# Supervised learning in function spaces

## Part III: Fourier Neural Operators

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

# A Neural Network Inspired Approach

- The basic MLP architecture is an alternating composition of linear (affine) and non-linear transformations.

$$W_L \circ \sigma \circ W_{L-1} \circ \cdots \circ \sigma \circ W_1$$

- For appropriate choices of $\sigma$ and enough width/depth, this can approximate any continuous map between finite dimensional spaces

# Applying the classic architecture to function spaces

- We could use an MLP directly as an architecture, but this would only act pointwise on the target space of the input function

$$\mathcal{X} \xrightarrow{\;u\;} \mathbb{R}^{d_u} \xrightarrow{\;\mathrm{MLP}\;} \mathbb{R}^{d_s}$$

- This won't be able to express general function to function mappings
    - Two curves that intersect at a point would always map to curves that intersect at the same point.  Does not use global curve information.

# Generalizing Linear/Nonlinear Compositions

- On function spaces we have many more linear transformations available besides pointwise operators.

- **Example:** On $C(\mathcal{X}, \mathbb{R})$, any continuous $\boldsymbol{k} : \boldsymbol{\mathcal{X}} \times \boldsymbol{\mathcal{X}} \to \mathbb{R}$ defines a linear map

$$T_k : C(\mathcal{X}, \mathbb{R}) \to C(\mathcal{X}, \mathbb{R})$$

$$T_k(u) := \int_{\mathcal{X}} k(\cdot, y) u(y) \, dy$$

These should be included in a layered architecture!

# Additional motivation for integral kernel transformations

- A classic operator to learn is the solution operator for an inhomogeneous partial linear differential equation

$$Lu = f$$

where L is a linear differential operator.

- The goal is to learn $L^{-1}$ such that given $f$ we can solve for $u$

- A Green's function $k(x, s)$ allows us to solve the PDE with

$$u = L^{-1} f = \boxed{\int k(\cdot, y) f(y) \, dy}$$

# Neural Operators

- A **Neural Operator** is a composition of layers of the form

$$v^{(\ell+1)}(y) = \sigma \left( W v^{(\ell)}(y) + \int_D k(y, z) v^{(\ell)}(z) dz \right)$$

where *W* is a pointwise linear transformation and σ is a pointwise nonlinearity

- No longer restricted to fixed number of input function measurements or even their locations

Li, Zongyi, et al. "Neural operator: Graph kernel network for partial differential equations." *arXiv preprint arXiv:2003.03485* (2020).

# How to compute integral part?

- **Option 1:** Graph Neural Operator
- Use a monte-carlo approximation

$$\int k(x,z)u(z)\;dz \approx \frac{1}{N}\sum_{i=1}^{N} k(x,z_i)u(z_i)$$

- If the kernel rapidly decays off its diagonal, (i.e. $\|k(x,y)\|$ quickly becomes small as $\|x-y\|$ grows), we can form a graph where each evaluation point $x$ has an edge to the other measurement points $z_i$ with non-negligible values of the kernel.
- This can be implemented with a message passing algorithm and the graph can be updated to include varying measurement points and locations
  - Additional tricks available (Nystrom/low rank approximations, multi-pole versions, etc.)

# How to compute integral part?

- **Option 2:** Fourier Neural Operator (FNO)

- When the kernel is stationary, $k(y,z) = k(y-z)$, the Fourier convolution theorem gives

