ECE 228 Spring 2025 Lecture 14: Neural Operators Part 2

Luke Bhan

May 15, 2025

Recap of last time

Introduced this notion of an operator which represented the solutions to many of our problems (ODEs, PDEs)

What is an operator?

Operators map spaces of functions to spaces of functions.

Give me an example of operators we did NOT talk about last class!

- Identity operator or zero operators
- Projection operator. (Project one function space onto another)
- Fourier operator (Performs Fourier transform).
- Laplace operator (Performs Laplace transform).

Key point: Many of the problems you are already know can be reformulated as operators!

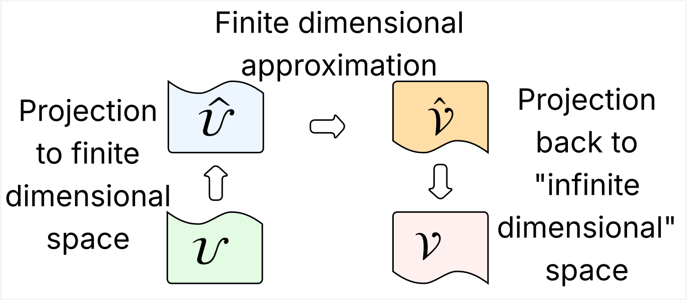
Generic neural operator structure

We discussed how to approximate operators with this framework deemed "neural operator". It had three components to the structure: Can you remind us what they were?

1. Encoder: Need a way to represent the function $f(x)$
2. Finite dimensional approximation - has the **non-local** integral term

$$\mathcal{L}(f)(x) = \sigma\left(Wf(x) + b + \int_{y\in X} K_l(x,y)f(y)dy\right).$$

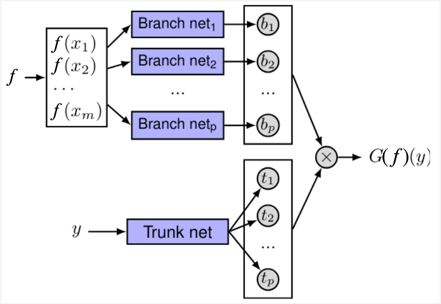
3. Decoder: Way to project our output to any desired point.



Finite dimensional approximation
Projection to finite dimensional space
Projection back to "infinite dimensional" space
$\hat{\mathcal{U}}$
$\hat{\mathcal{V}}$
$\mathcal{U}$
$\mathcal{V}$

Recap of DeepONet

- **Main idea:** Use the finite dimensional approximation to learn a *basis* for the target function which can be queried at any point. That is, from the functional inputs $f(x_1), ..., f(x_n)$, learn basis vectors $b_1, ... b_p$.
- Then the resulting output is a linear combination given by



$$\mathcal{G}(f)(y) \approx \sum_{i=1}^{p} b_i([f(x_1), ..., f(x_n)] \times \tau_i(y)$$

Today, we are going to explore a different approach - Fourier Neural Operator (FNO)

Let's revisit our favorite example: 1D transport PDE

$$\begin{aligned}\frac{\partial u(x,t)}{\partial t} &= \frac{\partial u(x,t)}{\partial x}\,, x \in (0,1], t \in \mathbb{R}^+\\ u(x,0) &= f(x)\,,\\ u(0,t) &= 0\,.\end{aligned}$$

Our operator we want to learn is:

$$f \mapsto u(x,T)\,,$$

where T is some fixed value (say $T = 5$). We constructed a dataset already by numerically solving the PDE until $T = 5$ for a series of initial conditions $f_1, f_2, ..., f_{\mathrm{num\_data}}$ with a discretization step size of $\delta x = 0.01$ and $\delta t = 0.0001$.

Our dataset for the 1D transport PDE looks like:

Thus, our dataset for our Neural Operator looks like:

$$\begin{bmatrix} f_1(x_1) & \cdots & f_1(x_{101}) \\ \vdots & \ddots & \vdots \\ f_{\text{num\_data}}(x_1) & \cdots & f_{\text{num\_data}}(x_{101}) \end{bmatrix}, \quad \begin{bmatrix} u(x_1, T) & \cdots & u(x_{101}, T) \\ \vdots & \ddots & \vdots \\ u(x_1, T) & \cdots & u(x_{101}, T) \end{bmatrix}$$

- Left side is our "encoded" input.
- Right side is our supervised output.

