From Plateaus to Progress: Unveiling Training Dynamics of PINNs

Rahil Pandya
Department of Computing
Imperial College London
London, United Kingdom
rrp18@@ic.ac.uk

Panos Parpas
Department of Computing
Imperial College London
London, United Kingdom
panos.parpas@imperial.ac.uk

Daniel Lengyel
Department of Computing
Imperial College London
London, United Kingdom
dl2119@ic.ac.uk

Abstract

Physics Informed Neural Networks (PINNs) promise performance gains in solving partial differential equations (PDEs) related to diverse applications. Yet, their training can be challenging, attributed in part to their unique loss function components. This study examines the optimization trajectory of PINNs for the heat equation, comparing it to a similarly-architected model trained on a numerical PDE solution. Our findings suggest that PINNs experience prolonged plateaus and unstable training behaviors predominantly due to poor and highly varied interactions between individual components of the PINN loss. While a complete understanding of training-dynamics remains open, we hope to shed more light on this problem.

1 Introduction

Physics Informed Neural Networks (PINNs) have gained prominence across diverse applications, spanning fields from fluid flow [3, 9, 13, 16, 14], diffusion processes [4], engineering [15] and medicine [20]. By combining data-driven learning with our understanding of physical laws, PINNs emerge as an exciting tool in physical sciences.

The defining feature of PINNs is the loss function [19]. The loss has two components: the *differential loss* which penalizes not satisfying the differential equation, and the *boundary loss* which biases the network to have correct initial and boundary condition. The training of this model is accomplished via automatic differentiation tools [18, 2, 1] and using continuous activation functions.

While PINNs hold great potential for solving PDEs, their training can be challenging in practice [11]. Adaptive sampling methods [6, 7, 16] and innovative architectures [8, 5] have been explored to address these challenges. Yet, difficulties persist, with their root causes remaining unclear. Current work on dynamics between differential and boundary loss [11] largely focuses on the loss landscape, not the optimization trajectory. Insights about the disproportionate impact of exploding gradients on differential loss [21] provide some clarity but don't paint a full picture [11].

To study the behavior of PINNs, we perform a comparative study to a network trained to fit a numerical solution, which we call a *Regression Model*. For this study, we let the architecture of the two networks be equivalent, with the only difference arising from the objective with respect to which they are trained. While in practical applications, we do not have numerical solutions available, by comparing the behavior between the networks, we hope to better understand failure modes of PINNs.

1.1 Contributions

Against this background we make the following observations

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

- PINNs remain in plateaus significantly longer than a network trained to fit a numerical solution—reflecting the difficulty in training PINNs. We additionally find that the update-step sizes of both methods are comparable during the plateau, indicating that the length of the plateau is not solely attributable to smaller steps or different learning rates.
- During the plateau, the PINN is displaying both update-steps which are frequently orthogonal, and an average distance travelled per update-step that is lower when compared to the Regression Model. Since update-steps that are not aligned often lead to an optimization trajectory with high variance, we may attribute the slower progress to such behavior. As the distance travelled by the PINN is also slightly larger, a lower distance travelled per step will lead to a prolonged plateau.
- Our main observation is that the gradient of the two key components of the PINN loss, a differential and boundary loss, are on average in opposition during the plateau training period and a bit more aligned after. During the plateau period such a behavior is problematic, as in conjunction with the exploding gradient phenomenon of the differential loss, this may lead to the observed misalignment of consecutive update-steps.

1.2 Notation

Let $\phi(x \in \mathbb{R}^d, t \in \mathbb{R}_+; \theta \in \mathbb{R}^M) \to y \in \mathbb{R}$ be a **neural network**. Let $A \subseteq \mathbb{R}^d$, then ∂A is the boundary of A and $\bar{A}$ is the closure. $|.|$ denotes the absolute value function. Δ is the Laplace operator $\Delta = \sum_{i=1}^{d} \frac{\partial^2}{\partial {x_i}^2}$. The $\times$ symbol is used for the Cartesian product of two sets. We call the optimization trajectory the set $\{\theta_i | 0 \leq i < N\}$ obtained during optimizing a network for N steps.

