Characterizing possible failure modes in physics-informed neural networks

Anonymous Author(s)
Affiliation
Address
email

Abstract

Recent work in scientific machine learning has developed physics-informed neural network (PINN) models by incorporating domain knowledge as soft constraints on the loss function. We analyze several distinct situations of widespread physical interest including learning differential equations with diffusion or convection operators. We show that the PINN method only works for relatively easy parameter regimes (i.e. with low convection/diffusion coefficients, and/or small forcing terms), and that it fails to learn the relevant physical phenomena for even moderately more challenging regimes. In each case, the reason can be attributed to the implicit use of the Lagrange dual form of the optimization (which amounts to replacing the constrained optimization problem with a regularized unconstrained problem) and the fact that the regularization term in PINN models involves derivatives (i.e., it is a differential operator, and can introduce a number of subtle problems including making the problem ill-conditioned). We show that this makes the loss landscape difficult to optimize. Finally, we show that these issues are not necessarily caused by a lack of expressivity in the NN architecture. Instead, we discuss approaches such as warm starting the initialization and/or posing the problem as a sequence-to-sequence learning task may actually make the problem easier to learn rather than learning to predict the entire space-time at once.

1 Introduction

Partial differential equations (PDEs) describing real-world physical phenomena are conventionally solved by using numerical methods that iteratively update and improve a candidate solution until convergence. These PDEs are often derived by starting from governing first principles (e.g., conservation of mass, energy, etc). For most applications, it is not possible to find analytical solutions to these PDEs, and as a result different numerical methods (e.g., the finite element method (FEM) [31], pseudo-spectral methods [6], etc.) are used to compute the solution. However, often times these PDEs are quite complex, and it is challenging to solve them numerically even on a supercomputer (e.g., turbulence simulations).

The need for computational efficiency to solve such problems—coupled with the increasing quantities of data available in many scientific applications—has resulted in an interest in deep learning (DL) approaches to replace or augment the traditional numerical PDE solvers. As a result, the area of scientific machine learning (SciML), which aims to couple traditional scientific mechanistic modeling (differential equations) with DL has emerged. In this vein, there have been a number of DL approaches to incorporate scientific knowledge into such problems while keeping the automatic, data-driven estimates of the solution [11, 12, 22].

While these SciML methods perform well in certain cases, we explore these models for several, broader scientific situations. For example, we look at scenarios such as forced convection which

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

describes the motion generated by an external source such as pumps, fans, or suction devices, and high viscosity diffusion which is present in many chemical and bioengineering applications including modeling polymeric solutions and microfluidic flow.

In the case of scientific modeling, many problems fit the following formulation,

$$\mathcal{F}(u(x,t)) = 0, \qquad x \in \Omega \subset \mathbb{R}^d, t \in [0,T], \tag{1}$$

where $\mathcal{F}$ can be a differential equation, and $u(x,t)$ is the state variable (i.e., parameter of interest).

In the context of PDEs, $\mathcal{F}$ can be a parabolic, hyperbolic, or elliptic PDE. Quintessential examples of $\mathcal{F}$ include the diffusion equation (a parabolic PDE), where $u(x,t)$ could be the temperature distribution over space and time, the convection equation (a hyperbolic PDE), where $u(x,t)$ could be fluid movement (such as air or some liquid) over space and time, or Laplace's equation (an elliptic PDE) where $u(x)$ could model a steady-state diffusion equation (in the limit as $t \to \infty$).

One possible approach to solve such a system is to obtain a large dataset u under different conditions and then train a neural network (NN) to directly predict the solution. Even with a large training dataset, it does not guarantee that the NN would obey the conservation laws or the governing equations of Eq. 1. In SciML problems, the constraints on the system matter, as they correspond to physical mechanisms of the system. For example, if energy is only approximately constrained, then the system being simulated may behave qualitatively differently or even fail to reach an answer.

Thus, another possibility is to apply Eq. 1 as a hard constraint when training the NN on the data. This can be formulated as the following constrained optimization problem, where $\mathcal{L}(u)$ is the data-fit term (such as the initial condition), and $\mathcal{F}$ is a constraint on the residual of the PDE system under consideration (i.e. the "physics" knowledge is the equation itself),

$$\min_{\theta} \mathcal{L}(u) = \mathcal{L}_{u_0} + \mathcal{L}_{u_b}, \text{ s.t. } \mathcal{F}(u) = 0, \tag{2}$$

where L_{u_0} and L_{u_b} measure the misfit to initial and boundary conditions (which are pre-specified/given as input data), and θ denotes the parameters of a NN model $\mathcal{M}$ that gets (x,t) (and possibly other inputs) and outputs $u(x,t)$. The goal is to train this NN model to minimize the loss in Eq. 2 to best match the initial and boundary conditions and also satisfy the PDE constraint. Also note that in general, $\mathcal{L}$ and $\mathcal{F}$ are potentially non-linear and non-convex.

It is often very difficult to directly solve Eq. 2 with $\mathcal{F}(u)$ as a hard constraint. However, the constrained optimization problem can be relaxed into an unconstrained one by imposing "soft constraints" (which can be viewed as a form of regularization) on the outputs of the NN that effectively aim to penalize the violations of $\mathcal{F}(u)$ for some $\lambda_{\mathcal{F}} \geq 0$:

$$\min_{\theta} \mathcal{L}(u) + \lambda_{\mathcal{F}} \mathcal{F}(u), \tag{3}$$

where $\lambda_{\mathcal{F}}$ is a regularization parameter. Here, the NN learns to minimize the loss function that includes both the objective and the constraint. This formulation has become one of the most common in physics-constrained machine learning, and in particular, physics-informed neural networks (PINNs) [18] is one of the most popular of such approaches.

Our main contributions are as follows:

- We first start by testing the physics-informed neural network (PINN) on simple, but widely used, problems of diffusion and convection. We find that the PINN fails to learn the relevant physics even for cases with simple analytical solutions. In particular, we find that the PINN achieves very high errors of almost 100% for PDEs with relatively large (but physically relevant) diffusion, or convection coefficients. The PINN also achieves high errors for strongly forced (but again, physically relevant) non-homogeneous PDEs (see § 3).
- We analyze the PINN loss landscape and find that adding the PDE-based soft constraint makes it very complex and hard to optimize, especially for cases with large diffusion and convection coefficients. We also study how the loss landscape changes as the regularization parameter is changed. While we find that reducing the regularization parameter helps alleviate the complexity of the loss landscape, it does not solve the problem (see § 4 for details).