Learn a neural operator solution that is **discretization invariant**. That is, we give it $[f(x_1), \ldots, f_(x_{101})]$ and it gives us $[u(y_1, T), ..., u(y_n, T)]$ where $y_1, ..., y_n \in [0, 1]$.

Webclicker

- We will be using a tool called **Webclicker** as a polling system for participation quizzes during lecture.
- On the Webclicker page (https://webclicker.web.app/), click on the button "SIGN IN WITH GOOGLE". Make sure to use your UCSD email (i.e., @ucsd.edu) to sign in.
- If everything goes well, you will now be logged into the tool. Enter **FTMODV** in the course code field. You will now be able to select **ECE 228**.
- During lecture, you can vote on the polls by logging into the Webclicker site with your phone, laptop or computer, and selecting our course. Once we start a poll, you will see the voting buttons appear.

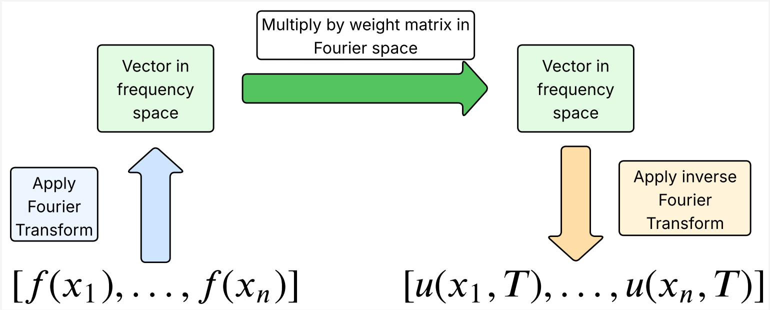
Participation Quiz

- How's the progress with your team's final project?
  A. Going well — we're making solid progress!
  B. Some progress — but still figuring things out.
  C. Not much progress yet — we need to work harder...
  D. We're looking for a team member — let me know since we have students looking for a team!

Cast your vote in 1 minute

Fourier Neural Operator

- **Key idea:** Apply a Fourier transform to our input data, learn in Fourier space, and then our inverse Fourier transform *can be at any desired grid-resolution*


Multiply by weight matrix in Fourier space
Vector in frequency space
Vector in frequency space
Apply Fourier Transform
Apply inverse Fourier Transform
$[f(x_1), \ldots, f(x_n)]$
$[u(x_1, T), \ldots, u(x_n, T)]$

Fourier transform converts time into frequency

Mathematically, we can represent any function as

$$f(x) = \int_{-\infty}^{\infty} A(w) e^{i\omega x} d\omega$$

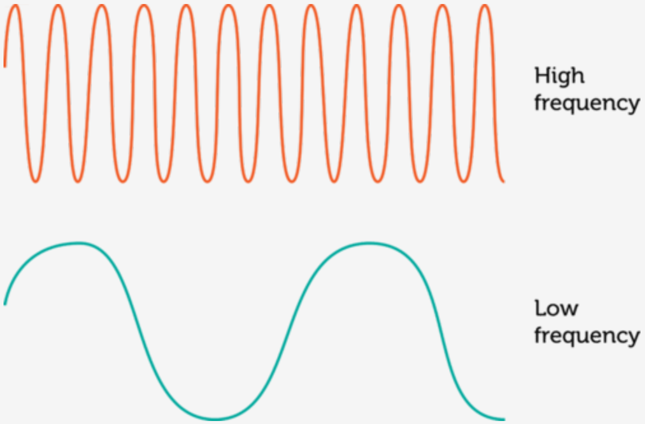
where each $A(\omega)$ represents some set of coefficients. Using Euler's formula we have

$$A(\omega)e^{i\omega x} = A(\omega)\cos(\omega x) + iA(\omega)\sin(\omega x)$$

That is, we can write any function $f(x)$ as a *infinitesimal sum* of sin and cos.

What is frequency?

We call ω the frequency as it controls the speed of oscillation of the sin and cos waves.



Interpretation of $A(\omega)$.

