Supervised learning in function spaces

Part IV: Advanced topics and applications

https://github.com/PredictiveIntelligenceLab/TRIPODS_Winter_School_2022

Instructors:

- Paris Perdikaris (University of Pennsylvania, pgp@seas.upenn.edu)
- Jacob Seidman (University of Pennsylvania, seidj@sas.upenn.edu)
- Georgios Kissas (University of Pennsylvania, gkissas@seas.upenn.edu)

2022 TRIPODS Winter School & Workshop on Interplay between Artificial Intelligence (AI) and Dynamical Systems
Mathematical Institute for Data Science (MINDS)
Johns Hopkins University

LOCA

LEARNING OPERATORS WITH COUPLED ATTENTION

A PREPRINT

Georgios Kissas [*1], Jacob Seidman [*2], Leonardo Ferreira Guilhoto[2],
Victor M. Preciado[3], George J. Pappas[3], and Paris Perdikaris[1]

[1]Department of Mechanical Engineering and Applied Mechanics, University of Pennsylvania, Philadelphia, PA 19104
[2]Graduate Group in Applied Mathematics and Computational Science , University of Pennsylvania, Philadelphia, PA 19104
[3] Department of Electrical and Systems Engineering , University of Pennsylvania, Philadelphia, PA 19104

DOE (grant DE-SC0019116), US Air Force (grant AFOSR FA9550-20-1-0060), DOE (grant DE-AR0001201), AFOSR (grant FA9550-19-1-0265) NSF(grant 2031985)

Bahdanau Attention

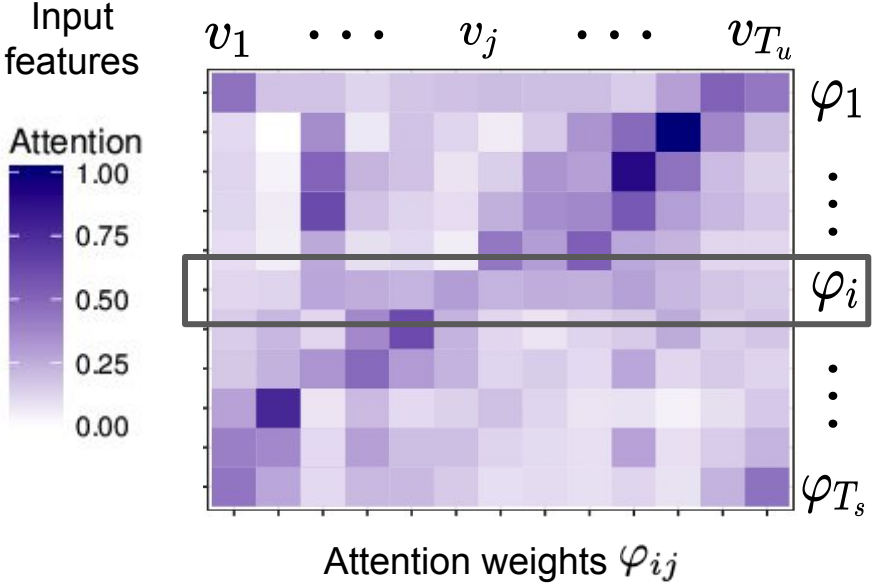
- Goal was to translate $\{u_1, \ldots, u_{T_u}\}$ to $\{s_1, \ldots, s_{T_s}\}$
- First map input to features $\{u_1, \ldots, u_{T_u}\} \longrightarrow \{v_1, \ldots, v_{T_u}\}$
- Each location in the output sentence gets a context vector c_i to determine the probability of a word at that location.
- Context vector is constructed as a position-dependent average of the features.

Bahdanau, Dzmitry, Kyung Hyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." *3rd International Conference on Learning Representations, ICLR 2015*. 2015.

Bahdanau Attention

Input sentence $[u_1, \ldots, u_i, \ldots, u_{T_u}]$

Input features



Attention weights φ_{ij}

$$\begin{bmatrix} c_1 \\ \vdots \\ c_i \\ \vdots \\ c_{T_s} \end{bmatrix} \rightarrow c_i = \sum_{j=1}^{T_u} \varphi_{ij} v_j$$

Context vector

$$p(s_i \mid c_i, u)$$

Probability of word s_i at output position i

$[s_1, \ldots, s_i, \ldots, s_{T_s}]$

Output sentence

Bahdanau, Dzmitry, Kyung Hyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." *3rd International Conference on Learning Representations, ICLR 2015*. 2015.

Creation of Context Vector

- The context vector is an output-query dependent average of the input features

$$c_i = \sum_{j=1}^{T_u} \varphi_{ij} v_j$$

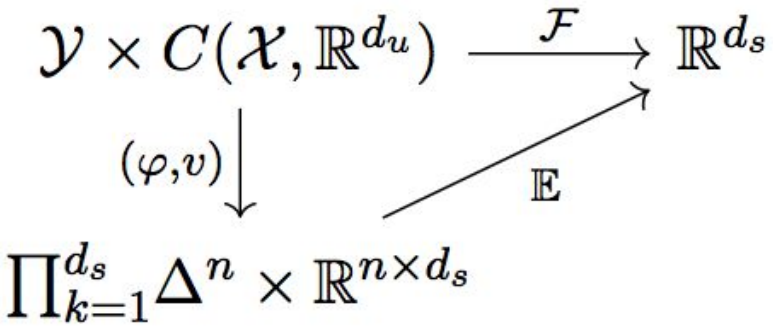
depends on output query location i

$$\begin{array}{ccc} [T_s] \times \mathbb{R}^{T_u \times d} & \xrightarrow{\text{Attn}} & \mathbb{R}^l \\ \downarrow{\scriptstyle (\varphi, v)} & \nearrow{\scriptstyle \mathbb{E}} & \\ \Delta^{T_u} \times \mathbb{R}^{T_u \times l} & & \end{array}$$

Attention Based Operator Learning Architecture

- First map input to features $v(u) \in \mathbb{R}^{n \times d_s}$
- Average these row-wise with distributions $\varphi(y) \in \prod_{k=1}^{d_s} \Delta^n$ which depend on the output query location

$$\mathbb{E}_{\varphi(y)}[v(u)] = \sum_{i=1}^{n} \varphi(y)_i \odot v(u)_i$$



Choosing the form of Attention Weights $\varphi(y)$

- Simplest choice is to use a score function $g : \mathcal{Y} \to \mathbb{R}^{n \times d_s}$, and normalize across rows with any function $\sigma : \mathbb{R}^n \to \Delta^n$ e.g. softmax.
- Using a sufficiently rich class of functions for g will allow us to approximate any φ in this way

HOWEVER:

We are relying on the function g to learn any relationships between the distributions at different points $\varphi(y), \varphi(y')$

Is there a way to model/learn these relationships more explicitly?

Kernel Coupled Attention

- To explicitly learn correlations between the attention weights $\varphi(y)$, we will couple the score function over its inputs with a kernel integral transform.

$$\tilde{g}(y) = \int_{\mathcal{Y}} k(y, z) g(z) dz$$

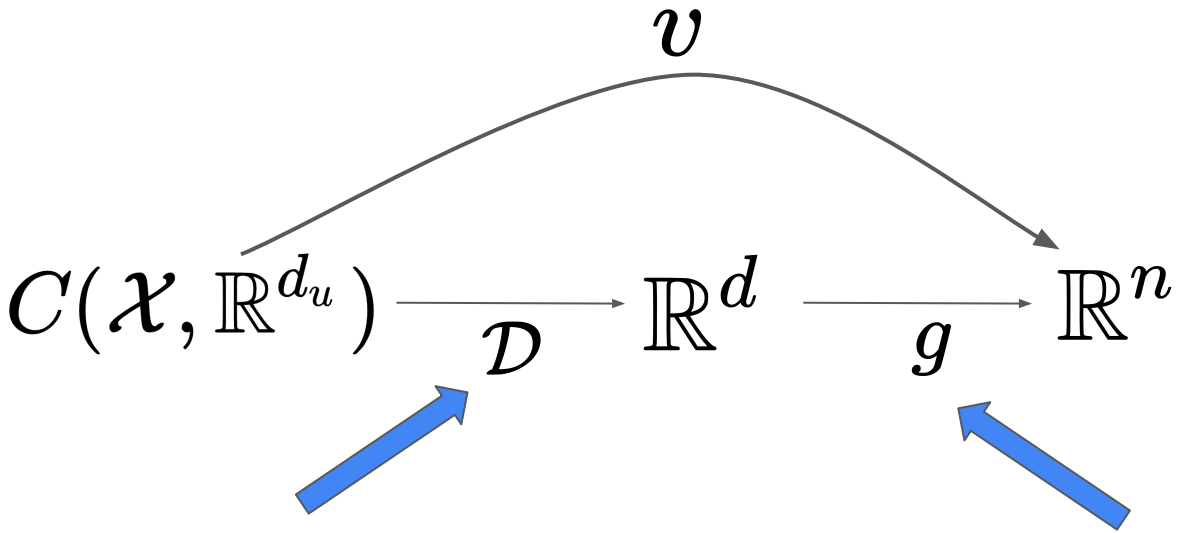
$$\varphi(y) = \sigma(\tilde{g}(y)) = \sigma\left(\int_{\mathcal{Y}} k(y, z) g(z) dz \right)$$