- We show that the problem arising from the PINN's failures is not due to the limited capacity of the NN architecture. We do so by showing how a simple reinitialization technique to warm start the NN weights can resolve the failures for simple cases, which proves that the NN has the capacity to find a good solution. We argue that the failure to do so is due to the optimization difficulty associated with adding the PINN's soft PDE constraint as regularization (see § 5 for details).

- We show that changing the problem paradigm from learning to predict the entire space-time to a sequence-to-sequence problem can reduce the PINN error, without any change to the NN architecture (see § 5 for details).

We note that this approach of incorporating physics-based regularization, where the regularization constraint, $\mathcal{L}_{\mathcal{F}}$, corresponds to a differential operator, is *very* different than incorporating much simpler norm-based regularization (such as L_1 or L_2 regularization). Specifically, simple norm-based regularization is convex in nature. As a result, the loss function is convex. This means that the general formulation (in our case, $f(x)$ is $\mathcal{L}(u)$ and $g(x)$ is $\mathcal{F}(u)$) of $\min_x f(x)$ s.t. $g(x) \leq \alpha$ and $\min_x f(x) + \lambda g(x)$ are equivalent in the sense that given $\lambda > 0$, there exists a $\alpha > 0$ such that the optimal solutions to both the problems coincide.

In contrast, $\mathcal{L}_{\mathcal{F}}$ is non-trivially structured. Therefore, when the loss function is non-convex, *there is no such guarantee of equivalency between the two expressions*. For a fixed $\alpha > 0$, there may be solutions W^*_α of $\min_x f(x)$ s.t. $g(x) \leq \alpha$ for which there exists no $\lambda > 0$ such that $W^*_\alpha = W^*_\lambda$ [1]. This is problematic because $\mathcal{L}_{\mathcal{F}}$ corresponds to actual physical quantities, and there is an important distinction between satisfying the constraint exactly versus approximately (the soft constraint approach only doing the latter).

As we will show, the procedure of training a PINN can lead to infeasible outputs for a number of common scientific use cases. The reasons we identify—having to do with hard versus soft constraints and using convex norms versus differential operators as regularization terms—are central to both ML and PDEs; and to deliver on the promise of SciML, it is important to couple them appropriately.

2 Related work

Deep learning and PDEs. DL approaches for PDE problems have been increasing rapidly in recent years [8, 13]. A number of tools and methodologies now exist to solve scientific problems via combining DL and domain insights [14, 17, 18, 26]. As mentioned earlier, a popular approach to combine DL and physical knowledge is to include aspects of the PDE term as part of the optimization process via regularization. A notable aspect of this approach is that the NN is only trained on data that comes from the governing equation(s) itself. This has garnered interest and shown successful results in a wide variety of problems and applications [2, 10, 19–21, 30].

However, there have also been issues noticed with this formulation. For example, it did not work well for the stiff ordinary differential equations (ODEs) describing chemical kinetics [9]. Wang *et al.* attributed some problems found with PINNs to imbalanced back-propagated gradients in the loss function during training and proposed a learning-rate annealing scheme to mitigate some of these issues [23]. They also looked at the PINN model in the context of neural tangent kernels (i.e. towards the infinite width limit) to analyze the convergence [24, 25]. They found some cases where the model failed (such as if the target function exhibits "high frequency features") and showed some preliminary solutions via the lens of the neural tangent kernel.

Physical priors and constraints in NNs. Imposing physical priors and constraints on NN systems is common in scientific ML problems, as a way to try to enforce a relevant property of interest. Some approaches have focused on embedding specialized physical constraints into NNs, such as conservation of energy or momentum [3, 7]. While methods focusing on constraining the output of the NN are more common, it is difficult to exactly enforce such constraints in DL settings. Previous work has tried to impose hard constraints in DL (both within the context of scientific ML and otherwise) [4, 15, 16, 27], but this can be challenging.

3 Possible failure modes for physics-informed neural networks

In this section, we highlight several examples where the physics-informed neural network formulation defined in Eq. 3 does not predict the solution well. We demonstrate this with two different types of simple, canonical PDE systems (for which we have an analytical solution), convection (§ 3.1) and diffusion (§ 3.2). While the PINN models work for easier parameter regimes (homogeneous PDEs with low convection or diffusion coefficients), we demonstrate that they fail to learn the relevant physical phenomena in the harder parameter regimes (which are relevant for a number of scientific problems) that deviate from the "easy" ones. As we see, while adding the physical constraint as a soft regularization may be easier to deploy and optimize with existing unconstrained optimization methods, it does come with trade-offs, including that in many cases the optimization problem becomes much more difficult to solve. Note, however, that getting a "bad" solution doesn't mean the optimization problem was hard, and one can still get a "good" solution even if the optimization is hard.

Experiment setup. We consider both homogeneous and non-homogeneous PDEs and vary the convection and diffusion coefficients. For each problem, we aim to minimize the loss function in Eq. 3. We use a 4-layer fully-connected NN with 50 neurons per layer, a hyperbolic tangent activation function, and randomly sample collocation points (x, t) on the domain. We train this network using the L-BFGS optimizer and sweep over learning rates from $1\text{e}{-4}$ to 2.0. We run the models at least ten times with different preset random seeds and average the relative and absolute errors in $u(x, t)$ over all trials. We list the error bars (standard deviations) in the Appendix (§ C) for all results.

In § 3.1.1 and § 3.2.1, the systems have periodic boundary conditions. We enforce this through an extra term in the loss function that takes the difference between the predicted NN solution at each boundary. In § 3.1.2 and § 3.2.2, the initial and boundary terms are included in the same loss term, targeting the exact u value at those points.

3.1 Learning convection

We consider a one-dimensional convection equation, which is commonly used to model transport phenomena and is a standard example of a hyperbolic PDE:

$$\begin{aligned} \frac{\partial u}{\partial t} + \beta \frac{\partial u}{\partial x} &= q, \quad x \in \Omega,\ t \in [0, T] \\ u(x, 0) &= h(x) \quad x \in \Omega. \end{aligned} \tag{4}$$

