Fourier Neural Operator for Parametric Partial Differential Equations

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, Anima Anandkumar, 2020.

arXiv:2010.08895

December 21, 2020

PAPER REVIEW

Fourier Neural Operator for Parametric Partial Differential Equations

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, Anima Anandkumar, 2020. arXiv:2010.08895

Introduction

- ***Finite-dimensional operators***

Parameterize the solution operator as a deep convolutional neural network between finite-dimensional Euclidean spaces.

- ***Neural-FEM***

Model one specific instance of the PDE, not the solution operator. It is mesh-independent and accurate, but for any given new instance of the functional parameter/coefficient, it requires training a new neural network.

- ***Neural Operators***

Remedies the mesh-dependent nature of the finite-dimensional operator methods discussed above by producing a single set of network parameters that may be used with different discretizations.

- ***Fourier Transform***

Propose a neural operator architecture defined directly in Fourier space with quasi-linear time complexity.

Li et al., 2020, arXiv:2010.08895

What is Neural Operator?

Conventional PDE solvers	Neural operators
Solve one instance	Learn a family of PDE
Require the explicit form	Black-box, data-driven
Speed-accuracy trade-off on resolution	Resolution-invariant, mesh-invariant
Slow on fine grids; fast on coarse grids	Slow to train; fast to evaluate

v → input vector, u → output vector. A standard deep neural network: $u = (K_l \odot \sigma_l \odot \ldots K_1 \odot \sigma_1)v$, K → linear layer or convolution layer; σ → activation function (ReLU).

The linear transformation K is formulated as an integral operator. Let x, y be the points in the domain.

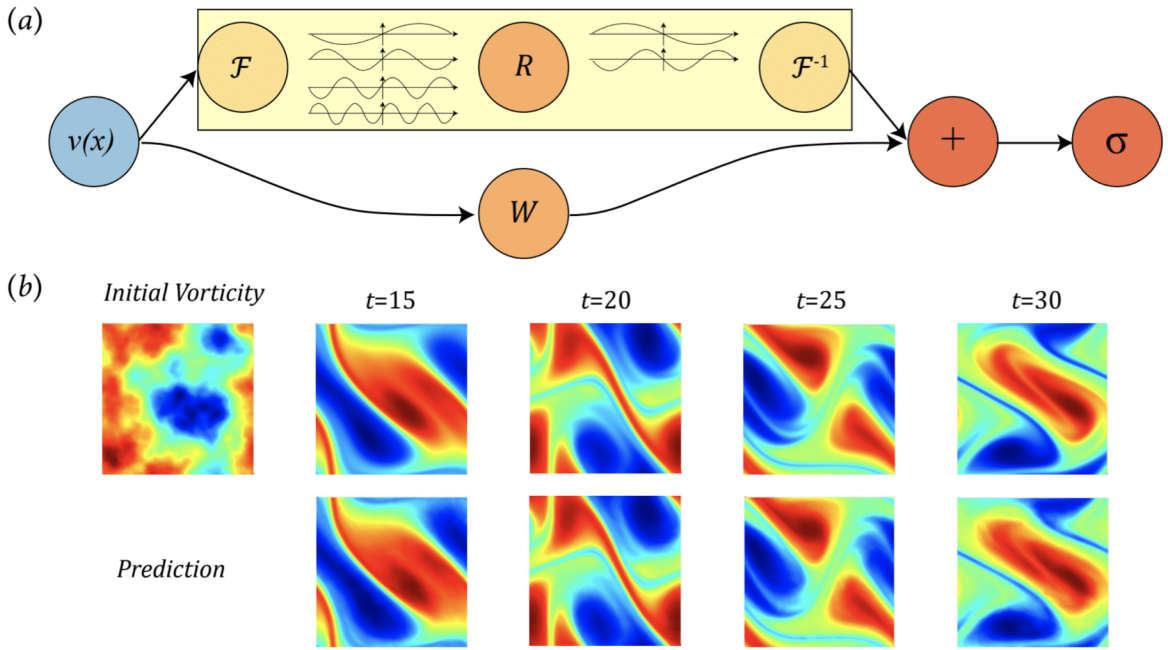
The map K: $v_t \mapsto v_{t+1}$ is parameterized as

$$v' = \int \kappa(x, y)v(y)dy + \mathrm{Wv(y)}$$

Li et al., 2020, [arXiv:2010.08895]

Learning Operators

We introduce the Fourier neural operator, a novel deep learning architecture able to learn mappings between infinite-dimensional spaces of functions; the integral operator is instantiated through a linear transformation in the Fourier domain as shown in Figure.



