DeepONet: Learning Operators with Neural Networks

Krishna Kumar

University of Texas at Austin

krishnak@utexas.edu

Overview

Outline

Learning Objectives

- Understand the leap from function approximation to operator learning
- Master the Universal Approximation Theorem for Operators
- Learn the DeepONet architecture: Branch and Trunk networks
- Implement operator learning for the derivative operator
- Apply DeepONet to the 1D nonlinear Darcy problem
- Extend to Physics-Informed DeepONet with PDE constraints

Open Notebook: DeepONet

Open Notebook: PI-DeepONet

The Fundamental Question

We've learned how neural networks can approximate **functions**: $f : \mathbb{R}^d \to \mathbb{R}^m$.

But what if we want to learn mappings between infinite-dimensional function spaces?

Enter **operators**: mappings that take functions as input and produce functions as output.

$$\mathcal{G} : \mathcal{A} \to \mathcal{U}$$

where $\mathcal{A}$ and $\mathcal{U}$ are function spaces.

Examples of operators:

- **Derivative operator:** $\mathcal{G}u = \frac{du}{dx}$
- **Integration operator:** $\mathcal{G}f = \int_0^x f(t)dt$
- **PDE solution operator:** Given boundary conditions or source terms, map to the PDE solution

Traditional Function Approximation

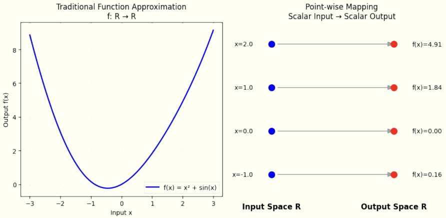
What we're familiar with:

Point-wise mapping:

- **Input:** A number $x = 2.5$
- **Output:** A number $f(x) = x^2 = 6.25$

Characteristics:

- Input: Single numbers (scalars)
- Output: Single numbers (scalars)
- Learn: Point-wise mappings
- Architecture: Standard feedforward



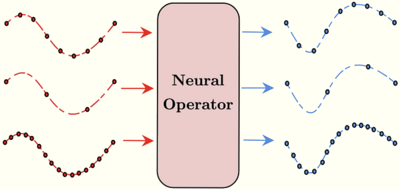
Traditional neural networks learn scalar-to-scalar mappings.

Operator Learning: The Next Level

The paradigm shift:

Function-to-function mapping:

- **Input:** An entire function $u(x) = \sin(x)$
- **Output:** Another entire function $\mathcal{G}[u](x) = \cos(x)$ (the derivative!)



Operators map entire functions to entire functions.

Examples in science:

- $\mathcal{D}[u] = \frac{du}{dx}$ (Differentiation)
- $\mathcal{I}[f] = \int_0^x f(t)dt$ (Integration)
- $\mathcal{S}[f] = u$ where $\nabla^2 u = f$ (PDE solution)

The Challenge with Traditional Neural Networks

Standard neural networks learn point-wise mappings: $\mathbb{R}^n \to \mathbb{R}^m$. But operators map functions to functions.

Traditional approach limitations:

- **Fixed discretization:** Networks trained on specific grids can't generalize to different resolutions
- **Curse of dimensionality:** High-dimensional function spaces are computationally intractable
- **No theoretical foundation:** No guarantee that standard networks can approximate operators

How do we represent infinite-dimensional functions with finite data?

Solution: We need a fundamentally different architecture that can handle function inputs and outputs while maintaining theoretical guarantees.

Outline

The Breakthrough Theorem

Just as the Universal Approximation Theorem tells us neural networks can approximate functions, there's a remarkable extension:

Theorem (Chen & Chen, 1995)

Neural networks can approximate **operators** that map functions to functions!

Mathematical Statement: For any continuous operator $\mathcal{G} : V \subset C(K_1) \to C(K_2)$ and $\epsilon > 0$, there exist constants such that:

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

Decoding the Theorem

This looks complex, but the insight is beautiful:

1. **Branch Network:** Processes function u sampled at sensor points $\{x_j\}$
2. **Trunk Network:** Processes output coordinates y
3. **Combination:** Multiply branch and trunk outputs, then sum

Key insight

Any operator can be written as:

$$\mathcal{G}(u)(y) \approx \sum_{k=1}^{p} b_k(u) \cdot t_k(y)$$

where b_k depends only on the input function and t_k depends only on the output location!