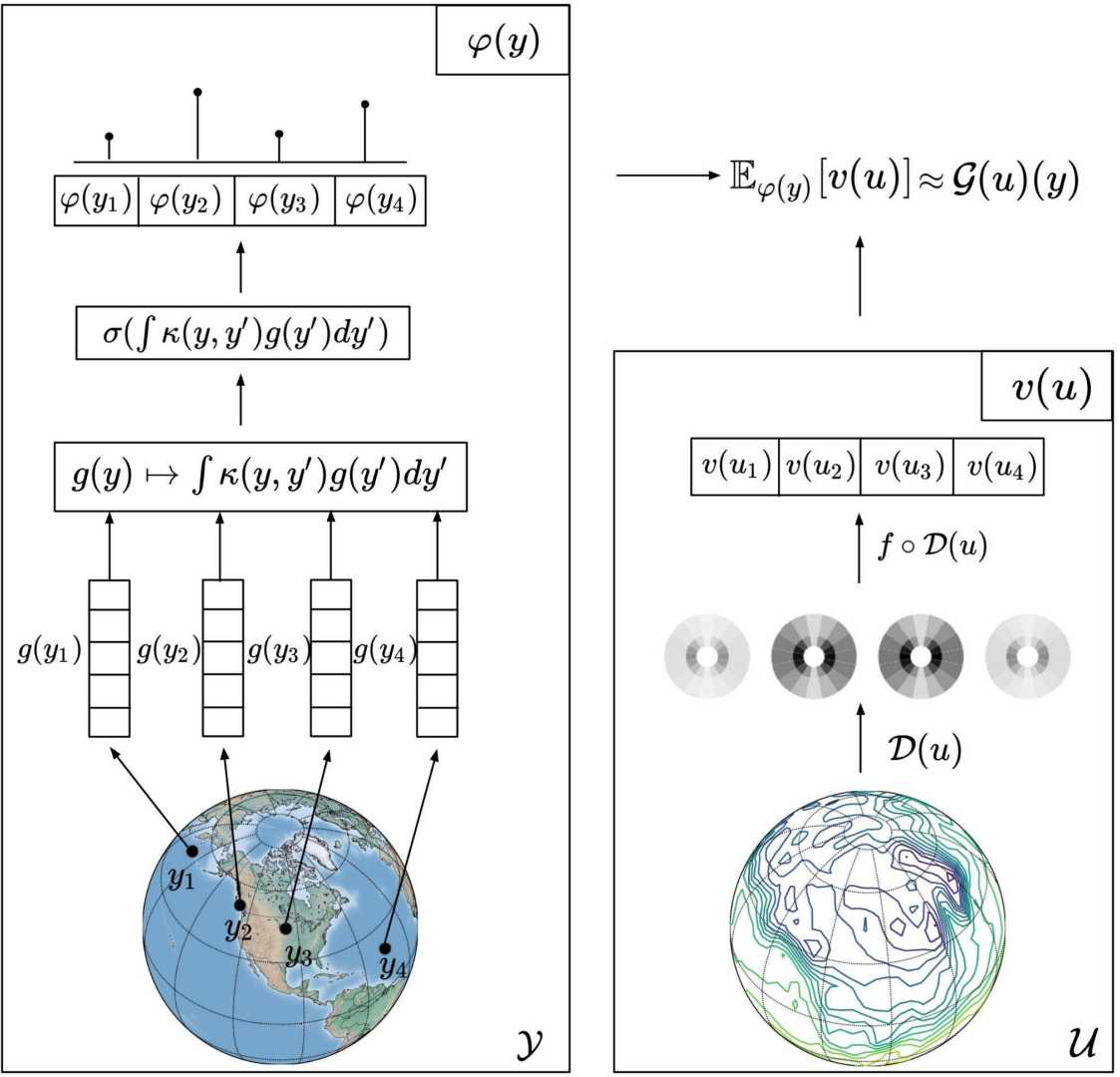
- By parameterizing the kernel the model can learn how to couple the score functions between different locations.

Input Feature Encoding υ

- Ideally want to be flexible with respect to number of measurements and measurement locations.

- An idea:





$$\mathbb{E}_{\varphi(y)}[v(u)] \approx \mathcal{G}(u)(y)$$

LOCA Overview

$$\sum_{i=1}^{n} \sigma\left(\int_{\mathcal{Y}} \kappa(y,y')g(y')\,dy'\right)_i \odot v_i(u)$$

Universality Guarantees

$$\mathcal{F}(u)(y) = \sum_{k=1}^{d} g_k(y) v_k(u)$$

Theorem (Chen & Chen '95): If $\mathcal{X} \subset \mathbb{R}^{d_x}, \mathcal{Y} \subset \mathbb{R}^{d_y}, \mathcal{U} \subset C(U, \mathbb{R}^{d_u})$ are all compact, given a continuous $\mathcal{G} : \mathcal{U} \to C(D, \mathbb{R})$, for any $\epsilon > 0$, there exists F of the above form such that

$$\sup_{u \in \mathcal{U}} \sup_{y \in \mathcal{Y}} \| F(u)(y) - \mathcal{G}(u)(y) \| < \epsilon$$

From the basic model to LOCA

$$\sum_{k=1}^{d} g_k(y) \odot v_k(u)$$

normalized version of the $g_k(y)$

$$\mathbb{E}_{\varphi(y)}\left[v_k(u)\right]$$

Kernel Coupling of weights

$$\mathbb{E}_{\varphi_{KCA}(y)}\left[v_k(u)\right]$$

Normalization Preserves Universality

$$\sum_{k=1}^{d} g_k(y) \odot v_k(u) \quad \Longrightarrow \quad \mathbb{E}_{\varphi(y)}[v_k(u)] = \sum_{i=1}^{n} \varphi_i(y) \odot v_i(u)$$

normalized version of the $g_k(y)$

Theorem (Normalization Preserves Universality) *If $\mathcal{U} \subset C(\mathcal{X}, \mathbb{R}^{d_u})$ is a compact set of functions and $\mathcal{G} : \mathcal{U} \to C(\mathcal{Y}, \mathbb{R}^{d_s})$ is a continuous operator with $\mathcal{X}$ and $\mathcal{Y}$ compact, then for every $\epsilon > 0$ there exists $n \in \mathbb{N}$, functionals $v_{j,k} : \mathcal{U} \to \mathbb{R}$, for $j \in [n]$, $k \in [d_s]$, and functions $\varphi_j : \mathcal{Y} \to \mathbb{R}^{d_s}$ with $\varphi_j(y) \in [0,1]^{d_s}$ and $\sum_{j=1}^{n} \varphi_j(y) = 1_{d_s}$ for all $y \in \mathcal{Y}$ such that*

$$\sup_{u \in \mathcal{U}} \sup_{y \in \mathcal{Y}} \left\| \mathcal{G}(u)(y) - \mathbb{E}_{\varphi(y)}[v(u)] \right\|_{\mathbb{R}^{d_s}}^2 < \epsilon.$$

Kernel Coupling Preserves Universality

- Universality statement requires us to approximate any $\varphi : \mathcal{Y} \to \Delta^n$ with

$$\sigma\left(\int_{\mathcal{Y}} k(y, z) g(z) dz \right)$$

- Suffices to show that functions of the form

$$\int_{\mathcal{Y}} k(y, z) g(z) dz$$

  are dense in $C(\mathcal{Y}, \mathbb{R}^n)$

Kernel Coupling Preserves Universality

$$\mathbb{E}_{\varphi(y)}\left[v_k(u)\right] \longrightarrow \mathbb{E}_{\varphi_{KCA}(y)}\left[v_k(u)\right]$$

Kernel Coupling
of weights

Proposition (Kernel Coupling Preserves Universality) *Let $\kappa : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ be a positive definite and symmetric universal kernel with associated RKHS $\mathcal{H}_\kappa$ and define the integral operator*

$$T_\kappa : C(\mathcal{Y}, \mathbb{R}^n) \to C(\mathcal{Y}, \mathbb{R}^n),$$
$$f \mapsto \int_{\mathcal{Y}} \kappa(y, z) f(z) dz.$$

If $\mathcal{A} \subseteq C(\mathcal{Y}, \mathbb{R}^n)$ is dense, then $T_\kappa(\mathcal{A}) \subset C(\mathcal{Y}, \mathbb{R}^n)$ is also dense.

Computing the kernel integral

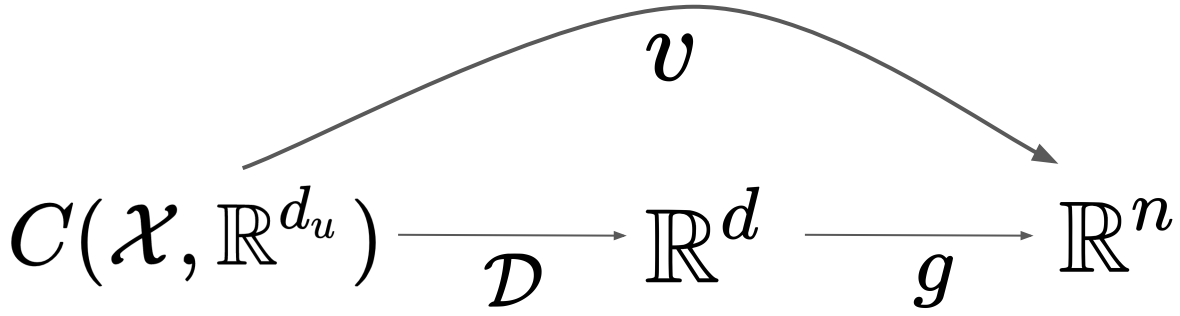
- **Option 1:** Monte-Carlo

$$\int_{\mathcal{Y}} \kappa(y,z)g(z)dz \approx \frac{\mathrm{vol}(\mathcal{Y})}{N} \sum_{i=1}^{N} \kappa(y,z_i)g(z_i)$$

- **Option 2:** (Gaussian) Quadrature

$$\int_{\mathcal{Y}} \kappa(y,z)g(z)dz \approx \sum_{i=1}^{N} w_i \kappa(y,z_i)g(z_i)$$

Computing feature encoding $v : C(\mathcal{X}, \mathbb{R}^{d_u}) \to \mathbb{R}^n$



Spectral projection

$$u \mapsto \begin{pmatrix} \langle u, \phi_1 \rangle \\ \vdots \\ \langle u, \phi_d \rangle \end{pmatrix}$$

Is this expressive enough?

Universal class of functions

$g : \mathbb{R}^d \to \mathbb{R}^n$

Spectral Feature Encoder $v : C(\mathcal{X}, \mathbb{R}^{d_u}) \to \mathbb{R}^n$

Proposition: Let $\mathcal{A}_m \subset C(\mathbb{R}^m, \mathbb{R}^n)$ be dense and $\{\phi_i\}_{i=1}^{\infty}$ such that for some $\mathcal{U} \subset C(\mathcal{X}, \mathbb{R}^{d_u})$, $\sum_{i=1}^{\infty} \langle u, \phi_i \rangle \phi_i$ converges to u uniformly over $\mathcal{U}$.