2 Background

Heat Equation We focus on the heat equation on a rectangular domain $\bar{\Omega} \subseteq \mathbb{R}^2$ and time interval $[0, T]$. We let $f(x)$ be the initial condition and $g(x)$ be the boundary condition. The exact formulation of the physical process is given in Appendix A. We selected the heat equation as a representative of parabolic and diffusion-type equations. Given the diverse qualitative behaviors of processes represented by differential equations, it is impractical to study them all simultaneously. Focusing on a specific process, like the heat equation, offers a more targeted approach.

PINN Loss. Let the sets of collocation points be given by $S \subseteq \Omega \times (0, T]$, $S_{init} \subseteq \Omega$ and $\partial S \subseteq \partial\Omega \times [0, T]$. We define the PINN loss as for our problem as $\mathcal{L}_{PINN}(\theta) := \mathcal{L}_{Diff} + \lambda_b \mathcal{L}_{Boundary}$, where the differential loss is given by $\mathcal{L}_{Diff}(\theta) := \sum_{(x,t)\in S} |\Delta\phi(x, t; \theta) - \partial_t \phi(x, t; \theta)|^2$ and $\mathcal{L}_{Boundary}(\theta) := \sum_{(x,t)\in\partial S} |\phi(x, t; \theta) - g(x)|^2 + \sum_{x\in S_{init}} |\phi(x, 0; \theta) - f(x)|^2$ provides the loss based on the boundary and initial conditions. The λ_b parameter adjusts the weighting of the differential and boundary losses.

Regression Loss. The regression loss requires a numerical solution to the PDE problem, which we describe in Appendix C. We then construct a dataset $\mathcal{D} \subseteq \bar{\Omega} \times [0, T] \times \mathbb{R}$ and define the regression loss to be $\mathcal{L}_{Regression}(\theta) := \sum_{(x,t,y)\in\mathcal{D}} |\phi(x, t; \theta) - y|^2$. The model trained to minimize the regression loss is then called the Regression Model.

3 Methodology

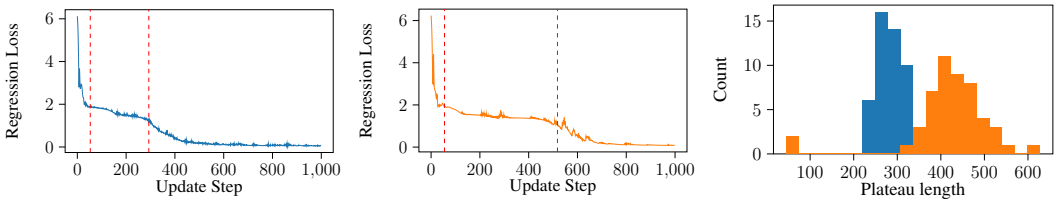
Training and Evaluation. For each model we use Adam [10] as our optimization method and complete 1000 update-steps, irrespective of batch-size, to make the optimization procedure more comparable. The PINN and Regression Model collocation sets are obtained via the same grid discretisation of the domain. In addition to a grid-discretisation, the Regression Loss requires ground truth values for the solution to the PDE. We obtain these via standard numerical methods for solving PDEs. The details of the above construction are provided in Appenidx B and C respectively. For our study, we evaluate each model on both losses. For each loss type, we use the respective training set as the evaluation set. As we are interested in the approximation rather than generalisation power of each model, we chose a fine discretisation of the domain for training and such an evaluation set is appropriate.

Network and Hyperparameter selection. We used the same architecture and learning rate for both the PINN and Regression Model. One difference being that a full-batch is used for the PINN and a batch of size 512 for the Regression Model. The other difference being that we varied the λ_b parameter for the PINN loss. We chose these values after performing a hyperparameter search over 2000 combinations as demonstrated in Appendix D. We then ranked the networks according to their Regression loss on the hold out set and found the best performing parameters for both models to be nearly identical. To compare the methods we reran the best performing network architecture and learning rate on fifty different seeds. While the same learning rate is used, it remains important to investigate the true update-step sizes for each method as the objective function may be differently conditioned. We emphasise that due to an equivalent architecture, both methods have the same representation power and the same model bias–inductive bias is solely due to the objective function.

Plateau Definition. Intuitively a plateau is a period during training where the loss curve does not significantly change. Due to noise, this is difficult to make precise. So, we define the plateau to be the outcome of the following procedure. We smooth the curve twice with an exponential moving average, with a span of ten and twenty respectively–the method for this is given by *pandas* [17]. This smooths the curve without displacing it much. We let the plateau be the first contiguous region within which the loss does not decrease by more than 0.15 over 20 steps. For visual clarity we mark the plateau region with red-dashed lines.