Li et al., 2020, arXiv:2010.08895

Learning Operators

"Our methodology learns a mapping between two infinite dimensional spaces from a finite collection of observed input-output pairs."

Learns a ***mapping*** between two infinite dimensional spaces from a finite collection of observed input-outputs.

Let $D \subset \mathbb{R}^d$ be bounded, where $\mathcal{A} = \mathcal{A}(D, \mathbb{R}^{da}), \mathcal{U} = \mathcal{U}(D, \mathbb{R}^{du})$. $G^\dagger: \mathcal{A} \to \mathcal{U}$ nonlinear map. $\{a_j, u_j\}_{j=1}^N$ observations → $u_j = G^\dagger(a_j)$. Build parametric mapping:

$$G: \mathcal{A} \times \Theta \to \mathcal{U} \quad \text{or} \quad G_\theta: \mathcal{A} \to \mathcal{U}, \theta \in \Theta$$

for some finite-dimensional parameter space Θ by choosing $\theta^\dagger \in \Theta$ so that $G(\cdot, \theta^\dagger) = G\theta^\dagger \approx G^\dagger$.

Cost function $C: \mathcal{U} \times \mathcal{U} \to \mathbb{R}$. The minimizer of the problem takes the form

$$\min_{\theta \in \Theta} \mathbb{E}_{a \sim \mu} \left[C\left(G(a, \theta), G^\dagger(a) \right) \right]$$

Li et al., 2020, arXiv:2010.08895

Learning Operators

Learning the Operator.

Approximating the operator $G^{\dagger}$; existing methods → find PDE solution ($u \in \mathcal{U}$) for parameter (a $\in \mathcal{A}$) → expensive, indirectly.

Discretization.

Let $D_j = \{x_1, \ldots, x_n\} \subset D$ → n-point discretization of the domain D; → observations: $a_j\big|_{D_j} \in \mathbb{R}^{n \times d_a}$,

$u_j\big|_{D_j} \in \mathbb{R}^{n \times d_u}$. → neural operator can produce an answer u(x) for any x ∈ D, potentially x ∉ D_j.

Li et al., 2020, [arXiv:2010.08895]

Neural Operator

Iterative updates.

$$v_t \mapsto v_{t+1}: \quad v_{t+1}(x) = \sigma\big(W v_t(x) + (\mathcal{K}(a;\varphi)v_t)(x)\big), \forall x \in D$$

where $\mathcal{K}: \mathcal{A}\times\Theta_{\mathcal{K}} \to \mathcal{L}\left(\mathcal{U}\big(D,\mathbb{R}^{dv}\big),\mathcal{U}\big(D,\mathbb{R}^{dv}\big)\right)$; $W: \mathbb{R}^{dv} \to \mathbb{R}^{dv}$ linear operator. $\sigma: \mathbb{R} \to \mathbb{R}$ activation function.

$\mathcal{K}(a;\varphi) \to$ kernel integral transformation parameterized by a neural network

$$(\mathcal{K}(a;\varphi)v_t)(x) := \int_D \kappa(x,y,a(x),a(y);\varphi)v_t(y)dy, \forall \in D$$

where κ_φ: $\mathbb{R}^{2(d+d_a)} \to \mathbb{R}^{d_v\times d_v}$ is a neural network parameterized by $\phi \in \Theta_{\mathcal{K}}$.

κ_φ plays the role of a kernel function which we learn from data.

Li et al., 2020, [arXiv:2010.08895]

Fourier Neural Operator

Let $\mathcal{F}$ denote the Fourier transform of a function $f : D \rightarrow \mathbb{R}^{dv}$ and $\mathcal{F}^{-1}$ its inverse then

$$(\mathcal{F}f)_j(k) = \int_D f_j(x)\, e^{-2i\pi\langle x,k\rangle} dx, \qquad (\mathcal{F}^{-1}f)_j(x) = \int_D f_j(k)\, e^{-2i\pi\langle x,k\rangle} dk$$

Let $\kappa_\varphi\big(x, y, a(x), a(y)\big) = \kappa_\varphi(x - y) \rightarrow (\mathcal{K}(a;\varphi)v_t)(x) = \mathcal{F}^{-1}\Big(\mathcal{F}\big(\kappa_\varphi\big) \cdot \mathcal{F}(v_t)\Big)(x)$

$$(\mathcal{K}(\varphi)v_t)(x) = \mathcal{F}^{-1}\Big(R_\varphi \cdot \mathcal{F}(v_t)\Big)(x)$$

Fourier integral operator

The Fourier transform of the periodic function: $\overline{D} \rightarrow \mathbb{R}^{d_v \times d_v}$

parameterized by $\phi \in \Theta_{\mathcal{K}}$

The discrete case and the FFT.