This decomposition is the theoretical foundation for the DeepONet architecture.

Outline

DeepONet: The Practical Implementation

DeepONet (Deep Operator Network) is the practical implementation of the Operator Universal Approximation Theorem.

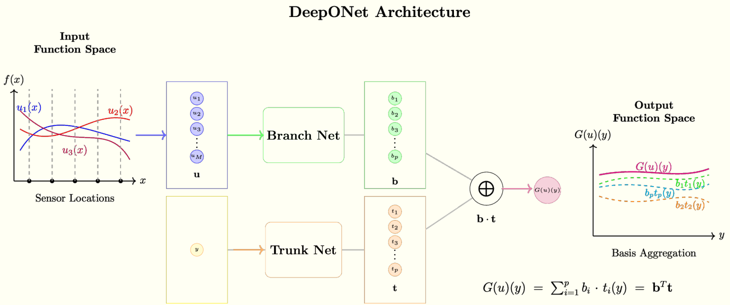
Core Architecture

$$\mathcal{G}_\theta(u)(y) = \sum_{k=1}^{p} b_k(u) \cdot t_k(y) + b_0$$

where:

- **Branch network:** $b_k(u) = \mathcal{B}_k([u(x_1), u(x_2), \ldots, u(x_m)])$
- **Trunk network:** $t_k(y) = \mathcal{T}_k(y)$
- p**:** Number of basis functions (typically 50-200)
- b_0**:** Bias term

DeepONet Architecture Diagram



DeepONet architecture showing branch and trunk networks.

Data flow:

1. Input function u sampled at sensor points $\rightarrow$ Branch network $\rightarrow$ Coefficients $\{b_k\}$
2. Query points y $\rightarrow$ Trunk network $\rightarrow$ Basis functions $\{t_k\}$
3. Element-wise multiplication and summation $\rightarrow$ Output $\mathcal{G}(u)(y)$

Training Data Structure and Loss Function

Input-output pairs: $(u^{(i)}, y^{(j)}, \mathcal{G}(u^{(i)})(y^{(j)}))$

- N input functions: $\{u^{(i)}\}_{i=1}^{N}$
- Each function sampled at m sensors: $\{u^{(i)}(x_j)\}_{j=1}^{m}$
- Corresponding outputs at **query points**: $\{\mathcal{G}(u^{(i)})(y_k)\}$

Loss Function

$$\mathcal{L}(\theta) = \frac{1}{N \cdot P} \sum_{i=1}^{N} \sum_{k=1}^{P} \left| \mathcal{G}_\theta(u^{(i)})(y_k) - \mathcal{G}(u^{(i)})(y_k) \right|^2$$

Key Advantages:

- **Resolution independence:** Train on one grid, evaluate on any grid
- **Fast evaluation:** Once trained, instant prediction (no iterative solving)
- **Generalization:** Works for new functions not seen during training

Outline

Perfect Starting Point: Learning Derivatives

Problem Setup: Learn the derivative operator $\mathcal{D}[u] = \frac{du}{dx}$

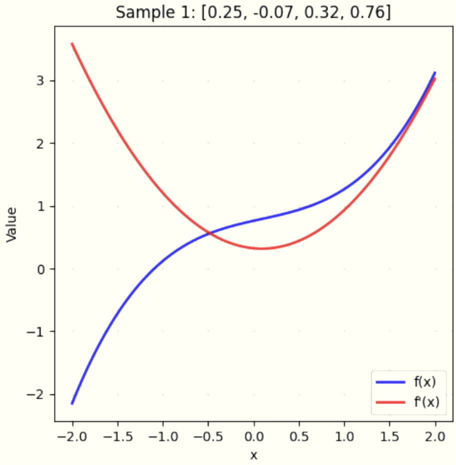
Why this example:

- Simple and intuitive
- Exact analytical solution for verification
- Shows how DeepONet learns basis decompositions
- Bridges function approximation $\rightarrow$ operator learning

Input functions: Cubic polynomials

$$u(x) = ax^3 + bx^2 + cx + d$$

Target operator:



Sample cubic polynomials and their derivatives

The DeepONet Challenge

Key insight: The derivative of a cubic is always quadratic, so it can be written as:

$$\frac{du}{dx} = w_1 \cdot 1 + w_2 \cdot x + w_3 \cdot x^2$$

where $w_1 = c$, $w_2 = 2b$, $w_3 = 3a$.

