Neural Operator methods: p. 1, DeepONets

Monday, October 21, 2024 10:43 AM

(many competing research groups lay claim to this terminology and fortunately use it pretty similarly

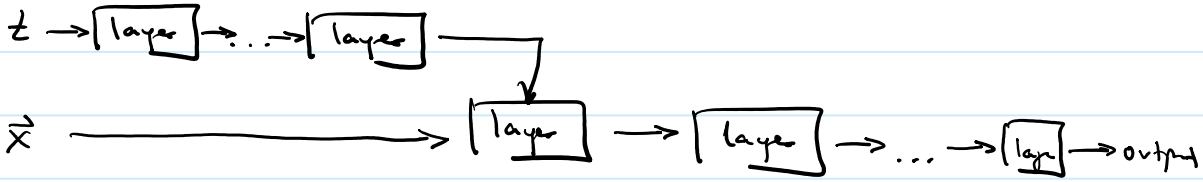
Background: Hypernetworks (Ha, Dai, Li, arXiv '16, ICLR '17)

to model $u(t, \vec{x})$ via implicit neural representation (à la SIRENs)

they think of

$t \longmapsto$ network weights

network: $x \to$ output



DeepONets

"Inspired by" universal approximation thm. for operators (and POD / low-rank representations), though no actual guarantees on training or generalization (or even expressiveness w/ concrete hyperparams).

Applicable for inverse problems:

Stage 1: slow "offline" training to generate

$$\{u_{\theta_k}(y_j)\}_{k,j}$$ of many solutions to PDE

at different parameters θ_k ← all must be same size

Stage 1a: collect training data (slow, use classical forward solvers)

Stage 1b: train neural net (also slow, up to 75 hours in paper)

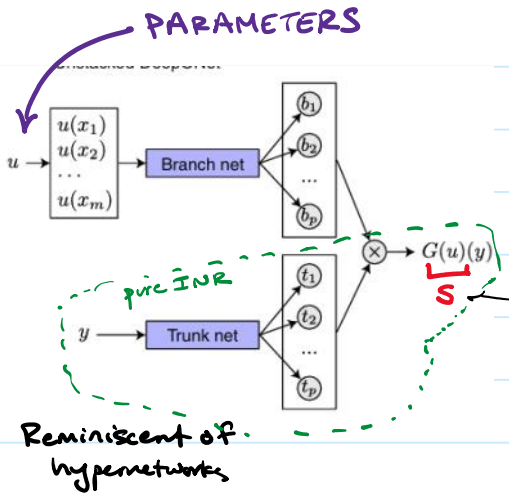
Stage 2: use ML model to predict output for new θ

cheap (and hopefully accurate)

...So you'd never do this if you just want to solve a new PDE once (for a single θ)...

DeepONets (p. 2)

Wednesday, October 23, 2024 9:23 AM



Different notation than before!

Ex. 1 Generic ODE system, Solve for $s(x)$

$$s' = g(s(x), u(x), x) \qquad s(0) = s_0$$

$$= -s^2(x) + u(x)$$ in their numerical experiment.

Find s, evaluate at arbitrary point y

We wish to solve a class of problems, eg. ODE of form for all parameters u

Ex 1 simplified ("Problem 1.A") let $s' = u$ so $s(y) = s_0 + \int_0^y u(x)\,dx$

Think of it as learning the operator G (in ↑ this case, it's the antiderivative operator)

$G(u)$

output is a function s,
can evaluate it anywhere (at "y")

Training Data

$$\{ (u_i, y_k, G(u_i)(y_k)) \}_{i=1}^{500 \text{ to } 10{,}000} \quad k=1 \text{ (depending on example: 1, 20, 100 or 2000)}$$

$G(u_i)(y_k)$: computed via classical forward solver (slow to generate!)

each u_i is a function

(ex: $u_{10}(x) = \sin(10 \cdot x)$)

① how to choose good training u?

They try drawing Gaussian processes (so smooth), so well-defined distribution, so it's clear how to test (not clear if it's useful / realistic)

② how to discretize u?

Choose fixed "sensor" points $\{x_k\}_{k=1}^{m}$

$m \approx 20$ to 100 (use more if less smooth / more oscillatory)

DeepONets (p. 3)

Wednesday, October 23, 2024 9:54 AM

Sometimes called "POD-DeepONet"

How they think about it

$$(*) \quad G(u)(y) \approx \sum_{l=1}^{\text{rank}} \underbrace{b_l(u)}_{\text{branch}} \cdot \underbrace{t_l(y)}_{\text{trunk}}$$

Sum of separable terms

after discretizing, a function of $\vec{u} = (u(x_1), \dots, u(x_m))$

and cite some universal approx. thm for operators (see Nelsen, Stuart SIREV p.539 for a long list of refs)

good news **Theorem 2** (DeepONet paper)

X a Banach space (eg. functions u), $K_1 \subset X$ compact

$K_2 \subseteq \mathbb{R}^d$ compact a domain, V a compact set of $C(K_2)$,

assume $G: V \to C(K_2)$ is continuous (hence uniformly cts!)

then $\forall \varepsilon > 0$, if rank is sufficiently large,

$(*)$ is uniformly accurate in u and y, up to ε

i.e. $\forall u \in V, \forall y \in K_2, \quad \left| G(u)(y) - \sum_{l=1}^{\text{rank}} b_l(u)\, t_l(y) \right| < \varepsilon$

(in the sense that $\exists$ b, t functions.

