

PINN: Investigating its design and exploring its complexity and capability in solving coupled PDEs

Submitted in partial fulfilment of the requirements for the degree of
Master of Technology

By

Shivprasad Kathane

Roll No. 180110076

Under the guidance of
Prof. Shyamprasad Karagadde



Centre for Machine Intelligence and Data Science
INDIAN INSTITUTE OF TECHNOLOGY BOMBAY

June 2023

Approval Sheet

This dissertation entitled “*PINN: Investigating its design and exploring its complexity and capability in solving coupled PDEs*” by Shivprasad Kathane is approved for the degree of Master of Technology in AI and Data Science (C-MInDS).

Examiners

Prof. Amuthan Ramabathiran

Prof. M.P. Gururajan

Supervisor

Prof. Shyamprasad Karagadde

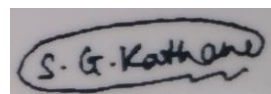
Chairman

Date: 30 June 2023

Place: IIT Bombay, Mumbai

Declaration

I declare that this written submission represents my ideas based on my understanding and I have adequately cited and referenced the original source where others' ideas or words have been included. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the institute and can evoke penal action from the sources which have thus not been properly cited or from whom the proper submission has not been taken when needed.

A handwritten signature in black ink, enclosed in an oval shape. The signature reads "S. G. Kathane".

Shivprasad Kathane

180110076

Acknowledgement

I would like to take this opportunity to express gratitude to my guide, Prof. Shyamprasad Karagadde for guiding me in this project, especially with his suggestions on what possible aspects I can work on and the steps I can take as part of my research. Meetings with him, updating on my work and receiving his inputs have instilled a sense of confidence in me. I also thank all the professors of the Indian Institute of Technology, Bombay especially from Metallurgical Engineering and Material Science and Machine Intelligence and Data Science, who have imparted me foundational knowledge and instilled in me an interest to pursue research. I am also thankful to my friends and family for their support and encouragement.

Shivprasad Kathane

IITB, 20 June 2023

Abstract

Physics-Informed Neural Networks (PINNs) have been a popular deep-learning framework for building models that are both data-driven and physics-informed. This report delves into two crucial aspects related to PINNs. One, it investigates the design aspects of PINNs and probes whether one can engineer a unique PINN structure to solve a given partial differential equation (PDE). Hyperparameters in PINN and their possible values are identified based on the literature. Model tuning for several PDEs is performed using Bayesian optimisation and the relationship between the optimal number of layers in PINN and the features of PDE is studied using ML. Second, the report showcases how the framework can be used to solve complex problems. Problems of increasing complexity are discussed. The strategies present in the literature and insights gained from experiments are highlighted. A scheme is employed within the PINN framework to solve systems consisting of coupled PDEs and involving a moving interface, with a solidification problem taken as a benchmark.

Table of Contents

Introduction.....	6
1. Investigating the Design of PINN	7
1.1. Introduction to PINN.....	7
1.2. Design Aspects of PINN.....	14
1.2.1. Data Distribution and Sampling	14
1.2.2. Loss Function Design	14
1.2.3. Optimisation.....	14
1.2.4. Neural Network Architecture	15
1.2.5. Error Analysis	17
1.2.6. Generalisability	17
1.3. Hyperparameter Tuning using Bayesian Optimisation.....	18
1.4. Example: Linear Convection Diffusion Equation	19
1.5. Learning the PINN Design from PDE	22
1.6. Are the number of layers enough to solve a PDE?	24
2. Solving Complex Problems using PINN	26
2.1. Introduction.....	26
2.2. Burgers' Equation.....	29
2.3. Two Phase Stefan's Problem.....	31
2.4. Directional Solidification Problem.....	34
2.5. A benchmark problem involving multiple PDEs	37
2.5.1. Motivation: Porosity Prediction.....	37
2.5.2. Problem Description	37
2.5.3. Techniques Considered	41
2.5.4. Results.....	44
Conclusion.....	52
References	53

Introduction

Modelling and forecasting the dynamics of multiphysics and multiscale systems is a crucial challenge in the scientific community with several problems being critical to engineering applications. Traditionally, numerical methods have been used to simulate systems but they can be time-consuming and fail to model highly complex systems. However, enhanced computing power has given rise to the field of deep learning and consequently, neural networks have been developed to model systems. But conventionally, these are data-intensive and ignore any process information or knowledge of conditions available. Physics-Informed Neural Network (PINN) is a novel framework which enables data-driven and physics-informed learning simultaneously via the modification of loss function to enforce the physical constraints which can be the partial differential equations (PDEs) representing the process or the initial and boundary conditions. However, there isn't an established theory on how to design them to best solve a given PDE.

Chapter 1 of this research delves into the design aspects of PINNs such as the data, the neural network architecture, the loss function and its optimisation. The strategies affecting the learning process are discussed, the hyperparameters and their values are identified, and model tuning is performed using Bayesian optimisation for several PDEs, with a specific example of the linear convection-diffusion equation. The results are tabulated and an important question is posed: whether a unique structure exists for PINN to best solve a given PDE? A regression model fit on the number of layers in a tuned PINN versus the features of the PDE such as the number of operations in the equation indicates such a possibility. Another exercise is performed which suggests that not the number of layers alone but the optimal combination of hyperparameters best solves the problem.

Chapter 2 of this research ventures deep into the intricacies and flexibilities of the PINN framework to understand how different strategies and algorithms can be adopted for solving complex problems. PINNs are expected to possess the potential to address complexities such as shocks, steep gradients, discontinuities, missing conditions etc. better compared to the numerical methods. Problems of increasing complexities are taken up such as the nonlinear Burgers' equation, the 2-phase Stefan's problem, the directional solidification problem. Several processes such as alloy solidification play a crucial role in the manufacturing of critical aerospace/automotive components. Defects like porosity might arise during such processes, which makes their modelling and prediction important. We take up a benchmark complex problem of a solidification system governed by multiple coupled PDEs and involving two phases with a moving interface which can be useful to model porosity in an alloy. Drawing insights from experiments and adopting strategies from literature, a scheme is employed within the PINN framework to solve such problems to a good extent.

1. Investigating the Design of PINN

1.1. Introduction to PINN

The framework of Physics Informed Neural Network (PINN) first introduced in [1] has been used to solve partial differential equations (PDEs) with several studies reported in the literature. PINNs are neural networks that take into account information from the differential equation reflecting the physics of the problem within the learning process. A need for such kind of framework arose mainly because standard numerical approaches tend to be time-consuming or too difficult to solve some PDEs that represent significant non-linearities, convection dominance, or shocks and also to take advantage of the learning power of neural networks. PINNs represent a multi-task learning framework with the dual objective of fitting the observed data and minimising the PDE residual. Such a hybrid approach ensures that the trained model can recognise patterns from the data (observational bias), respects the conservation laws (inductive bias) and can be used to predict the dynamics of the system (learning bias) which forms the basis of the field of physics-informed machine learning. [2] See Fig 1.

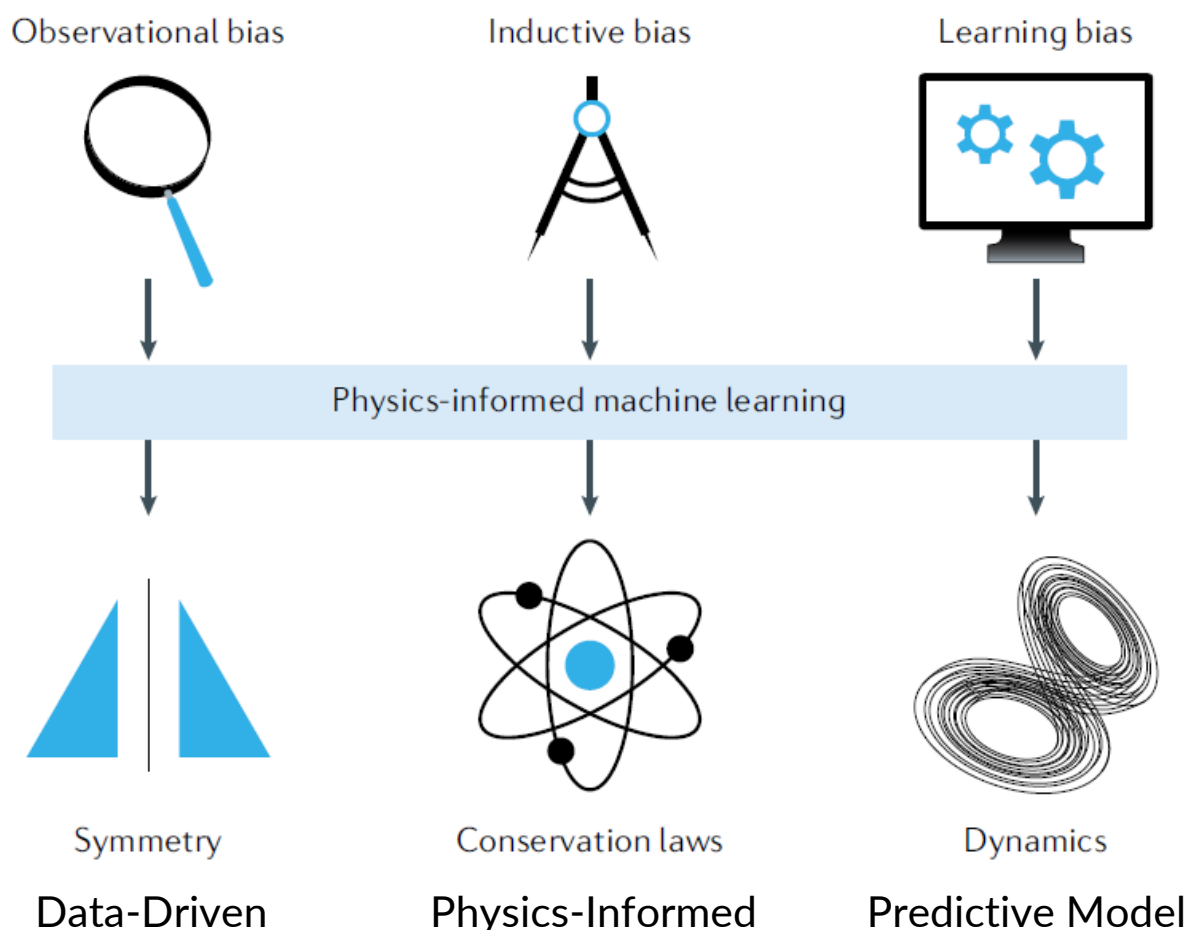


Fig 1: The concept of building a comprehensive model under Physics-Informed ML [2]

The idea of PINN originates from the universal function approximation capabilities of neural networks because of which they can be used to model any function, say u , dependent on inputs, say x (space) and t (time). If NN is the neural net output, then:

$$NN \approx u(x, t) \quad (1)$$

Now, if we have a differential equation in u then neural networks can be used to solve it if we can iteratively learn to model u while simultaneously obeying the physical law represented by the equation. This idea can be traced back to [3] which presented a method to solve initial and boundary value problems using artificial neural networks. It proposed to start with a trial solution of the differential equation as a sum of two parts, one satisfying the known conditions and containing no adjustable parameters (A) while the other uses a feedforward neural network (with adjustable weights) to satisfy the differential equation without affecting the conditions (F).

Let the differential equation be represented by $G = 0$. Here G will be a function of input x , the solution to be computed ψ and the derivatives of ψ (upto order n say 2):

$$G(x, \psi(x), \nabla \psi(x), \nabla^2 \psi(x)) = 0 \quad x \in D, \text{the domain} \quad (2)$$

If a neural network with weights p is used to model ψ , then we train the network (i.e. vary p) to satisfy Eq. 2 subject to the given conditions. The trial solution is written as:

$$\psi(x) = A(x) + F(x, NN(x, p)) \quad (3)$$

Here, by construction, ψ satisfies the conditions given in A while the objective of neural network training is to minimise the sum of G values over data points x .

This approach was general enough to be applicable to single ordinary differential equations (ODEs), systems of coupled ODEs, and also to PDEs although a separate trial function needs to be constructed depending on the equation.

Similarly, PINNs approximate PDE solutions by training a neural network to minimise a loss function which includes terms reflecting the initial conditions (at start time) and boundary conditions (in space), and the PDE residual at selected points in the domain (collocation data). Traditionally, neural networks learn solely from data by minimising the error between predictions and labels but PINNs are novel in the sense that they make the neural network learn to respect any symmetries, invariances, or conservation principles by integrating the mathematical model into the network by designing a custom loss function. The PDE residual acts as a penalising term in the loss function and restricts the space of acceptable solutions. The PINN training is an unsupervised mesh-free strategy that does not require results from prior simulations or experiments (necessary for a general supervised deep learning problem) and is free from step limitations (that otherwise arise in numerical simulations). [4]

A parametrized and nonlinear partial differential equation can be represented as:

$$u_t + \mathfrak{N}[u; \lambda] = 0, \quad x \in \Omega, \quad t \in [0, T] \quad (4)$$

where $u(t, x)$ as before denotes the latent (to be determined) solution and \mathfrak{N} denotes a non-linear operator parameterized by λ , while $\Omega \subset \mathfrak{R}^d$ is the domain.

Two distinct problems are of interest when provided with noisy measurements of the system. Determining the hidden state $u(t, x)$ of the system given fixed model parameters λ (corresponding to the PDE) forms the forward problem of inference, filtering, and smoothing, or data-driven solution of PDE. The inverse problem is of determining the parameters λ that best explain the observed data which is also known as learning, system identification, or data-driven discovery of PDE. [1] showcased how to build continuous-time models in the PINN framework for solving Schrodinger's Equation in the forward case and Navier-Stokes Equation in the inverse scenario.

$$\text{Define } f = u_t + \mathfrak{N}[u; \lambda] \quad (5)$$

Then, approximating $u(t, x)$ by a deep neural network results in the physics-informed neural network $f(t, x)$. f can be derived by applying the chain rule on the composition of functions using automatic differentiation. It has the same parameters as u but due to the action of \mathfrak{N} has extra final activation functions.

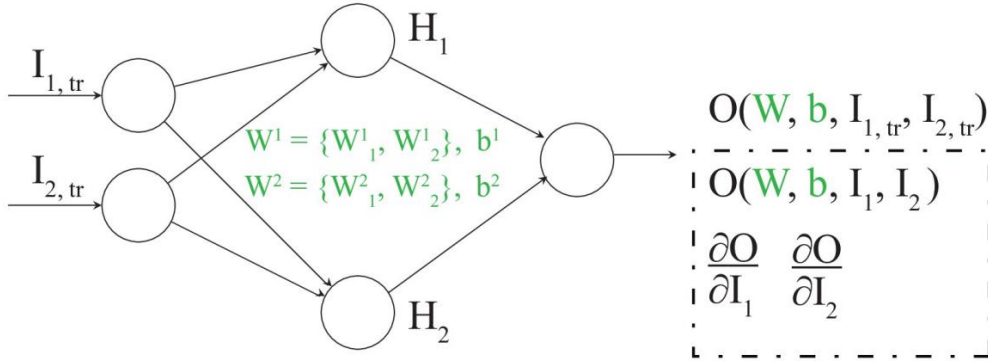


Fig 2: A traditional neural network with 2 neurons in the input layer corresponding to inputs I_1 and I_2 , 2 neurons in the hidden layer with weights $W1$ & $W2$ and biases $b1$ & $b2$, and 1 neuron in the output layer corresponding to O expressed as a function of W, b, I_1 , and I_2 . [5]

A traditional neural network such as the one shown above is used to train over initial ($t = 0$) and boundary ($\partial\Omega_D$) data points (collection of inputs $I_1 = x$ & $I_2 = t$ and output label $O = u$). This represents a supervised learning setup.

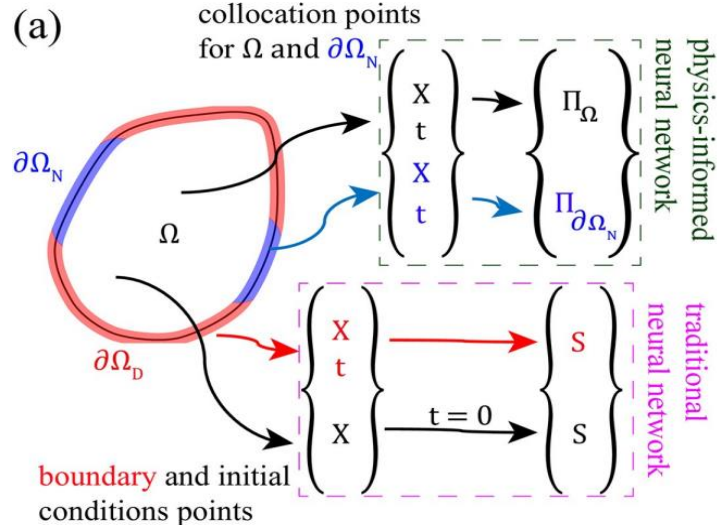


Fig 3: An illustration showing the parts of input space used for training PINN [5]

The collocation input data (points within the domain or on the boundary) after passing through the traditional neural network yield corresponding outputs which along with their differentiations wrt the inputs and PDE parameters $\lambda = \theta$ are used to evaluate the PDE as $f = \Pi$. This forms a physics-informed neural network which represents an unsupervised learning setup (no label for collocation data).