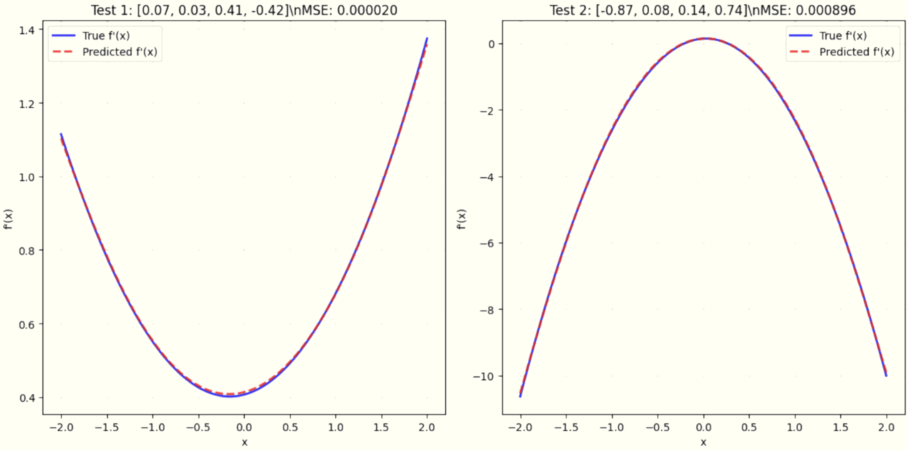
The DeepONet challenge

Can it learn this mapping automatically without being told the explicit form?

What we expect:

- Branch network should learn to extract coefficients $\{a, b, c\}$
- Trunk network should learn basis functions $\{1, x, x^2\}$
- The combination should reproduce the derivative exactly

Prediction Results

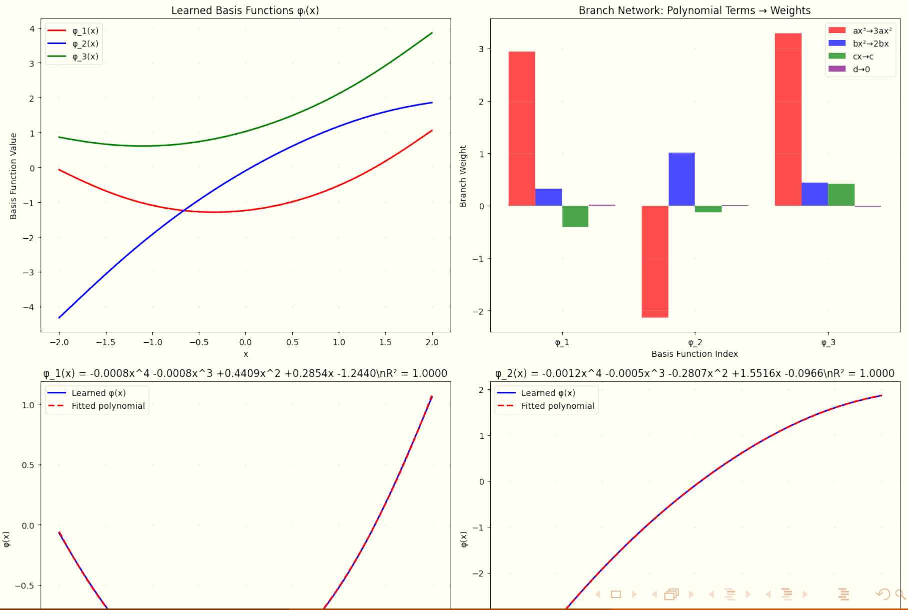


DeepONet predictions vs. true derivatives for test polynomials.

Results:

- Perfect agreement between predicted and true derivatives

Understanding What DeepONet Learned



Outline

A Real PDE: The 1D Nonlinear Darcy Problem

Now for something more challenging: A real PDE with nonlinear physics!