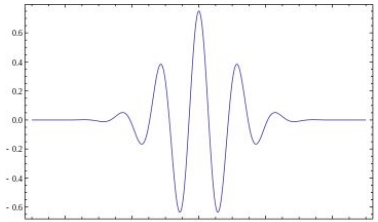
Let $\mathcal{D}_m : \mathcal{U} \to \mathbb{R}^m$ denote the projection onto $\{\phi_1, \ldots, \phi_m\}$. Then for any continuous functional $h : \mathcal{U} \to \mathbb{R}^n$, and any $\epsilon > 0$, there exists m and $f \in \mathcal{A}_m$ such that

$$\sup_{u \in \mathcal{U}} \left\| h(u) - f \circ \mathcal{D}_m(u) \right\| < \epsilon$$

Another Feature Encoder

- Given a wavelet $\psi(x)$, $j \in \mathbb{Z}$, and an element of a rotation group $r \in G$, denote

$$\psi_\lambda(x) = 2^{d_x j}\psi(2^{-j}r^{-1}x), \quad \lambda = (j, r)$$



- For a path $p = (\lambda_1, \ldots, \lambda_m)$, and a low pass filter ϕ, define

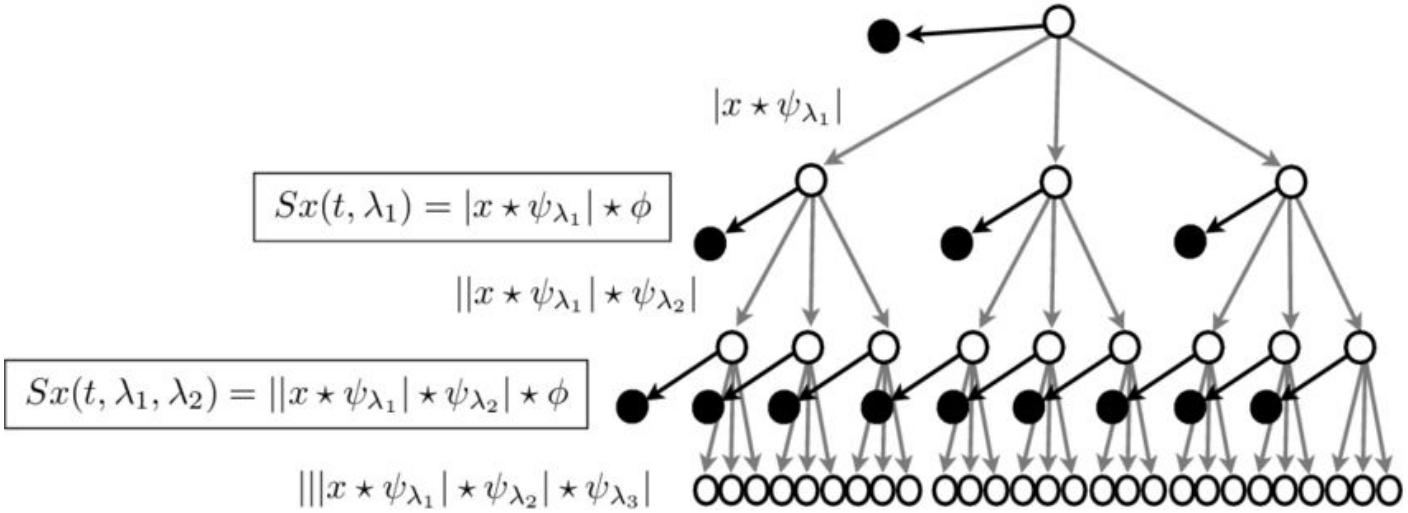
$$S[p]u = \left|u \star \psi_{\lambda_1}\right| \star \psi_{\lambda_2} \left| \cdots \right| \star \psi_{\lambda_m} \left| \star \phi(x)\right.$$

- The Scattering Transform of maximum scale J, rotations G with |G|=L, and depth m_0 is given by

$$\{S[p]u \mid p = (\lambda_1, \ldots, \lambda_{m_0}),\ \lambda_i = (j, r),\ j > -J, r \in G\}$$

[1]Bruna, Joan, and Stéphane Mallat. "Invariant scattering convolution networks." *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013): 1872-1886.

The Wavelet Scattering Transform



- Fast algorithm based on FFTs (can be applied to varying resolutions)
- Robust to small deformations of the input/corruption by small noise.

This works very well in practice

Recap and location of trainable parameters

$$\mathcal{F}(u)(y) = \sum_{i=1}^{n} \varphi(y)_i v(u)_i$$
$$= \mathbb{E}_{\varphi(y)}[v(u)]$$

$$\varphi(y) = \mathrm{softmax}\left(\int_{\mathcal{Y}} k_\theta(y, z) g_\theta(z) dz \right)$$

$$v(u) = f_\theta \circ \mathcal{D}_d(u)$$

Comparison to DeepONets

- We can recover the DeepONet architecture from our model as follows
  - Remove normalization and kernel coupling from construction of φ
  - In feature encoding, choose *m* input function locations and use pointwise evaluation functionals for $\mathcal{D} : C(\mathcal{X}, \mathbb{R}^{d_x}) \to \mathbb{R}^d$ with $d = m \cdot d_u$

$$\mathcal{D}(u) = \begin{bmatrix} u(x_1) \\ \vdots \\ u(x_m) \end{bmatrix}$$

Lu, Lu, et al. "Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators." *Nature Machine Intelligence* 3.3 (2021): 218-229.

Comparison to Neural Operators

- Connection between DeepONets and Neural Operators* can be applied after transformation from LOCA to DeepONet

- The scattering transform itself is a collection of layered compositions of wavelet convolutions and pointwise nonlinearities (complex modulus)
    - This can be seen as several stacked Neural Operator layers with fixed weights

* Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces. arXiv preprint arXiv:2108.08481, 2021.

Properties of the Operator Learning methods

We consider **three properties** that **Operator Learning** methods should possess in order to become useful tools for engineering applications,

- **Data efficiency**: Operator Learning methods should be data efficient, meaning that they should be able to provide a small median error and error spreads while being trained using a small number of labeled data.

- **Robustness**: Operator Learning methods should be robust, meaning that their accuracy should not be drastically affected if either the the testing or both the training and testing datasets are corrupted by noise.

- **Generalization:** Operator Learning methods should generalize well to the testing dataset and ideally would behave well under small distribution shifts.

Comparison Metrics

For making comparisons with the other methods we employ the following metrics:

- Relative $\mathcal{L}_2$ error for each individual input/output function pair separately:

$$e^i = \frac{\|s^i(y) - \hat{s}^i(y)\|_2^2}{\|s^i(y)\|_2^2},$$

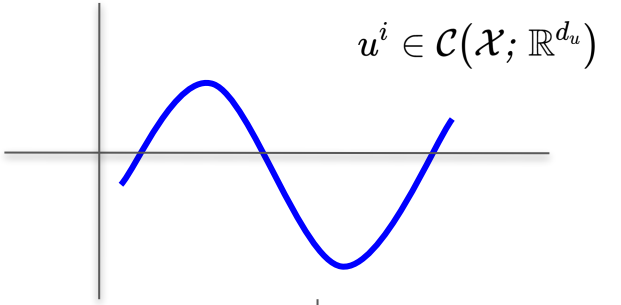
  to create an error vector

$$\mathbf{e} = \left[e^i\right]_{i=1}^N$$

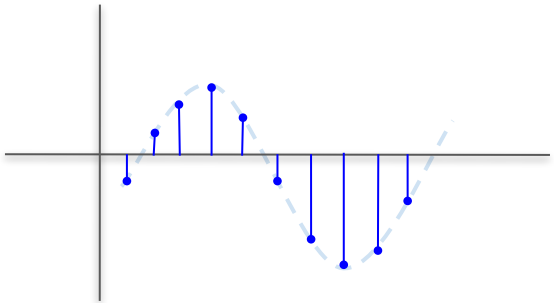
  where *N* the number of input/output function pairs. Then we compute statistics for the error vector, such as the median, percentiles and outliers.

- To promote fair comparisons between different methods, we try to make sure that the wall-clock training time for each method, as well as the number of trainable parameters of each model are close or at least comparable.

Sampling output labels

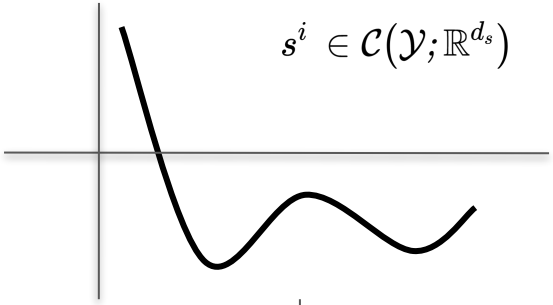


Use all *m* available points

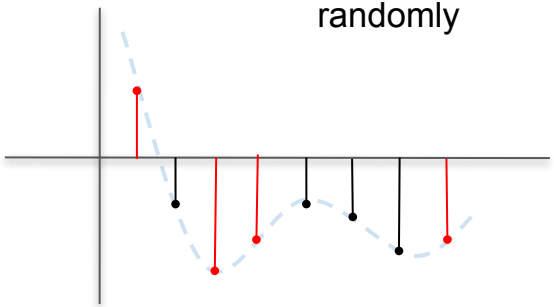


$(u^i(x_1^i), \ldots, u^i(x_m^i))$

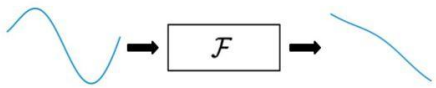
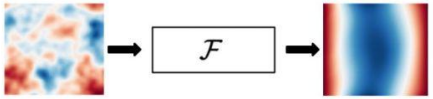
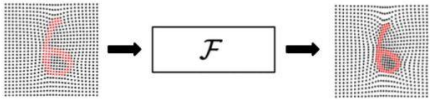
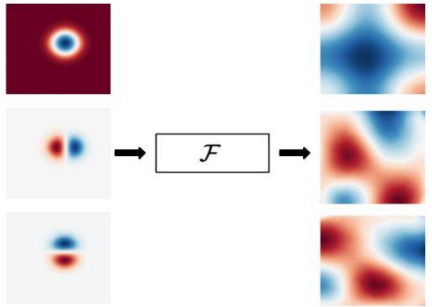
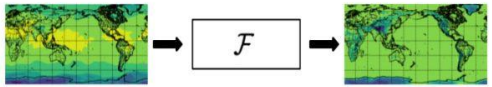
We use a percentage, rounded to the nearest half-integer, of the available output function measurements per example for training.



