Physics-informed machine learning

Emmanuel Skoufris

University of Queensland

July 16, 2024

Discretization invariance

- These definitions are based on (Li et al., 2020).
- For any subset $D \subseteq \mathbb{R}^d$, we call a sequence of increasing, nested sets $D_1 \subseteq D_2 \subseteq \cdots \subseteq D$ satisfying $|D_k| = k$ for every $k \in \mathbb{N}$ a *discrete refinement* of D if, for arbitrary $\epsilon > 0$, there exists some $K \in \mathbb{N}$ such that

$$D \subseteq \bigcup_{x \in D_L} B_\epsilon(x) = \bigcup_{x \in D_L} \{y \in D : \|x - y\|_2 < \epsilon\}.$$

- Let $\mathcal{A}$ be a Banach space of $\mathbb{R}^m$-valued functions defined on $D \subset \mathbb{R}^d$. Let $\mathcal{G} : \mathcal{A} \to \mathcal{U}$ be an operator, D_L an L-point discretization of D, and $\hat{\mathcal{G}} : \mathbb{R}^{Ld} \times \mathbb{R}^{Lm} \to \mathcal{U}$ some map. For any compact $K \subseteq \mathcal{A}$, we define the *discretized uniform risk* by

$$\begin{aligned} R_K(\mathcal{G}, \mathcal{G}_L, D_L) &= \sup_{a \in K} \|\hat{\mathcal{G}}(D_L, a_{|D_L}) - \mathcal{G}(a)\|_{\mathcal{U}} \\ &= \sup_{a \in K} \int_{D'} [\hat{\mathcal{G}}(D_L, a_{|D_L})(x) - \mathcal{G}(a)(x)]^2 dx. \end{aligned}$$

Discretization invariance

- Let $\Theta \subseteq \mathbb{R}^p$ be a parameter subspace, and $\mathcal{G} : \mathcal{A} \times \Theta \to \mathcal{U}$ a parametric map. Given a discrete refinement $\{D_L\}_{L=1}^{\infty}$ of a domain $D \subseteq \mathbb{R}^d$, $\mathcal{G}$ is said to be *discretization invariant* if there exists a sequence of maps $\hat{\mathcal{G}}_1, \hat{\mathcal{G}}_2, \dots$ with $\hat{\mathcal{G}}_L : \mathbb{R}^{Ld} \times \mathbb{R}^{Lm} \times \Theta \to \mathcal{U}$ such that for all $\theta \in \Theta$, and for any compact $K \subseteq \mathcal{A}$,

$$\lim_{L\to\infty} R_K(\mathcal{G}(\cdot,\theta), \hat{\mathcal{G}}_L(\cdot,\cdot,\theta), D_L).$$

- Question: What is the proper interpretation and motivation of this definition in relation to neural operators?

Introduction

- Last week, we examined the ideas of Neural Operators, and, more specifically, the Fourier Neural Operator (FNO).
- These networks construct an operator between the space of boundary/initial conditions to the solution space.
- The benefits of a neural operator include discretization invariance and convergence, which are unique features of these models.
- Now, is there a way to further universalise these solvers, by training them to learn the operator mapping the domain + intial/boundary conditions to the solution?
- The idea of the Geometry-informed Neural Operator is to learn how to map various subdomains of $\mathcal{D}$ to the solution.
- We will need to place some technical restrictions on which geometries we can consider.

Problem setup

- What follows is based on (Li et al., 2020) and (Li et al., 2023).
- Consider some Lipschitz domain[1] $\mathcal{D} \subseteq \mathbb{R}^d$, and some Banach space of real functions $\mathcal{A}$ defined on $\mathcal{D}$.
- We let $\mathcal{T} \subseteq \mathcal{A}$ be some subset of *distance functions* such that, for each $T \in \mathcal{T}$, the set

$$S_T = T^{-1}(\{0\}) = \{x \in \mathcal{D} : T(x) = 0\}$$

  is a $(d-1)$-dimensional sub-manifold.
- S_T is the "surface" of interest in our PDE.
- Assume that for each T, S_T is simply-connected, closed (compact with trivial *geometric* boundary), smooth, and that there exists some $\epsilon > 0$ such that $B_\epsilon(x) \cap \partial\mathcal{D} = \varnothing$ for every $x \in S_T$, for every $T \in \mathcal{T}$.

[1] Meaning that, at each point x of $\partial\mathcal{D}$, $\mathcal{D}$ is locally the set of points located above some Lipschitz function.

