Inverse PINNs

Tutorial 4

PINNs for inverse problems: recovering the conductivity of the heat equation from noisy measurements.

Bogdan Raonic · Oct 12, 2026 · Seminar for Applied Mathematics (D-MATH) & ETH AI Center

Outline

01 From forward to inverse problems

02 Implementation in PyTorch

03 Results and common mistakes

04 Practice

01

From forward to inverse

Forward and inverse problems

Same PDE, different unknowns. In the inverse problem the conductivity k is unknown and we have measurements of u instead.

Forward · Tutorial 3

$u_t = u_{xx}$

Known: the PDE coefficients, boundary and initial condition

Unknown: the solution u

One network u_θ.

Inverse · today

$u_t = k(x)\, u_{xx} + s$

Known: source s, boundary and initial condition, noisy measurements of u

Unknown: the solution u and the conductivity k

Two networks: u_θ for the solution, k_ϕ for the conductivity.

The model problem

PDE

$$u_t = k(x)\, u_{xx} + s(t, x)$$

$t \in [0, 0.1]$, $x \in [-1, 1]$

BC, IC

$$u(t, \pm 1) = 0,\ u(0, x) = -\sin(\pi x)$$

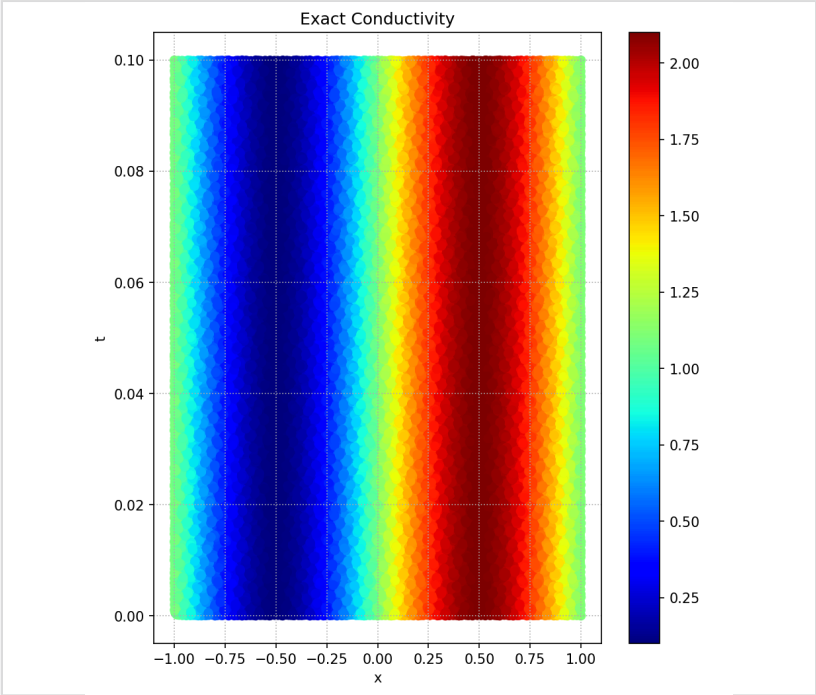
DATA

$$u^{\text{meas}} = u_{\text{ex}} + 0.01\,\xi,\ \xi \sim N(0, 1)$$

TRUTH

$$k(x) = \sin(\pi x) + 1.1$$

Used only to build s and the data, and to measure the error



Inverse PINNs: from two networks to one loss

Goal: find θ and ϕ so that u_θ matches the measurements and u_θ, k_ϕ together satisfy the PDE, the boundary and the initial condition.

01 · Networks

$(t, x) \mapsto u_\theta, k_\phi$

Same input, two separate MLPs: one guesses the temperature, one the conductivity.

02 · Autograd

$\partial_t u_\theta, \partial_{xx} u_\theta$

As in Tutorial 3. Only u is differentiated; k_ϕ enters the PDE as a factor.

03 · Residuals

PDE, BC, IC + data

The PDE residual now contains k_ϕ. A new residual compares u_θ with the measurements.

E.g. $r = u_\theta(t_i, x_i) - u_i^{\text{meas}}$

at each sensor reading

04 · Loss

$L(\theta, \phi)$

Mean squared residuals, minimized over both θ and ϕ at the same time.

Residuals and loss

Interior

$$r_{\text{int}} = \partial_t u_\theta - k_\phi \, \partial_{xx} u_\theta - s$$

on S_{int}: the only term that depends on ϕ

Boundary, initial

$$r_{\text{sb}} = u_\theta(t, \pm 1), \; r_{\text{tb}} = u_\theta(0, x) + \sin(\pi x)$$

on S_{sb} and S_{tb}, as in Tutorial 3

Measurements

$$r_{\text{meas}} = u_\theta(t_i, x_i) - u_i^{\text{meas}}$$

on S_{meas}: the sensor readings, new in the inverse problem

Loss

$$L(\theta, \phi) = L_{\text{int}} + \lambda_u (L_{\text{sb}} + L_{\text{tb}} + L_{\text{meas}})$$

