Nonlinear Model Reduction for Operator Learning

Hamidreza Eivazi*, Stefan Wittek & Andreas Rausch
ISSE, Clausthal University of Technology, 38678 Clausthal-Zellerfeld, DE

The Twelfth International Conference on Learning Representations
May 7-11 2024, Vienna, Austria

*he76@tu-clausthal.de

Operator learning

Let us consider $\mathcal{U}$ and $\mathcal{V}$ as two separable Banach spaces and assume that

$$\mathcal{G} : \mathcal{U} \longrightarrow \mathcal{V}$$

is an arbitrary operator.

Operator learning

Let us consider $\mathcal{U}$ and $\mathcal{V}$ as two separable Banach spaces and assume that

$$\mathcal{G} : \mathcal{U} \longrightarrow \mathcal{V}$$

is an arbitrary operator. We only have access to partially observed input and output data $\{u_i, v_i\}_{i=1}^{N}$ as N elements of $\mathcal{U} \times \mathcal{V}$ such that

$$\mathcal{G}(u_i) = v_i, \qquad i = 1, \cdots, N.$$

Operator learning

$$
\begin{tikzcd}
\mathcal{U} \arrow[r, "\mathcal{G}"] \arrow[d, "\mathcal{P}"'] & \mathcal{V} \arrow[d, "\mathcal{Q}"] \\
\mathbb{R}^n & \mathbb{R}^m
\end{tikzcd}
$$

Operator learning

$$
\begin{array}{ccc}
\mathcal{U} & \xrightarrow{\color{blue}{\mathcal{G}}} & \mathcal{V} \\
\mathcal{P}\big\downarrow & & \big\downarrow\color{red}{\mathcal{Q}} \\
\mathbb{R}^n & & \color{red}{\mathbb{R}^m}
\end{array}
$$

We consider $\mathcal{P}$ and $\mathcal{Q}$ as two linear and bounded evaluation operators

$$
\mathcal{P} : u \mapsto (u(\boldsymbol{x}_1), u(\boldsymbol{x}_2), \cdots, u(\boldsymbol{x}_n))^T,
$$
$$
\mathcal{Q} : v \mapsto (v(\boldsymbol{y}_1), v(\boldsymbol{y}_2), \cdots, v(\boldsymbol{y}_m))^T.
$$

Operator learning

$$
\begin{tikzcd}
\mathcal{U} \arrow[r, "\mathcal{G}"] \arrow[d, "\mathcal{P}"'] & \mathcal{V} \arrow[d, "\mathcal{Q}"] \\
\mathbb{R}^n & \mathbb{R}^m
\end{tikzcd}
$$

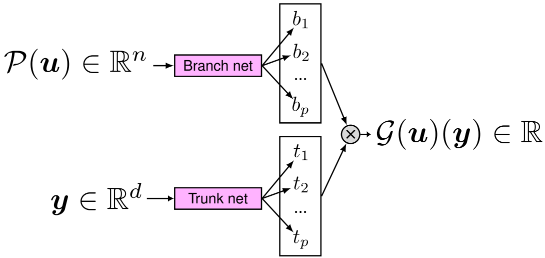
We consider $\mathcal{P}$ and $\mathcal{Q}$ as two linear and bounded evaluation operators

$$\mathcal{P} : u \mapsto \left(u(\boldsymbol{x}_1),\, u(\boldsymbol{x}_2),\, \cdots,\, u(\boldsymbol{x}_n)\right)^T,$$

$$\mathcal{Q} : v \mapsto \left(v(\boldsymbol{y}_1),\, v(\boldsymbol{y}_2),\, \cdots,\, v(\boldsymbol{y}_m)\right)^T.$$

Considering $U_i = \mathcal{P}(u_i)$ and $V_i = \mathcal{Q}(v_i)$, our goal is to learn $\mathcal{G}$ from dataset $\{U_i, V_i\}_{i=1}^N$.

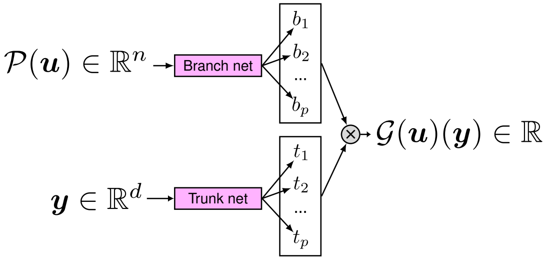
Deep operator networks (DeepONets)



Deep operator networks (DeepONets)