Problem setup

- Let Q_T be the open volume[2] enclosed by the hypersurface S_T, so that $\partial Q_T = S_T$.
- Finally, define $\Omega_T = D \setminus \bar{Q}_T$, meaning that $\partial \Omega_T = \partial \mathcal{D} \cup S_T$.
- Let $\mathcal{L}$ be a differential operator and consider the problem

$$\begin{aligned} \mathcal{L}(u)(x) &= f(x),\ x \in \Omega_T \\ u(x) &= g(x),\ x \in \partial\Omega_T, \end{aligned}$$

for some $f \in \mathcal{F}$ and $g \in \mathcal{B}$, where $\mathcal{F}$ and $\mathcal{B}$ are Banach spaces of functions defined on $\mathbb{R}^d$.

[2] We also assume Q_T to be a Lipschitz domain

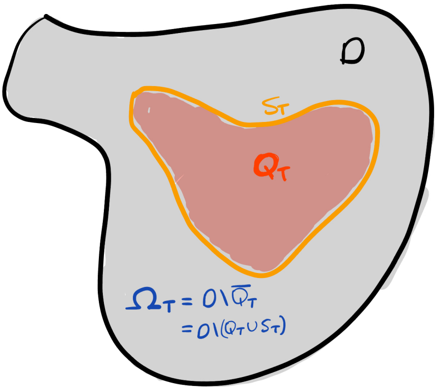
Problem setup

- Let $\mathcal{U}$ denote a Banach space of functions on $\mathcal{D}$ and U_T a Banach space of functions on Ω_T.
- Assume that $\mathcal{L}$ is such that for any such triplet (T, f, g), the PDE has a unique solution.
- Let $\{E_T : \mathcal{U}_T \to \mathcal{U}\}$ denote a family of extension operators that are linear and bounded (i.e. continuous).
- The operator we wish to estimate is

$$\Psi : \mathcal{T} \times \mathcal{F} \times \mathcal{B} \to \mathcal{U},$$

  where $\Psi(T, f, g) = E_T(u)$.
- It is not entirely clear in (Li et al., 2023) why the final output function has to be an extended operator defined on all of $\mathcal{D}$.

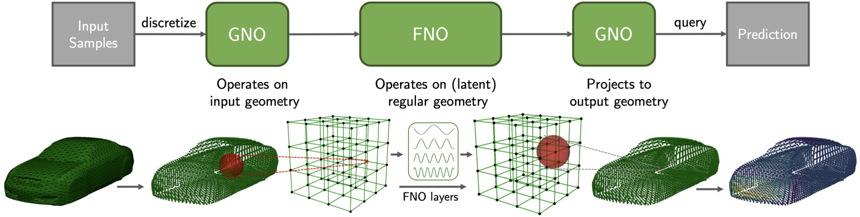
Example



Geometry-informed neural operator

- The *Geometry-informed neural operator* (GINO) can be used to estimate Ψ.
- We feed the model a discretization of the geometry, the distance function T, and the initial and boundary conditions evaluated on the discretization.
- A graph neural operator (GNO) constructs a function on a latent regular grid, which is then fed through to a FNO.
- Using the output function of the FNO, a second GNO then constructs the solution function defined on Ω_T.

Geometry-informed neural operator



Graph Neural Operator

- Recall that the innovation of neural operators is to recursively define a series of kernels

$$(K_\ell v_{\ell-1})(x) = \int_{\mathcal{D}} \kappa_\ell(x, y) v_{\ell-1}(y) dy, \; x \in \mathcal{D}.$$

- Last time we parameterised the kernel in Fourier space, and leveraged the speed of the FFT algorithm.
- The original parameterisation of κ_ℓ however was defined via a GNN, which we'll now explore.

Graph neural operator

- To simplify calculations, for each x, we reduce the integral to be defined over a ball $B_r(x)$:

$$(K_\ell v_{\ell-1})(x) = \int_{B_r(x)} \kappa_\ell(x,y) v_{\ell-1}(y) dy, \; x \in \mathcal{D}$$

  for some uniform $r > 0$.
- Given an N-point discretization $\mathcal{D}_N$, for each $x, y \in \mathcal{D}$, we therefore have that $\kappa_\ell(x,y) \in \mathbb{R}^{d \times d}$.
- Therefore, in discrete form, $\kappa_{\ell-1} \in \mathbb{R}^{Nd \times Nd}$, i.e. $\kappa_{\ell-1}$ is a $N \times N$ block matrix, the entries of which we wish to learn.
- To ensure that the GNO is discretization invariant, we keep the matrix entries the same across the N^2 blocks of $\kappa_{\ell-1}$.

