DeepONet

Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators, Lu Lu et al.

Presented by Gaspard Krief and Emilia Pucher

Overview

- Motivating Problems and Operators
- Previous Approaches and Related Literature
- Theoretical Background and DeepONet Architecture
- DeepONet Performance, Accuracy and Learning Speed
- DeepONet and Function Dimension
- DeepONet and Sensor Constraints
    - Demonstration of our Results
- Critique and Further Research

Introduction: Operators and Motivating Problems

What are we trying to solve?

- Physical Problems
  - May involve complex equations e.g. ODEs, Derivatives, Integrals
  - Numerical solutions are inefficient

- Idea: **use Neural Networks to solve Physical Problems**

- Problem:
  - Not limited to estimating functions
  - Different types of structures and inputs

- Goal:
  - Find a **Neural Network Architecture** suitable **for learning operators**

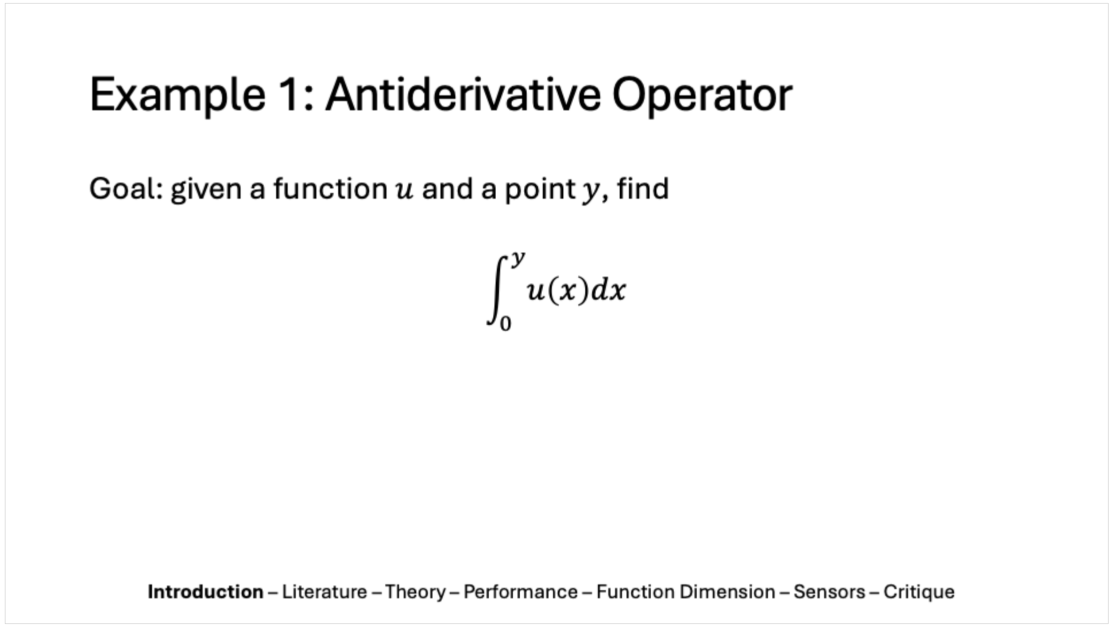
Goal of DeepONet: Learn operators!

An operator loosely refers to a **mapping from a space of functions into another space of functions**

Operator: function $\longmapsto$ function

- Examples include:
  - derivative: $x(t) \mapsto x'(t)$
  - integral: $x(t) \mapsto \int K(s,t)x(s)ds$

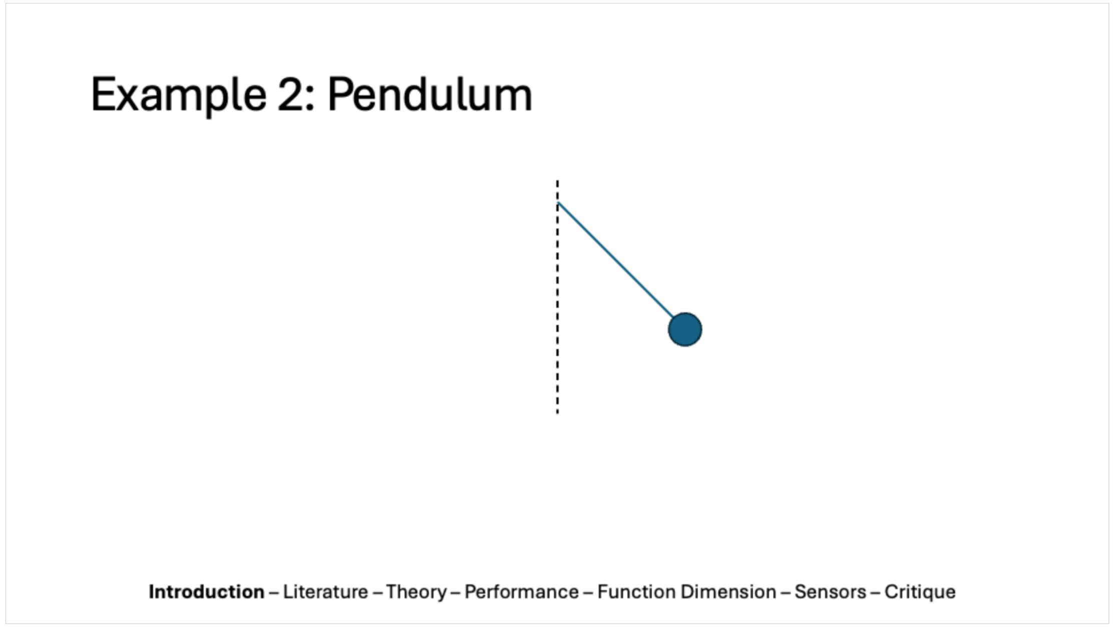
Two motivating Problems



Example 1: Antiderivative Operator

Goal: given a function u and a point y, find

$$\int_0^y u(x)dx$$



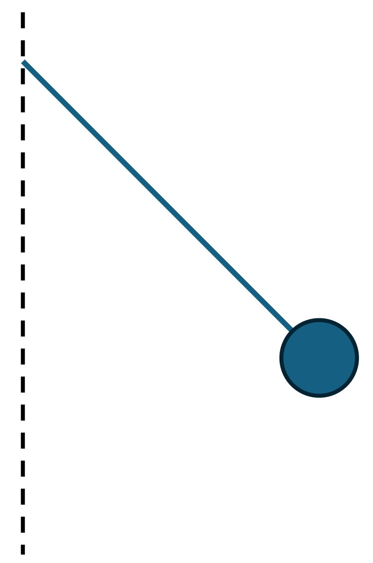
Example 2: Pendulum

Example 1: Antiderivative Operator

Goal: given a function u and a point y, find

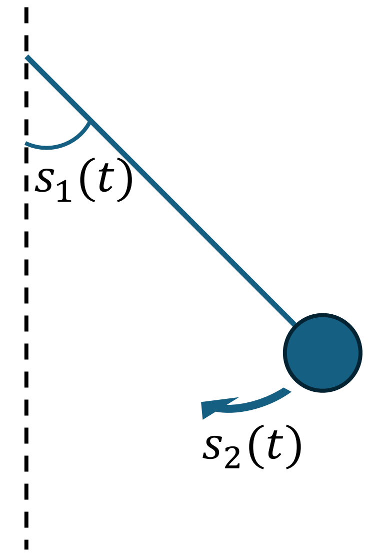
$$\int_0^y u(x)dx$$

Example 2: Pendulum



Example 2: Pendulum

- Angle at time t: $s_1(t)$
- Angular velocity at time t: $s_2(t) = s_1'(t)$
- External force at time t: $u(t)$



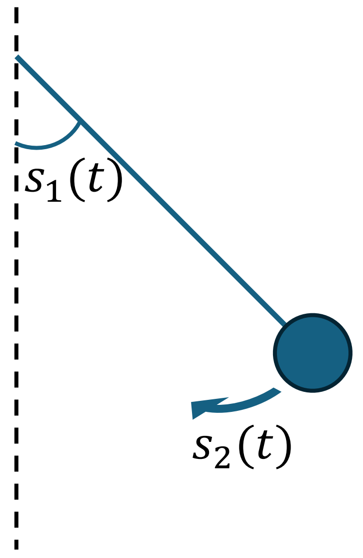
Example 2: Pendulum

- Angle at time t: $s_1(t)$
- Angular velocity at time t: $s_2(t) = s_1'(t)$
- External force at time t: $u(t)$

- Giving the ODE:

$$s_2'(t) = -k \sin s_1(t) + u(t)$$

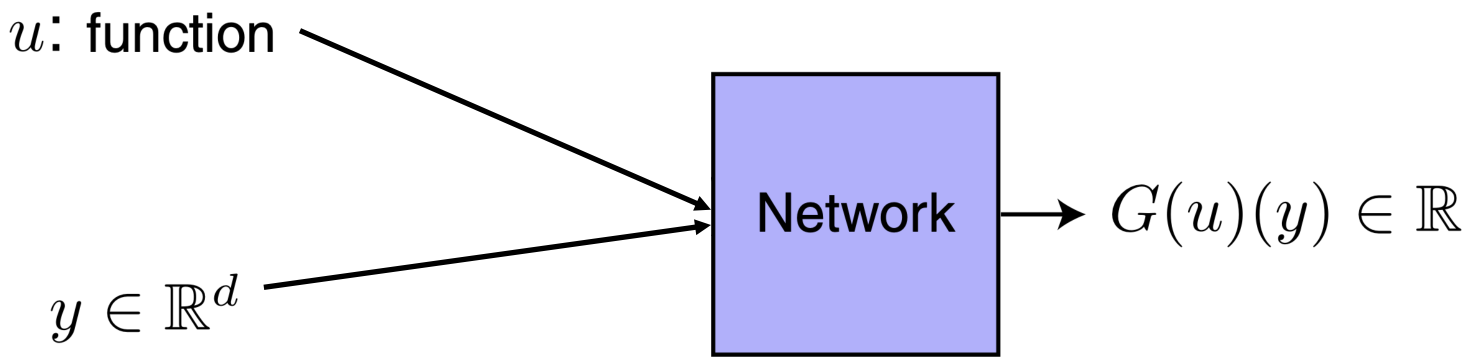
Goal: given $\boldsymbol{u}$ and $\boldsymbol{t}$, find $s_1(t)$ and $s_2(t)$



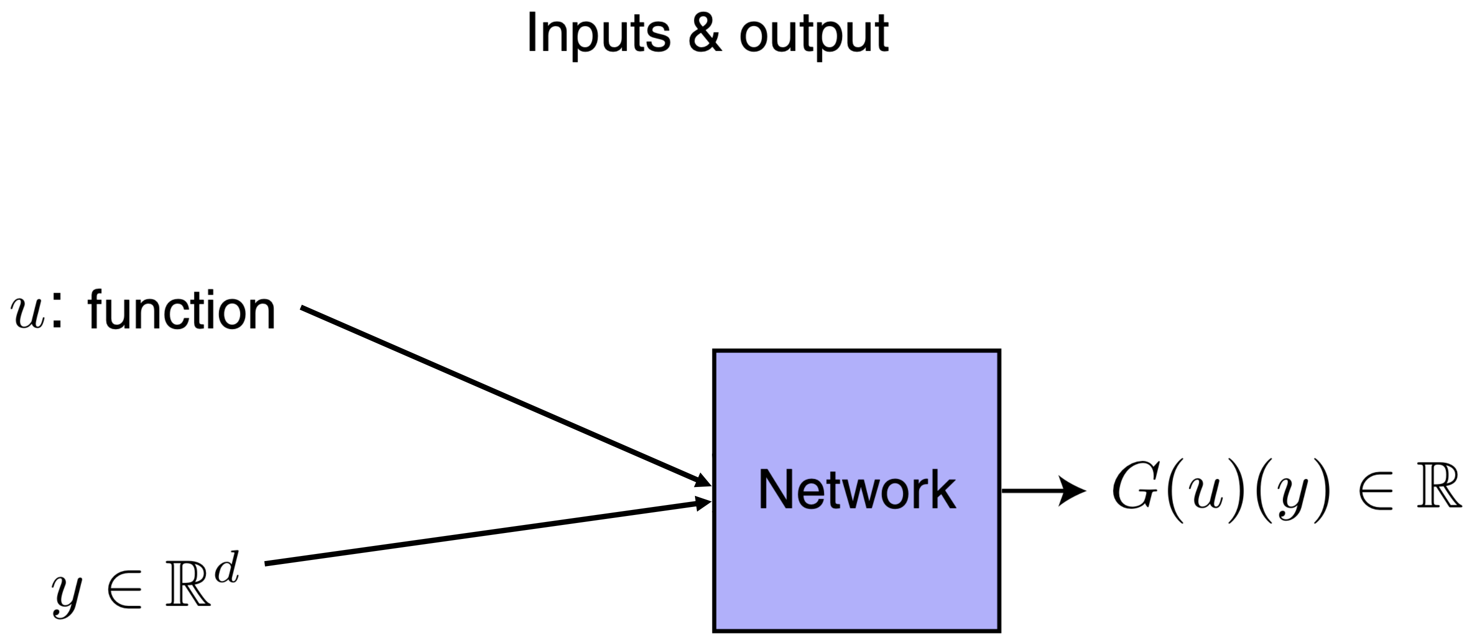
Previous Approaches and Related Literature

