



Physics-Informed Neural Networks (PINNs)

Scientific ML 01 | The PINN loss (residual, initial/boundary, data terms), collocation and autodiff derivatives, a residual computed by hand, PyTorch code, inverse problems, failure modes and their fixes, variants, and when a classical solver wins (as of 2026)

#SciML #PINN #PDE #InverseProblems #PyTorch

The idea

A student checked by substitution	Physics-informed training
The student proposes a formula for $u(t)$	The network $u_\theta(t)$ is the candidate solution
The teacher plugs it into the equation at a few points	The residual is evaluated at collocation points
Marks lost for every point where it does not balance	Squared residuals form the physics loss
The starting value must also match	Initial and boundary penalty terms
A few measured values given as hints	Optional data loss from observations
WHERE THE ANALOGY BREAKS A teacher checks the final formula; a PINN only ever sees finitely many points, so it can satisfy them and still be wrong in between or converge to a wrong, smooth solution.	

A PINN is graded by the equation, not by an answer key

Before Training a PINN, can you explain...

	Not yet	Partly	Yes
MUST KNOW ODEs, initial-value problems Learn: maths 19	☐	☐	☐
MUST KNOW Partial derivatives, chain rule Learn: maths 05-06	☐	☐	☐
MUST KNOW Autograd and training loop Learn: programming (PyTorch)	☐	☐	☐
MUST KNOW MLP with smooth activations Learn: deep-learning 01	☐	☐	☐
HELPFUL Adam, L-BFGS Learn: deep-learning 03, maths 17	☐	☐	☐
ADVANCED Finite elements or differences Learn: a numerical-PDE text	☐	☐	☐

Readiness; each item recalled in one line where used

Recall (maths 19): a solver marches an ODE forward in steps; a neural ODE learns the right-hand side. A PINN is different: the network **is** the solution $u_\theta(x, t)$, a smooth function of its inputs, and automatic differentiation gives its exact derivatives $\partial u / \partial t$, $\partial^2 u / \partial x^2$ at any point, with no mesh. The idea dates to Lagaris, Likas and Fotiadis (1998); the 2019 PINN paper (J. Comput. Phys.) made it popular with modern autodiff.

The loss

Term	Formula
Residual (physics)	$\mathcal{L}_r = \frac{1}{N_r} \sum_{i=1}^{N_r} \mathcal{N}[u_\theta](x_i, t_i) ^2$
Initial and boundary	$\mathcal{L}_b = \frac{1}{N_b} \sum_{j=1}^{N_b} u_\theta(x_j, t_j) - g(x_j, t_j) ^2$
Data (optional)	$\mathcal{L}_d = \frac{1}{N_d} \sum_{k=1}^{N_d} u_\theta(x_k, t_k) - y_k ^2$
Total	$\mathcal{L}(\theta) = w_r \mathcal{L}_r + w_b \mathcal{L}_b + w_d \mathcal{L}_d$

where u_θ = network output: candidate solution with weights θ (field unit); $\mathcal{N}[u]$ = differential operator, e.g. $u_t + k \cdot u$ or $u_t - \nu \cdot u_{xx}$ (field unit per time); x_i, t_i = collocation points sampled inside the domain (space, time); g = prescribed initial or boundary values (field unit); y_k = measured values at observation points (field unit); N_r, N_b, N_d = numbers of residual, boundary and data points (count); w_r, w_b, w_d = loss weights (fixed or adapted) (-)

PINN loss for a PDE $\mathcal{N}[u] = 0$ on a domain Ω with boundary $\partial\Omega$ and an initial state

PROBLEM Score the trial function $u(t) = 1 - 2t + 1.5t^2$ as a PINN would, at collocation points $t = 0, 0.5, 1$.

GIVEN

- Operator $N[u] = du/dt + k \cdot u$ with $k = 2$ (decay rate, $1/s$); t in s
- Trial derivative $du/dt = -2 + 3t$