Graph neural operator

- We then form a graph G that has nodes $\mathcal{D}_N$, with vertex features $v_{\ell-1} \in \mathbb{R}^{N\times d}$, and edge weights $e(x,y) = (x, y, a(x), a(y)) \in \mathbb{R}^{N\times 4d}$.
- The neighbourhoods of each node are given by $\mathcal{N}(x) = B_r(x) \cap \mathcal{D}_N$.
- Then

$$(K_\ell v_{\ell-1})(x) = \sum_{y\in N(x)} \kappa_\ell(x,y) v_{\ell-1}(y)\mu(y).$$

- Thus, we have a message passing GNN with average aggregation:

$$v_\ell(x) = \sigma\left(W v_{\ell-1}(x) + \sum_{y\in N(x)} \kappa_\ell(x,y) v_{\ell-1}(y)\mu(y)\right), \ell = 2, \dots, L.$$

- One can add as many GNO layers as needed, although we must define the first layer differently, as we'll see.

Graph neural operator

- Given a set of points $\{x_1^{\text{in}}, \dots, x_N^{\text{in}}\} \subseteq S_T \subseteq \mathcal{D}$, we use a GNO-encoder to obtain v_0, a function defined on a uniform grid, which is then fed through any further GNO layers, and eventually a Fourier layer.
- Let $\{x_1^{\text{grid}}, \dots, x_S^{\text{grid}}\} \subseteq D$ represent the latent grid, for a fixed resolution S. Then we compute:

$$v_0(x^{\text{grid}}) = \sum_{y^{\text{in}} \in B_r(x^{\text{grid}})} \kappa(x^{\text{grid}}, y^{\text{in}})\mu(y^{\text{in}}),$$

where the weights $\mu(y^{\text{in}})$ are also fine-tuned during training.

Fourier operator block

- We feed the output of the GNO into an FNO - for simplicity, assume the GNO only contains one layer, so that its output is the function v_0 defined on the latent regular grid.
- Then the output of the FNO is given by

$$(Kv_0)(x) = \sigma(Wv_0(x) + \mathcal{F}^{-1}(\mathcal{F}(\kappa) \cdot \mathcal{F}(v_0)), \, x \in \text{grid},$$

where $\mathcal{F}$ is the DFT, evaluated via the FFT algorithm, which can be used since this v_0 is defined on a uniform grid.

GNO Decoder

- Given a function defined on the grid, we randomly sample points $\{x_1^{\text{out}}, \ldots, x_N^{\text{out}}\} \subseteq \Omega_T$, and compute

$$u(x^{\text{out}}) = \sum_{y^{\text{grid}} \in B_r(x^{\text{out}}) \cap \text{grid}} \kappa(x^{\text{out}}, y^{\text{grid}}) v(y^{\text{grid}}) \mu(y^{\text{grid}}),$$

where we set $\mu(y^{\text{grid}}) = 1/S$ since the grid is uniform.

What's next

- The most obvious shortfalls of the GINO is that it is not physics-informed, and it has not been tested on a diverse range of geometries.
- Speaking generally, it seems that future work in this area will aim to further "universalize" PDE solvers.
- Recall that we went from using a neural network to solve a single instance of a PDE using PINNs, to solving a PDE for a range of boundary conditions, to now solving a PDE for a range of a boundary conditions and geometries.
- To my knowledge, creating a solver adapted to a variety of different differential operators has not yet been considered.

What's next

- We could, for example, let Λ denote some class of differential operators over a space.
- In general, if Λ is continuous, it could be infinite dimensional, although we could fix a basis and take its span, e.g.:

$$\Lambda = \operatorname{span}_{(\mathbb{R} \text{ or } \mathbb{C})} \left\{ \frac{\partial u}{\partial x}, \frac{\partial^2 u}{\partial y^2}, u\frac{\partial^2 u}{\partial x \partial y}, \sin\left(\frac{\partial u}{\partial y}\right) \right\} \cong \mathbb{R}^4.$$

- This basis is an illustrative example only: in reality, one would likely pick basis operators that appear in common differential equations.
- Then, the operator of interest is

$$\Psi : \Lambda \times \mathcal{T} \times \mathcal{F} \times \mathcal{B} \to \mathcal{U}.$$

References

- Li, Z. et al. (2020). "Neural Operator: Learning Maps Between Function Spaces with Applications to PDEs". In: *arXiv preprint arXiv:2003.03485*.
- Li, Z. et al. (2023). *Geometry-Informed Neural Operator for Large-Scale 3D PDEs*. arXiv: 2309.00583 [cs.LG]. URL: https://arxiv.org/abs/2309.00583.