Fourier Neural Operators (FNO)

Tuesday, October 29, 2024 1:37 PM

- Based on the 2021 ICLR paper **Fourier Neural Operator for Parametric Partial Differential Equations** by Li, Zongyi, Kovachki, Nikola, Azizzadenesheli, Kamyar, Liu, Burigede, Bhattacharya, Kaushik, Stuart, Andrew, and Anandkumar, Anima [Caltech group]
  - https://github.com/neuraloperator/neuraloperator, https://zongyi-li.github.io/neural-operator/ project page
  - See their blog post https://zongyi-li.github.io/blog/2020/fourier-pde/
  - Extends the group's prior 2021 work **Neural Operator: Graph Kernel Network for Partial Differential Equations**
    - see their blog post https://zongyi-li.github.io/blog/2020/graph-pde/
    - and JMLR journal version 2023, https://jmlr.org/papers/v24/21-1524.html **Neural Operator: Learning Maps Between Function Spaces**
  - They have followups, like **Physics-informed neural operator for learning partial differential equations**
  - Andrew Stuart has followups on "Operator Learning Using Random Features: A Tool for Scientific Computing" (SIAM Review, 2024)
  - Anima Anandkumar has followups on "Physics-Informed Neural Operator for Learning Partial Differential Equations" (PINO) and
- For comparisons with other methods
  - Ch 6 of the book *Machine Learning in Modeling and Simulation Methods and Applications* (Timon Rabczuk and Klaus-Jürgen Bathe, eds.) "**Physics-Informed Deep Neural Operator Networks**" by Somdatta Goswami, Aniruddha Bora, Yue Yu, and George Em Karniadakis
    - compares DeepONets with FNO etc., and describes physics-informed DeepONets
  - **Operator Learning: Algorithms and Analysis**, by Nikola B. Kovachki, Samuel Lanthaler, Andrew M. Stuart (2024, https://arxiv.org/abs/2402.15715)
    - Math focus; describes PCA-Net, DeepONet, FNO, Random Features
  - **A mathematical guide to operator learning** by Nicolas Boullé and Alex Townsend (2023, https://arxiv.org/abs/2312.14688)
    - Discusses DeepONets, FNO, Deep Green networks, Graph neural op, etc.
    - Makes connection with recovering structured matrices

Idea of "operator learning"

standard neural nets learn a mapping from X to Y

$\hat{y} = NN(x), \quad x \in X, \hat{y} \in Y$

finite dim. vector spaces

using a mixture of:

(1) linear/affine layers] "GLOBAL"

(2) pointwise nonlinearities] "LOCAL"

Operator learning learns a map from X to Y

function spaces (∞-dim. vector spaces)

FNO (p. 2)

Tuesday, October 29, 2024 2:07 PM

First task: what are "linear layers"?

Def: let X, Y be (real) vector spaces. A transformation/operator ("T") is linear if $\forall f, g \in X \quad \forall \alpha \in \mathbb{R}$, then

$$T(f + \alpha g) = T(f) + \alpha T(g)$$

f a function, $h = T(f) \in Y$ also a function.

Ex T = differentiation, $T(f) = f'$

eg. f is the function defined as $f(x) = 3x^2$

then $(Tf)(x) = 6 \cdot x$

Ex kernel operator / integral operator

$$T(f)(x) = \int_{\Omega} \underbrace{K(x,y)}_{\text{"kernel"}} f(y)\, dy$$

eg. $K(x,y) := \begin{cases} 1 & y \le x \\ 0 & y > x \end{cases}$ then

$$(Tf)(x) = \int^{x} f(y)\, dy$$

is anti-derivative

Finite-rank operators

We say a linear op. T is finite rank if $\dim(\text{range}(T)) < \infty$

- differentiation, integration are not finite rank
- an integral operator with $K(x,y) = \sum_{j=1}^{r} \varphi_j(x)\, \psi_j(y)$ is finite-rank

since

$$Tf \in \text{span}\{\varphi_j\}_{j=1}^{r}$$

Compact operators

(in other spaces it's more complicated)

If X, Y are Hilbert space, we say a linear operator T is compact iff T is the limit (in an appropriate sense) of finite rank operators

FNO (p. 3)

Tuesday, October 29, 2024 2:20 PM

Informally, compact operators (bounded, linear) are operators that we can approximate on a computer (and finite rank operators are dense among compact op.)

