-1$	$\frac{v+1}{2}$

There are some well-known strictly proper composite binary surrogates, with their partial losses, link functions and corresponding inverse linkd. In the paper Learning Surrogate Loss, Kolmogorov–Arnold representation theorem was used to formulate the loss as

$$\hat{l}(y, \hat{y}) = h(\frac{1}{N}\sum_{i=1}^{N} g(y_i, \hat{y}_i)) \tag{18}$$

where j and g are deep forward neural network. Example: Normalized discounted cumulative gain (A Guided Learning Approach for Item Recommendation via Surrogate Loss Learning https://dl.acm.org/doi/10.1145/340483
Current methods for surrogate model are Kriging, Kriging using Stheno, Radial Basis, Wendland, Linear, Second Order Polynomial, Support Vector Machines (Wait for LIBSVM resolution), Neural Networks, Random Forests, Lobachevsky Inverse-distance, Polynomial expansions, Variable fidelity,

Earth, Gradient Enhanced Kriging. However, their capability of dealing with non-differential function have not been validated.

6 Problem this week

7 To-do List

- Reproduce experiments

References

[1] M.Raissi, P.Perdikaris, G.E.Karniadakis *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations.* Journal of Computational Physics, 2019

[2] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis *Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations* . November 30, 2017