Deep Learning in Scientific Computing 2023: Lecture 9

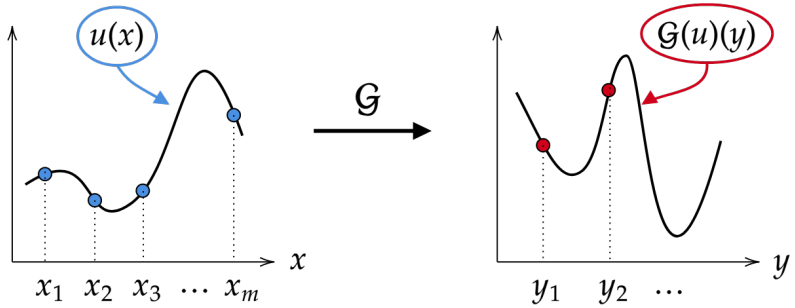
Siddhartha Mishra

Seminar for Applied Mathematics (SAM), D-MATH (and),
ETH AI Center,
ETH Zürich, Switzerland.

Operator Learning

Operator: Input ⟶ Output

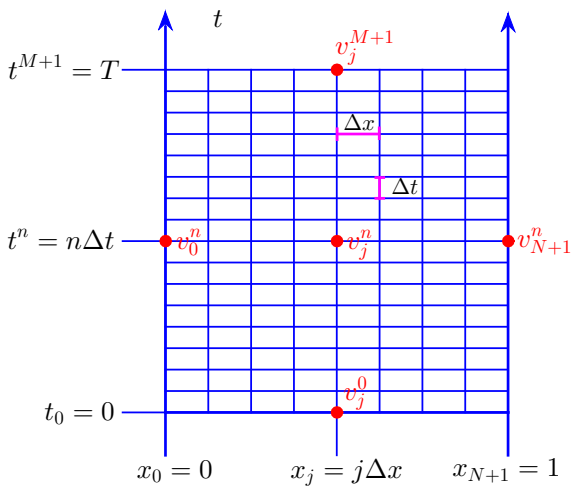
Function → Continuous objects



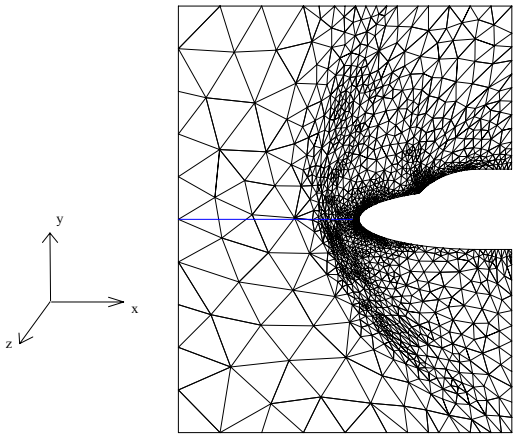
- Underlying Solution Operator: $\mathcal{G}(a) = u$ for PDE $\mathcal{D}u = a$ | ODE
- Task: Find a Surrogate (based on DNNs) $\mathcal{G}^* \approx \mathcal{G}$ from data.
- Inputs+Outputs for $\mathcal{G}^*$ are Functions.
- Some notion of Continuous-Discrete Equivalence