Naïve data-oriented approach

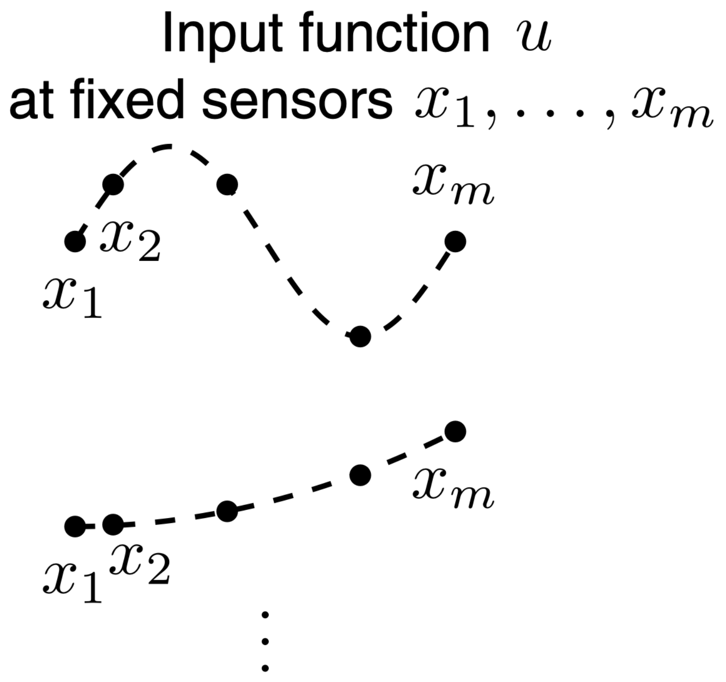
Inputs & output

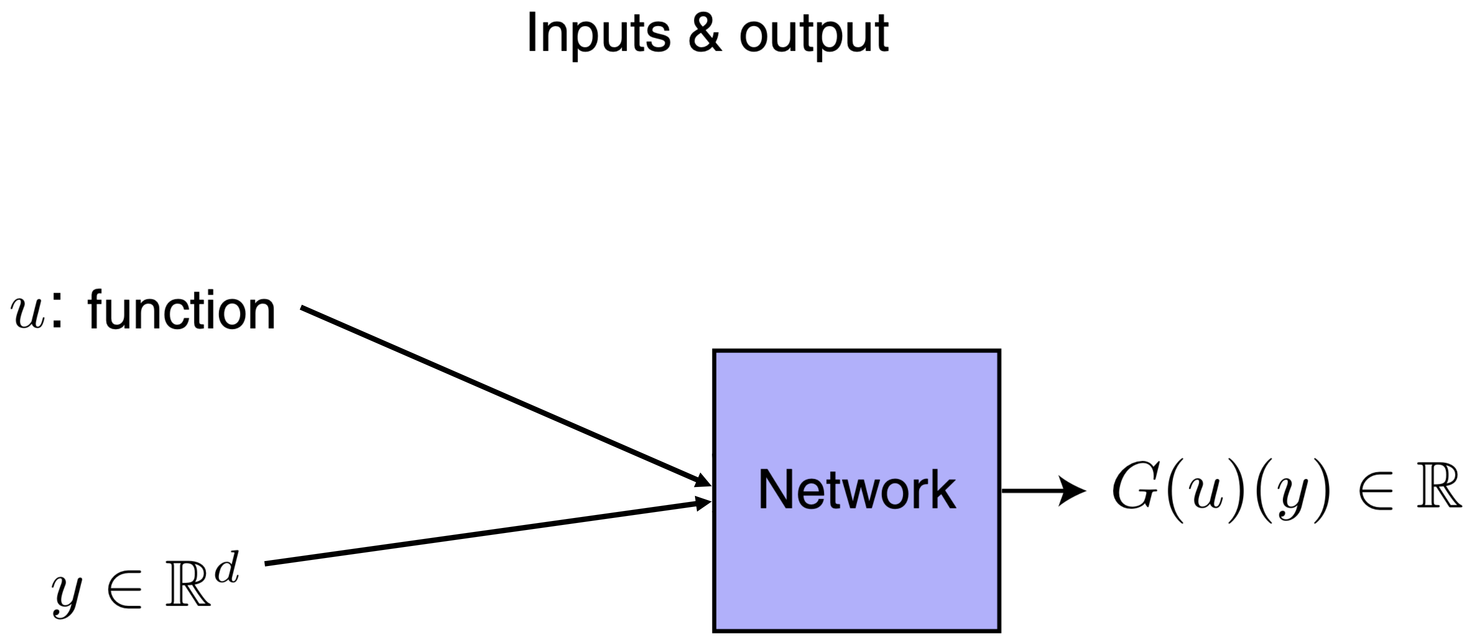


Naïve data-oriented approach

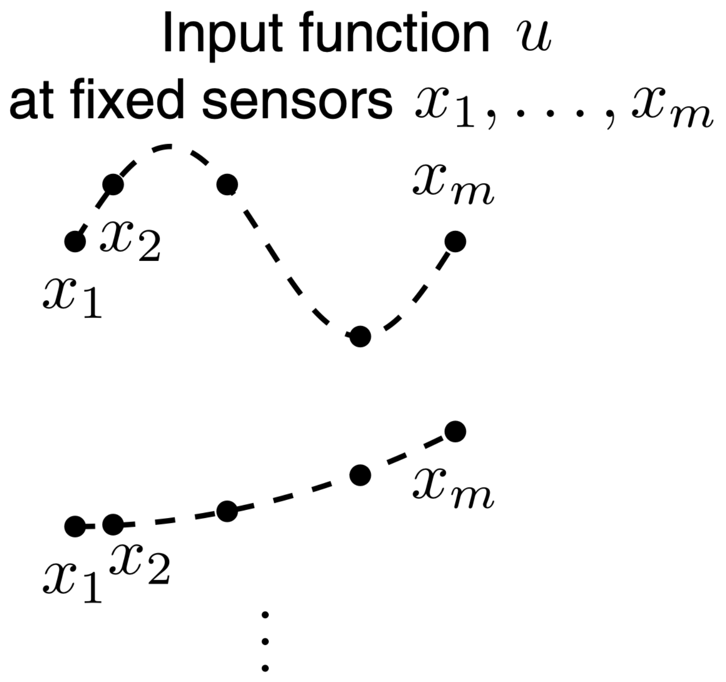


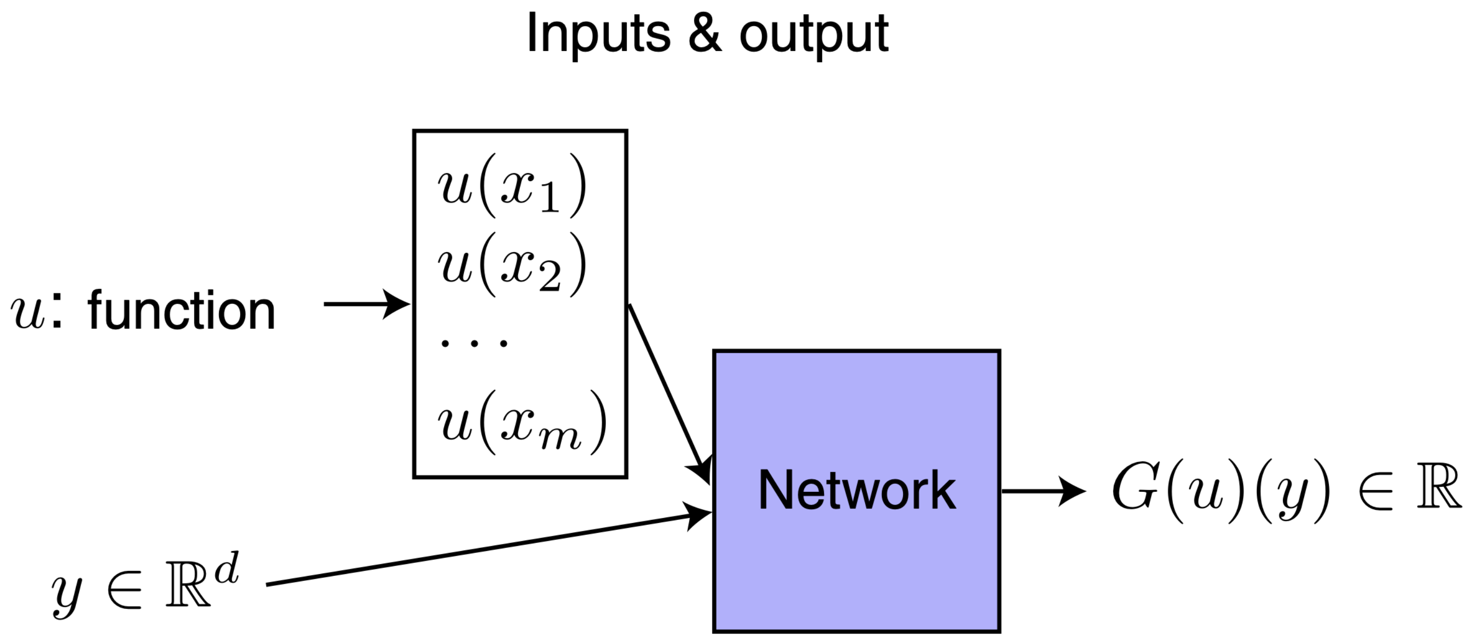
Naïve data-oriented approach





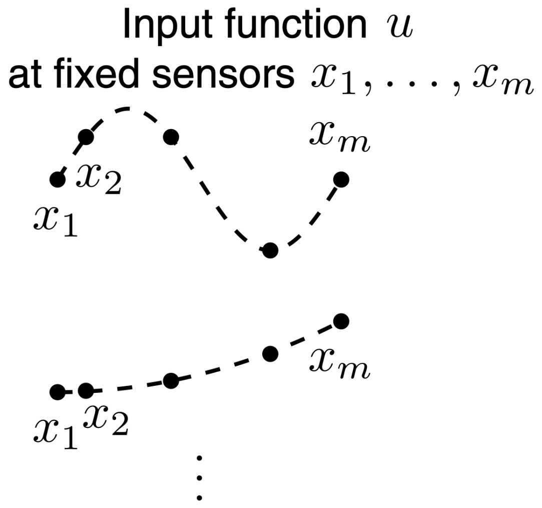
Naïve data-oriented approach





Data and Loss Function

- Input: $[u(x_1), u(x_2), \dots u(x_m), y]$
  - With m sensors
  - And point y



and $y \in \mathbb{R}^d$

- One data point consists of the triplet $(u, y, G(u)(y))$

Data and Loss Function

- Constraints:
  - Sensor locations are the same all input functions u
  - Not necessarily on a lattice
  - An input u may appear in multiple data points with different values of y

- No additional information about Operator or System
  - In contrast to regular PINNs

- Loss Function: MSE
  - Minimize $(G(u)(y) - Network\ Output)^2$

Approaches for solving Nonlinear Operators (Related Literature)

1. Traditional Numerical Methods

- Finite Difference Methods (FDM) and Finite Element Methods (FEM)
  - Widely used
  - Computationally intensive and may struggle with high dimensional problems
- Spectral Methods
  - High accuracy
  - Require complex implementations for nonlinear operators

Approaches for solving Nonlinear Operators (Related Literature)

2. Classical and Recurrent Neural Networks:

- Neural Networks use the universal approximation theorem
  - Does not say anything about operators

- MLPs (Multilayer Perceptrons)
- LSTM-RNN (Long Short-Term Memory Recurrent Neural Networks)

Approaches for solving Nonlinear Operators (Related Literature)

- **(R)MLPs Recurrent Multilayer Perceptrons (Yu, W & Li, X 2005)**
  - Can approximate ODEs
  - Require extensive data for retraining new inputs

- **Continuous-time recurrent multilayer perceptrons for nonlinear system identification**
  Yu, W., & Li, X. (2005). Continuous-time recurrent multilayer perceptrons for nonlinear system identification. Proceedings of 2005 IEEE Conference on Control Applications, 2005. CCA 2005., 1636-1641. https://consensus.app/papers/continuoustime-recurrent-multilayer-perceptrons-yu/fa01c3662b095dbeaddff4c869137db9/?utm_source=chatgpt

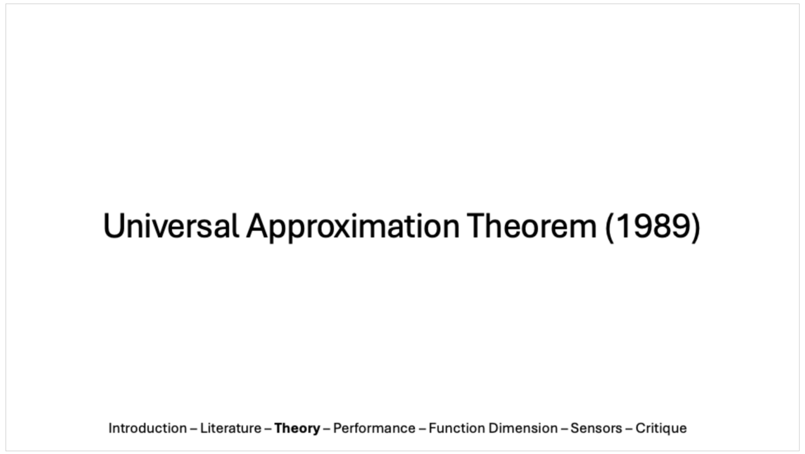
Approaches for solving Nonlinear Operators (Related Literature)

- **Long Short-Term Memory Recurrent Neural Networks (Bassi et al. 2023)**
  - Used to learn nonlinear integro-differential equations (IDEs)
  - Can be used for learning operators beyond the integro-differential equations
  - Limited since they often require large datasets for training

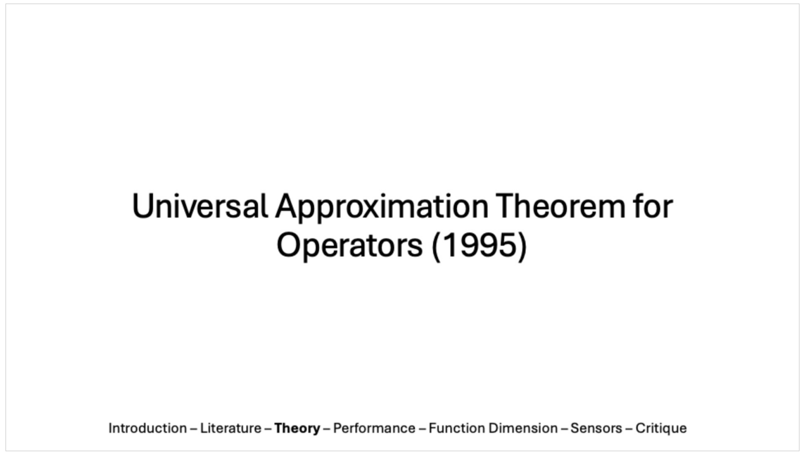
- **Learning nonlinear integral operators via recurrent neural networks and its application in solving integro-differential equations**
  Bassi, H., Zhu, Y., Liang, S., Yin, J., Reeves, C. C., Vlček, V., & Yang, C. (2024). Learning nonlinear integral operators via recurrent neural networks and its application in solving integro-differential equations. Machine Learning With Applications, 15, 100524. https://doi.org/10.1016/j.mlwa.2023.100524

Theoretical Background and DeepONet Architecture

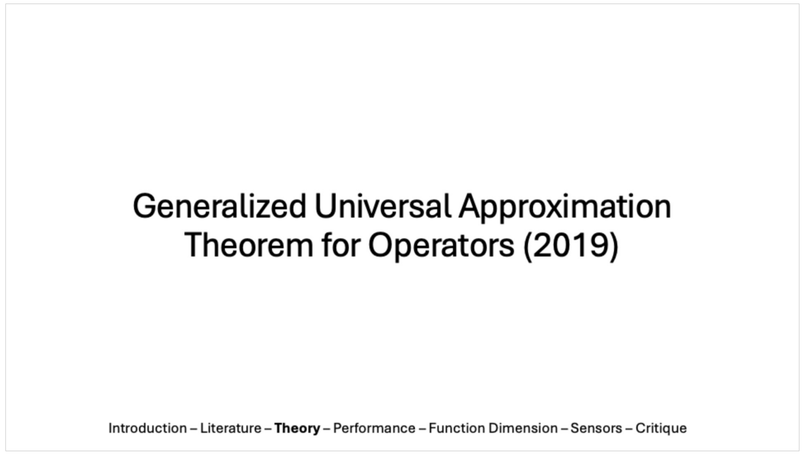
Theory behind DeepONet


Universal Approximation Theorem (1989)

Introduction – Literature – **Theory** – Performance – Function Dimension – Sensors – Critique


Universal Approximation Theorem for Operators (1995)

Introduction – Literature – **Theory** – Performance – Function Dimension – Sensors – Critique


Generalized Universal Approximation Theorem for Operators (2019)

Introduction – Literature – **Theory** – Performance – Function Dimension – Sensors – Critique

Universal Approximation Theorem (1989)

Universal Approximation Theorem (1989)

Let f be any continuous function on $[0,1]^d$, φ any sigmoidal activation function and $\epsilon > 0$. There exists an $\hat{f}$ of the form

$$\hat{f}(x) = \sum_{j=1}^{p} w_j^{(2)} \varphi(w_j^{(1)T} x + w_{j,0}^{(1)})$$

such that for all $x \in [0,1]^d$, we have $\left|f(x) - \hat{f}(x)\right| \leq \epsilon$