To get them as neural nets, "recurse" and use standard universal approx. thm)

bad news Quantitatively, even if Lipschitz cts (stronger than uniformly cts), # of parameters grows exponentially with ε^{-1} (in the worst case)

but we can get lucky!

See Nelsen, Stuart

The DeepONet framework is very broad

eg. ch 6 in "ML in Modelling + Sim", eds. Rabczuk, Bathe

- their followup papers claim it subsumes other group's approaches
- ... but it's so general, it's unclear if it counts as a specific algorithm. Lot's of hyperparameters
- output points y could be anywhere, but input "sensor" points $\{x_k\}_{k=1}^m$ cannot change. So not fully discretization invariant
- output has some structure, other works (vaguely) claim that limits it

DeepONets (p. 4)

Wednesday, October 23, 2024 9:42 AM

DeepONet Results

- Between 1 and 100 GPU hours of computing to train!
- Early stopping to prevent overfitting
- Some results discard 1% worst predictions!
  Sometimes it just fails!
- Architectures simple, depths 2 to 4, widths ≤ 100, ReLU

Chapter 6
Physics-Informed Deep Neural Operator Networks

Somdatta Goswami, Aniruddha Bora, Yue Yu, and George Em Karniadakis

→ Also describes "physics-informed" hybrid

Ex: Darcy Flow (adapting notation to original DeepONet paper)

$$-\vec{\nabla}\cdot(K(x)\cdot\nabla s(x)) = f, \quad s(x)=0 \text{ on } \partial\Omega, \quad \Omega=[0,1]^2\subset\mathbb{R}^2$$

K: parameter "u" conductivity; s: "s" is hydraulic head (≈ pressure); f: f is source

models sub-surface (ground) flow of a liquid (water) through a porous medium (dirt)

100 samples of K's, all piecewise constant (via thresholding a spatially correlated random field)
16 layer deep network

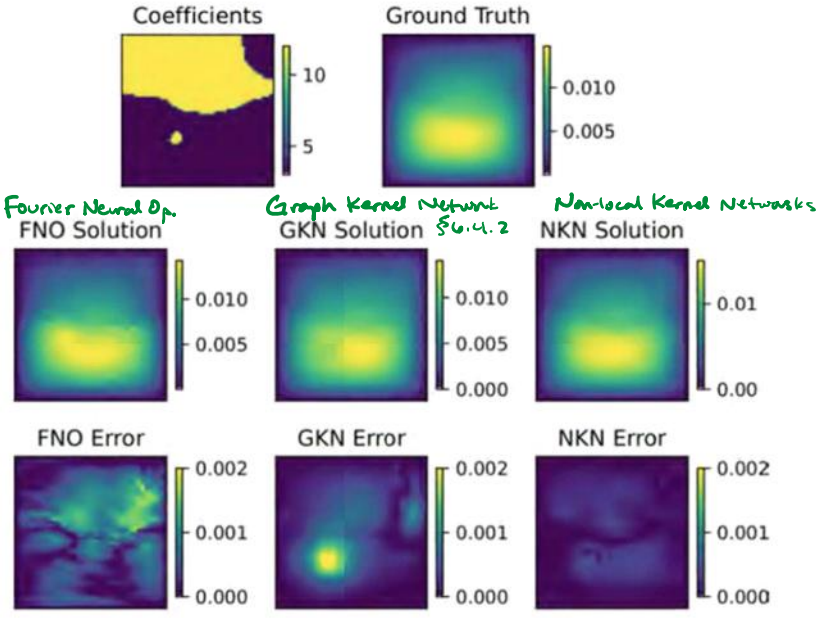


Fig. 6.7 2D Darcy flow in a square domain. A visualization of 16-layer FNO, GKN, and NKN performance on an instance of conductivity parameter field $K(\mathbf{x})$, when using (normalized) "coarse" training dataset ($\Delta x = 1/15$) and test on the "finer" dataset ($\Delta x = 1/60$). Here, the absolute point-wise error $|u^{data}(\mathbf{x}) - u^{pred}(\mathbf{x})|$ is plotted

] all "special types" of DeepONets, in a sense