Learning Function Operators with Neural Networks

appliedAI Seminar

Samuel Burbulla

February 1st, 2024

Simulation and AI
TransferLab @ appliedAI Institute for Europe

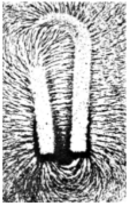
Table of Contents

Introduction

Physical Modeling

We model complex physical problems for predicting future outcomes or engineering!

Examples: *Weather forecasting, fluid flow, aerodynamics, structural mechanics, electromagnetic fields, sound wave propagation, heat conduction, …*

    [1]

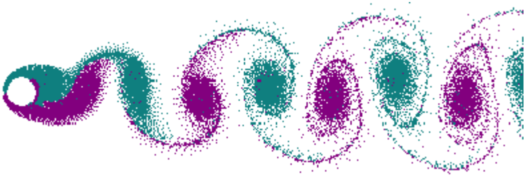
Mathematical laws describe such phenomena, e.g., *partial differential equations (PDEs)*.

[1] *Images: Wikipedia*

Example (Incompressible Navier–Stokes equations)

$$\rho \frac{\partial \mathbf{u}}{\partial t} + \rho(\mathbf{u} \cdot \nabla)\mathbf{u} - \nabla \cdot \sigma(\mathbf{u}, p) = \mathbf{f}, \qquad \nabla \cdot \mathbf{u} = 0$$

u *fluid velocity, p fluid pressure*

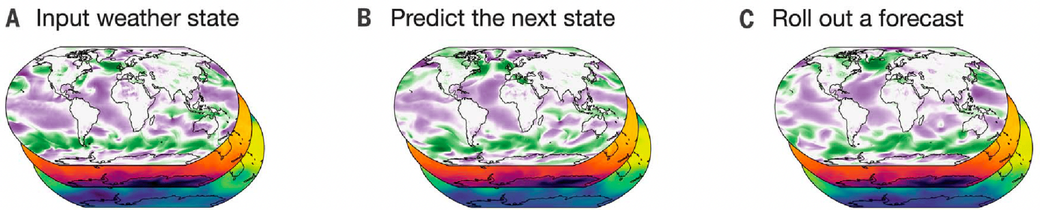


Traditionally, in **Scientific Computing**, we use numerical methods (such as FEM) to approximate solutions to these systems of PDEs.

Machine learning develops statistical algorithms that learn from data, and thus perform tasks without explicit instructions.

Recent example regarding physical modeling: **GraphCast.**[2,3]

→ Outperforms traditional methods in speed and accuracy!

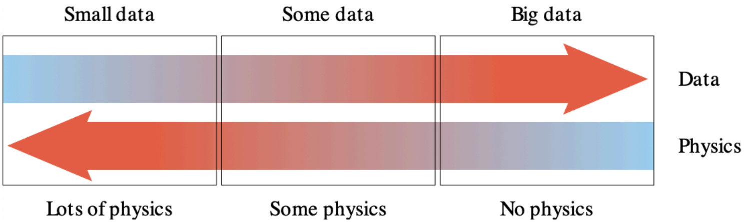


[2] R. Lam et al. **"Learning skillful medium-range global weather forecasting".** *Science* (2023).
[3] *Paper pill: transferlab.ai/pills/2024/graphcast/*

Scientific Machine Learning (SciML)

→ **Scientific Machine Learning** = Scientific Computing + Machine Learning

Examples: *physics-informed neural networks (PINNs), ML-accelerated simulations, ML for scientific discovery, surrogate modeling, neural operators, ...*



Physics-informed ML[4] is a sub-discipline, e.g., incorporating PDEs into the training loss.

[4] G. Karniadakis et al. **"Physics-informed machine learning".** *Nature Reviews Physics* (2021).

Function Operators

In mathematics, **operators** are function mappings: they map functions to functions.

Examples

1. The gradient operator

$$\nabla(\cdot) = \left(\frac{\partial}{\partial x_i}(\cdot)\right)_i$$

maps a function $u : \mathbb{R}^d \to \mathbb{R}$ to its gradient $\nabla u : \mathbb{R}^d \to \mathbb{R}^d$.

