Solving Inverse Problems Using Physics-Informed Neural Network

Alia Yusaini Conrad Struss Zhaoming Li Zeyuan Liu Zhenhao Guo

December 4, 2024

Abstract

Physics-Informed Neural Networks (PINNs) integrate model equations, such as partial differential equations (PDEs), directly into the neural network architecture. This innovative approach is used to solve various types of equations, including PDEs, fractional equations, integral-differential equations, and stochastic PDEs. PINNs operate within a multi-task learning framework, where the neural network fits observed data while minimizing the residuals of the PDE. In this paper, we implemented a forward neural network-based method for solving the inverse problem for our heat equation, $u_t - a\Delta u = 0$, while also finding the solution of the PDE.

1 Introduction

In this project, we're focusing on recovering the variable a in the heat equation, $u_t - a\Delta u = 0$ when $x \in [-1, 1]$ and $t \in [0, 1]$. Physics-Informed Neural Network (PINN) has been introduced by Raissi et. al. [1] to solve the forward and inverse problem for partial differential equations (PDE). PINN incorporated the physics part of the PDE by letting the function be $f(x, t)$ defined by

$$f(t, x) := u_t - a\Delta u$$

and approximated $u(x, t)$ by a deep neural network. We aim to recover the connection between $f(x, t)$ and $u(x, t)$ using 2 losses,

$$MSE = MSE_u + MSE_f$$

$$MSE_u := \frac{1}{N}\sum_{i=1}^{N}(u_{\text{pred}} - u_{\text{true}})^2$$

$$MSE_f := \frac{1}{N}\sum_{i=1}^{N}(f(x, t))^2$$

Unlike the original paper, this project provides the implementation of PINN on recovering the diffusion coefficient using Tensorflow v2 instead of Tensorflow v1.

2 Related work

Recent studies have demonstrated that deep learning is a promising method for creating metamodels for fast predictions of dynamic systems. Neural networks (NNs) are particularly effective at representing the nonlinear input-output relationships in complex systems. However, these high-dimensional systems are still subject to the curse of dimensionality, a concept first described by Bellman in 1966 in the context of optimal control problems (see [5]). Despite this challenge, machine learning-based algorithms show great potential for solving partial differential equations (PDEs). Blechschmidt and Ernst suggest that machine learning-based approaches to PDE solutions will remain a significant area of study as deep learning continues to advance in methodology, theory, and algorithms (see [6]). Early work, such as that by Lagaris et al. [7], used simple neural network models like multi-layer perceptrons (MLPs) with a few hidden layers to solve differential equations (see [8]). Modern methods leverage optimization frameworks and auto-differentiation, as seen in the unified deep neural network technique proposed by Berg and Nyström for estimating PDE solutions (see [9]). Currently, there is no standardized terminology in the literature for integrating prior knowledge of physical phenomena with deep learning. Terms such as 'physics informed,' 'physics-based,'

'physics-guided,' and 'theory-guided' are commonly used. Kim et al. (2021b) developed a comprehensive taxonomy for what they termed 'informed deep learning,' accompanied by a literature review in the field of dynamical systems. Their taxonomy is organized into three conceptual stages: (i) the type of deep neural network used, (ii) the representation of physical knowledge, and (iii) the integration of physical information (see [11]).

Although we're only working on recovering one variable in our inverse problem, if we were working on recovering a, b, and c in the equation $u_t - au_{xx} - bu_x - cu = 0$ instead, there might be some more convergence issues that would turn up. Some works that are focused on solving convergence issues include a robust Gated PINN project, where Lee et. al. fixed the problem by partitioning the domains into smaller sections to get better convergence [2].

3 Formulation

We address the solution and parameter inference of the one-dimensional heat equation, a second-order partial differential equation (PDE) given by:

$$\frac{\partial u}{\partial t} = a\frac{\partial^2 u}{\partial x^2}, \quad x \in [-1, 1],\ t \in [0, 1]$$

With the initial and boundary conditions:

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

Using the Fourier transform, the solution is given by:

$$u(x, t) = \sin(\pi x)e^{-a\pi^2 t}$$

where $u(x, t)$ represents the temperature distribution over space and time and a is the thermal diffusivity parameter.