Here, β is the convection coefficient, and $h(x)$ is the initial condition. This problem has a simple analytical solution for constant β and period boundary conditions as:

$$u_{\text{analytical}}(x, t) = F^{-1}\big(F(h(x))e^{-i\beta k t}\big), \tag{5}$$

where F is the Fourier transform and $i = \sqrt{-1}$. The general loss function for this problem is,

$$\mathcal{L}(\theta) = \frac{1}{N_u}\sum_{i=1}^{N_u}\Big((\hat{u}(\theta, x_0^i, 0) - u_0^i)^2\Big) + \frac{1}{N_f}\sum_{i=1}^{N_f}\lambda_i\Big(\frac{\partial \hat{u}(\theta, x_f^i, t_f^i)}{\partial t} + \beta\frac{\partial \hat{u}(\theta, x_f^i, t_f^i)}{\partial x} - q\Big)^2 + \mathcal{L}_{\mathcal{B}}, \tag{6}$$

where $\hat{u} = \mathcal{M}(\theta, x, t)$ is the output of the NN, and $\mathcal{L}_{\mathcal{B}}$ is the boundary loss. For periodic boundary conditions with $\Omega = [0, 2\pi)$ (as in § 3.1.1), this loss is:

$$\mathcal{L}_{\mathcal{B}} = \frac{1}{N_b}\sum_{i=1}^{N_b}\Big(\hat{u}(\theta, 0, t) - \hat{u}(\theta, 0, 2\pi, t)\Big)^2 \tag{7}$$

3.1.1 Evaluating non-homogeneous (forced) PDEs for different speeds of propagation

First, we consider the simplest case for Eq. 4, when the forcing function q is a constant. When q is non-zero, it acts as a *forcing term*, with the following simple initial and boundary conditions:

$$\begin{aligned} u(x, 0) &= sin(x), \\ u(0, t) &= u(2\pi, t). \end{aligned} \tag{8}$$

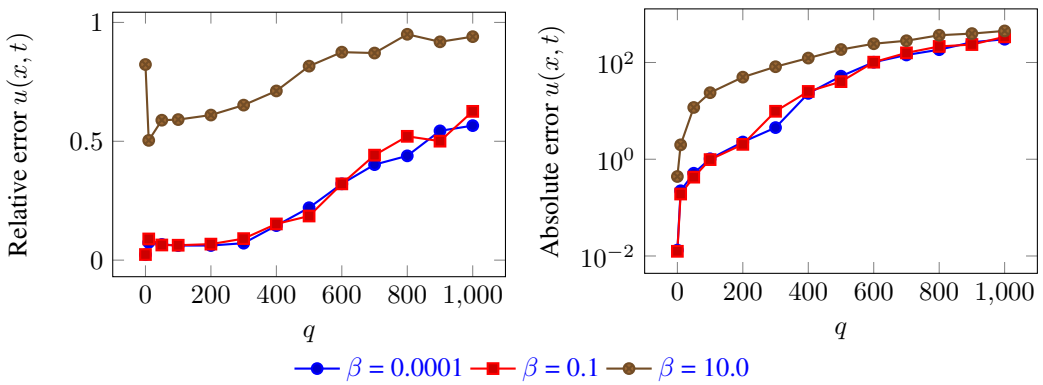


Figure 1: *Relative (left) and absolute (right) error for predicted solutions of $u(x,t)$ against q, the forcing term. Absolute error is on the log scale. Each line represents a value of β. The general trend is that 1) as q, the forcing, is increased, the prediction error also increases and 2) the model does especially poorly at high β, where the error is high even with no forcing.*

We apply the PINN's soft regularization to this problem and optimize the loss function in Eq. 6. After training, we measure the relative error between PINN's predicted solution and the analytical solution and report it in Fig. 1. As one can see, the PINN model runs into two possible failure modes: 1) when q is large (*forced* convection), regardless of β, and 2) when β is large; in particular, the relative error is very high (almost 100%) when $\beta = 10.0$ and $q = 0$, i.e. in a situation with no forcing. Figure 2 shows that in the latter case, the predicted solution predicts something close to zero (very different from what it should predict) past a certain time.

3.1.2 Convection with an exact solution that does explicitly not depend on β

In this section, we test another case where the solution does not explicitly depend on the convection coefficient. This eliminates the variability of large β affecting the solution (as we saw previously, large β was one of the harder regimes to learn), as now we learn the same solution for each value of β. We do this by imposing the following initial and boundary conditions:

$$\begin{aligned} u(x,0) &= sin(\pi x), \\ u(-1,t) &= 0. \end{aligned} \tag{9}$$

By setting the forcing term to be $q = -e^{-t}sin(\pi x) + \beta\pi e^{-t}cos(\pi x)$, the analytical solution will be:

$$u(x,t) = sin(\pi x)e^{-t}, \tag{10}$$

which implies that the solution does not explicitly depend on β, and that β only affects the PDE residual term (i.e., $\mathcal{L}_{\mathcal{F}}$).

Figure 3 shows the error after training as we vary β. Interestingly, the error still increases, even though the solution is not explicitly dependent on β.

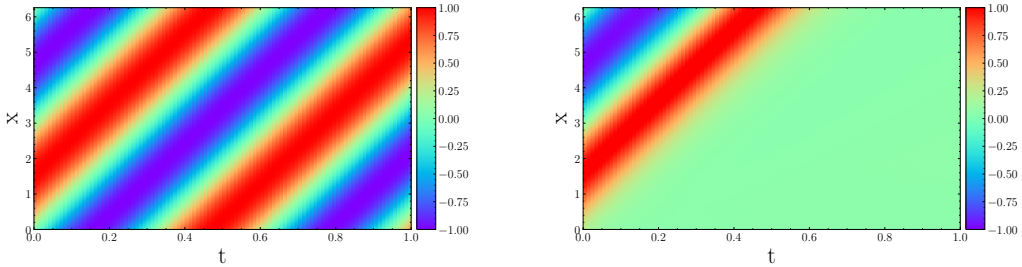


Figure 2: *Left: Heatmap of the exact solution to the 1D convection equation, Eq. 4, when $\beta = 10.0$ and $q = 0$ (no forcing). Right: Physics-informed NN predicted solution: the network has difficulty predicting the solution past a certain timestep. The average relative u error across all trials is 0.82.*

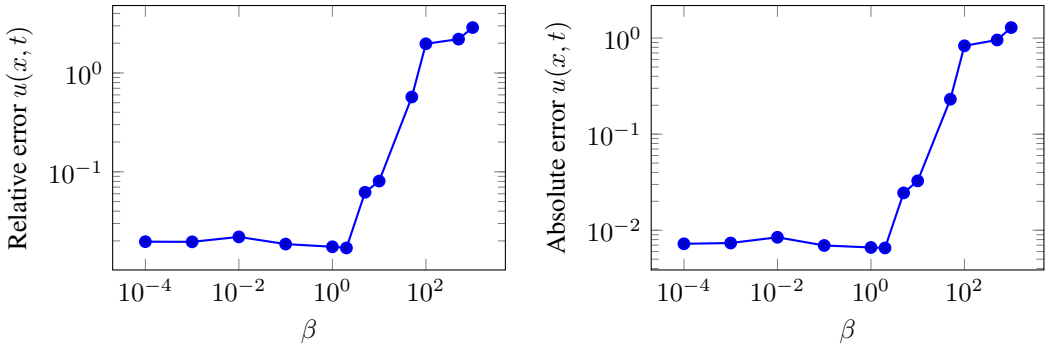


Figure 3: *Relative (left) and absolute (right) error for predicted solutions of $u(x,t)$ against β, the speed of propagation, where $u(x,t) = sin(\pi x)e^{-t}$. This time, the solution does not explicitly depend on β: the high errors as β increases come from the regularization term. Error sharply increases when $\beta > 1.0$.*