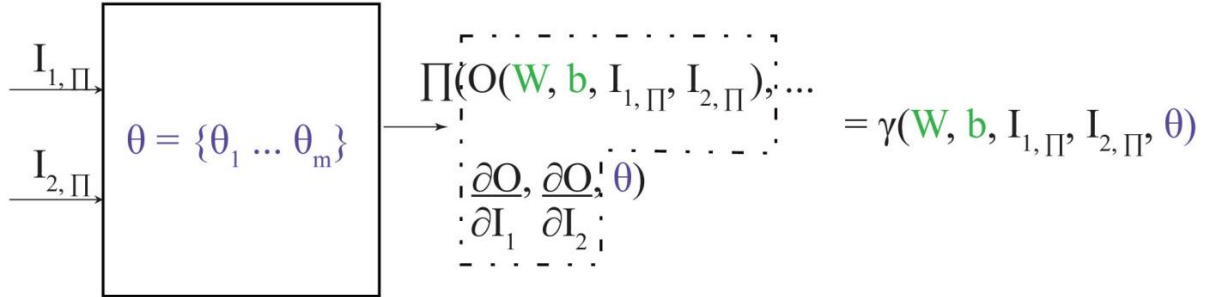


Fig 4: The forward problem is identifying u . This requires evaluating PDE value Π (system output) given inputs I_1 and I_2 , system parameter θ , and neural network output O (approximate of u). This value can be expressed as γ being a function of W , b , I_1 , I_2 , and θ . Estimating the unknown parameter set θ given input-output data is the inverse problem. [5]

The PINN training proceeds by minimising the loss function L wrt parameters p :

$$L_p(X) = L_p^o(X^o) + L_p^b(X^b) + L_p^r(X^r) \quad (6)$$

where X denotes the collection of training data

This is a non-convex loss function often minimised using gradient-based optimization. The PDE residual loss term is formed by calculating the derivatives in the PDE using automatic differentiation of the neural-network outputs with respect to their inputs. While the data-fit loss terms are composed of errors between the predicted (by NN) and true outputs.

The loss term consists of three mean square terms:

1. Mean Squared Residual over Collocation Data $X^r = \{(t_i^r, x_i^r) \text{ for } i = 1 \text{ to } N_r\}$

$$L_p^r(X^r) = 1/N_r * \sum_{i=1}^{N_r} |f_p(t_i^r, x_i^r)|^2 \quad (7)$$

2. Mean Squared Error over Initial Data $X^o = \{(t_i^o, x_i^o) \text{ for } i = 1 \text{ to } N_o\}$

$$L_p^o(X^o) = 1/N_o * \sum_{i=1}^{N_o} |u_p(t_i^o, x_i^o) - u_o(x_i^o)|^2 \quad (8)$$

where u_o is a known function for generating labels for initial data

3. Mean Squared Error over Boundary Data $X^b = \{(t_i^b, x_i^b) \text{ for } i = 1 \text{ to } N_b\}$

$$L_p^b(X^b) = 1/N_b * \sum_{i=1}^{N_b} |u_p(t_i^b, x_i^b) - u_b(t_i^b, x_i^b)|^2 \quad (9)$$

where u_b is a known function for generating labels for boundary data

Each of these terms can have an associated individual weight in the final loss term.

There can be additional loss terms corresponding to other labelled data if provided.

It is this loss term which acts as a feedback mechanism from the physics-informed network that tunes the weights of the neural network and thereby ensures that the model learns the underlying physical laws while fitting to the observed/known data.

The below figure comprehensively illustrates the 2-module PINN framework:

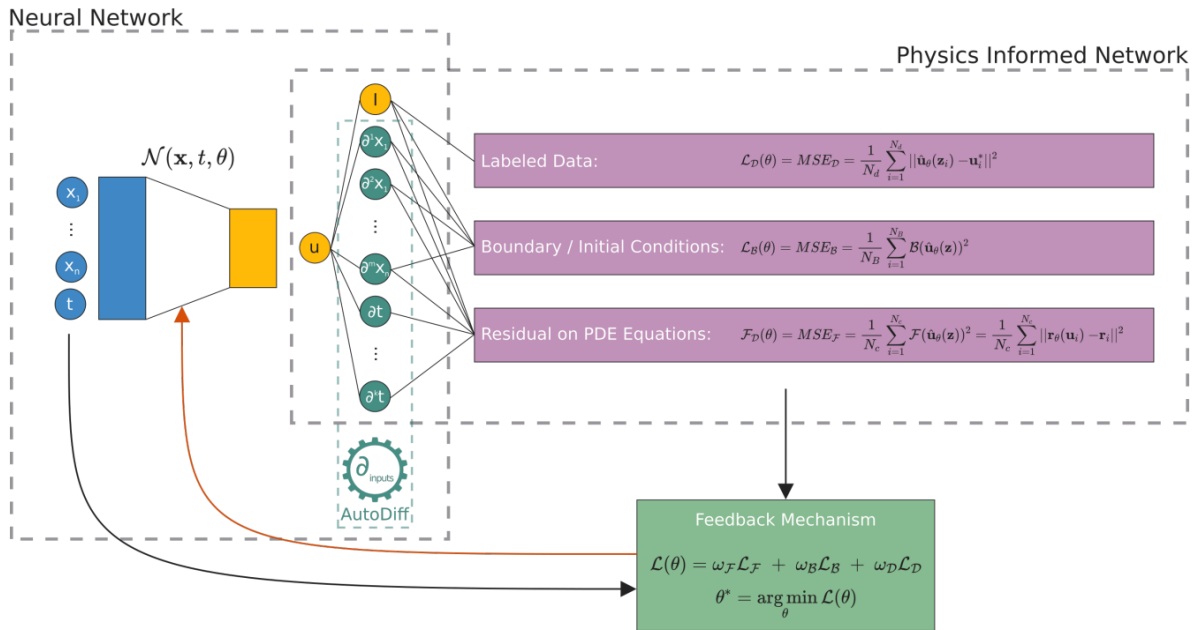


Fig 5: Building blocks of Physics-Informed Neural Networks [4]

Let us look at a few examples from literature where PINNs were used to solve PDEs

For the Schrodinger's Equation in the domain $x \in [-5, 5]$, $t \in [0, \pi/2]$

$$f(t, x) = i\hbar_t + 0.5\hbar_{xx} + |\hbar|^2\hbar \quad \text{subject to conditions} \quad (10)$$

$$\hbar(0, x) = 2 \operatorname{sech}(x), \quad \hbar(t, -5) = \hbar(t, 5), \quad \hbar_x(t, -5) = \hbar_x(t, 5)$$

For this problem solved in [1], the training set consisted of $N_0 = 50$ data points on $\hbar(0, x)$, $N_b = 50$ collocation points for enforcing the periodic boundaries and $N_r = 20,000$ collocation points used to enforce Eq. 10 inside the solution domain. All of them were randomly sampled using a space-filling Latin Hypercube Sampling strategy. A 5-layer deep neural network with 100 neurons per layer and a hyperbolic tangent activation function was used as the architecture to model PDE solution $\hbar(t, x)$. Loss function was optimised using the L-BFGS technique.

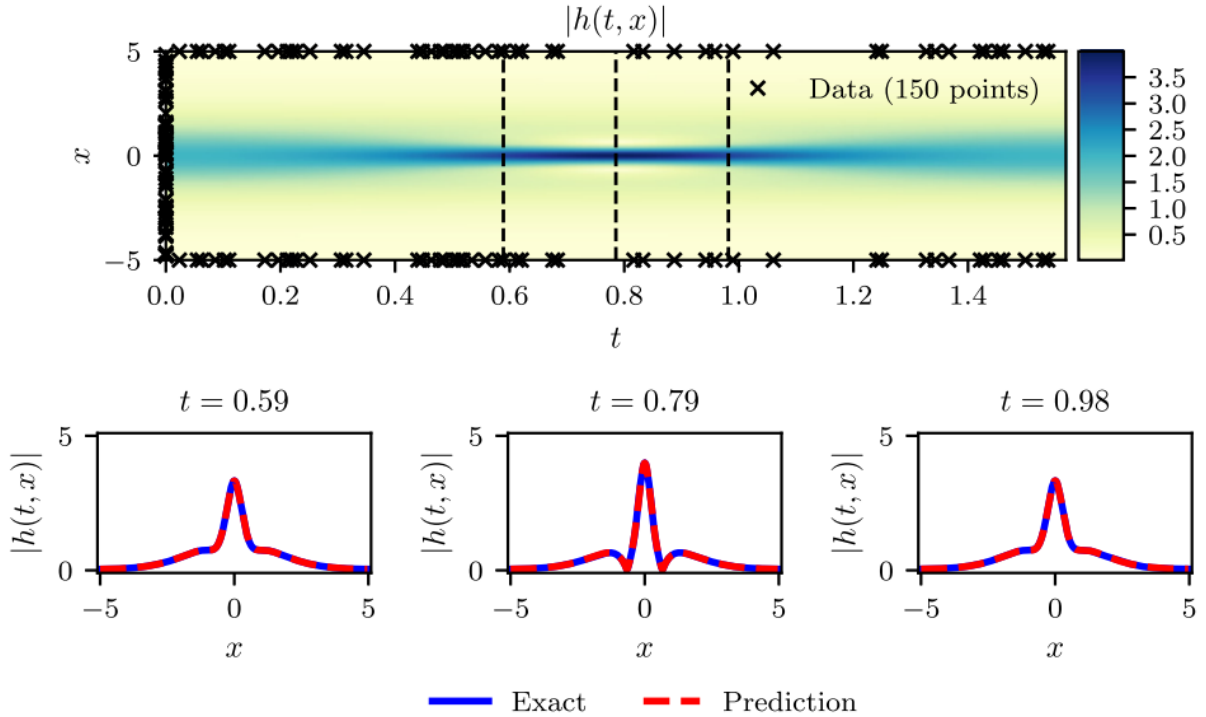


Fig 6: Top: Predicted Solution $|h(t, x)|$ along with initial and boundary training data
Bottom: Comparison of predicted and exact solutions for three temporal snapshots [1]

[5] demonstrated usage of PINN for solving a non-linear diffusivity equation:

$$f(x, t) = \phi c_t \frac{\partial p}{\partial t} - \kappa_o \frac{\partial}{\partial x} p^2 \left(\frac{\partial}{\partial x} p \right) - g \quad (11)$$

where ϕ = initial porosity, c_t = total compressibility, p = fluid pressure, κ_o = hydraulic conductivity (scalar), g represents sink/source, $\theta = (\phi, c_t, \kappa_o)$ is PDE parameter set

The system with domain $\Omega = [0, 1]$ and $T = [0, 1]$ was subject to simple conditions:

$$\text{IC: } p = p_0 \text{ in } \Omega \text{ at } t = 0 \quad \text{and} \quad \text{BC: } p = p_D \text{ on } \partial\Omega_p \times T$$

The number of hidden layers (N_{hl}) and the number of neurons in each layer (N_n) were taken as hyperparameters varied over a range of values with fixed number of training ($N_b = 96$) and collocation ($N_n = 160$) points yielding least error for $N_{hl} = 6$ and $N_n = 5$. This was an example of tuning PINNs. Latin Hypercube was used for sampling and loss was minimised using L-BFGS.

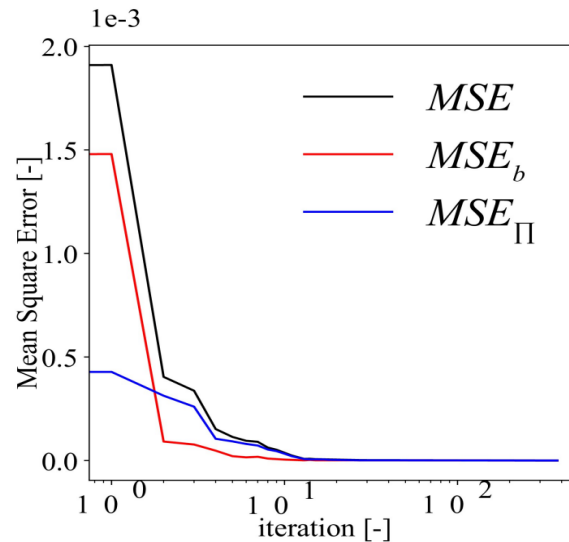


Fig 7: Variation of MSE loss (split into training & collocation) with iterations [5]

There are many examples of solving problems using PINN, however, without any uniformity in the PINN design. There doesn't really exist any theory for the structure of PINN for solving a given problem. Either an arbitrary procedure is applied or some heuristics are identified and utilised or an experimental validation is performed.

In order to come up with the most appropriate PINN architecture that fits the data and follows the PDE, several values are possible for various modifiable parameters (called hyperparameters) which affect the training process. There could also be minor modifications in the approach or methodology for different aspects associated with PINNs. Varying any of them will result in a different PINN solution for a given problem. These include data sampling strategy, neural network architecture details, loss function design, and optimisation strategy. There, thus naturally arises the question of whether any unique PINN design exists to optimally solve a particular problem. Answering it requires an investigation into these design aspects of a PINN solution. An exercise is performed wherein several PDEs are solved using the PINN framework: varying the NN design while keeping the other aspects of the solution strategy fixed as identified to be best from the insights of such an investigation. Hyperparameter tuning is performed to determine the optimal PINN design for each PDE (based on the least loss obtained on validation data). Further, it is intended to figure out whether any special relationship exists between the characteristics of the given problem (PDE) and its optimal PINN structure. Discovery of any such relationship would eliminate the need to experiment with PINN designs to solve a problem which would be helpful as the training of each model is time-consuming - one of the major drawbacks of PINNs. This would provide a basis for one to rapidly engineer a solution for any new PDE. We aren't aware of any studies performed in this direction, to the best of our knowledge.

1.2. Design Aspects of PINN

1.2.1. Data Distribution and Sampling

In this study, data has been sampled randomly from a uniform distribution for simplicity. In the literature, the training data associated with the conditions are generally few scattered observations (here 50 initial and 50 boundary points are taken) while a greater number of collocation points (here 10000) are chosen from the domain to evaluate the residual. Additionally, here, for model validation, data is sampled in 1:5 ratio with respect to training.

1.2.2. Loss Function Design

The general loss function defined in Fig 5 is for the case of enforcing soft-boundary constraints, meaning the neural network learns to fit the training data for the conditions gradually but the conditions are satisfied only approximately. To satisfy the conditions exactly, hard-boundary constraints can be applied apriori by encoding them within specific components of the network as done for periodic boundary conditions in [6] and with distance functions in [7]. Here, soft-boundary constraints are employed via loss-term for each condition as defined in Eq. 6. Further, there can be weights associated with the different loss terms as hyperparameters which would affect the learning efficiency, and determining them is a complexity to be addressed. For flexibility though, the weight associated with the PDE residual will help to vary the fidelity of the PDE model i.e. the balance between the data-fit and physics-obeying constraint. However, it is ignored for now as it poses an additional hyperparameter optimisation problem.

1.2.3. Optimisation

The PINN loss function in most literature is optimised via minibatch sampling using Adam or the limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) algorithm which will yield one of several local minima. Here, the optimizer is chosen to be Adam as a straightforward implementation for it is available in the TensorFlow library. Adam, which combines adaptive learning rate and momentum methods is known to have higher convergence speed compared to SGD and had also resulted in lower loss while solving the Burgers' Equation. A decreasing learning rate is popular for faster convergence away from minima (and reduces oscillations when nearby) while the stopping criteria could be a fixed number of iterations or a lower bound on the loss. If the model by itself is good enough for solving the problem, a better optimisation scheme will help converge to the solution faster. In order to compare different models, here the optimisation strategy is kept fixed. The neural network model parameters thus are the most significant hyperparameters, which need to be optimised before other aspects can be explored to get the most accurate solution. Here, each model is trained for fixed 5000 epochs with the learning rate decaying with iteration n in a piecewise constant fashion as in Eq 12:

$$\delta(n) = 0.01 * 1_{\{n < 1500\}} + 0.001 * 1_{\{1500 \leq n < 3500\}} + 0.0005 * 1_{\{3500 \leq n\}} \quad (12)$$

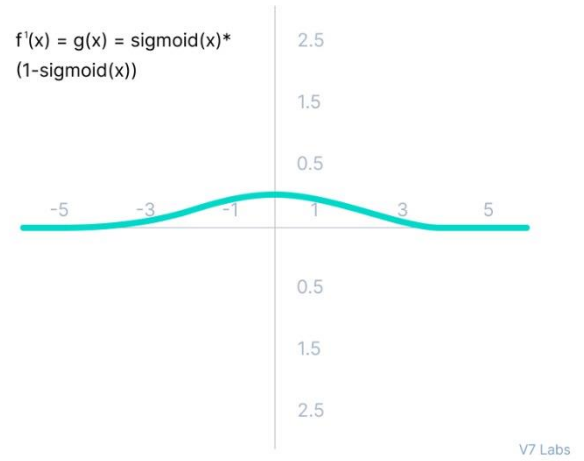
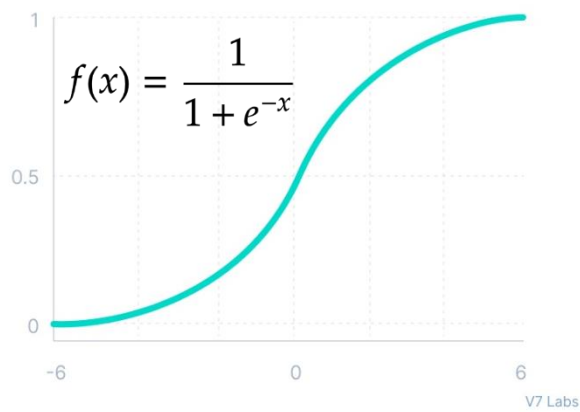
1.2.4. Neural Network Architecture

Most of the PINNs in literature are single feed-forward neural networks. There have been instances of wide networks (say 100 neurons per layer as in [1]) as well as deep networks (say >5 layers). Intuitively, it seems more hidden layers are required to model complicated non-linear relationships and an attempt is made to formally address this in the next section but having more layers would result in high training costs and efficiency issues (in terms of computing time and memory). Hence, shallow networks (say <5 layers) have also been used. In fact, sparse architectures designed using kernel networks and yielding interpretable results (SPINNs) were proposed in [8] which act as a bridge between traditional meshless methods and deep neural networks for solving PDEs. There also have been instances of the use of multiple fully-connected networks mostly to model individual equations or solution regions which are blended together to solve a larger model or domain. These include [9], [10] and [11].

