Stony Brook University

AI for Partial Differential Equations (PDEs) and Neural Operators

Presenter: Wenhan Gao

Advisor: Yi Liu

Department of Applied Mathematics and Statistics

FAR BEYOND

Stony Brook University

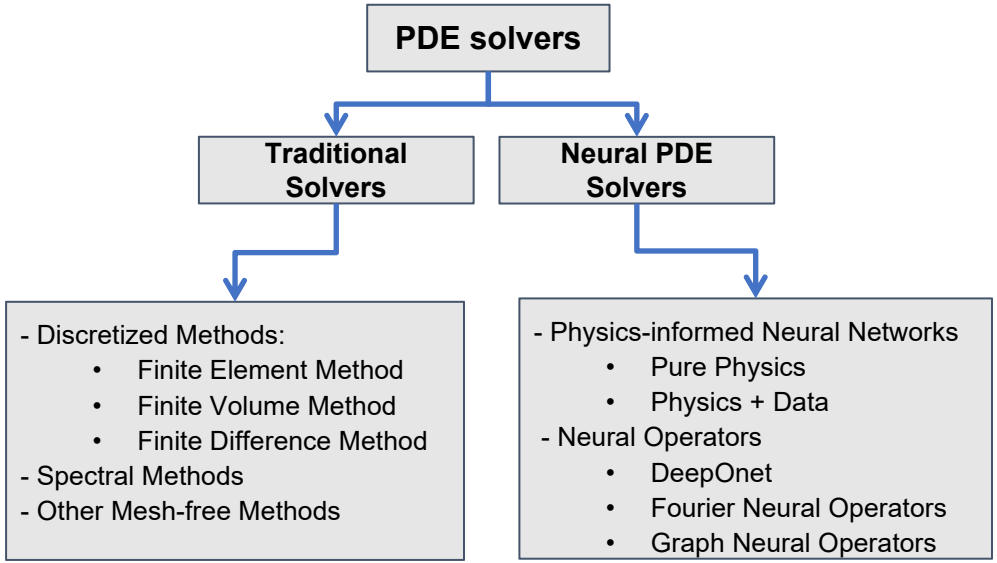
PDEs and PDE solvers

A PDE mathematically describes the behavior of a system by prescribing constraints relating partial derivatives.

Example: The Wave Equation

$$\frac{\partial^2 u}{\partial t^2} = D\nabla^2 u,$$

where D is the diffusion coefficient. We can visualize, by solving the PDE, a disturbance in the medium which will then propagate in all directions.



FAR BEYOND

Traditional PDE solvers

- Solve one instance
- Require the explicit form
- Speed-accuracy trade-off on resolution
- Slow on fine grids; fast on coarse grids
- Suffers from the Curse of Dimensionality (CoD)

PINN

- Solve one instance
- Incorporate known physics
- Can train without data
- Mesh-free
- Can be slow to train
- Lessen the CoD issue

Neural Operator PDE solver

- Learn a family of PDE
- Black-box, data-driven
- Compatible with physics-informed ideas
- Can be resolution-invariant
- Slow to train; fast to evaluate (can be several orders of magnitudes faster)
- Most neural operators suffer from CoD

Neural Networks

A feedforward neural network is given by a compositional function:

$$\phi(\boldsymbol{x};\boldsymbol{\theta}) = h_L \circ h_{L-1} \circ \ldots \circ h_1 \circ h_0(\boldsymbol{x})$$

Each h consists of a learnable linear transformation and non-linear activation functions such as Tanh, Sigmoid, ReLU.

- Universal Approximation Theorems indicate that deep neural networks, DNNs, can approximate most functions in the Sobolev space.

Intuition for Universal Approximation for those not familiar with deep learning/neural networks:

> (Extensions of) Fourier series can be universal approximators of continuous functions, on smooth enough models, they can even get spectral convergence, meaning that the error decreases exponentially. However, “the curse of dimensionality” arises; for a desired precision, the number of Fourier modes grows exponentially with respect to the input dimension. In most practical applications, such as image classification, the input is high-dimensional. Neural Networks are universal approximators that can “overcome the curse of dimensionality”.

