Introduction to Fourier Neural Operator

Yaohua Zang

June 23, 2025

0.1 Review of Parametric PDE Problems

We consider a general parametric PDE in residual form:

$$F(x, y, u, u_x, u_y, u_{xx}, u_{yy}, \cdots, a) = 0, \quad (x, y) \in \Omega,$$

where:

- $u(x, y)$ is the solution,
- $a(x, y)$ is a parameter (e.g., conductivity or source) that may vary across instances,
- and appropriate boundary conditions are applied on $\partial\Omega$ (Dirichlet, Neumann, or mixed).

The goal is to solve for $u(x, y; a)$ given any input parameter $a(x, y)$. This defines an operator learning problem, where we seek to approximate the solution operator:

$$\mathcal{G} : a(x, y) \in \mathcal{A} \longrightarrow u(x, y) \in \mathcal{U}, \tag{1.1}$$

with $\mathcal{A}$ and $\mathcal{U}$ being function spaces for the inputs and outputs, respectively.

0.2 Kernel-Based Neural Operator

0.2.1 Standard Neural Network Layer

In classical neural networks, we apply a sequence of linear transformations and nonlinear activations to vectors. A generic solution operator can be written as:

$$u = (K_L \circ \sigma_{L-1} \circ \cdots \circ \sigma_1 \circ K_0)(v),$$

where each K_l is a linear or convolutional layer, and σ_l is a nonlinear activation (e.g., ReLU). The layer-wise mapping is as follows:

$$v_{l+1} = \sigma_l(K_l v_l + W_l),$$

with σ_l the activation function and W_l a bias term.

0.2.2 Kernel-Based Neural Operator

The **kernel-based neural operator** extends this idea to function spaces. Instead of vector inputs, it learns mappings between discretized functions (e.g., on meshes or grids). Here, linear transformations are modeled as **integral operators**:

$$v_{l+1}(x) = \sigma_l \left(\int_\Omega k_l(x, y) v_l(y)\, dy + W_l v_l(x) \right), \tag{2.1}$$

where:

- $x, y \in \Omega$ are spatial points in the domain,
- $k_l(x, y)$ is a learnable **kernel function**,
- σ_l is the activation function,
- and W_l applies a pointwise linear transformation (bias).

This kernel-based structure allows the model to process and learn **function-to-function** mappings directly, making it naturally suited for solving parametric PDE problems.

0.3 What is the Fourier Neural Operator (FNO)?

0.3.1 Motivation Behind FNO

In computer vision, convolutional neural networks (CNNs) are highly effective because real-world images contain edges and local patterns, which CNNs can capture using **local convolution kernels**. However, in parametric PDEs, both the input coefficients and output solutions are continuous functions and often exhibit **global correlations** (e.g., smooth variations across the domain). Therefore, local convolutions may not be the most efficient approach.

The **Fourier Neural Operator (FNO)** was developed to address this limitation by:

- Replacing local convolution with **global convolution** using the Fourier transform;
- Leveraging the fact that smooth functions (common in PDEs) are more compactly and efficiently represented in **Fourier space**.

Thus, FNO chooses the kernel map K to be a global convolution, implemented via **Fourier transformation**.

0.3.2 The Fourier Layer: Core of FNO

The Fourier layer is the fundamental building block of FNO. It enables efficient and expressive global transformations by working in the frequency domain.

Why Fourier?

- **Speed**: Direct integration over n points is $\mathcal{O}(n^2)$, but Fourier-based convolution is **quasilinear** in n using the Fast Fourier Transform (FFT).
- **Compactness**: PDE-related functions often have energy concentrated in lower frequency modes, so a small number of Fourier coefficients can encode a lot of information.

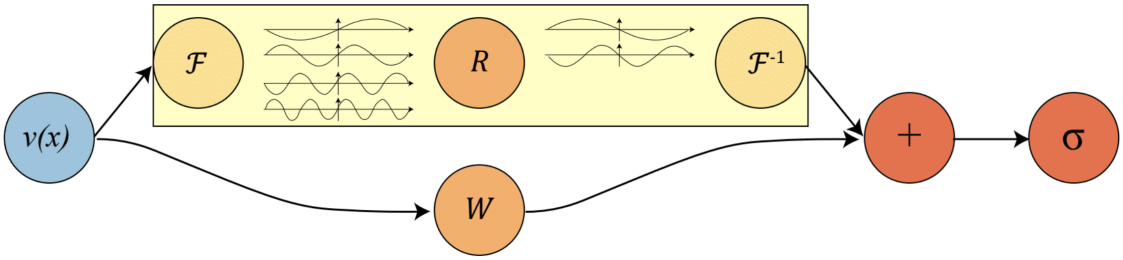

Figure 1: The Fourier layer

How the Fourier Layer Works The Fourier layer replaces standard convolution with a three-step process:

1. **Fourier Transform**: Convert the function v_l from spatial domain to frequency domain using $\mathcal{F}$.