Practice; — discrete representations

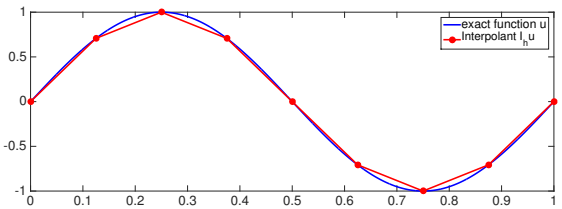
Traditional Numerical Methods



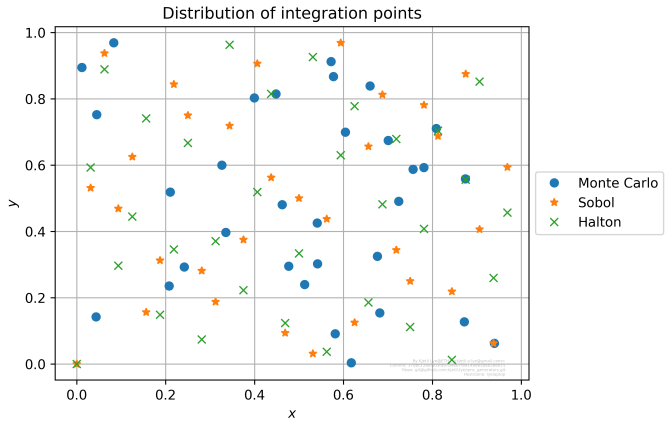
Finite Difference



Finite Volume



Finite Element

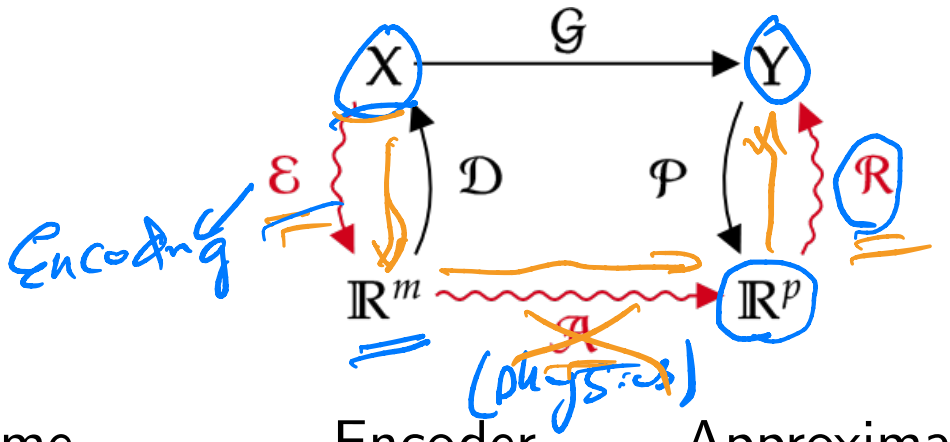


Collocation

Revisiting Numerical Methods

$G(u) \approx R \circ A \circ \mathcal{E}(u)$

- Can be reinterpreted in the following abstract Paradigm:



Scheme	Encoder	Approximator	Reconstructor
Finite Difference	Point values	Scheme	Poly. Interpolant
Finite Element	Node Values	Scheme	Galerkin Basis
Finite Volume	Cell Averages	Scheme	Poly. Interpolant
Spectral	Fourier Coeffs.	Scheme	Fourier Interpolant

Recall: Aim is to Learn Operators from Data

A First OpLearn Architecture

Spectral Neural Operator

Given an Input $a \in X$

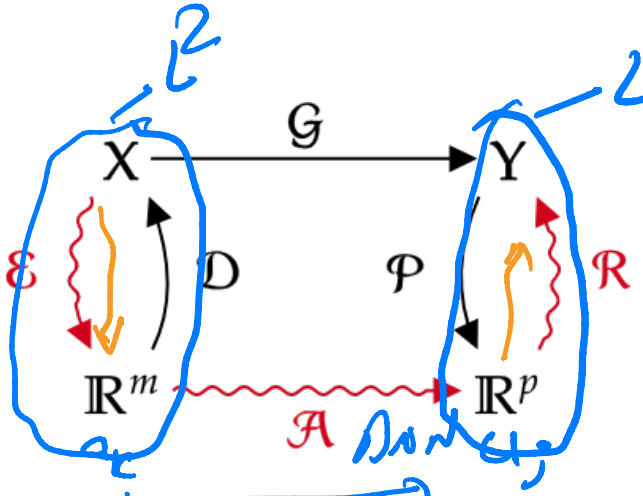
Prior: Fix a basis for X, Y a priori

$$a = \sum_{k} a_k \varphi_k(x)$$

$$a \approx \sum_{k=1}^{m} a_k \varphi_k(x)$$

$$\mathcal{E}a = \{a_k\}_{k=1}^{m}$$

$$\varphi_k(x) = e^{2\pi i k x} \quad (\cos(k\pi x), \sin(k\pi x))$$



Architecture	Encoder	Approximator	Reconstructor
SNO[1]	Coeffs	DNNs	Fourier/Chebychev basis