Recall, we said $f(x)$ is given by:

$$f(x) = \int_{-\infty}^{\infty} A(w) e^{i\omega t} d\omega$$

Notice that $A(\omega)$, our coefficients change with each frequency. Furthermore, notice that if we know $A(\omega)$ for all ω, then we know $f(x)$. To obtain $A(\omega)$, Fourier showed:

$$A(\omega) = \int_{-\infty}^{\infty} f(x) e^{-i\omega x} dx$$

That is $A(\omega)$ and $f(x)$ are directly related through very similar functions (these are known as **duals** of each other in mathematics). Instead of using $A(\omega)$, we typically use $\hat{f}(\omega)$ due to this relation.

Discrete Fourier transforms

Fourier transform of a function $f(x)$ is given by

$$\hat{f}(s) = \int_{-\infty}^{\infty} f(x) e^{-i\omega x} dx$$

At a high level: What is the discrete Fourier transform?

We can't evaluate every $\omega \in (-\infty, \infty)$, so we need a discrete number of frequencies (these are called the modes)

$$\hat{f}(k) = \sum_{j=1}^{n} f(x_j) \cdot e^{-i2\pi \frac{k}{n} j}, \quad k = 1, 2, \ldots, n$$

Assumptions on DFT

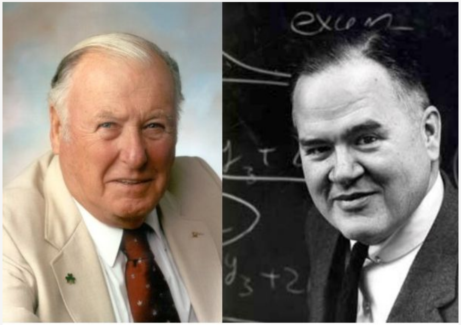
What assumptions do I make about the function $f(x)$ when I apply a DFT?

- $f(x)$ is periodic in n. That is $f(x_1), \ldots, f(x_n)$ will repeat if I have $f(x_{n+1}, \ldots, x_{2n})$.
- $x_1, \ldots, x_n$ are *uniformly spaced*. That is $x_2 - x_1 = x_3 - x_2 = x_n - x_{n-1}$.
- Why do we need 2π in the coefficient for a DFT? **Answer:** We assume $f(x_1), \ldots, f(x_n)$ is period and so our data on $x_1, \ldots, x_n$ completes one full periodic revolution. For a sin, cos equivalent it takes 2π to complete a revolution over n samples and so we *normalize* our spacing to be the frequencies from $[0, 2\pi]$.

Fast Fourier Transforms

What is a fast Fourier transform? What is its time complexity?

- An algorithm that lets you compute $\hat{f}(k)$ for all $k \in [1, ..., n]$ in $O(n \log(n))$ time.



James William Cooley (1926-)   John Wilder Tukey (1915-2000)

So when I say compute the Fourier transform in FNO, what does this mean?

It means, according to our input data

$$[f(x_1), ..., f(x_n)]$$

compute the FFT yielding:

$$[\hat{f}(k_1), ..., \hat{f}(k_n)]$$

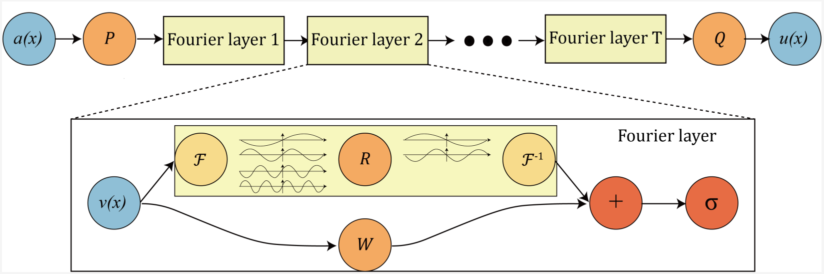
where

$$\hat{f}(k) = \sum_{j=1}^{n} f(x_j) \cdot e^{-i2\pi \frac{k}{n} j}, \quad k = 1, 2, \ldots, n$$

What does k represent?

