A Tutorial on Physics-Informed Neural Networks (PINNs)

From 1D ODEs to 2D PDEs, and the Black–Scholes Equation

Simon Scheidegger

Department of Economics, University of Lausanne; Grantham Research Institute, LSE

Wednesday, August 26th, 2025

Table of Contents

What Are Physics-Informed Neural Networks (PINNs)?

- PINNs incorporate physical laws (ODEs, PDEs, etc.) into the training of neural networks.
- Instead of (or in addition to) fitting standard labeled data, we penalize the network if it violates the governing equations.
- Use automatic differentiation (AD) to compute derivatives for the PDE or ODE residual.
- Boundary/initial conditions are enforced as part of the loss function.
- We effectively transform solving differential equations into a data-driven, neural-network–based learning problem.

General PINN Strategy

1. Define a neural network $u_\theta(x)$ (or $u_\theta(x, y)$, etc. for PDEs).
2. Use automatic differentiation to compute partial derivatives needed for the PDE/ODE.
3. Define the **residual** for the equation, e.g. $F(u_\theta, x)$.
4. Incorporate boundary or initial conditions by adding penalty terms to the loss.
5. Train (optimize) to minimize:

$$\mathcal{L}(\theta) = \underbrace{\mathrm{MSE}(F(u_\theta, x))}_{\text{Equation Residual}} + \underbrace{\mathrm{MSE}(u_\theta(\text{boundary}) - \mathrm{BC})}_{\text{Boundary Loss}} + \cdots$$

6. After training, u_θ approximates the solution.

1D ODE with Zero Boundary Conditions

Example ODE:

$$\frac{d^2y}{dx^2} = -1, \quad x \in (0, 1),$$

with **boundary conditions**

$$y(0) = 0, \quad y(1) = 0.$$

Analytical solution:

$$y(x) = -\frac{x^2}{2} + \frac{x}{2}.$$

PINN Setup:

- Define $y_\theta(x)$ as a neural net.
- PDE residual: $r(x) = y_\theta''(x) + 1$.
- Enforce boundary conditions $y_\theta(0) = 0$ and $y_\theta(1) = 0$.

Pointers to Code

Code Reference:

- `day2/Scheidegger/code/01_ODE_ZBC.ipynb` demonstrates the full PyTorch implementation.
- It constructs a small network, sets up the ODE residual, and enforces boundary conditions within the loss.
- Training is run via Adam, and a final plot compares the learned solution to the analytical solution.

1D ODE with Non-Zero Boundary Conditions

Example ODE (same interior PDE, different BCs):

$$y''(x) = -1, \quad x \in (0, 1),$$

$$y(0) = 1, \quad y(1) = 2.$$

Analytical solution:

$$y(x) = -\frac{x^2}{2} + \frac{x}{2} + 1.$$

Pointers to Code

Main Difference:

- The boundary conditions now are $y(0) = 1$ and $y(1) = 2$.
- Only the boundary part of the loss changes.

Code Reference:

- `day2/Scheidegger/code/02_ODE_NZB.ipynb` shows the minor modifications in the boundary loss terms.
- `day2/Scheidegger/code/02_ODE_NZB_HARD.ipynb` shows the modification of the Ansatz function.

Hard vs. Soft Enforcement of Boundary/Initial Conditions

Hard (exact) enforcement

- BCs satisfied *by construction* via ansatz:

$$u_\theta^{\mathrm{hard}}(x) = g(x) + B(x)N_\theta(x), \quad B|_{\partial\Omega} = 0,\ g|_{\partial\Omega} = u_{\mathrm{BC}}.$$

Example (1D, $y(0) = a$, $y(1) = b$):

$$y_\theta^{\mathrm{hard}}(x) = a(1-x) + bx + x(1-x)N_\theta(x).$$

- Pros: zero BC error; often better conditioning near $\partial\Omega$.
- Cons: need B, g design; awkward for complex/Neumann BCs; may limit expressivity.

Soft (penalty) enforcement

- Add BC penalties to loss:

$$\mathcal{L} = \mathrm{MSE}\Big(F(u_\theta)\Big) + \lambda_{\mathrm{BC}}\mathrm{MSE}\Big(u_\theta|_{\partial\Omega} - u_{\mathrm{BC}}\Big) + \lambda_{\mathrm{IC}}\mathrm{MSE}(\cdot).$$

- Pros: simple; works for varied BCs and irregular domains.
- Cons: only approximate; depends on λ and boundary sampling.