1 **t = 0**

$$u = 1, du/dt = -2, r = -2 + 2 \cdot 1 = 0$$

Why: the trial matches the slope at the start

2 **t = 0.5**

$$u = 1 - 1 + 0.375 = 0.375, du/dt = -0.5, r = -0.5 + 0.75 = 0.25$$

Why: exact $u(0.5) = 0.368$, close

3 **t = 1**

$$u = 0.5, du/dt = 1, r = 1 + 1 = 2$$

Why: the parabola turns upward: physics violated

4 **Residual loss**

$$L_r = (0^2 + 0.25^2 + 2^2) / 3 = 4.0625 / 3 = 1.354$$

Why: mean squared residual

5 **Initial-condition loss**

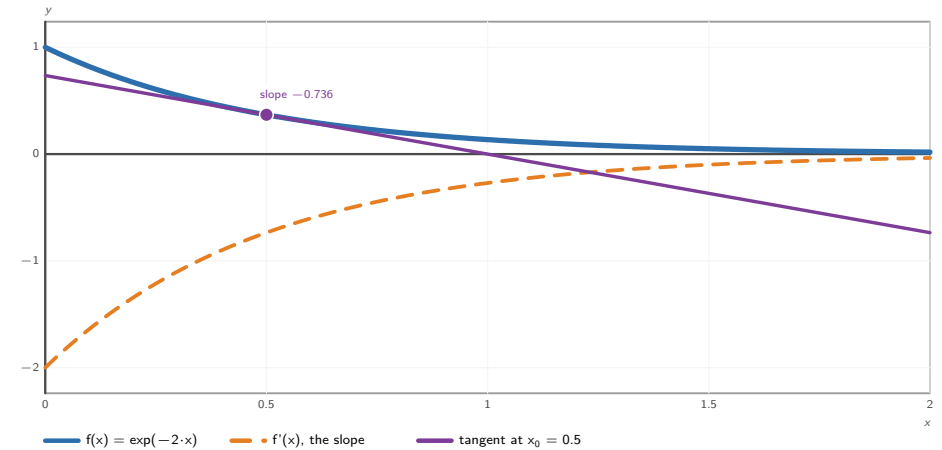
$$L_b = (u(0) - 1)^2 = 0$$

Why: $u(0) = 1$ holds exactly

ANSWER $L_r \approx 1.354$, $L_b = 0$: the optimiser will bend the curve down at large t

CHECK ☒ Plug in the exact solution $u = e^{-2t}$: $du/dt = -2e^{-2t}$, so $r = -2e^{-2t} + 2e^{-2t} = 0$ at every point

Residual of a trial solution by hand: the ODE $du/dt = -k \cdot u$, $u(0) = 1$, $k = 2$ (exact $u = e^{-2t}$)



$f(x) = \exp(-2 \cdot x)$

No root on $[0, 2]$: f never crosses 0 there

$f'(x) \approx [f(x+h) - f(x-h)] / (2h)$, $h = 2 \times 10^{-5}$ (central difference)

Tangent at $x_0 = 0.5$: $y = f(x_0) + f'(x_0) \cdot (x - x_0) = 0.3679 + (-0.7358) \cdot (x - 0.5)$

Where: x : input value; $f(x)$: output; $f'(x)$: derivative = slope, change of f per unit of x ; h : small step of the difference quotient; x_0 : point of tangency

The exact solution $u(t) = e^{-2t}$: the tangent at $t = 0.5$ has slope $-0.736 = -k \cdot u(0.5)$, which is exactly what the residual checks at that point

Total PINN loss with weights (the same trial solution plus one observation)

FORMULA $L = w_r \times R + w_i \times I + w_d \times D$

WHERE

- L = total loss minimised by the optimiser (-)
- R = mean squared ODE residual at the collocation points (computed above) $((1/s)^2)$
- I = squared initial-condition error $(u(0) - 1)^2$ (-)
- D = squared data error at $t = 1$: $(0.5 - 0.135)^2$, the measured value being $e^{-2} \approx 0.135$ (-)
- w_r = weight of the residual term (-)
- w_i = weight of the initial-condition term (-)
- w_d = weight of the data term (-)

WORKED

$$\begin{aligned} w_r &= 1 \\ R &= 1.354 \\ w_i &= 100 \\ I &= 0 \\ w_d &= 10 \\ D &= 0.133 \\ L &= 1 \times 1.354 + 100 \times 0 + 10 \times 0.133 \\ L &= 2.684 \end{aligned}$$

RESULT $L = 2.684$

READ IT

The data term contributes about half of the loss here; with $w_d = 1000$ it would dominate and the optimiser would fit the observation at the expense of the physics. The weights are hyperparameters with large effects.