4 Method Description

Our methodology involves constructing a PINN to solve two tasks:

1. Forward Problem: Predict $u(x, t)$ given a known a.
2. Inverse Problem: Infer a from observed data $u(x, t)$.

We design a fully connected neural network to approximate $u(x, t)$, leveraging its smooth and continuous nature. The structure is made up of:

- Hidden Layers: 5 dense layers with 50 neurons each and the hyperbolic tangent ($\tanh$) activation function.
- Output Layer: A single neuron with a linear activation for predicting $u(x, t)$.

The neural network's predictions must satisfy the governing PDE. Using automatic differentiation in TensorFlow, we compute the necessary derivatives:

$$\text{Residual} = \frac{\partial u}{\partial t} - a\frac{\partial^2 u}{\partial x^2}.$$

This residual serves as a physics-based regularizer in the loss function, ensuring that the model predictions adhere to the physical laws.
To train the PINN, we minimize a composite loss function combining data-driven and physics-based terms:

- Forward Problem: The loss is computed as the mean squared error (MSE) between the predicted and true solutions:

$$L = \frac{1}{N}\sum_{i=1}^{N}(u_{\text{pred}} - u_{\text{true}})^2.$$

- Inverse Problem: The loss includes an additional term penalizing deviations from the PDE residual:

$$L = \frac{1}{N}\sum_{i=1}^{N}(u_{\text{pred}} - u_{\text{true}})^2 + \frac{1}{M}\sum_{j=1}^{M}(\text{Residual})^2.$$

Firstly, we need to process our training data. We need to generate our own training data; therefore, the training data for x and t is sampled uniformly from the domain $[0, 1]$. The exact solution $u(x, t) = \sin(\pi x)e^{-a\pi^2 t}$ is computed for these inputs to serve as ground truth. Then the Adam optimizer is employed with a learning rate of 0.001. For the inverse problem, the parameter a is optimized alongside the neural network weights. For each epoch, gradients of the loss with respect to model parameters are computed and applied. The model is trained for 5000 epochs, with periodic reporting of the loss and, in the inverse problem, the learned parameter a. Secondly, we need to do testing and evaluation. After training, the model is evaluated on a test grid of x and t values. For the forward problem part, the predicted solution $u_{\text{pred}}(x, t)$ is compared to the exact solution $u_{\text{true}}(x, t)$. And for the inverse problem part, the learned parameter a is compared to the true value, and the predicted solutions are validated against the exact solutions. Finally, we need to visualize our true solution vs. the solution we found. Here, we use 3D to visualize. 3D surface plots to get a comprehensive view of the predicted and true solutions over the domain.

5 Results

5.1 Forward Problem without PINN loss

Epoch	Loss
0	0.19866541028022766
500	0.00013473595026880503
1000	2.4055207177298144e-05
1500	8.6979940533638e-05
2000	7.593285772600211e-06
2500	4.15438898926368e-06
3000	2.9897391868871637e-06
3500	4.953811185259838e-06
4000	1.8199642681793193e-06
4500	1.82122266778606e-06

Table 1: Loss result for each epoch

The result shows that at the start of training, the loss is relatively high (0.1987), indicating that the model is far from the correct solution, as expected when initialized with random weights. Between Epoch 0 and Epoch 500, the loss drops significantly to 0.000135, a decrease of several orders of magnitude. This implies that the underlying pattern in the data is rapidly discovered by the model. After Epoch 2000, the loss decreases very slowly and stabilizes around 10^{-6} to 10^{-7}. This indicates that the model has reached an optimal solution within numerical precision limits, and further training offers diminishing returns. The loss exhibits slight oscillations (e.g., at Epochs 1500 and 3500). These tiny oscillations are probably caused by the Adam optimizer's stochastic nature as well as the intricacy of the function being approximated. This model performs exceptionally well. The final loss (1.8×10^{-6}) is very small, indicating that the model predicts the target solution $u(x, t)$ with high accuracy on the training data. Without incorporating the PINN loss, the model relies entirely on fitting the training data. This approach can achieve a good fit as the training data directly maps $(x, t) \to u_{\text{true}}$. Therefore, the model demonstrates excellent convergence for the forward problem without PINN loss, achieving high accuracy on the training data. However, the absence of physics-informed constraints raises concerns about its robustness and generalization.