Universal Approximation Theorem for Operators (1995)

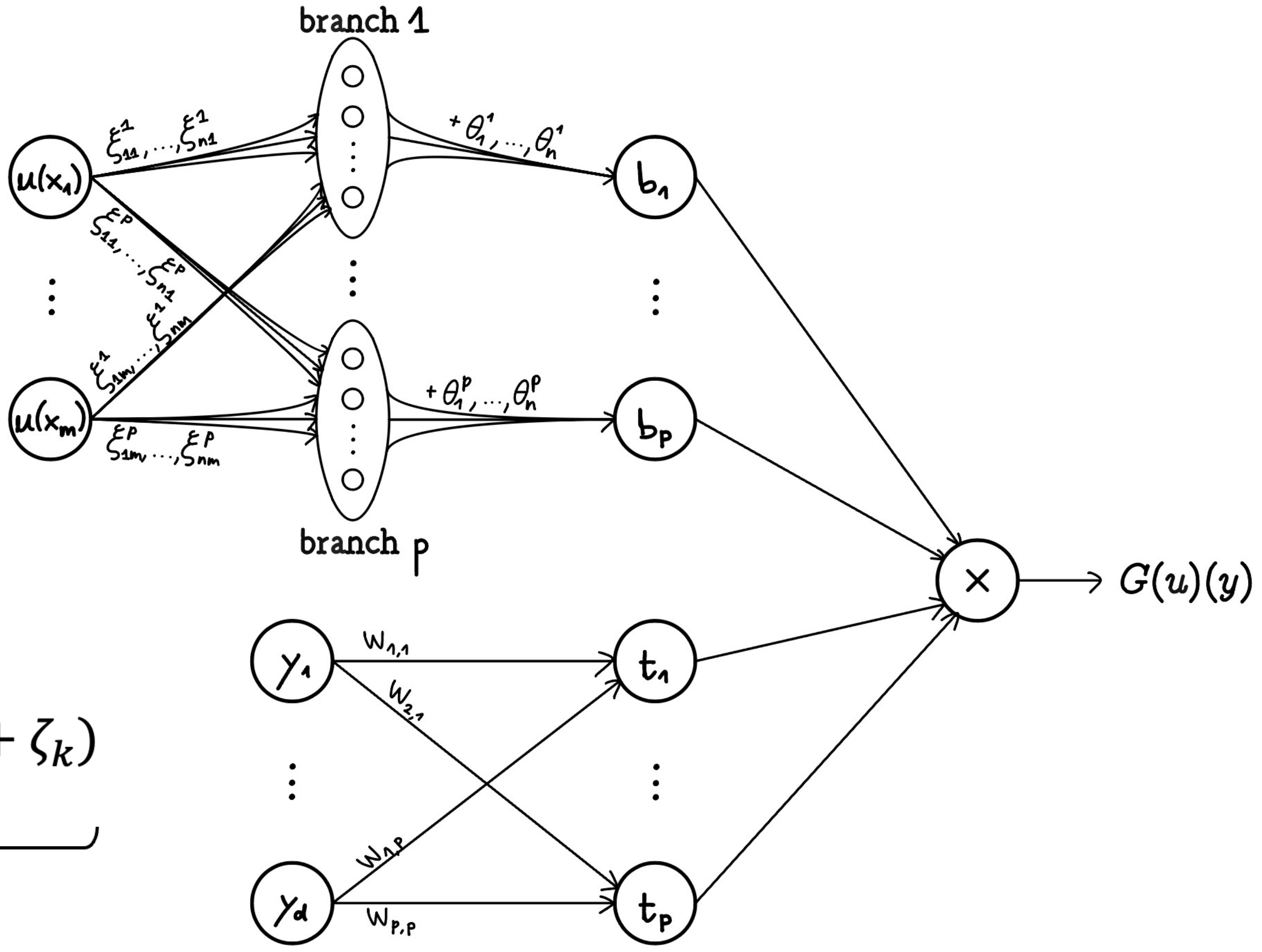
Universal Approximation Theorem for Operators (1995)

Let G be a nonlinear continuous operator, σ a continuous non-polynomial function, X a Banach space, $K_1 \subset X$ and $K_2 \subset \mathbb{R}^d$. For any $\epsilon > 0$, there exist $n, p, m \in \mathbb{N}$, $c_i^k, \xi_{ij}^k, \theta_i^k, \zeta_k \in \mathbb{R}$, $w_k \in \mathbb{R}^d$, $x_j \in K_1$, such that

$$\left| G(u)(y) - \sum_{k=1}^{p} \underbrace{\sum_{i=1}^{n} c_i^k \, \sigma\Big(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k\Big)}_{\text{branch}} \underbrace{\sigma(w_k y + \zeta_k)}_{\text{trunk}} \right| < \epsilon$$

holds for all $u \in V \subset C(K_1)$ and $y \in K_2$

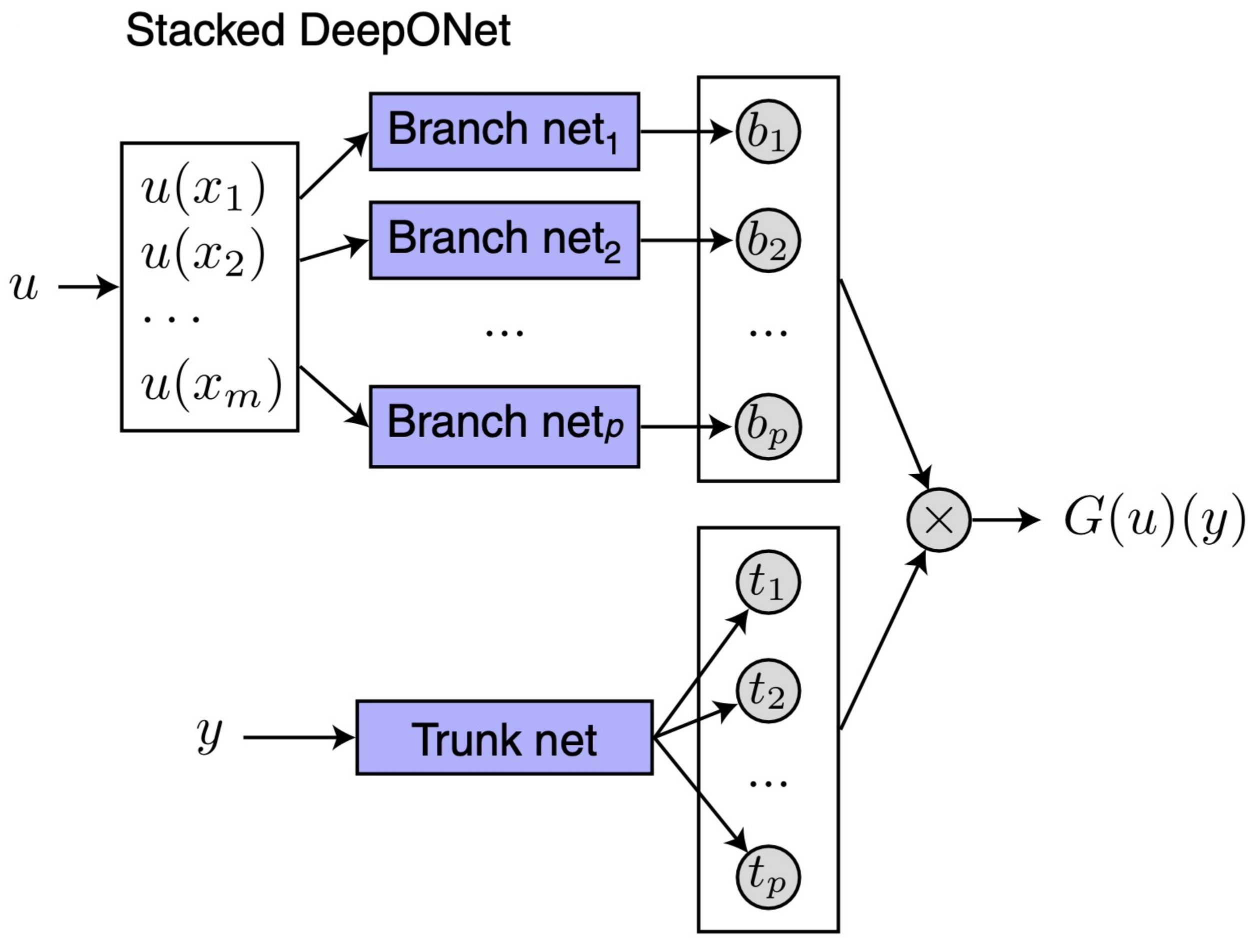
Universal Approximation Theorem for Operators (1995)



branch 1
$\xi_{11}^1, \dots, \xi_{n1}^1$
$+\theta_1^1, \dots, \theta_n^1$
$u(x_1)$
b_1
$\xi_{11}^p, \dots, \xi_{n1}^p$
$\xi_{1m}^1, \dots, \xi_{nm}^1$
$u(x_m)$
$+\theta_1^p, \dots, \theta_n^p$
b_p
$\xi_{1m}^p, \dots, \xi_{nm}^p$
branch p
$\times$
$G(u)(y)$
y_1
$w_{1,1}$
$w_{2,1}$
t_1
$w_{1,p}$
y_d
$w_{p,p}$
t_p

$$\sum_{k=1}^{p} \underbrace{\sum_{i=1}^{n} c_i^k \, \sigma\Big(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k\Big)}_{\text{branch}} \underbrace{\sigma(w_k y + \zeta_k)}_{\text{trunk}}$$

Universal Approximation Theorem for Operators (1995)



Generalized Universal Approximation Theorem for Operators (2019)

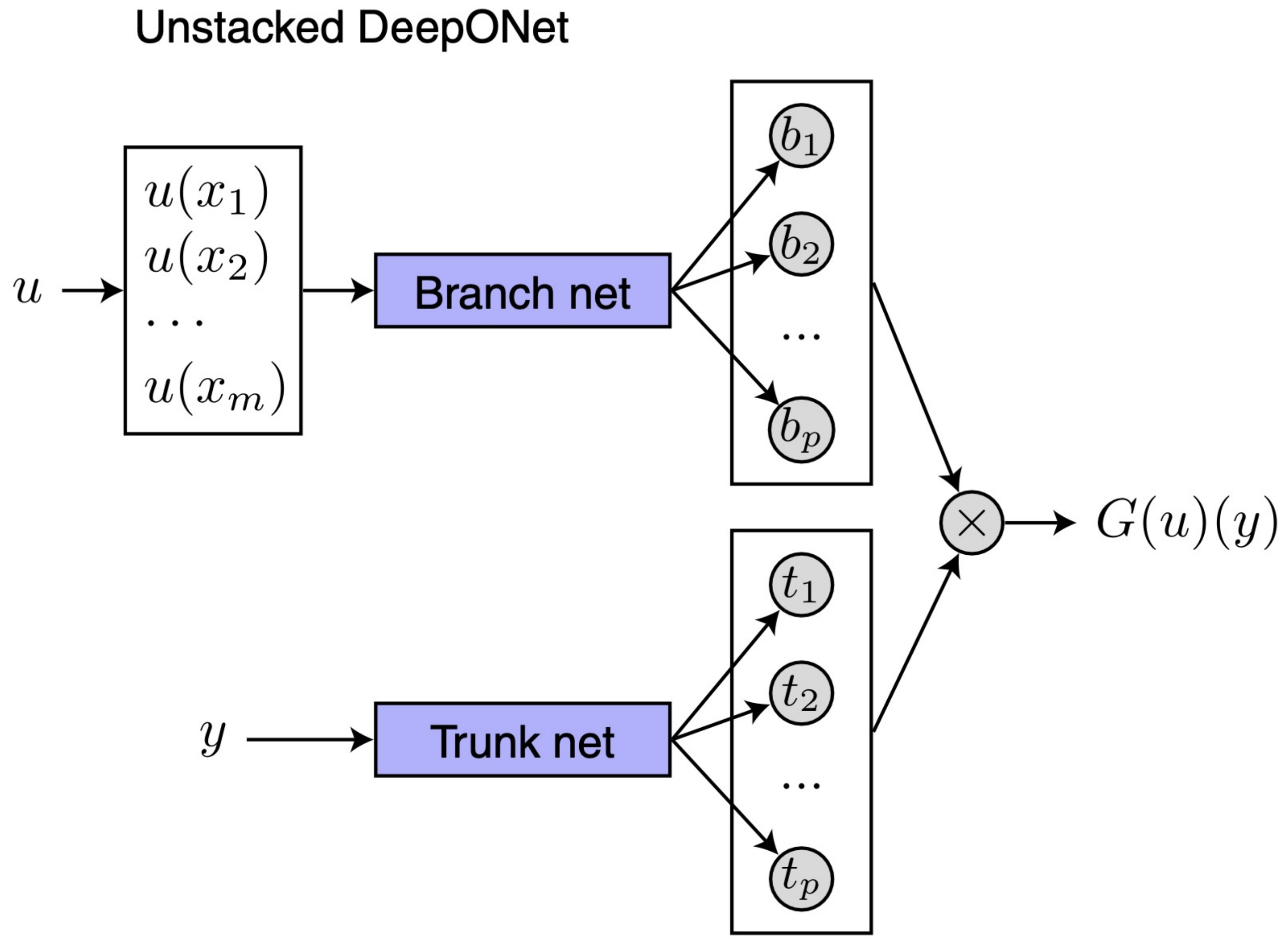
Generalized Universal Approximation Theorem for Operators (2019)

Let X be a Banach space, $K_1 \subset X$ and $K_2 \subset \mathbb{R}^d$, $V \subset C(K_1)$. Let $G: V \to C(K_2)$ be a nonlinear continuous operator. For any $\epsilon > 0$, there exist $n, p, m \in \mathbb{N}$, $\mathbf{g}: \mathbb{R}^m \to \mathbb{R}^p$, $\mathbf{f}: \mathbb{R}^d \to \mathbb{R}^p$, $x_j \in K_1$, such that

