Lecture 6: Neural Operators 2, Theory and Practice

Chris Budd[1]

[1]University of Bath

Seattle, June 2025

Some papers to look at

- Stuart et. al. *Fourier Neural Operator for PDEs*
- Kovachi et. al. *Operator learning: algorithms and analysis*
- Liu at. al. Phil Trans
- Chen and Chen, DeepONet

Motivation: Solving PDEs

Seek to solve PDE problems of the form

$$\mathbf{u}_t = F(\mathbf{x}, u, \nabla u, \nabla^2 u) \quad \text{with BC}$$

$$u(x,0) = u_0(x) \in H^1.$$

eg.

$$-\Delta u = f(x), \quad i u_t + \Delta u + u|u|^2 = 0, \quad u_t = \Delta u + f(x,u).$$

In the previous lecture we saw how the solution $u_T(x) \equiv u(x,T)$ could be described using an operator over an infinite dimensional function space

Now use machine learning to approximate this operator over a subset of H^1

Operator based methods such as FNO

Idea: Evolutionary PDE defined for $x \in \Omega$

$$u_t = F(x, t, u, u_x, u_x x), \quad u(0, x) \equiv u_0(x) \in H^1(\Omega).$$

If $u_T(x) \equiv u(T, x)$ this *can* induce a *map* $N : H^1(\Omega) \to H^1(\Omega)$

$$N : u_0 \equiv u_T$$

- If F is linear then so is N. Otherwise nonlinear
- If F is Lipschitz then N is continuous. Otherwise properites of N are unclear and it may not even exist!
- If u satisfies a conservation law then so should N.

Neural operator methods try to learn the map N. Most literature is for the linear case.At best for the Lipshitz case

Basic construction

Assume that

$$u_T = N(u_0), \quad N : H^1 \to H^1 \quad \text{(or similar)}, \quad N \quad \text{continuous}$$

- Use a high accurate 'traditional' or PINN based numerical method to construct a training set comprising a large number of solution pairs

$$(u_0^i, u_T^i), \quad i = 1 \dots M \quad M \gg 1$$

- Train a NN to find a Neural Operator $\Psi : H^1 \to H^1$ so that

$$L = \sum_i \|\Psi(u_0^i) - u_T^i\|^2$$

  (or similar) is minimised over the training set
- Test over a suitable test set of solution pairs
- If valid, use Ψ as an emulator for the PDE, for example in control applications or uncertainty quantification or weather forecasting

Notes

- Many different architectures for NN. Many make use of latent finite dimensional structures. Examples include DeepONet and FNO.
- This is supervised learning of the operator as we are generating the pairs in advance. It differs significantly from the semi-supervised training of a PINN.
- Quality of the NN depends significantly on the quality of the training set.
- Various theorems exist on the convergence/accuracy of the Neural operator. They rely on properties of N which depend in a very subtle way on the properties of the underlying PDE. See [Kovachi et. al.]

Before we start

Some key issues with the construction of the Neural Operator

- How to construct an operator over on infinite dimensional space
- Architecture of the NN (FNO, DeepONet, CNO) etc.
- Selection of the training set (convex hulls etc.)
- Convergence as $W, L, M \to \infty$
- Building in the physics (conservation laws etc.)

Example

Simplest example is the linear heat equation

$$u_t = u_{xx}, \quad x \in \Omega.$$

Then have if $\Omega = R$ we have the convolution

$$N : u_0 \to u_1 \equiv G * u_0 \equiv \frac{1}{\sqrt{2\pi t}} \int_{-\infty}^{\infty} e^{-(x-y)^2/t} u_0(y) \, dy.$$

Also if u satisfies *periodic boundary conditions* $x \in [0, 2\pi]$

$$N : u_0(x) \equiv \sum_n a_n \, e^{inx} \to u_1(x) \equiv \sum_n e^{-n^2} \, a_n \, e^{inx}.$$

So the linear map N is defined in terms of convolutional integral operators and simple spectral operations.