2. Time-stepping for Navier-Stokes momentum balance equation could be

$$\mathbf{u}^{n+1} = G(\mathbf{u}^n)$$

$$\text{where} \quad G(\mathbf{u}) = \mathbf{u} + \frac{\Delta t}{\rho}\left(-\rho(\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla\cdot\sigma(\mathbf{u}, p) + \mathbf{f}\right).$$

Operators are omnipresent in physical modeling. Neural networks can learn operators!

Operator Learning

Definition (Operator)

Let $\mathcal{U}$ and $\mathcal{V}$ be (Banach) spaces of functions on bounded domains $D \subset \mathbb{R}^d$ and $D' \subset \mathbb{R}^{d'}$. An *operator* is a map

$$G : \mathcal{U} \to \mathcal{V}.$$

Suppose we have observations $(u_i, v_i)_{i=1,\dots,N}$ where $u_i \in \mathcal{U}$ and $v_i \approx G(u_i)$.

Definition (Operator Learning)

Operator learning is the task of building a parametric map $G_\Theta : \mathcal{U} \to \mathcal{V}$ with parameters $\Theta \in \mathbb{R}^p$ that minimizes

$$\min_{\Theta \in \mathbb{R}^p} \frac{1}{N} \sum_{i=1}^{N} \|v_i - G_\Theta(u_i)\|_{\mathcal{V}}^2.$$

Neural Operators

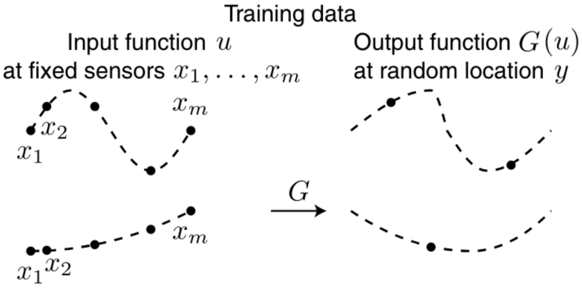
Neural Operators are neural networks that learn operators.

A non-exhaustive list of some relevant architectures includes:

- **DeepONet** (2019)
- **Fourier Neural Operator** (2020)
- and many others:
  - POD-DeepONet (2021), MIONet (2022), **BelNet** (2023), ...
  - **GraphNO** (2020), MultipoleGNO (2020), LowrankNO (2021), ...
  - LaplaceNO (2023), WaveletNO (2023), ConvolutionalNO (2023), ...

$\rightarrow$ They differ in motivation, task-specific performance, and **important properties**!

The **DeepONet**[5,6] (Deep Operator Networks or DON) architecture is directly *motivated* by the **Universal Approximation Theorem for Operators** described by *function evaluations*.



[5] L. Lu et al. **"Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators".** *Nature Machine Intelligence* (2021).
[6] *Paper pill: transferlab.ai/pills/2023/learning-nonlinear-operators-deeponet/*

Theorem 1 (Universal Approximation Theorem for Operator). *Suppose that σ is a continuous non-polynomial function, X is a Banach Space, $K_1 \subset X$, $K_2 \subset \mathbb{R}^d$ are two compact sets in X and $\mathbb{R}^d$, respectively, V is a compact set in $C(K_1)$, G is a nonlinear continuous operator, which maps V into $C(K_2)$. Then for any $\epsilon > 0$, there are positive integers n, p, m, constants $c_i^k, \xi_{ij}^k, \theta_i^k, \zeta_k \in \mathbb{R}$, $w_k \in \mathbb{R}^d$, $x_j \in K_1$, $i = 1, \ldots, n$, $k = 1, \ldots, p$, $j = 1, \ldots, m$, such that*

$$\left| G(u)(y) - \sum_{k=1}^{p} \underbrace{\sum_{i=1}^{n} c_i^k \sigma \left(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k \right)}_{branch} \underbrace{\sigma(w_k \cdot y + \zeta_k)}_{trunk} \right| < \epsilon \qquad (1)$$