Comparison Methodology. Our comparison focuses on evaluating various metrics and presenting summary statistics for within the plateau and after it. We choose this separation because we observed a distinctive shift in the qualitative behavior of our chosen metrics within these regimes. Our summary statistics rely on the average and standard deviation of any metrics within the training period considered. Here we assess all metrics based on the hold-out set of the corresponding loss. To demonstrate the behavior of the metrics over the optimization trajectory we plot them for a chosen PINN and Regression Model. The optimization trajectories remain the same for every such plot.

4 Results

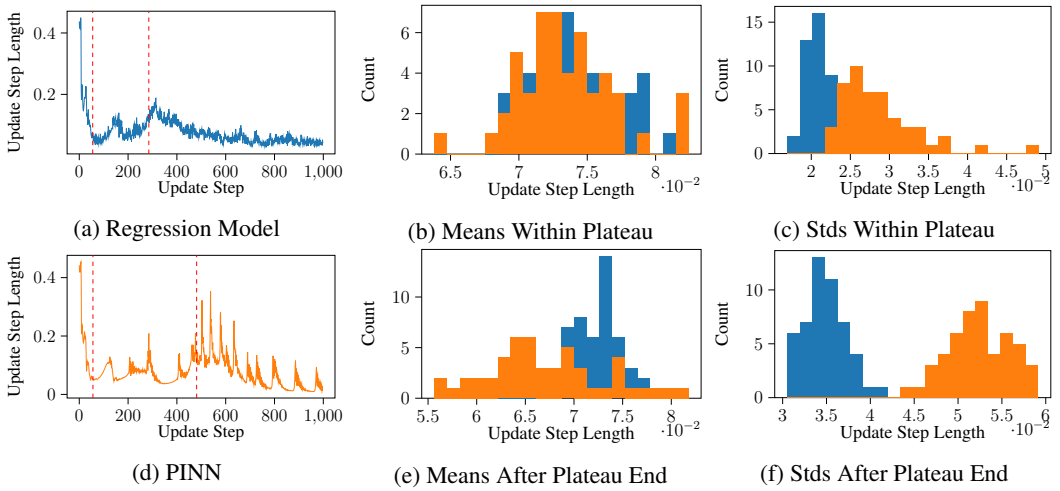


(a) Regression Model (b) PINN Network (c) Plateau Lengths

Figure 1: Regression Loss and Plateau Length.

The PINNs are stuck on plateaus longer than the Regression Models while retaining comparable update-step lengths. We see in Figure 1c that the number of steps the PINN spends in a plateau is roughly twice as much as for the Regression Model. This appears to be reflective of the general difficulty of training PINNs that has been observed in the literature. During such a plateau, it is often understood that small update-steps lead to longer escape times. However, we observe in Figure 2b that while the step lengths for the PINN are slightly shorter, they are generally comparable and can not fully explain the discrepancy in time spent in the plateau.