Tips: Oversample boundaries, normalize/weight losses, use augmented Lagrangian or λ-annealing for better BC accuracy.

2D Poisson: PDE, Boundary Conditions, Analytic Solution

Domain: $\Omega = (0,1)^2$, with boundary $\partial\Omega$.

$$\begin{aligned} -\Delta u(x,y) &= f(x,y), \quad (x,y) \in \Omega, \\ u(x,y) &= g(x,y), \quad (x,y) \in \partial\Omega. \end{aligned}$$

Analytic solution (used for f and g):

$$u^*(x,y) = x^2 + y + \sin(\pi x)\sin(\pi y), \qquad f(x,y) = 2 - 2\pi^2 \sin(\pi x)\sin(\pi y).$$

Dirichlet boundary values (non-zero on all sides):

$$\begin{aligned} g(0,y) &= y, & g(1,y) &= 1 + y, \\ g(x,0) &= x^2, & g(x,1) &= x^2 + 1. \end{aligned}$$

- `day2/Scheidegger/code/02_PDE_NZB.ipynb`.

Black–Scholes Recap

$$\frac{\partial V}{\partial t} + \tfrac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0.$$

$$\text{Terminal: } V(S,T) = \max(S-K,0).$$

$$\text{Boundaries: } V(0,t) = 0, \quad V(S_{\max},t) \approx S_{\max} - Ke^{-r(T-t)}.$$

Pointers to Code

Key Steps in the PINN Setup:

- PDE residual:

$$V_t + 0.5\,\sigma^2 S^2\, V_{SS} + rS\, V_S - r\, V\,.$$

- Boundary conditions at $S = 0$ and $S = S_{\max}$.
- Terminal condition at $t = T$.

Code Reference:

- `day2/Scheidegger/code/03_Black_Scholes_PINNs.ipynb` contains the full implementation.

Exercise: 1D ODE with Non-Zero Boundary Conditions

Problem (domain $x \in (0,1)$):

$$\boxed{y''(x) + y(x) = x}, \qquad \boxed{y(0) = 1, \quad y(1) = 0.3}$$

This is a second-order, linear, non-homogeneous ODE with *non-zero* Dirichlet boundary conditions.

Tasks:

1. Implement a PINN with *soft* boundary enforcement (penalty): minimize $\mathrm{MSE}\big(y''_\theta + y_\theta - x\big)$ on $(0,1)$ + $\lambda\,\mathrm{MSE}(\{y_\theta(0) - 1,\ y_\theta(1) - 0.3\})$.
2. Implement a PINN with *hard* boundary enforcement (lifting): use $y_\theta(x) = A(x) + B(x)N_\theta(x)$ where $A(0) = 1,\ A(1) = 0.3$ and $B(0) = B(1) = 0$, train only on $\mathrm{MSE}\big(y''_\theta + y_\theta - x\big)$.
3. Compare solutions vs. $y^*(x)$ on a dense grid: report relative L^2 errors, boundary errors, and show plots.

Hints (optional):

- A simple choice: $A(x) = (1 - 0.3)(1 - x) + 0.3x = 1 - 0.7x, \qquad B(x) = x(1 - x)$.
- Solution: `04_Solution.ipynb`.

Summary & Next Steps

- We introduced PINNs: enforcing PDE/ODE constraints by building them into the loss function.
- Showed ODEs, PDEs, boundary conditions, and terminal conditions.
- Demonstrated examples: 1D boundary value problems, 2D Black–Scholes.
- In practice, you can adapt these templates to more complex PDEs, domains, or multi-dimensional states.

Some References

- M. Raissi, P. Perdikaris, and G. E. Karniadakis, Physics-Informed Neural Networks: A Deep Learning Framework (2019).
- J. Berg and K. Nystroem, A Unified Deep Artificial Neural Network Approach to PDEs in Complex Geometries (2017).
- For HJB PDE references in reinforcement-learning contexts, see: D. Jiang, F. Meng, Q. Sun, X. Xue, and Y. Zou. DeepRitz Method (2019).

Thank You!

Questions?