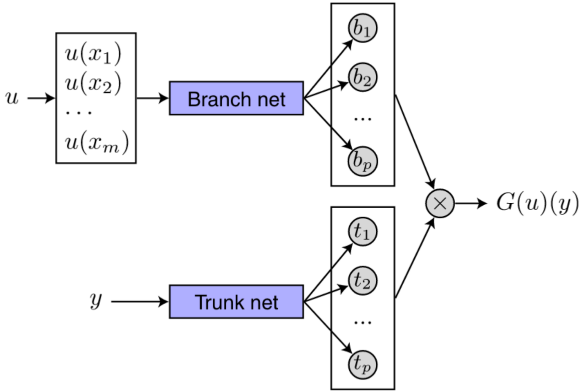
holds for all $u \in V$ and $y \in K_2$. [7]

---

[7] T. Chen and H. Chen. **"Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems".** *IEEE Transactions on Neural Networks* (1995).

DeepONet

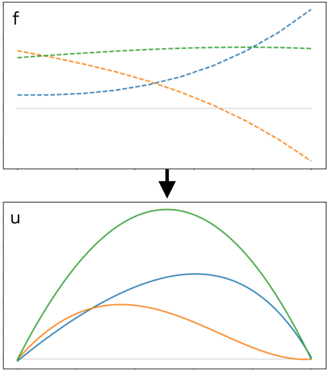
Theorem 1 motivates the distinction into **branch** and **trunk** networks.



The **trunk** learns *basis functions* and the **branch** corresponding *coefficients*.

Physics-Informed DeepONet

Outputs are functions, we can build **physics-informed DeepONets** for parametric PDEs.[8,9]



Consider the Poisson equation in 1D:

$$-u''(x) = f(x), \quad x \in [0,1],$$
$$u(0) = u(1) = 0,$$

for $f \in \mathbb{P}^3$.

$\rightarrow$ We can learn the *solution operator* $G : f \mapsto u$!

[8] S. Wang et al. **"Learning the solution operator of parametric partial differential equations with physics-informed DeepONets".** *Science Advances* (2021).

[9] *deepxde.readthedocs.io/en/latest/demos/operator/poisson.1d.pideeponet.html*

(Fourier) Neural Operators

Another way to look at function mapping:

Integral Kernel Operator

Let $\kappa : D \times D' \to \mathbb{R}^{m \times n}$ be a continuous *kernel* function.
An integral kernel $\mathcal{K}$ maps a function $u : D \to \mathbb{R}^n$ by

$$\mathcal{K}(u)(y) := \int_D \kappa(x, y) u(x) dx \qquad \forall y \in D'$$

to a function $\mathcal{K}(u) = v : D' \to \mathbb{R}^m$.

For $D = D'$ and $\kappa(x, y) = \kappa(x - y)$, $\mathcal{K}$ is a convolution $\mathcal{K}(u) = (\kappa * u)$.

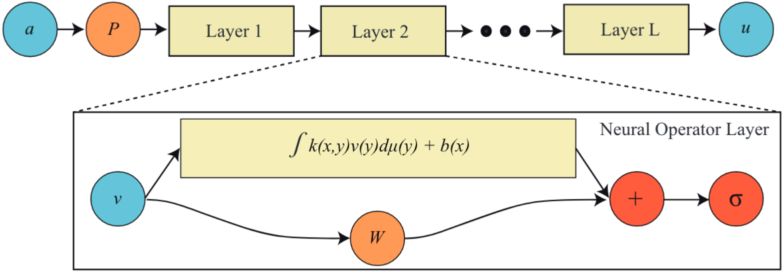
This operation is well-known and broadly used (CNNs, fundamental solutions, ...)

This is the **main building block** of (Fourier) neural operators!

(Fourier) Neural Operators

The neural operator framework by Kovachki et al.[10,11] mimics that of a neural network.