3.2 Learning diffusion

We next consider the one-dimensional diffusion equation, which models how a quantity (of mass, heat, etc) diffuses through a given region. It is a prototypical example of a parabolic PDE; and it is an overarching equation in science and engineering. It is:

$$\begin{aligned} \frac{\partial u}{\partial t} - \nu \frac{\partial^2 u}{\partial x^2} &= q, \quad x \in \Omega,\ t \in (0, T] \\ u(x, 0) &= h(x) \quad x \in \Omega. \end{aligned} \tag{11}$$

Here, ν ($\nu > 0$) is a constant representing viscosity, which can be thought of as a system's "resistance" to deformation, and $h(x)$ is the initial condition. Similar to the convection problem, this diffusion problem has a simple analytical solution. For constant ν and periodic boundary conditions, it is:

$$u_{\text{analytical}}(x,t) = F^{-1}\big(F(h(x))e^{i\nu kt}\big), \tag{12}$$

The general loss function for this problem is,

$$\mathcal{L}(\theta) = \frac{1}{N_u}\sum_{i=1}^{N_u}\Big((\hat{u}(\theta, x_0^i, 0) - u_0^i)^2\Big) + \frac{1}{N_f}\sum_{i=1}^{N_f}\lambda_i\Big(\frac{\partial \hat{u}(\theta, x_f^i, t_f^i)}{\partial t} - \nu\frac{\partial^2 \hat{u}(\theta, x_f^i, t_f^i)}{\partial x^2} - q\Big)^2 + \mathcal{L}_{\mathcal{B}}. \tag{13}$$

Similar to before, periodic boundary conditions can be enforced by an extra term in the loss (Eq. 7).

3.2.1 Evaluating non-homogeneous (forced) PDEs for different viscosity

We look at a simple case where solution does not depend on ν at all. We assume the following initial and boundary conditions,

$$\begin{aligned} u(x, 0) &= \sin(x), \\ u(0, t) &= u(2\pi, t). \end{aligned} \tag{14}$$

We look at how well the NN predicts the solution for varying values of ν and q. Once again, as Fig. A.1 shows, we see that the NN formulation has some possible failure modes including 1) increasing q increases error (when ν is also large, we could classify this is a *stiff* problem) and 2) high ν does poorly under all conditions (including when q is 0). Fig. 4 shows that in the latter case, the network largely mispredicts further timesteps and is also not able to capture the symmetry of the system.

3.2.2 Diffusion with an exact solution that does not explicitly depend on ν

As we did with the convection equation, we test another case where the solution does not depend on the viscosity coefficient at all, by imposing the following initial and boundary conditions:

$$\begin{aligned} u(x, 0) &= \sin(\pi x), \\ u(-1, t) &= u(1, t) = 0. \end{aligned} \tag{15}$$

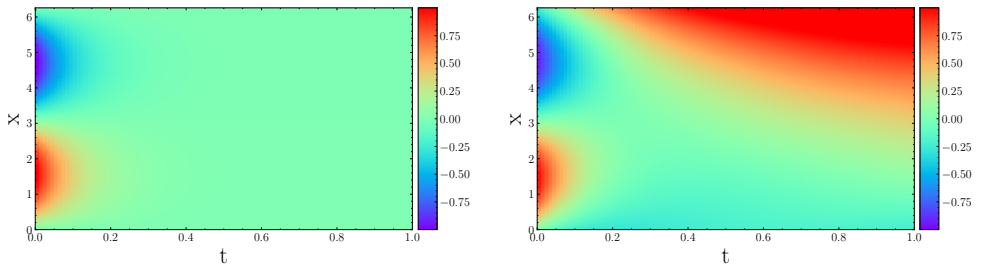


Figure 4: *Left: Heatmap of the exact solution to 1D diffusion equation, Eq. 11, when $\nu = 10.0$ and $q = 0$. Right: Physics-informed NN predicted solution. Note that the NN is not able to capture the symmetry of the system. The average relative u error across all trials is 0.98.*

By setting the forcing term to be $q = -e^{-t}\sin(\pi x) + \nu\pi^2 e^{-t}\sin(\pi x)$, the analytical solution is:

$$u(x,t) = \sin(\pi x)e^{-t}. \tag{16}$$

The solution does not explicitly depend on ν, i.e., ν is only present in the PDE residual term via q. We look at how the error varies as a function of different ν. It is clear from Fig. A.2 that once ν is greater than 0.1, the error starts to sharply increase.

4 Diagnosing possible failure modes for physics-informed NNs

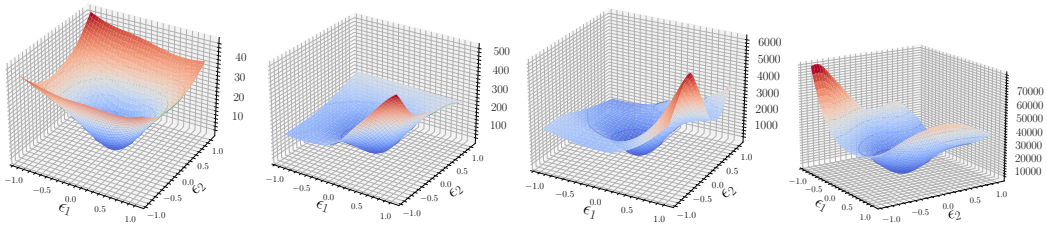
As discussed above, PINN can result in high errors even for simple physics constraints. Here, we demonstrate that the underlying reasons for this have to do with the subtle problems introduced through the PDE-based soft constraint of $\mathcal{L}_\mathcal{F}$ which makes the loss landscape very difficult to optimize.

4.1 Changing loss landscapes with respect to the loss or physical parameters

Here, we analyze loss landscape changes for different regimes for the diffusion problem in § 3.2 and show that adding the soft constraint regularization may actually make the problem harder to optimize (see § D.1 for the corresponding convection results). We plot the loss landscape by perturbing the (trained) model across the first two dominant Hessian eigenvectors and computing the corresponding loss values. This tends to be more informative than perturbing the model parameters in random directions [28, 29].