Choose *P* out of *M* available points randomly



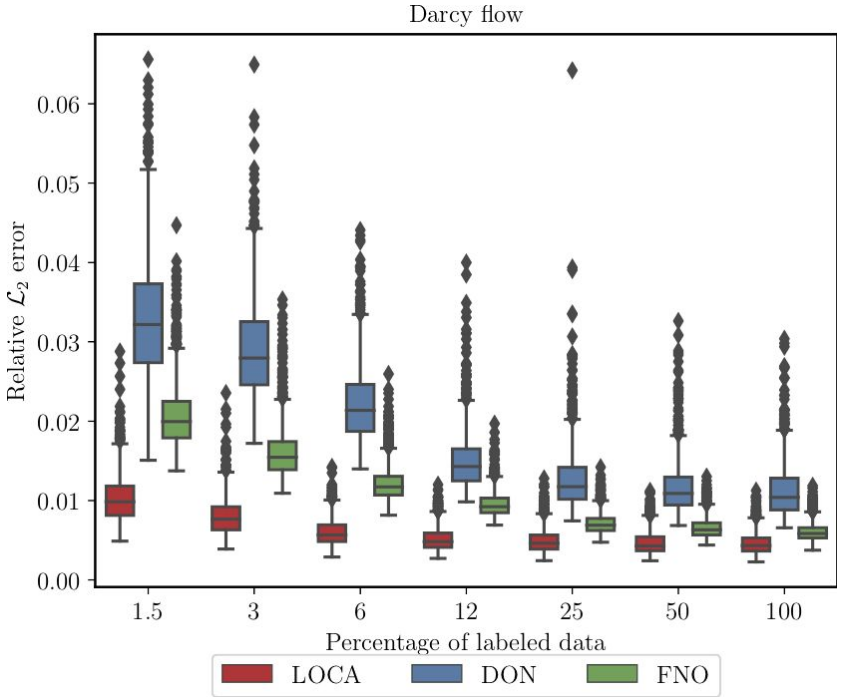
$(s^i(y_1^i), \ldots, s^i(y_M^i))$

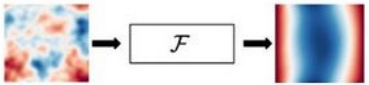
Example	Input Function $u(x)$	Output Function $s(y)$	Visualization of the operator $\mathcal{F}$
Antiderivative	Smooth Univariate Function $u(x) \sim GP(0, o \exp\left(\frac{\|x-x'\|}{l}\right)$	Antiderivative solution $\frac{ds(x)}{dx} = u(x), \quad s(x) = s_0 + \int_0^x u(\tau)d\tau$	
Darcy Flow	Random permeability $u_0 \sim \mathcal{N}(0, 7^{3/2}(-\Delta + 49I)^{1.5})$ $u(x) = \exp(u_0 cos(x))$	Pressure field $\nabla \cdot (u(x)\nabla s(x)) = f(x)$	
Mechanical MNIST	Initial displacement vector field $v_1(x; d_1), v_2(x; d_1)$	Later displacement vector field $v_1(x; d_2), v_2(x; d_2)$	
Shallow Water Equations	Random initial condition $\rho(0, x_1, x_2) = 1 + h \exp^{-((x_1-\xi)^2+(x_2-\zeta)^2)/w}$ $v_1(0, x_1, x_2) = v_1(dt, x_1, x_2),$ $v_2(0, x_1, x_2) = v_2(dt, x_1, x_2)$ $h = \mathcal{U}(1.5, 2.5)$ $w = \mathcal{U}(0.002, 0.008)$ $\xi = \mathcal{U}(0.4, 0.6)$ $\zeta = \mathcal{U}(0.4, 0.6)$	Solution at specified time instances $\frac{\partial \vec{U}}{\partial t} + \frac{\partial \vec{F}}{\partial x_1} + \frac{\partial \vec{G}}{\partial x_2} = 0$ $\vec{U} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \end{pmatrix}, \quad \vec{F} = \begin{pmatrix} \rho u \\ \rho u^2 + \frac{1}{2} g \rho^2 \\ \rho u v \end{pmatrix}, \quad \vec{G} = \begin{pmatrix} \rho v \\ \rho u v \\ \rho v^2 + \frac{1}{2} g \rho^2 \end{pmatrix}$	
Climate Modeling	Surface air temperature $T(x)$	Surface air pressure $P(y)$	

Data Efficiency

- In many engineering applications, **acquiring a large number of labeled data** for training is either **expensive** or even **infeasible**.
  - We investigate the performance of the different models in the **small labeled data** regime.
  - We examine the error vector statistics while **increasing, each time, the percentage** of the available labels used for training.
- An important aspect of a data efficiency study is considering the **presence of outliers** in the error vector. Outliers quantify the **worst case prediction** for the particular dataset.
- We **investigate the effect** of the Kernel Coupled Attention in the performance of the LOCA method in the small data regime.
  - We consider a **small number** of input/output function pairs and labeled data for each output function.
  - We perform an experiment, **with and without** the Kernel Coupled Attention mechanism to examine its effect.

Data Efficiency



Darcy Flow	Random permeability $u_0 \sim \mathcal{N}(0, 7^{3/2}(-\Delta + 49I)^{1.5})$ $u(x) = \exp(u_0 cos(x))$	Pressure field $\nabla \cdot (u(x)\nabla s(x)) = f(x)$	

Effect of Kernel Coupling in Low Data Regime

- Kernel Coupling seems to be necessary for good performance when number of output function measurements per example is small

Model \ Test Error Metric	Mean	Standard deviation	Minimum	Maximum
LOCA without KCA	0.463	0.184	0.406	3.023
LOCA with KCA	**0.017**	**0.004**	**0.008**	**0.041**

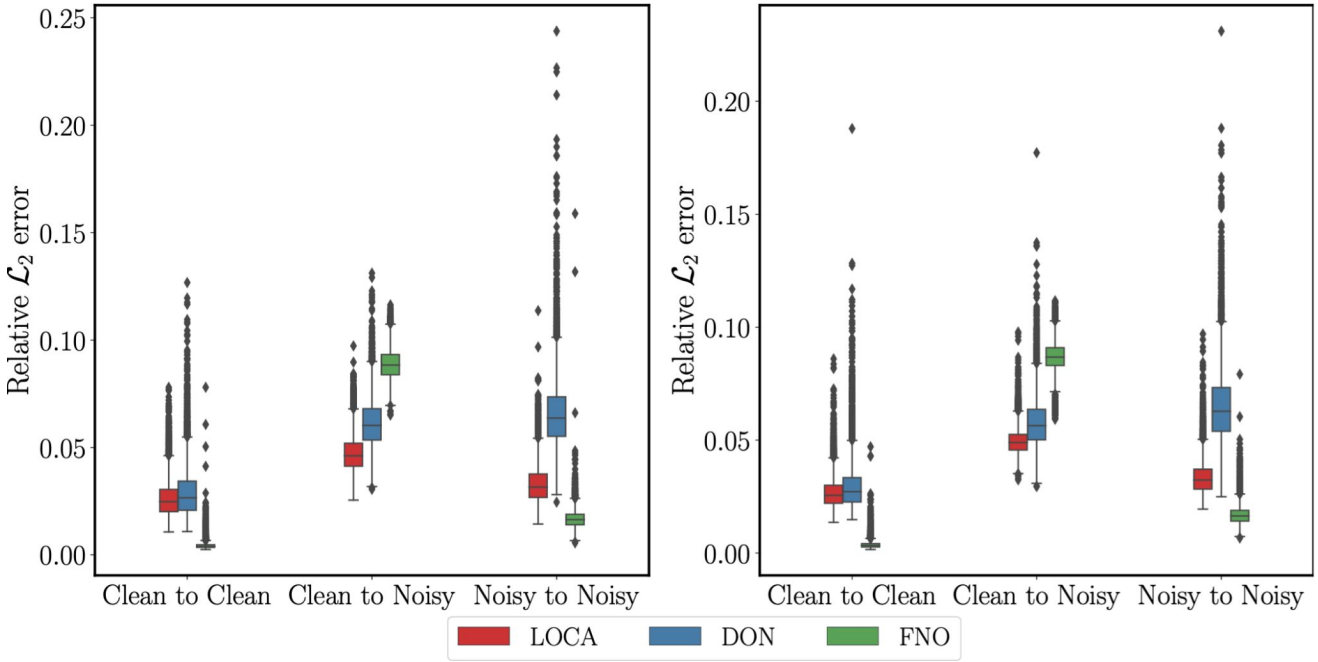
Performance for Darcy equation with and without KCA when using 1.5% of available output function measurements per training example for 200 examples.

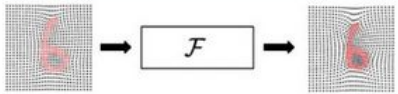
Robustness

- Operator learning can be a powerful tool for cases where:
  - We have access to **clean** simulation data for training and wish to deploy the model on **clean** simulation data as well (Clean-to-Clean scenario).
  - We have access to **clean** simulation data for training, but wish to deploy the model on **noisy** experimental data (Clean-to-Noisy scenario).
  - We have access to **noisy** simulation data for training and wish to deploy the model on **noisy** data as well (Noisy-to-Noisy scenario).
- We quantify the ability of the different Operator Learning models **to handle noise in the data** by measuring the **percentage increase in mean error between clean and noisy data scenarios**.
- The **presence of outliers** is also used to assess the different model performance as they provide a quantification of the **worst case scenario** in our predictions.
- For the experiments, we consider **7% of the available output function measurements** as training labels.
- For corrupting the data, we consider **Gaussian noise** sampled from $\mathcal{N}(0, .15I)$.