Lifting → Iterative Kernel Integrations → Projection



[10] N. Kovachki et al. **"Neural Operator: Learning Maps Between Function Spaces With Applications to PDEs".** *Journal of Machine Learning Research* (2023).
[11] *Paper pill: transferlab.ai/pills/2023/neural-operators/*

(Fourier) Neural Operators

How to choose a kernel function $\kappa_\phi : D \times D' \to \mathbb{R}^{m \times n}$ to evaluate the integral efficiently?

$\lightning$ *If we just sample J points in D, we have complexity $O(J^2)$ to evaluate the integrals.*

Truncation

Integrate only over subset $S(y) \subset D$, e.g., $B_r(y)$:

$$\mathcal{K}(v)(y) = \int_{S(y)} \kappa_\phi(x, y) v(x) dx \qquad \forall y \in D' \qquad \to \text{still } O(J^2)$$

Graph Neural Operator

Treat a discretization $\{y_1, \ldots, y_J\} \subset D'$ with neighborhoods $\mathcal{N}(y_j) \subset D$ of y_j:

$$\mathcal{K}(v)(y_j) = \frac{1}{|\mathcal{N}(y_j)|} \sum_{x \in \mathcal{N}(y_j)} \kappa_\phi(x, y_j) v(x) \qquad \forall j = 1, \ldots, J \qquad \to O(J\, |\mathcal{N}|)$$

Convolutional Neural Networks are a special case of Graph Neural Operators!

Fourier Neural Operators (FNO)

Idea: Represent the kernel operator in Fourier space.[12]

Assume $D = D'$ and all functions are complex valued.
Let $\mathcal{F}$ denote the **Fourier transform** and $\mathcal{F}^{-1}$ its inverse.

By letting $\kappa_\phi(x, y) = \kappa_\phi(x - y)$ and applying the *convolution theorem*, we find that

$$\mathcal{K}(v) = (\kappa_\phi * v) = \mathcal{F}^{-1}\left(\mathcal{F}(\kappa_\phi) \cdot \mathcal{F}(v)\right).$$

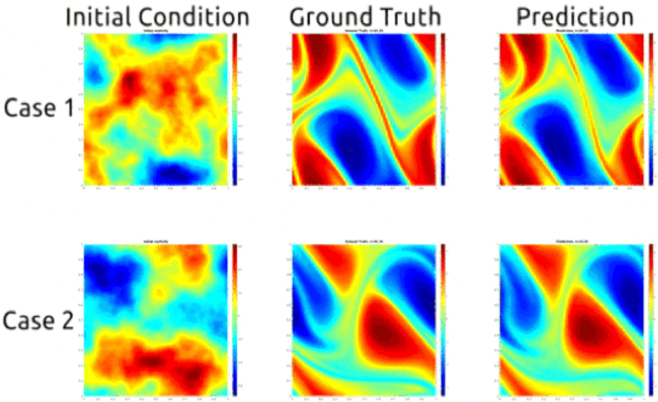
Therefore, we can directly parameterize κ_ϕ in Fourier space with $R_\phi \in \mathbb{C}^{m \times n}$.

Fourier Integral Operator

$$\mathcal{K}(v)(y) = \mathcal{F}^{-1}\left(R_\phi \cdot \mathcal{F}(v)\right)(y) \qquad \forall y \in D \qquad\qquad \rightarrow O(J\ logJ) \quad (FFT)$$

[12] Z. Li et al. **"Fourier Neural Operator for Parametric Partial Differential Equations".** *International Conference on Learning Representations* (2021).

Fourier Neural Operators



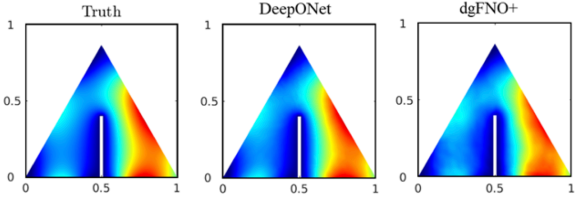
FNO predicting the next time step for turbulent flow.