Figure 5 shows the loss landscape for the diffusion problem discussed in § 3.2.2, for different diffusion coefficients ν. Note that the analytical solution for this problem does not depend on ν (Eq. 16).

Interestingly, the loss landscape at a relatively low $\nu = 0.1$ is rather smooth, but increasing ν further results in a complex and non-symmetric loss landscape. It is also evident that the optimizer has gotten



ν	0.1	1.0	2.0	10.0
Relative error	0.020	0.054	0.098	0.60
Absolute error	0.007	0.019	0.034	0.21

Figure 5: *Loss landscapes for varying values of ν. When ν is low, the loss landscape is more smooth. As ν increases, more deformations and ridges are introduced, resulting in a more complex and non-symmetric loss landscape. The error also increases.*

stuck in a local minima with a very high loss function for large ν values. This clearly shows that adding the PDE soft term results in not being able to add enough regularization to the optimization loss landscape. We also observe a similar behavior for the diffusion problem in § 3.2.1 (shown in § D.2), and also for the convection problem in § 3.1.1 (shown in § D.3).

One might perhaps argue that the amount of soft constraint added needs to be adjusted by changing its weight (i.e. the λ parameter in Eq. 3) [23]. However, we find that while tuning λ is helpful, it cannot resolve the problem, as shown in Fig. D.1. Interestingly, we observe that for small values of the regularization parameter, the relative/absolute prediction error is reduced in comparison to no PINN regularization. However, note that as the regularization parameter is increased, the loss landscape becomes increasingly more complex and harder to optimize (see the z-axis scale).

Ill-conditioned regularization: Taking a step back, the behaviour that we are observing should not be surprising as the PDE based regularization operator (i.e., $\mathcal{L}_\mathcal{F}$ term) in the PINN is in fact ill-conditioned, which leads to unstable numerical behaviour. It is easy to show that the condition number for the regularization operator for the diffusion problem is $\mathcal{O}(\nu N^2)^2$, where N is the grid size, and the square comes from the L_2 loss term (see § E). Similarly, the condition number for the convection problem scales as $\mathcal{O}(\beta N)^2$ which is still quite high. As such, it is not surprising that this ill-conditioned behaviour would lead to instability which can manifest itself in large gradients, and/or poor convergence behaviour that was also reported in [23]. This also explains why in the original PINN work the NN was able to learn the physics, as a very small diffusion coefficient of $\nu = 0.003$ was used for learning Burger's equation [18]. Future work should consider preconditioning the regularization operator and/or consider other penalty functions with a better condition number.

As we see, adding the PDE regularization term as a soft constraint can introduce a number of subtle problems. Importantly, note that we do not observe these behaviors with simple norm-based regularization methods, such as ridge regression. Additionally, this is an area where the combination of a non-linear NN and PDE regularization can actually make the problem more challenging to optimize.

5 Expressivity versus optimization difficulty

In this section, we explore addressing some of the issues that we observed in the previous sections. One important counter argument could be that the NN architecture used in our experiments does not have enough expressivity/capacity to solve for high convection/diffusion coefficient cases in § 3.1.2 and § 3.2.2. However, we provide two counter arguments. First, we show that the NNs that we considered in the previous section do have the capacity to minimize the loss function to a low loss value in § 5.1. Second, we show that changing the paradigm of having the NN output all the state-space can make the problem much harder. Instead, we argue that in some cases one may get much better performance (by to an order of magnitude better accuracy) by training the NN to predict one time step at a time in § 5.2.

5.1 Does the NN have enough capacity?

As mentioned above, one possible argument for the failure modes shown in § 3 could be that the NN does not have enough capacity. Here, we show that this is not necessarily the underlying issue (at least for the experiments that we considered). To illustrate this, we devise a simple method to warmup start the NN training by finding good initialization for the weights. To do so, we first solve the problem with relatively small coefficients (for either convection/diffusion). Note that from the previous experiments in § 3.1.2 and § 3.2.2, we observed that the PINN can achieve good accuracy for these cases. As such, we use the weights corresponding to this small coefficient case, to warm start the training for the larger coefficient experiments. Interestingly, the results show that using this warm start re-initialization does lead to very low errors, irrespective of the convection/diffusion coefficient as illustrated in Fig. 6. This clearly illustrates that the NN does have the capacity to learn this problem, but that the corresponding loss landscape becomes very hard to optimize (which is in line with the results in § 4.1).

5.2 Sequence-to-sequence learning vs learning the entire space-time solution

Another subtle issue in the PINN approach of [18] is that the NN is trained to learn to predict the entire space-time at once. That is, the NN has to predict the state variable u for all locations and

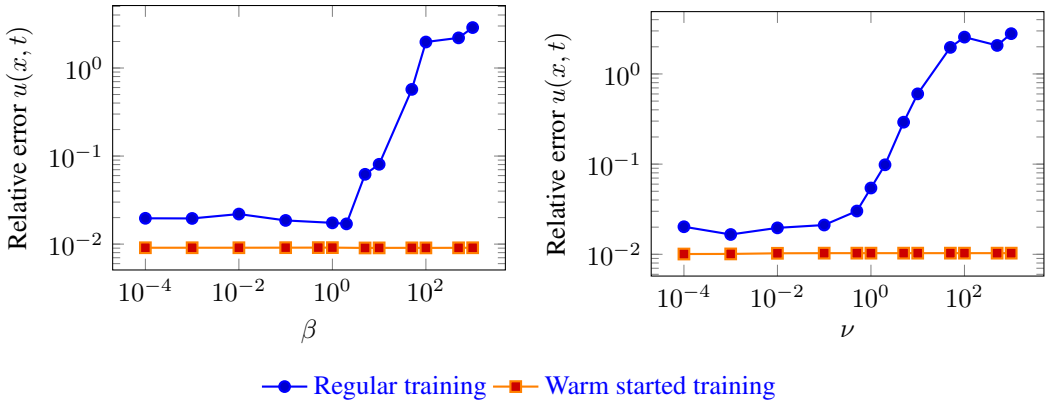


Figure 6: *Convection (left) from § 3.1.2 and diffusion (right) from § 3.2.2 (from § 3.1.2 and § 3.2.2 respectively) relative errors for predicted solutions of $u(x,t)$ for different values β and ν respectively. We use the weights corresponding to a "good" model (i.e., a low β or low ν model that exhibits low error) to warm start the training for the larger coefficient experiments. The model is now able to find low error solutions for both high β and high ν.*

time points. This is a very difficult task to learn, especially for an ill-conditioned problem. Here, we argue that it may be better to pose the problem as a sequence-to-sequence learning task, where the NN learns to predict the solution at the next time step, instead of all times as done in PINN [18]. This way we can utilize a marching-in-time scheme to predict different sequences/time points. We test this scheme by using the exact same NN architecture as previous sections and report the results in Tab. F.1 for the convection problem and Tab. F.2 for the diffusion problem. Interestingly, we find that for both cases, posing the problem as learning to predict the sequence results in lower errors. Again this is somewhat expected since the problem is ill-conditioned and restricting the dimensions is indeed expected to help. This behaviour also has corollaries with numerical methods used in scientific computing, where space-time problems are typically harder to solve for as compared to time marching methods [5].