The FNO: In general

The FNO architecture is based on the process of solving the linear heat equation:

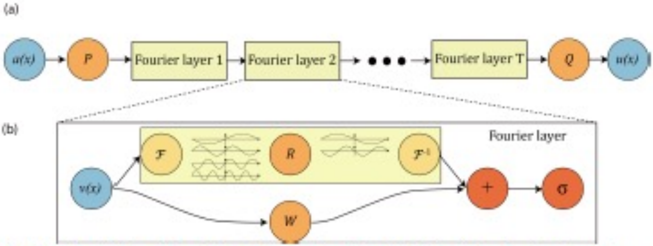
$$\Psi(a,\theta)_{FNO} \equiv Q \circ \mathcal{L}_L \circ \ldots \circ \mathcal{L}_2 \circ \mathcal{L}_1 \circ P(a).$$

$$\mathcal{L}_n(v)(x,\theta) = \sigma(W_n v(x) + b_n + K(v))$$

Here W is a pointwise linear local map. $K(v)$ is a global convolutional integral operator, kernal $G_n(\theta)$. Evaluate Kv using an FFT via

$$FFT(Kv) = FFT(G_n)\; FFT(v).$$

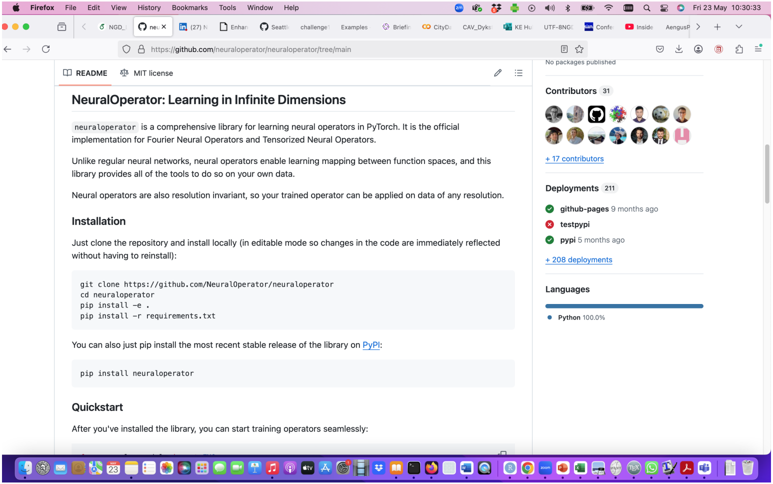
FFT restricted to M modes. Nonlinearity and higher order modes introduced via the activation function σ.



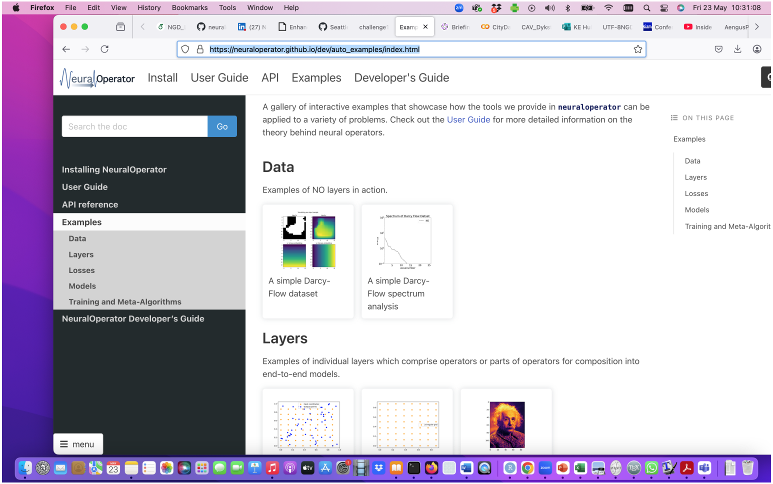
(a) The full architecture of neural operator: start from input a. 1. Lift to a higher dimension channel space by a neural network P. 2. Apply four layers of integral operators and activation functions. 3. Project back to the target dimension by a neural network Q. Output u. **(b) Fourier layers**: Start from input v. On top: apply the Fourier transform $\mathcal{F}$; a linear transform R on the lower Fourier modes and filters out the higher modes; then apply the inverse Fourier transform $\mathcal{F}^{-1}$. On the bottom: apply a local linear transform W.

Figure 2: **top:** The architecture of the neural operators; **bottom:** Fourier layer.