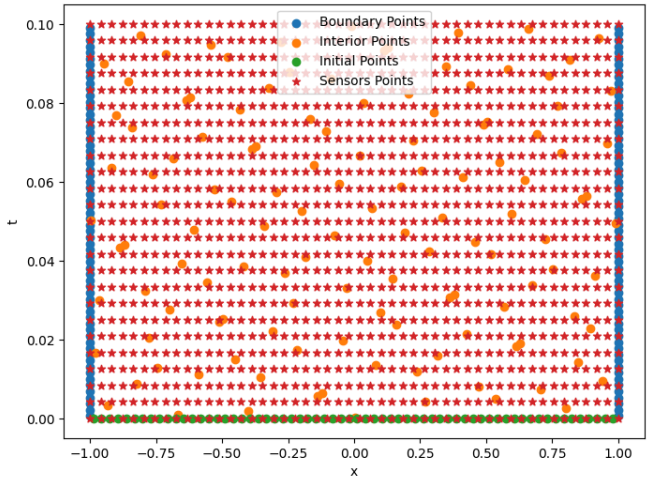
each L is the mean squared residual on its set

02

Implementation

Training points and sensors

Collocation points from a Sobol sequence, as in Tutorial 3. The sensors sit on a regular grid.



128

interior points, for the PDE residual

2 × 64 + 64

boundary and initial points

25 × 50

measurements: 50 sensor positions, 25 readings in time

Measurements: synthetic sensor data

```
def get_measurement_data(self):
    # 25 time steps x 50 sensor positions
    t = torch.linspace(0, 0.1, 25)
    x = torch.linspace(-1, 1, self.n_sensor)
    input_meas = torch.cartesian_prod(t, x)

    u = exact_solution(input_meas).reshape(-1, 1)
    noise = 0.01 * torch.randn_like(u)
    return input_meas, u + noise
```

Synthetic data

Taken from the exact solution. In a real experiment these would be thermometer readings.

cartesian_prod

All (t, x) pairs: 25 × 50 = 1 250 points.

Noise

Gaussian, standard deviation 0.01, about 1 % of the largest |u|.

PDE residual: k enters as a factor

```
def compute_pde_residual(self, input_int):
    input_int.requires_grad = True
    u = self.approximate_solution(input_int)
    k = self.approximate_coefficient(input_int)
    grad_u = torch.autograd.grad(
        u.sum(), input_int, create_graph=True)[0]
    grad_u_t = grad_u[:, 0]
    grad_u_x = grad_u[:, 1]
    grad_u_xx = torch.autograd.grad(
        grad_u_x.sum(), input_int,
        create_graph=True)[0][:, 1]
    s = source(input_int)
    return grad_u_t - k.reshape(-1) * grad_u_xx - s
```

Second network

k_ϕ is evaluated at the same interior points as u_θ.

Same autograd

The two grad calls are unchanged from Tutorial 3. No derivative of k is needed.

Known source

s is computed from the exact u and k, so that u_{ex} solves the PDE.

Loss: one extra data term

```
r_int = self.compute_pde_residual(inp_train_int)
r_sb = u_train_sb - u_pred_sb
r_tb = u_train_tb - u_pred_tb
r_meas = u_train_meas - u_pred_meas

loss_sb = torch.mean(abs(r_sb) ** 2)
loss_tb = torch.mean(abs(r_tb) ** 2)
loss_int = torch.mean(abs(r_int) ** 2)
loss_meas = torch.mean(abs(r_meas) ** 2)

loss_u = loss_sb + loss_tb + loss_meas
loss = torch.log10(
    self.lambda_u * loss_u + loss_int)
```

Measurement residual

Same form as the boundary residual, but at sensor points inside the domain.

$\lambda_u = 10$

All data terms are weighted together against the PDE residual.

How k is learned

k_ϕ appears only in loss_int. The data fix u; the PDE then fixes k.

Optimizer: train both networks

```
net_u = pinn.approximate_solution
net_k = pinn.approximate_coefficient
params = (list(net_u.parameters())
          + list(net_k.parameters()))

optimizer_LBFGS = optim.LBFGS(
    params,
    lr=0.5,
    max_iter=1000,
    max_eval=50000,
    history_size=150,
    line_search_fn="strong_wolfe")
```

One optimizer

θ and ϕ are updated together on the same loss.

Check the Adam line

In the notebook, Adam gets only net_u's parameters. With it, k_ϕ never changes.