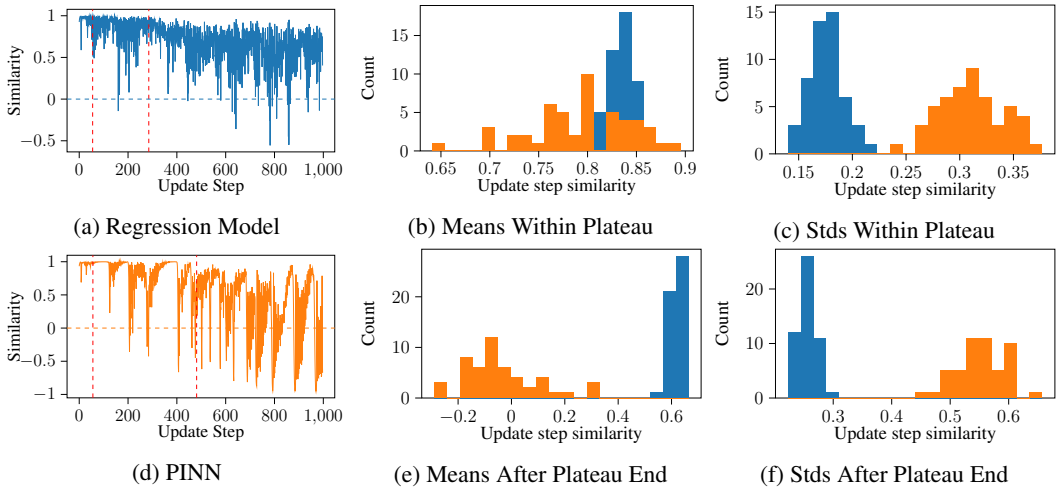
For the PINN models, consecutive update directions are often orthogonal or in slight contrast which may lead to an observed lower average distance traveled per step. In Figure 3b we observe that the mean cosine-similarity, and hence the similarity in direction, between two consecutive update-steps is significantly lower for the PINN model. Pairing this with the observation in Figure 3c that the variance of the similarities is also larger, we may reason that the optimization trajectory is significantly more varied for the PINN model. While not always the case, such a varied optimization trajectory often leads to a lower pace, as measured by average distance covered per update-step, which we confirm to be true via Figure 4f. Additionally, as seen in Figure 4b, the PINN covers slightly more distance during the plateau, which may be expected with a varied optimization trajectory. In combination the PINN will spend more time in the plateau. We emphasize that the average distance travelled per update-step is different to the true length of each update-step. The average distance



(a) Regression Model (b) Means Within Plateau (c) Stds Within Plateau

(d) PINN (e) Means After Plateau End (f) Stds After Plateau End

Figure 2: The update-step length (norm of the difference of two consecutive networks).



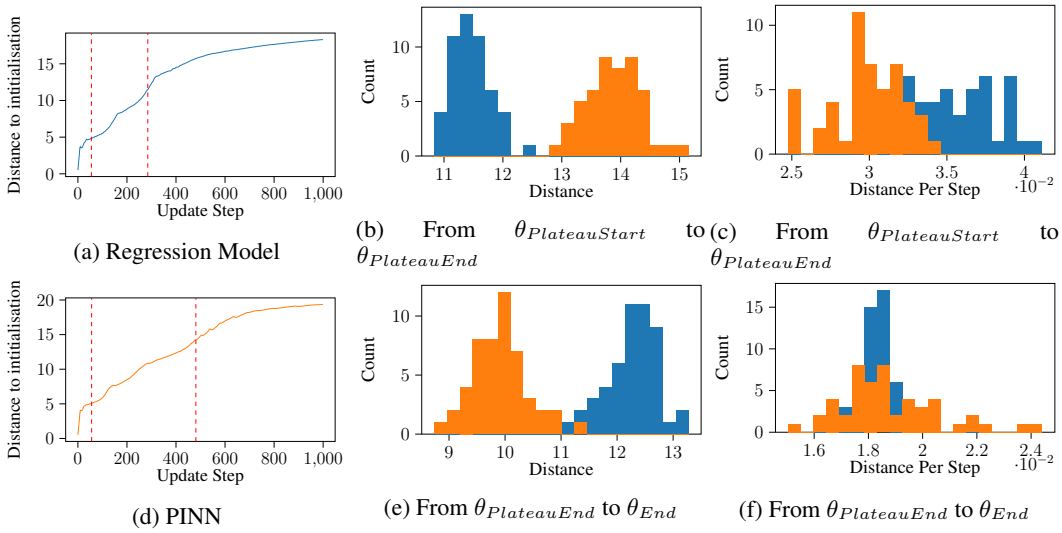
(a) Regression Model (b) Means Within Plateau (c) Stds Within Plateau

(d) PINN (e) Means After Plateau End (f) Stds After Plateau End

Figure 3: The cosine similarity between two consecutive update-steps.

travelled is simply the distance travelled over some number of update-steps and is hence only a measure of the overall rate of the optimization trajectory. While at first surprising, the decrease in update-step similarity after the plateau as seen in Figure 3e may be explained by the process finding a minimum and hence leading to slightly more "bouncing around" due to reaching more of a bowl shape. When this happens the optimization process slows down in distance travelled, as we observe in Figure 4e. This requires further study, as it has often been observed that due to symmetries level-sets are connected and hence minima may instead form valleys [12].