DNNs are universal

$$\{u_j\}_{j=1}^{p}$$

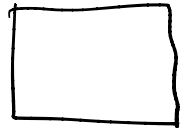
$$R\left(\{u_j\}\right) \longmapsto \sum_{j=1}^{p} u_j \varphi_j(x)$$

[1] Fanaskov and Oseledets, 2022

SNO: Input a Output a^*

Encoding: $a \longmapsto \mathcal{E}a = \{a_k\}_{k=1}^m$ — m-Fourier coefficients of a:

$$\mathcal{F}a(k) = \int_D f(x) e^{-2\pi i k \cdot x} dx$$

$D =$  DFT (FFT)

Approximation $\{a_k\}_{k=1}^m \xrightarrow{DNN} \mathcal{A}\{a_k\} = \{u_k\}_{k=1}^p$

Reconstruction: $u^* = \sum_{k=1}^{p} u_k e_k(x)$

Generalized SNO → freedom to choose any bases, a priori:

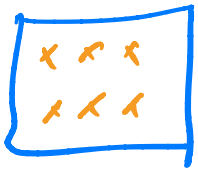
X, Y

$\{\varphi_k\}$ $\{\psi_j\}$

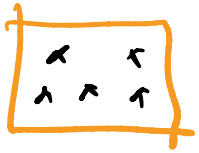
Issue: finding Basis a priori

Operator Networks

Idea → learn the basis from data too !!!
using DNNs

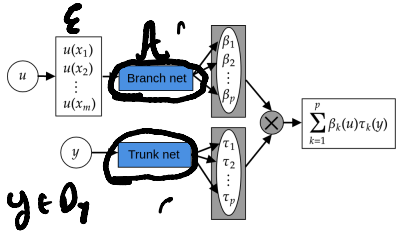


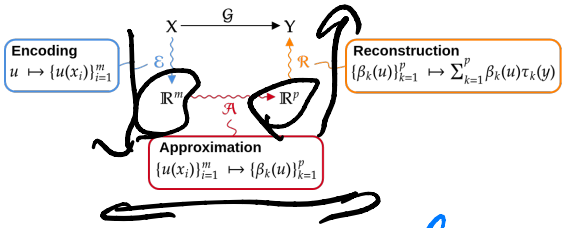
- DeepONets: Chen, Chen 1995, Lu et al, 2020:



Reconstruction $\mathcal{N}(a)(y) = \tau_0(y) + \sum_{k=1}^{p} \beta_k(a)\tau_k(y) \approx \mathcal{G}(a)(y)$