Robustness

MMNIST errors for different noise scenarios



Mechanical MNIST	Initial displacement vector field $v_1(x;d_1), v_2(x;d_1)$	Later displacement vector field $v_1(x;d_2), v_2(x;d_2)$	

Robustness

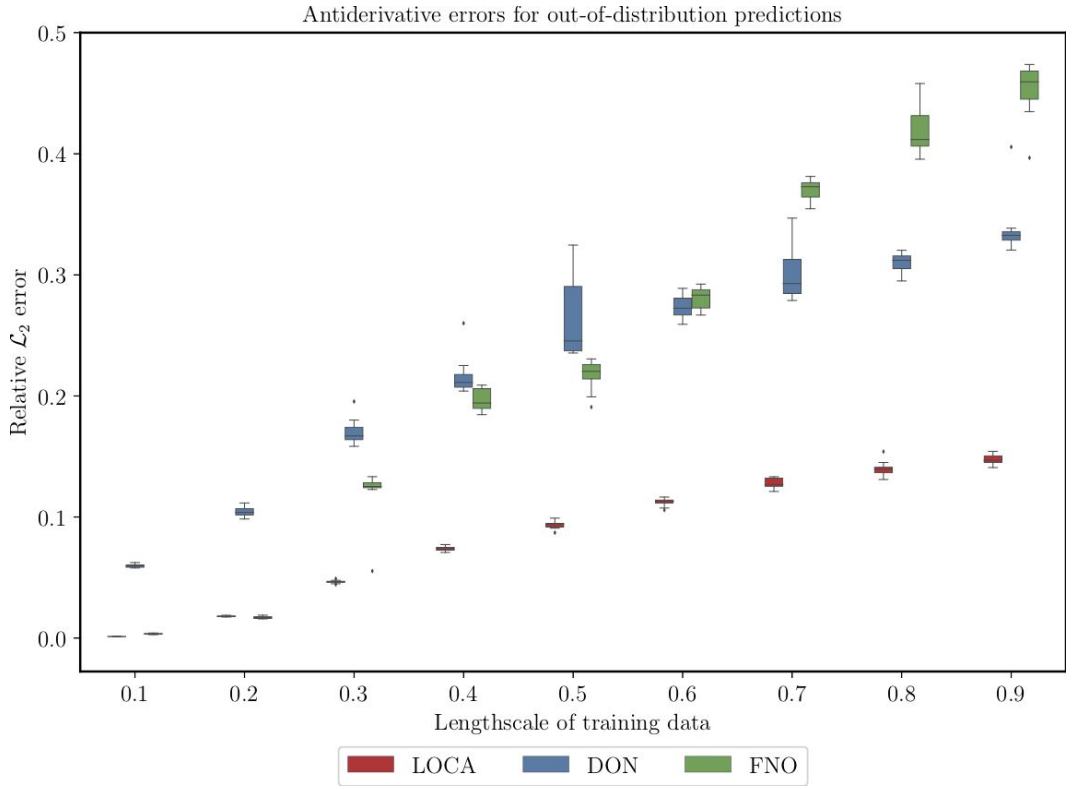
Noise scenario and error \ Model	FNO	DON	LOCA
Clean to Clean (CC)	**[0.004, 0.003]**	[0.028, 0.029]	[0.026, 0.026]
Clean to Noisy (CN)	[0.088, 0.087]	[0.061, 0.057]	**[0.047, 0.049]**
Noisy to Noisy (NN)	**[0.016, 0.016]**	[0.065, 0.064]	[0.032, 0.033]
Percentage error increase from CC to CN	[1,930 %, 2,238 %]	[112%, 96%]	**[80%, 85%]**
Percentage error increase from CC to NN	[280 %, 347%]	[128%,120%]	**[26%, 25%]**

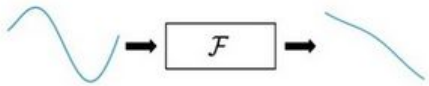
Relative errors for vertical and horizontal displacements in MMNIST experiment for each model with percentage changes from clean baseline.

Generalization

- The ultimate goal of data-driven methods is to perform well outside of the data set they are trained on.
- To study this property, we examine the testing error statistics of the different models under a incremental distribution shift of the training data set:
  - Learn the Antiderivative operator where training and the testing data sets are sampled from a Gaussian process.
  - Fix the length-scale of the testing data set distribution to 0.1 and have the length-scale of the training distribution range from 0.1 to 0.9
- Run the experiment 10 times for each length-scale:
  - Choose a different random network initializations each time
  - Use the mean error of each run to create an error vector for each length-scale and get its statistics.

Generalization

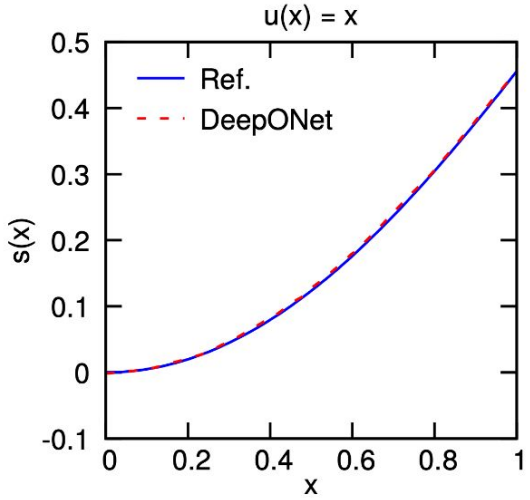


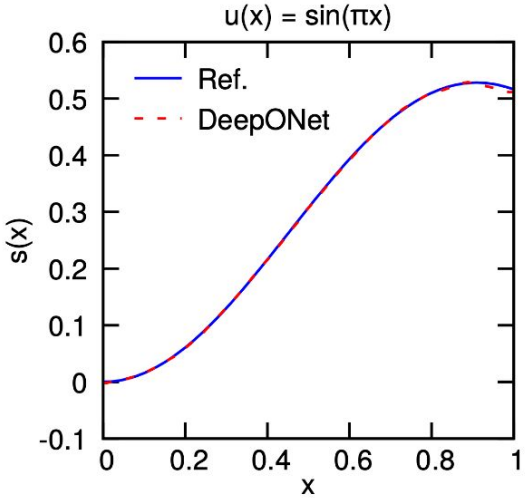
Antiderivative	Smooth Univariate Function $u(x) \sim GP(0, o \exp\left(\frac{\|x-x'\|}{l}\right)$	Antiderivative solution $\frac{ds(x)}{dx} = u(x), \quad s(x) = s_0 + \int_0^x u(\tau)d\tau$	