Also, finite rank op. are dense among a class of compact operators known as "trace class" which are equivalent to the class of integral operators (wikipedia)

$\Rightarrow$ Any reasonable* (compact) operator we wish to approximate on the computer can also be approximated by an integral operator.

* excludes differentiation!

So, we'll choose all linear operators in our neural operator network to be integral operators.

Motivation, part 2: GREEN'S FUNCTIONS → modern name is "Fundamental Solution"

Suppose we have a linear differential equation

$$(*) \qquad \mathcal{L}(u) = f, \quad u = 0 \text{ on } \partial\Omega, \quad u = u(x) \text{ (could include time)}$$

If $\mathcal{L}$ is linear and nice enough (uniformly elliptic or parabolic; cf. Evans, §2.2.4 2010 book)
then if G is the Green's Function
which solves

$$\mathcal{L}(G(\cdot, y)) = \delta(y - \cdot) \quad \text{independent of "f"}$$

then $$u(x) = \int_\Omega G(x, y) f(y) dy$$

Why? Assuming we can commute $\mathcal{L}$ and $\int$ then

$$(\mathcal{L}u)(x) = \int \mathcal{L}(G(x,y) f(y)) dy$$
$$= \int \delta(y - x) f(y) dy \quad \text{convolution}$$
$$= \int \delta(y) f(y + x) dy$$
$$= f(x) \quad \checkmark$$

... so "solution operators" (mapping $f \mapsto u$ that solves $*$) can be written as integral operators in this case.

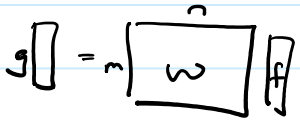
Another reason to use integral operators

FNO (p. 4)

Tuesday, October 29, 2024 3:17 PM

Motivation, part 3: analogy

$$\vec{g} = W \cdot \vec{f}$$



Finite dim.

$$g[x] = \sum_{y=1}^{n} W_{x,y} \cdot f[y] \quad \text{for } x \in \{1, 2, \ldots, m\}$$

∞ dim.

$$g = T(f) := \int_{\Omega} K(\cdot, y) f(y)\, dy$$

i.e. $$g(x) = \int_{\Omega} K(x, y) \cdot f(y)\, dy \quad \text{for } x \in \Omega$$

... going even further...

some differential equations have Green's functions of the form

$$K(x,y) = \tilde{K}(x-y)$$

↑ usually write "K" still

...which means $\int K(x,y) f(y) dy$ is a convolution.

Back to the paper:

Goal is to learn the solution operator to a (parametric) PDE

$$G_\theta : \mathcal{A} \to \mathcal{U}$$

$\mathcal{U}$: function space of PDE solutions

$\mathcal{A}$: parameters of PDE

θ: trainable neural net parameters

Ex: 1D viscous Burger's Eq. $u = u(x,t)$, $x \in (0,1)$ $t \in (0,1]$

$$\frac{\partial}{\partial t} u + \frac{\partial}{\partial x}\left(\frac{1}{2} u^2\right) = \nu \frac{\partial^2}{\partial x^2} u$$

$u(x, t=0) = u_0(x)$ initial condition, u spatially periodic (boundary condition)

Goal: map u_0 $(\in \mathcal{A})$ to $u(\cdot, t=1)$ (1D function)

Ex: 2D Darcy Flow (steady-state)

$$-\vec{\nabla} \cdot (a(x) \nabla u(x)) = f(x) \qquad x \in (0,1)^2 \subseteq \mathbb{R}^2, \quad u(x) = 0 \text{ on boundary}$$

Goal: map diffusion coefficient a $(\in \mathcal{A})$ to u (2D function)

FNO (p. 5)

Tuesday, October 29, 2024 4:15 PM

Example: Navier-Stokes 2D, viscous, incompressible in vorticity form

$$\frac{\partial}{\partial t} w + u \cdot \nabla w = \nu \Delta w + f \qquad x \in (0,1)^2, \quad t \in (0, T]$$

(ν: viscosity; w: vorticity; u: velocity)

$$w := \nabla \times u$$

$$\nabla \cdot u = 0 \quad \text{(incompressible)}$$

$$w(x, t=0) = w_0(x)$$

Goal: map vorticity at time up to $t = 10$ (includes w_0, plus more) to vorticity up to some later time T ($T > 10$)

"Fourier layer"

FNO layer

$$\vec{v}_{t+1}(x) := \sigma\left(W \vec{v}_t(x) + T_\theta(\vec{v}_t; a)(x) \right)$$