$R[\beta_k] \mapsto \sum_{k=1}^{p} \beta_k \tau_k$ — DNNs





Encoding Input a, choose a set of points $\{x_i\}_{i=1}^{m} \in D$

$\mathcal{E}a = \{a(x_i)\}_{i=1}^{m} = \{a_i\}$

$A(a_i) \mapsto \{\beta_k\}_{k=1}^{p}$ — DNN

Deep Onets: $G: X \longmapsto Y$

Domain D_X → Domain D_Y

On D_X → fix point set ~ $\{x_i\}_{i=1}^{m}$ ~ Random / Uniform grid points

$\mathcal{E}: a \in X \longmapsto \{a_i = a(x_i)\}_{i=1}^{m}$

$A: \{a_i\}_{i=1}^{m} \xrightarrow{DNN} \{\beta_k\}_{k=1}^{p}$

Branch-net

R: DNN - Trunk-net: $y \in D_Y \xrightarrow{DNN} \{\tau_k(y)\}_{k=1}^{p}$

$$R(\beta_k, \tau_k) \approx \sum_{k=1}^{p} \beta_k \tau_k$$

Deep Onet $\mathcal{D} = R \circ A \circ \mathcal{E}(a)$

$$y \longmapsto \{\tau_k(y)\}_{k=1}^{p}$$

How to train DeepONets

Operator: $G : X \longmapsto Y \quad a \longmapsto u = Ga, \ \mu \in Prob(X)$

Data: $a_i \sim \mu \quad u_i = Ga_i + \eta$ (noise), $\eta \equiv 0$

Data pair (samples) $(a_i, u_i)_{i=1}^{M}$

DeepOnet | SNO | Oplearn: $G^* \quad a \longmapsto G^* a = u^*$

Mismatch

Idealized loss $= \int_X \| Ga - G^* a \|_Y \, d\mu(a)$

(X: $\| \cdot \|_Y$ → function space)

Empirical loss: Replace $\mu \approx \frac{1}{m} \sum_{i=1}^{m} \delta_{a_i}$ (Monte-Carlo)

$$\mathcal{L} = \frac{1}{m} \sum_{i=1}^{m} \| Ga_i - G^* a_i \|_Y$$

$u_i = Ga_i$ no noise

Loss $\mathcal{L} \simeq \frac{1}{m} \sum_{i=1}^{m} \| Ga_i - G^\dagger a_i \|_Y$

$= \frac{1}{m} \sum_{i=1}^{m} \| u_i - G^\dagger a_i \|_Y$ (1)

Assume: $Y = L^p(D_Y)$ $\quad 1 \le p < +\infty$, $p = 1, 2$

$p = 2$ $\quad \| u_i - G^\dagger a_i \|_{L^2}^2 = \int_{D_Y} | u_i(y) - G^\dagger a_i(y) |^2 dy$

Use Quadrature: $\int f(x) dx \simeq \sum_{j=1}^{J} \omega_j f(x_j)$

Discrete loss:

$$\mathcal{L} \simeq \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{J} \omega_j | u_i(y_j) - G^\dagger a_i(y_j) |^2$$

Loss $\alpha = \frac{1}{m} \sum_{j=1}^{m} \sum_{i=1}^{s} \omega_j \left| u_i(x_j) - G^* a_i(y_j) \right|^2$

Algorithm : Given Inputs a_i | Data u_i (ground truth)

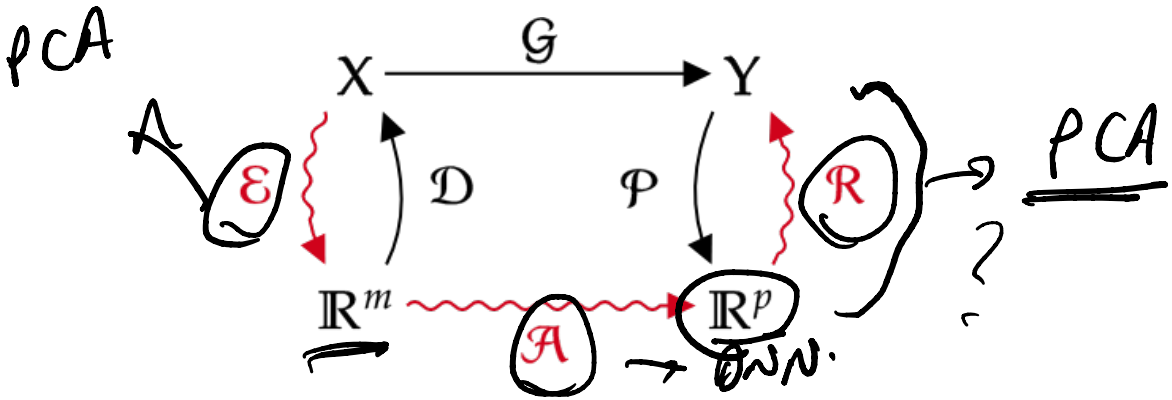
u_i need to be evaluated at quadrature points

Run DeepOnet on a_i to obtain $G^* a_i$

$G^* a_i$ can be evaluated at $\{x_j\}$

Compute the loss-term for samples and ξ

Another OpLearn Architecture



Architecture	Encoder	Approximator	Reconstructor
SNO[2]	Coeffs	DNNs	Chebychev basis
DeepOnet	Sensor Evals.	DNNs	DNNs
PCA-Net[3]	Input PCA	DNNs	Output PCA