PINNs (Solving One Instance)

Consider the following general form of a PDE for $u(\boldsymbol{x})$:

$$\begin{cases} \mathcal{D}u(\boldsymbol{x}) = f(\boldsymbol{x}), & \text{in } \Omega, \\ \mathcal{B}u(\boldsymbol{x}) = g(\boldsymbol{x}), & \text{on } \partial\Omega, \end{cases}$$

we wish to approximate $u(\boldsymbol{x})$ with a neural network, denoted by $\phi(\boldsymbol{x};\boldsymbol{\theta})$. We can train the neural network with physics-informed loss. That is, we aim to solve the following optimization problem:

$$\begin{aligned} \boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) &:= \arg\min_{\boldsymbol{\theta}} \|\mathcal{D}\phi(\boldsymbol{x};\boldsymbol{\theta}) - f(\boldsymbol{x})\|_2^2 + \lambda\|\mathcal{B}\phi(\boldsymbol{x};\boldsymbol{\theta}) - g(\boldsymbol{x})\|_2^2 \\ &= \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x}\in\Omega}\left[|\mathcal{D}\phi(\boldsymbol{x};\boldsymbol{\theta}) - f(\boldsymbol{x})|^2\right] + \lambda\mathbb{E}_{\boldsymbol{x}\in\partial\Omega}\left[|\mathcal{B}\phi(\boldsymbol{x};\boldsymbol{\theta}) - g(\boldsymbol{x})|^2\right] \\ &\approx \arg\min_{\boldsymbol{\theta}} \frac{1}{N_1}\sum_{i=1}^{N_1} |\mathcal{D}\phi(\boldsymbol{x}_i;\boldsymbol{\theta}) - f(\boldsymbol{x}_i)|^2 + \frac{\lambda}{N_2}\sum_{j=1}^{N_2} |\mathcal{B}\phi(\boldsymbol{x}_j;\boldsymbol{\theta}) - g(\boldsymbol{x}_j)|^2 \end{aligned}$$

Allocation points in the domain

Difference between the L.H.S. and R.H.S. of the differential equation. If the network predicts the ground truth, this is going to be 0.

Intuition: We penalize the neural network by the extend to which it violates the PDE/boundary/initial conditions.

Neural Operators (Solving a Family of PDEs)

- In PINNs or traditional solvers, we are interested in a function, which solves only one instance of PDEs.
- In numerous fields, such as electromagnetism, researchers seek to study the behavior of physical systems under various parameters, such as different initial conditions, boundary values, and forcing functions.
- Solving one instance at a time can be excessively time-consuming!
- Neural operators approximate the mapping from parameter function space to solution function space.
- Once trained, obtaining a solution can be several orders of magnitude faster than numerical methods.

Function: A Function is a mapping between finite-dimensional vector spaces.

Example: $f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu k}{\sigma}\right)^2}, x \in \mathbf{R}.$

Operator: An operator is a mapping between infinite-dimensional function spaces:

$$G(f(x)) = u(x).$$

Examples: Derivative Operator, Nabla Operator, Differential Operator, etc..

Parametric PDEs and the Learning Task

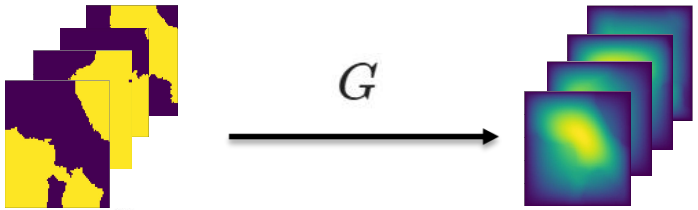
A neural operator refers to the use of neural networks to approximate or learn an operator in infinite-dimensional function spaces.

Consider a parametric PDE of the form:

$$\begin{aligned} \mathcal{N}(a,u)(x) &= f(x), \qquad x \in \Omega \\ u(x) &= 0, \qquad x \in \partial\Omega \end{aligned}$$

where $\Omega \subset \mathbb{R}^d$ is a bounded open set, $\mathcal{N}$ is a, possibly non-linear, differential operator, a is the parametric input function, f is a given fixed function in an appropriate function space determined by the structure of $\mathcal{N}$, and u is the PDE solution. A concrete example of $\mathcal{N}(a,u)(x) = f(x)$ can be $u(x)u_x(x) - a(x)u_{xx}(x) = f(x)$.

The PDE solution operator is defined as $G(a) = u$. We are interested in learning the operator $G(\cdot)$ through a given finite collection of observations of input-output pairs $\{a_j, u_j\}_{j=1}^N$, where each a_j and u_j are functions, in practice, that come with a discretization.