(σ: "local" (pointwise) nonlinearity; t: layer number, not time; $W \vec{v}_t(x)$: "local"; $T_\theta(\vec{v}_t; a)(x)$: integral operator)

$$T_\theta(\vec{v}; a)(x) = \int K_\theta(x, y, a(x), a(y)) v(y) \, dy$$

(global since all y; most general form)

$$T_\theta(\vec{v})(x) = \int K_\theta(x - y) v(y) \, dy$$

(what they actually do) — convolution!

learn neural net weights θ to parameterize K_θ

... do this via parameterizing in Fourier Space: (parameterize this directly)

$$T_\theta(v)(x) = \mathcal{F}^{-1}\left(R_\theta \cdot \mathcal{F}(v) \right)(x)$$

($\mathcal{F}^{-1}$: inverse Fourier; $\mathcal{F}$: Fourier transform)

"Assume" K is periodic, so it has discrete Fourier Series, which we then truncate to low(ish) frequencies so it's finite. ($|k| \le k_{max}$; k = frequency mode)

$\vec{v}(x) \in \mathbb{R}^{d_v}$, discretize input to n points, $v \in \mathbb{R}^{n \times d_v}$

Use FFT to compute $(\mathcal{F} v)[k, j]$ $|k| \le k_{max}$ $j = 1, \ldots, d_v$

then $R_\theta \mathcal{F}(v)$ is

$$(R_\theta \mathcal{F}(v))[k, l] = \sum_{j=1}^{d_v} R_{k,l,j} \cdot (\mathcal{F} v)[k, j]$$

("θ" is this tensor of size (#frequencies) $\times$ $d_v \times d_v$)

FNO (p. 6)

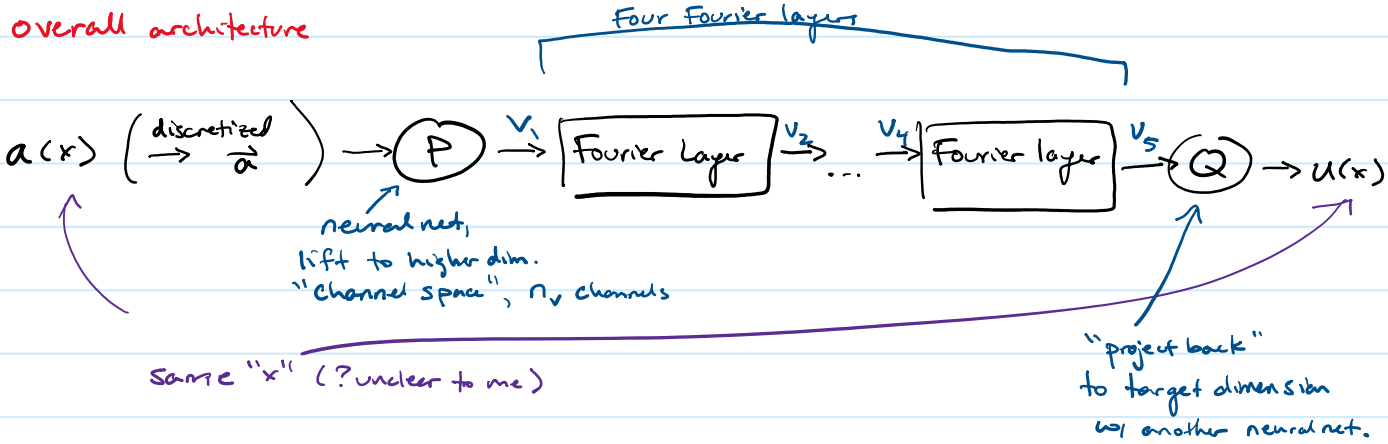
Tuesday, October 29, 2024 4:46 PM

like DeepONets, while we can evaluate output u at any point x, our input function a (on domain D) seems to need a consistent discretization w/ n points, ??

(and simple domain + uniform discretization if we want to use FFT)

but their paper claims "discretization invariant" since learning weights is in Fourier Space, i.e. wavenumber k_{max} is an absolute number, invariant to discretization.

Overall architecture



Universal Approximation ... yes, if periodic, but size of network can grow exponentially fast w/ ε unless Fourier coefficients decay fast enough (i.e. input is smooth)

cf. Kovachki, Lanthaler, Mishra. JMLR '21

Limitations Assumes input, output are periodic (and same domain !?) — so simple domains too

and uniform discretization to use FFT

Experiments 2D Navier Stokes