2. **Linear Transformation in Frequency Space**: Apply a learned transformation R to the low-frequency modes.

3. **Inverse Fourier Transform**: Map back to the spatial domain using $\mathcal{F}^{-1}$.

This gives:

$$v_{l+1}(x) = \sigma\left(\mathcal{F}^{-1}\big(R \cdot \mathcal{F}(v_l)\big)(x) + W v_l(x)\right)$$

where:

- σ is a pointwise activation function (e.g., ReLU),

- R is a learnable weight matrix applied only on selected Fourier modes,

- W is a linear transformation applied in the spatial domain.

Remarks:

- **Truncation**: — In practice, we **discard high-frequency Fourier modes** and apply R only on the lower modes, as they carry the most useful information for PDEs.

- **Activation in Spatial Domain**: — Activations are applied after the inverse Fourier transform. This helps recover:

  - **High-frequency details** omitted by truncation,
  - **Non-periodic boundary behaviors** not captured by pure Fourier modes.

0.3.3 Architecture of the Full FNO Model

Now that we understand the Fourier layer, let's look at how it fits into the full FNO framework for solving operator learning problems like:

$$\mathcal{G} : a(x, y) \in \mathcal{A} \to u(x, y) \in \mathcal{U} \tag{3.1}$$

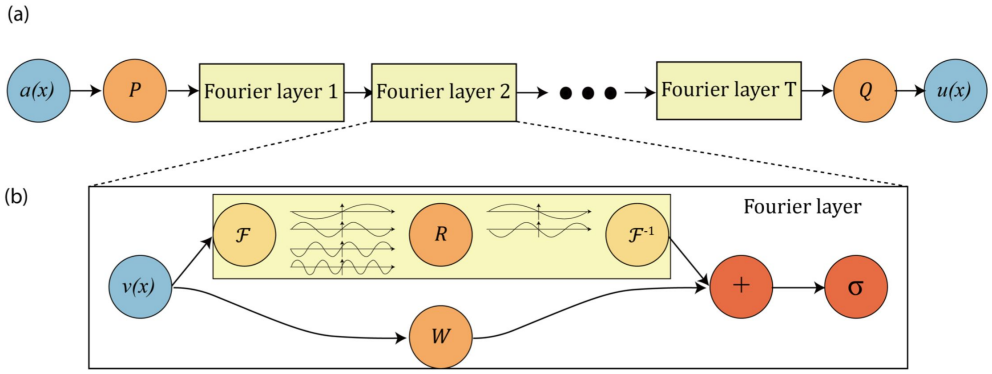

Figure 2: *
Fig 2: The FNO architecture

The FNO model consists of three main components:

- **Input Layer: Lifting the Input Function** To begin, the input coefficient function $a(x)$ (and optionally the coordinate x) is lifted into a higher-dimensional representation using a neural network P_θ. This creates the initial latent vector v_0, which serves as the input to the first Fourier layer:

  $$v_0 = P_\theta(a, x)$$

  This step allows the model to map the input from function space into a latent space where the Fourier layers can operate more effectively.

- **Fourier Layers: Learning Global Transformations** The core of FNO consists of multiple stacked Fourier layers, which learn global representations efficiently in the frequency domain. Each layer updates the latent representation v_l using the Fourier-layer operation:

  $$v_{l+1} = \sigma\left(\mathcal{F}^{-1}(R \cdot \mathcal{F}(v_l)) + W v_l\right)$$

  - $\mathcal{F}$ and $\mathcal{F}^{-1}$ denote the Fourier and inverse Fourier transforms.
  - R is a learnable matrix applied to selected low-frequency modes.
  - W is a spatial-domain linear transformation.
  - σ is a nonlinear activation function (e.g., ReLU).

