Neural Implicit Functions (NIF)

Implementation and Analysis

Christian Beneke

10.02.2025

Outline

Problem Space & Motivation

- Spatio-temporal data modeled by PDEs is computationally challenging
- Current reduction methods (SVD, CAE) fail with:
  - Variable geometry
  - Adaptive meshing
- Need for a scalable, mesh-agnostic approach
- Real-time engineering applications require efficient solutions

Core Concept

- NIFs represent continuous functions through neural networks
- Combines two neural networks:
  - **ShapeNet:** Encodes spatial complexity
  - **ParameterNet:** Models temporal/parametric dependencies
- Mesh-agnostic representation
- Continuous output for any input coordinate

Architecture Overview

- HyperNetworks generate weights for target networks
- Two main architectures implemented:
  - ShortCut HyperNetwork
  - SIREN HyperNetwork
- Enables dynamic adaptation to different conditions

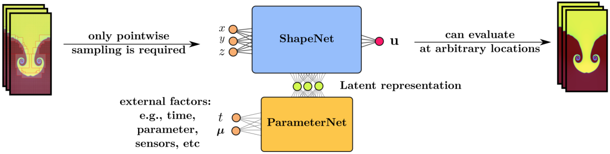


Figure: NIF Hypernetwork Architecture

Implementation Variants

Upstream & PyTorch

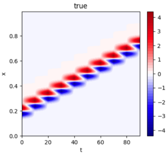
- Object-oriented design
- Native framework features
- Direct tensor operations

Functional API

- Functional programming paradigm
- Improved composability
- Clear data flow

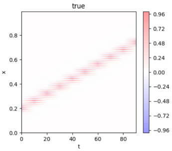
Test Cases

- Low frequency wave
  - Simple periodic function
  - Baseline for comparison



Low Frequency Wave

- High frequency complex wave
  - Tests model capacity
  - Challenges interpolation



High Frequency Wave

Network Architectures

ShortCut HyperNetwork

- Direct skip connections
- Efficient parameter usage
- Faster convergence

SIREN HyperNetwork

- Sinusoidal activations
- Better frequency fitting
- Smooth representations

Implementation Results - Upstream

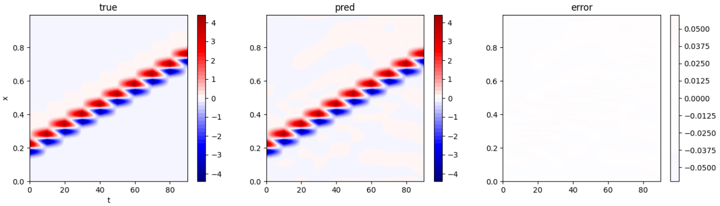


Figure: Upstream Implementation Visualization

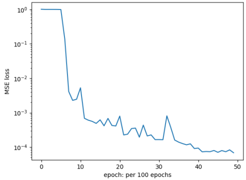


Figure: Upstream Implementation Training Loss

Implementation Results - Functional API

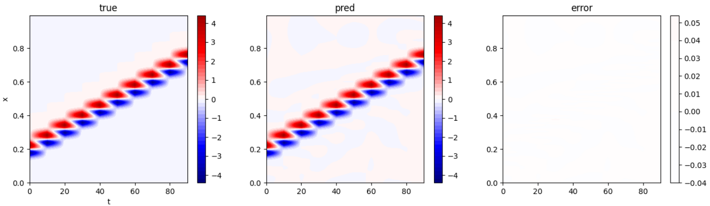


Figure: Functional API - Low Frequency Wave

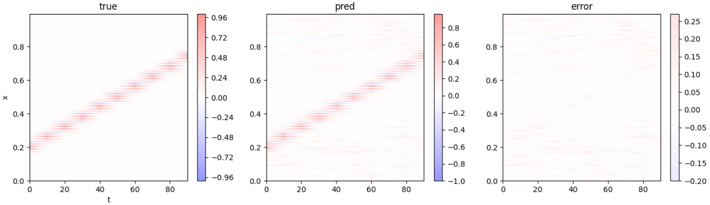
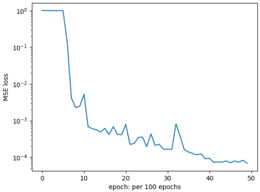


Figure: Functional API - High Frequency Wave

Training Loss Comparison



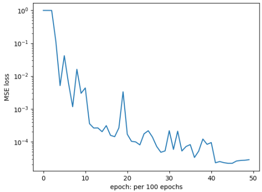


Figure: Training Loss Comparison (Upstream, Functional API)

- Upstream shows fast convergence (Best loss: 7.316e-05)
- Functional API shows 4x better performance (Best loss: 2.198e-05)

Training Loss Comparison

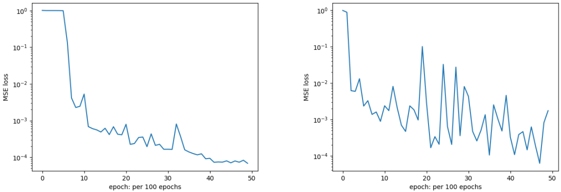


Figure: Training Loss Comparison (Upstream, PyTorch)

- Upstream shows fast convergence (Best loss: 7.316e-05)
- PyTorch shows similar performance (Best loss: 6.178e-05)
- PyTorch shows more variance in performance

Key Findings & Future Work

- Successfully implemented three variants of NIF
- Demonstrated effectiveness on both simple cases
- Key findings:
  - Framework-specific trade-offs
  - Performance characteristics
  - Implementation complexity
- Future work:
  - Additional architectures
  - More complex test cases
  - Performance optimizations

Thank you for your attention!

Questions?