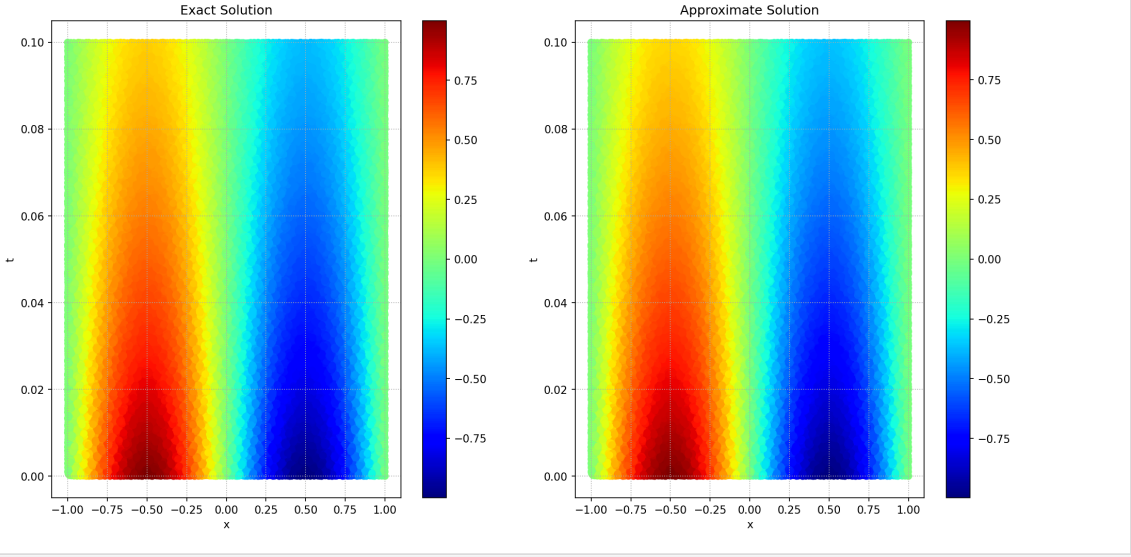
L-BFGS

Same closure and settings as Tutorial 3, with max_iter = 1000.

03

Results

Result: the solution u

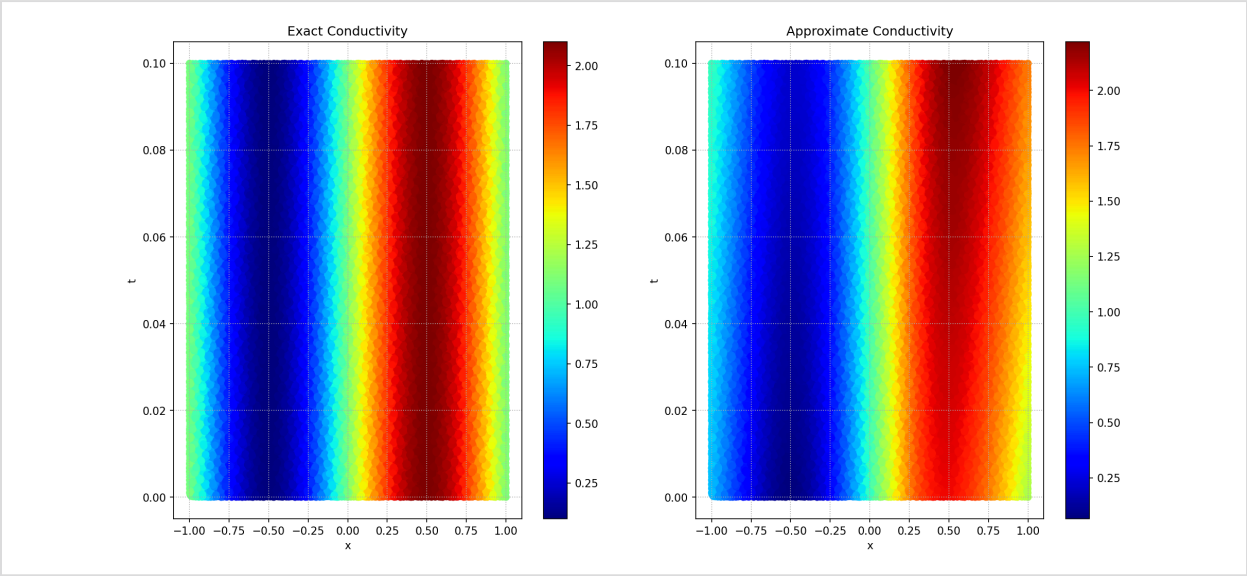


1.1%

relative L2 error for u, on 100 000 Sobol points

Close to the forward problem, even though k was unknown and the data are noisy.

Result: the conductivity k



10.1%

relative L2 error for k

k is never measured. It is recovered from the PDE and the measurements of u.

Common mistakes: inverse PINNs

WHAT YOU SEE	CAUSE	FIX
k_ϕ does not change during training	Its parameters are not in the optimizer	Pass both parameter lists
Loss is low, but k is wrong	Too little data: many (u, k) pairs fit the PDE	Add sensors or readings
k is off near x = 0 and x = ±1	$u_{xx} \approx 0$ there, so the PDE says little about k	Expected; judge k away from these points

04

Practice

Practice: the inverse heat problem on Colab

- Check link on the Moodle.
- Run the notebook as is: about 1 % error for u and 10 % for k.
- Increase the noise level and watch the error in k.
- Use fewer sensors, or switch to Adam with both parameter lists.