Orders of magnitudes faster (10.000x), but restricted to periodic unit square (FFT).

DeepONet vs. FNO

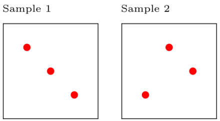
Comparison of DeepONet and FNO (and extensions):[13]

- Vanilla methods may lead to sub-optimal results
- FNO and DeepONet of same size exhibit **same accuracy** (using proper extensions)
- Some architectures are more flexible in terms of problem settings



[13] L. Lu et al. **"A comprehensive and fair comparison of two neural operators (with practical extensions) based on FAIR data".** *CMAME* (2022).

Important Properties

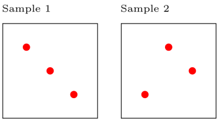


Discretization-invariant

Locations of sensors in the input function domain are not fixed.

$\rightarrow$ *important for unstructured input data, e.g., meshes*

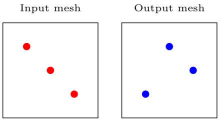
Important Properties



Discretization-invariant

Locations of sensors in the input function domain are not fixed.

$\rightarrow$ *important for unstructured input data, e.g., meshes*



Prediction-free

Discretization of the input can differ from the one of the output.

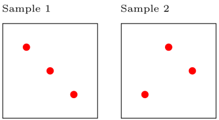
$\rightarrow$ *enables physics-informed training or super-resolution*

Important Properties

Discretization-invariant

Locations of sensors in the input function domain are not fixed.

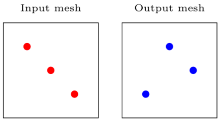
→ *important for unstructured input data, e.g., meshes*



Prediction-free

Discretization of the input can differ from the one of the output.

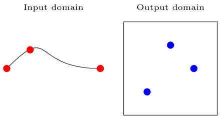
→ *enables physics-informed training or super-resolution*



Domain-independent

Output function domain is independent of input function domain.

→ *much more flexible, e.g., map boundary to solution*



Important Properties

Comparison of DeepONet (DON), FNO, and another architecture, BelNet:[14]

	Discretization-invariant	Prediction-free	Domain-independent
DON	✗	✓	✓
FNO	✓	✗	✗
BelNet	✓	✓	✓

[14] Z. Zhang et al. **"BelNet: basis enhanced learning".** *Proceedings of the Royal Society A* (2022).

BelNet: Basis enhanced learning

Assuming that $\kappa(x,y) = \sum_{k=1}^{K} p_k(y)q_k(x)$ and using a quadrature rule $(w_j, y_j)_{j=1,\ldots,N}$ we get:

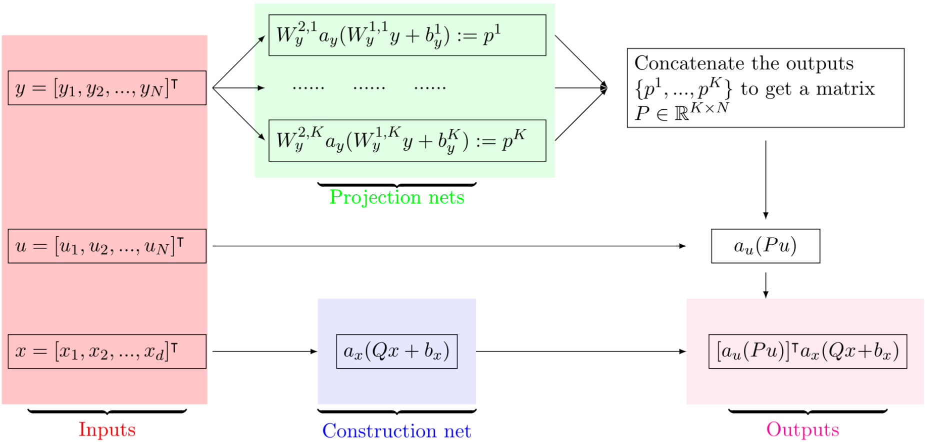
$$\int \kappa(x,y)u(y)dy = \sum_{k=1}^{K} q_k(x) \int p_k(y)u(y)dy \approx \sum_{k=1}^{K} q_k(x) \sum_{j=1}^{N} w_j p_k(y_j)u(y_j)$$