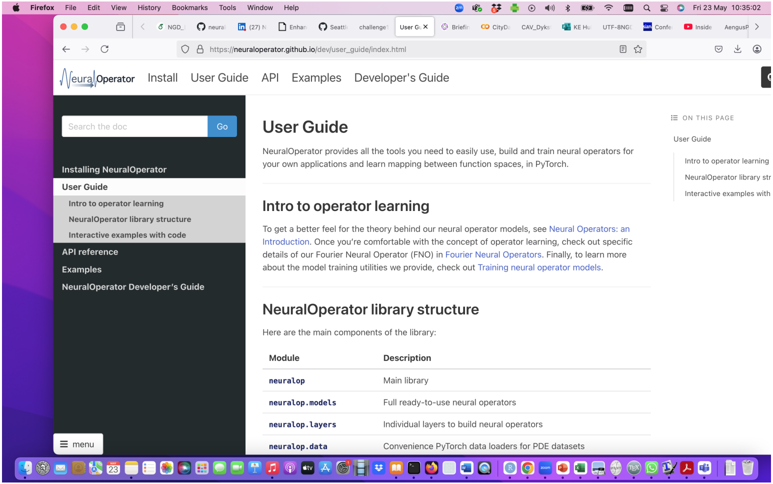
FNO online 1


Firefox File Edit View History Bookmarks Tools Window Help
Fri 23 May 10:30:33
https://github.com/neuraloperator/neuraloperator/tree/main
README
MIT license
NeuralOperator: Learning in Infinite Dimensions
neuraloperator is a comprehensive library for learning neural operators in PyTorch. It is the official implementation for Fourier Neural Operators and Tensorized Neural Operators.
Unlike regular neural networks, neural operators enable learning mapping between function spaces, and this library provides all of the tools to do so on your own data.
Neural operators are also resolution invariant, so your trained operator can be applied on data of any resolution.
Installation
Just clone the repository and install locally (in editable mode so changes in the code are immediately reflected without having to reinstall):
git clone https://github.com/NeuralOperator/neuraloperator
cd neuraloperator
pip install -e .
pip install -r requirements.txt
You can also just pip install the most recent stable release of the library on PyPI:
pip install neuraloperator
Quickstart
After you've installed the library, you can start training operators seamlessly:
No packages published
Contributors 31
+ 17 contributors
Deployments 211
github-pages 9 months ago
testpypi
pypi 5 months ago
+ 208 deployments
Languages
Python 100.0%

FNO online 2


Firefox File Edit View History Bookmarks Tools Window Help
Fri 23 May 10:31:08
https://neuraloperator.github.io/dev/auto_examples/index.html
NeuralOperator
Install
User Guide
API
Examples
Developer's Guide
Search the doc
Go
Installing NeuralOperator
User Guide
API reference
Examples
Data
Layers
Losses
Models
Training and Meta-Algorithms
NeuralOperator Developer's Guide
menu
A gallery of interactive examples that showcase how the tools we provide in neuraloperator can be applied to a variety of problems. Check out the User Guide for more detailed information on the theory behind neural operators.
Data
Examples of NO layers in action.
A simple Darcy-Flow dataset
A simple Darcy-Flow spectrum analysis
Layers
Examples of individual layers which comprise operators or parts of operators for composition into end-to-end models.
ON THIS PAGE
Examples
Data
Layers
Losses
Models
Training and Meta-Algorit

FNO online 3


Firefox File Edit View History Bookmarks Tools Window Help
Fri 23 May 10:35:02
https://neuraloperator.github.io/dev/user_guide/index.html
NeuralOperator
Install
User Guide
API
Examples
Developer's Guide
Search the doc
Go
Installing NeuralOperator
User Guide
Intro to operator learning
NeuralOperator library structure
Interactive examples with code
API reference
Examples
NeuralOperator Developer's Guide
menu
User Guide
NeuralOperator provides all the tools you need to easily use, build and train neural operators for your own applications and learn mapping between function spaces, in PyTorch.
Intro to operator learning
To get a better feel for the theory behind our neural operator models, see Neural Operators: an Introduction. Once you're comfortable with the concept of operator learning, check out specific details of our Fourier Neural Operator (FNO) in Fourier Neural Operators. Finally, to learn more about the model training utilities we provide, check out Training neural operator models.
NeuralOperator library structure
Here are the main components of the library:
Module
Description
neuralop
Main library
neuralop.models
Full ready-to-use neural operators
neuralop.layers
Individual layers to build neural operators
neuralop.data
Convenience PyTorch data loaders for PDE datasets
ON THIS PAGE
User Guide
Intro to operator learning
NeuralOperator library str
Interactive examples with