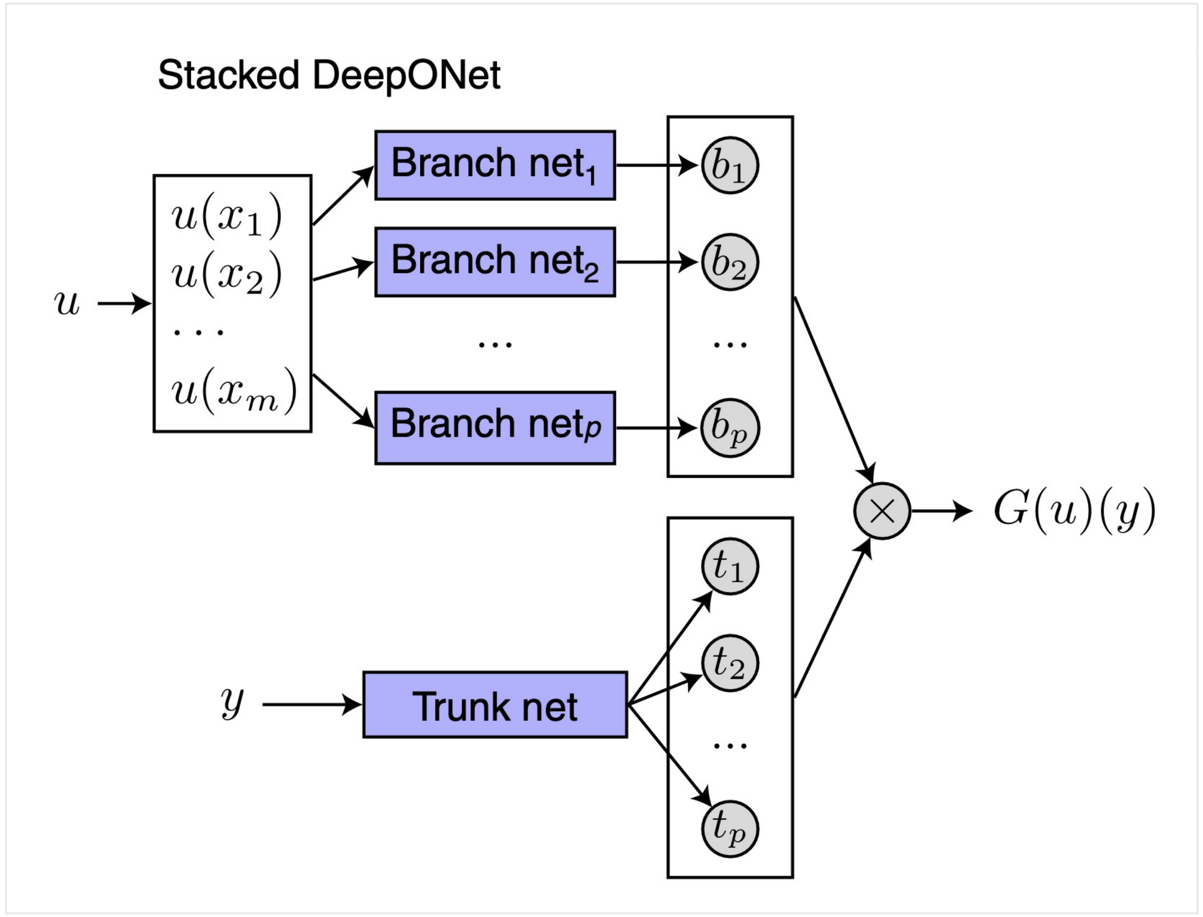
$$\left| G(u)(y) - \left\langle \underbrace{\mathbf{g}\big(u(x_1), \dots, u(x_m)\big)}_{\text{branch}}, \underbrace{\mathbf{f}(y)}_{\text{trunk}} \right\rangle \right| < \epsilon$$

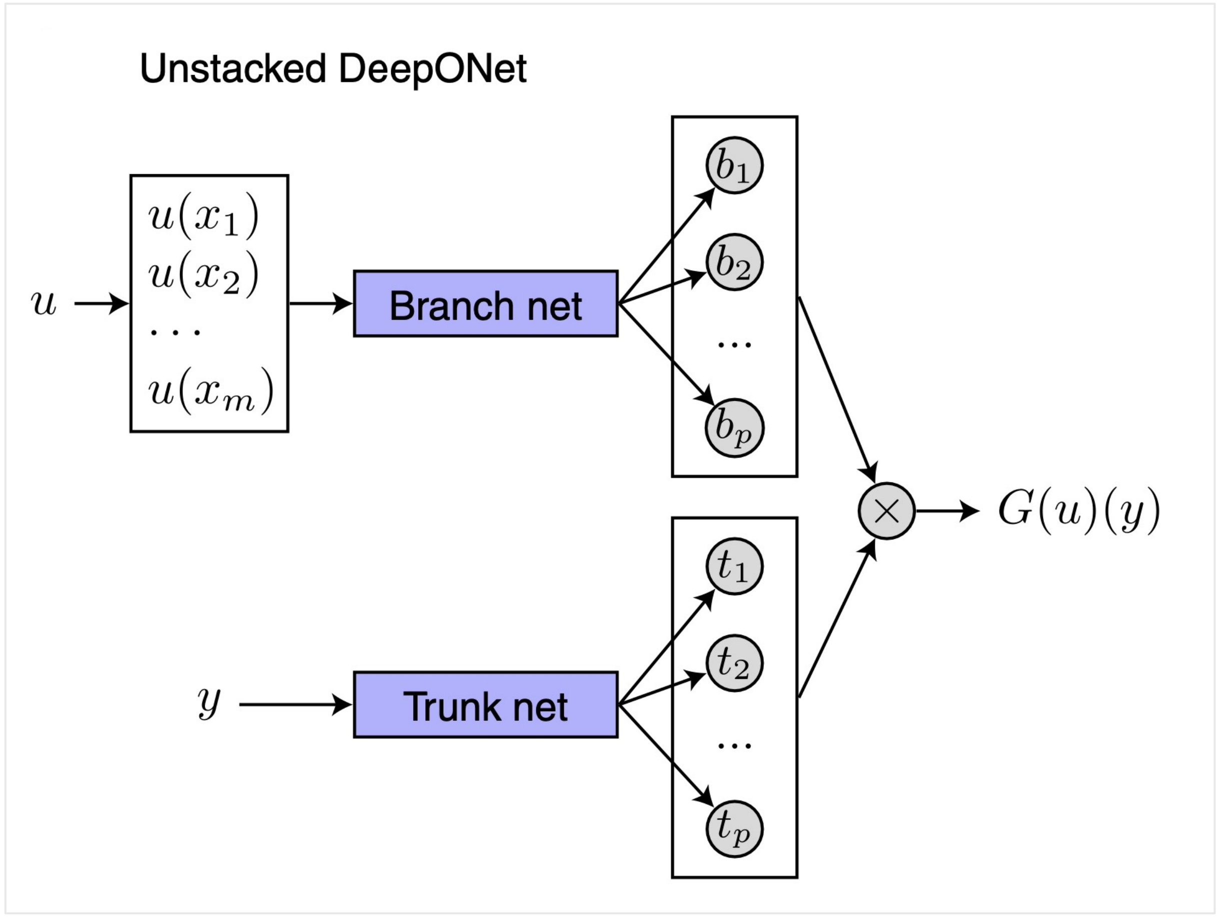
holds for all $u \in V$ and $y \in K_2$. $\mathbf{g}$ and $\mathbf{f}$ can be FNNs, RNNs or CNNs.

Generalized Universal Approximation Theorem for Operators (2019)



Stacked vs. Unstacked DeepONet





DeepONet Performance, Accuracy and Learning Speed

Antiderivative Operator

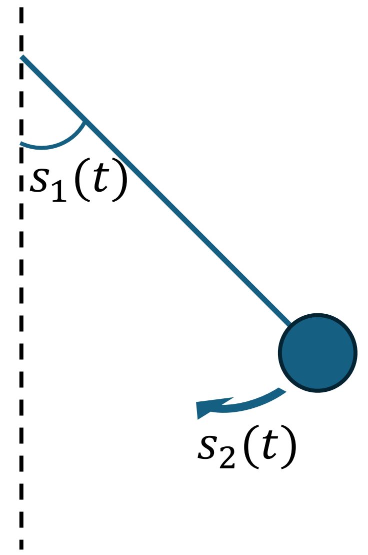
- Branch network input: $u(x_1), \dots, u(x_m)$
- Trunk network input: y

- Output/Target: $G(u)(y) = \int_0^y u(x)dx$

Pendulum

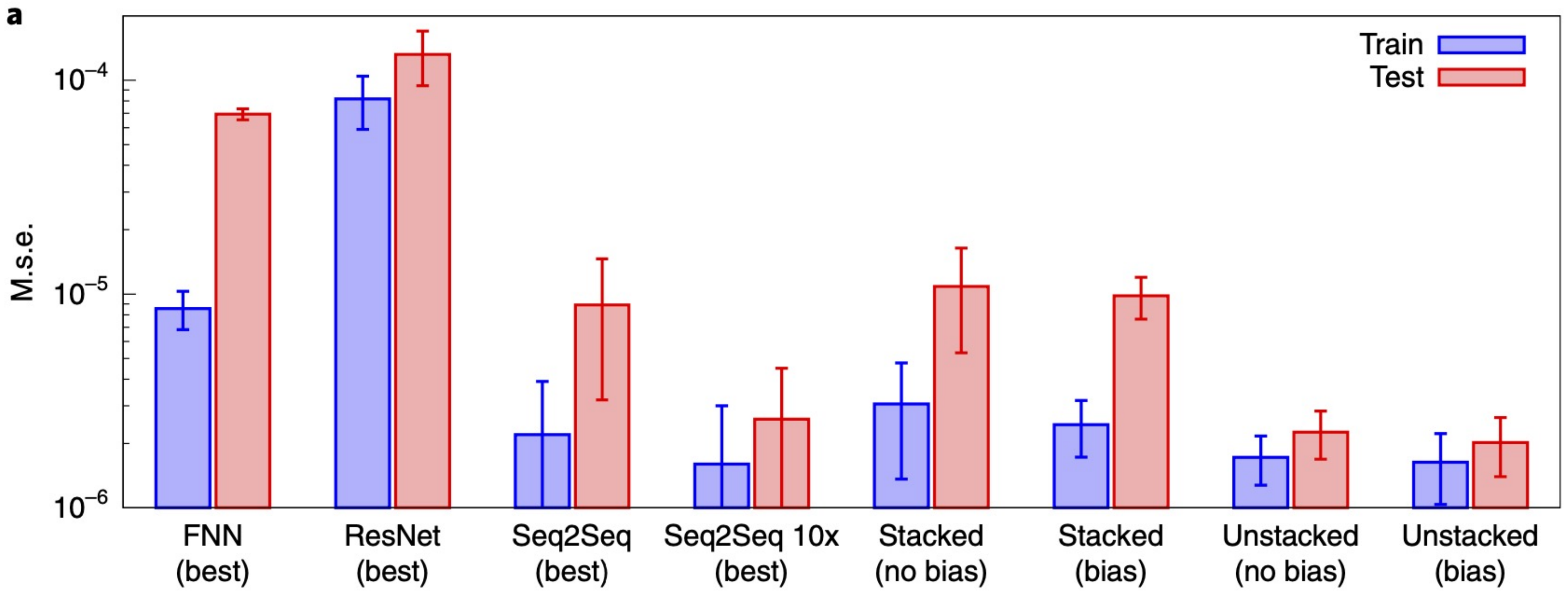
- Branch network input: $s_2'(t) = -k \sin s_1(t) + u(t)$
- Trunk network input: t

- Output/Target: $s_1(t)$



DeepONet Accuracy

- Errors of different network architectures trained to learn the antiderivative operator

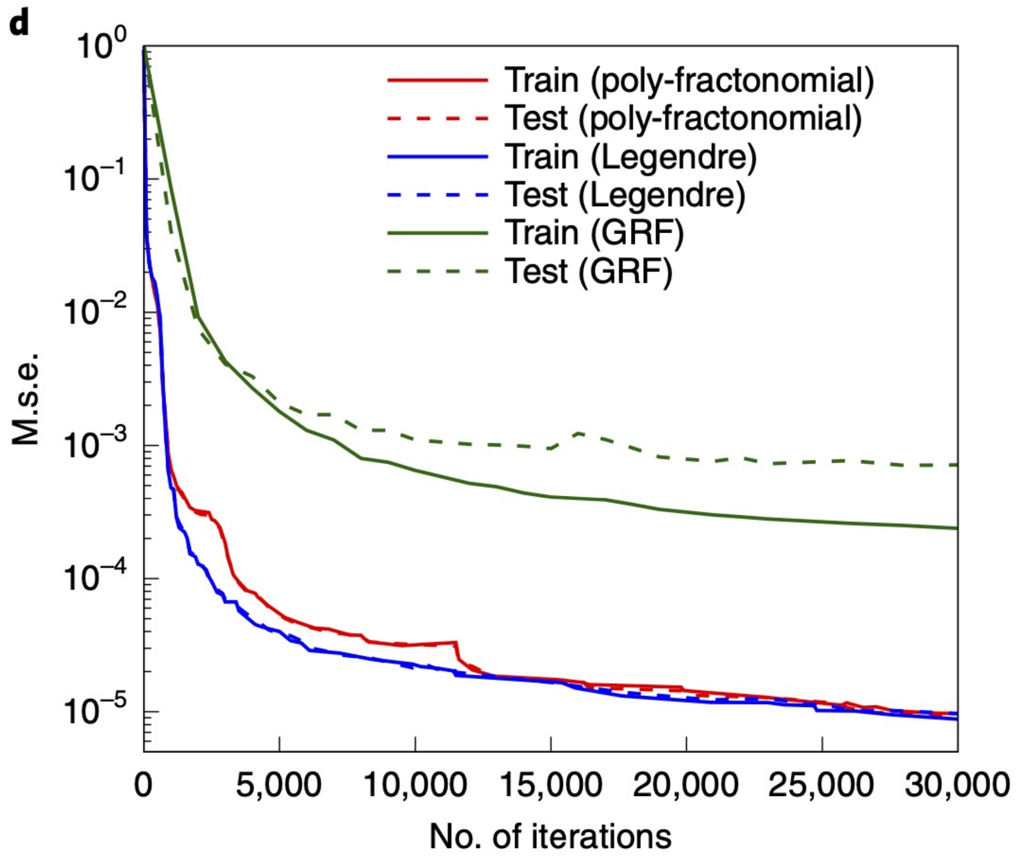


Introduction – Literature – Theory – **Performance** – Function Dimension – Sensors – Critique

DeepONet Accuracy with Different Input Function Spaces

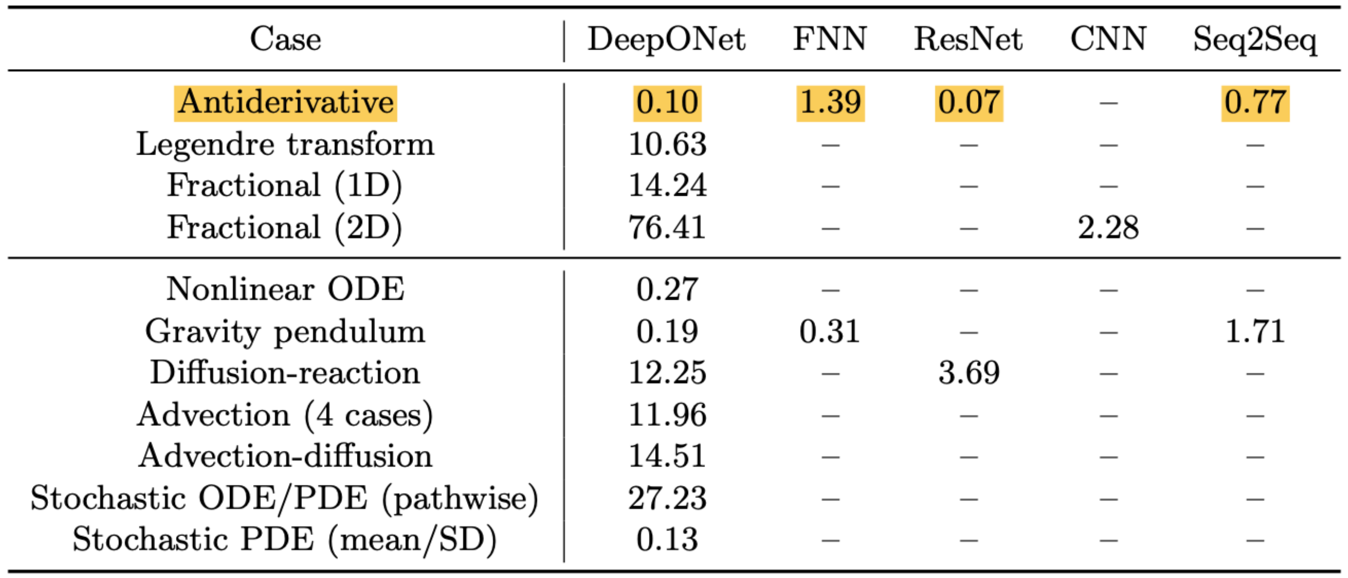
- Example of learning the Caputo fractional derivative:

- Functions types:
  - Poly-fractonomials
  - Legendre polynomials
  - Or from Gaussian Random Field



DeepONet Performance

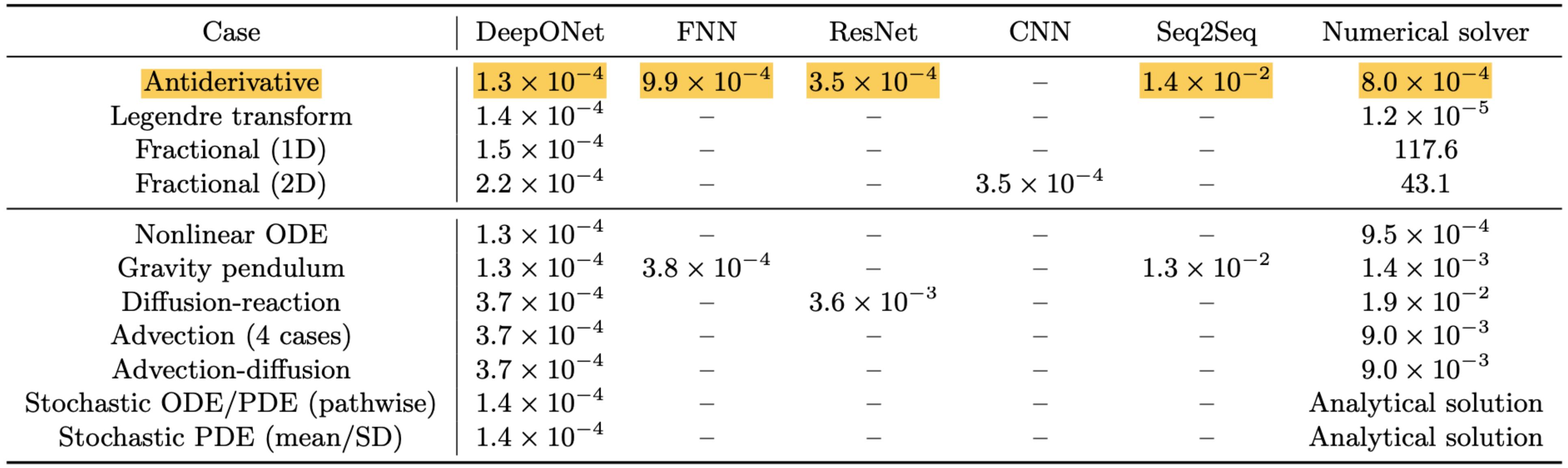
- Table S4: Computational cost (hours) of training of DeepONet and NN baselines



Case	DeepONet	FNN	ResNet	CNN	Seq2Seq
Antiderivative	0.10	1.39	0.07	–	0.77
Legendre transform	10.63	–	–	–	–
Fractional (1D)	14.24	–	–	–	–
Fractional (2D)	76.41	–	–	2.28	–
Nonlinear ODE	0.27	–	–	–	–
Gravity pendulum	0.19	0.31	–	–	1.71
Diffusion-reaction	12.25	–	3.69	–	–
Advection (4 cases)	11.96	–	–	–	–
Advection-diffusion	14.51	–	–	–	–
Stochastic ODE/PDE (pathwise)	27.23	–	–	–	–
Stochastic PDE (mean/SD)	0.13	–	–	–	–

DeepONet Performance

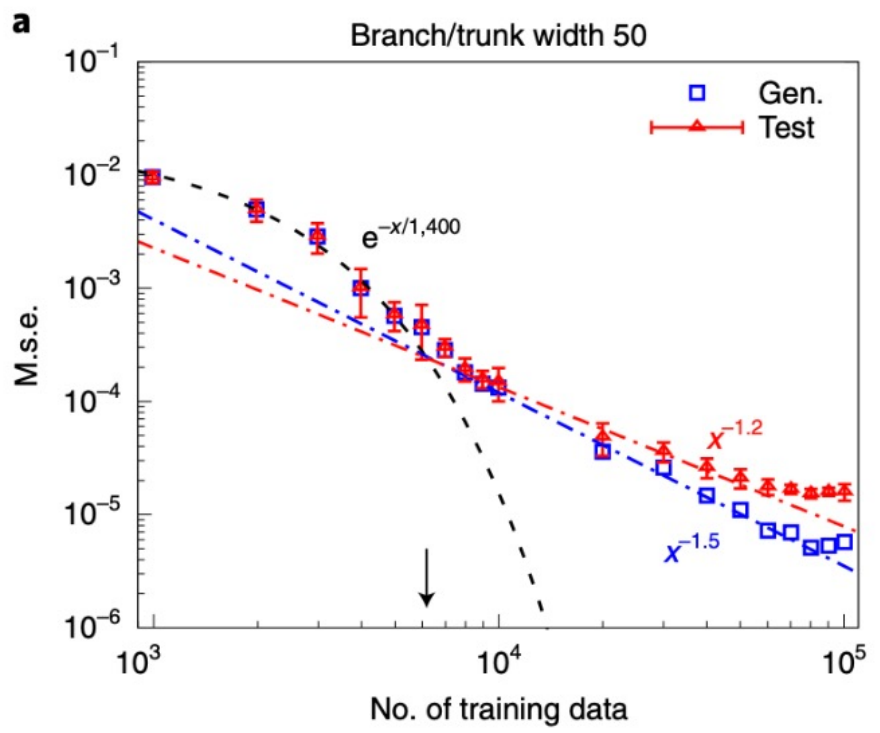
- Table S5: Computational cost (seconds) of running inference of DeepONet, NN baselines and numerical solvers.

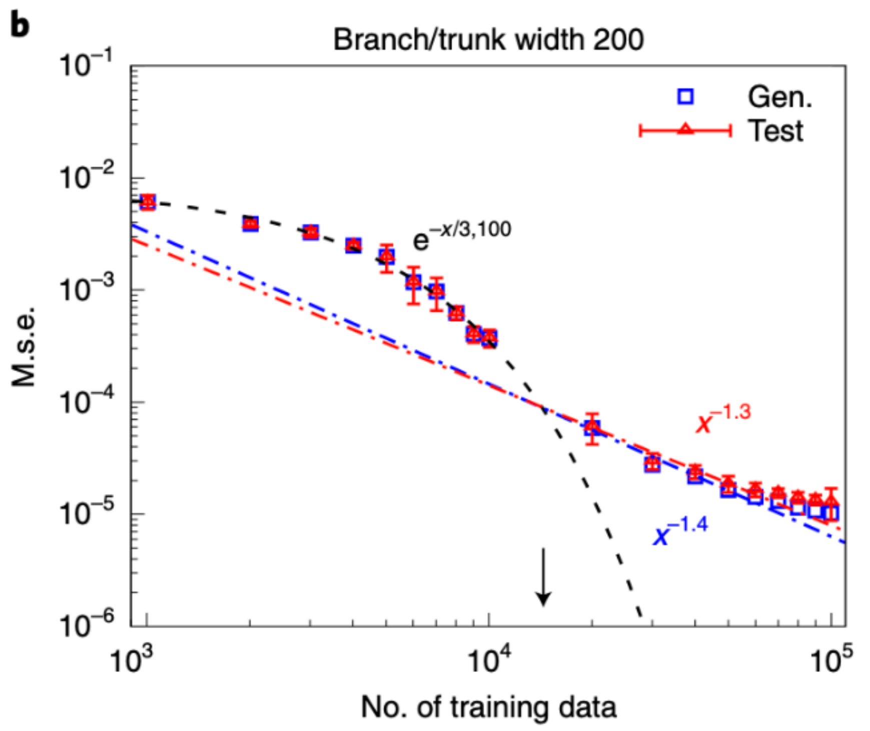


Case	DeepONet	FNN	ResNet	CNN	Seq2Seq	Numerical solver
Antiderivative	1.3×10^{-4}	9.9×10^{-4}	3.5×10^{-4}	–	1.4×10^{-2}	8.0×10^{-4}
Legendre transform	1.4×10^{-4}	–	–	–	–	1.2×10^{-5}
Fractional (1D)	1.5×10^{-4}	–	–	–	–	117.6
Fractional (2D)	2.2×10^{-4}	–	–	3.5×10^{-4}	–	43.1
Nonlinear ODE	1.3×10^{-4}	–	–	–	–	9.5×10^{-4}
Gravity pendulum	1.3×10^{-4}	3.8×10^{-4}	–	–	1.3×10^{-2}	1.4×10^{-3}
Diffusion-reaction	3.7×10^{-4}	–	3.6×10^{-3}	–	–	1.9×10^{-2}
Advection (4 cases)	3.7×10^{-4}	–	–	–	–	9.0×10^{-3}
Advection-diffusion	3.7×10^{-4}	–	–	–	–	9.0×10^{-3}
Stochastic ODE/PDE (pathwise)	1.4×10^{-4}	–	–	–	–	Analytical solution
Stochastic PDE (mean/SD)	1.4×10^{-4}	–	–	–	–	Analytical solution

DeepONet Error Convergence

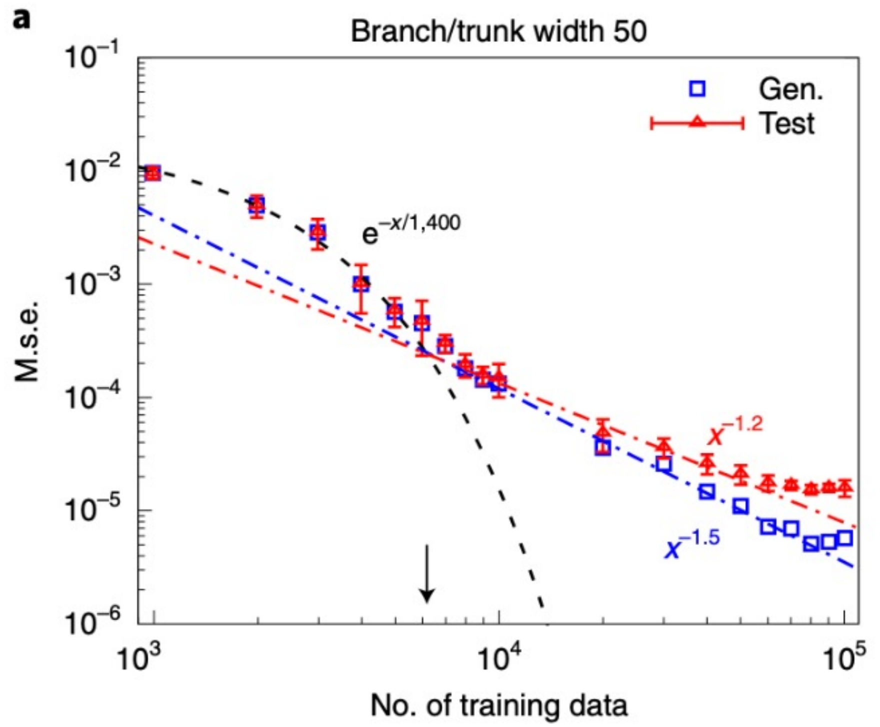
- MSE for different amounts of training data
  - in a nonlinear pendulum problem for k = 1, T = 3