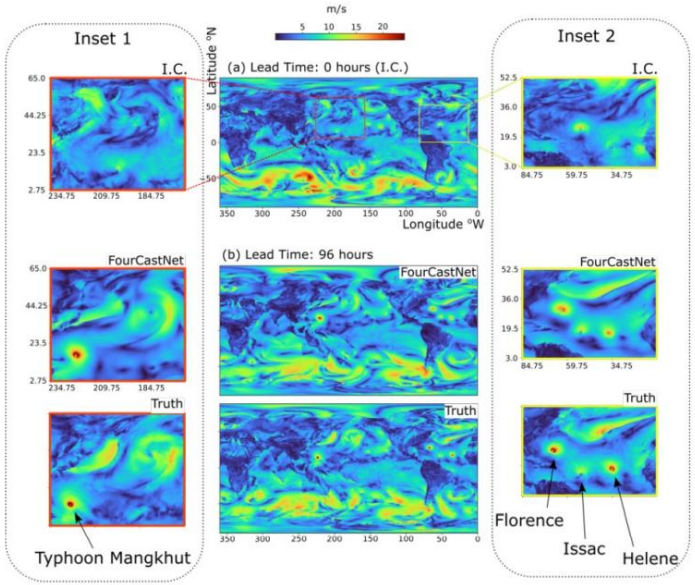
Summary of Operator Learning Task

- Domain
  - $\Omega \subset \mathbb{R}^d$ be a bounded open set of spatial coordinates
- Input and output function spaces on Ω (e.g., Banach spaces, Hilbert spaces)
  - $\mathcal{A}$ and $\mathcal{U}$
- Ground truth solution operator
  - $G : \mathcal{A} \to \mathcal{U}$ with $G(a) = u$
- Training data
  - Observed (possibly noisy) function pairs $(a_i, u_i) \in \mathcal{A} \times \mathcal{U}, u_i = G(a_i)$ with measures $a_i \sim \nu_a, u_i \sim \nu_u$, where ν_u is the pushforward measure of ν_a by G
- Task: Learn operators from data
  - $G_\theta(a) \approx u$

Stony Brook University

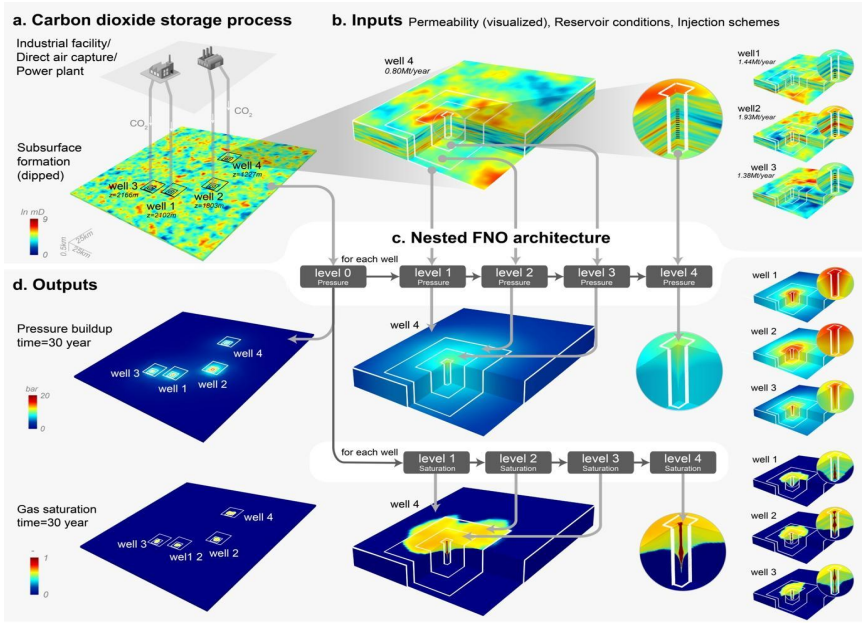
Applications of Operator Learning

Weather Forecasting



Pathak et al. (2022), FourCastNet: A Global Data-driven High-resolution Weather Model using Adaptive Fourier Neural Operators, arXiv: https://arxiv.org/abs/2202.11214

Carbon Storage Modeling



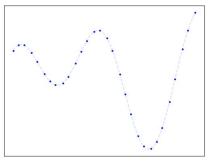
Source: https://developer.nvidia.com/blog/using-carbon-capture-and-storage-digital-twins-for-net-zero-strategies/