Problem Formulation

The 1D nonlinear Darcy equation models groundwater flow with solution-dependent permeability:

$$\frac{d}{dx}\left(-\kappa(u(x))\frac{du}{dx}\right) = f(x), \quad x \in [0, 1]$$

where:

- $u(x)$ is the **solution field** (pressure/hydraulic head)
- $\kappa(u) = 0.2 + u^2(x)$ is the **nonlinear permeability**
- $f(x) \sim \mathrm{GP}(0, k(x, x'))$ is a **Gaussian random field source**
- Homogeneous Dirichlet BCs: $u(0) = 0$, $u(1) = 0$

The Operator Learning Challenge

Goal: Learn the solution operator $\mathcal{G}$ such that:

$$\mathcal{G}[f] = u$$

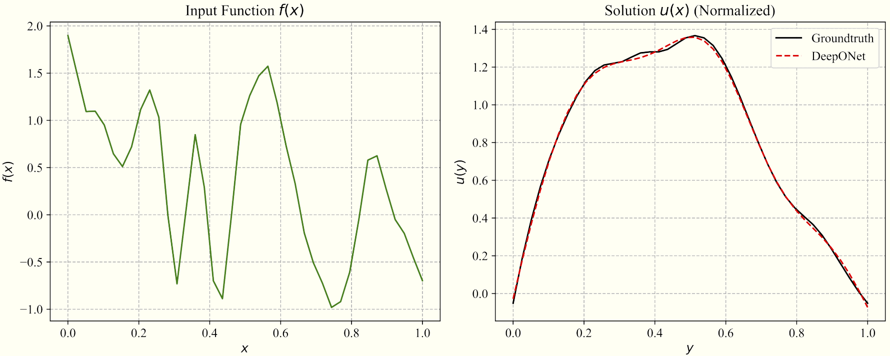
where u is the solution to the nonlinear Darcy equation for source f.

Why this is much harder than the derivative operator:

1. **Nonlinear PDE:** No analytical solution
2. **Random sources:** Infinite variety of input functions
3. **Complex physics:** Solution depends on entire source profile

Traditional approach: For each new source f, solve the PDE numerically (expensive!)
DeepONet approach: Learn the operator once, then instant evaluation for any new source

Darcy Dataset Generation

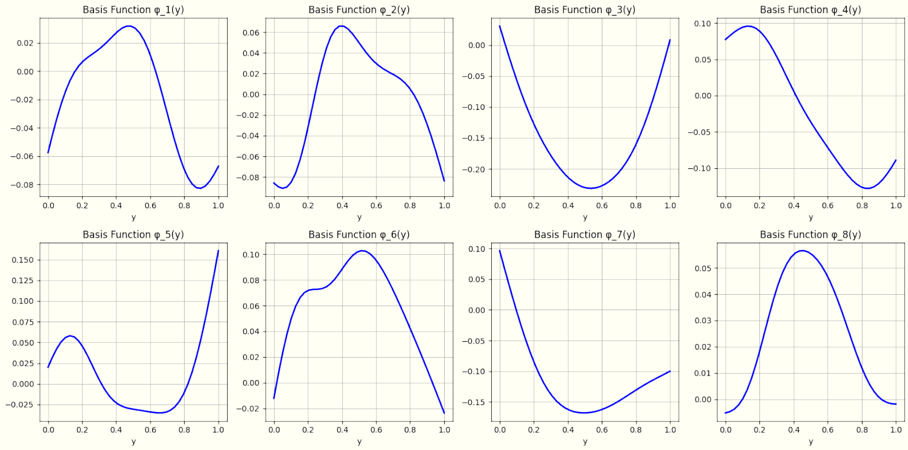


Sample Gaussian random field sources and their corresponding solutions.

Dataset characteristics:

- 1000 source functions $f(x)$ generated from a Gaussian process
- Each source solved numerically using iterative methods
- Solutions exhibit complex, nonlinear relationships to sources

Learned Basis Functions for Darcy



First 8 basis functions learned by the trunk network.

Observations:

- Basis functions capture diverse spatial patterns

Outline

The next frontier: What if we could incorporate physics directly into operator learning?