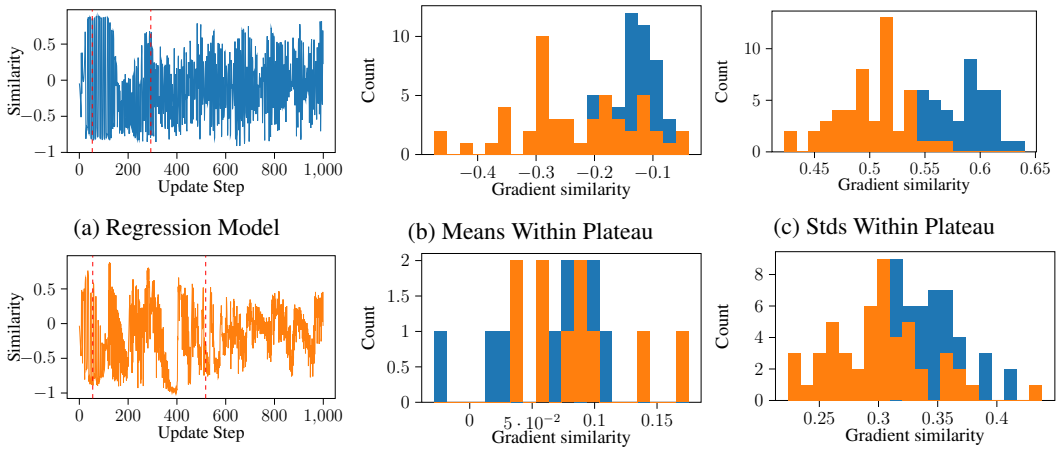
Our key observations center around the misalignment of the gradients of the differential and boundary loss. Specifically, we consider the cosine-similarity of the gradient of $\mathcal{L}_{Diff}$ and $\mathcal{L}_{Boundary}$. We observe that for both models the average similarity is close to zero, as seen in Figure 5b, and the variance is very high, as seen in Figure 5c. Even more so, we observe that for the PINN the gradients on average point in opposite directions. While λ_b adjusts the weighting of each loss term and hence the direction of a PINN update-step, this lack of similarity leads update-steps to not be well-aligned with either the differential or boundary loss descent direction. Specifically, when approaching regions with increased differential loss, due to the exploding gradient property subsequent differential loss gradients may be large and lead to update-step instabilities as observed in Figure 3. While the Regression Model moves through parameter regions where under the PINN loss it displays similar qualitative behavior to the PINN models, the Regression Model is ultimately unaffected as it does not depend explicitly on the differential loss. Hence, it does not suffer from



(a) Regression Model (b) From $\theta_{PlateauStart}$ to $\theta_{PlateauEnd}$ (c) From $\theta_{PlateauStart}$ to $\theta_{PlateauEnd}$

(d) PINN (e) From $\theta_{PlateauEnd}$ to θ_{End} (f) From $\theta_{PlateauEnd}$ to θ_{End}

Figure 4: The distance travelled by the network.



(a) Regression Model (b) Means Within Plateau (c) Stds Within Plateau

(d) PINN (e) Means After Plateau End (f) Stds After Plateau End

Figure 5: The cosine similarity between the gradient of $\mathcal{L}_{Diff}$ and $\mathcal{L}_{Boundary}$.

update-step instabilities and can move through areas which may exhibit larger underlying differential loss. Interestingly, the average similarity for the Regression Model is slightly positive, possibly a result of such reduced instabilities. Once the plateau ends, the variance of the similarity metric is reduced as seen in Figure 5f, and the update-steps become more aligned on average as seen in Figure 5e–potentially reflecting a more stable learning period after the plateau.

5 Conclusion

In our comparative study of PINN and Regression Model's optimization trajectories, we focused on the heat equation to provide a specific type of physical process. Reflective of the commonly observed difficulty of training PINN's, the PINN model exhibited prolonged stagnation in training compared to a Regression Model. This behavior appears to stem from unstable update-steps, which we find evidence to arise from an unfavorable interplay of the differential and boundary loss components. Based on our work, we believe optimization procedures which correctly account for conflicting descent directions from the components of the PINN loss, rather than merely re-weighting the components, may show better performance. Overall, further investigation is necessary to fully understand the properties of training under a PINN loss.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[3] Shengze Cai, Zhiping Mao, Zhicheng Wang, Minglang Yin, and George Em Karniadakis. Physics-informed neural networks (pinns) for fluid mechanics: A review. *Acta Mechanica Sinica*, 37(12):1727–1738, 2021.