Weight tensor $R \in \mathbb{C}^{k_{max} \times d_v \times d_v}$

Domain D is discretized with n ∈ ℕ points → $\big(R \cdot (\mathcal{F}v_t)\big)_{k,l} = \sum_{j=1}^{d_v} R_{k,l,j}(\mathcal{F}v_t)_{k,j}$

Li et al., 2020, [arXiv:2010.08895]

Fourier Neural Operator

Let $\mathcal{F}$ denote the Fourier transform of a function $f : D \rightarrow \mathbb{R}^{dv}$ and $\mathcal{F}^{-1}$ its inverse then

$$(\mathcal{F}f)_j(k) = \int_D f_j(x)\, e^{-2i\pi\langle x,k\rangle} dx, \qquad (\mathcal{F}^{-1}f)_j(x) = \int_D f_j(k)\, e^{-2i\pi\langle x,k\rangle} dk$$

Let $\kappa_\varphi\big(x, y, a(x), a(y)\big) = \kappa_\varphi(x - y) \rightarrow (\mathcal{K}(a;\varphi)v_t)(x) = \mathcal{F}^{-1}\Big(\mathcal{F}\big(\kappa_\varphi\big) \cdot \mathcal{F}(v_t)\Big)(x)$

$$(\mathcal{K}(\varphi)v_t)(x) = \mathcal{F}^{-1}\Big(R_\varphi \cdot \mathcal{F}(v_t)\Big)(x)$$

Fourier integral operator

The Fourier transform of the periodic function: $\overline{D} \rightarrow \mathbb{R}^{d_v \times d_v}$

parameterized by $\phi \in \Theta_{\mathcal{K}}$

The discrete case and the FFT.

Weight tensor $R \in \mathbb{C}^{k_{max} \times d_v \times d_v}$

Domain D is discretized with n ∈ ℕ points → $\big(R \cdot (\mathcal{F}v_t)\big)_{k,l} = \sum_{j=1}^{d_v} R_{k,l,j}(\mathcal{F}v_t)_{k,j}$

Li et al., 2020, [arXiv:2010.08895]

Fourier Neural Operator

When the discretization is uniform → replace $\mathcal{F}$ with FFT $\hat{\mathcal{F}}$:

$$\left(\hat{\mathcal{F}}f\right)_l(k) = \sum_{x_1=0}^{s_1-1} \cdots \sum_{x_d=0}^{s_d-1} f_l(x_1, \ldots, x_d) e^{-2\pi i \sum_{j=1}^{d} \frac{x_j k_j}{s_j}}$$

$$\left(\hat{\mathcal{F}}^{-1}f\right)_l(k) = \sum_{x_1=0}^{s_1-1} \cdots \sum_{x_d=0}^{s_d-1} f_l(x_1, \ldots, x_d) e^{-2\pi i \sum_{j=1}^{d} \frac{x_j k_j}{s_j}}$$