6 Conclusions

PINNs—and SciML more generally—hold great promise for expanding the scope of ML methodology to important problems in science and engineering. For these problems, however, integrating ML methods with PDE-based domain-driven constraints as a soft regularization term can lead to subtle and critical issues. In particular, we show that this approach can have fundamental limitations which results in failure modes for learning relevant physics commonly used in different fields of science. To show this, we picked two fundamental PDE problems of diffusion and convection and showed that the PINN only works for very simple cases, failing to learn the relevant physical phenomena for even moderately more challenging regimes. We then analyzed the problem to characterize the underlying reasons why these failures occur. In particular, we studied the PINN loss landscape behavior and found it becomes it becomes increasingly complex for large values of diffusion or convection coefficients, and with/without non-homogeneous forcing. Then we discussed that the problem is not necessarily due to the limited capacity of the NN, but that it is partly an optimization problem resulting in the PDE-based soft constraint used in PINNs. Furthermore, we showed that the PINN approach of solving for the entire space-time at once, may not be efficient, and instead posing the problem as a sequence-to-sequence learning task can provide lower error rates.

References

[1] S. Boyd, S. P. Boyd, and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[2] Y. Chen, L. Lu, G. E. Karniadakis, and L. Dal Negro. Physics-informed neural networks for inverse problems in nano-optics and metamaterials. *Optics express*, 28(8):11618–11633, 2020.

[3] M. Cranmer, S. Greydanus, S. Hoyer, P. Battaglia, D. Spergel, and S. Ho. Lagrangian neural networks. *arXiv preprint arXiv:2003.04630*, 2020.

[4] P. L. Donti, D. Rolnick, and J. Z. Kolter. Dc3: A learning method for optimization with hard constraints. *arXiv preprint arXiv:2104.12225*, 2021.

[5] K. Eriksson, D. Estep, P. Hansbo, and C. Johnson. *Computational differential equations*. Cambridge University Press, 1996.

[6] B. Fornberg. *A practical guide to pseudospectral methods*. Number 1. Cambridge university press, 1998.

[7] S. Greydanus, M. Dzamba, and J. Yosinski. Hamiltonian neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/26cd8ecadce0d4efd6cc8a8725cbd1f8-Paper.pdf.

[8] J. Han, A. Jentzen, and E. Weinan. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018.

[9] W. Ji, W. Qiu, Z. Shi, S. Pan, and S. Deng. Stiff-pinn: Physics-informed neural network for stiff chemical kinetics. *arXiv preprint arXiv:2011.04520*, 2020.

[10] X. Jin, S. Cai, H. Li, and G. E. Karniadakis. Nsfnets (navier-stokes flow nets): Physics-informed neural networks for the incompressible navier-stokes equations. *Journal of Computational Physics*, 426:109951, 2021.

[11] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang. Physics-informed machine learning. *Nature Reviews Physics*, 2021. doi: 10.1038/s42254-021-00314-5. URL https://doi.org/10.1038/s42254-021-00314-5.

[12] A. Karpatne, G. Atluri, J. H. Faghmous, M. Steinbach, A. Banerjee, A. Ganguly, S. Shekhar, N. Samatova, and V. Kumar. Theory-guided data science: A new paradigm for scientific discovery from data. *IEEE Transactions on knowledge and data engineering*, 29(10):2318–2331, 2017.

[13] Z. Long, Y. Lu, X. Ma, and B. Dong. Pde-net: Learning pdes from data. In *International Conference on Machine Learning*, pages 3208–3216. PMLR, 2018.

[14] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis. Deepxde: A deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, 2021.

[15] P. Márquez-Neila, M. Salzmann, and P. Fua. Imposing hard constraints on deep networks: Promises and limitations. *arXiv preprint arXiv:1706.02025*, 2017.

[16] Y. Nandwani, A. Pathak, P. Singla, et al. A primal dual formulation for deep learning with constraints. 2019.

[17] C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, A. Ramadhan, and A. Edelman. Universal differential equations for scientific machine learning. *arXiv preprint arXiv:2001.04385*, 2020.

[18] 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.

[19] M. Raissi, A. Yazdani, and G. E. Karniadakis. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, 367(6481):1026–1030, 2020.

[20] J. Sirignano and K. Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.

[21] V. Sitzmann, J. Martel, A. Bergman, D. Lindell, and G. Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33, 2020.

[22] L. von Rueden, S. Mayer, K. Beckh, B. Georgiev, S. Giesselbach, R. Heese, B. Kirsch, J. Pfrommer, A. Pick, R. Ramamurthy, et al. Informed machine learning–a taxonomy and survey of integrating knowledge into learning systems. *arXiv preprint arXiv:1903.12394*, 2019.

[23] S. Wang, Y. Teng, and P. Perdikaris. Understanding and mitigating gradient pathologies in physics-informed neural networks. *arXiv preprint arXiv:2001.04536*, 2020.

[24] S. Wang, H. Wang, and P. Perdikaris. On the eigenvector bias of fourier feature networks: From regression to solving multi-scale pdes with physics-informed neural networks. *arXiv preprint arXiv:2012.10047*, 2020.

[25] S. Wang, X. Yu, and P. Perdikaris. When and why pinns fail to train: A neural tangent kernel perspective. *arXiv preprint arXiv:2007.14527*, 2020.

[26] E. Weinan, J. Han, and A. Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics*, 5(4):349–380, 2017.

[27] K. Xu and E. Darve. Physics constrained learning for data-driven inverse modeling from sparse observations. *arXiv preprint arXiv:2002.10521*, 2020.

[28] Z. Yao, A. Gholami, Q. Lei, K. Keutzer, and M. W. Mahoney. Hessian-based analysis of large batch training and robustness to adversaries. *Advances in Neural Information Processing Systems*, 2018.

[29] Z. Yao, A. Gholami, K. Keutzer, and M. W. Mahoney. Pyhessian: Neural networks through the lens of the hessian. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 581–590. IEEE, 2020.

[30] Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, and P. Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics*, 394:56–81, 2019.