[4] Shengze Cai, Zhicheng Wang, Sifan Wang, Paris Perdikaris, and George Em Karniadakis. Physics-informed neural networks for heat transfer problems. *Journal of Heat Transfer*, 143(6):060801, 2021.

[5] Marco Ciccone, Marco Gallieri, Jonathan Masci, Christian Osendorfer, and Faustino Gomez. Nais-net: Stable deep networks from non-autonomous differential equations. *Advances in Neural Information Processing Systems*, 31, 2018.

[6] Arka Daw, Jie Bu, Sifan Wang, Paris Perdikaris, and Anuj Karpatne. Rethinking the importance of sampling in physics-informed neural networks. *arXiv preprint arXiv:2207.02338*, 2022.

[7] Zhiwei Gao, Liang Yan, and Tao Zhou. Failure-informed adaptive sampling for pinns. *SIAM Journal on Scientific Computing*, 45(4):A1971–A1994, 2023.

[8] Batuhan Güler, Alexis Laignelet, and Panos Parpas. Towards robust and stable deep learning algorithms for forward backward stochastic differential equations. *arXiv preprint arXiv:1910.11623*, 2019.

[9] Xiaowei Jin, Shengze Cai, Hui Li, and George Em Karniadakis. Nsfnets (navier-stokes flow nets): Physics-informed neural networks for the incompressible navier-stokes equations. *Journal of Computational Physics*, 426:109951, 2021.

[10] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[11] Aditi S. Krishnapriyan, Amir Gholami, Shandian Zhe, Robert M. Kirby, and Michael W. Mahoney. Characterizing possible failure modes in physics-informed neural networks, 2021.

[12] Daniel Lengyel, Janith Petangoda, Isak Falk, Kate Highnam, Michalis Lazarou, Arinbjörn Kolbeinsson, Marc Peter Deisenroth, and Nicholas R. Jennings. Genni: Visualising the geometry of equivalences for neural network identifiability. 2020.

[13] Zhiping Mao, Ameya D Jagtap, and George Em Karniadakis. Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, 360:112789, 2020.

[14] Zhiping Mao, Ameya D. Jagtap, and George Em Karniadakis. Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, 360:112789, 2020.

[15] George S Misyris, Andreas Venzke, and Spyros Chatzivasileiadis. Physics-informed neural networks for power systems. In *2020 IEEE Power & Energy Society General Meeting (PESGM)*, pages 1–5. IEEE, 2020.

[16] Shikhar Nilabh and Fidel Grandia. Efficient training of physics-informed neural networks with direct grid refinement algorithm. *arXiv preprint arXiv:2306.08293*, 2023.

[17] The pandas development team. pandas-dev/pandas: Pandas, February 2020.

[18] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[19] 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*, 378:686–707, 2019.

[20] Francisco Sahli Costabal, Yibo Yang, Paris Perdikaris, Daniel E Hurtado, and Ellen Kuhl. Physics-informed neural networks for cardiac activation mapping. *Frontiers in Physics*, 8:42, 2020.

[21] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient pathologies in physics-informed neural networks, 2020.

A Heat Equation

The problem is solving the PDE:

$$\begin{cases} \Delta u(x,y,t) = \partial_t u(x,y,t) \\ u(0,y,t) = a_1 \\ u(L,y,t) = a_2 \\ u(x,0,t) = u(x,W,t) = 5sin(\frac{2\pi x}{L} \times freq) + a_1 + (a_2 - a_1)\frac{x}{L} \\ u(x,y,0) = 0 \text{ for } 0 < x < L,\ 0 < y < W \end{cases}$$

for $t \in [0, T]$.

Interpretation of above PDE The PDE describes the diffusion of heat on a rectangular region $[0, L] \times [0, W]$. Initially the region has zero temperature and at $t = 0$ the edges are given fixed temperature profiles for all subsequent time. The left edge is held at a_1, the right edge at a_2 and on the top and bottom edges oscillatory boundary conditions are imposed.

We fix the values of the parameters that define the PDE, $a_1 = 0$, $a_2 = 5$, $freq = 2$, $L = 1$ and $W = 1$. Changing the parameters governing the PDE and then conducting a network and hyperparameter search leads us to observe similar behaviour to what is described in this work.

