Solving Inverse Problems via Physics-Informed Neural Networks (PINNs)

Kristian Boroz

December 23, 2025

Introduction to Inverse Problems

Defining the Problem Type

- While forward problems solve for a state variable u given parameters, **inverse problems** involve identifying unknown parameters (λ) or source terms using available data.
- PINNs are uniquely suited for this by treating parameters as **trainable variables** within the neural network optimization process.
- This approach leverages both the **underlying physics** (PDE) and **experimental observations** (Data) to converge on the correct solution.

The Unified Loss Function

Combining Physics and Data

- The total loss function ($\mathcal{L}$) minimized during training for an inverse problem includes:
  - **Physics Loss** ($\mathcal{L}_{PDE}$): Ensures the solution satisfies the governing equations.
  - **Data Loss** ($\mathcal{L}_U$): Forces the network to match observed data points at specific locations.
  - **Boundary/Initial Condition Loss** ($\mathcal{L}_{BC/IC}$): Maintains physical constraints at the domain edges.
- **Soft Constraints** are typically used, penalizing violations of the PDE and data via loss minimization.

Evaluating Derivatives: Jacobian and Hessian

The Engine of Automatic Differentiation (AD)

- PINNs require precise derivatives to calculate the PDE residual; DeepXDE utilizes **Jacobian** and **Hessian** functions for this purpose.
- **Jacobian:** Computes first-order derivatives (e.g., $\frac{\partial u}{\partial t}$), essential for first-order systems like coupled ODEs.
- **Hessian:** Evaluates second-order derivatives (e.g., $\frac{\partial^2 u}{\partial x^2}$), used in equations like the Burgers or Schrödinger equations.
- Using AD avoids truncation errors inherent in manual finite differences.

Handling Complex Systems

Coupled and Nonlinear PDEs

- DeepXDE can solve **coupled systems**, such as the Nonlinear Schrödinger equation, by separating real and imaginary components into distinct output neurons.
- Residuals for each equation are sent as a **list of tensors**, allowing the library to handle loss computations independently for each interacting variable.
- For time-dependent problems, a unified **space-time geometry** is created, combining spatial coordinates and temporal variables into one framework.

Model Refinement: Adam to LBFGS

Two-Stage Training Strategy

- **Stage 1: Adam Optimizer.** Used initially to handle complex loss landscapes and provide robust initial convergence.
- **Stage 2: LBFGS Optimizer.** Employed for **fine-tuning** to reach high-precision solutions.
- LBFGS dynamically determines convergence based on gradient norms and loss improvement rather than a fixed number of iterations.
- This strategy can reduce PDE and boundary losses to the order of 10^{-6} or lower.

Monitoring with Callbacks

Automating the Workflow

- **Callbacks** act as checkpoints that perform actions at predefined stages of training without interrupting the process.
- **Predictive Saving:** Save model predictions at regular intervals (e.g., every 10th epoch) to analyze how the solution evolves over time.
- **Early Stopping:** Automatically halt training if the test loss stops improving, saving computational resources.
- **Logging:** Monitor metrics like accuracy and loss in real-time to debug or optimize hyperparameters.

Validation in the Absence of Reference Data

Beyond Mean Squared Error

- In many inverse problems, a reference analytical solution is unavailable.
- Accuracy must be verified through:
  - **Residual Evaluation:** Using `model.predict(operator=pde)` to ensure the PDE residual is near zero across the domain.
  - **Physical Consistency:** Checking properties like **energy conservation** or **mass conservation**.
  - **Boundary Check:** Verifying that the learned solution strictly honors the given boundary values.

Conclusion: The Power of PINNs

Key Insights

- PINNs transform the search for physical parameters into a **multivariable optimization problem**.
- The integration of **automatic differentiation** and **sophisticated optimizers** like LBFGS allows for highly accurate inverse modeling.
- By embedding the physics directly into the neural network, the solution remains **physically consistent** even when data is sparse.

Analogy: Solving an inverse problem is like a detective using the "laws" of the crime scene (PDE) and a few "clues" (data) to reveal the hidden "culprit" (unknown parameter).

Case Study: Identifying Diffusion Coefficients

Problem Setup

- Consider the Burgers equation: $\frac{\partial u}{\partial t} + uu_x - \nu u_{xx} = 0$, where the viscosity ν is **unknown**.
- **Goal:** Use a set of sparse experimental observations $u_{obs}(x, t)$ to identify the true value of ν while simultaneously solving for the velocity field u.
- **Data Integration:** Unlike forward problems, we provide the model with a reference data set (e.g., a `.npz` file containing spatial-temporal coordinates and corresponding u values).

Defining Trainable Parameters in DeepXDE

Transforming Constants into Variables

- In DeepXDE, unknown parameters (λ) are defined as **trainable variables** within the backend (e.g., PyTorch or TensorFlow).
- **The PDE Residual:** The function is defined as:

$$f = \text{Jacobian}(u, t) + u \cdot \text{Jacobian}(u, x) - \nu \cdot \text{Hessian}(u, x)$$

  where ν is updated by the optimizer in every iteration alongside the neural network weights.
- The optimizer (Adam or LBFGS) minimizes the residual $f \to 0$ by adjusting both the solution shape and the value of ν.

Loss Function for Parameter Discovery

Balancing Physics and Observation

- The optimization relies on a multi-objective loss function:

$$\mathcal{L} = \omega_f \mathcal{L}_{PDE}(\nu) + \omega_d \mathcal{L}_{Data} + \omega_{bc} \mathcal{L}_{BC}$$

- $\mathcal{L}_{Data}$: Measures the Mean Squared Error (MSE) between the PINN prediction and the reference data set.
- $\mathcal{L}_{PDE}$: Acts as a regularizer, ensuring the identified ν remains physically consistent with the governing laws.
- This dual-constraint approach allows PINNs to find the "hidden" parameter even with **noisy data**.

Validation of Identified Parameters

Ensuring Accuracy without Ground Truth

- **Convergence Monitoring:** Use LBFGS for a second stage of training to fine-tune the parameter ν until the gradient norm is minimized.
- **Residual Check:** Even if the true ν is unknown, a correct identification must result in a PDE residual near zero (e.g., 10^{-4} to 10^{-6}).
- **Physical Properties:** For complex systems like the Schrödinger equation, check if the identified parameters maintain **energy conservation** over the time domain.

Analogy: Identifying a parameter is like tuning a musical instrument; you adjust the tension (parameter) until the sound (model) matches the reference pitch (data) while following the laws of harmonics (physics).