Standard DeepONet:

- Learns from data only
- Requires many training examples
- No physics knowledge
- May violate physical laws

Loss function:

$$\mathcal{L} = \|u_{pred} - u_{true}\|^2$$

Physics-Informed DeepONet:

- Incorporates governing equations
- Enforces boundary conditions
- Requires fewer training examples
- Guarantees physical consistency

Enhanced loss function:

$$\mathcal{L} = \mathcal{L}_{data} + \mathcal{L}_{physics} + \mathcal{L}_{boundary}$$

Physics-Informed DeepONet for 1D Poisson Equation

Problem: Learn the solution operator for the 1D Poisson equation:

$$\frac{d^2 u}{dx^2} = -f(x), \quad x \in [0, 1]$$

with Dirichlet boundary conditions: $u(0) = 0$, $u(1) = 0$

Physics-Informed Loss Components

1. **Data Loss:** $\mathcal{L}_{data} = \frac{1}{N} \sum_{i=1}^{N} \| u_{pred}^{(i)} - u_{true}^{(i)} \|^2$
2. **Physics Loss:** $\mathcal{L}_{physics} = \frac{1}{N} \sum_{i=1}^{N} \left\| \frac{d^2 u_{pred}^{(i)}}{dx^2} + f^{(i)} \right\|^2$
3. **Boundary Loss:** $\mathcal{L}_{boundary} = \frac{1}{N} \sum_{i=1}^{N} \left[|u_{pred}^{(i)}(0)|^2 + |u_{pred}^{(i)}(1)|^2 \right]$

▸ Open Notebook: PI-DeepONet

Key Modifications from Standard DeepONet

1. Automatic Differentiation

- Use PyTorch's autograd to compute $\frac{d^2u}{dx^2}$
- Enable gradient computation through the network
- Essential for physics loss calculation

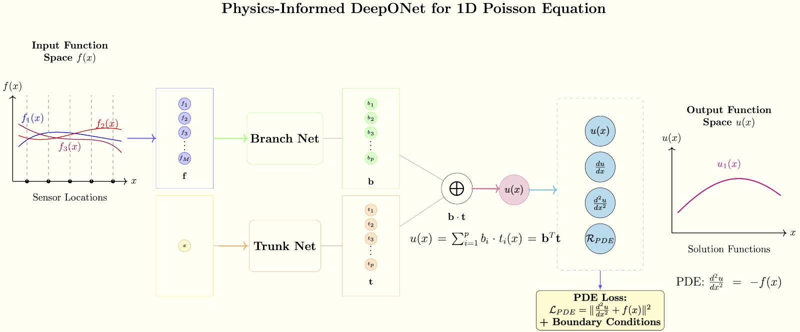
2. Multi-Objective Training

- Balance data fidelity and physics consistency
- Tunable weights: $\lambda_{physics}$, $\lambda_{boundary}$
- Monitor all loss components during training

3. Soft Constraint Enforcement

- Physics and boundary conditions as loss terms
- Flexible framework for different PDEs
- Automatically satisfied solutions

Architecture: Physics-Informed DeepONet

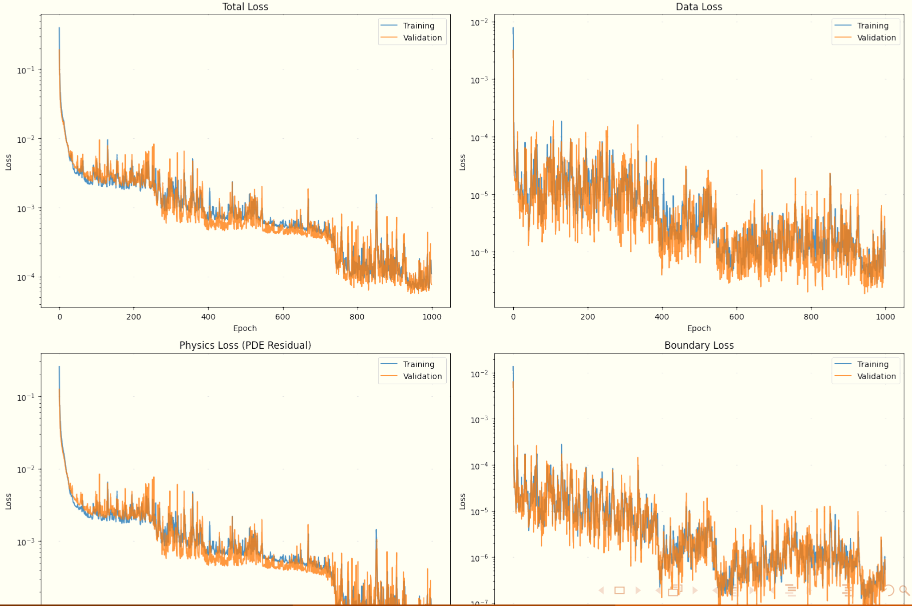


Physics-Informed DeepONet architecture with automatic differentiation.