[31] O. C. Zienkiewicz, R. L. Taylor, P. Nithiarasu, and J. Zhu. *The finite element method*, volume 3. McGraw-hill London, 1977.

7 Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] Yes, and general limitations.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] In the supplemental material.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] In the supplementary.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowd sourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Learning diffusion results

We present the error plots for the cases described in ....

A.1 Evaluating non-homogeneous (forced) PDEs for different viscosity

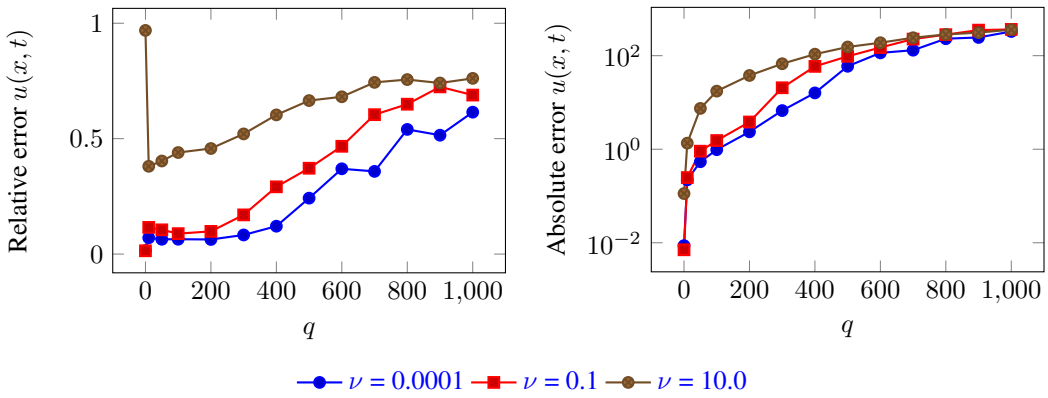


Figure A.1: *Relative (left) and absolute (right) error for predicted solutions of $u(x,t)$ for different values of ν, the viscosity, against q, the forcing term. Absolute error is on the log scale. Each line represents a value of ν. Similar to the convection case, we see a general trend that 1) error increases as q, the forcing, increases and 2) at high ν, without any forcing, the relative error is almost 100% and stays high for other values of q.*

A.2 Diffusion with an exact solution that does not depend explicitly on ν

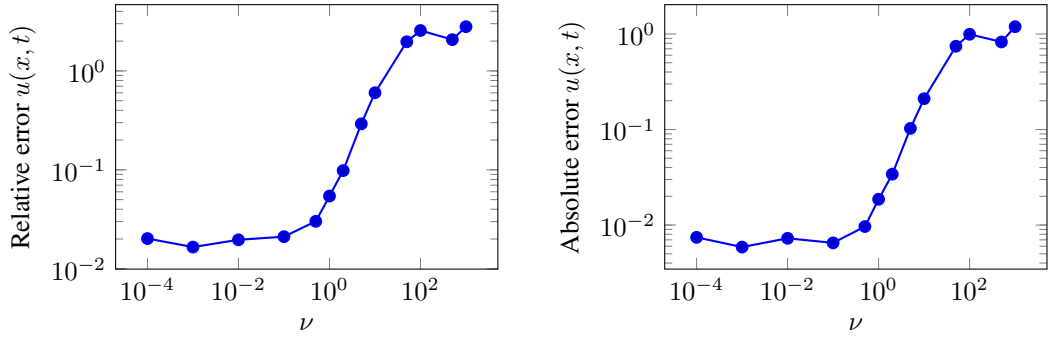


Figure A.2: *Relative (left) and absolute (right) error for predicted solutions of $u(x,t)$ against ν, the viscosity, where $u(x,t) = sin(\pi x)e^{-t}$. The solution does not explicitly depend on ν. The prediction error steadily increases for $\nu > 0.1$, where ν values only influence the regularization term (as ν is present in the forcing term).*

B Exact and predicted results for different systems

B.1 Convection, q

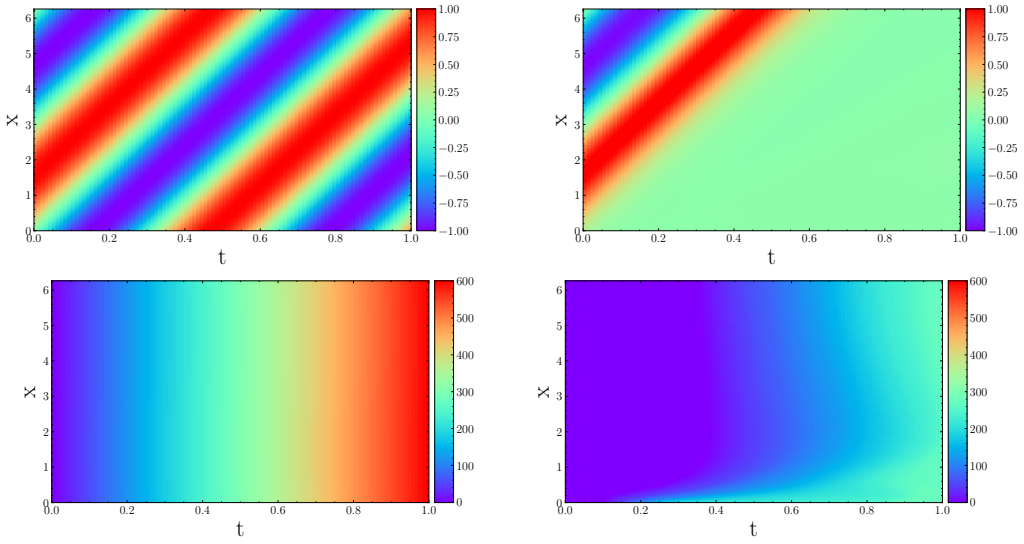


Figure B.1: *Top left: Heatmap of the exact solution to the 1D convection equation, Eq. 4, when $\beta = 10.0$ and $q = 0$. Top right: Physics-informed NN predicted solution: the network has difficulty predicting the solution past a certain timestep. The average relative u error across all trials is 0.82. Bottom left: Heatmap of the exact solution to the 1D convection equation, Eq. 4, when $\beta = 0.0001$ and $q = 600$ (strongly forced). Bottom right: Physics-informed NN predicted solution. The average relative u error across all trials is 0.32.*