Back to FNO

The architecture is given by



where each Fourier layer computes:

$$\text{Fourier\_Layer}(f(x)) = \mathcal{F}^{-1}(R\mathcal{F}(f(x)))\,,$$

where $\mathcal{F}, \mathcal{F}^{-1}$ are Fourier transforms and R is a weight matrix (learnable).

How does this relate to our theory?

- **Encoder:** $[f(x_1), ..., f(x_n)]$ and project up to high dimension via MLP.
- **Approximator:** Recall hidden layers were given by:

$$\mathcal{L}(f)(x) := \sigma\left(W_l f(x) + b_l + \int_{y \in X} K_l(x,y) f(y) dy\right)$$

  If I choose my weight matrix $K_l(x,y) = K_l(x-y)$, then I obtain a function of one variable. If I choose this to be given by

$$K_l(z) = \sum_{k=1}^{n} \hat{P}_{l,k} e^{ikz}$$

  Then, I recover the exact form of the Fourier layer we discussed with the R learnable weight matrix as the same thing as the $\hat{P}_{l,k}$ parameters.
- **Decoder:** Project back to same dimensional space as input shape via MLP.

Practical details

- What do I mean by "project up to/from high dimension" in encoding and decoding?
  Consider our example, with input $[f(x_1), ..., f(x_n)]$. Say $f(x_1) \in \mathbb{R}$ is a scalar. Then this is a n-length vector. When I say project this, I mean turn this into a $n \times d$ length vector where d is some chosen projected dimension size.
  Similarly, in decoding, if the vector is $n \times d$, a projection would turn it into a vector of length n.
- A DFT assumes periodicity. What happens when my f is not periodic?
  Answer (engineering trick): Add fake "padding" of 0's to make it periodic. That is $[f(x_1), ..., f(x_n)]$ becomes $[0, 0, 0, f(x_1), ..., f(x_n), 0, 0, 0]$.
- We typically use relative L_2 loss for PDEs. That is

$$\text{Relative } L_2 \text{ Loss} = \frac{\|u_{\text{pred}} - u_{\text{true}}\|_2}{\|u_{\text{true}}\|_2}$$

.

Key advantage of the approach: Super-Resolution

That is, since I can project the IFFT from my number of modes to any number of points desired, I can train on $(x_1 \dots, x_n)$ spread far apart (cheaper to build dataset) and deploy at fine resolution on $(y_1, \dots, y_{10n})$,(i.e. $10\times$ resolution) close together.

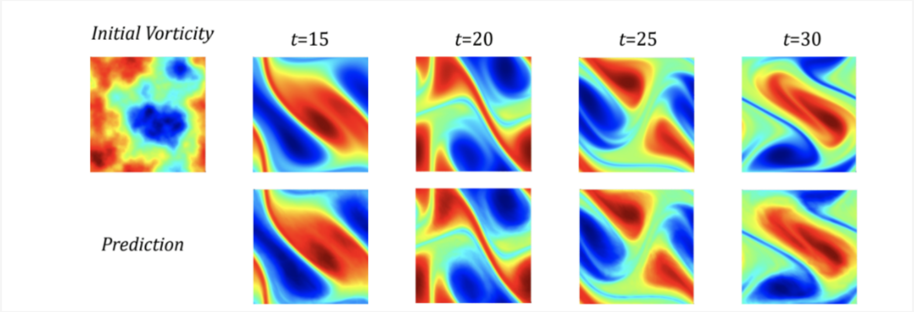
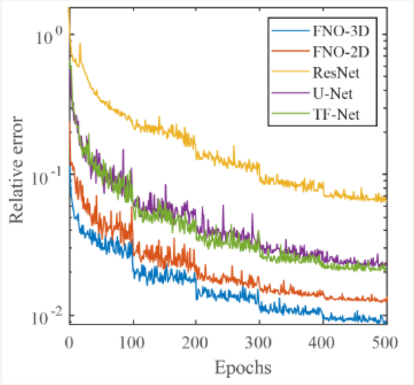
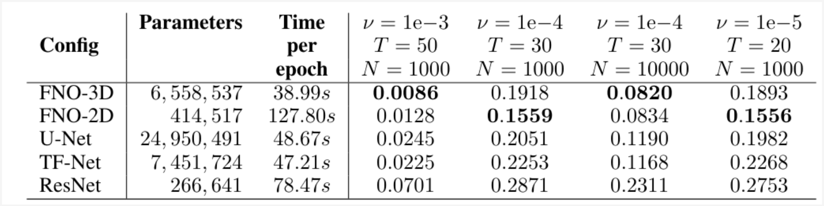