FNO in detail 1

- Input $a_j(x) \in \mathcal{A}$ output $u_j(x) = N(a_j) \in \mathcal{U}$ are functions on $x \in D \subset R^d$
- Assume have access to pointwise observations of a only at points in $x_i \in D_j \subset D$. Output u does not depend on D_j: super-resolution
- Lift a to a higher dimensional representation $v_0(x) = P(a(x))$ by a shallow NN.
- Calculate a series of updates $v_n \to v_{n+1}$ via the local W_n and global (integral) K_n operators:

$$v_{n+1}(x) = \sigma\left(W_n v_n(x) + (K_n(\theta) v_n))(x)\right).$$

- For example $\sigma = ReLU$: This introduces nonlinearity into the map.
- Project $v_L \to u(x) = Q(v_L)$
- Learn P, Q, W_n, K_n from the data pairs

FNO in detail 2

If $u(x), x \in R^d$ have Fourier Transform (FT)

$$F(u)(\omega) = \int e^{-i\omega \cdot x} u(x) \, dx.$$

If $u_k = u(x_k)$, with $x_k \in R, k = 0 \dots N-1$ have DFT approximation

$$F_j = \sum_{k=0}^{N-1} e^{-2\pi ijk/N} u_k, \quad u_k = \frac{1}{N} \sum_{k=0}^{N-1} e^{2\pi ijk/N} F_j$$

with a natural extension to $d-$dimensions.

This can be evaluated very rapidly using the FFT `np.fft.fft` in Python

Implement the FNO using the FFT to calculate all Fourier Transforms.
Efficient, but requires a uniform discretisation of $u(x)$.

If K is a convolutional kernal with Fourier Transform G then

$$FT(K * u) = FT(K)FFT(u) = GFT(u)$$

Hence if we have a FFT approximation G_j to G

Truncate the number of modes through the DFT.
The nonlinearity of the activation function σ
generates higher order modes automatically, albeit in a somewhat uncontrolled manner!

Training

- Assume input $a \in \mathcal{A}$
- In the original FNO paper take a_j as an i.i.d sequence from $\mathcal{A}$.
- Construct pairs $(a_j, N(a_j))$ using an accurate solver eg. pseudo-spectral method
- In FNO paper take $N = 1000$ training and 200 training instances. Adam optimiser to find parameters θ via:

$$min_\theta E_{a \sim \mu}[\|\psi(a,\theta) - N(a)\|]$$

- Can significantly improve training by a more careful selection of input and output pairs [Liu, B, et. al.]

More examples

Burgers Equation

$$u_t + uu_x = \nu u_{xx}, \quad \nu \ll 1, \quad \textit{periodic BC}$$

$$N : u_0(x) \to u_1(x).$$

Darcy Flow

$$-\nabla \cdot (a(x)\nabla u)) = f(x), \quad x \in [0,1] \times [0,1] \quad u = 0, \quad x \in \partial[0,1] \times [0,1].$$

$$N : a \to u.$$

More Examples:

2D Allen-Cahn equation modeling phase separation:

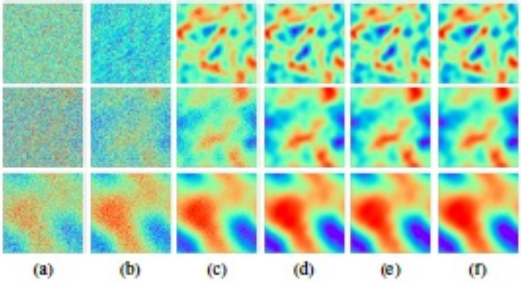
$$\begin{aligned} u_t &= u - u^3 + \epsilon^2 \Delta u, & \mathbf{x} \in [0,1]^2, t \in (0, T) \\ u(0,\mathbf{x}) &= u_0(\mathbf{x}), & \mathbf{x} \in [0,1]^2 \end{aligned} \tag{1}$$

