Physics-Informed Neural Networks

Physics-Informed Neural Networks

Foundations: ODE vs PDE

Definition

An **ODE** (ordinary differential equation) relates a function to derivatives in one independent variable (often time).
A **PDE** (partial differential equation) relates a multivariable field to partial derivatives in space and/or time.

$$\text{ODE example: } \frac{dy}{dt} = g(t, y), \qquad \text{PDE example: } \frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2}$$

Interpretation: ODEs model lumped dynamics; PDEs model distributed dynamics that vary across space and time.

- ODE state is low-dimensional (for example, position/velocity of one body).
- PDE state is a field $u(x, t)$ over a domain.

Why Physics Uses Differential Equations

- Core laws are local rate laws: momentum, energy, charge, and mass are written as change rates.
- Differential equations encode **causality**: current state plus forcing determines future state.
- Space-dependent media require PDEs to represent transport, waves, and diffusion.
- Initial/boundary conditions map directly to experimental setup and geometry.
- Once calibrated, the model supports prediction, control design, and inverse parameter estimation.

$$\text{Physics pipeline: laws + IC/BC} \Rightarrow \text{well-posed ODE/PDE model}$$

Interpretation: PINNs are useful because they learn solutions while enforcing this same physics structure.

Raissi et al. (2019): What It Established

- Physics-informed neural networks as deep networks constrained by PDE structure.
- Two core problem classes: data-driven solution of PDEs and data-driven discovery of PDE parameters.
- Two algorithmic families: continuous-time PINNs and discrete-time PINNs with Runge-Kutta structure.

Preliminaries: General Problem Setup

$$u_t + \square[u; \lambda] = 0, \qquad x \in \Omega,\ t \in [0, T]$$

Interpretation: learn either the hidden field $u(t, x)$, the PDE parameters λ, or both, from sparse/noisy observations.

- u: latent state approximated by a neural network.
- $\square[\cdot; \lambda]$: nonlinear differential operator.
- Inference task: given λ, estimate u.
- Discovery task: estimate λ from measurements and physics constraints.

Preliminaries: Continuous-Time PINN

$$f(t,x) := u_t + \square[u]$$

$$\mathrm{MSE} = \mathrm{MSE}_u + \mathrm{MSE}_f$$

Interpretation: MSE_u *fits observed IC/BC/data points, while* MSE_f *enforces PDE physics at collocation points.*

- $u_\theta(t,x)$ and $f_\theta(t,x)$ share parameters via automatic differentiation.
- Collocation points are sampled in the interior domain.
- Key small-data idea: physics loss acts as a strong regularizer.

Runge-Kutta: Core Idea

For an ODE $y'(t) = f(t, y)$, advance one step h from t_n to t_{n+1} using multiple slope samples:

$$y_{n+1} = y_n + h \sum_{j=1}^{q} b_j \, k_j, \qquad k_j = f\left(t_n + c_j h, \; y_n + h \sum_{m=1}^{q} a_{jm} \, k_m \right)$$

Interpretation: Runge-Kutta builds the next state from a weighted combination of intermediate slope evaluations inside one time step.

- Euler is the $q = 1$ special case (one slope only).
- RK4 is a common explicit choice with $q = 4$.
- Implicit high-order RK can remain stable at larger Δt, which motivates discrete-time PINNs.

Preliminaries: Discrete-Time PINN via Runge-Kutta

$$u^{n+c_i} = u^n - \Delta t \sum_{j=1}^{q} a_{ij} \square[u^{n+c_j}], \qquad u^{n+1} = u^n - \Delta t \sum_{j=1}^{q} b_j \square[u^{n+c_j}]$$

Interpretation: the network predicts all Runge-Kutta stage states and the next-time state in one structured objective.

- Multi-output network represents stage variables $u^{n+c_1}, \ldots, u^{n+c_q}$ and/or u^{n+1}.
- Enables very large time steps when high-order implicit RK is used.

Why This Topic Matters

- PINNs combine governing equations with data, enabling forward and inverse modeling in one framework.
- Core motivation: in small-data regimes, prior physics regularizes learning and improves generalization.
- Known PINN weaknesses: imbalance, long-time drift, and boundary sensitivity.