$$\int_D k(y-z)v(z)dz = F^{-1}\left(\hat{k}(\xi)\hat{v}(\xi)\right)(y)$$

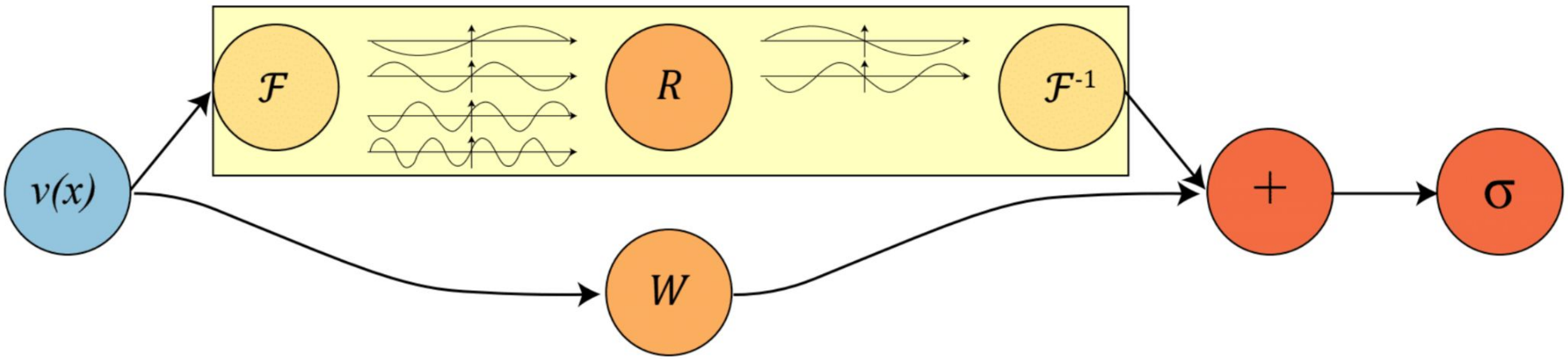
- We can learn $\hat{k}$ directly and approximate the integral with an FFT, IFFT, and a multiplication

Li, Zongyi, et al. "Fourier neural operator for parametric partial differential equations." *arXiv preprint arXiv:2010.08895* (2020).
[1]http://tensorlab.cms.caltech.edu/users/anima/pubs/GraphPDE_Journal.pdf
[2]Kovachki, Nikola, Samuel Lanthaler, and Siddhartha Mishra. "On universal approximation and error bounds for Fourier Neural Operators." arXiv preprint arXiv:2107.07562 (2021).

# Benefits of Fourier Version

- FFTs are *fast*
  - The basic Graph Neural Operator implementation has a quadratic cost in the number of measurement points
- Trained models can be immediately deployed on higher resolution inputs without rebuilding neighborhood graph
- Naturally handles **non-local** transformations of functions
  - Efficient without strict assumptions decay of kernel away from its diagonal

# Fourier Neural Operator Layer




Note the similarity to residual-like architectures (ResNets, etc.)

$$v^{+}(y) = \sigma\left(Wv(y) + F^{-1}\left(\hat{k}(\xi)\hat{v}(\xi)\right)(y)\right)$$

# Universality of FNO

**Theorem 5 (Universal approximation)** *Let* $s, s' \geq 0$. *Let* $\mathcal{G} : H^s(\mathbb{T}^d; \mathbb{R}^{d_a}) \to H^{s'}(\mathbb{T}^d; \mathbb{R}^{d_u})$ *be a continuous operator. Let* $K \subset H^s(\mathbb{T}^d; \mathbb{R}^{d_a})$ *be a compact subset. Then for any* $\epsilon > 0$, *there exists a FNO* $\mathcal{N} : H^s(\mathbb{T}^d; \mathbb{R}^{d_a}) \to H^{s'}(\mathbb{T}^d; \mathbb{R}^{d_u})$, *of the form* (6), *continuous as an operator* $H^s \to H^{s'}$, *such that*

$$\sup_{a \in K} \|\mathcal{G}(a) - \mathcal{N}(a)\|_{H^{s'}} \leq \epsilon.$$

Kovachki, Nikola, Samuel Lanthaler, and Siddhartha Mishra. "On universal approximation and error bounds for Fourier Neural Operators." *Journal of Machine Learning Research* 22 (2021)

- See below for universality statement with general Neural Operators

  Kovachki, Nikola, et al. "Neural operator: Learning maps between function spaces." *arXiv preprint arXiv:2108.08481* (2021).