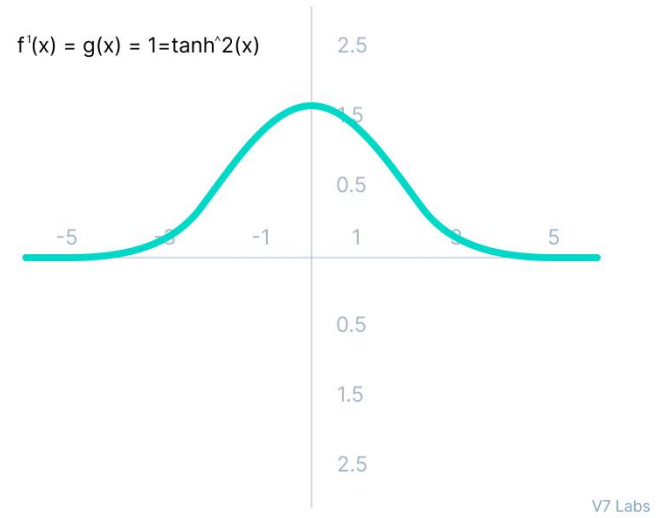
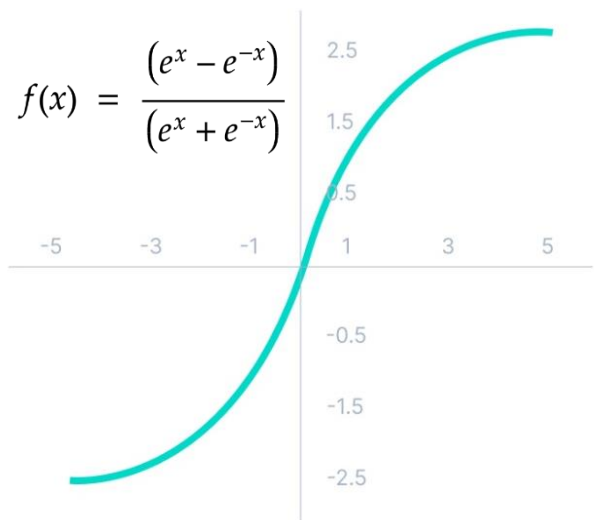
Here, a single fully-connected feed-forward neural network is chosen with the number of hidden layers (h) and the number of neurons in each layer (n) taken as hyperparameters for tuning. h is allowed to be any integer from 1 to 20 while n takes up a multiple of 4 (powers of 2 are expected to result in faster convergence via matrix multiplications performed by the computing hardware) in the range [4, 256]. For combinations of h and n , which result in big networks with many trainable parameters, there is a possibility of overfitting. To compensate, L2 weight regularizer with penalty value among {0, 10E-1, 10E-2, 10E-3, 10E-4, 10E-5, 10E-6} and dropout fraction among {0, 0.1, 0.2, 0.3, 0.4, 0.5} is considered for each layer and taken as a hyperparameter. The dropout value denotes the fraction of neurons which would be randomly deactivated in each training iteration resulting in a sparse or less-complex model. Whereas, L2 weight regularisation aims to reduce the value of weights by penalizing them via additional L2-norm loss term (penalty coefficient multiplied by the square of the weights) so that the predicted output is not too sensitive to the input.

Activation functions are the nonlinear transformations introduced in every layer of the neural network and functions such as ReLU, Sigmoid, Tanh, and Swish are found in the PINN literature with the infinitely differentiable Tanh being the most popular. Here, the activation function (a), same for every layer, is taken as a hyperparameter. The weight initialisation scheme is crucial to determine the convergence rate of training and along with the activation, it plays a major role in any vanishing or exploding gradient issues with the network. ReLU is considered to be relatively robust to such issues and earlier studies have shown He weight initialiser works best with it. For other activation functions, Glorot weight initialiser is assumed and the type of distribution (Normal or Uniform) from which the initial weights are sampled is taken as a hyperparameter. See Fig 8 for more detailed analysis on the activation functions.

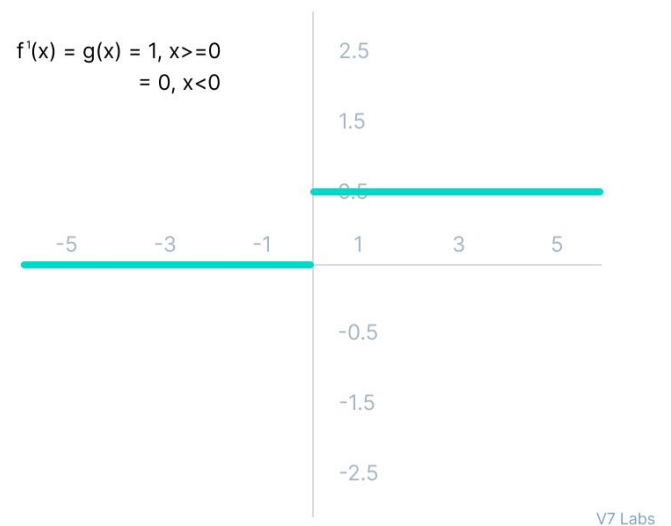
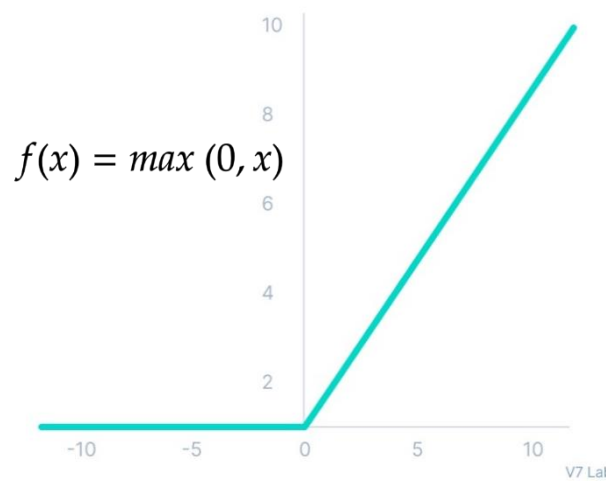
Sigmoid / Logistic



Tanh



ReLU



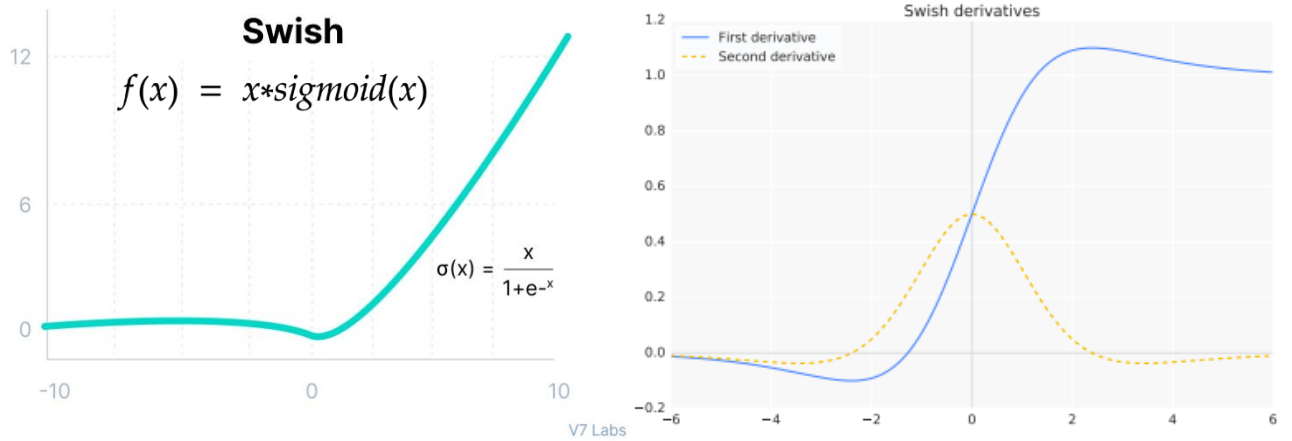


Fig 8: Graphs of activation functions (Sigmoid, Tanh, Relu, Swish) and their derivatives.

Notice that the derivative of sigmoid and tanh is very low beyond $|x| = 3$ and zero in case of ReLU for $x < 0$ (which is the vanishing gradient case). While derivative of swish drops for $x < 5$. Sigmoid, Tanh, Swish are smooth functions while there is discontinuity in derivative of ReLU. Output of sigmoid is always positive while output of ReLU is zero for negative input, which might affect the training ability of model although this sparsity in ReLU makes it more efficient. Unlike sigmoid, tanh is zero centered and the gradients are not restricted to move in a certain direction, therefore tanh is always preferred to sigmoid. ReLU is preferred over sigmoid and tanh in hidden layers because it is less susceptible to the vanishing gradient issue. Swish has consistently matched or outperformed ReLU on many tasks. [21], [22]

1.2.5. Error Analysis

The accuracy of a PINN can be measured in terms of the error which can be decomposed into three terms: approximation error, optimisation error, and the generalisation error. The approximation error is the error associated with the neural network approximation and this can be measured wrt the available exact values. As the loss function is non-convex, optimisation will not always yield the global minimum. The error associated with this is the optimisation error which can't be determined. The error measured in the predictions by a trained network wrt unseen data is the generalisation error. Here, the MSE loss on labelled data recorded during training is the approximation error while the MSE loss evaluated on domain grid (validation data) is an estimate of the generalisation error. Bounds need to be established on these errors to design PINNs with sufficient accuracies. For now, we don't venture into error evaluation explicitly, rather, take the recorded loss as a representative of the error.

1.2.6. Generalisability

The generalisation aspect in terms of the capability of a single model to solve a variety of equations, say a family of PDEs needs to be investigated. In this regard, in [12], researchers proposed the DeepONet framework which is based on the operator approximation properties of neural networks. It consists of two sub-networks: one called branch-net for encoding the input functions and another called trunk-net for encoding the locations of the output functions. Once trained on inputs and outputs for a set of PDEs, it will have learnt the operator for that family of PDEs. So, if it is provided

with data representing a new PDE (belonging to that family), it yields data representing the solution to that PDE. [23] This operator learning framework has been demonstrated for predicting multiscale bubble growth dynamics in [13]. However, for this study, we stick to the PINN framework originally proposed in [1].

1.3. Hyperparameter Tuning using Bayesian Optimisation

A hyperparameter is a parameter whose value is used to control the learning process of the model in contrast to other parameters (typically the weights of the neural net for example) whose value is learnt. In order to tune the model so that it yields the best results to serve the purpose, it is essential to optimise the value of these hyperparameters. Heuristics as mentioned earlier were used to reduce the high-dimensional search space for hyperparameter optimisation which is another complexity given slow training itself being an issue in PINNs. While grid search becomes computationally impractical in evaluating models for all values of several hyperparameters, random search could be a possible option. However, a more subtle approach that takes into account results after every iteration to optimise the search process is desirable. The gaussian-based bayesian optimization (BO) is one such technique. It is a sequential design strategy for the global optimization of black-box functions. [24] If we have input-output pairs (data) but have no knowledge of the relationship between them, then we can consider it as a black-box function and if we intend to maximise the output, we can apply BO. It works by treating it as a random function and places a prior over it which captures its behaviour. After knowing the function evaluations, it updates the prior to form a posterior distribution, which is improved as the number of observations grows with iterations resulting in a more restricted parameter search space each time. It also provides flexibility to the user to decide how to balance random exploration with the exploitation of learnings from the results of prior iterations. It has been used for hyperparameter tuning of PINNs such as in [14] for solving Helmholtz problems. They used 5 hyperparameters: learning rate for optimisation $[10E-4, 5E-2]$, number of NN layers $[1-10]$, number of neurons in each layer $[5,500]$, activation function $\{\sin, \text{sigmoid}, \tanh\}$, and the weight associated with the boundary loss term $[1, 10^7]$. From the results, they concluded that wide shallow networks with sin activation perform better at solving the Helmholtz problems. Here, we consider the hyperparameters and their possible values as mentioned in Table 1. In every iteration of BO, a combination of the hyperparameter values is chosen as the input and the PINN is trained while the negative of the loss on validation data is considered as the output to be maximised. With access to this data, the BO updates its understanding of the relationship between the hyperparameters and the loss (which quantifies the goodness of our PINN model) and accordingly chooses the next set of hyperparameters to train the PINN. This is why, BO is popularly used for getting the best possible hyperparameters for expensive-to-evaluate functions and thus it is expected to be helpful here too as PINNs are slow to train. Here, 10 iterations of BO were performed with an additional 5 for random exploration.

Hyperparameter	Symbol	Range/Set of Values
No of Hidden Layers	h	Integer in [1, 20]
No of Neurons	n	Multiplier of 4 in [4, 256]
Dropout Fraction	d	{0, 0.1, 0.2, 0.3, 0.4, 0.5}
Activation Function	a	{relu, sigmoid, tanh, swish}
Weight Initialiser	wi	If ReLU then He else Glorot {normal, uniform}
L2 Weight Penalty	wr	{0, 10E-1, 10E-2, 10E-3, 10E-4, 10E-5, 10E-6}

Table 1: Search space for Hyperparameter Tuning

1.4. Example: Linear Convection Diffusion Equation

This is a linear partial differential equation representing physical phenomena involving both convection and diffusion. Let $u(t,x)$ (temperature) be the dependent variable under consideration with x (position) and t (time) being the independent variables. Then the following PDE is the linear convection diffusion (LCD) equation.

$$\frac{\partial u}{\partial t} + v \frac{\partial u}{\partial x} - c \frac{\partial^2 u}{\partial x^2} = 0 \quad \text{with } c = \text{diffusion coefficient} = 1 \text{ and } v = \text{velocity} = 1 \quad (13)$$

Consider the domain: $x \in [-1, 1]$ and $t \in [0, 1]$

Let the initial condition (IC) be: $u(0,x) = -\sin(\pi x) = f(x)$

Let the boundary condition (BC) be: $u(t,-1) = u(t,1) = 0$

Design a neural network in `tf.keras` to model $u(t, x)$ given x and t

Neural Network Architecture (with hyperparameters h,n,a,w_i,w_r,d chosen by BO)

- Input Layer = Two Neurons for x and t
- h Hidden Layers each with
 - Number of Neurons = n
 - Activation Function = a
 - Weight Initialisation Scheme = w_i
 - L2 Weight Regularisation Penalty = w_r
 - Dropout Fraction = d
- Output Layer = One Neuron for u

The data sampling strategy and learning rate schedule is same as in Chapter 1

Pseudocode for Training PINN (repeated for every combination of hyperparameters chosen by the bayesian optimisation algorithm, here its 15 times)

Repeat (for every iteration upto 5000):

- For each data sample x , approximate u using PINN
- Calculate Loss = MSE over initial & boundary data + PDE residual
- Use Adam to minimise loss
- Update NN model weights

The least validation loss was recorded for the setting $h = 7$, $n = 68$, $w_r = 0$, $d = 0.5$, $w_i = \text{glorot normal}$, $a = \text{swish}$. Training the PINN for 5000 epochs with these settings took ~31 mins on Google Colab GPU yielding:

Final Training Loss = $4.51\text{E-}05$ with the break-up being loss on IC data = $2.16\text{E-}05$, loss on BC data = $9.89\text{E-}06$ and loss on collocation data = $1.37\text{E-}05$

Final Validation Loss = $5.38\text{E-}05$ with the break-up being loss on IC data = $3.47\text{E-}05$, loss on BC data = $5.10\text{E-}06$ and loss on collocation data = $1.40\text{E-}05$

The loss curves are plotted in Fig 9, they indicate good generalisation

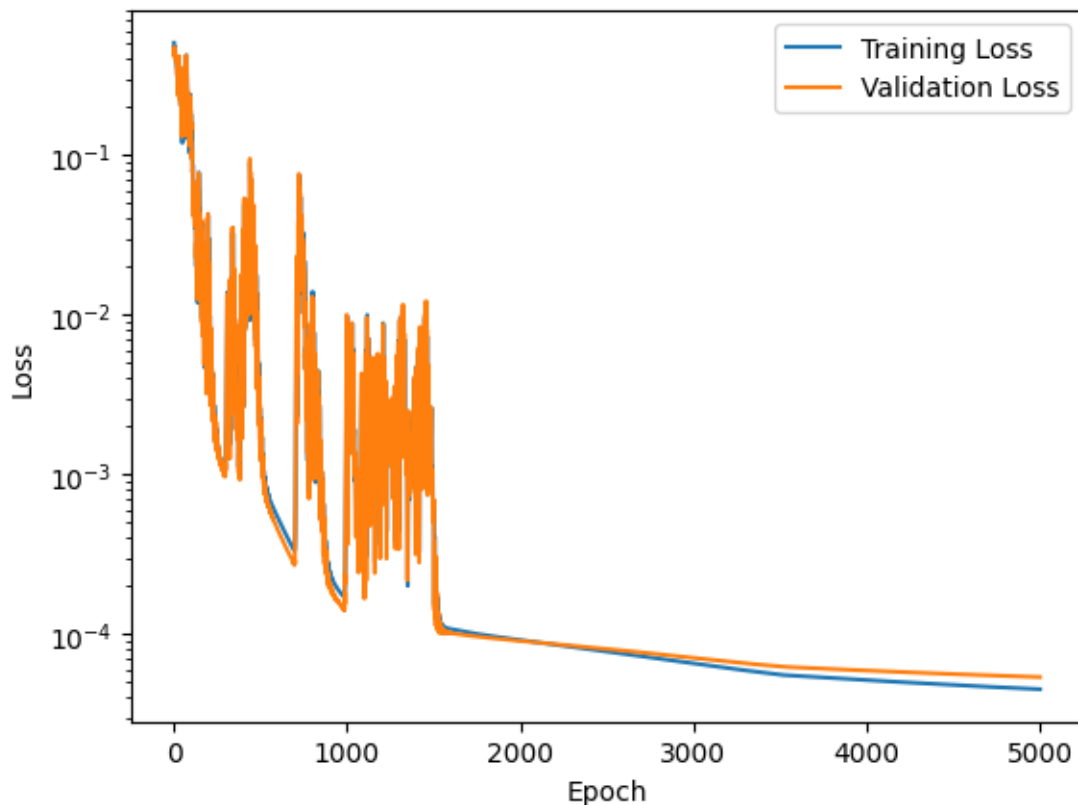


Fig 9: The training and validation loss curves wrt epochs are close to each other

To evaluate the PINN predictions lets take a grid of 60 points in the domain of x and 10000 points in the domain of t and compare with the numerical and analytical solutions.