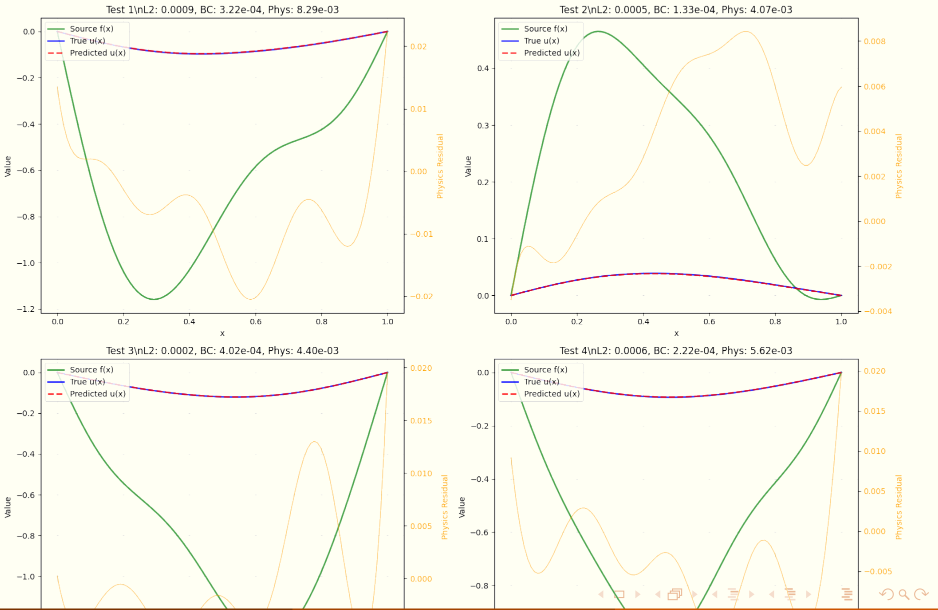
Key components:

- **Branch Network:** Processes source functions $f(x)$
- **Trunk Network:** Processes spatial coordinates x
- **Automatic Differentiation:** Computes derivatives for physics loss
- **Multi-Component Loss:** Combines data, physics, and boundary terms

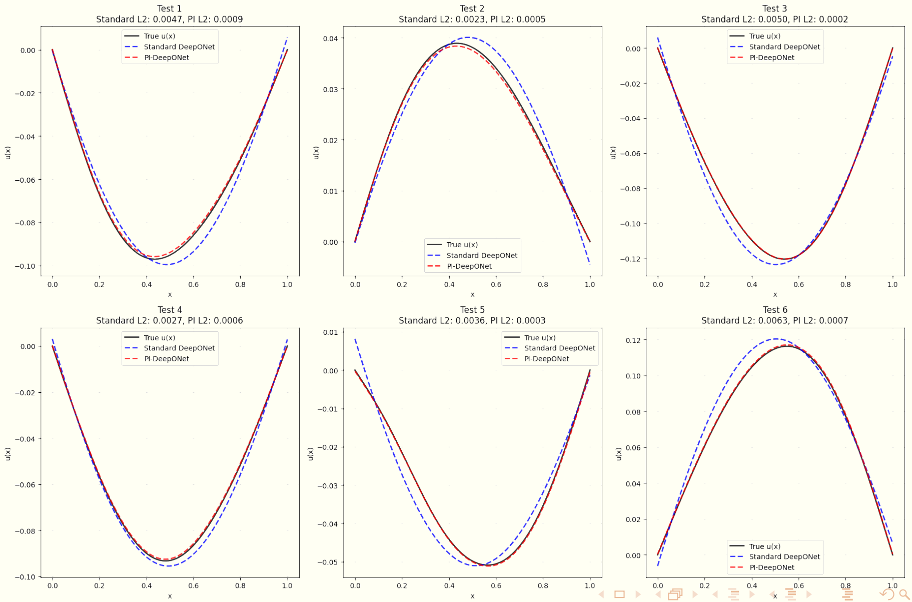
Training Dynamics: Loss Components



Physics Consistency Verification



Comparison: Standard vs Physics-Informed DeepONet



Key Advantages of Physics-Informed DeepONet

Enhanced Performance

- **Better generalization:** Physics constraints improve extrapolation
- **Guaranteed consistency:** Solutions automatically satisfy PDEs
- **Reduced data requirements:** Physics knowledge compensates for limited data