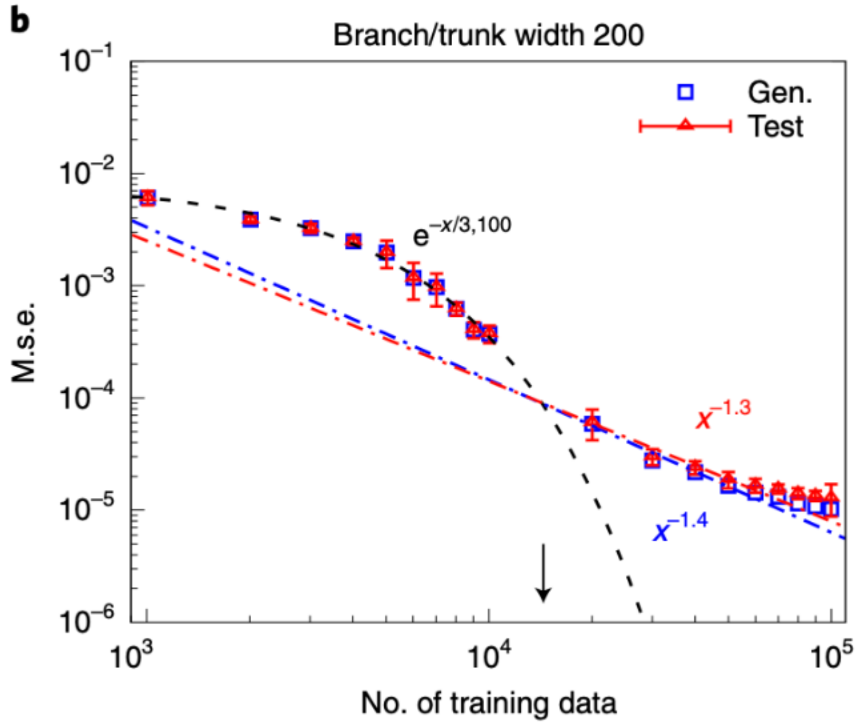




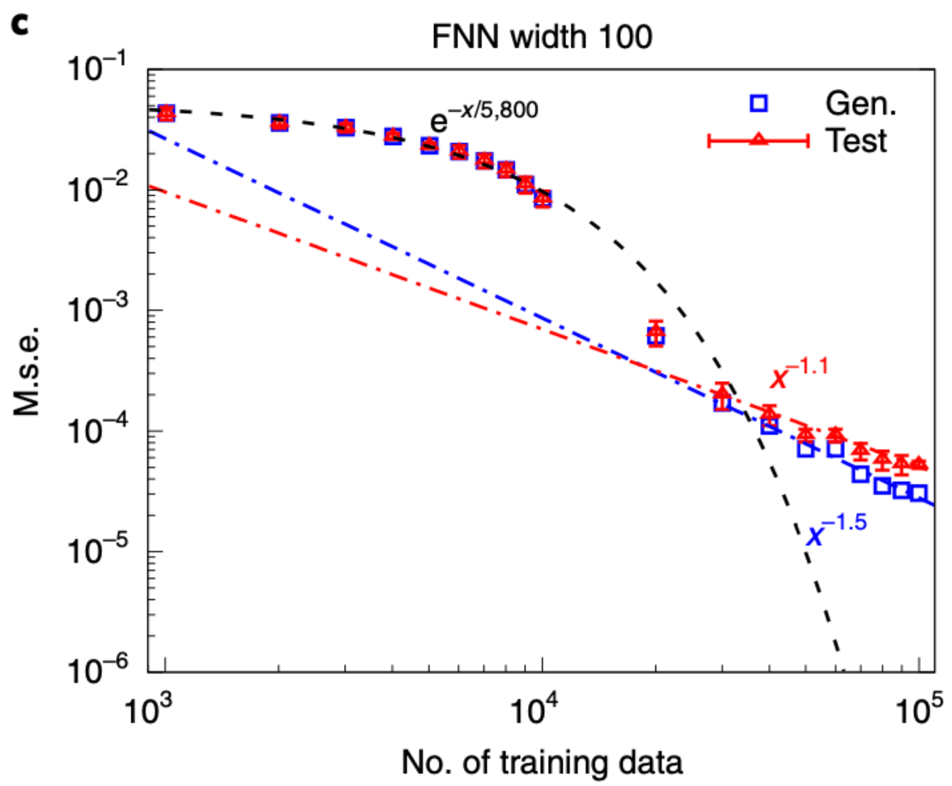
DeepONet Error Convergence

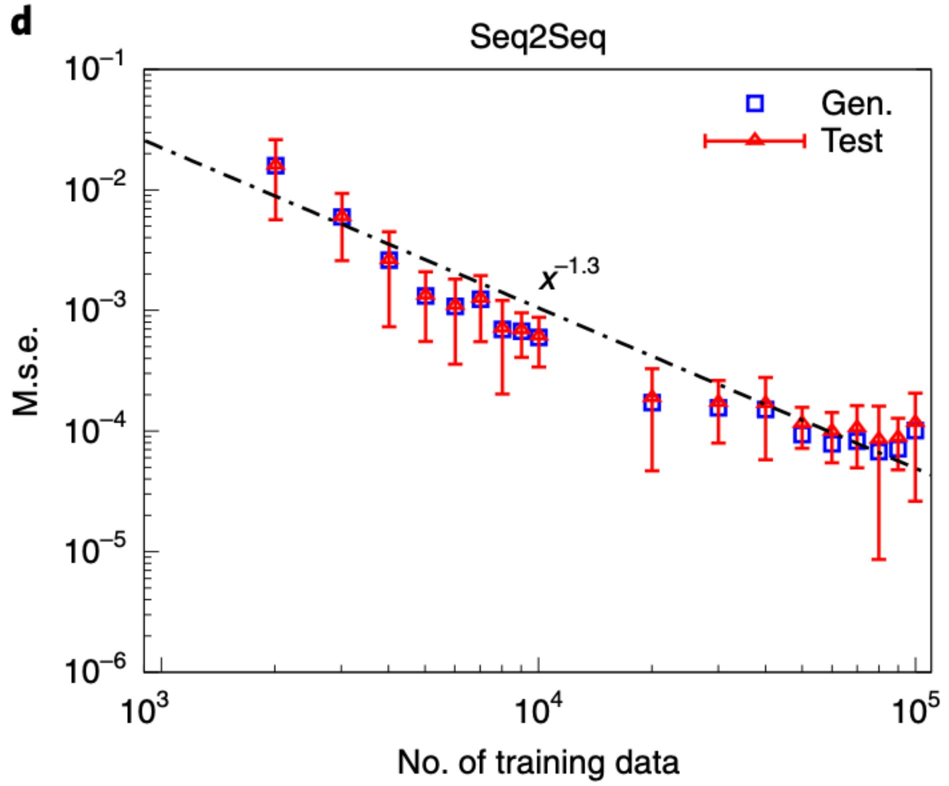
- small dataset: exponential convergence
- large dataset: polynomial rates





DeepONet Error Convergence





DeepONet and Function Dimension

Stochastic Operators

How does DeepONet behave with high-dimensional input?

Stochastic Operators

How does DeepONet behave with high-dimensional input?

Consider the following ODE, representing a population growth model:

$$\frac{d}{dt}y(t;\omega) = k(t;\omega)y(t;\omega)$$

$$t \in (0,1] \text{ and } \omega \in \Omega, \qquad y(0) = 1$$

Stochastic Operators

$$\frac{d}{dt}y(t;\omega) = k(t;\omega)y(t;\omega)$$

Stochastic Operators

$$\frac{d}{dt}y(t;\omega) = k(t;\omega)y(t;\omega)$$

Stochastic Operators

$$\frac{d}{dt} y(t;\omega) = k(t;\omega) y(t;\omega)$$

$$k(t;\omega) \sim \mathcal{GP}(0, \mathrm{Cov}(t_i, t_j))$$

Stochastic Operators

$$\frac{d}{dt}y(t;\omega) = k(t;\omega)y(t;\omega)$$

$$\begin{pmatrix} k(t_1;\omega) \\ \vdots \\ k(t_m;\omega) \end{pmatrix} \sim \mathrm{Norm}\left(0, \mathrm{Cov}\left(t_i, t_j\right)\right)$$

Stochastic Operators

$$\frac{d}{dt} y(t;\omega) = k(t;\omega) y(t;\omega)$$

$$\begin{pmatrix} k(t_1;\omega) \\ \vdots \\ k(t_m;\omega) \end{pmatrix} \sim \mathrm{Norm}\left(0, \mathrm{Cov}(t_i, t_j)\right)$$

Stochastic Operators

$$\frac{d}{dt}y(t;\omega) = k(t;\omega)y(t;\omega)$$

$$\begin{pmatrix} k(t_1;\omega) \\ \vdots \\ k(t_m;\omega) \end{pmatrix} \sim \mathrm{Norm}\left(0, \mathrm{Cov}(t_i, t_j)\right)$$

$$\begin{pmatrix} k(t_1;\omega) \\ \vdots \\ k(t_m;\omega) \end{pmatrix} \approx \sum_{h=1}^{N} \sqrt{\lambda_h} \begin{bmatrix} e_h(t_1) \\ \vdots \\ e_h(t_m) \end{bmatrix} \xi_h(\omega)$$

Stochastic Operators

$$\frac{d}{dt}y(t;\omega) = k(t;\omega)y(t;\omega)$$

$$\begin{pmatrix} k(t_1;\omega) \\ \vdots \\ k(t_m;\omega) \end{pmatrix} \sim \mathrm{Norm}\left(0, \mathrm{Cov}(t_i, t_j)\right)$$

$$\begin{pmatrix} k(t_1;\omega) \\ \vdots \\ k(t_m;\omega) \end{pmatrix} \approx \sum_{h=1}^{N} \sqrt{\lambda_h} \underbrace{\begin{bmatrix} e_h(t_1) \\ \vdots \\ e_h(t_m) \end{bmatrix}}_{\text{Eigenfunctions of } \mathrm{Cov}(t_i, t_j)} \underbrace{\xi_h(\omega)}_{\text{Independent standard Gaussian random variables}}$$

Introduction – Literature – Theory – Performance – **Function Dimension** – Sensors – Critique

Stochastic Operators

Branch network input:

$$[\sqrt{\lambda_1}e_1(t), \dots, \sqrt{\lambda_N}e_N(t)] \in \mathbb{R}^{N\times m}$$

$$\sqrt{\lambda_h}e_h(t) = \sqrt{\lambda_h}[e_h(t_1), \dots, e_h(t_m)] \in \mathbb{R}^{m}$$

Trunk network input:

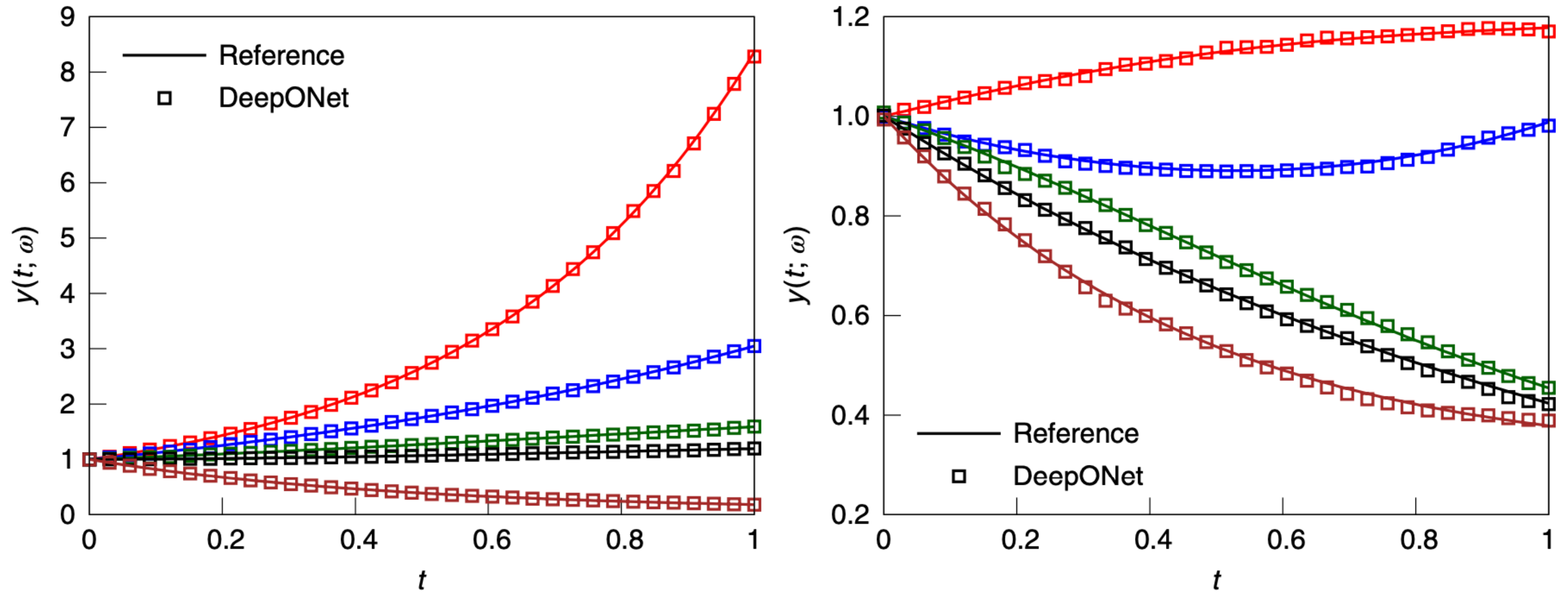
$$[t, \xi_1, \dots, \xi_N] \in \mathbb{R}^{N+1}$$

Target/Output:

$$y(t;\omega)$$

Performance and Function Dimension

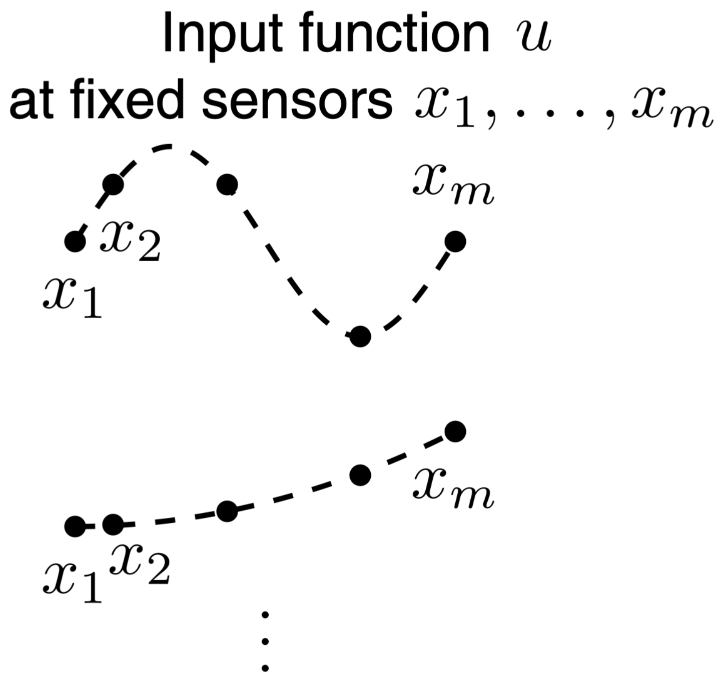
- DeepONet prediction for a stochastic ODE.
  - DeepONet prediction (symbols) is very close to the reference solution for 10 different random samples (five in each panel) from $k(x;\omega)$ with $l = 1.5$

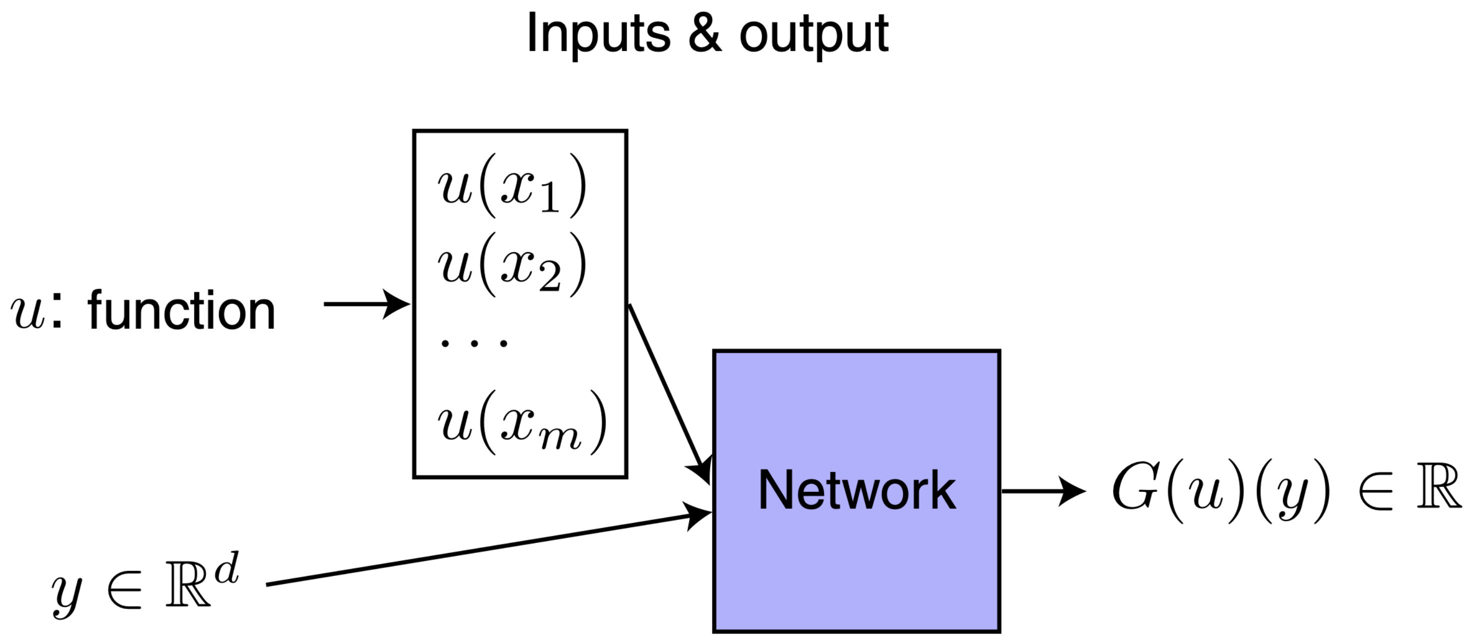


Test mean squared error of $8.0{\times}10^{-5} \pm 3.4{\times}10^{-5}$

Sensor Constraints and our Results

Choosing different values for m





Theorem 3

Consider the following ODE:

$$G(u)(x) = s_0 + \int_a^x g(G(u)(t), u(t), t)dt$$

Theorem 3

Consider the following ODE:

$$G(u)(x) = s_0 + \int_a^x g(G(u)(t), u(t), t)dt$$

$$g(G(u)(t), u(t), t) = u(t)$$

Theorem 3

Consider the following ODE:

$$G(u)(x) = s_0 + \int_a^x \qquad\qquad\qquad\qquad dt$$

$$g(G(u)(t), u(t), t) = u(t)$$

Theorem 3

Consider the following ODE:

$$G(u)(x) = s_0 + \int_a^x u(t)\,dt$$

$$g(G(u)(t), u(t), t) = u(t)$$

Theorem 3

Consider the following ODE:

$$G(u)(x) = s_0 + \int_a^x g(G(u)(t), u(t), t)dt$$

Theorem 3

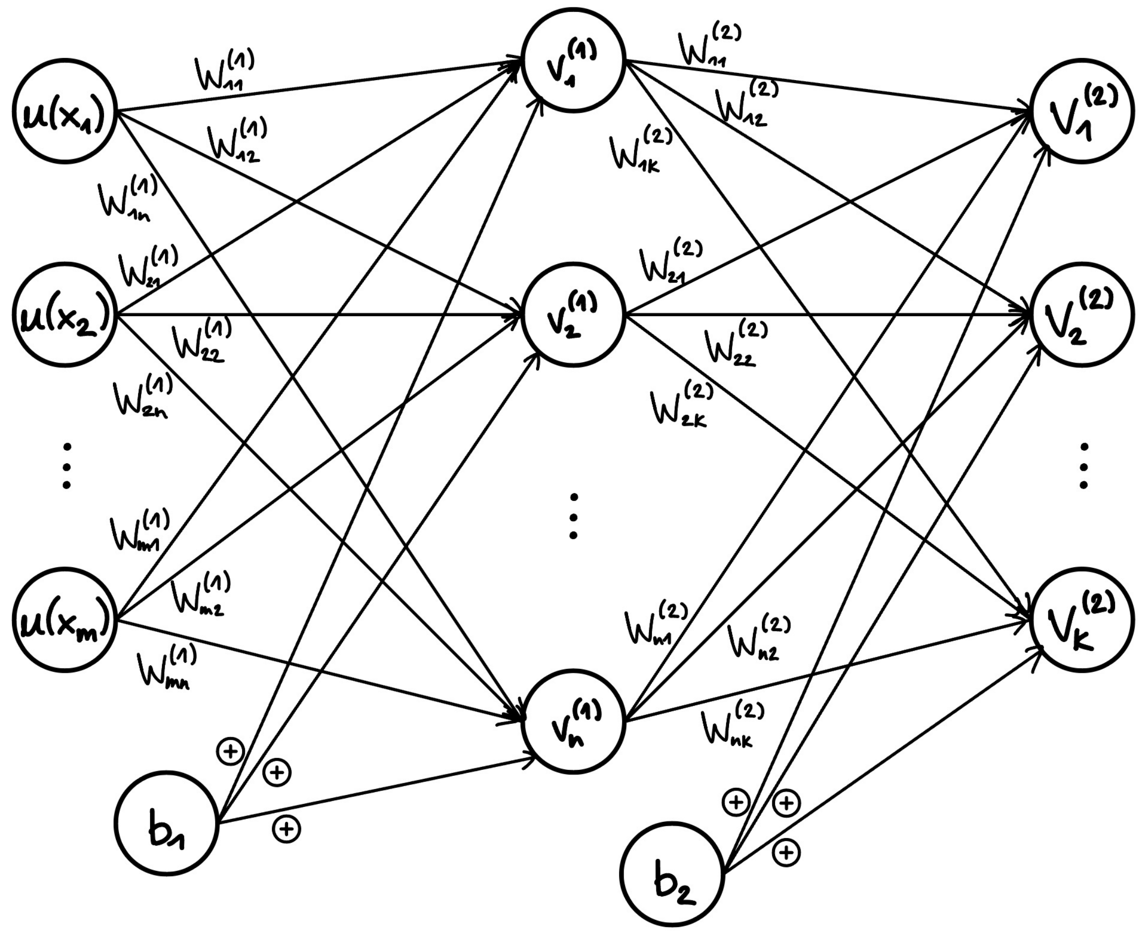
$$G(u)(x) = s_0 + \int_a^x g(G(u)(t), u(t), t)dt$$

For a given number m of sensors, a function space V and boundaries $[a, b]$ making the constant $\kappa'(m, V, a, b)$ less than ε, there exist for any $d \in [a, b]$, parameters $W_1 \in \mathbb{R}^{n \times m}, W_2 \in \mathbb{R}^{K \times n}$, $b_1 \in \mathbb{R}^m, b_2 \in \mathbb{R}^K$ such that

$$\|G(u)(d) - W_2\sigma(W_1[u(x_1), \dots, u(x_m)]^T + b_1) + b_2)\|_2 < \varepsilon$$

holds for all $u \in V$.

Theorem 3

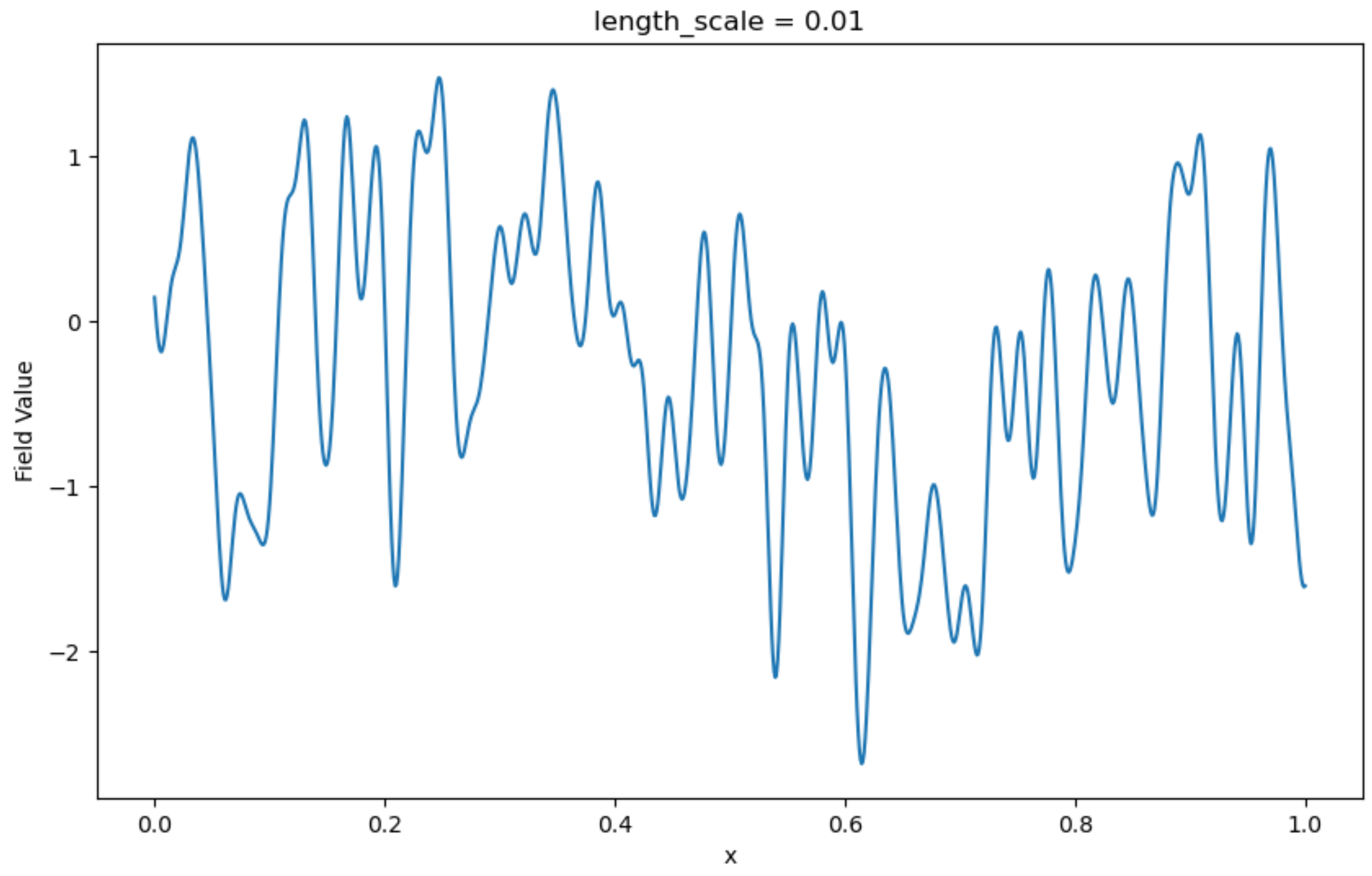


$$\left\| G(u)(d) - \begin{pmatrix} v_1^{(2)} \\ \vdots \\ v_K^{(2)} \end{pmatrix} \right\|_2 < \varepsilon$$

Demonstration of our Results

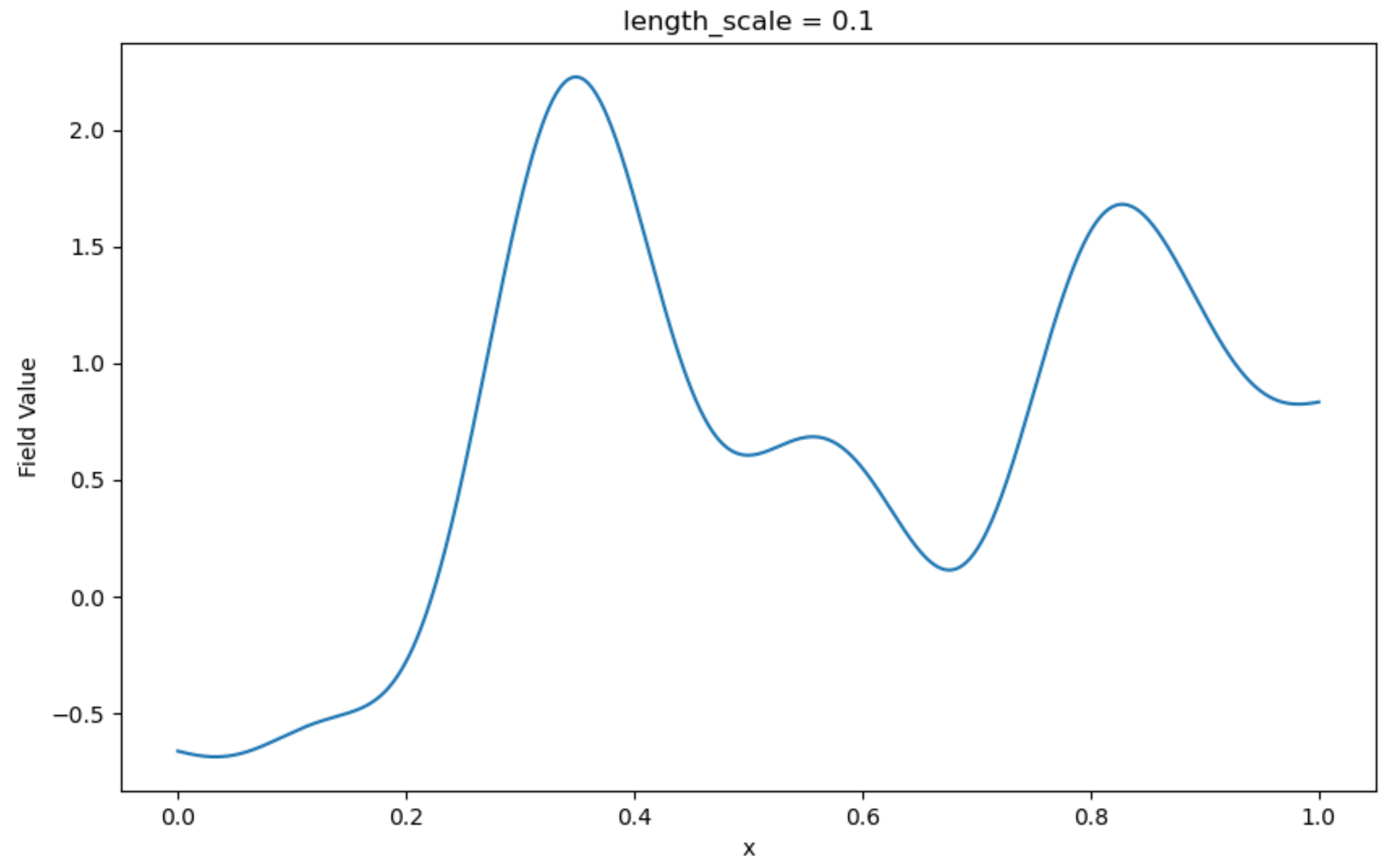
Choosing different values for m

$l = 0.01$:



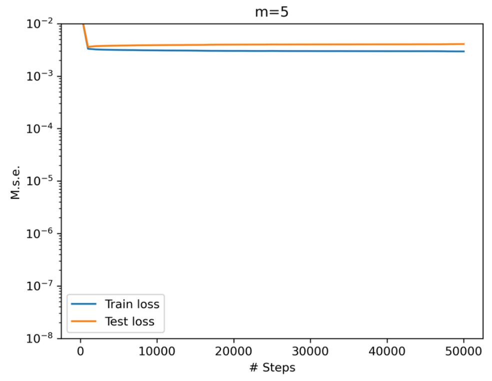
Choosing different values for m

$l = 0.1$:

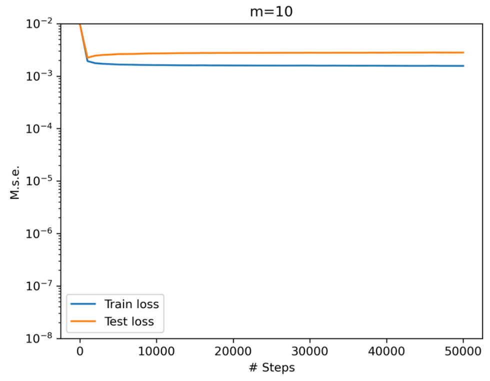


$l = 0.025$

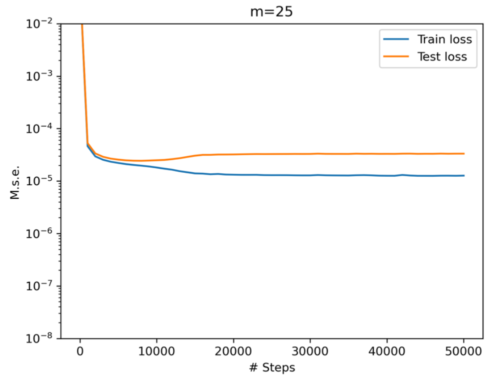
$m = 5$



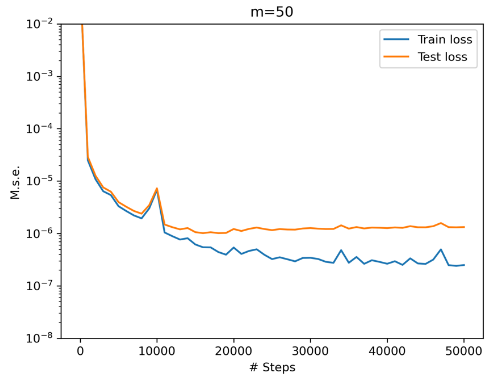
$m = 10$



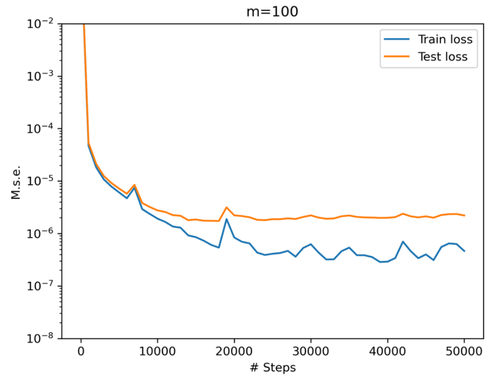
$m = 25$



$m = 50$

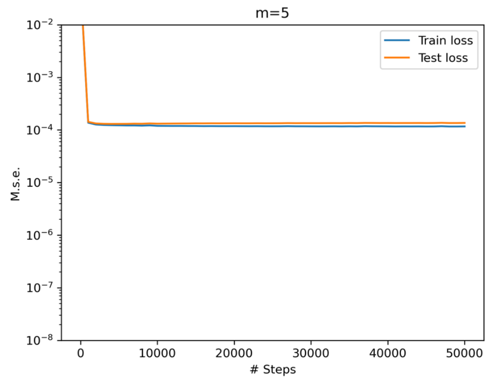


$m = 100$

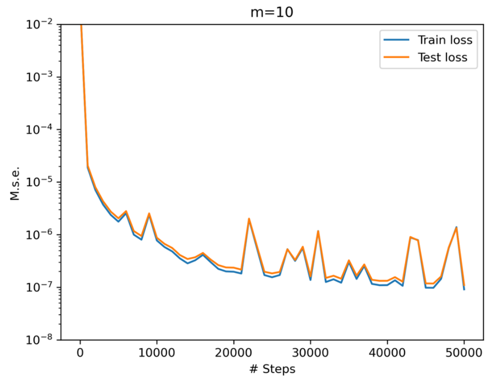


$l = 0.2$

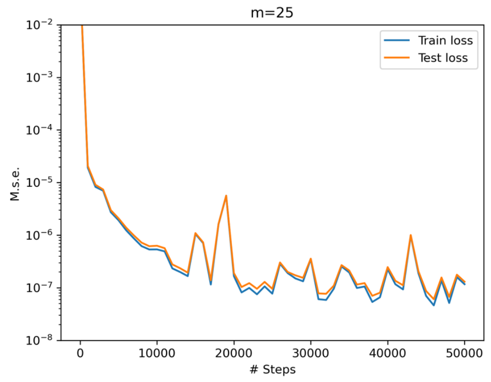
$m = 5$



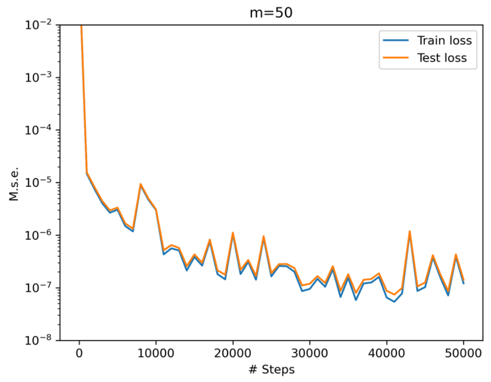
$m = 10$



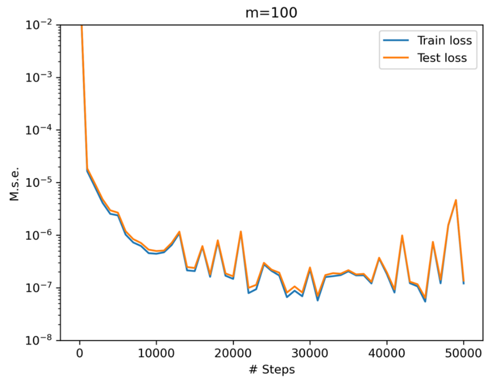
$m = 25$



$m = 50$



$m = 100$



Critique and Future Research

Paper Critique

- Error Convergence is not explained
- Overfitting potential is not addressed
- Universal approximation theorem does not account for optimization and generalization errors
- No proper comparison to numerical solvers

- Requirement for fixed sensor locations for the input functions is limiting
    - Real-world data may not satisfy the limitations

Further Research

- Further generalize data constraints
  - Potentially no more requirement for fixed sensor locations
- Operator approximation is much more computationally intensive than function approximation
- How does DeepONet perform using CNNs?
  - May further improve the accuracy

DeepONet: Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators, Lu Lu et al.

Presented by Gaspard Krief and Emilia Pucher

Additional Slide: Stacked vs. Unstacked DeepONet Peformance

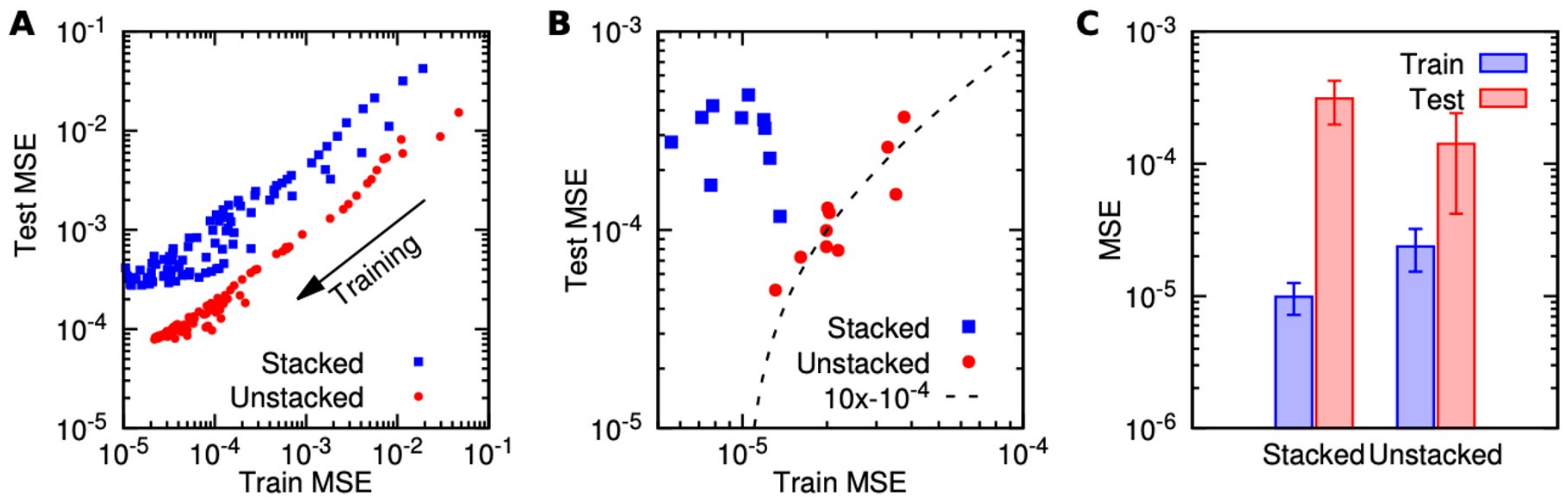
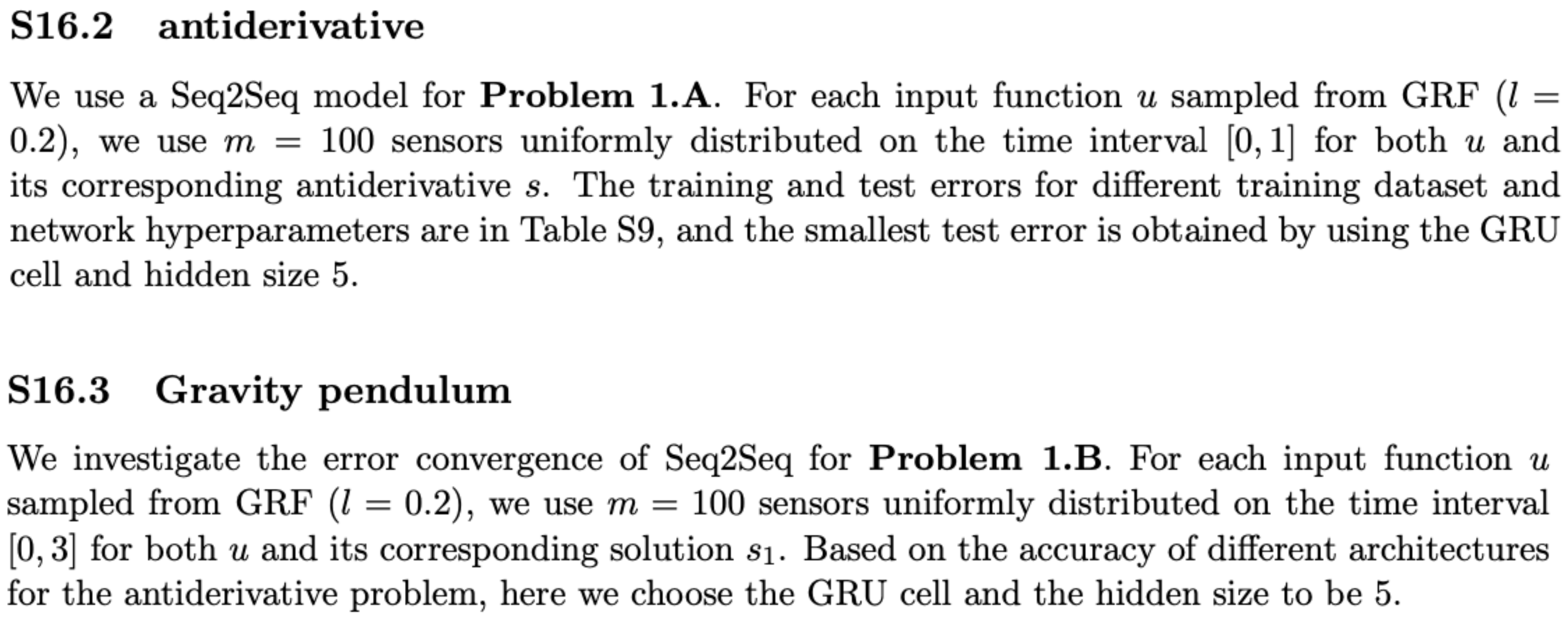


Fig. S3: **Nonlinear ODE: unstacked DeepONets have smaller generalization error and smaller test MSE than stacked DeepONets.** (**A**) The correlation between the training MSE and the test MSE of one stacked/unstacked DeepONet during the training process. (**B**) The correlation between the final training MSE and test MSE of stacked/unstacked DeepONets in 10 runs with random training dataset and network initialization. The training and test MSE of unstacked DeepONets follow a linear correlation (black dash line). (**C**) The mean and one-standard-derivation of data points in B.

Additional Slide: Seq2Seq Architecture



S16.2 antiderivative

We use a Seq2Seq model for **Problem 1.A**. For each input function u sampled from GRF ($l = 0.2$), we use $m = 100$ sensors uniformly distributed on the time interval $[0, 1]$ for both u and its corresponding antiderivative s. The training and test errors for different training dataset and network hyperparameters are in Table S9, and the smallest test error is obtained by using the GRU cell and hidden size 5.

S16.3 Gravity pendulum

We investigate the error convergence of Seq2Seq for **Problem 1.B**. For each input function u sampled from GRF ($l = 0.2$), we use $m = 100$ sensors uniformly distributed on the time interval $[0, 3]$ for both u and its corresponding solution s_1. Based on the accuracy of different architectures for the antiderivative problem, here we choose the GRU cell and the hidden size to be 5.

Additional Slide: DeepONet Sizes for Testing

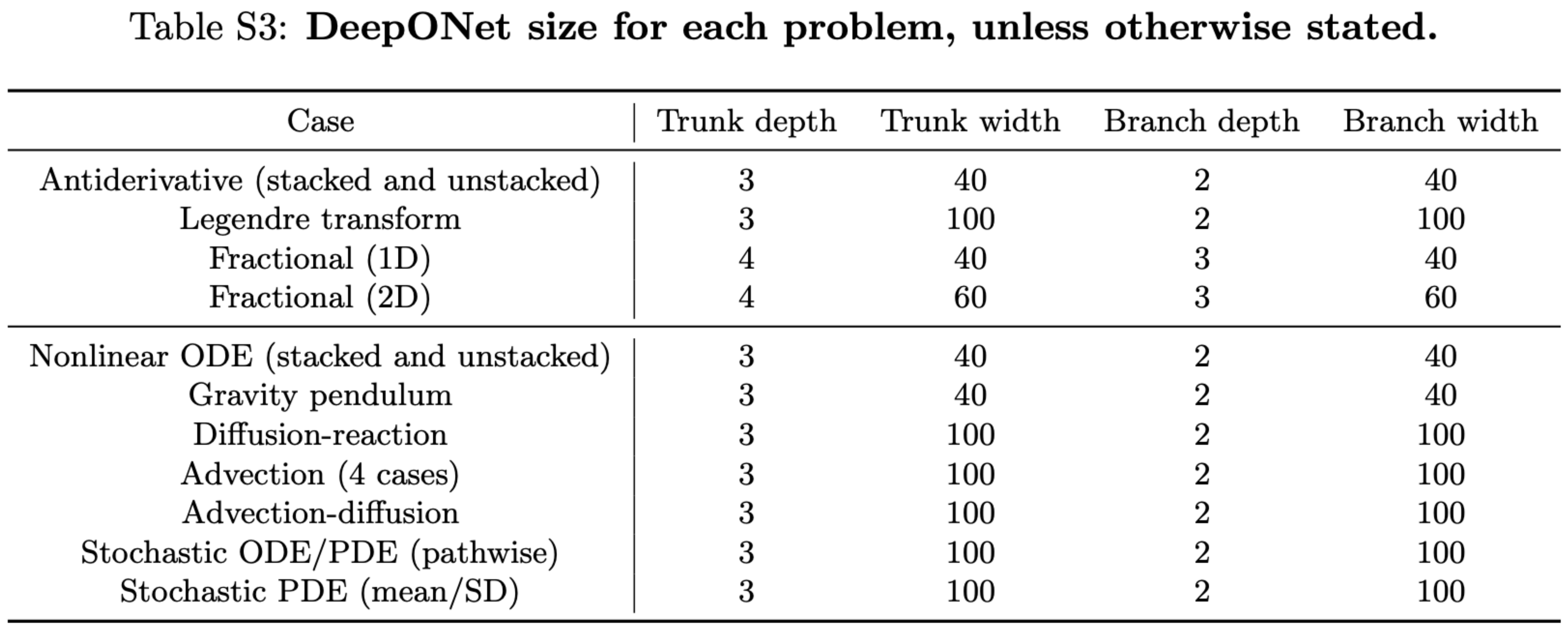


Table S3: **DeepONet size for each problem, unless otherwise stated.**

Case	Trunk depth	Trunk width	Branch depth	Branch width
Antiderivative (stacked and unstacked)	3	40	2	40
Legendre transform	3	100	2	100
Fractional (1D)	4	40	3	40
Fractional (2D)	4	60	3	60
Nonlinear ODE (stacked and unstacked)	3	40	2	40
Gravity pendulum	3	40	2	40
Diffusion-reaction	3	100	2	100
Advection (4 cases)	3	100	2	100
Advection-diffusion	3	100	2	100
Stochastic ODE/PDE (pathwise)	3	100	2	100
Stochastic PDE (mean/SD)	3	100	2	100