Operator $\mathcal{G}$ can be approximated as

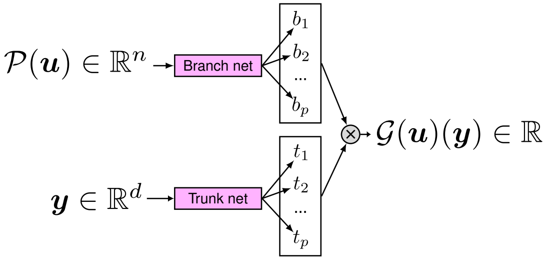
$$\mathcal{G}(u)(\boldsymbol{y}) \approx \sum_{k=1}^{p} b_k(U) t_k(\boldsymbol{y}) + b_0$$



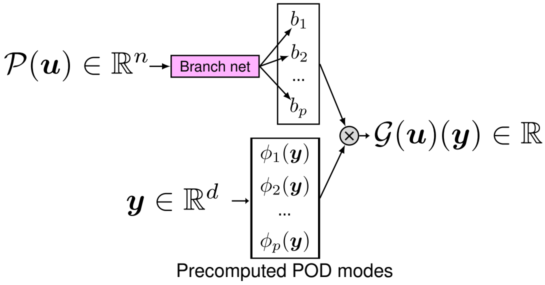
Deep operator networks (DeepONets)

Operator $\mathcal{G}$ can be approximated as

$$\mathcal{G}(u)(\boldsymbol{y}) \approx \sum_{k=1}^{p} \underbrace{b_k(U)t_k(\boldsymbol{y})}_{\text{linear reconstruction}} + b_0$$



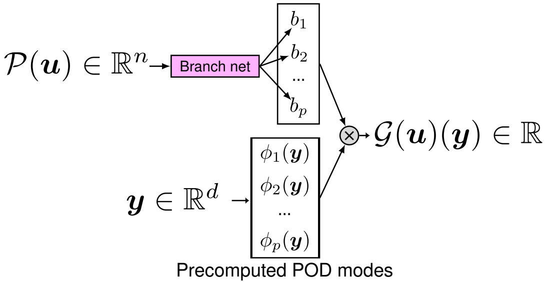
Proper orthogonal decomposition (POD)-DeepONets



Proper orthogonal decomposition (POD)-DeepONets

Trunk net is replaced by a set of POD basis functions

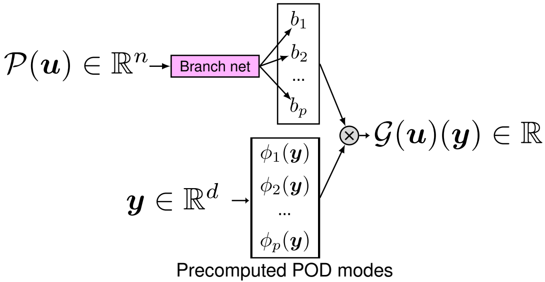
$$\mathcal{G}(u)(\boldsymbol{y}) \approx \sum_{k=1}^{p} b_k(U)\phi_k(\boldsymbol{y}) + \phi_0(\boldsymbol{y})$$



Proper orthogonal decomposition (POD)-DeepONets

Trunk net is replaced by a set of POD basis functions

$$\mathcal{G}(u)(\boldsymbol{y}) \approx \sum_{k=1}^{p} \underbrace{b_k(U)\phi_k(\boldsymbol{y})}_{\text{linear reconstruction}} + \phi_0(\boldsymbol{y})$$



Kernel principal component analysis (KPCA)-DeepONets

$$\begin{tikzcd}
\mathcal{U} \arrow[rr, "\mathcal{G}"] \arrow[d, "\mathcal{P}"'] & & \mathcal{V} \arrow[d, "\mathcal{Q}"] \\
\mathbb{R}^n \arrow[r, "b"'] & \mathbb{R}^p \arrow[ur, "h\ \textbf{(KRR)}"] & \mathbb{R}^m \arrow[l, "f\ \textbf{(KPCA)}"]
\end{tikzcd}$$

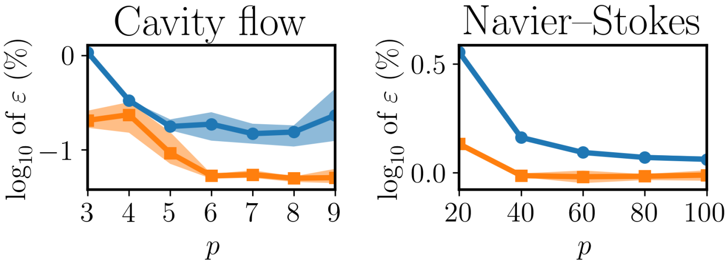
Kernel principal component analysis (KPCA)-DeepONets

$$
\begin{tikzcd}
\mathcal{U} \arrow[rr, "\mathcal{G}"] \arrow[d, "\mathcal{P}"'] & & \mathcal{V} \arrow[d, "\mathcal{Q}"] \\
\mathbb{R}^n \arrow[r, "b"'] & \mathbb{R}^p \arrow[ur, "h\ \textbf{(KRR)}"] & \mathbb{R}^m \arrow[l, "f\ \textbf{(KPCA)}"]
\end{tikzcd}
$$

Operator $\mathcal{G}$ can be approximated as

$$
\mathcal{G}(u)(\boldsymbol{y}) \approx \sum_{i=1}^{N} \alpha_i(\boldsymbol{y})\, k_z(\boldsymbol{b}(U), \boldsymbol{z}_i^t) + \phi_0(\boldsymbol{y}).
$$

Numerical experiments



Comparison of KPCA-DeepONet (orange, ■) and POD-DeepONet (blue, ●).