Definition: PINN

Definition

A **Physics-Informed Neural Network (PINN)** is a neural network u_θ trained to fit data and satisfy a governing differential equation through residual penalties computed with automatic differentiation.

PINN Loss Decomposition

$$\mathrm{MSE} = \mathrm{MSE}_u + \mathrm{MSE}_f$$

$$\mathrm{MSE}_u = \frac{1}{N_u}\sum_{i=1}^{N_u} |u(t_i^u, x_i^u) - u_i|^2, \qquad \mathrm{MSE}_f = \frac{1}{N_f}\sum_{i=1}^{N_f} |f(t_i^f, x_i^f)|^2$$

This is the original continuous-time objective

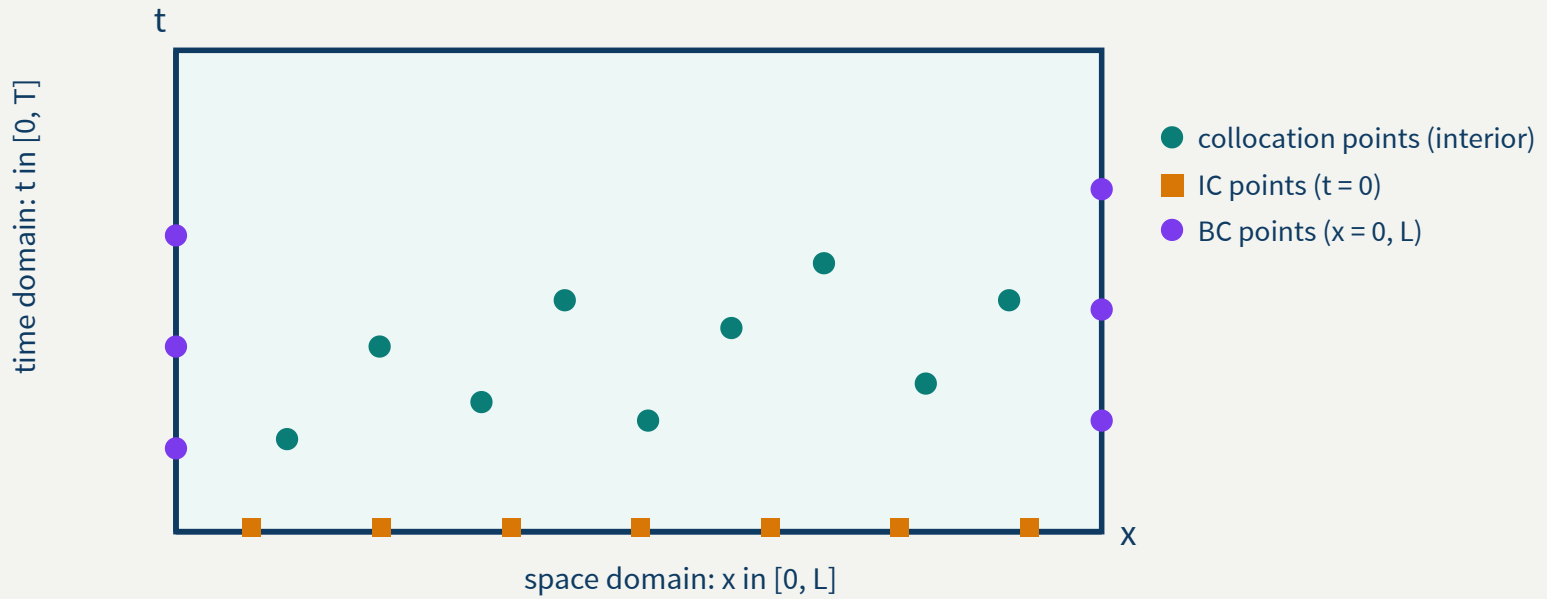
- MSE_u: observed initial/boundary/data points.
- MSE_f : PDE residual at interior collocation points.
- Practice expands this into separate IC and BC penalties with task-specific weights.
- Weight choices strongly influence whether all constraints are satisfied.