Equation 13 is a simple problem to solve numerically, take the explicit finite difference method (Euler's) for example, which took just ~0.7 secs for computation.

The analytical solution for a 1D advection equation with Dirichlet boundary conditions is available at [25]. Applying the transformation $x \rightarrow x + L/2$ to match the BC considered here without changing the IC function yields the analytical solution to Equation 13. As it is in the form of an infinite series, a 100 terms approximation is taken for computation.

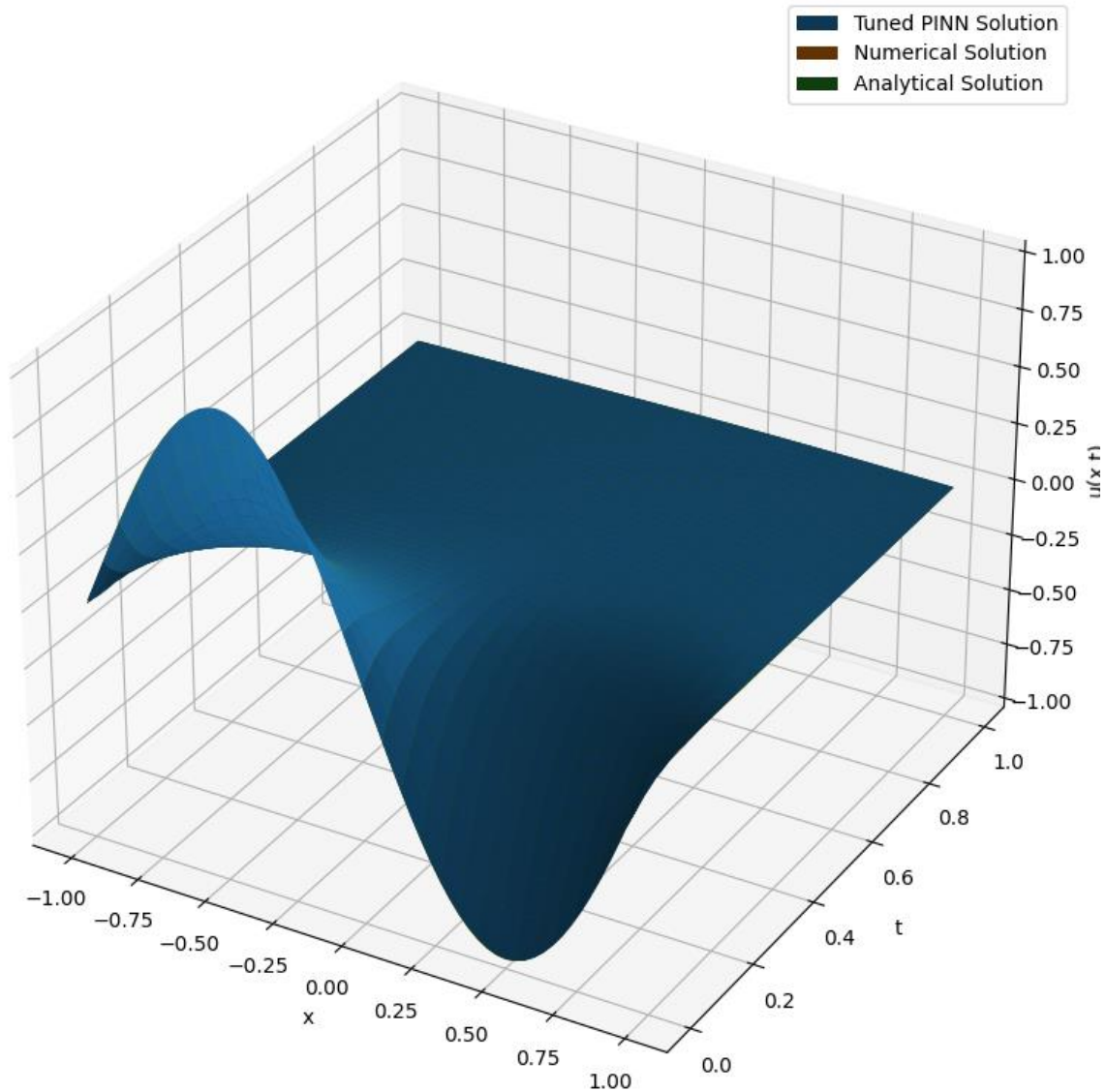


Fig 10: Tunned PINN prediction $u(t, x)$ plotted along with the numerical and analytical solution

Visually there seems to be a good match between the 3 solutions as seen in Fig 10 and Fig 11. The MSE between the analytical and numerical solution = $3.49E-09$ while the MSE between the PINN prediction and analytical solution = $2.23E-06$. This clearly indicates that for solving simple problems, the numerical methods are more accurate and faster compared to the PINN technique.

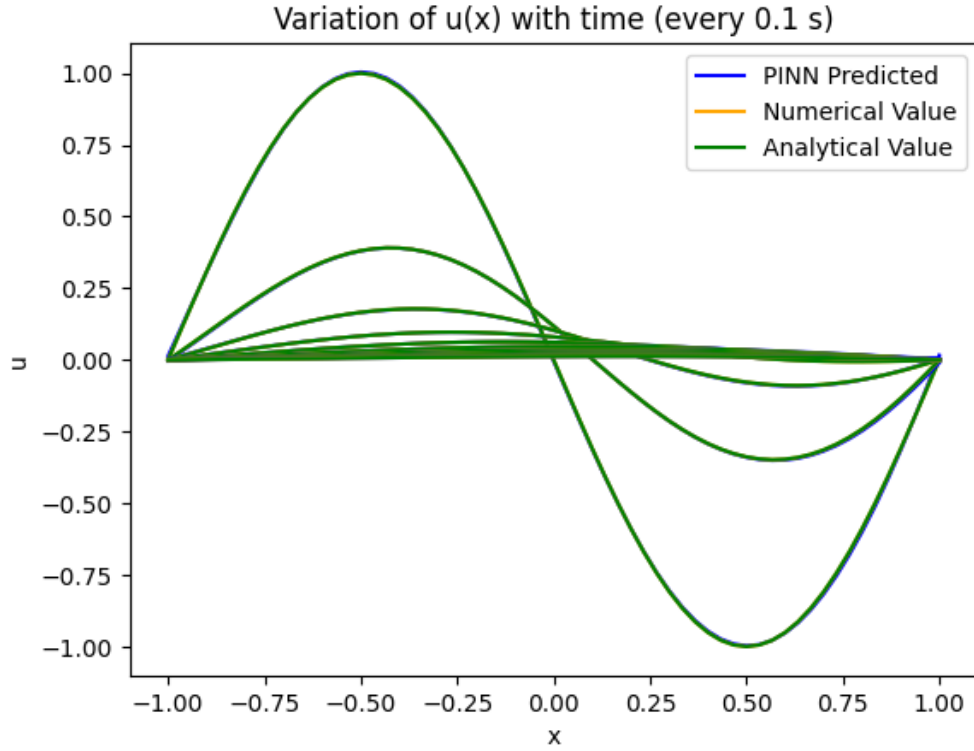


Fig 11: Graphs showing $u(x)$ variation with time (2D snapshots of the 3D plot in Fig 10)

1.5. Learning the PINN Design from PDE

Hyperparameter tuning was performed for 9 different PDEs keeping the IC and BC to be same as in the case of LCD equation earlier. For each PDE, information such as the type (linear L or non-linear NL), order, degree, number of variables and terms in the PDE, number of operations performed in the equation and the optimal values found for the hyperparameters are recorded as in Table 2.

PDE Information							PINN Design					Val. Loss
Equation	Type	Variables	Terms	Ops	Order	Degree	Layers	Activation	Dropout Rate	Weight Initialiser	L2 Penalty	
$u_t + u_x - u_{xx} = 0$	L	2	3	6	2	1	7	Swish	0.5	GN	0	$3.35 \cdot 10^{-5}$
$u_t + u_x = 0$	L	2	2	3	1	1	1	ReLu	0	HU	0	$1.15 \cdot 10^{-1}$
$u_t - u_{xx} = 0$	L	2	2	4	2	1	5	Swish	0	GN	10^{-6}	$2.23 \cdot 10^{-6}$
$u_t + uu_x - u_{xx} = 0$	NL	2	3	7	2	1	14	Swish	0.5	GN	0	$6.94 \cdot 10^{-5}$
$u_t + uu_x = 0$	NL	2	2	4	1	1	7	Swish	0.5	GN	0	$4.76 \cdot 10^{-2}$
$u_t^2 - u_{xx} = 0$	L	2	2	7	4	1	16	ReLu	0	HU	10^{-4}	$3.89 \cdot 10^{-7}$
$u_t - u = 0$	L	1	2	2	1	1	4	Swish	0	GU	0	$1.56 \cdot 10^{-5}$
$u_t^2 - u = 0$	NL	1	2	3	1	2	11	Swish	0	GN	10^{-6}	$5.60 \cdot 10^{-2}$

Table 2: Information related to the PDE and the optimal PINN design for each of the 9 PDEs

The following general observations can be noted from Table 2 and Table 3:

1. The optimal number of hidden layers increases to capture complexities such as the presence of more terms, non-linearity, double-differentials in the PDE
2. The choice of the number of neurons isn't as much uniquely critical as compared to the number of hidden layers in solving the PDE
3. Using Swish activation results in a higher performance as compared to others
4. For deeper networks, dropout gets enabled possibly to avoid overfitting
5. Regularisation is applied if the model has a very high number of weights

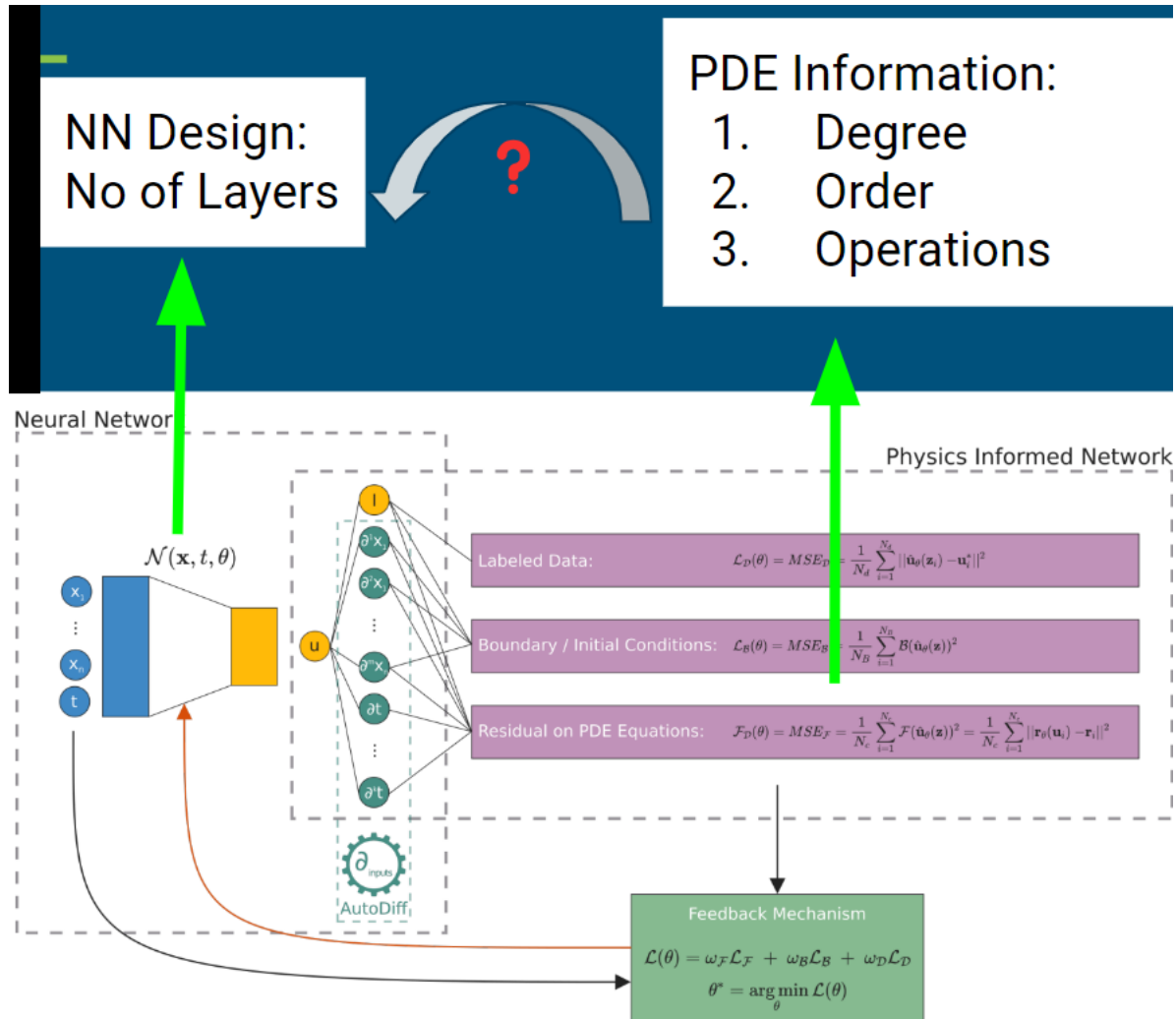


Fig 12: Can we uniquely determine the PINN design from the features of the PDE?

Based on the first observation, it can be hypothesised that a relationship might exist between the nature of the PDE and the number of hidden layers in the optimal PINN design for solving the PDE. Thus, a machine learning model can be developed fitting the number of hidden layers (marked in dark green – the dependent variable) with the quantitative information defining the PDE (marked in dark yellow – the independent variables) to formally check it. The relationships may not be necessarily linear, hence, polynomial features (restricted to degree 2) over the independent variables are taken.

A linear regression model with L1 (Lasso) regularisation is built. Lasso applies a penalty coefficient over L1-norm of parameters, which shrinks to zero for a predictor irrelevant to the target variable thereby acting as a feature eliminator. The coefficient of determination for the trained model is $R^2 = 0.9283$ which indicates a strong fit. The following equation can be constructed from the model coefficients to express layers:

$$L = Operations * (0.77 * Type - 0.13 * Variables + 0.47 * Order + 0.37 * Degree) + 0.07 * Degree^2 - 0.38 * Variables^2 \quad (14)$$

This equation highlights a probable relationship to determine the optimal number of layers based on 5 features of the PDE: type (0 for L, 1 for NL), number of variables, number of operations, order and degree. It indicates that the number of operations in the equation most closely affect the number of layers followed by its degree. The general assumption that more layers are needed to model higher complexity (noted from equation in the form of a greater number of operations, non-linear nature, higher order and degree) seems to remain valid as seen by the positive signs. The negative signs of the variables terms may imply that more the input variables for the PINN, better will be its predictive capability and less complex structure it will require.

1.6. Are the number of layers enough to solve a PDE?

A natural question might arise from the above exercise. Are the number of hidden layers in PINN independently alone enough to specialise in solving a particular PDE optimally? For this, we again perform tuning of the PINN for 7 of those PDEs but this time fix the hyperparameters other than the number of hidden layers and record the results in Table 4. The values of the fixed hyperparameters are computed by either taking mean or based on popularity as noted in Table 3.

PINN Optimised Parameters for PDEs from previous experiment				
Neurons	Activation	Dropout Rate	Weight Initialiser	L2 Penalty
68	Swish	0.5	Glorot Normal	0
44	ReLu	0	He Uniform	0
256	Swish	0	Glorot Normal	1.00E-06
44	Swish	0.5	Glorot Normal	0
36	Swish	0.5	Glorot Normal	0
248	ReLu	0	He Uniform	1.00E-04
44	Swish	0	Glorot Uniform	0
40	Swish	0	Glorot Normal	1.00E-06
100	Averaging (values) & popularity (categories) used to fix parameters			

Table 3: The fixed values of the 5 hyperparameters decided based on the previous exercise

PDE Information							PINN	Val. Loss
Equation	Type	Variables	Terms	Ops	Order	Degree	Layers	
$u_t + u_x - u_{xx} = 0$	L	2	3	6	2	1	8	$4.87 \cdot 10^{-5}$
$u_t + u_x = 0$	L	2	2	3	1	1	7	$2.57 \cdot 10^{-3}$
$u_t - u_{xx} = 0$	L	2	2	4	2	1	7	$3.83 \cdot 10^{-6}$
$u_t + uu_x - u_{xx} = 0$	NL	2	3	7	2	1	8	$2.69 \cdot 10^{-5}$
$u_t + uu_x = 0$	NL	2	2	4	1	1	3	$6.92 \cdot 10^{-2}$
$u_t - u = 0$	L	1	2	2	1	1	6	$5.42 \cdot 10^{-6}$
$u_t^2 - u = 0$	NL	1	2	3	1	2	8	$5.49 \cdot 10^{-2}$

Table 4: Tuning just the number of layers in PINN for each of the 7 PDEs

From Table 4, it is clear that no relationship exists between the number of layers and PDE information. So, the design of PINN has to be significantly different for each problem. The best model for each PDE occurs only for an optimal set of hyperparameter values with the number of layers being more critical to the design.