5.2 Inverse Problem

Epoch	Loss	a
0	0.25602567195892334	0.4990001618862152
500	0.0012559713795781136	0.09890186041593552
1000	0.00020690172095783055	0.09958001971244812
1500	5.6673932704143226e-05	0.10022115707397461
2000	0.00646325433626771	0.1003895103931427
2500	2.7723774110199884e-05	0.10019935667514801
3000	1.204957879963331e-05	0.1002221331000328
3500	8.751060704526026e-0	0.10020679235458374
4000	6.906848739163252e-06	0.10019223392009735
4500	1.3816452337778173e-05	0.10006316751241684

Table 2: Inverse Part Result

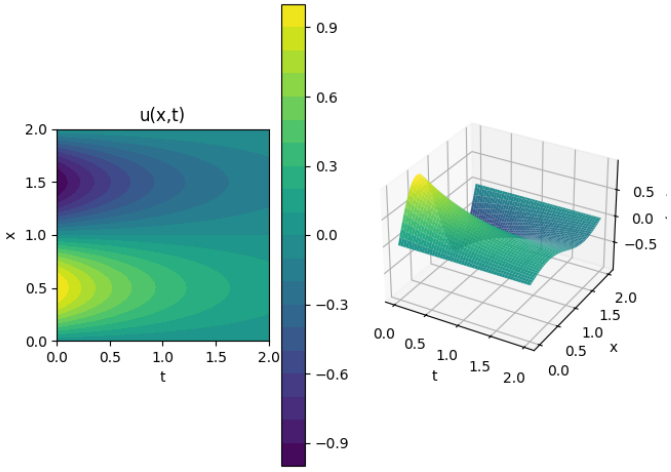


Figure 1: plot of $3D(x, t, u_{exact})$

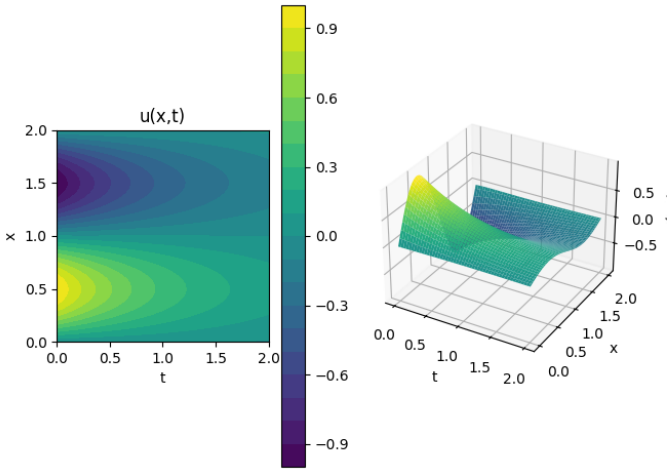


Figure 2: plot of $3D(x, t, u_{pred})$

The initial loss is relatively high: 0.2560, indicating the model's deviation from the exact solution at initialization. The parameter a is initialized at 0.4990, which is close to the specified initial value of 0.5. At this stage, the model has no prior knowledge of the underlying physics or solution. During the Epoch 500–1500, we know that the loss decreases rapidly, reaching 0.0013 at Epoch 500 and 5.67×10^{-5} by Epoch 1500. The parameter a approaches the true value ($a_{\text{true}} = 0.1$) during this phase. At Epoch 1500, $a \approx 0.1002$. This demonstrates that the model effectively learns both the parameter and the solution from data and physics-informed constraints. Slight loss fluctuations are observed, such as at Epoch 2000 (0.0065). These are likely due to optimizer dynamics or exploration of local minima. Despite these variations, the parameter a stabilizes around 0.1 after Epoch 2000, with only minor adjustments. By Epoch 4500, we have $a = 0.1001$, which is very close to the true value ($a_{\text{true}} = 0.1$). During the Epoch 4500: The final loss is exceptionally small (1.38×10^{-5}), indicating high accuracy in fitting the data and satisfying the physics-informed