Collocation Points

Definition

Collocation points are interior points (x_i^f, t_i^f) where no direct target value is given; the model is trained by forcing the PDE residual $r_\theta(x_i^f, t_i^f)$ toward zero.

$$\square_{\text{PDE}} = \frac{1}{N_f} \sum_{i=1}^{N_f} |r_\theta(x_i^f, t_i^f)|^2$$



- Teal interior points contribute to $\square_{\text{PDE}}$.
- Orange and purple points enforce IC/BC constraints in separate loss terms.

Full PINN Loss Decomposition (IC/BC + Tunable Weights)

$$\square(\theta) = \lambda_r \square_{\text{res}} + \sum_{k=1}^{K_{ic}} \lambda_{ic,k} \square_{\text{IC},k} + \sum_{\ell=1}^{K_{bc}} \lambda_{bc,\ell} \square_{\text{BC},\ell} + \lambda_d \square_{\text{data}}$$

$$\square_{\text{res}} = \frac{1}{N_r} \sum_{i=1}^{N_r} |r_\theta(\mathbf{x}_i^r, t_i^r)|^2, \qquad r_\theta(\mathbf{x}, t) = \square[u_\theta](\mathbf{x}, t)$$

$$\square_{\text{IC},k} = \frac{1}{N_{ic,k}} \sum_{i=1}^{N_{ic,k}} |\square_k[u_\theta](\mathbf{x}_i, 0) - g_k(\mathbf{x}_i)|^2$$

$$\square_{\text{BC},\ell} = \frac{1}{N_{bc,\ell}} \sum_{i=1}^{N_{bc,\ell}} |\square_\ell[u_\theta](\mathbf{x}_i^b, t_i^b) - b_\ell(\mathbf{x}_i^b, t_i^b)|^2$$

$$\square_{\text{data}} = \frac{1}{N_d} \sum_{i=1}^{N_d} |u_\theta(\mathbf{x}_i^d, t_i^d) - u_i^d|^2 \qquad \text{(optional supervised points)}$$

Interpretation: weights λ_* tune the trade-off between physics residual, each IC constraint, each BC constraint, and optional data fit.

Notation

- $u_\theta(\mathbf{x}, t)$: PINN approximation of the unknown state; θ are trainable network parameters.
- $\mathbf{x} \in \Omega \subset \mathbb{R}^d$, $t \in [0, T]$: spatial and temporal coordinates.
- $\square[\cdot]$: governing differential operator (ODE/PDE); homogeneous form is $\square[u] = 0$.
- $\square_k[\cdot] = g_k$: the k-th initial-condition operator/target pair.
- $\square_\ell[\cdot] = b_\ell$: the ℓ-th boundary-condition operator/target pair.
- $N_r, N_{ic,k}, N_{bc,\ell}, N_d$: numbers of residual, IC, BC, and supervised data points.
- $\lambda_r, \lambda_{ic,k}, \lambda_{bc,\ell}, \lambda_d \geq 0$: tunable loss weights.

- Common BC operators: Dirichlet (value), Neumann/Robin (flux/mixed), periodic (value and derivative matching).
- Practical start: initialize all $\lambda_* = 1$, then rebalance with gradient-norm or adaptive weighting.

Baseline PINN Workflow

- Baseline is simple and (hopefully) reproducible, but not always stable on long-time runs.
- Keep separate diagnostics for PDE, IC, BC and data losses during training.

What Breaks in Practice

Failure mode	Typical symptom in tasks	Practical impact
Gradient imbalance	BC/IC satisfied but interior residual remains high	Physically inconsistent trajectories
Spectral bias	Missed sharp or oscillatory components	Wrong wave-speed/phase behavior
Long-time drift	Error grows with larger T	Unreliable forecasting
Boundary sensitivity	Instability near edges or periodic joins	Large local artifacts
Sampling brittleness	Training depends heavily on point placement	Unstable reproducibility

Hyperparameter Findings

Axis	Reported finding
Optimizer	**Adam + L-BFGS-B** best
Activation	**tanh** best, ReLU worst
Network size	About **5 hidden layers** and **100 nodes/layer**
Learning rate	**1e-3**
Training points	accuracy improves with denser training sets in tested range