However, a more intelligent potential future exercise could be to model the relationship between the PINN design and the characteristics (for example, number of terms and operations like earlier) of the loss function which gets created for any general problem (and not just a single PDE). As any neural net training minimises the loss function itself, it is expected that a more complex network will be required if the loss optimisation is difficult i.e. the problem is more complex. A loss function more directly governs the learning process than how the PDE appears at the outset.

2. Solving Complex Problems using PINN

2.1. Introduction

There have been an increasing number of papers on PINNs addressing problems of increasing complexities over the years starting from the 1D Schrodinger's Equation (SE) in 2018, followed by Euler Equation (EE) to model high-speed aerodynamic flows, incompressible Navier-Stokes Equation (NSE) and the coupled system of NSE and temperature equation to analyse heat flow convection (NSE+HE) in 2021.

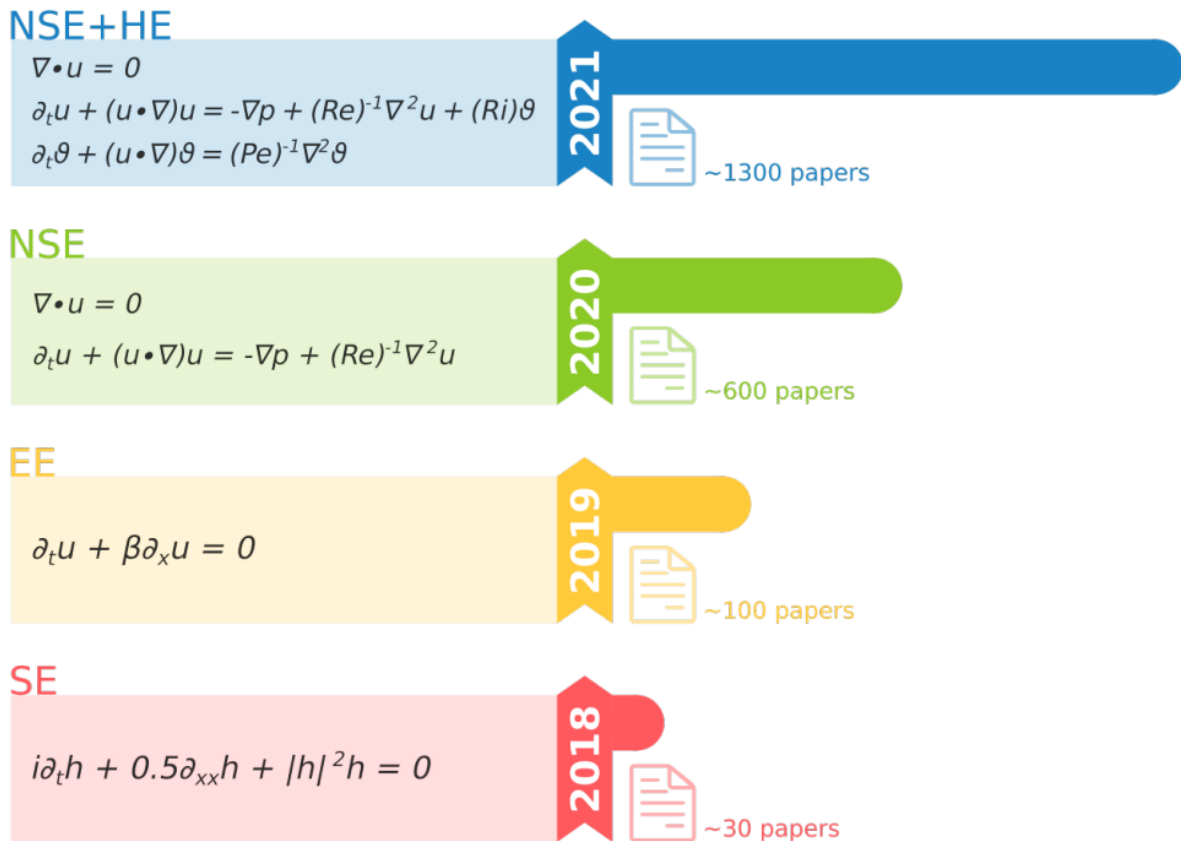


Fig 13: History snapshot of problems solved using PINN in literature

PINNs with some improved strategies have also been used to solve a system of coupled equations such as the Stokes-Darcy equation as in [11]. Initially, all loss terms corresponding to both the equations are simply added to make up the final loss function and the PDE is solved using a single PINN. However, if the parameters of the PDE are too small or if there is a discontinuity at the interface, then the PINN doesn't work well for solving the PDE. A discontinuity would imply that the solutions of the two regions on either side of the interface will be very different, and loss function optimisation will become very difficult using a single network to simulate in the entire domain.

Hence, the authors suggest some strategies to tackle such issues. These include adding weights to the different loss terms. This ensures that just because of the magnitude of the loss values, only certain specific terms aren't focussed upon for training. Another strategy is to use parallel network architecture where each network caters to a different sub-region of the solution. This helps improve interface discontinuity-related issues if there is a separate network for either side of the interface. As can be seen in Fig 14, the error in prediction vs exact solution for variable V present in the coupled Equation 15 (here for Stokes region) was lower when 2 networks were used instead of 1 network. Finally, it was suggested to have a layer-wise locally defined activation function so that the layers have different activations to capture the different aspects of complex problems.

$$\begin{aligned} 2\nu \mathbf{n}_s \cdot D(\mathbf{u}_s) \cdot \mathbf{n}_s &= p_s - p_d + g_1, \\ 2\mathbf{n}_s \cdot D(\mathbf{u}_s) \cdot \boldsymbol{\tau} &= -\alpha K^{-1/2} \mathbf{u}_s \cdot \boldsymbol{\tau} + g_2, \end{aligned} \quad (15)$$

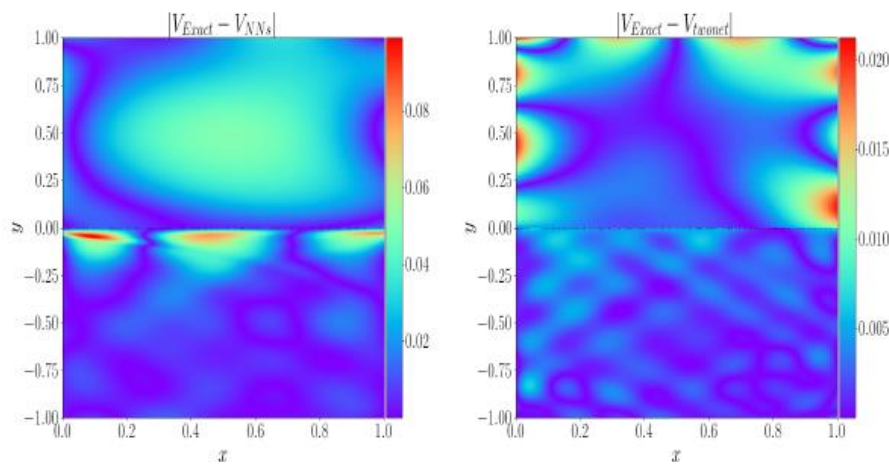


Fig 14: 2D graphs (y vs x) to visualise error magnitudes between the PINN prediction and exact solution across the interface at $y=0$ (1 network on left and 2 networks on right)

In [9], PINN is used for the simulation of the thermochemical evolution of a composite tool material undergoing curing in an autoclave. They design a PINN with two disconnected sub-networks to model the different behaviours of the two differential equations in the coupled system: thermal conduction (heat transfer) and resin cure kinetics. Their PINN design is illustrated in Fig 14.

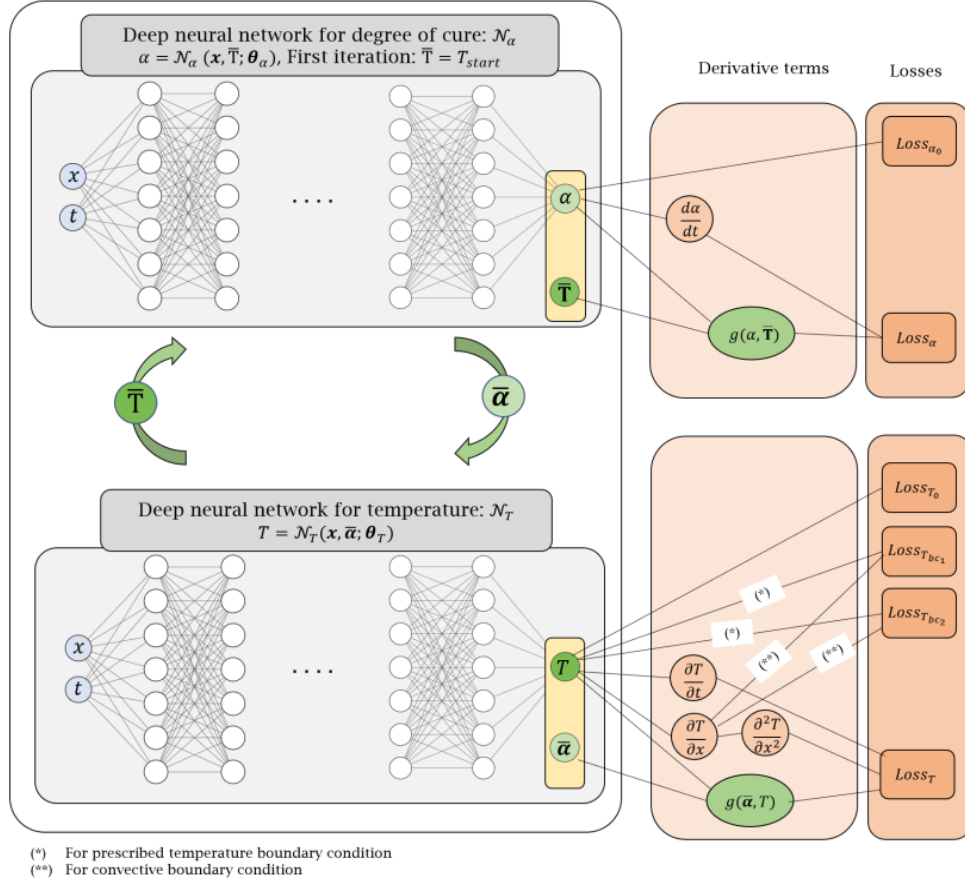


Fig 15: Two separate networks for determining the degree of cure (α) and temperature (T). The PINN is trained sequentially, first on α and then on T , until convergence. [9]

An instance of using PINN for defect detection was found in [10] where it was employed to identify defects using pulsed thermography images of CFRP specimens. A second neural network was used as a surrogate model for determining the second-order thermal gradients (present in the PDE) for subsurface temperatures which weren't measured during experiments. The PINN reconstructs the thermogram which after background elimination reveals the anomalies present.

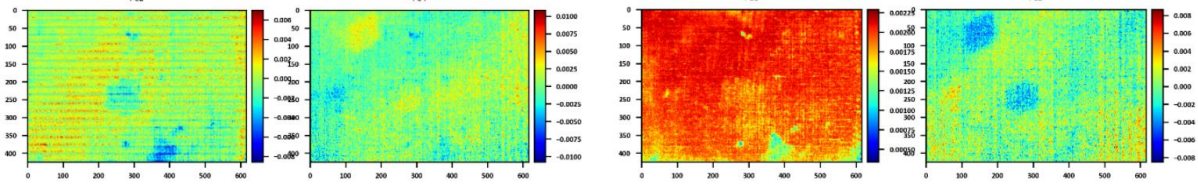


Fig 16: Predicted images (right 2) with the three defects as in original images (left 2) [10]

In the following sections, problems with increasing complexity are taken up such as the nonlinear Burgers' Equation, the 2-phase Stefan's problem, the directional solidification problem and a binary alloy solidification problem. It is shown how such systems involving shocks, steep gradients, discontinuities, moving interfaces etc. can be solved to a good extent within the PINN framework.

2.2. Burgers' Equation

The Burgers' Equation is a Partial Differential Equation (PDE) commonly encountered in various areas of applied mathematics such as traffic flow, fluid mechanics, gas dynamics, and non-linear acoustics. It is given by:

$$f(t, x) = u_t + uu_x - vu_{xx} = 0 \quad (16)$$

where $u(t, x)$ = speed of fluid, u_t represents transiency, u_x represents convection and u_{xx} represents the diffusion term with coefficient v (= viscosity taken as $0.01/\pi$)

It is valid in the domain $x \in [-1, 1]$ and $t \in (0, 1]$ subject to the conditions:

$$\text{Initial Condition: } u_0 = u(0, x) = -\sin(\pi x) \quad (17)$$

$$\text{Boundary Condition: } u_b = u(t, -1) = u(t, 1) = 0 \quad (18)$$

This problem contains non-linearity (in term uu_x), has a small parameter $v = 0.00318$, and its solution is expected to have steep gradients making it a fairly complex problem compared to a linear PDE such as Equation 13. It was solved in [15] and we attempt to replicate their solution here.

Design a neural network in tf.keras to model $u(t, x)$ given x and t

Neural Network Architecture

- Input Layer = Two Neurons for x and t
- 8 Hidden Layers each with
 - Number of Neurons = 20
 - Activation Function = Tanh
 - Weight Initialisation Scheme = Glorot Normal
- Output Layer = One Neuron for u

The data sampling strategy and learning rate schedule are the same as in Chapter 1. The training procedure is the same as in Section 1.4. The training (5000 epochs) took ~1 min on Google Colab T4 GPU (~5 mins on CPU) resulting in final training loss = $2.03\text{E-}04$ and validation loss = $1.96\text{E-}04$ with the loss curve variation with epochs shown in Fig 17 indicating smooth learning and good generalisation.

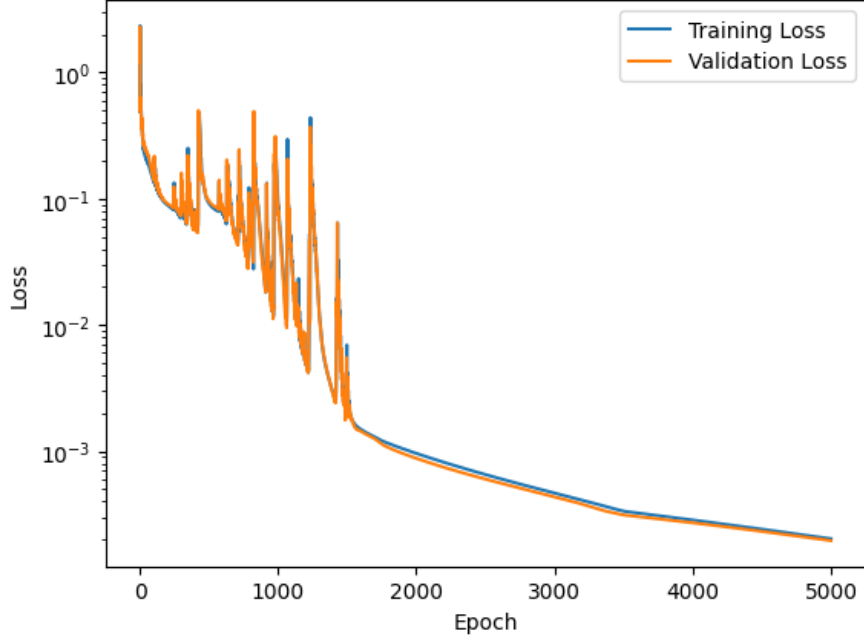


Fig 17: Training and Validation Loss Curves vs Epochs

The trained PINN was used to predict u over domain grid (500 x 500) of x & t points:

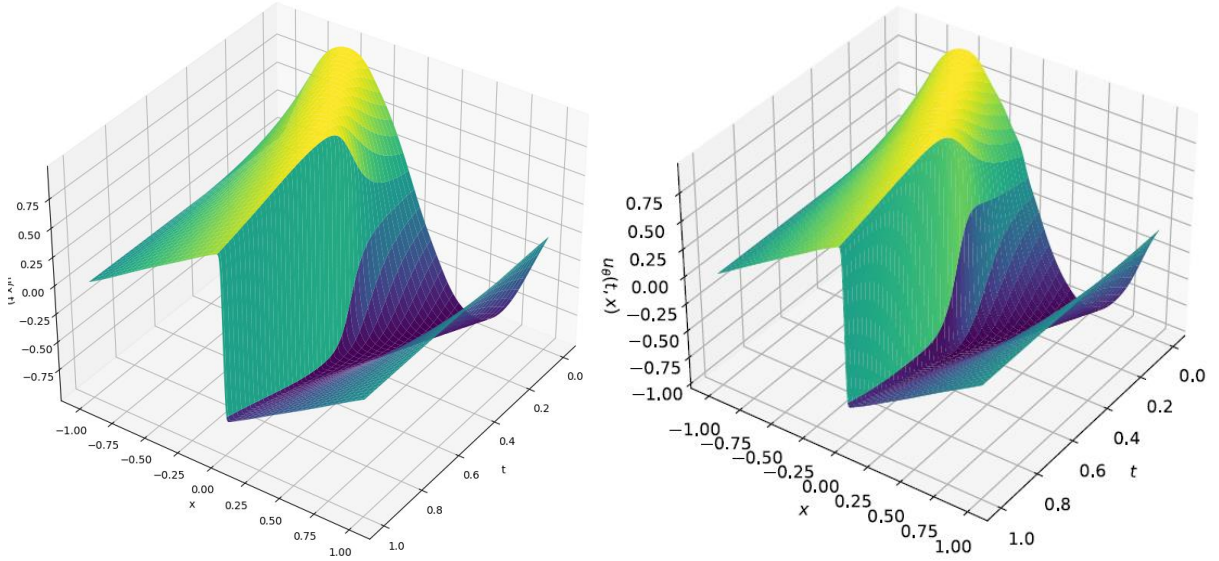


Fig 18: PINN predicted BE solution $u(t, x)$ plotted over the domain grid (L). Shock formation begins at $t = 0.4s$ and ends up in a steep gradient at $x = 0$ similar to as noted in [15] (R).