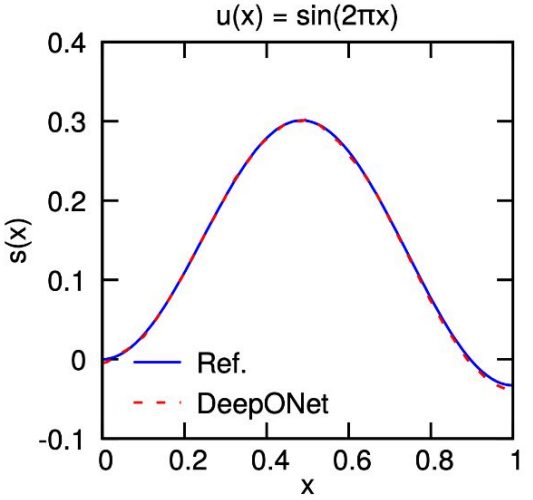
A pedagogical example

$$\frac{ds(x)}{dx} = u(x), \quad x \in [0,1]$$
$$s(0) = 0$$

$$G : u(x) \longrightarrow s(x) = s(0) + \int_0^x u(t)dt, \quad x \in [0,1]$$



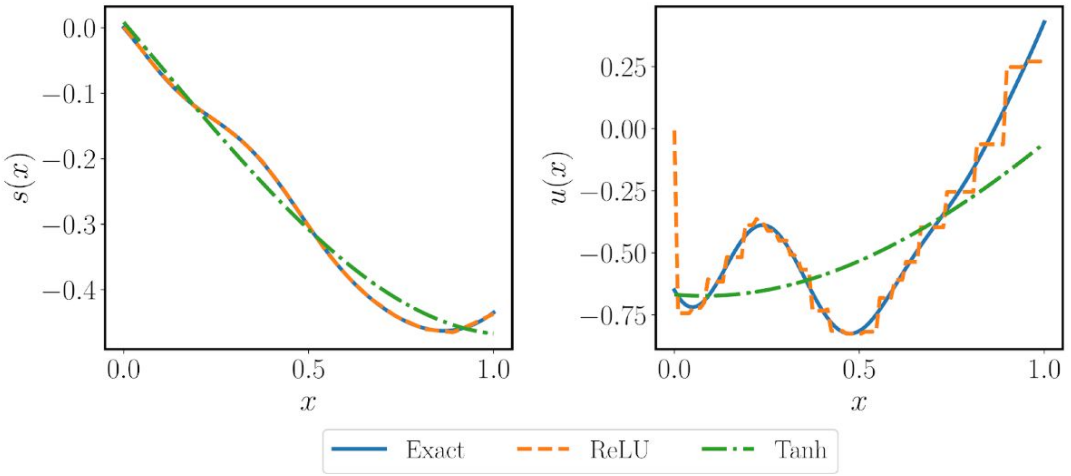




A consistency check

$$\frac{ds(x)}{dx} = u(x), \quad x \in [0,1]$$
$$s(0) = 0$$

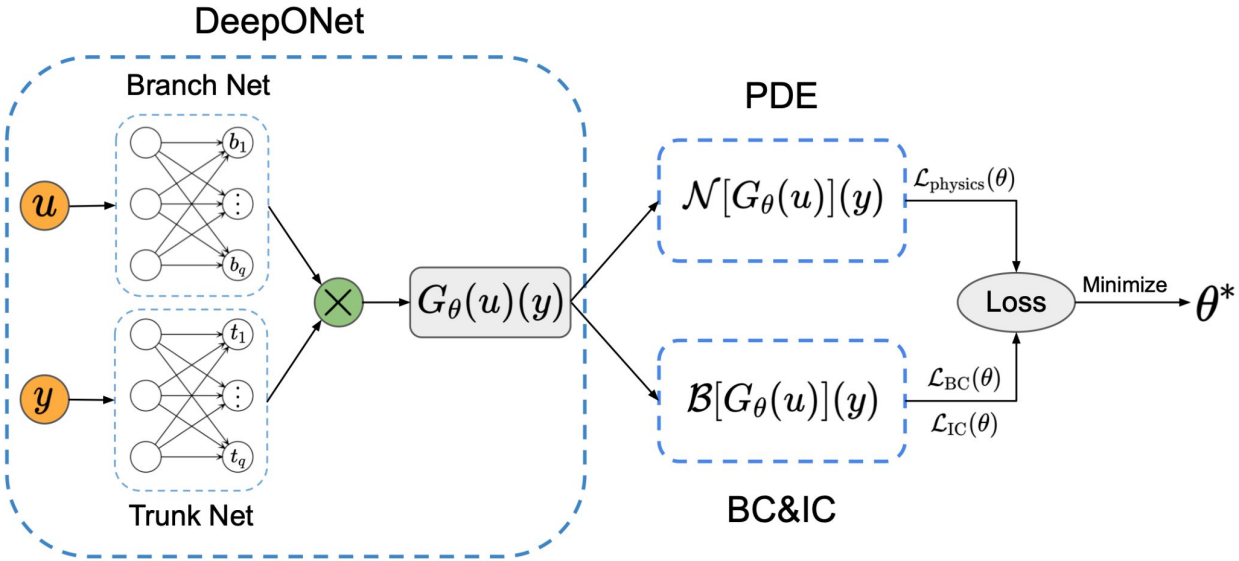
$$G : u(x) \longrightarrow s(x) = s(0) + \int_0^x u(t)dt, \quad x \in [0,1]$$



<u>DeepOnets are great, but:</u>

- They require large training data-sets consisting of paired input-output observations.
- Their predictions may be incompatible with respect to the true operator that generated the data.

Physics-informed DeepONets



$$\mathcal{L}_{\text{operator}}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left|G_\theta\left(u^{(i)}\right)\left(y^{(i)}\right) - s^{(i)}\left(y^{(i)}\right)\right|^2$$

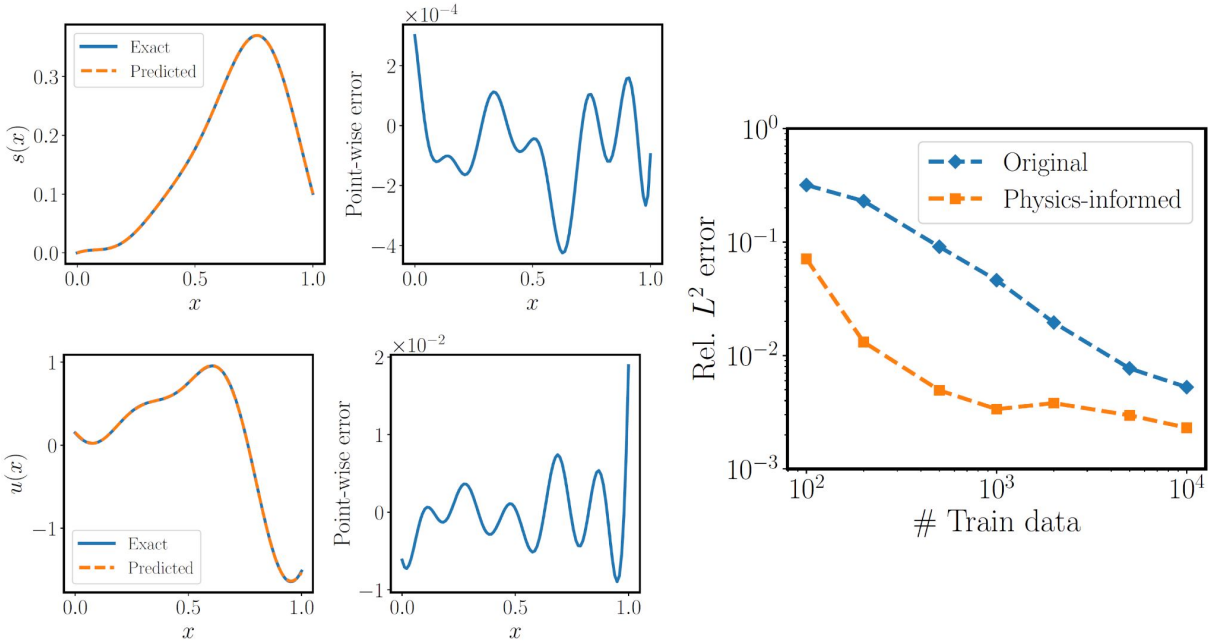
$$\mathcal{L}_{\text{physics}}(\theta) = \frac{1}{NQm}\sum_{i=1}^{N}\sum_{j=1}^{Q}\sum_{k=1}^{m}\left|\mathcal{N}\left(u^{(i)}(x_k), G_\theta\left(u^{(i)}\right)\left(y_j^{(i)}\right)\right)\right|^2$$

Wang, S., Wang, H., & Perdikaris, P. (2021).Learning the solution operator of parametric partial differential equations with physics-informed DeepOnets

Physics-informed DeepONets

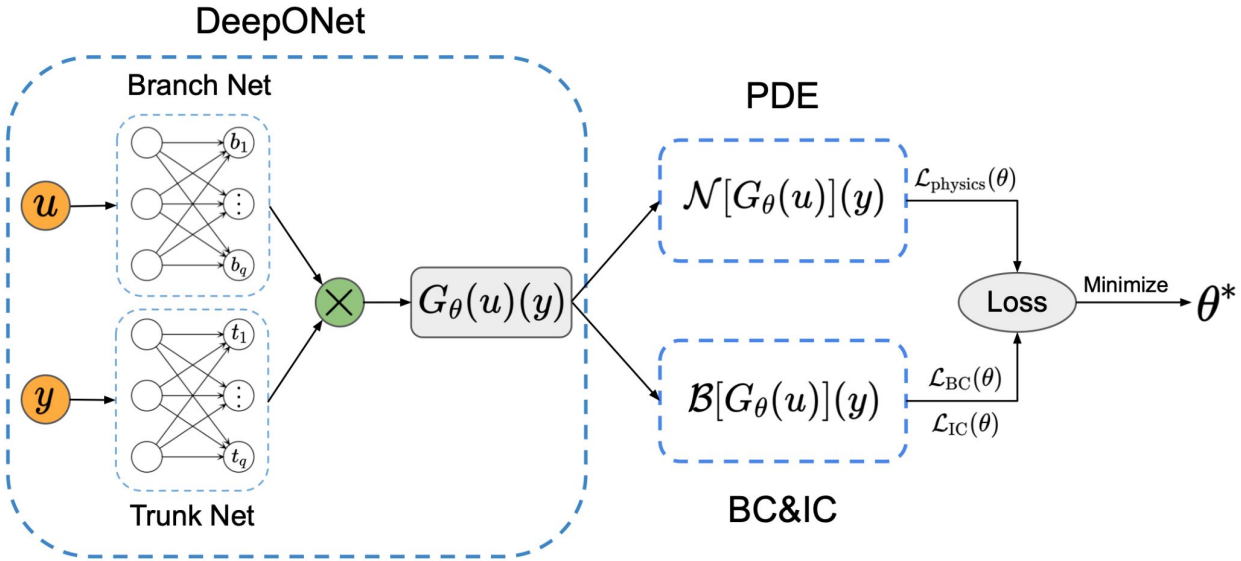
$$\frac{ds(x)}{dx} = u(x), \quad x \in [0,1]$$
$$s(0) = 0$$

$$G : u(x) \longrightarrow s(x) = s(0) + \int_0^x u(t)dt, \quad x \in [0,1]$$



Wang, S., Wang, H., & Perdikaris, P. (2021).Learning the solution operator of parametric partial differential equations with physics-informed DeepOnets

Physics-informed DeepONets



$$\mathcal{L}_{\text{operator}}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left|G_\theta\left(u^{(i)}\right)\left(y^{(i)}\right) - s^{(i)}\left(y^{(i)}\right)\right|^2$$

$$\mathcal{L}_{\text{physics}}(\theta) = \frac{1}{NQm}\sum_{i=1}^{N}\sum_{j=1}^{Q}\sum_{k=1}^{m}\left|\mathcal{N}\left(u^{(i)}(x_k), G_\theta\left(u^{(i)}\right)\left(y_j^{(i)}\right)\right)\right|^2$$

A striking observation:

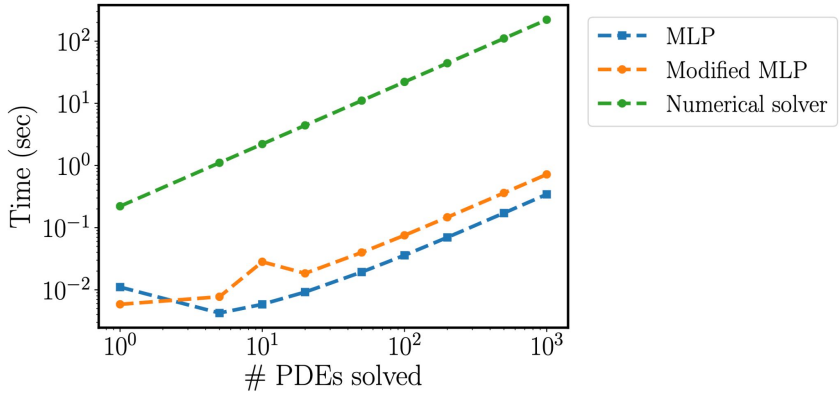
- Only knowledge of the initial and boundary conditions is enough to train the model.
- No additional measurements for the target output functions are required.