constraints. And the learned parameter a (0.1001) closely matches the true value ($a_{\text{true}} = 0.1$), validating the model's performance in solving the inverse problem. Apart from it, observing the 3D plot, we know that the prediction and the exact result have analog figures, which also convince us of the model's excellent performance. Therefore, we conclude that the inclusion of the PDE residual in the loss ensures that the solution respects the governing equations, enabling robust estimation of both $u(x,t)$ and a, and the model estimates the parameter a with high precision (< 0.001 error), demonstrating the strength of the physics-informed approach for inverse problems. While minor loss fluctuations occur, the model's performance validates the effectiveness of the PINN framework.

5.3 Model Robustness check

Adding noise to the model involves introducing randomness, which can simulate real-world imperfections or measurement errors. We introduce some noise to our model's outputs. Gaussian noise $\mathcal{N}(0, \sigma^2)$ was added to the target temperature values $u_{\text{true}}(x,t)$. The noisy inputs are defined as:

$$u(x,t)_{\text{noisy}} = u(x,t) + \epsilon$$

where ϵ is random perturbations sampled from a normal distribution. Different standard deviations stand for different magnitudes of noise. We define σ = 0.01 as small noise. σ = 0.1 is large noise

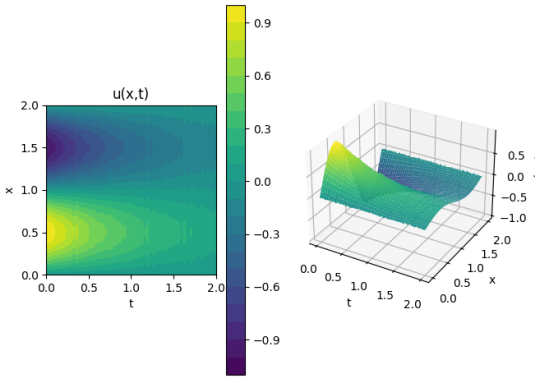
Attempts	Loss	loss(small noise)	loss(large noise)
First attempt	3.20263325193082e-06	7.05e-05	0.001643214
Second attempt	3.46e-05	8.67e-05	0.000272094
Third attempt	4.41e-06	9.63e-05	0.001793897

Table 3: Robustness check result in forward problem

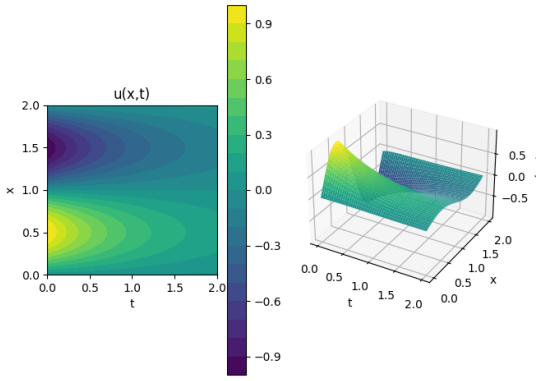
Attempts	a	a(small noise)	a(large noise)
First attempt	0.099727459	0.100368127	0.091405839
Second attempt	0.100144103	0.10061232	0.093132533
Third attempt	0.099997103	0.100184955	0.100494124