  By stacking several Fourier layers, the model can capture both low-frequency (smooth, global) and high-frequency (sharp, local) behavior of PDE solutions.

- **Output Layer: Projecting Back to Function Space** Finally, the output from the last Fourier layer v_L is passed through another neural network Q_θ, which maps the latent representation back to the physical solution space and produces the predicted solution $u(x)$:

$$u = Q_\theta(v_L)$$

This layer ensures that the model output has the same format and resolution as the ground truth solution of the PDE.

0.3.4 Summary

The FNO architecture follows a simple but powerful design:

Input function $a(x) \to$ **Latent encoding** via $P_\theta \to$ **Fourier layers for global transformation** $\to$ **Final decoding** via $Q_\theta \to$ **Predicted solution** $u(x)$

It combines the expressiveness of deep networks with the efficiency of Fourier methods, making it especially suitable for learning operators from parametric PDE problems.

0.4 The Procedure of FNO in Solving Parametric PDEs

The Fourier Neural Operator (FNO) is a **data-driven**, **supervised learning** method designed to approximate the solution operator of parametric PDEs. The overall training and inference pipeline can be broken down into the following steps:

Step 1: Prepare Labeled Training Data

FNO learns from a dataset of input-output function pairs (a, u), discretized on **a fixed regular mesh grid**.

- Let the **(regular) mesh** be denoted by $\Xi = \{\xi_1, \xi_2, \ldots, \xi_n\}$.
- Each input function a and its corresponding solution u are evaluated on Ξ, producing: $a(\Xi), u(\Xi)$.
- The dataset consists of N_{data} such pairs:

$$\mathcal{D} = \{(a^{(i)}(\Xi), u^{(i)}(\Xi))\}_{i=1}^{N_{\text{data}}}$$

Remakr: Since FNO uses Fourier transforms, it requires a **uniform grid**. Inputs a can be sampled from a function space $\mathcal{A}$, while outputs u are typically computed using accurate solvers (e.g., FEM or FDM).

Step 2: Build the FNO Model

The FNO model $\mathcal{G}_\theta$ learns to approximate the true operator $\mathcal{G}$. The forward pass through the model proceeds as:

$$\begin{aligned} v_0 &= P_\theta(a(\Xi)) \quad \text{(input lifting)} \\ v_{l+1} &= \sigma\left(\mathcal{F}^{-1}(R \cdot \mathcal{F}(v_l)) + W v_l\right), \quad l = 0, \ldots, L-1 \\ u(\Xi) &= Q_\theta(v_L) \quad \text{(projection to output)} \end{aligned}$$

Here:

- P_θ, Q_θ: input/output networks
- R, W: learnable parameters in Fourier layers
- σ: activation function (e.g., ReLU)

Step 3: Define the Loss Function

To train the model, we compare its predictions with the true solutions using the mean squared error (MSE):

$$\theta^* = \arg\min_\theta L(\theta) = \frac{1}{N_{\text{data}}} \sum_{i=1}^{N_{\text{data}}} \left|\mathcal{G}_\theta(a^{(i)}(\Xi)) - u^{(i)}(\Xi)\right|^2$$

Remark: Other loss functions (e.g., relative error, Sobolev norms) can also be used depending on the problem.

Step 4: Train the Model

The parameters θ are optimized using stochastic gradient descent (SGD) or its variants (e.g., Adam). The weights are updated iteratively as:

$$\theta_{t+1} = \theta_t - l_r \nabla_\theta L(\theta_t)$$

- l_r: learning rate
- t: training step

Training continues until the loss converges or a stopping criterion is met.

0.5 When Should We Use FNO? – Pros and Cons

0.5.1 Advantages of FNO

- **High accuracy and efficiency**: FNO typically outperforms DeepONet in both prediction accuracy and computational speed.
- **Fast inference**: Once trained, FNO can rapidly solve PDEs for any new input $a \in \mathcal{A}$, making it suitable for real-time or many-query applications.

0.5.2 Disadvantages of FNO

- **Data inefficiency**: Like other deep operator networks, FNO requires a large number of labeled training pairs (a, u), which may involve expensive numerical simulations or experiments.
- **Limited generalization**: FNO may struggle to generalize well when the input a is outside the distribution seen during training.
- **Mesh-dependence**: FNO relies on a regular grid for applying the Fourier transform, making it less flexible for problems on irregular domains or with complex geometries. It may also have reduced accuracy at non-mesh locations.