PIDONs can be trained even in the absence of paired input-output observations!

Wang, S., Wang, H., & Perdikaris, P. (2021).Learning the solution operator of parametric partial differential equations with physics-informed DeepOnets

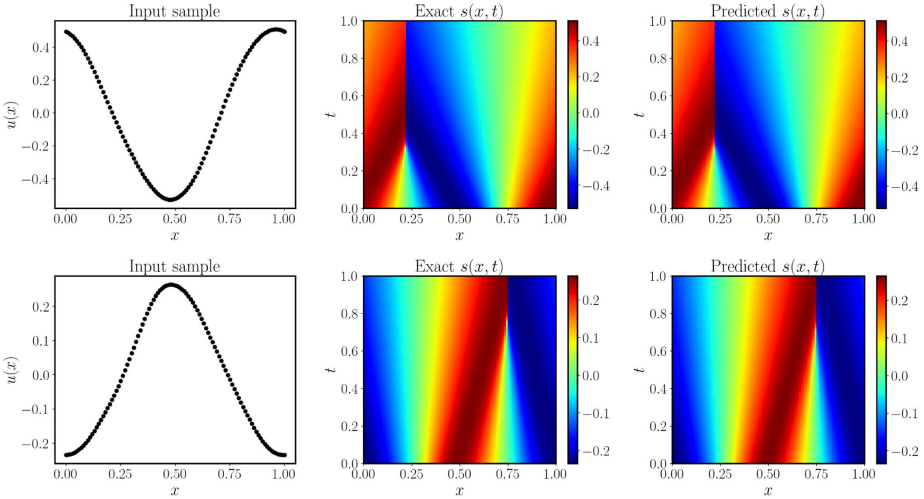
Self-supervised learning

Governing law	Equation form	Random input	Test error
Linear ODE	$\frac{ds(x)}{dx} = u(x)$	Forcing terms	$0.33 \pm 0.32\%$
Diffusion reaction	$\frac{\partial s}{\partial t} = D\frac{\partial^2 s}{\partial x^2} + ks^2 + u(x)$	Source terms	$0.45 \pm 0.16\%$
Burgers'	$\frac{\partial s}{\partial t} + s\frac{\partial s}{\partial x} - \nu\frac{\partial^2 s}{x^2} = 0$	Initial conditions	$1.38 \pm 1.64\%$
Advection	$\frac{\partial s}{\partial t} + u\frac{\partial s}{\partial x} = 0$	Variable coefficients	$2.24 \pm 0.68\%$
Eikonal	$\| \nabla s \|_2 = 0$	Domain geometries	$0.42 \pm 0.11\%$



Example: Learning the solution operator G from the initial condition u(x) to the full solution s(x,t) of the Burgers PDE.

$$\frac{ds}{dt} + s\frac{ds}{dx} - \nu\frac{d^2 s}{dx^2} = 0, \quad (x,t) \in (0,1) \times (0,1]$$
$$s(x,0) = u(x), \quad x \in (0,1),$$
$$\nu = 10^{-4}$$



PDE-constrained optimization

$$\begin{aligned} \min \quad & \mathcal{J}(\boldsymbol{u}, \boldsymbol{s}) \\ \text{s.t.} \quad & \mathcal{E}(\boldsymbol{u}, \boldsymbol{s}) = 0 \\ & \boldsymbol{u} \in U_{ad}, \boldsymbol{s} \in S_{ad} \end{aligned}$$

$\boldsymbol{u}$: Input/control functions

$\boldsymbol{s}$: Target output functions

$\mathcal{J}(\boldsymbol{u}, \boldsymbol{s})$: Cost functional

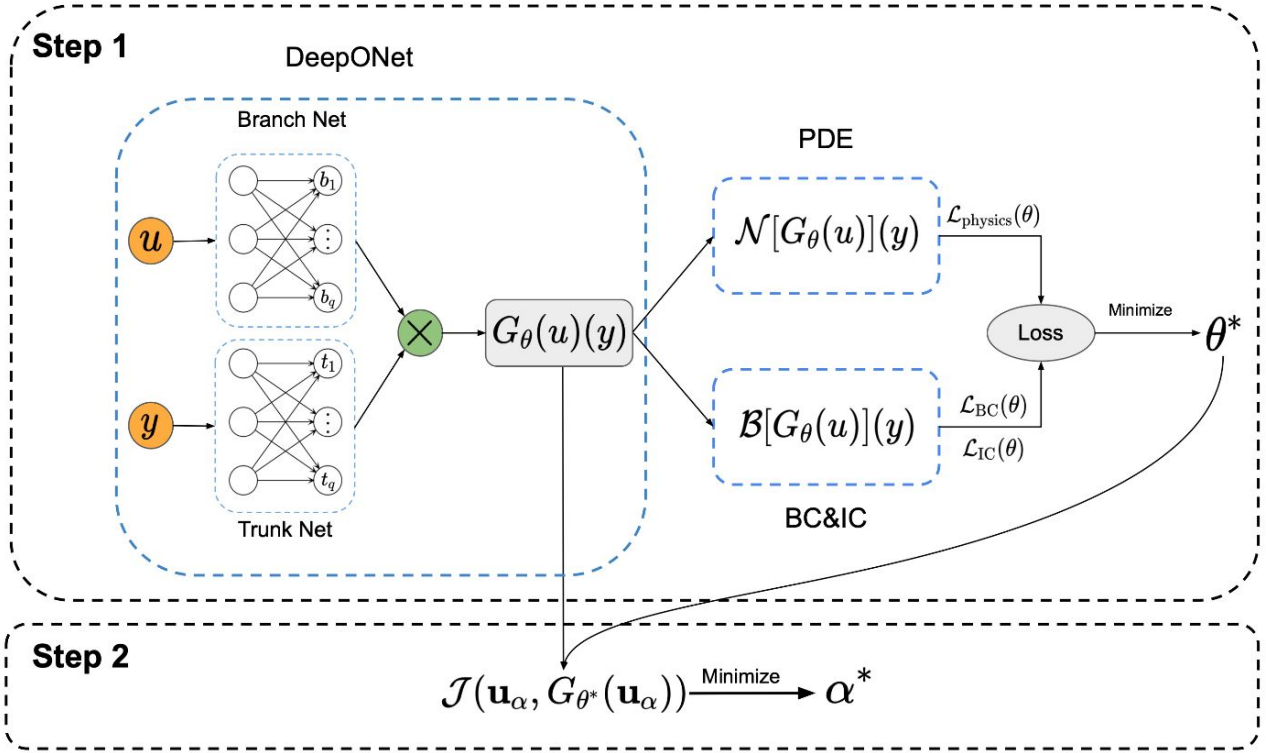
$\mathcal{E}(\boldsymbol{u}, \boldsymbol{s})$: PDE constraints

U_{ad} : Admissible space of input functions

S_{ad} : Admissible space of output functions

Challenges: Infinite-dimensional inputs, cost of solving the PDE.

PDE-constrained optimization



Train a self-supervised physics-informed DeepONet to learn the PDE solution operator:

$$G_\theta : \mathcal{U} \to \mathcal{S}$$

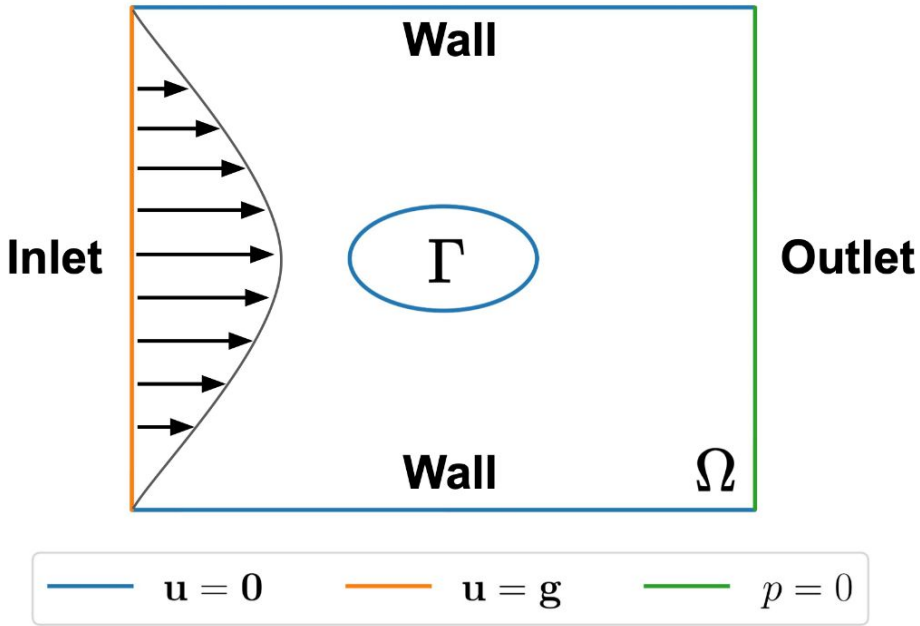
Use G_{θ^*} as a differentiable surrogate to minimize J.

* All gradients (i.e. $\partial\mathcal{L}/\partial\boldsymbol{\theta}$, $\partial\mathcal{J}/\partial\boldsymbol{\alpha}$ and $\partial\boldsymbol{s}/\partial\boldsymbol{x}$) are computed using automatic differentiation).