FAR BEYOND

Stony Brook University

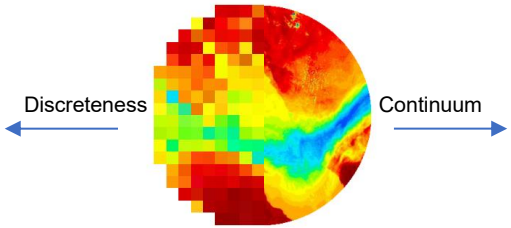
Challenge in Operator Learning

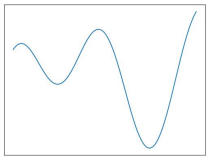
Deep neural networks can only take finite-dimensional inputs and produce finite-dimensional outputs $\phi_{\text{network}} : \mathbb{R}^{d_{in}<\infty} \mapsto \mathbb{R}^{d_{out}<\infty}$.



	cat	mat	on	sat	the
the =>	0	0	0	0	1
cat =>	1	0	0	0	0
sat =>	0	0	0	1	0
...			...		

Finite-dimensional Features





$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_\infty \end{bmatrix}$$

Infinite-dimensional Features

There are different ways of thinking about this challenge and approaching the design of neural operators:

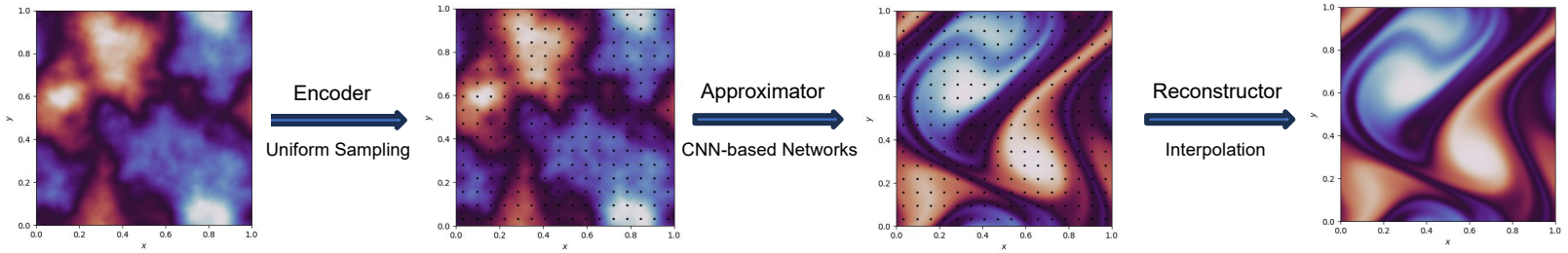
- ❑ Finite-dimensional Learning: Following the same workflow as numerical schemes, functions are encoded into finite-dimensional features.
- ❑ Infinite-dimensional Learning: Extending the concept of linear layers in deep neural networks to infinite-dimensions.

FAR BEYOND

Stony Brook University

Finite-Dimensional Features and Learning

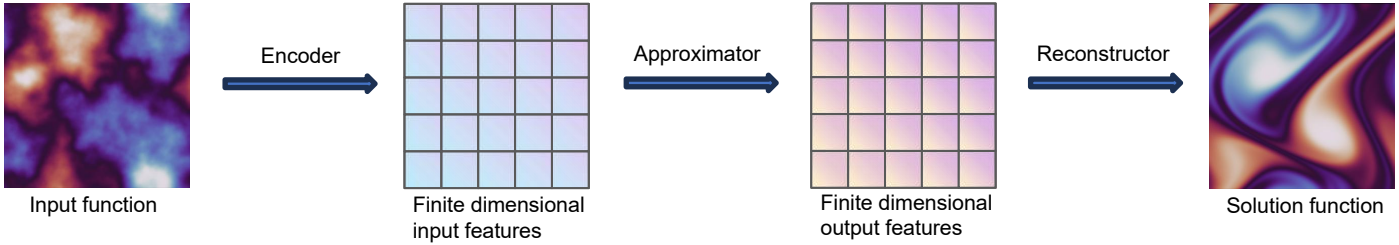
As mentioned, neural networks are mappings between finite-dimensional spaces. To adapt neural networks to learn operators, a workaround is to use a simplified setting in which functions are characterized by finite-dimensional features. An example is to discretize the function with uniform point values, which can then be taken as input to a conventional neural network such as the Convolutional Neural Network (CNN).