# Training FNOs

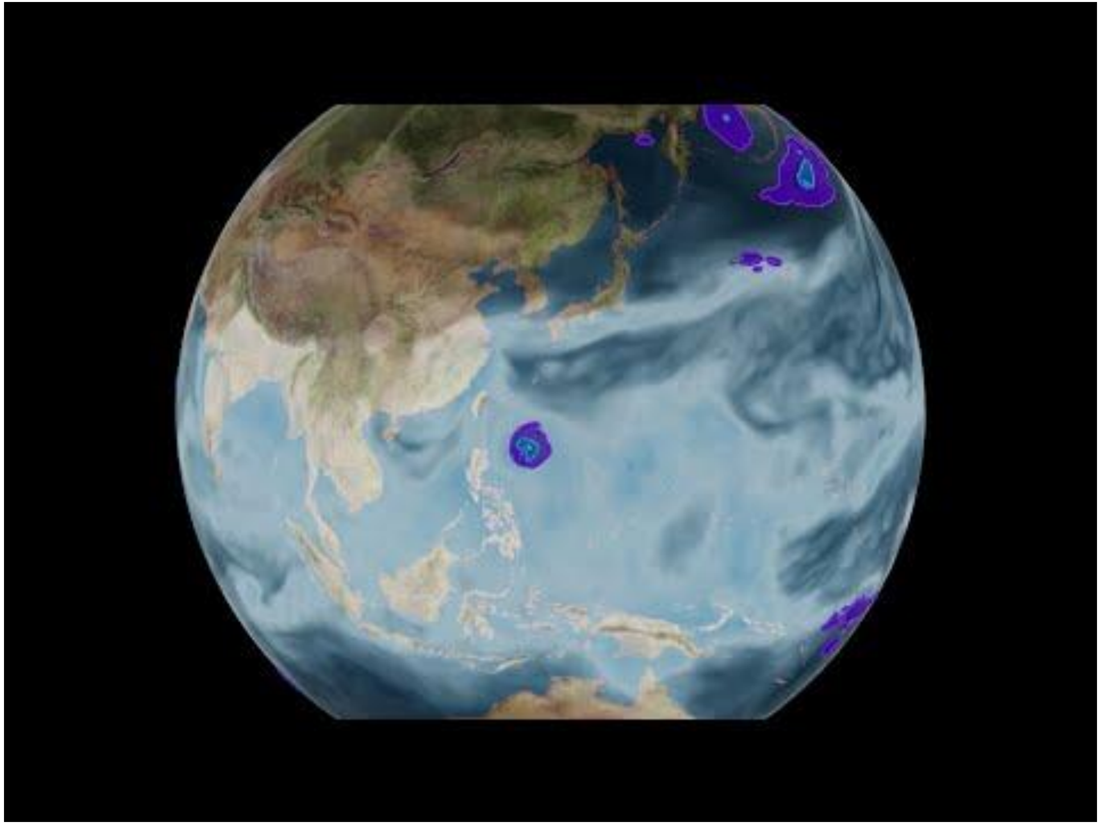
- Input and output function locations $\{x_i\}$, $\{y_i\}$ typically on regular grids so FFT can be used.

- Single training example is

$$\Big\{\{x_{i_1,...,i_{d_x}}\}, \{u(x_{i_1,...,i_{d_x}})\}, \{y_{j_1,...,j_{d_y}}\}, \{s(y_{j_1,...,j_{d_y}})\}\Big\}$$

where $(i_1, \ldots, i_{d_x})$ and $(j_1, \ldots, j_{d_y})$ range over the grids for *x* and *y*, respectively

# Applications highlights



100,000x speed-up over traditional numerical weather models in emulating global climate variables (temperature, pressure, wind velocity)

# NEXT UP

- FNO in JAX
    - An even more in-depth introduction to JAX and an example implementation of the FNO method.

# THEN

- Introduction to LOCA, PIDONS, and applications.