That motivates for $\mathbf{y} = [y_1, \ldots, y_N]$ and $\mathbf{u} = [u(y_1), \ldots, u(y_N)]$ an architecture like:

$$G_\Theta(u)(x) \approx \sum_{k=1}^{K} Q_k(x) \left(P^k(\mathbf{y}) \cdot \mathbf{u} \right)$$

where $Q_k(x) \in \mathbb{R}$ and $P^k(\mathbf{y}) \in \mathbb{R}^N$.

BelNet



BelNet is a generalization of DeepONet: it projects u into the space spanned by a trainable basis p. FNO is a special case of BelNet.

Just Interpolation?

Question: Are neural operators just interpolation?

We can always interpolate a function into a finite-dimensional function space ($\rightarrow$ discretize) and map the coefficients with neural networks.

- FNO (at its core) uses Fourier transform, this is interpolation!
- BelNet *learns* an interpolation, but gives continuous output.
- DeepONet...?

People started investigating the dependence of discretization and representation.[15]

[15] F. Bartolucci et al. **"Representation Equivalent Neural Operators: a Framework for Alias-free Operator Learning".** *NeurIPS* (2023).

Software

Software

At TransferLab, we care about accessible software.

Open-Source Projects

- **DeepXDE** *`deepxde.readthedocs.io`* *(L. Lu)*
  Physics-informed ML, DeepONets | multiple Python backends
- **NeuralOperator** *`neuraloperator.github.io`* *(Z. Li, N. Kovachki)*
  Official implementation of FNOs and more | pyTorch
- **Modulus** *`github.com/NVIDIA/modulus`* *(Nvidia)*
  Deep learning pipelines for physics-ML, FNO, SphericalFNO | pyTorch
- **SciML/NeuralOperators.jl** *`docs.sciml.ai/NeuralOperators`* *(J. Ning, C. Rackauckas)*
  DeepONet, FNO, MarkovNO | written in Julia
- **torch-physics**, ...

Continuity (shameless self-plug)

We started with the development of *Continuity*[16] to establish a **high-level library** for operator learning with neural networks.

- **Unified operator framework**

$$v(\mathbf{y}) = G(u)(\mathbf{y}) \approx G_\theta(\mathbf{x}, u(\mathbf{x}), \mathbf{y})$$

```
v = operator(x, u, y)
```

- Various neural operator architectures
- PDEs for physics-informed training
- Expressive benchmarks

Continuity

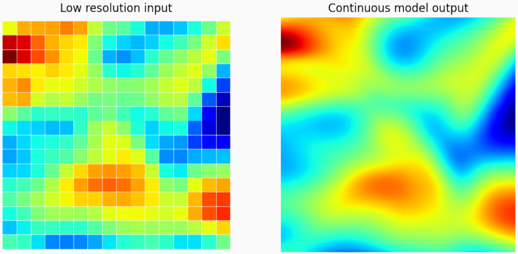
[16] *aai-institute.github.io/Continuity*

Super-Resolution

Neural operators can be used for **super-resolution**, mapping to continuous functions.[17,18]

Example (DeepONet for super-resolution of turbulent flows)

FLAME AI Challenge: Up-sample flow samples from 32x32 to 128x128 (or whatever)



[17] M. Wei et al. **"Super-Resolution Neural Operator".** (2023). arXiv: `2303.02584`.

[18] *See our example: aai-institute.github.io/Continuity/examples/superresolution*

Conclusion

Summary

- Neural operators transfer the **concept of mathematical operators** into ML.
- They have gained **significant attention** in recent years.
- There are **many architectures** with various characteristics.
- We want **discretization-invariant**, **prediction-free** and **efficient** neural operators.