FAR BEYOND

Finite-Dimensional Features and Learning

Many numerical schemes can be represented by this diagram as well:



Method	Encoder	Approximator	Example Reconstructor
Finite Difference	Point Values	Numerical Scheme	Polynomial Interpolantion
Finite Element	Node Values	Numerical Scheme	Galerkin Basis
Finite Volume	Cell Averages	Numerical Scheme	Polynomial Interpolantion
Spectral Methods	Fourier Coefs.	Numerical Scheme	Fourier Basis

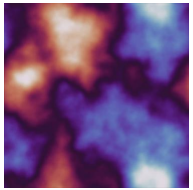
How we make choices of encoders, decoders, and approximators gives rise to different neural operators. Moreover, different choices may lead to different properties or various pros and cons for a particular neural operator.

Stony Brook University

Neural Operator Instantiation: Spectral Neural Operator

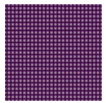
(Generalized) Spectral neural operators [1] encodes a function as function coefficients with respect to a predefined function basis such as Fourier, Chebyshev, etc..
Given appropriate bases, under fairly general assumptions, function can be written in function series forms $f = \sum_{i=0}^{\infty} c_i f_i$.

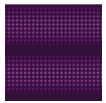
Encoder:



Input function

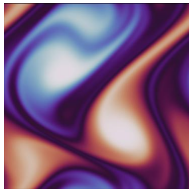
$=$

$a_1 \cdot$  $+ \; a_2 \cdot$  $+$

$a_3 \cdot$  $+ \; a_4 \cdot$  $+ \; \cdots$

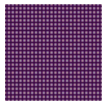
Coefs. with respect to a given basis

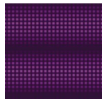
Reconstructor:



Solution function

$=$

$b_1 \cdot$  $+ \; b_2 \cdot$  $+$

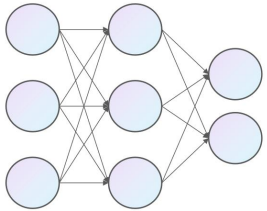
$b_3 \cdot$  $+ \; b_4 \cdot$  $+ \; \cdots$

Coefs. with respect to a given basis

Approximator:

a_1
a_2
a_3
a_4
$\cdots$
a_m

Input Coefs.



b_1
b_2
b_3
b_4
$\cdots$
b_n

Output Coefs.

FAR BEYOND

[1] Spectral neural operators. arXiv, 2022. V. Fanaskov and I. Oseledets.

Stony Brook University

Neural Operator Instantiation: DeepOnet

In the previous approach, Spectral Neural Operators, we encounter the necessity of understanding the function space within which we operate and specifying function bases. However, this requirement can pose limitations from a data-driven standpoint. Often, we only have access to the data itself without a priori knowledge of the function space or the optimal function bases.
DeepOnet [2] is an architecture, based the universal approximation theorem for operators [3], that can overcome this limitation; it takes point values as inputs and makes the reconstructor learnable by learning some function basis (as well as the coefficients, in a sense).

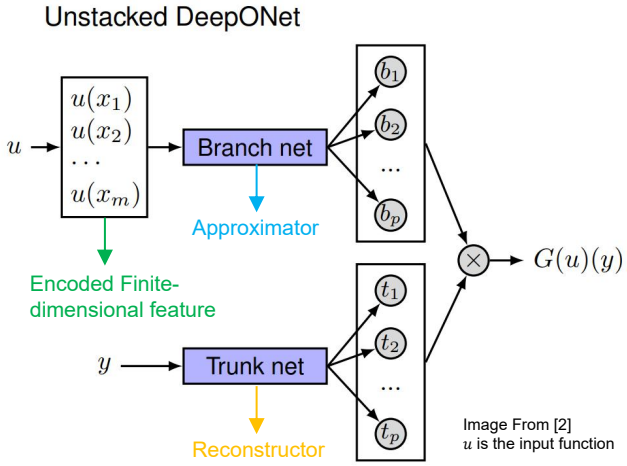


Image From [2]
u is the input function

Encoder	Point Values
Approximator	Neural networks that map fixed sensor point values to coefficients
Decoder	Neural networks that map an inference coordinate point to basis function values at these points

The learned solution function is given by:

$$f(y) \approx G(u)(y) = \sum_{i=0}^{p} b_i \tau_i(y) = \sum_{i=0}^{p} b_i t_i$$

where $\{\tau_i\}_{i=1}^{p}$ are p basis function learned by the trunk network.

Thus, DeepOnet takes in function values at fixed locations, but we can infer the learned solution function at any point in the domain.

FAR BEYOND

[2] Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. NMI, 2021. Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis.
[3] Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. Chen T, Chen H.