B PINN Training Dataset

The dataset of points for PINN training is constructed by taking a 3D grid with the range $0 < x < L$, $0 < y < W$ and $0 < t \leq T$ and taking N equally spaced values for each dimension, using the `linspace` function. For the boundary points, including the initial condition, we use boundary and initial conditions to obtain the correct PDE solution values at these points. The remaining (x, y, t) points form the collocation points dataset

C Regression Loss and Numerical Solution

The numerical solution to the PDE can be obtained using a finite difference approximation. For a uniform grid the value, $\tilde{u}_{m,n,k+1}$, the approximate value of u at the coordinate $(m\epsilon, n\epsilon, (k+1)\delta)$ assuming the point does not lie on the boundary is given by

$$\tilde{u}_{m,n,k+1} = \frac{\delta}{(\epsilon)^2}(\tilde{u}_{m-1,n-1,k} + \tilde{u}_{m-1,n+1,k} + \tilde{u}_{m+1,n-1,k} + \tilde{u}_{m+1,n+1,k} - 4\tilde{u}_{m,n,k+1}) + \tilde{u}_{m,n,k+1}$$

the term in the brackets can be implemented using the convolution operation. The method is numerically stable when $\delta \leq \frac{\epsilon^2}{4}$, or equivalently $4(N-1)^2 \leq (N_t - 1)$ where N is the number of x coordinates and N_t is the number of timesteps used. For the boundary points the values are set such to what is given by the boundary condition. If only the solution at some time is desired, to get accurate results many intermediate timesteps are used.

For the Regression Loss we obtain a numerical solution with $N = 26$ and $Nt = 10,001$. The dataset which is ultimately used is a subset of this numerical solution such that 26 points are used in each axis direction. This is to ensure that we match the discretisation of the data used for the PINN loss.

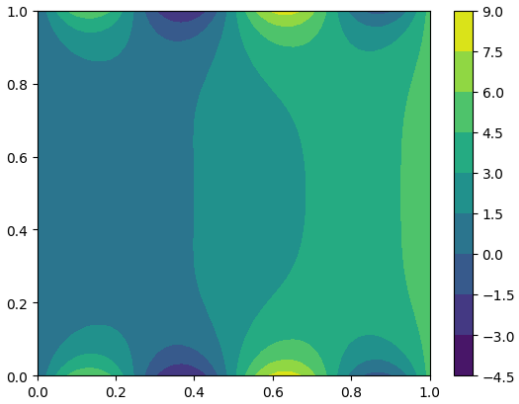


Figure 6: Numerical Solution at $t = 0.1$

Dataset generation for regression training To construct the dataset for regression training we set a value for N and Nt and then proceed with computing the numerical solution whilst collecting the approximate solution values for all of the N^2 grid points at each timestep.

D Network construction and Hyperparameter selection

Table 1 displays the associated parameters for the best three networks for the PINN and Regression Models as measured by regression loss.

The following are the hyperparameter ranges and general results for PDE coefficients, network construction and hyperparameter selection.

- Adam is the optimizer for all of the networks and the learning rate is chosen from $\{0.0001, 0.0005, 0.001\}$. The best learning rate was 0.001 for all top three models.
- The number of neurons in the hidden layer is taken from $\{32, 64, 128, 256\}$.
- The number of hidden layers is taken from $\{2, 3, 4\}$.
- For the PINN networks the batch size used for calculating the loss at each update-step is in $\{128, 256, 512, \text{full batch}\}$. For the Regression Models the batch sizes are the same except full batch is not an option due to the large size of the regression dataset.
- In the search we try two values of λ_b, 1 and 100 and all of the best networks shown here prefer the latter.

	Regression			**PINN**		
Setting	**1**	**2**	**3**	**1**	**2**	**3**
Model						
Architecture:	FCN	FCN	FCN	FCN	FCN	FCN
Activation:	$GELU$	sin	$GELU$	$GELU$	sin	sin
Width: 256	256	256	256	256	256	256
Number of Layers:	4	3	4	4	2	3
Data						
Batch Size:	512	512	128	Full Batch	Full Batch	Full Batch

Table 1: Top 3 Regression and PINN Configuration Settings. Note that the N_t parameter is not applicable for the PINN setups and the time spacing is governed by N in these cases. For the regression setups the parameters λ_b are not applicable.