MISTAKES

- Leaving all weights at 1 when terms differ by orders of magnitude
- Comparing losses across runs with different weights

🔗 Code and inverse problems

```
A minimal PINN for  $du/dt = -k \cdot u$ ,  $u(0) = 1$  (PyTorch)
import torch, torch.nn as nn
k = 2.0
net = nn.Sequential(nn.Linear(1, 32), nn.Tanh(),
                    nn.Linear(32, 32), nn.Tanh(),
                    nn.Linear(32, 1))
opt = torch.optim.Adam(net.parameters(), lr=1e-3)
t0 = torch.zeros(1, 1)
for step in range(5000):
    t = torch.rand(128, 1, requires_grad=True) # collocation
    u = net(t)
    du = torch.autograd.grad(u, t, torch.ones_like(u),
                             create_graph=True)[0]
    loss_r = ((du + k * u) ** 2).mean() # ODE residual
    loss_i = ((net(t0) - 1.0) ** 2).mean() #  $u(0) = 1$ 
    loss = loss_r + 100.0 * loss_i
    opt.zero_grad(); loss.backward(); opt.step()
```

👩 Make the constraint hard, and the inverse problem one line

Write the output as $u(t) = 1 + t \cdot N_{\theta}(t)$: $u(0) = 1$ for any weights, so the initial-condition term disappears. For an **inverse problem**, declare the unknown coefficient trainable, $k = nn.Parameter(torch.tensor(0.5))$, add it to the optimiser and add a data term on the observations: the same loss now recovers k . This is where PINNs are most useful, e.g. inferring velocity and pressure fields from dye-concentration videos (Raissi, Yazdani and Karniadakis, Science 2020).

1. Non-dimensionalise

scale x, t, u to order 1; unscaled PDEs are the first cause of failure

2. Hard-code BC/IC

output transform where possible

3. Sample

random or quasi-random collocation; add points where the residual is large (RAR)

4. Balance

adaptive loss weights; causal weights in time

5. Optimise

Adam, then L-BFGS to polish

6. Verify

error against a reference solver, not only the loss

A practical PINN recipe (DeepXDE and NVIDIA PhysicsNeMo, formerly Modulus, automate most steps, as of 2026)

⚠ When PINNs fail and what fixes them

Symptom	Cause	Remedy
Smooth but wrong solution in convection or reaction problems with large coefficients	Hard optimisation landscape, not lack of capacity (Krishnapriyan et al., NeurIPS 2021)	Curriculum on the coefficient, sequence-to-sequence in time
Boundary fitted, interior ignored (or the reverse)	Loss-term gradients differ by orders of magnitude (Wang, Teng and Perdikaris, 2021)	Learning-rate-annealing or NTK-based adaptive weights
Late times solved before early times	Residual minimised everywhere at once	Causal weights in time (Wang, Sankaran and Perdikaris, 2024)
High-frequency or multi-scale features missing	Spectral bias of MLPs	Fourier feature embeddings, domain decomposition (XPINN, FBPINN)

Shocks and discontinuities smeared

A smooth network cannot represent a jump

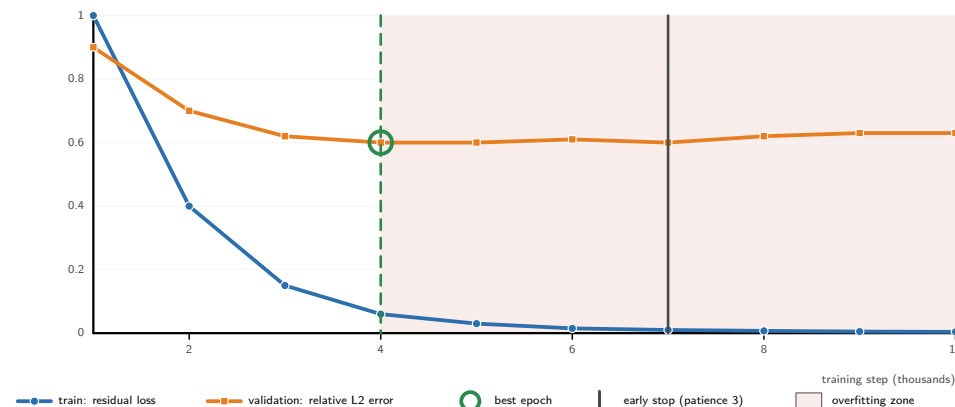
Weak forms (VPINN), adaptive points, or a classical shock-capturing solver

Training takes hours for one solution

One network per PDE instance

Use a neural operator for many instances (sheet 02)

Documented failure modes and remedies (papers and years given for the source)