Data driven Dimension Reduction algorithm

[2] Fanaskov and Oseledets, 2022

[3] Bhattacharya et al, 2020

PCA-net

Finite-dimensional PCA:

Given Data $\{v_k\}_{k=1}^{K}$ $\quad v_k \in \mathbb{R}^d$ $\quad K \to$ large

"Project" Data onto a more convenient basis: $\{\omega_j\}_{j=1}^{J}$ $\quad J \ll K$

$v_k \to$ zero-mean data

$$v_k = \sum_{j=1}^{J} \beta_j^k \omega_j \qquad \beta_j^k \to \text{coefficients}$$

$$\min_{\beta_j^k,\, \omega_j} \sum_{k=1}^{K} \left| v_k - \sum_{j=1}^{J} \beta_j^k \omega_j \right|^2 \iff \text{SVD on } \sum_{k=1}^{K} v_k \otimes v_k$$

the Co-variance:

λ_l, ω_l are $1 \le l \le J$ leading singular vals, vecs.

$$V_k \approx \sum_{\lambda=1}^{J} \lambda_\lambda \omega_\lambda$$ — Singular values, vectors

Function-Space version:

Singular-values, Singular functions of the Co-variance operator

$$\mathbb{E}_{U \sim M}[U \otimes U]$$

PCA-Net: Inputs — $\{a_i\}_{i=1}^{M}$

Do a PCA → coefficients:

Output — output PCA

Numerical Results for Forced Pendulum

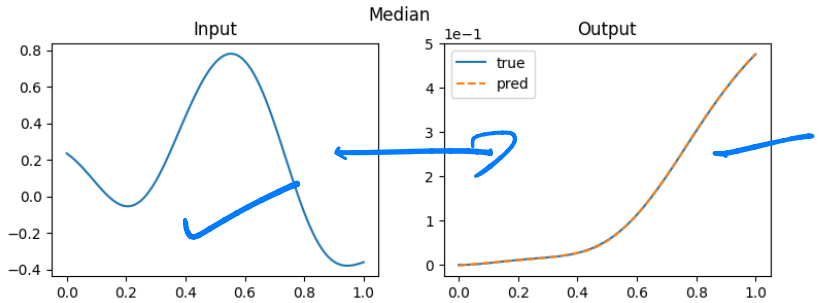
$-u'' = \sin(u) + a(t)$ — forcing

displacement

Numerical ODE Solver.

$n = 1000$ samples

- Measure μ is Law of a Gaussian Random Field (GRF). / Gaussian process.



DeepOnet

- DeepOnet: 0.35%
- PCA-NN: 0.18%

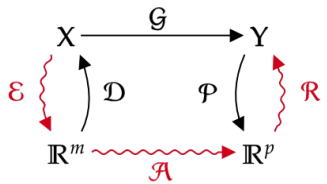
Covariance kernel

$a \sim GP(0, k_l)$

$k_l(t_1, t_2) \sim e^{-\frac{|t_1 - t_2|^2}{2l^2}}$

Correlation length

Why do ONets work ?: Lanthaler, SM, Karniadakis, 2022



- Universal Approximation Thm: For $\mu \in Prob(L^2(D))$ and any measurable $\mathcal{G} : H^r \mapsto H^s$ and $\epsilon > 0$, $\exists \mathcal{N}$ (Onet): $\hat{\mathcal{E}} < \epsilon$
- Upper (and lower) bounds via $\mathcal{E}_{\mathcal{R}} \leq \mathcal{E} \leq C(\mathcal{E}_{\mathcal{E}} + \mathcal{E}_{\mathcal{A}} + \mathcal{E}_{\mathcal{R}})$.
- $\mathcal{E}_{\mathcal{E},\mathcal{R}}$ decay as spectrum of Covariance operator of μ and $\mathcal{G}\#_{\mu}$.
- For $G = \mathcal{P} \circ \mathcal{G} \circ \mathcal{D} \in C^k(\mathbb{R}^m, \mathbb{R}^p) \Rightarrow \text{size } (\mathcal{N}) \sim \mathcal{O}\left(\epsilon^{-\frac{m(\epsilon)}{k}}\right)$.
- Curse of Dimensionality (CoD) for Onets !!!