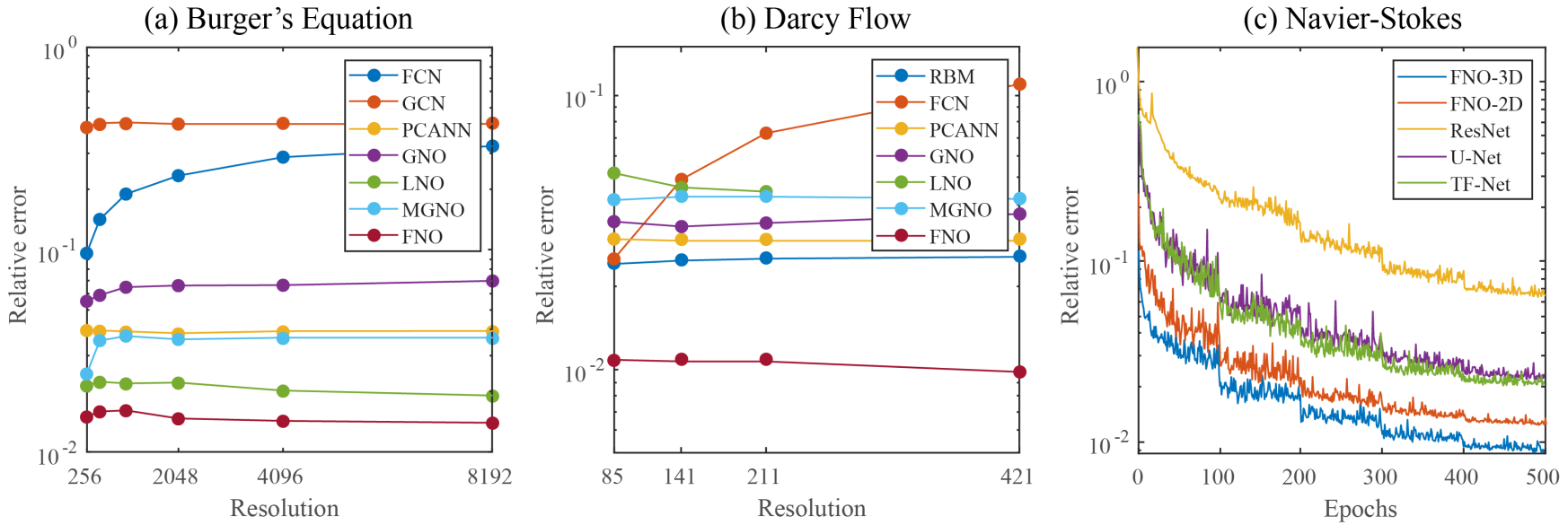
Truncated modes:

$$Z_{k_{max}} = \left\{(k_1, \ldots, k_d) \in \mathbb{Z}_{s_1} \times \cdots \times \mathbb{Z}_{s_d} \,|\, k_j \leq k_{max,j} \text{ or } S_j - k_j \leq k_{max,j}, \text{for } i = 1{:}\, d\right\}$$

Li et al., 2020, [arXiv:2010.08895]

Numerical experiments

Compare the proposed Fourier neural operator with multiple finite-dimensional architectures as well as operator-based approximation methods on the 1-d Burgers' equation, the 2-d Darcy Flow problem, and 2-d Navier-Stokes equation.



Left: benchmarks on Burgers equation for different resolutions; Mid: benchmarks on Darcy Flow for different resolutions; Right: the learning curves on Navier-Stokes $\nu = 1e{-}3$ with different benchmarks.

Li et al., 2020. [arXiv:2010.08895]

Numerical experiments

1D Burgers' equation

$$\partial_t u(x,t) + \partial_x(u^2(x,t)/2) = \nu \partial_{xx} u(x,t), \qquad x \in (0,1), t \in (0,1]$$
$$u(x,0) = u_0(x), \qquad x \in (0,1)$$

Networks	$s = 256$	$s = 512$	$s = 1024$	$s = 2048$	$s = 4096$	$s = 8192$
NN	0.4714	0.4561	0.4803	0.4645	0.4779	0.4452
GCN	0.3999	0.4138	0.4176	0.4157	0.4191	0.4198
FCN	0.0958	0.1407	0.1877	0.2313	0.2855	0.3238
PCANN	0.0398	0.0395	0.0391	0.0383	0.0392	0.0393
GNO	0.0555	0.0594	0.0651	0.0663	0.0666	0.0699
LNO	0.0212	0.0221	0.0217	0.0219	0.0200	0.0189
MGNO	0.0243	0.0355	0.0374	0.0360	0.0364	0.0364
FNO	**0.0149**	**0.0158**	**0.0160**	**0.0146**	**0.0142**	**0.0139**

Li et al., 2020. arXiv:2010.08895

Numerical experiments

2D Darcy's flow

$$
\begin{aligned}
-\nabla \cdot (a(x)\nabla u(x)) &= f(x) \qquad && x \in (0,1)^2 \\
u(x) &= 0 && x \in \partial(0,1)^2
\end{aligned}
$$

Networks	$s = 85$	$s = 141$	$s = 211$	$s = 421$
NN	0.1716	0.1716	0.1716	0.1716
FCN	0.0253	0.0493	0.0727	0.1097
PCANN	0.0299	0.0298	0.0298	0.0299
RBM	0.0244	0.0251	0.0255	0.0259
GNO	0.0346	0.0332	0.0342	0.0369
LNO	0.0520	0.0461	0.0445	–
MGNO	0.0416	0.0428	0.0428	0.0420
FNO	**0.0108**	**0.0109**	**0.0109**	**0.0098**