Diagnosis	computed from the loss values
Best epoch	4: validation: relative L2 error = 0.6
Generalisation gap	at best: 0.54 · at the last common epoch: 0.626
Early stopping, patience 3	stops at epoch 7 (no gain for 3 epochs), restores epoch 4
Overfitting onset	after epoch 4: validation: relative L2 error worsens while train: residual loss still improves
Spikes	none
Plateaus	none
Verdict	Overfitting after epoch 4: keep those weights (early stopping), add regularisation or data.

```
gap(t) = L_val(t) - L_train(t)
spike at t: e(t) > k · median(e(t - w) ... e(t - 1)), k = 1.5, w = 5
plateau: |e(t) - e(t - 1)| ≤ ε · e(t - 1) for ≥ m epochs, ε = 0.01, m = 5
early stopping: stop at the first t with t - t* ≥ p, p = 3
```

where

t	epoch (the first value is epoch 1)	ε	relative change counted as flat
e(t)	loss at epoch t	m	shortest plateau (epochs)
gap	how much worse validation is than training (≥ 0 usually)	t*	best epoch so far of the monitored curve
k	spike factor	p	patience (epochs without gain)
w	window of previous epochs		

Illustrative numbers showing the pattern of the convection failure: the residual loss falls 250-fold while the relative error against the reference solver stalls near 60 %. A falling loss proves nothing; always monitor an error against a reference

⚠ PINNs vs classical solvers on forward problems

For a single forward solve of a low-dimensional PDE, a tuned finite-element or finite-volume solver is usually faster and more accurate: Grossmann et al. (IMA J. Appl. Math., 2024) found finite elements ahead in both time and accuracy on the problems they tested. PINNs earn their place on **inverse problems with sparse data**, on **irregular or high-dimensional domains** where meshing is the bottleneck, and when one network must be **differentiable with respect to inputs**. Report the baseline's wall-clock time, not only yours (sheet 08).

≡ Relatives of the PINN

Deep Ritz (E and Yu, 2018) minimises an energy functional; the **Deep Galerkin Method** (Sirignano and Spiliopoulos, 2018) targets high-dimensional PDEs; **VPINN** / **hp-VPINN** use the weak form; **B-PINN** (Yang, Meng and Karniadakis, 2021) puts a Bayesian posterior on the weights for uncertainty; **PINO** adds the residual loss to a neural operator (sheet 02).

🚩 Myths and recap

YOU MAY HEAR

"PINNs need no data"

BUSTED

✓ ACTUALLY

They need no labelled solution data, but they need the equation, its coefficients and boundary data

Without data or a complete problem statement the solution is not unique.

YOU MAY HEAR

"PINNs are mesh-free, so they scale to any geometry for free"

BUSTED

✓ ACTUALLY

Collocation points still must cover the domain

Sampling and training cost grow with domain complexity and frequency content.

YOU MAY HEAR

"A small residual loss means an accurate solution"

BUSTED

✓ ACTUALLY

The residual is measured only at sampled points and trades off with other terms

Check against a reference solution or a manufactured solution.

YOU MAY HEAR

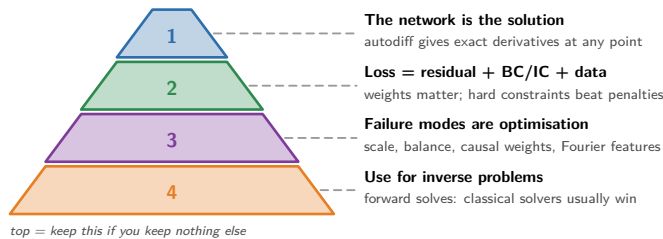
"Bigger networks fix PINN failures"

BUSTED

✓ ACTUALLY

Most failures are optimisation problems

Scaling, weighting, causal training and curriculum help more than width.



Q1 Residual of $u = 1 - 2t + 1.5t^2$ for $du/dt = -2u$ at $t = 1$?

A $du/dt = 1$, $u = 0.5$, $r = 1 + 2 \cdot 0.5 = 2$.

Q2 How does a PINN get derivatives of its output?

A Automatic differentiation of the network with respect to its inputs (x, t) , with `create_graph=True` so the loss can be differentiated again.

Q3 Soft vs hard constraint?

A Soft: a penalty term in the loss, approximately satisfied. Hard: built into the output form, exactly satisfied for any weights.

Q4 Hard-constrained form for $u(0) = 1$?

A $u(t) = 1 + t \cdot N_{\theta}(t)$.

Q5 How do you solve an inverse problem with a PINN?

A Make the unknown coefficient a trainable parameter and add a data term on the observations.

Q6 Why do PINNs miss high-frequency features?

A Spectral bias: gradient descent fits low frequencies first; Fourier feature embeddings help.

Q7 Fix when boundary and residual terms fight?

A Adaptive loss weights (learning-rate annealing or NTK-based), and non-dimensionalised variables.

Q8 What is causal training?

A Down-weighting later-time residuals until earlier times are solved, so the solution propagates forward in time.

Q9 Paper that named PINNs?

A Raissi, Perdikaris and Karniadakis, Journal of Computational Physics, 2019.