Figure: 2D Navier stokes equation (x, y, t). True values on top. Predictions with FNO on bottom evaluated at $256 \times 256 \times 80$ resolution. FNO was trained with $64 \times 64 \times 20$ resolution

Comparison with other learning methods





Config	Parameters	Time per epoch	$\nu = 1\text{e}{-3}$, $T = 50$, $N = 1000$	$\nu = 1\text{e}{-4}$, $T = 30$, $N = 1000$	$\nu = 1\text{e}{-4}$, $T = 30$, $N = 10000$	$\nu = 1\text{e}{-5}$, $T = 20$, $N = 1000$
FNO-3D	6,558,537	$38.99s$	**0.0086**	0.1918	**0.0820**	0.1893
FNO-2D	414,517	$127.80s$	0.0128	**0.1559**	0.0834	**0.1556**
U-Net	24,950,491	$48.67s$	0.0245	0.2051	0.1190	0.1982
TF-Net	7,451,724	$47.21s$	0.0225	0.2253	0.1168	0.2268
ResNet	266,641	$78.47s$	0.0701	0.2871	0.2311	0.2753

Disadvantages

What are the downsides of the FNO architecture?

- Requires uniform meshes to compute the Fourier transform.
- Truncation of the modes in frequency space can make it more challenging to capture sharp transitions in the model output (Model is biased to very smooth outputs, PDE solutions in many cases are not smooth. This is known as **spectral bias**.)
- FFT's can still be expensive, especially when the batch size is large.
- Comparison with DeepONet - cannot evaluate at any target point. (See this paper for more details on a "fair" comparison)
- How to handle boundary conditions in FNO especially when doing time-based super-resolution?

Application in climate modeling: FourCastNet(FNO)

- "FourCastNet predicts, with unparalleled accuracy at forecast lead times of up to one week, challenging variables such as surface winds and precipitation. No deep learning (DL) model thus far has attempted to forecast surface winds on global scales."

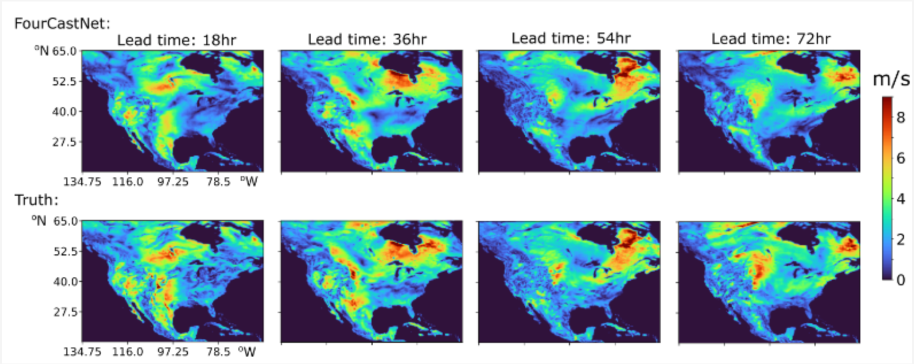


Figure: Overland wind speed (useful for turbulence)

Application in Earthquake Epicenter Detection (DeepONet)

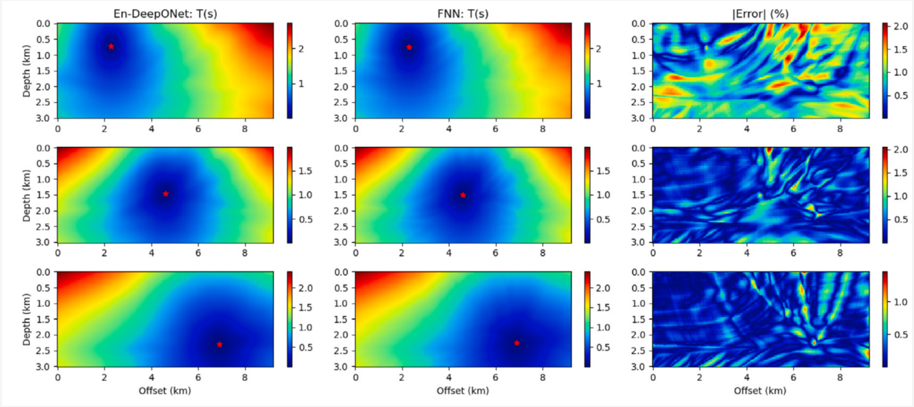


Figure: Estimation of earthquake travel times. Input is a sample of the earthquake magnitude at a few sensor locations. (Left is DeepONet and middle is classical PDE solver). See paper for more details.