On CoD for Operator Networks

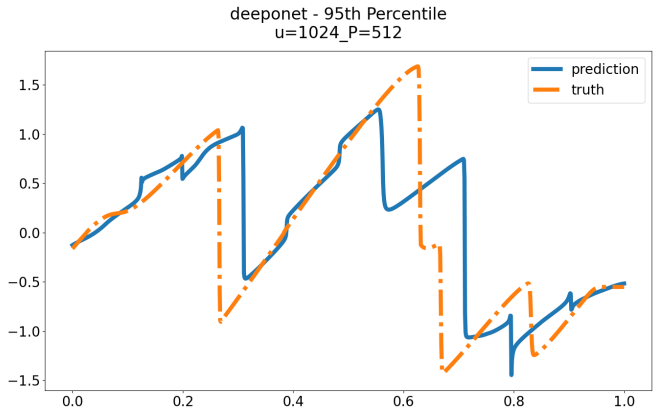
- DeepOnet breaks the Curse of Dimensionality !!
- For operators $\mathcal{G}$ corresponding to many PDEs:

PDE	Operator	Complexity
$-u'' = \sin(u) + f$	$\mathcal{G} : f \to u(T)$	$M \sim \mathcal{O}(\epsilon^{-\eta})\ \eta \approx 0$
$-\mathrm{div}(a\nabla u) = f$	$\mathcal{G} : a \to u$	$M \sim \mathcal{O}(\epsilon^{-\eta})\ \eta \approx 0$
$u_t = \Delta u + f(u)$	$\mathcal{G} : u_0 \to u(T)$	$M \sim \mathcal{O}(\epsilon^{-2(d+1)})$
$u_t + u \cdot \nabla u + \nabla p = \nu \Delta u$	$\mathcal{G} : u_0 \to u(T)$	$M \sim \mathcal{O}(\epsilon^{-(d+1)})$
$u_t + \mathrm{div}(f(u)) = 0$	$\mathcal{G} : u_0 \to u(T)$	$M \sim \mathcal{O}(\epsilon^{-\alpha(d+1)})$

- Case by Case basis: Operator Holomorphy, Emulation of Numerical Schemes etc...
- Unified framework in DeRyck, SM, 2022.

DeepONets: Issues

- Affine Reconstructors $\Rightarrow \sqrt{\sum_{\ell > p} \lambda_\ell} \leq \mathcal{E}_{\mathcal{R}} \leq \mathcal{E}$
- Large error for Slow decay of eigenvalues of Covariance operator of $\mathcal{G}\#_\mu$.
- Holds for Transport dominated problems
- Error of 30% for Burgers' equation with GRF initial data !!



Alternative: Neural Operators

- Formalized in Kovachki et al, 2021.
- Recall: DNNs are $\mathcal{L}_\theta = \sigma_K \odot \sigma_{K-1} \odot \ldots\ldots \sigma_1$
- Single hidden layer: $\sigma_k(y) = \sigma(A_k y + B_k)$
- Neural Operators generalize DNNs to ∞-dimensions:
- NO: $\mathcal{N}_\theta = \mathcal{N}_L \odot \mathcal{N}_{L-1} \odot \ldots\ldots \mathcal{N}_1$
- Single hidden layer;

$$(\mathcal{N}_\ell v)(x) = \sigma\left(A_\ell v(x) + B_\ell(x) + \int_D K_\ell(x,y)v(y)dy\right)$$

- Kernel Integral Operators
- Learning Parameters in A_ℓ, B_ℓ, K_ℓ
- Different Kernels $\Rightarrow$ Low-Rank NOs, Graph NOs, Multipole NOs, ...