The 2 plots in Fig 18 are similar not same. Note that the one on left was trained only for 5000 epochs and is still learning if run for more iterations. However, it can be concluded that a standard PINN is enough to handle complexities in this problem.

2.3. Two Phase Stefan's Problem

This is a class of problems involving dynamic interactions between two material (pure) phases separated by a free interface such as the ice solidification problem describing the temperature and boundary evolution in a liquid and solid water system undergoing heat transfer across the interface as phase change occurs. Fig 19 showcases such a system of length L with the position of the interface as a function of time given by $s(t)$ and temperature at position x at time t given by $T(x,t)$.

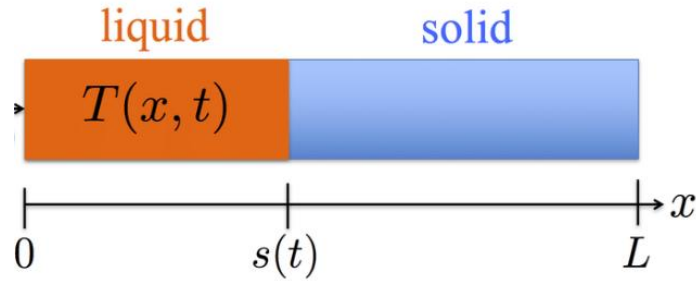


Fig 19: Interface and Temperature in a system undergoing phase change [26]

Let $u_1(x,t)$ and $u_2(x,t)$ denote the temperature on the left and right sides of the interface respectively. Then the system can be defined by 2 PDEs representing the physics (heat equation) for each side of the interface, 1 PDE for the interactions at the boundary (interface physics), 1 equation for temperature continuity across the interface, 1 equation for initial interface position and the domain defined with $L = 2$ and $T = 1$ as in Fig 20. As this system involves 3 PDEs catering to two phenomena (heat transfer and interface movement), it is a more complex problem to solve compared to earlier cases. Moreover, as is the case in the industrial set-ups, we have missing thermal boundary conditions. We attempt to replicate the work done in [16] to showcase the utility of PINN in solving such real-world problems. The temperature at the interface u^* is known to be zero. Other constants are given by:

$$k_1 = 2, k_2 = 1, \alpha_1 = -2, \alpha_2 = 1, s_0 = \frac{1}{2}$$

$$\frac{\partial u_i}{\partial t} = k_i \frac{\partial^2 u_i}{\partial x^2}, \quad (x, t) \in \Omega_i, \quad i = 1, 2$$

$$\Omega_1 = \{(x, t) \in \Omega : 0 < x < s(t), t \in (0, T]\}$$

$$\Omega_2 = \{(x, t) \in \Omega : s(t) < x < L, t \in (0, T]\}$$

$$u_1(s(t), t) = u_2(s(t), t) = u^*, \quad t \in [0, 1]$$

$$s'(t) = \alpha_1 \frac{\partial u_1}{\partial x}(s(t), t) + \alpha_2 \frac{\partial u_2}{\partial x}(s(t), t), \quad t \in [0, 1]$$

$$s(0) = s_0$$

Fig 20: System describing the Stefan's Problem [16]

This problem was solved with a PINN architecture consisting of 2 networks - one to predict temperatures u_1 and u_2 with inputs x and t , and another to predict s with input t . Each network consisted of 5 layers (same as taken in [16]), each with 100 neurons, swish activation and glorot normal weight initialiser (the average setting from Table 3). See Fig 21 for a schematic of the PINN architecture. All the loss terms corresponding to the conditions, PDEs and data-fit are added (See Fig 22) and the loss is minimised to obtain the optimal weights for the 2 networks.

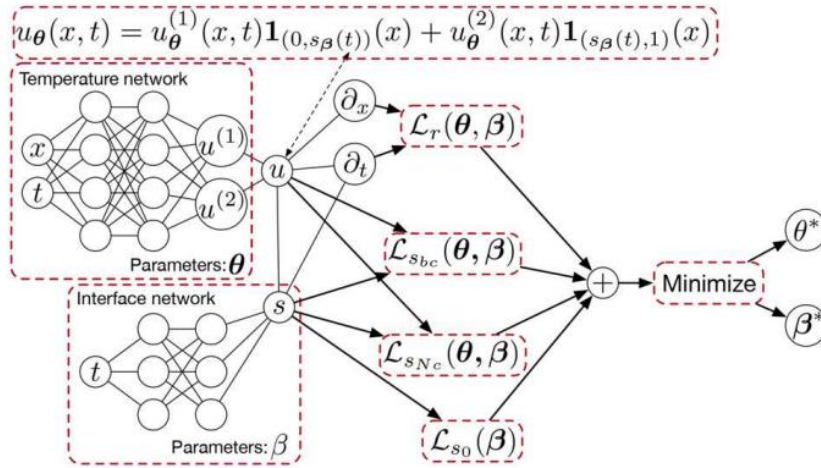


Fig 21: The PINN architecture for solving the Stefan's Problem [16]

$$\mathcal{L} = \sum_{k=1}^2 [\mathcal{L}_r^{(k)} + \mathcal{L}_{sbc}^{(k)}] + \mathcal{L}_{sNc} + \mathcal{L}_{s0} + \mathcal{L}_{data} \quad (26)$$

where

$$\mathcal{L}_r^{(k)} = \frac{1}{N} \sum_{i=1}^N \left| \frac{\partial u_{\theta}^{(k)}}{\partial t}(x^i, t^i) - \lambda_k \frac{\partial^2 u_{\theta}^{(k)}}{\partial x^2}(x^i, t^i) \right|^2, \quad k = 1, 2 \quad (27)$$

$$\mathcal{L}_{sbc}^{(k)} = \frac{1}{N} \sum_{i=1}^N |u_{\theta}^{(k)}(s_{\beta}(t^i), t^i)|^2, \quad k = 1, 2 \quad (28)$$

$$\mathcal{L}_{sNc} = \frac{1}{N} \sum_{i=1}^N \left| 2 \frac{\partial u_{\theta}^{(1)}}{\partial x}(s_{\beta}(t^i), t^i) - \frac{\partial u_{\theta}^{(2)}}{\partial x}(s_{\beta}(t^i), t^i) + \frac{ds_{\beta}}{dt}(t^i) \right|^2 \quad (29)$$

$$\mathcal{L}_{s0} = \frac{1}{N} \sum_{i=1}^N |s_{\beta}(0) - s_0|^2 \quad (30)$$

$$\mathcal{L}_{data} = \frac{1}{M} \sum_{i=1}^M |u_{\theta}(x_{data}^i, t_{data}^i) - u^i|^2 \quad (31)$$

Fig 22: Formulae for all the loss terms making up the loss function [16]

\mathcal{L}_r denotes the domain PDE residual, \mathcal{L}_{sbc} denotes the loss for interface temperature condition, \mathcal{L}_{sNc} denotes the interface PDE residual, \mathcal{L}_{s0} denotes the loss for initial interface condition, \mathcal{L}_{data} denotes the data-fit MSE with $k = 1, 2$ for each side of the interface

For training, $M = 200$ measurement (labelled) data points were sampled from the domain while at every iteration (till 5000 epochs) 10,000 collocation data points were randomly sampled. To get the labels, the analytical solution was used. See Fig 23.

$$u_1(x, t) = 2(\exp((t + 1/2 - x)/2) - 1)$$

$$u_2(x, t) = \exp((t + 1/2 - x) - 1)$$

$$s(t) = t + 1/2$$

Fig 23: Analytical Solution for the Stefan's Problem [16]

The training took ~25 mins on Google Colab GPU resulting in final loss = 6.206E-06. The variation of loss with epochs is shown in Fig 24 and indicates smooth learning.

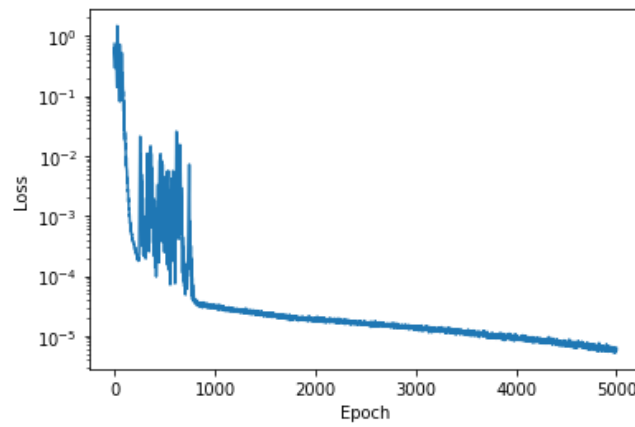
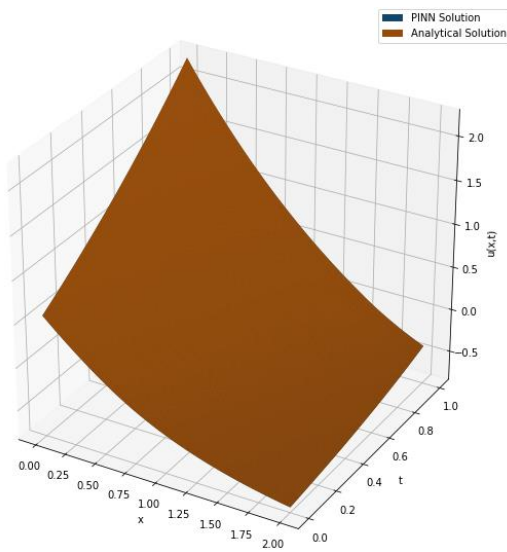
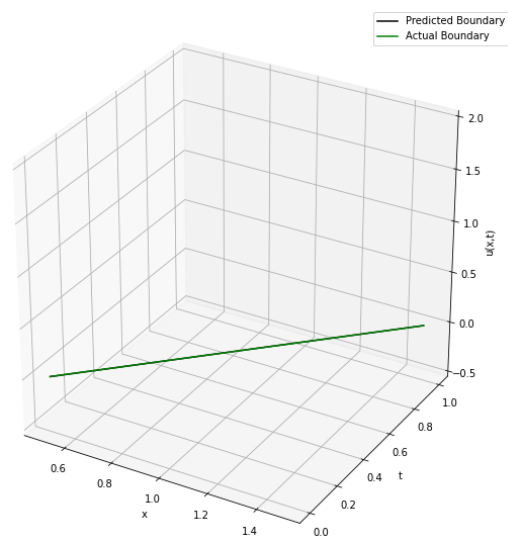


Fig 24: Loss variation (log-scale) with epochs

The PINN predicted temperature matches closely with the analytical solution (See Fig 25) with a MSE over 500 x 500 grid in domain being 4.735E-07. Same is the case for the evolution of interface position (Fig 26) with the MSE being 5.455E-08.



(L) Fig 25: The predicted temperature $u(x,t)$ 3D surface plot matches with the analytical plot



(R) Fig 26: The predicted evolution of interface $s(x=0,t)$ matches with the analytical line

2.4. Directional Solidification Problem

Directional solidification is the process where a material undergoes solidification from one end of the system to another such that the liquid feed is always available to the portion that is just solidifying. For example, in Fig 27, there is a form filled with molten metal which is placed inside a furnace. As the form is slowly drawn out of the furnace and into a coolant, a temperature gradient develops across it. This initiates crystal growth on the bottom of the form, which then slowly propagates upwards. The crystals elongate in the direction of temperature gradient as the solid-liquid interface moves up until a fully solidified and oriented polycrystalline cast is obtained.

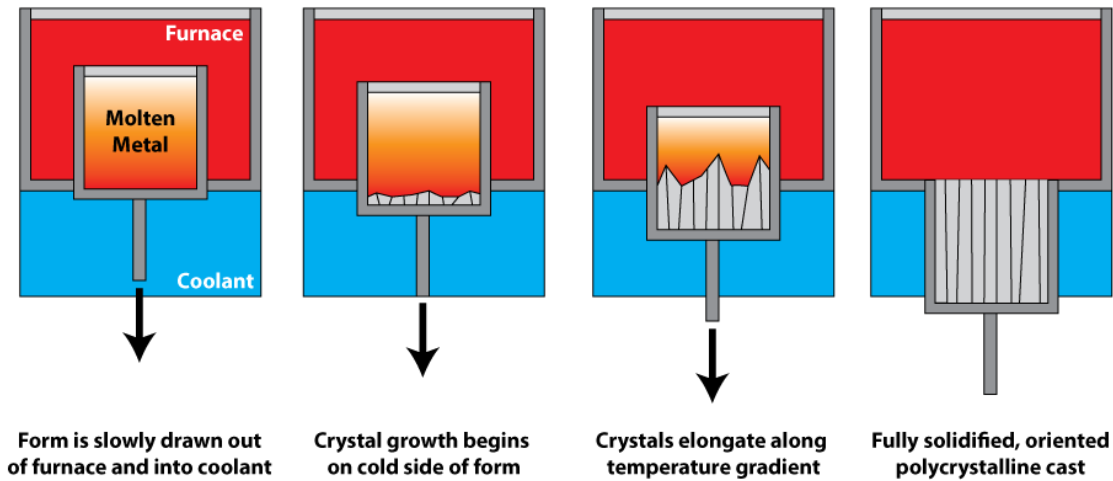


Fig 27: Schematic of the directional solidification process [27]

A 1D directional solidification process with a planar front moving with a constant velocity and negligible diffusion in the solid is considered for solving with PINN. This is another complex problem involving interface movement and diffusion in liquid.

The diffusion equation for the composition in the liquid (C_l) is given by:

$$\frac{\partial C_l}{\partial t} = D_l \frac{\partial^2 C_l}{\partial x^2} \quad \text{where } D_l = \text{diffusion coefficient} \quad (19)$$

The position of the interface will be given by $x^* = v_p t$ where v_p is the velocity of the moving interface which was initially at the origin. Any position x can be expressed relative to the interface position by the transformation $z = x - v_p t$. This simplification can combine Equation 19 and the equation for x^* into a single equation:

$$\frac{\partial C_l}{\partial t} - v_p \frac{\partial C_l}{\partial z} = D_l \frac{\partial^2 C_l}{\partial z^2} \quad (20)$$

The initial liquid concentration is taken to be $C_l(z, 0) = C_0$. The partition coefficient is given by $k_0 = C_s^* / C_l^*$ where C_s^* and C_l^* are concentrations of either phase at the interface. The equation representing the thermodynamic equilibrium and solute balance at the interface (whose position will always be given by $z = 0$) is given by:

$$-D_l \frac{\partial C_l}{\partial z} = C_l^* (1 - k_0) v_p \quad (21)$$

Construct a PINN consisting of a single network (5 hidden layers each with 100 neurons, swish activation and glorot normal weight initialiser as discussed in Chapter 1) to determine the liquid concentration at relative position z and time t $C_l(z, t)$.

For training, $M = 200$ measurement (labelled) data points were sampled from the domain while at every iteration (till 5000 epochs) 10,000 collocation data points were randomly sampled. To get the labels, the analytical solution was used. See Fig 28.

$$\frac{C_\ell}{C_0} = 1 + \frac{1 - k_0}{2k_0} \exp\left(-\frac{v_p z}{D_\ell}\right) \operatorname{erfc}\left(\frac{z - v_p t}{2\sqrt{D_\ell t}}\right) - \frac{1}{2} \operatorname{erfc}\left(\frac{z + v_p t}{2\sqrt{D_\ell t}}\right) \\ + \left(1 - \frac{1}{2k_0}\right) \exp\left(-\frac{(1 - k_0)v_p(z + k_0 v_p t)}{D_\ell}\right) \operatorname{erfc}\left(\frac{z + (2k_0 - 1)v_p t}{2\sqrt{D_\ell t}}\right)$$

Fig 28: Analytical Solution for $C_l(z, t)$ [17]

Take the values of constants as $C_0 = D_\ell = v_p = 1$ and $k_0 = 0.1$.

The training took ~14 mins on Google Colab GPU resulting in final loss = 1.528E-04.

The variation of loss with epochs shown in Fig 29 indicates good enough learning.