Interpretable Learning

- **Physics loss tracking:** Monitor PDE satisfaction during training
- **Boundary condition enforcement:** Explicit constraint satisfaction
- **Multi-scale learning:** Different components learn at different scales

Flexible Framework

- **Adaptable to different PDEs:** Change physics loss for new equations
- **Various boundary conditions:** Dirichlet, Neumann, Robin, periodic

Implementation Best Practices

1. Loss Balancing

- Carefully tune $\lambda_{physics}$ and $\lambda_{boundary}$ weights
- Monitor loss ratios during training
- Adjust weights based on problem complexity

2. Training Strategy

- Use sufficient collocation points for physics loss
- Enable automatic differentiation properly
- Track all loss components separately

3. Validation

- Always verify physics satisfaction on test data
- Check boundary condition compliance
- Compare with standard DeepONet baseline

Extensions and Future Directions

Immediate Extensions:

- **Different boundary conditions:** Neumann, Robin, periodic
- **Higher dimensions:** 2D/3D Poisson equations
- **Time-dependent PDEs:** Parabolic and hyperbolic equations
- **Nonlinear PDEs:** Variable coefficients and nonlinear terms

Advanced Applications:

- **Inverse problems:** Learn unknown parameters from data
- **Multi-physics:** Coupled PDE systems
- **Uncertainty quantification:** Bayesian neural operators
- **Real-time control:** Fast PDE-constrained optimization

Physics-informed DeepONet represents a powerful paradigm that combines the flexibility of neural networks with the rigor of physical laws.

Outline

What We've Learned: The Paradigm Shift

1. Conceptual Leap

- **Functions:** $\mathbb{R}^d \to \mathbb{R}^m$ (point to point)
- **Operators:** $\mathcal{F}_1 \to \mathcal{F}_2$ (function to function)

2. Theoretical Foundation

- Universal Approximation Theorem for Operators
- Branch-trunk architecture emerges naturally
- Basis function decomposition: $\mathcal{G}(u)(y) = \sum_k b_k(u) \cdot t_k(y)$

3. Practical Implementation

- **Branch network:** Encodes input functions into coefficients
- **Trunk network:** Generates basis functions at query points
- **Training:** Learn from input-output function pairs

Key Advantages of DeepONet

- **Resolution independence:** Train on one grid, evaluate on any grid
- **Fast evaluation:** Once trained, instant prediction
- **Generalization:** Works for new functions not seen during training
- **Physical consistency:** Learns the underlying operator, not just patterns

DeepONet represents a paradigm shift:

- **Traditional numerical methods:** Solve each problem instance
- **Operator learning:** Learn the solution pattern once, apply everywhere

When to Use DeepONet

Ideal scenarios:

- **Parametric PDEs:** Need solutions for many different source terms/boundary conditions
- **Real-time applications:** Require instant evaluation
- **Complex geometries:** Traditional methods struggle
- **Multi-query problems:** Same operator, many evaluations

Limitations:

- **Training data:** Need many solved examples
- **Complex operators:** Very nonlinear mappings may be challenging
- **High dimensions:** Curse of dimensionality still applies

This opens new possibilities for:

Scientific Applications:

- **Inverse problems:** Learn parameter-to-solution mappings
- **Control applications:** Real-time system response
- **Multi-physics:** Coupled operator learning
- **Scientific discovery:** Understanding operator structure

Future Directions:

- **Multi-output operators:** Vector-valued mappings
- **Higher dimensions:** 2D/3D PDEs
- **Physics-informed training:** Incorporate governing equations
- **Fourier Neural Operators:** Alternative architectures

We've seen how physics-informed DeepONet combines PINNs with operator learning!

Questions?

Thank you!

Contact:

Krishna Kumar
krishnak@utexas.edu
University of Texas at Austin

Interactive Demo:

DeepONet Notebook