Application in Motion planning in Robotics (FNO)

Given a binary occupancy grid and a goal location, compute the cost to get from any point to the goal location (See paper).

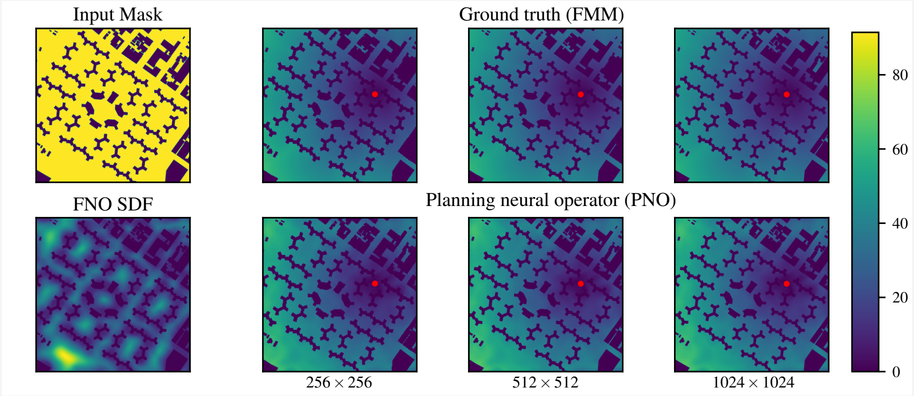
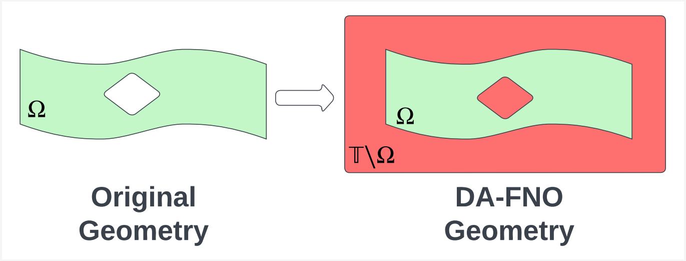


Figure: Dataset is trained using 64×64 resolution and here we see super-resolution on a map of navigating New York city.

What if my geometry is not a square (and so I can't apply FFT)?

Two approaches (there are many more):

1. Create an encoder-decoder structure that maps your geometry to a square (See paper). Downside is how to preserve the super-resolution property across the encoder-decoder?
2. Domain agnostic Fourier Neural Operators. Circumscribe geometry in a square:



Then, in the kernel function modify $K_l(x, y)$ to be zero if x or y is in the square, but not in the original geometry.

What other types of kernels can I do

- Instead of the Fourier transform, we can use other types of transforms (Wavlet transform or Laplace transform)
- Separable or (Sec 4.2 of this paper, low rank) neural operator. Treat the kernel as the product of two functions

$$K_l(x, y) = \phi(x)\psi(y)$$

  where $\phi(\cdot)$ and $\psi(\cdot)$ are classic MLPs.
- Graph Neural Operator: It can be very expensive to compute $K_l(x, y)$ at all points x, y. So, approximate it:

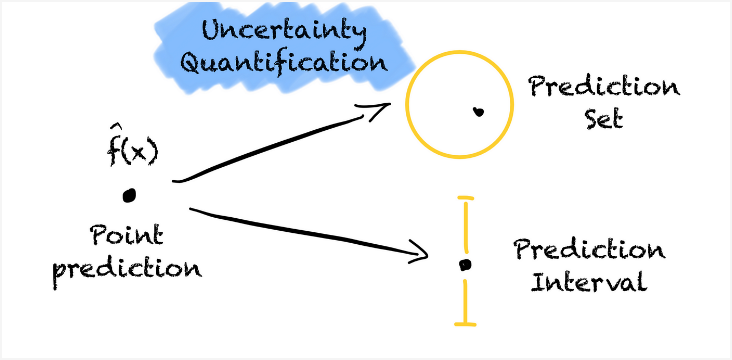
$$\int_{y \in X} K_l(x, y) f(y) dy \approx \frac{1}{S} \sum_{j=1}^{S} k(x, y_s) f(y_s)$$

  where we chose S to be some subset of all our points in X. This is called a Nystrom approximation. This is called graph neural operator as $K_l(x, y_s)$ is akin to an adjacency matrix.
- Transformers are *one instantiation* of neural operators (See Sec 5.2 of this work).

Uncertainty quantification

What is uncertainty quantification?

Answer: Uncertainty quantification tells us how confident a model is in its prediction.



Types of UQ

What are some types of UQ that you already know for ML?

- **Softmax weighting gives some sense of UQ**: If a model gives 0.9 for a class versus 0.55, in both cases we choose that class but one is "more confident"
- **Ensemble methods:** Train 5 NNs and compare how much they vary on any one test case (tight spread=more confident ideally)
- **Gaussian Processes (GP):** Essentially they are a tool that return a mean and standard deviation of a estimated prediction (That is constrained to follow a multivariate Gaussian) instead of just the prediction itself.
- **Conformal prediction:** Gives prediction intervals such that the true value is within the interval with probability $1 - \delta$.
- **Bayesian Neural Networks:** Weights of NN aren't fixed, but are sampled from a probability distribution at each forward inference call.

UQ for neural operators

We talked about how its important to do uncertainty quantification for neural operators. Here's whats been done:

- Conformal prediction for neural operators. Idea: Train a FNO. Then, train a second FNO to give you an uncertainty ball around that point.
- Ensembled neural operators. Train a bunch of neural operators and take an "average" of the outputs in a smart way.
- Bayesian/GP Neural Operators. Instead of learning a static output, they augment the NO architecture such that it outputs a mean and standard deviation that follow's a Gaussian prior at each point.

There are many more ideas here to be explored and Yuexin will explore UQ in more detail in the future weeks!

Theory on neural operators

- Universal approximation theorem of neural operators (See paper)
- Discretization error estimates. That is if I have more $x_1, ..., x_n$, how does my error theoretical change?
- Sample complexity estimates If I have more f_i, u_i pairs, how does my error change?
- Parameter complexity estimates If I want to achieve an accuracy of ϵ, I need $C\exp(ce^{-\gamma})$ $(C, c, \gamma$ problem dependent) parameters for my NN.

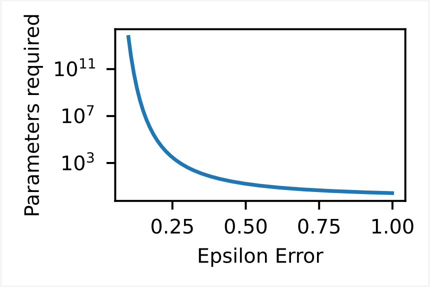
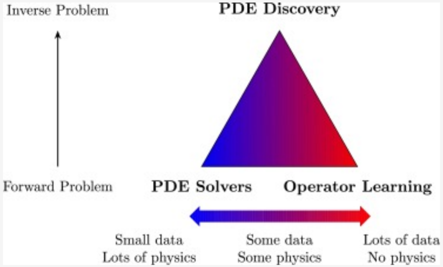


Figure: Scaling of parameters with respect to ϵ, $(C, c = 1, \gamma = 1.5)$

Open problems left in the field

- **Applications**. Many applications left for YOU to apply these.
- Complicated geometries and boundary conditions.
- Learn PDEs with discontinuities. (Very hard learning problem for FNO which is biased towards smooth functions)
- Encoding of the function f can be bad. Symbolic-based encoding (See paper)
- Integrate physics into our design (See you next lecture on PINNs)


Inverse Problem
PDE Discovery
Forward Problem
PDE Solvers
Operator Learning
Small data
Lots of physics
Some data
Some physics
Lots of data
No physics