Drag minimization of obstacles in Stokes flow

$$-\Delta \boldsymbol{u} + \nabla p = 0, \quad (x, y) \in \Omega/\Gamma,$$
$$\nabla \cdot \boldsymbol{u} = 0, \quad (x, y) \in \Omega/\Gamma,$$

$$\boldsymbol{u} = \boldsymbol{0}, \quad (x, y) \in \Lambda_1 \cup \partial\Gamma,$$
$$\boldsymbol{u} = \boldsymbol{g}, \quad (x, y) \in \Lambda_2$$
$$p = 0, \quad (x, y) \in \Lambda_3,$$



$$\partial\Gamma = \partial\boldsymbol{\Gamma}(\phi) = (a\cos(\phi) + \frac{1}{2}, b\sin(\phi) + \frac{1}{2}), \quad \phi \in [0, 2\pi)$$

Drag minimization of obstacles in Stokes flow

Step 1: Train a physics-informed DeepONet to learn the solution operator:

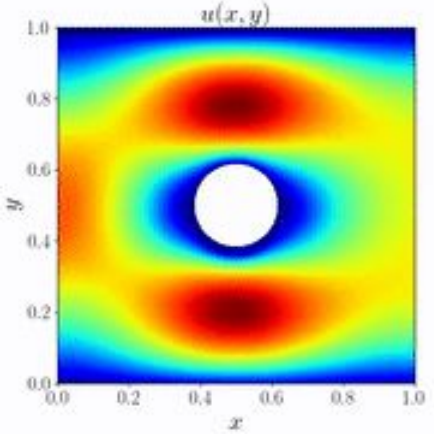
$$\partial\mathbf{\Gamma} \xrightarrow{G_{\boldsymbol{\theta}}} [G_{\boldsymbol{\theta}}^{(u)}, G_{\boldsymbol{\theta}}^{(v)}, G_{\boldsymbol{\theta}}^{(p)}]$$

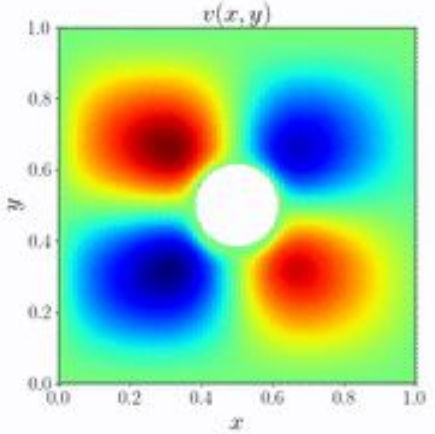
Step 2: Use G_{θ^*} as a differentiable surrogate to minimize J via gradient descent:

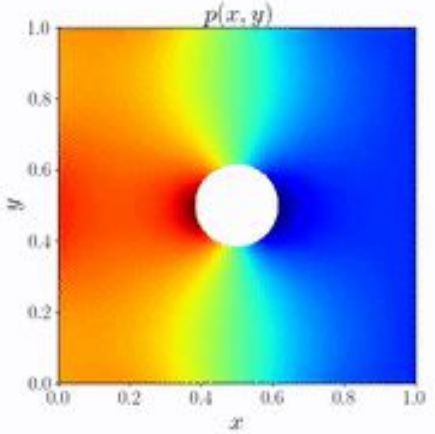
$$\begin{aligned}\mathcal{J}(\partial\mathbf{\Gamma}) = \int_{\Omega/\Gamma} &\left(\frac{\partial G_{\boldsymbol{\theta}^*}^{(u)}(\partial\mathbf{\Gamma})}{\partial x}\right)^2 + \left(\frac{\partial G_{\boldsymbol{\theta}^*}^{(u)}(\partial\mathbf{\Gamma})}{\partial y}\right)^2 \\ &+ \left(\frac{\partial G_{\boldsymbol{\theta}^*}^{(v)}(\partial\mathbf{\Gamma})}{\partial x}\right)^2 + \left(\frac{\partial G_{\boldsymbol{\theta}^*}^{(v)}(\partial\mathbf{\Gamma})}{\partial y}\right)^2 \mathrm{d}x \\ &+ \alpha\left|\mathrm{Vol}(\Gamma) - \mathrm{Vol}\left(\Gamma_0\right)\right|^2 \\ &+ \beta\sum_{j=1}^{2}\left|\mathrm{Bc}(\Gamma) - \mathrm{Bc}\left(\Gamma_0\right)\right|^2,\end{aligned}$$

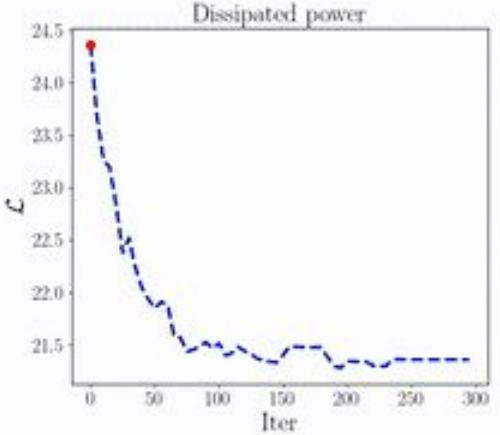
where $\partial\mathbf{\Gamma} = [\partial\Gamma(\phi_1), \partial\Gamma(\phi_2), \ldots, \partial\Gamma(\phi_m)]$ denotes an input closed curve evaluated at evenly-spaced grid points $\{\phi\}_{i=1}^{m}$ in $[0, 2\pi]$.

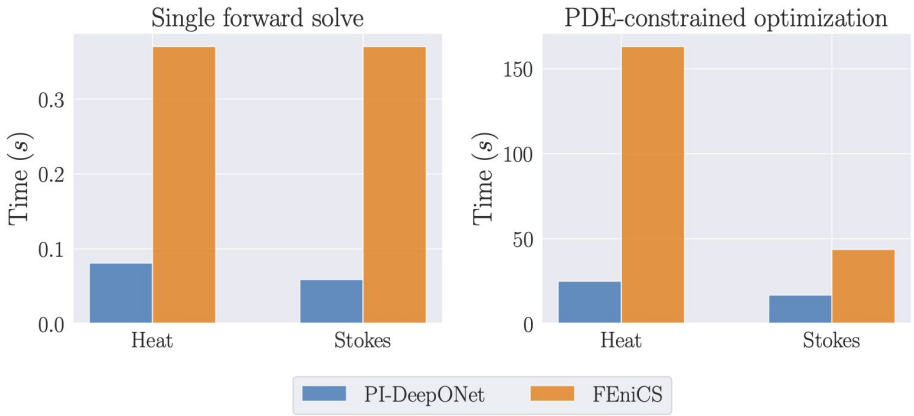
Drag minimization of obstacles in Stokes flow

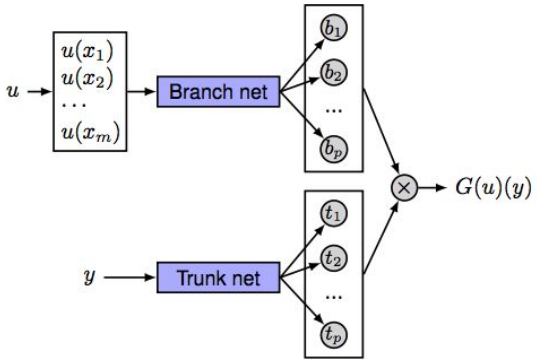


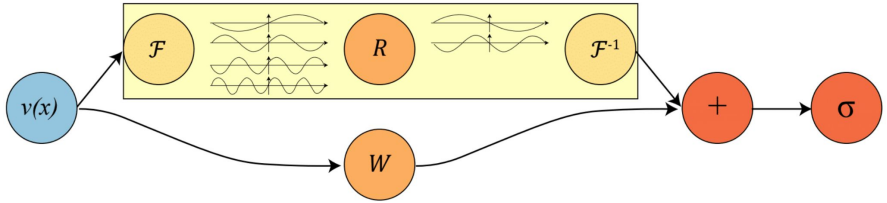


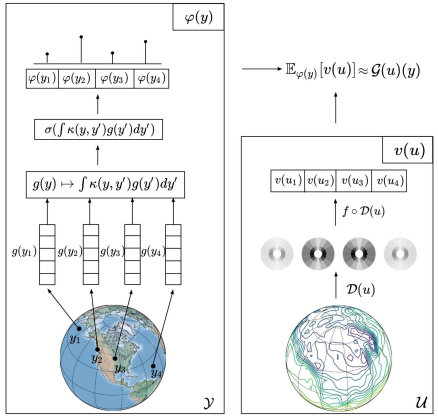


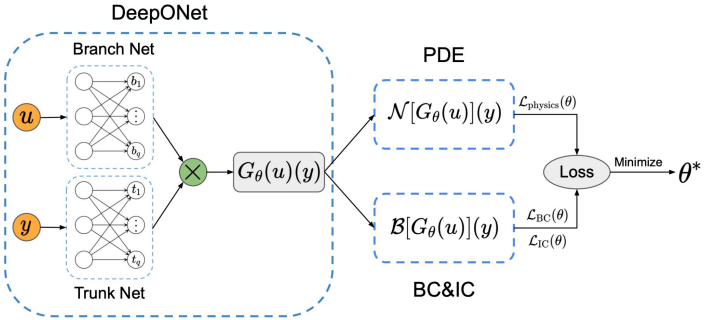





u
$u(x_1)$
$u(x_2)$
...
$u(x_m)$
Branch net
b_1
b_2
...
b_p
$G(u)(y)$
y
Trunk net
t_1
t_2
...
t_p


v(x)
$\mathcal{F}$
R
$\mathcal{F}^{-1}$
W
+
σ


$\varphi(y)$
$\varphi(y_1)$ $\varphi(y_2)$ $\varphi(y_3)$ $\varphi(y_4)$
$\sigma(\int \kappa(y,y')g(y')dy')$
$g(y) \mapsto \int \kappa(y,y')g(y')dy'$
$g(y_1)$ $g(y_2)$ $g(y_3)$ $g(y_4)$
y_1 y_2 y_3 y_4
y
$\mathbb{E}_{\varphi(y)}[v(u)] \approx \mathcal{G}(u)(y)$
$v(u)$
$v(u_1)$ $v(u_2)$ $v(u_3)$ $v(u_4)$
$f \circ \mathcal{D}(u)$
$\mathcal{D}(u)$
$\mathcal{U}$


DeepONet
Branch Net
u
b_1
b_q
y
t_1
t_q
Trunk Net
$G_\theta(u)(y)$
PDE
$\mathcal{N}[G_\theta(u)](y)$
$\mathcal{B}[G_\theta(u)](y)$
BC&IC
$\mathcal{L}_{\text{physics}}(\theta)$
$\mathcal{L}_{\text{BC}}(\theta)$
$\mathcal{L}_{\text{IC}}(\theta)$
Loss
Minimize
θ^*

Thanks!