$\rightarrow$ Despite **open questions**, neural operators have exposed a lot of **promising results**!

Exciting times ahead!

Thank you for your attention.

References i

Bartolucci, F. et al. **"Representation Equivalent Neural Operators: a Framework for Alias-free Operator Learning".** *NeurIPS* (2023).

Chen, T. and H. Chen. **"Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems".** *IEEE Transactions on Neural Networks* (1995).

Karniadakis, G. et al. **"Physics-informed machine learning".** *Nature Reviews Physics* (2021).

Kovachki, N. et al. **"Neural Operator: Learning Maps Between Function Spaces With Applications to PDEs".** *Journal of Machine Learning Research* (2023).

Lam, R. et al. **"Learning skillful medium-range global weather forecasting".** *Science* (2023).

Li, Z. et al. **"Fourier Neural Operator for Parametric Partial Differential Equations".** *International Conference on Learning Representations* (2021).

Lu, L. et al. **"A comprehensive and fair comparison of two neural operators (with practical extensions) based on FAIR data".** *CMAME* (2022).

References ii

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

Wang, S. et al. **"Learning the solution operator of parametric partial differential equations with physics-informed DeepONets".** *Science Advances* (2021).

Wei, M. et al. **"Super-Resolution Neural Operator".** (2023). arXiv: `2303.02584`.

Zhang, Z. et al. **"BelNet: basis enhanced learning".** *Proceedings of the Royal Society A* (2022).

(Fourier) Neural Operators

Neural Operator (Kovachki et al.)

$$G_\Theta := \mathcal{Q} \circ \sigma_T(W_{T-1} + \mathcal{K}_{T-1} + b_{T-1}) \circ \cdots \circ \sigma_1(W_0 + \mathcal{K}_0 + b_0) \circ \mathcal{P}$$

Lifting $\mathcal{P}$ and projection $\mathcal{Q}$ are mappings $\mathcal{P} : \mathbb{R}^{d_a} \to \mathbb{R}^{d_{v_0}}$ and $\mathcal{Q} : \mathbb{R}^{d_{v_T}} \to \mathbb{R}^{d_u}$.

We add matrices $W_t \in \mathbb{R}^{d_{v_{t+1}} \times d_{v_t}}$ and bias functions $b_t : D_{t+1} \to \mathbb{R}^{d_{v_{t+1}}}$.

Integral Kernel Operators

Let $\kappa_t \in C(D_t \times D_{t+1}; \mathbb{R}^{d_{v_{t+1}} \times d_{v_t}})$ be a *kernel* function and define $\mathcal{K}_t$ by

$$\mathcal{K}_t(v_t)(y) = \int_{D_t} \kappa_t(x, y) v_t(x) dx \qquad \forall y \in D_{t+1}.$$

Hyperparameters: Dimensions $d^{v_0}, \ldots, d^{v_T}$, $d_1, \ldots, d_{T-1}$, domains $D_1, \ldots, D_{T-1}$ and σ_t.

$\to$ Such an operator has universal approximation properties!

Continuity

Let $u : X \subset \mathbb{R}^d \to \mathbb{R}^c$, $v : Y \subset \mathbb{R}^p \to \mathbb{R}^q$ and $G : u \mapsto v$. For n *sensor positions* $x_i \in X$ and m *evaluation points* $y_j \in Y$, we write $\mathbf{x} = (x_i)_i$, $\mathbf{y} = (y_j)_j$, and $u(\mathbf{x}) = (u(x_i))_i$.

Unified Operator Framework

The evaluations $v(\mathbf{y})$ are approximated by the neural operator G_Θ as follows:

$$v(\mathbf{y}) = G(u)(\mathbf{y}) \approx G_\theta(\mathbf{x}, u(\mathbf{x}), \mathbf{y}).$$

In Python

```
v = operator(x, u, y)
```

with tensors of shape (adding a batch size b):

x: `[b, n, d]` *u*: `[b, n, c]` *y*: `[b, m, p]` *v*: `[b, m, q]`