Stony Brook University

Finite-Dimensional Learning: Summary

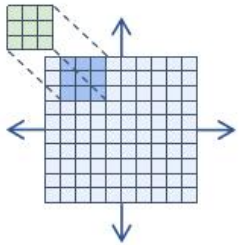
There have been a lot of work done in this direction, and to name a few: PCA-Net [4], IAE-Net [5], and CORAL [6].

Method	Encoder	Approximator	Example Reconstructor
Finite Difference	Point Values	Numerical Scheme	Polynomial Interpolantion
Finite Element	Node Values	Numerical Scheme	Galerkin Basis
Finite Volume	Cell Averages	Numerical Scheme	Polynomial Interpolantion
Spectral Methods	Fourier Coefs.	Numerical Scheme	Fourier Basis
CNN-based Networks	Grid Point Values	DNN	Interpolantion
SNO [1]	Fourier/Chebyshev Coefs.	DNN	Fourier/Chebyshev Basis
DeepOnet [2]	Sensor Point Values	Branch Net (DNN)	Trunk Net (DNN)
PCA-Net [4]	PCA	DNN	PCA
IAE-Net [5]	Auto-encoder	DNN	Auto-decoder
CORAL [6]	Implicit Neural Representation (DNN)	DNN	Implicit Neural Representation (DNN)

[4] Model Reduction And Neural Networks For Parametric PDEs. The SMAI Journal of computational mathematics, Volume 7 (2021), pp. 121-157. Kaushik Bhattachary et al..
[5] Integral autoencoder network for discretization-invariant learning. JMLR, 2022. Yong Zheng Ong, Zuowei Shen, and Haizhao Yang
[6] Operator learning with neural fields: Tackling PDEs on general geometries. arXiv, 2023. Louis Serrano, et al..

Infinite-Dimensional Learning: Introduction

For some of the methods we previously discussed, such as DeepOnet and CNN-based models, the network is highly dependent on the resolution of the data and/or sensor locations. When the resolution or input sensor location changes, the network might perform differently. For DeepOnet, fixed sensor locations are required, and for CNN-based methods, localized fixed size kernels converge to a point-wise operator as the resolution increases.



Another perspective on operator learning is to think in terms of the continuum:

The network should be resolution-invariant and mesh-independent.

In this perspective, we parameterize the model in infinite-dimensional spaces, so it learns continuous functions instead of discretized vectors.

Motivation: Green's Function Method

Under fairly general conditions on $\mathcal{L}_a$, a linear differential operator, we may define the Green's function $G : \Omega \times \Omega \to \mathbb{R}$, where Ω is the domain of the PDE, as the unique solution to the problem

$$\mathcal{L}_a G(x, \cdot) = \delta_x$$

where δ_x is the delta measure on $\mathbb{R}^d$ centered at x. Note that G will depend on the coefficient a thus we will henceforth denote it as G_a.

Then operator $\mathcal{F}_{\text{true}}$ can be written as an integral operator of green function:

$$u(x) = \int_\Omega G_a(x, y) f(y) dy.$$

Here, the green function is called the kernel of the integral operator. We can verify that the integral operator of green function gives the solution.

$$\begin{aligned}
\left(\mathcal{L}_a u\right)(x) &= \int_\Omega \left(\mathcal{L}_a G(x, \cdot)\right)(y) f(y) dy \\
&= \int_\Omega \delta_x(y) f(y) dy \\
&= f(x)
\end{aligned}$$

Generally the Green's function is continuous at points $x \neq y$, for example, when $\mathcal{L}_a$ is uniformly elliptic. Hence it is natural to model the kernel via a neural network κ. Just as the Green function, the kernel network κ takes input (x, y). Since the kernel depends on a, we let κ also take input $(a(x), a(y))$.

$$u(x) = \int_D \kappa(x, y, a(x), a(y)) f(y) dy.$$

Kernel Integral Operators

In a standard deep neural network, a layer can be written as:

$$\text{Input: } v_t \longrightarrow \text{Linear Transformation: } W^T v_t + b \longrightarrow \text{ Non-linearity } \longrightarrow \text{ Output: } v_{t+1}$$

Here the input, v_t, and the output, v_{t+1}, are both vectors.

However, we wish to learn continuous functions instead of discretized vectors. We need to adjust the formulation of our linear layers as it has to be able to take functions as input:

$$v_t(x) \longrightarrow \text{Integral Linear Operator: } \int \kappa(x,y) v_t(y) dy + b(x) \longrightarrow \text{ Non-linearity } \longrightarrow v_{t+1}(x)$$

Integral operators take functions as inputs and produce functions as outputs. Integral operators are also discretization-invariant; we can take inputs at different discretization (e.g. 128×128 or 256×256) representing the same function.