Navier-Stokes eqns (vorticity formulation)

$$\begin{aligned} \omega_t = -u \cdot \omega_x - v \cdot \omega_y + \nu \Delta \omega + f, \quad & \mathbf{x} \in [0,1]^2, t \in (0, T) \\ \omega = v_x - u_y, \quad w(0,\mathbf{x}) = w_0(\mathbf{x}), \quad & \mathbf{x} \in [0,1]^2, \end{aligned} \tag{2}$$

Results on Allen-Cahn

Predictions of FNO trained on different datasets for the Allen-Cahn equation ($\epsilon = 0.05$). (a) inputs with noise (b) 1000 data pairs (c) 10000 data pairs (d) 1000 data pairs + 1000 generated data pairs (e) 5000 data pairs + 5000 generated data pairs (f) ground truth [with Chaoyu Liu]



Results on Navier-Stokes

Test error of neural operators on PDE datasets with 1000 and 10000 training samples.

PDE Dataset	Training Samples	FNO	UNO	CNO	UNet-FNO
2D Darcy Flow	1000	3.157e-2	3.141e-2	2.295e-2	**1.312e-2**
	10000	1.686e-2	1.770e-2	1.034e-2	**7.142e-3**
2D Navier-Stokes ($\nu = 1e-3$)	1000	7.940e-3	7.775e-3	1.090e-2	**4.250e-3**
	10000	2.626e-3	1.937e-3	2.016e-3	**1.204e-3**
2D Navier-Stokes ($\nu = 1e-4$)	1000	9.410e-2	9.282e-2	6.701e-2	**4.135e-2**
	10000	5.271e-2	5.617e-2	2.036e-2	**1.570e-2**
2D Allen-Cahn ($\epsilon = 0.05$)	1000	1.376e-2	1.646e-2	2.980e-2	**5.008e-3**
	10000	2.945e-3	2.967e-3	1.321e-2	**2.060e-3**
2D Allen-Cahn ($\epsilon = 0.01$)	1000	1.050e-2	1.511e-2	1.910e-2	**3.347e-3**
	10000	7.098e-3	1.173e-2	9.981e-3	**1.135e-3**
2D Compressible Navier-Stokes	1000	2.740e-1	2.846e-1	5.644e-1	**2.355e-1**
	10000	2.330e-1	2.089e-1	2.911e-1	**1.640e-1**

Linear Conservation Equations

- **Conservative Allen-Cahn Equation:**

$$u_t = \nabla \cdot (\epsilon \nabla u) + u - u^3 - \frac{1}{|\Omega|} \int_\Omega u - u^3 d\mathbf{x}, \quad \mathbf{x} \in \Omega, \quad t > 0,$$

  $\int u \, dx$ is conserved

- **Shallow Water Equations:**

$$\begin{cases} h_t + \nabla \cdot (h\mathbf{u}) = 0, \\ (h\mathbf{u})_t + \nabla \cdot \left(h\mathbf{u} \otimes \mathbf{u} + \frac{1}{2} g h^2 \mathbf{I} \right) = 0, \qquad \mathbf{x} \in \Omega, \quad t > 0, \end{cases}$$

Norm Conservation Equations

- **Transport Equation:**

$$u_t + \nabla \cdot (u\mathbf{v}) = 0, \quad x \in \Omega, \quad t > 0.$$

The L^2 norm of u is conserved over time.

- **Schrödinger Equations:**

$$\text{Linear:} \quad i\psi_t + \frac{1}{2}\Delta\psi + V(x)\psi = 0, \quad x \in \Omega, \quad t > 0,$$

$$\text{Nonlinear :} \quad i\psi_t + \frac{1}{2}\Delta\psi + \lambda||\psi||^2\psi = 0, \quad x \in \Omega, \quad t > 0,$$

Conservation of the L^2-norm $\int_\Omega |\psi(\mathbf{x}, t)|^2 d\mathbf{x}$. Fundamental in quantum mechanics. Also quartic-power conservation for NLS