Numerical experiments

The ℓ_2 relative errors. Results for models marked with * are taken from Lu et al. (2022)[1].

Models	Cavity flow	Navier–Stokes
KPCA-DeepONet	$\mathbf{0.05 \pm 0.00\%}$	$\mathbf{0.96 \pm 0.05\%}$
POD-DeepONet*	$0.33 \pm 0.08\%$	$1.36 \pm 0.03\%$
DeepONet*	$1.20 \pm 0.23\%$	$1.78 \pm 0.02\%$
FNO*	$0.63 \pm 0.04\%$	$1.81 \pm 0.02\%$

[1] Lu Lu et al. A comprehensive and fair comparison of two neural operators (with practical extensions) based on FAIR data. Comput. Methods Appl. Mech. Eng., 2022.

Summary and conclusions

- KPCA-DeepONet benefits from kernel methods and non-linear model reduction.
- KPCA-DeepONet provides a non-linear reconstruction.
- Our method results in less than 1% error for the Navier–Stokes test case.

Thank you!

Any questions?

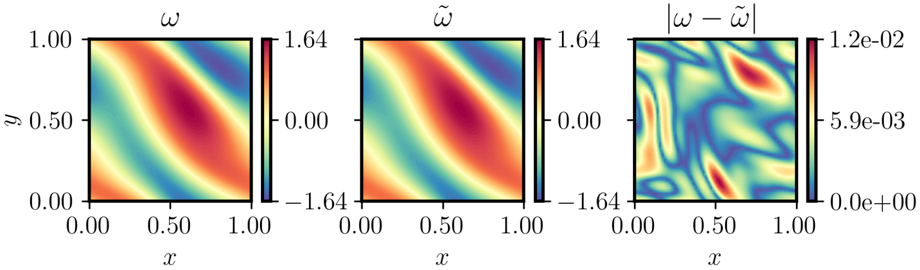
Installation

```
pip install kpca-deeponet
```

GitHub

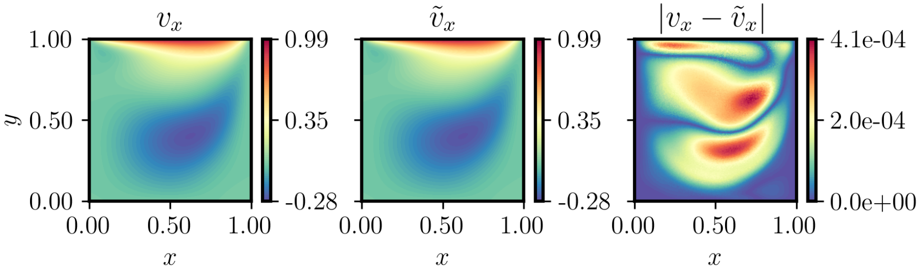
Paper

Numerical experiments



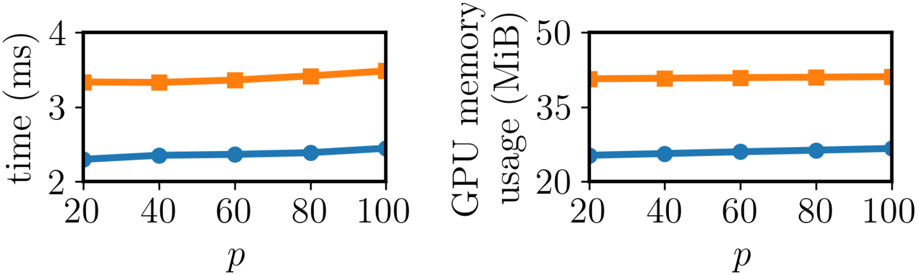
KPCA-DeepONet prediction against the reference data for one sample of the test dataset for the Navier–Stokes equation. $\tilde{\cdot}$ indicates the KPCA-DeepONet prediction.

Numerical experiments



KPCA-DeepONet prediction against the reference data for one sample of the test dataset for the cavity flow. $\tilde{\cdot}$ indicates the KPCA-DeepONet prediction.

Computational cost



Comp. cost and GPU mem. usage for the proposed KPCA-DeepONet (orange, ■) and POD-DeepONet (blue, ●).