B.2 Diffusion, fixed u

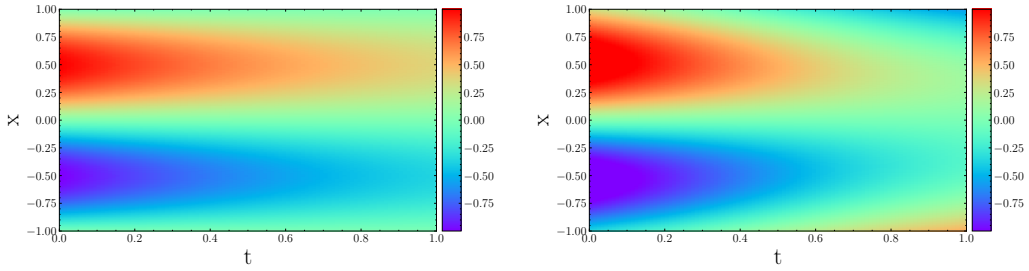


Figure B.2: *Left: Heatmap of the exact solution to 1D diffusion equation, Eq. 11, when $\nu = 5.0$ and $u(x,t) = sin(\pi x)e^{-t}$. Right: Physics-informed NN predicted solution. The network predicts the solution dissipation incorrectly, and also mispredicts at the boundaries.*

C Error bars for experimental results

For each experiment, we ran the models at least ten times with different preset random seeds and averaged relative and absolute errors in $u(x,t)$ over all trials. In this section, we list all of the errors and standard deviations across trials.

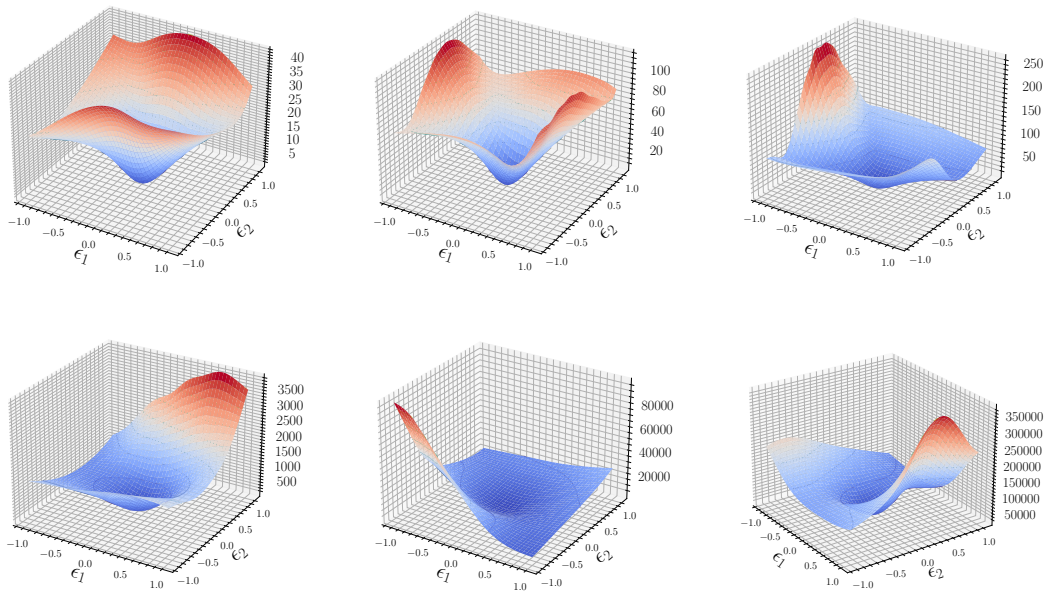
We ran all experiments on 1 GPU, GeForce RTX 3090.

C.1 Changing loss landscapes

We show the loss landscapes for the other use cases we showed in § 3.

D Loss landscapes for 1D diffusion with changing λ

We include the loss landscape results for 1D diffusion with an exact solution.



λ	0.0	0.001	0.01	0.1	1.0	10.0
Relative error	0.478	0.051	0.032	0.050	0.292	1.03
Absolute error	0.174	0.019	0.012	0.018	0.103	0.377

Figure D.1: *For the one-dimensional diffusion equation with an exact solution (§ 3.2.2), we vary the λ parameter, train the NN model, and plot the loss landscape. For this experiment, we use $\nu = 5.0$, as it is a point at which the error starts to grow as shown in Fig. A.2. The relative and absolute errors for u in each situation are also given: when the regularization term is overemphasized ($10\mathcal{L}_{\mathcal{F}}$), the prediction error is the worst. As the regularization parameter is increased, the loss landscape becomes increasingly more complex and harder to optimize (see the z-axis scale).*

D.1 Loss landscapes for 1D convection that does not explicitly depend on β

D.2 Loss landscapes for evaluating non-homogenous (forced) PDEs for different viscosity

We show the loss landscapes for the problem setup described in § 3.2.1.

D.3 Loss landscapes for evaluating non-homogenous (forced) PDEs for different speeds of propagation

We show the loss landscapes for the problem setup described in § 3.1.1.

E Derivation of Condition Number for PDE Regularization in PINN

F Discretizing the mesh

λ	Entire state space	$\Delta t = 0.0001$	$\Delta t = 0.001$	$\Delta t = 0.01$	$\Delta t = 0.1$
0.0	1.09	0.019	**0.024**	0.074	0.644
0.0001	1.10	0.026	0.028	0.070	0.629
0.001	0.946	0.045	0.046	0.072	0.408
0.01	0.700	**0.016**	0.031	**0.036**	**0.303**
0.1	**0.326**	0.039	0.058	0.096	0.323
1.0	0.820	0.282	0.257	0.231	0.493

Table F.1: *Predicting the relative error for $u(x,t)$ at t = 0 + Δt as a function of λ. This is shown for the situation described in § 3.1.1, when β is 10.0. We also include the relative error for predicting the whole state space for different values of λ. The lowest error for each Δt (and the entire state space) is bolded. The best space-time model has a relative error of 32.6%, while the best timestep model has a relative error of 1.9%*

λ	Entire state space	$\Delta t = 0.0001$	$\Delta t = 0.001$	$\Delta t = 0.01$	$\Delta t = 0.1$
0.0	2.23	**0.019**	0.024	0.073	0.576
0.0001	1.82	0.024	**0.022**	**0.070**	0.556
0.001	2.18	0.057	0.049	0.058	0.204
0.01	1.58	0.131	0.133	0.123	**0.157**
0.1	1.14	0.229	0.248	0.235	0.261
1.0	**0.929**	0.308	0.335	0.305	0.362

Table F.2: *Predicting the relative error for $u(x,t)$ at t = 0 + Δt as a function of λ. This is shown for the situation described in § 3.2.1, when ν is 10.0. We also include the relative error for predicting the whole state space for different values of λ. The lowest error for each Δt (and the entire state space) is bolded. The best space-time model has a relative error of 96.7%, while the best timestep model has a relative error of 1.6%.*