Conservation Laws	Equation	FNO	Loss	Liu and B
Mass Conservation	Transport Equation	8.29 ± 0.12	8.16 ± 0.11	$\mathbf{8.04 \pm 0.11}$
	Conservative Allen-Cahn	2.01 ± 0.26	2.24 ± 0.34	$\mathbf{1.65 \pm 0.19}$
	Shallow Water Equation	0.26 ± 0.01	0.28 ± 0.01	$\mathbf{0.23 \pm 0.01}$
Norm Conservation	Transport Equation	8.29 ± 0.12	8.17 ± 0.16	$\mathbf{8.01 \pm 0.16}$
	Linear Schrödinger Equation	0.38 ± 0.03	0.41 ± 0.09	$\mathbf{0.32 \pm 0.02}$
	Nonlinear Schrödinger Equation	3.82 ± 1.06	3.75 ± 0.32	$\mathbf{3.02 \pm 0.51}$

Table: Prediction error (%) on test dataset for the original FNO, the FNO with conservation included in the loss function, and the adaptive correction method.

Conservation Laws	Equation	FNO	Loss	Liu and B
Mass Conservation	Transport Equation	6.74 ± 1.2	5.27 ± 1.6	$\mathbf{0.00 \pm 0.0}$
	Conservative Allen-Cahn	46.7 ± 7.4	41.7 ± 5.2	$\mathbf{0.00 \pm 0.0}$
	Shallow Water Equations	13.3 ± 1.4	9.72 ± 0.9	$\mathbf{0.00 \pm 0.0}$
Norm Conservation	Transport Equation	31.6 ± 5.4	26.2 ± 5.8	$\mathbf{0.00 \pm 0.0}$
	Linear Schrödinger Equation	2.55 ± 0.4	2.27 ± 0.5	$\mathbf{0.00 \pm 0.0}$
	Nonlinear Schrödinger Equation	13.5 ± 6.2	11.2 ± 4.7	$\mathbf{0.00 \pm 0.0}$

Table: Conservation error for the original FNO, the FNO with conservation included in the loss function, and the adaptive correction method.

Areas for improvement and research on FNO

- Observe poor conservation laws at the moment
- Generating a good training set is crucial and can be slow. How to make it good and fast?
- FNO struggles away from the training set. This is OK for MCMC emulators for UQ.
- BUT Need to broaden its scope and extend the theorems on its convergence
- NONE of this theory applies, for example, to the nonlinear heat equation

$$u_t = u_{xx} + u^2.$$

DeepONet

Idea [Chen and Chen] :Approximate map $N : \mathcal{U} \to \mathcal{V}$

$$N \approx \psi = G_{\mathcal{V}} \circ NN \circ F_{\mathcal{U}}$$

By training the NN with encoder $F_{\mathcal{U}}$ and decoder $G_{\mathcal{V}}$.

- Make use of principal component analysis (PCA)
- Input space $\mathcal{U}$ with input (functions) from a probability measure μ.
- Construct principal components from the Covariance of μ.

See [Broomhead and King 1983] for the first use of this idea to reconstruct a dynamical system from data, coupled to a radial basis function architecture

- PCA basis functions ϕ_i
- Encoder $F_{\mathcal{U}}$ a linear map $L : \mathcal{U} \to R^{d\mathcal{U}}$ (typically) projects onto first $d\mathcal{U}$ basis functions

$$Lu_i = \langle u, \phi_i \rangle.$$

- $$\mathbf{a} = Lu$$

  is used as the input to a finite dimensional feed forward NN output $\mathbf{b}$
- decoder $G_{\mathcal{V}}$ given by

$$G_{\mathcal{V}} = \sum_j b_j \, \phi_j$$

- Train NN on a set of input/output pairs

Convergence [Kovachi et. al.]

- FNO and DeepONet can approximate a wide variety of operators
- Assume that input space $\mathcal{U}$ is a separable Banach space and the map N is compact
- Prove convergence on any finite dimensional set using the universal approximation theorem
- Take an appropriate limit (approximation theory of Banach spaces which applies to the sets over which PDEs are typically formulated)

Summary

- PINNS and NOs both show promise as a quick way of solving PDEs but have only really been tested on quite simple problems so far
- PINS not (yet) competitive with FE in like-for-like comparisons
- PINNs need careful meta-parameter tuning to work well
- NOs proving more promising. Now used for weather forecasting!
- Long way to go before we understand PINNS or NOs completey and have a satisfactory convergence theory for them in the general case.
- Lots of great stuff to do!