Intuitively, we learn the continuum as *we consider the input as a continuous function and parameterize the kernel also in a continuous sense*. To understand the linearity part, for a one-dimensional input function, if we were to discretize the integral operation with a fixed discretization, it is easy to see that it will again become a matrix-vector multiplication. However, in this way, we are viewing this process as numerical integration which will converge, under general conditions, to the true integral values.

Kernel Integral Operators: GNO

Assuming a uniform distribution of y, the integral can be approximated by a discrete sum

$$v(x) = \int \kappa(x,y)v(y)dy \approx \frac{1}{|N(x)|} \sum_{y_i \in N(x)} k\left(x, y_i\right) v\left(y_i\right).$$

$N(x)$ denotes the neighbors of x. If we construct a graph based on the physical domain using these sensor points, this integration can be viewed as message passing on the graph, with message passing representing the kernel integration. The Graph Neural Operator [7] has been developed with this perspective in mind.

For the full integration, the graph is a n^2 fully connected graph, we may redefine the connectivity to make it more efficient:

$$\frac{1}{|B(x,r)|} \sum_{y_i \in B(x,r)} k\left(x, y_i\right) v\left(y_i\right)$$

Here $B(x,\ r)$ defines the neighbors of x to be within a certain radius r for connectivity. Through layers of message passing, information from one node will eventually propagate through the entire graph.

By defining neighbors based upon radius, the graph construction and message passing is resolution-invariant.

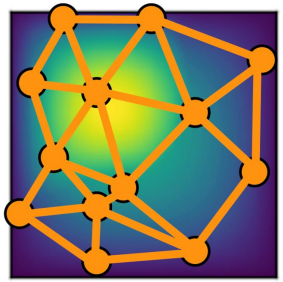


Image Source: https://zongyi-li.github.io/blog/2020/graph-pde/

[7] Neural operator: Graph kernel network for partial Differential equations. arXiv, 2020. Zongyi Li, et al..

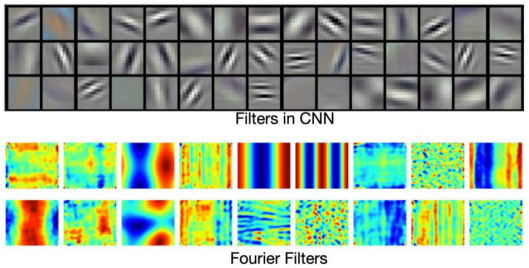
Stony Brook University

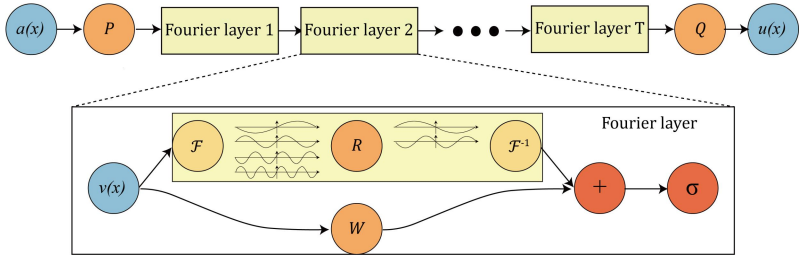
Kernel Integral Operators: FNO

While GNO may encounter challenges related to computational complexity and accuracy, FNO, another widely recognized kernel integral operator [8], stands out for its efficiency and precision. FNO imposes translation equivariance on the kernel: $\kappa(x, y) = \kappa(x - y)$. This is a natural choice from the perspective of fundamental solutions. As a result, the kernel integral operator becomes a (global) convolution operator, which can be solved in the Fourier domain for efficiency.

$$\begin{aligned} \text{Integral Linear Operator} \quad & \int \kappa(x, y) v(y) dy \\ \text{Convolution Operator} \quad & \int \kappa(x - y) v(y) dy \\ \text{Solving Convolution in Fourier domain} \quad & \mathcal{F}^{-1}(\mathcal{F}(\kappa) \cdot \mathcal{F}(v)) \end{aligned}$$

"Filters in convolution neural networks are usually local. They are good to capture local patterns such as edges and shapes. Fourier filters are global sinusoidal functions. They are better for representing continuous functions."