Li et al., 2020. arXiv:2010.08895

Numerical experiments

2D Navier-Stokes equation

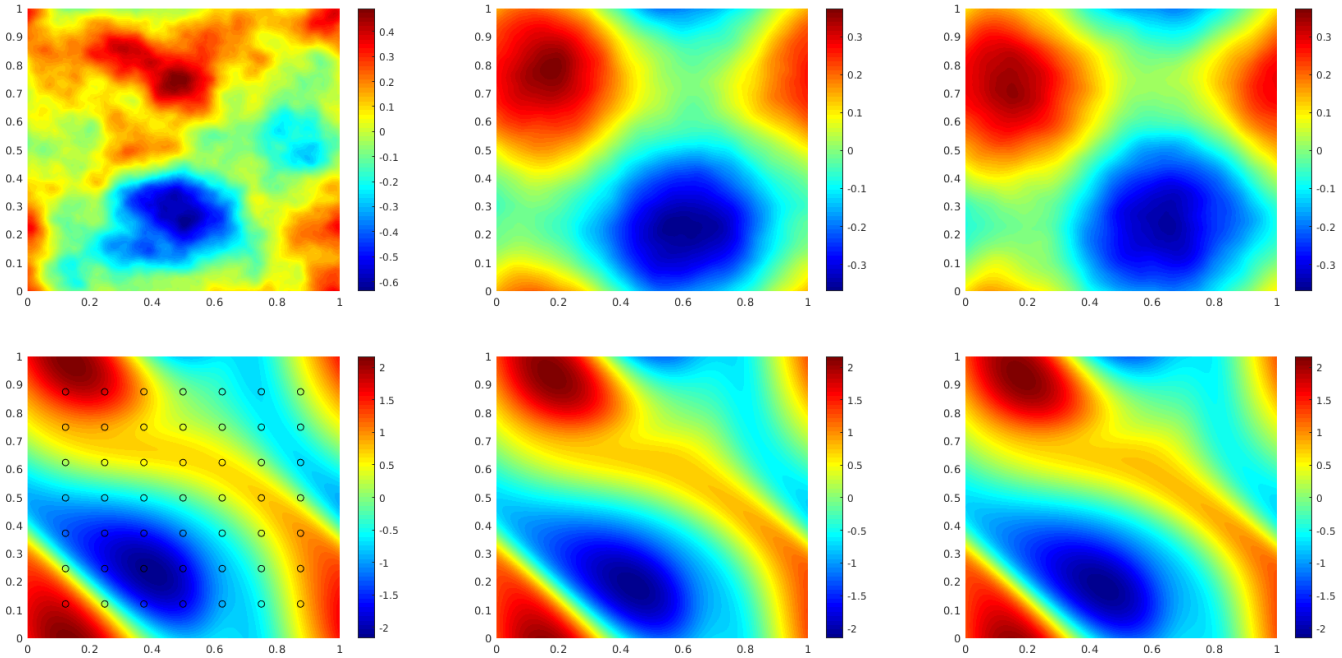
$$\begin{aligned} \partial_t w(x,t) + u(x,t) \cdot \nabla w(x,t) &= \nu \Delta w(x,t) + f(x), && x \in (0,1)^2, t \in (0,T] \\ \nabla \cdot u(x,t) &= 0, && x \in (0,1)^2, t \in [0,T] \\ w(x,0) &= w_0(x), && x \in (0,1)^2 \end{aligned}$$

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**	0.0973	**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

Li et al., 2020. [arXiv:2010.08895]

Numerical experiments

2D Navier-Stokes equation



The top left panel shows the true initial vorticity while bottom left panel shows the true observed vorticity at $T = 50$ with black dots indicating the locations of the observation points placed on a 7×7 grid. The top middle panel shows the posterior mean of the initial vorticity given the noisy observations estimated with MCMC using the traditional solver, while the top right panel shows the same thing but using FNO as a surrogate model. The bottom middle and right panels show the vorticity at $T = 50$ when the respective approximate posterior means are used as initial conditions.

Li et al., 2020. arXiv:2010.08895

REVIEW

Faster than traditional sample-targeted ML model, i.e. ResNet, PINN, etc.

Improved accuracy

More generally applied scenarios (?)

Fluctuations existed for ML models

- The End -