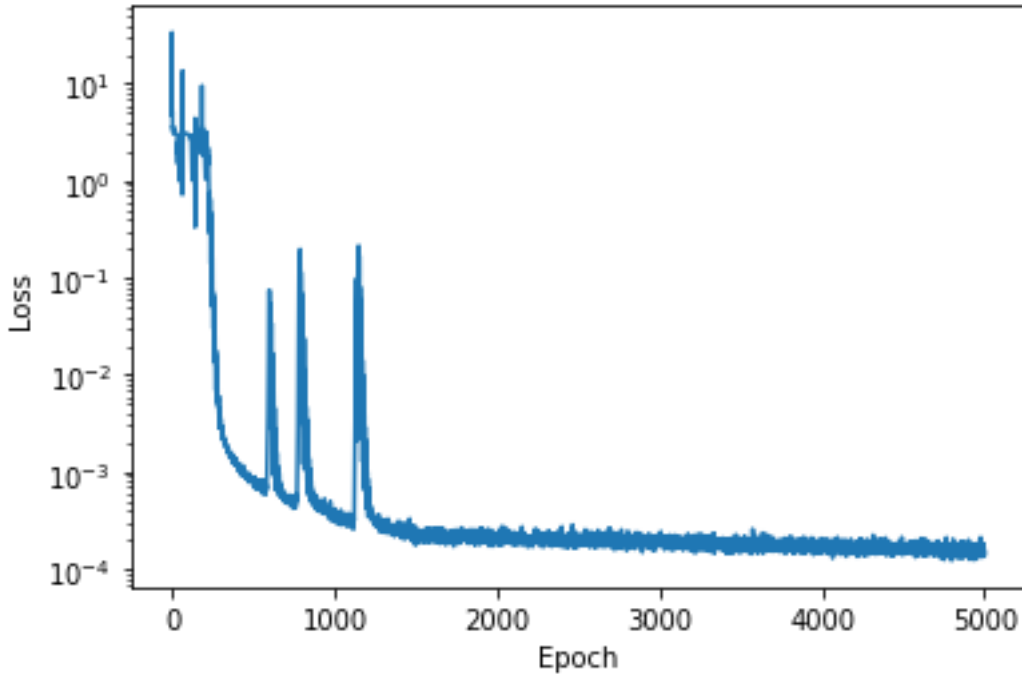


Fig 29: Loss variation (log-scale) with epochs

The PINN predicted liquid concentration profile matches closely with the analytical solution. It is plotted against the distance from interface (z) for the snapshot $t = 16s$ (Fig 30) as well as against position from origin ($x = z + v_p t$) at several times (Fig 31). The close match in concentration profiles in Fig 31 also indicates that the evolution of interface position (given by where profiles start) has been predicted correctly.

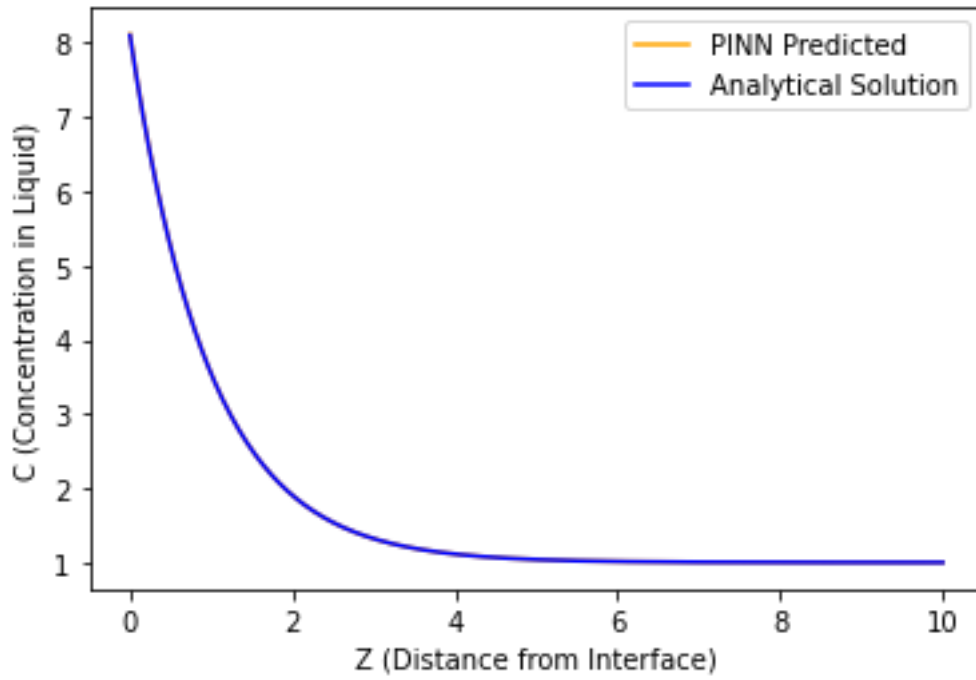


Fig 30: Predicted concentration profile (wrt interface) matches with the analytical at $t = 16s$

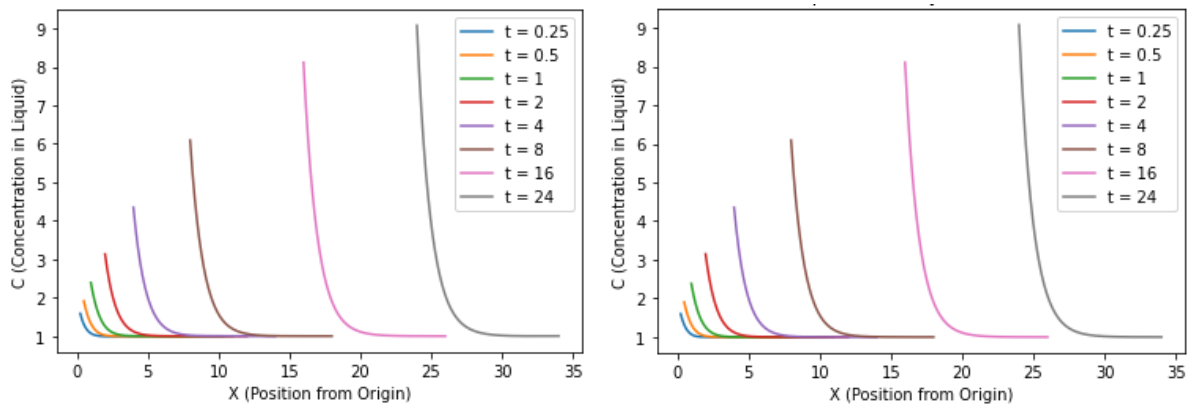


Fig 31: Liq. Conc. profiles at various times: PINN predicted (L) matches with analytical (R)

2.5. A benchmark problem involving multiple PDEs

2.5.1. Motivation: Porosity Prediction

Predicting defects in metal alloys is very critical in engineering applications to avoid failure and establish quality control. Defects such as porosity might arise during various manufacturing processes like casting and welding. Porosity (which can be considered as the gas phase) in a metal alloy (which is solid) can be modelled as a 2-phase problem. If the process is thermal, there is heat transfer involved. Further, there will also be a diffusion of species in the 2 regions. And predicting the position of the interface between the 2 phases would be equivalent to tracking the movement of porosity. Hence, there is a need to predict temperature (T – to model heat conduction) and composition (C – to model diffusion) at all locations (x) and interface position (S) at any instance of time (t) to model all aspects of this problem when considered for the 1D case.

2.5.2. Problem Description

Consider the 1D solidification of a binary alloy in a mold. Let it be a Fe-C alloy with composition C_0 as indicated in the phase diagram in Fig 32. The process begins with the alloy in the liquid state at uniform temperature T_∞ (greater than the liquidus temperature). The solidification starts from the wall (origin) which is at temperature T_0 . Let the equilibrium freezing temperature of pure material be T_f and let T^* be the interface temperature. The position of the flat interface between the 2 phases at any time t is given by $x^*(t)$. The problem geometry and expected C and T profiles at an instance are illustrated in Fig 32. A variable transformation is applied to solve the problem in terms of the following dimensionless quantities:

$$\xi = \frac{x}{L}; \quad \xi^* = \frac{x^*}{L}; \quad \tau = \frac{\alpha t}{L^2}; \quad \theta = \frac{T - T_0}{T_\infty - T_0}; \quad \theta_f = \frac{T_f - T_0}{T_\infty - T_0}; \quad \theta^* = \frac{T^* - T_0}{T_\infty - T_0} \quad (22)$$

Note that as composition is taken in mass% it is already dimensionless.

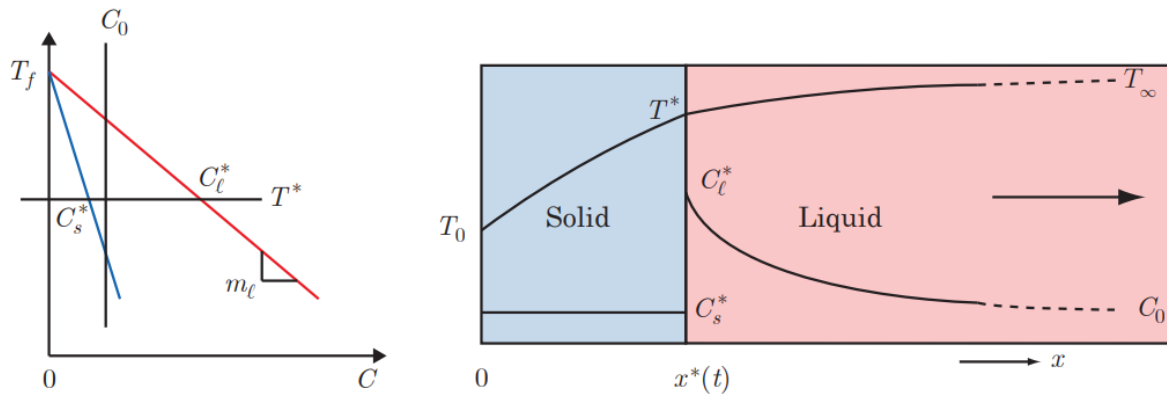


Fig 32: Sketch of phase diagram (L) and temperature & composition profiles (R) [17]

The meanings and values of constants used in this problem are given in Fig 33.

<i>Material properties</i>			
Nominal composition	C_0	0.05	mass %
Liquidus slope	m_ℓ	-81.1	K (mass %) ⁻¹
Solidus slope	m_s	-478	K (mass %) ⁻¹
Specific heat	c_p	820	J kg ⁻¹ K ⁻¹
Density	ρ	7×10^3	kg m ⁻³
Latent heat	L_f	2.76×10^5	J kg ⁻¹
Iron freezing point	T_f	1538	°C
<i>Boundary and initial condition data</i>			
Initial temperature	T_∞	1540	°C
Wall temperature	T_0	1510	°C
<i>Derived quantities</i>			
Segregation coefficient	k_0	0.17	—
Stefan number	Ste	0.089	—
Lewis number	Le	300	—
Dimensionless freezing point	θ_f	0.933	—

Fig 33: Meaning and values (along with units) of constants used in this problem [17]

The analytical solutions required for the labelled data fitting are available in Fig 34. These are functions of x and t and in terms of dimensionless quantities θ^* and ϕ . The interface temperature θ^* itself depends on the interface position parameter ϕ which can be computed from other constants via the transcendental equation given in Fig 35. Note that $C_s(x,t) = \text{constant} \approx 0.034$ (calculated from formulae for k_0 , C_l , ϕ). [17]

$$\begin{aligned}
 \theta_s &= \frac{\theta^*}{\text{erf}(\phi)} \text{erf}\left(\frac{\xi}{2\sqrt{\tau}}\right) \\
 \theta_\ell &= 1 - \frac{1 - \theta^*}{\text{erfc}(\phi)} \text{erfc}\left(\frac{\xi}{2\sqrt{\tau}}\right) \\
 \xi^* &= 2\phi\sqrt{\tau} \\
 C_\ell &= C_0 + \frac{C_\ell^* - C_0}{\text{erfc}(\phi\sqrt{\text{Le}})} \text{erfc}\left(\frac{\xi\sqrt{\text{Le}}}{2\sqrt{\tau}}\right) \\
 \theta^* &= \text{erf}(\phi) \left(1 + \frac{\sqrt{\pi}}{\text{Ste}} \phi \exp(\phi^2) \text{erfc}(\phi)\right)
 \end{aligned}$$

Fig 34: Analytical solutions for T & C variables and interface position [17]

$$\left\{ \theta_f - \text{erf}(\phi) \left(1 + \frac{\sqrt{\pi}}{\text{Ste}} \phi \exp(\phi^2) \text{erfc}(\phi)\right) \right\} \times \\
 \left\{ 1 - \sqrt{\pi \text{Le}} (1 - k_0) \phi \exp(\phi^2 \text{Le}) \text{erfc}(\phi\sqrt{\text{Le}}) \right\} = \frac{m_\ell C_0}{T_0 - T_\infty}$$

Fig 35: Transcendental equation to compute ϕ from other constants [17]

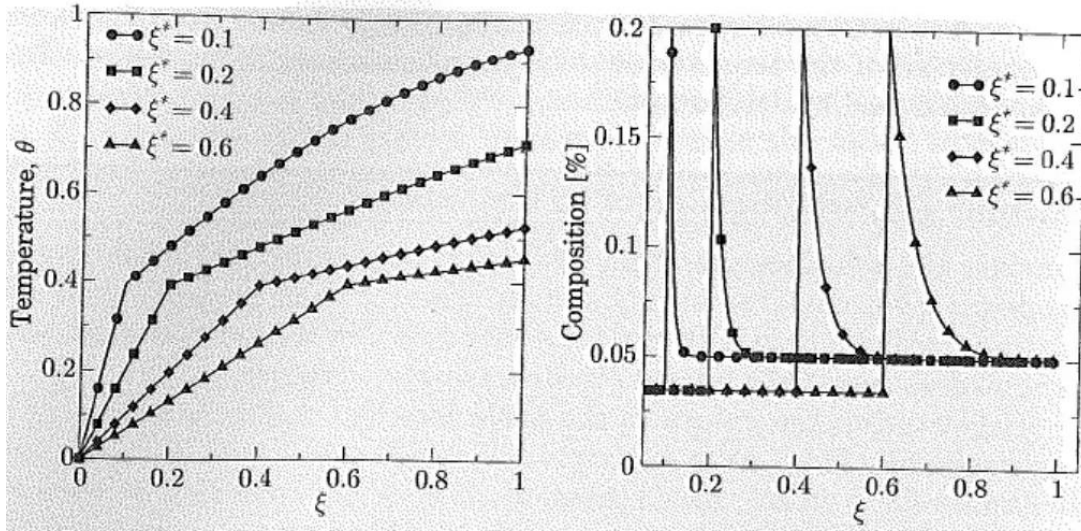


Fig 36: Solution curves $u(\epsilon)$ & $c(\epsilon)$ generated at 4 timesteps using formulae from Fig 34 [17]

Fig 37 shows a system of equations corresponding to both the temperature and concentration variables accounting for the heat conduction and diffusion in both the phases, the interface physics and the initial and boundary conditions. The Stefan number used in Fig 37 is defined as $Ste = c_p(T_\infty - T_0)/L_f$ where c_p is specific heat and L_f is latent heat. α is thermal diffusivity and D_s, D_l are diffusion coefficients.

$\frac{\partial \theta_s}{\partial \tau} = \frac{\partial^2 \theta_s}{\partial \xi^2} \quad 0 \leq \xi \leq \xi^*(\tau)$ $\frac{\partial \theta_\ell}{\partial \tau} = \frac{\partial^2 \theta_\ell}{\partial \xi^2} \quad \xi^*(\tau) \leq \xi < \infty$ $\theta_s = 0 \quad \xi = 0$ $\theta_s = \theta^* \quad \xi = \xi^*(\tau)$ $\frac{1}{Ste} \frac{d\xi^*}{d\tau} = \frac{\partial \theta_s}{\partial \xi} - \frac{\partial \theta_\ell}{\partial \xi} \quad \xi = \xi^*(\tau)$ $\theta_\ell = 1 \quad \xi \rightarrow \infty$ $\theta_\ell = 1 \quad \tau = 0$ $\xi^* = 0 \quad \tau = 0$	$\frac{D_\ell}{\alpha} \left(\frac{D_s}{D_\ell} \frac{\partial C_s}{\partial \xi} - \frac{\partial C_\ell}{\partial \xi} \right) = C_\ell^* (1 - k_0) \frac{d\xi^*}{d\tau} \quad \xi = \xi^*(\tau)$	$\frac{\partial C_s}{\partial \tau} = \frac{D_s}{\alpha} \frac{\partial^2 C_s}{\partial \xi^2} \quad 0 \leq \xi \leq \xi^*(\tau)$ $\frac{\partial C_\ell}{\partial \tau} = \frac{D_\ell}{\alpha} \frac{\partial^2 C_\ell}{\partial \xi^2} \quad \xi^*(\tau) \leq \xi < \infty$ $\frac{\partial C_s}{\partial \xi} = 0 \quad \xi = 0$ $C_s^* = k_0 C_\ell^* \quad \xi = \xi^*(\tau)$ $C_\ell = C_0 \quad \xi \rightarrow \infty$ $C_\ell = C_0 \quad \tau = 0$
---	--	---

Fig 37: System of equations for T (L) and C (R) representing the physics, ICs and BCs [17]

The phase diagram couples the temperature field to the composition field. Assuming a linear form for the liquidus curve (for simplicity) $T = T_f + m_l C_l$ allows us to express the liquid concentration at the interface in terms of the interface temperature by:

$$C_\ell^* = \frac{T_0 - T_\infty}{m_\ell} (\theta_f - \theta^*) \quad (23)$$

Typically, $D_s \ll D_l \ll \alpha$. RHS of liquid diffusion equation can't be neglected as doing so won't satisfy the BCs on C_l . However, the RHS of solid diffusion equation can be neglected as the implied $C_s = \text{constant}$ satisfies the BC on C_s . Further, the term with D_s/D_l in the parenthesis of interface physics DE for C can be neglected as the other term is of the first order. Take α/D_l as the Lewis number Le . We aren't probing what happens at infinity, rather restrict the right boundary to $\epsilon = 1$ and model till $t = 10s$.

Also, the partition coefficient (k_0) by definition mandates a relationship between the compositions of either phase at the interface given by $k_0 = C_s^*/C_l^*$. Here $C_s^* = C_s$.