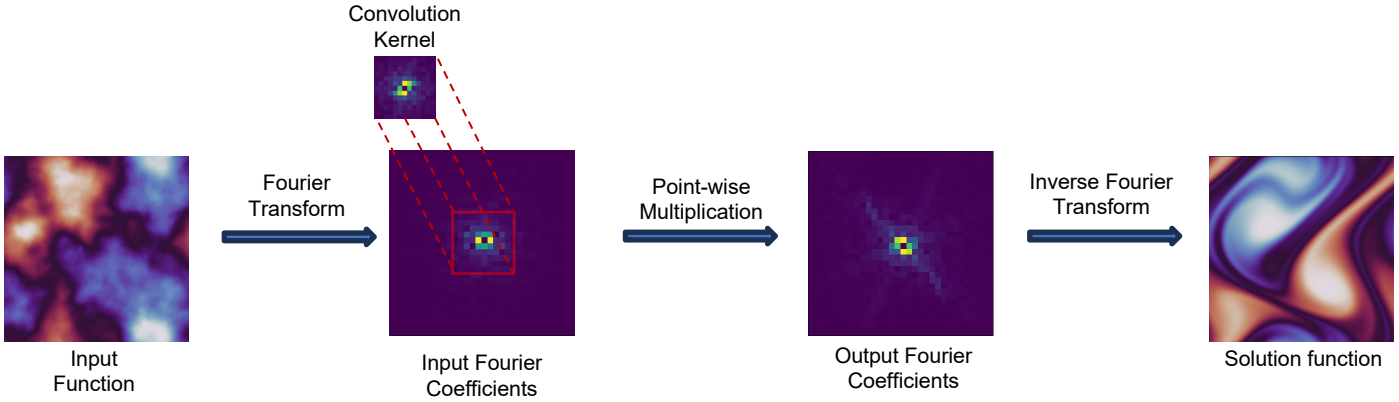
FAR BEYOND

[8] Fourier neural operator for parametric partial differential equations. ICLR, 2021. Zongyi Li, et al..
Image Source: Stanford CS159: Representation Learning for Science

Kernel Integral Operators: FNO

Why Fourier:

1. Whereas approximating the full integral incurs a complexity of $O(n^2)$, the Fast Fourier Transform (FFT) operates with a complexity of $O(n \cdot \log n)$.
2. We can learn the kernel in the Fourier domain directly (convolution is point-wise multiplication in frequency domain).
3. Fourier transform is discretization invariant.
4. By using an orthogonal basis (Fourier) to represent the data, we can facilitate information compression, thereby further enhancing efficiency.



FNO: Infinite-dimensional Learning?

While FNO can accept inputs of varying resolutions, its kernel remains fixed in size within the frequency domain. Later works [9] also suggest that FNO struggles with learning high-frequency modes in the frequency domain or local information in the spatial domain. While FNO operates as a kernel integral operator, the truncation of Fourier modes makes FNO to span both finite-dimensional and infinite-dimensional learning paradigms.
To harness the power of the Fast Fourier Transform (FFT), FNO is most efficient on rectangular domains with uniform sampling of sensor points, a condition that can be restrictive in practical applications.

It is desirable to have a neural operator with the following properties:

1. *Fast and generalizable*
2. *Capable of operating on arbitrary domain shapes*
3. *Able to takes point values freely (more than resolution-invariant, there should be no restriction on the number or locations of sensor points)*
4. *Scalable with polynomial (instead of exponential) dependence in input dimensions*
5. *Efficient in learning both local and global information*
6. *Grounded in theoretical reasoning behind its design (Universal Approximation)*

[1] Spectral neural operators. arXiv, 2022. V. Fanaskc and I. Oseledets.
[6] Operator learning with neural fields: Tackling PDEs on general geometries. arXiv, 2023. Louis Serrano, et al..
[9] Neural Operators with Localized Integral and Differential Kernels. arXiv, 2024.

Stony Brook University

Future Works

- Neural Operators that meets the requirements outlined previously
- Efficient neural operators on Riemannian manifolds with geometric priors
- Transformer-based neural operator designs
- LLM-guided operator learning
- Neural operator designs with physics-constraints such as symmetries/equivariances
- Apply operator learning to practical applications in sciences and engineering
- Tackling aliasing errors within the neural network but also from the data
- …

FAR BEYOND

[1] Spectral neural operators. arXiv, 2022. V. Fanaskc and I. Oseledets.
[6] Operator learning with neural fields: Tackling PDEs on general geometries. arXiv, 2023. Louis Serrano, et al..
[9] Neural Operators with Localized Integral and Differential Kernels. arXiv, 2024.