Table 4: Robustness check result in inverse problem

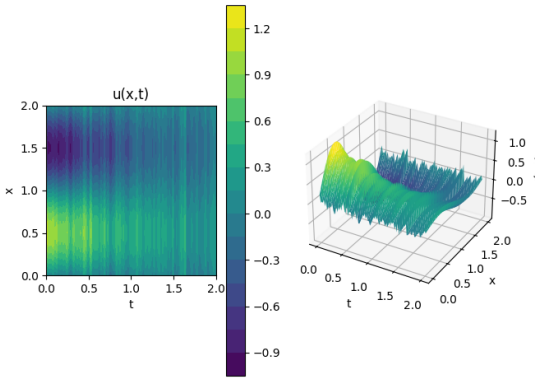
For the forward problem, the loss is not robust against noise. However, once we check the parameter prediction robustness, we'll find the parameter predicted by the inverse problem demonstrates strong resilience against random noise. The parameter remained between [0.09, 0.1] regardless of the noise added. Also, the robustness of loss and parameter prediction against attempts is significant, as depicted in the robustness checks in the forward problem and inverse problem. The possible reason is that the model has the ability to ignore information caused by randomness according to the performance on the test distribution. Although the noise is added, the pattern is captured by the model according to the test performance.



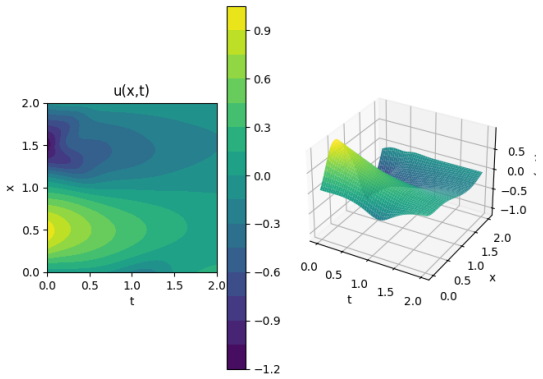
(a) test distribution with small noise



(b) model performance with small noise



(c) test distribution with large noise



(d) model performance with large noise

Figure 3: Plot of $3D(x, t, u_{exact})$ and $3D(x, t, u_{pred})$ in robustness check

References

[1] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686-707, 2019. 1

[2] S. Lee, B.-T. Lee, and S. K. Ko, "A robust Gated-PINN to resolve local minima issues in solving differential algebraic equations," *Results in Engineering*, vol. 21, pp. 101931, 2024. 2

[3] Y. Aytar, L. Castrejon, C. Vondrick, H. Pirsiavash, and A. Torralba. Cross-modal scene networks. *PAMI*, 2016.

[4] K. Bousmalis, N. Silberman, D. Dohan, D. Erhan, and D. Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *CVPR*, 2017.

[5] R. Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966. DOI: 10.1126/science.153.3731.34. Available at: https://pubmed.ncbi.nlm.nih.gov/17730601/. 1

[6] Patrick Blechschmidt and Oliver Ernst. Three ways to solve partial differential equations with neural networks – A review. *arXiv preprint arXiv:2102.11802*, 2021. Available at: https://arxiv.org/abs/2102.11802. 1

[7] I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 987-1000, 1998. 1

[8] Isaac E. Lagaris, Aristidis Likas, and Dimitrios I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *arXiv preprint arXiv:physics/9705023*, 1998. Available at: https://arxiv.org/abs/physics/9705023. 1

[9] Jan S. Hesthaven, Stefan U. R. H. Berg, and J. Nyström. A unified deep neural network approach to solving partial differential equations. *arXiv preprint arXiv:1711.06464*, 2018. Available at: https://arxiv.org/pdf/1711.06464. 1

[10] C. Irrgang, N. Boers, M. Sonnewald, E. A. Barnes, C. Kadow, J. Staneva, and J. Saynisch-Wagner, "Towards neural Earth system modelling by integrating artificial intelligence in Earth system science," *Nature Machine Intelligence*, vol. 3, no. 8, pp. 667-674, 2021.

[11] Byeong-Ho Kim, Soo-Hyun Byun, Rina Lee, Jung-Hoon Lee, Donghyeon Kim, Byung-Jun Yoon, Hongseok Yang, and Sungroh Yoon. Informed deep learning for computational and dynamical systems. *arXiv preprint arXiv:2101.09126*, 2021. Available at: https://arxiv.org/pdf/2101.09126. 2