Thus, the problem can be reduced to satisfying the following set of equations (in the domain, see Fig 38) and conditions (at $t = 0$, $x = 0$, $x = x^*$, see Fig 39):

$$\begin{aligned}
&\text{Domain } \epsilon \in [0, \epsilon^*] \\
&C_s = k_0 C_l^* \\
&\text{PDE1: } \frac{\partial \theta_s}{\partial \tau} = \frac{\partial^2 \theta_s}{\partial \epsilon^2} \\
&\text{Domain } \epsilon \in [\epsilon^*, 1] \\
&\text{PDE2: } \frac{\partial \theta_l}{\partial \tau} = \frac{\partial^2 \theta_l}{\partial \epsilon^2} \\
&\text{PDE3: } \frac{\partial C_l}{\partial \tau} = \frac{1}{Le} \frac{\partial^2 C_l}{\partial \epsilon^2} \\
&\text{Interface } (\epsilon = \epsilon^*) \\
&\text{PDE4: } \frac{1}{Ste} \frac{\partial \epsilon^*}{\partial \tau} = \frac{\partial \theta_s}{\partial \epsilon} - \frac{\partial \theta_l}{\partial \epsilon} \\
&\text{PDE5: } \frac{-1}{Le} \frac{\partial C_l}{\partial \epsilon} = C_l^* (1 - k_0) \frac{d\epsilon^*}{d\tau}
\end{aligned}$$

Fig 38: Equations to be satisfied within the domain

$$\begin{aligned}
&\text{Initial } (\tau = 0) \\
&\theta_l = 1; C_l = C_0; \epsilon^* = 0 \\
&\text{Interface } (\epsilon = \epsilon^*) \\
&\theta_s = \theta_l = \theta^* \\
&C_l^* = (T_0 - T_\infty)(\theta_f - \theta^*)/m_l \\
&\text{Left BC } (\epsilon = 0) \\
&\theta_s = 0
\end{aligned}$$

Fig 39: Initial, Boundary & Interface Conditions

This system involves 5 PDEs = 2 PDEs for heat conduction in each phase + 1 PDE for diffusion in liquid + 2 PDEs for interface multiphysics. There are initial conditions, interface conditions and a left boundary condition on the variables as well. There is an equation relating the solid and liquid concentrations at the interface and another relationship mandated by the phase diagram. In addition, an interface discontinuity is expected in the composition profile as visible in Fig 32. Thus, this becomes a highly complex problem to solve and shall pose a challenge in solving through PINNs.

2.5.3. Techniques Considered

Causal Training

Conventional PINN models are known to struggle with certain problems such as the 1D Allen-Cahn Equation. Authors in [6] illustrated the failure of PINN in learning the accurate solution to this equation. The predicted solution gets stuck in an intermediate state as the loss function doesn't change after decreasing for initial iterations implying that the network gets trapped in an incorrect local minimum which makes further optimisation difficult. This was also observed while solving the complex solidification problem. The authors state that this issue is a common one with PINNs especially while solving transient problems. After careful diagnosis, they conclude that this happens because the loss is minimised even when the predictions at previous time steps are inaccurate thereby violating temporal causality. Thus, this inspired a need to devise a scheme which ensures that training for future time steps proceeds only once the model has accurately learnt the behaviour at earlier times.

Hence, they come up with an approach called causal training. For this, they consider discrete data by taking a uniform mesh of size $N_t = 100 \times N_x = 256$ in the domain of t and x . So, the loss at every time step t_i averaged over all x points as a function of PINN network parameters θ is given by $L(t_i, \theta)$ with the average loss as given in Eq 3.4 of Fig 40. They introduce temporal weight w_i for the loss term at every time step t_i which is expected to be such that it is very large – allowing minimisation of $L(t_i, \theta)$ – only if all losses before t_i are minimised to the best extent. This is achieved by taking the form given in Eq 3.5 of Fig 41. Only if the sum of losses at previous times is small enough would then the inverse exponential make the weight large enough. The ϵ in w_i is a causality parameter which controls the slope of w_i and is gradually increased after every S iterations of training to increase the strength with which the PDE residual constraint is enforced across the temporal domain. This algorithm shall be effective in learning the temporal evolution of U , S & C in our problem.

Algorithm 1: Causal training for physics-informed neural networks

Consider a physics-informed neural network $u_\theta(t, x)$ imposed the exact boundary conditions, and the corresponding weighted loss function

$$\mathcal{L}(\theta) = \frac{1}{N_t} \sum_{i=0}^{N_t} w_i \mathcal{L}(t_i, \theta), \quad (3.4)$$

where $\mathcal{L}(t_0, \theta) = \lambda_{ic} \mathcal{L}_{ic}(\theta)$ and for $1 \leq i \leq N_t$, $\mathcal{L}(t_i, \theta)$ is defined in Equation 2.13. All w_i are initialized by 1. Then use S steps of a gradient descent algorithm to update the parameters θ as:

for $\epsilon = \epsilon_1, \dots, \epsilon_k$ **do**

for $n = 1, \dots, S$ **do**

 (a) Compute and update the temporal weights by

$$w_i = \exp \left(-\epsilon \sum_{k=1}^{i-1} \mathcal{L}(t_k, \theta) \right), \text{ for } i = 2, 3, \dots, N_t. \quad (3.5)$$

 Here $\epsilon > 0$ is a user-defined hyper-parameter that determines the "slope" of temporal weights.

 (b) Update the parameters θ via gradient descent

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta \mathcal{L}(\theta_n). \quad (3.6)$$

if $\min_i w_i > \delta$ **then**

 | break

end

end

end

The recommended hyper-parameters are $\lambda_{ic} = 10^3$, $\delta = 0.99$ and $\{\epsilon_i\}_{i=1}^k = [10^{-2}, 10^{-1}, 10^0, 10^1, 10^2]$.

Fig 40: The causal training algorithm [6]

Scaled Loss Coefficients

The labelled or sampled data may not be the same in number for all the equations and conditions which need to be satisfied. In order to treat all loss terms (i.e. constraints) equally, it thus becomes necessary to scale them such that they represent training using equal number of points, ensuring balanced training. For example, with a mesh of $N_t = 100 \times N_x = 256$ created via random sampling at every training iteration from uniform distribution in domain t and x , the number of data points for initial conditions on u and c (at $t=0$) is 256, the number of interface and boundary data points (at fixed x for every t) is 100, the number of collocation data points is 25600, the number of points for initial condition on s (particular x at $t=0$) is 1 and let the measurement (labelled) data points be 200 which are randomly sampled from domain prior to training. If the corresponding loss terms are denoted by L_i , L_b , L_c , L_{si} , L_m , then the total balanced loss using scaled coefficients $L = 100 \cdot L_i + 256 \cdot L_b + L_c + 25600 \cdot L_{si} + 128 \cdot L_m$ so that every loss term has contributions from effectively 25600 points. Additional scaling can be applied corresponding to the magnitudes of the loss terms. For example, here the composition values are much lower than the temperature values and will thus result in lower loss which would imply the learning of the composition profile won't be focussed upon. Hence, loss terms containing C variables can have higher coefficients.

Adaptive Weighting

The weights for the different loss-terms can also be adaptively varied as hyperparameters learnt online at every training iteration. This will ensure a proper balance is maintained among the different terms throughout the training. One such algorithm is the adaptive weighting or learning rate annealing put-forth in [18]. The authors identified that a common mode of failure in PINNs is the one related to unbalanced gradients during backpropagation. To create a remedy to overcome this pathology, they took inspiration from the popular Adam's optimiser in making use of the backpropagated gradient statistics computed during model training to automatically tune the weights. First, the instantaneous weights are calculated by computing the ratio between the maximum gradient of the residual loss and the mean of the gradient magnitudes of the data-fit loss under consideration as given in Eq 40 of Fig 41. Then, the actual weights λ_i are computed as the running average of previous values (Eq 41 of Fig 41) to reduce stochasticity arising from each gradient descent update. Finally, the gradient descent update takes place to update network parameters θ using weight values λ_i (Eq 42 of Fig 41). While solving the complex solidification problem it was identified that this algorithm is effective once applied after the model is trained using the causal algorithm. The interplay between the data-fit and residual loss terms is balanced and the solution improves considerably.

Algorithm 1: Learning rate annealing for physics-informed neural networks

Consider a physics-informed neural network $f_\theta(x)$ with parameters θ and a loss function

$$\mathcal{L}(\theta) := \mathcal{L}_r(\theta) + \sum_{i=1}^M \lambda_i \mathcal{L}_i(\theta),$$

where $\mathcal{L}_r(\theta)$ denotes the PDE residual loss, the $\mathcal{L}_i(\theta)$ correspond to data-fit terms (e.g., measurements, initial or boundary conditions, etc.), and $\lambda_i = 1, i = 1, \dots, M$ are free parameters used to balance the interplay between the different loss terms. Then use S steps of a gradient descent algorithm to update the parameters θ as:

for $n = 1, \dots, S$ **do**

(a) Compute $\hat{\lambda}_i$ by

$$\hat{\lambda}_i = \frac{\max_{\theta} \{|\nabla_{\theta} \mathcal{L}_i(\theta_n)|\}}{|\overline{\nabla_{\theta} \mathcal{L}_i(\theta_n)}|}, \quad i = 1, \dots, M, \quad (40)$$

where $|\overline{\nabla_{\theta} \mathcal{L}_i(\theta_n)}|$ denotes the mean of $|\nabla_{\theta} \mathcal{L}_i(\theta_n)|$ with respect to parameters θ .

(b) Update the weights λ_i using a moving average of the form

$$\lambda_i = (1 - \alpha)\lambda_i + \alpha\hat{\lambda}_i, \quad i = 1, \dots, M. \quad (41)$$

(c) Update the parameters θ via gradient descent

$$\theta_{n+1} = \theta_n - \eta \nabla_{\theta} \mathcal{L}_r(\theta_n) - \eta \sum_{i=1}^M \lambda_i \nabla_{\theta} \mathcal{L}_i(\theta_n) \quad (42)$$

end

The recommended hyper-parameter values are: $\eta = 10^{-3}$ and $\alpha = 0.9$.

Fig 41: The adaptive weighting algorithm [18]

Modified PINN Architecture

From Fig 32, it is expected that the temperature profile is continuous at the interface and the composition profile is discontinuous at the interface. Hence, a single network (5 hidden layers * 100 neurons + swish + GN) can be used to predict temperatures $u_1(x,t)$ and $u_2(x,t)$ on either side of the interface. As we know beforehand that C_s is constant, we can take it to be a separate parameter or have a simple 1-layer neural network to predict it. A separate network (similar to u) can be used to predict the liquid composition $C_l(x,t)$. This comes from the analysis in Section 2.1 and as suggested in [11]. There will be another network (similar to u) to predict the interface position $s(t)$. Note that the tensorflow swish activation is given by $x \cdot \text{sigmoid}(x)$. Its parameterised version available in `tf.nn.silu` as $x \cdot \text{sigmoid}(\beta x)$ can be employed instead, in order to have a locally adaptive activation function as suggested in [11] and [19]. As the learning behaviour for the different networks corresponding to the prediction of u , c , s may not be similar, different learning rates can be taken for them.

The composition values are low and there is a high jump (discontinuity) in their values at the interface which makes it complex to model C . Rather than using a standard multi-layer perceptron, an alternative could be a modified MLP structure to model C as shown in Fig 42 which was introduced in [18]. Here, the inputs X are embedded into a feature space via two encoders U and V , and merged in each hidden layer of a standard MLP using point-wise multiplication. This was employed in [6] and the authors suggest from prior experience that it improves predictive performance in terms of effectively minimising PDE residuals and capturing sharp gradients. However, such a complicated structure might increase the training time.

$$\begin{aligned}
U &= \phi(XW^1 + b^1), \quad V = \phi(XW^2 + b^2) \\
H^{(1)} &= \phi(XW^{z,1} + b^{z,1}) \\
Z^{(k)} &= \phi(H^{(k)}W^{z,k} + b^{z,k}), \quad k = 1, \dots, L \\
H^{(k+1)} &= (1 - Z^{(k)}) \odot U + Z^{(k)} \odot V, \quad k = 1, \dots, L \\
f_\theta(x) &= H^{(L+1)}W + b
\end{aligned}$$

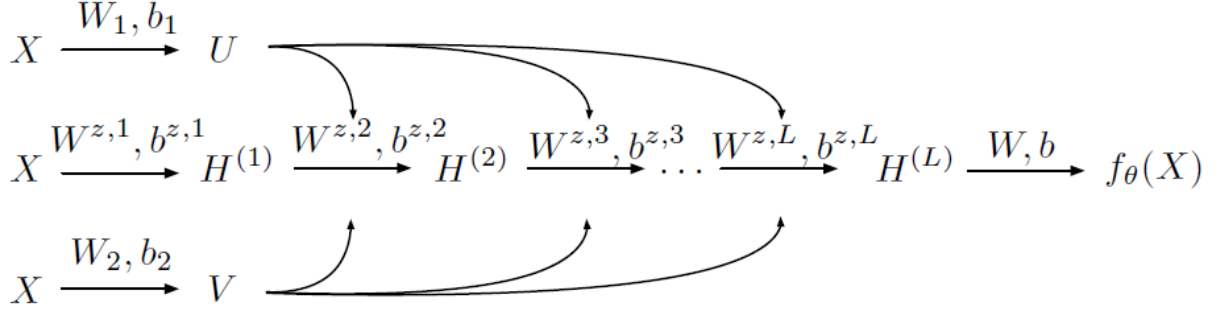


Fig 42: The modified MLP architecture depicted algebraically (T) and schematically (B) [18]

For this complex problem, the labelled domain data can be increased along with the addition of labelled initial and interface points. A higher number of collocation points can be sampled specifically nearby the interface. This might help better model the discontinuous composition profile. Further, the importance sampling algorithm [20] can be embedded into training which while sampling prefers those points at which high residual values are observed. However, this requires computing the PDE residuals (which are composed of compute-intensive gradient calculations) at each and every point individually which increases the training time significantly. Also, simplifications can be introduced by increasing k_0 (lowering the jump in composition at interface) and increasing C_0 (scaling up C variables to be comparable with T and S). Note that, this changes the problem and solution, so isn't an aspect to compare.

2.5.4. Results

We outline the final settings and approach which yielded the best results along with the relevant plots for the problem described in Section 2.5.2.

Data for Training = Labelled Data + Collocation Data

Labelled Data = 900 points randomly sampled from the domain ($x = [0, 1]$, $t = [0, 10]$) + 50 initial points (randomly sampled at $t = 0$) + 50 interface points (random times)

Collocation Data = N_t (100) \times N_x (256) = 25600 grid points over t and x domain

All sampling was performed randomly assuming uniform distribution. Importance sampling [20] turned out to be computationally expensive, hence its use was scrapped in the final implementation. Also, there wasn't any noticeable effect of adding a few residual points nearby the interface to the collocation data.

A standard MLP (fully-connected feed-forward neural net) model consisting of 5 hidden layers each with 100 neurons + swish activation + glorot normal weight initialiser is designed in tensorflow using Keras' Sequential API. Three such networks are taken – one to predict temperatures $u_1(x,t)$ (solid phase) and $u_2(x,t)$ (liquid phase) on either side of the interface, one for predicting the interface position $s(t)$ and another for predicting the composition of liquid phase $C_l(x,t)$. We take a trainable parameter to predict the constant solid composition (C_s). Thus, the weights of the 3 networks and this parameter are trained simultaneously (parallel network strategy). Taking a single parameter for C_s abstracts the problem making use of known information while allowing a separate network to model the complex non-linear high gradient curves for C_l . Most curves for u_1 and u_2 are linear without any discontinuity (Fig 36) and using separate networks for them unnecessarily complicated the training and a single network was found to do a good enough job. Using a modified MLP network (Fig 42) for modelling liquid composition had resulted in high loss values and oscillations for learning rates between 0.01 and 0.001, so ultimately it wasn't used. The silu activation wasn't tried out but it does introduce additional hyperparameters (corresponding to each layer) to optimise making the training complicated.

The loss terms used are as follows (valid for each iteration):

Lm1 = MSE over labelled temperature data (1000 points)

Lm2 = MSE over labelled composition data (1000 points)

Lm3 = MSE over labelled interface data (1000 points)

Lm = Total error over measurement (labelled) data = Lm1 + Lm2 + Lm3

Li1 = MSE for initial interface position ($t=0$, $x=0$ from collocation data = 1 point)

Li2 = MSE for initial condition on composition ($t=0$ from collocation data = 256 points)

Li3 = MSE for initial condition on temperature ($t=0$ from collocation data = 256 points)

Li = Total error in enforcing initial conditions = Li1 + Li2 + Li3

Lc1, Lc2, Lc3, Lc4, Lc5 = MSEs over collocation data for the 5 PDEs (25600 points)

Lc = Total PDE residual error in enforcing physics = Lc1 + Lc2 + Lc3 + Lc4 + Lc5

Ls = MSE in enforcing the definition of partition coefficient ($x=x^* \rightarrow 100$ points)

Lb1 = MSE in enforcing the left boundary condition ($x=0 \rightarrow 100$ points)

Lb2 = MSE in enforcing the temperature continuity at interface ($x=x^* \rightarrow 100$ points)

Lb3 = MSE in enforcing Equation 23 at interface ($x=x^* \rightarrow 100$ points)

Lb = Total error in enforcing the boundary conditions = Ls + Lb1 + Lb2 + Lb3

Total Error (or Simple Loss) = Lm + Li + Lc + Lb

Total Weighted Error (or Scaled Loss) = Lm*25.6 + Li1*25600 + (Li2+Li3)*100 + Lc + Lb*256

When 3 networks (one to predict u_1 & u_2 , one to predict c_1 & c_2 , one to predict s) are trained with the above error as a loss function, after an initial decrease in the loss, the loss value keeps oscillating and the solution gets stuck in an incorrect intermediate state with negligible learning having taken place. For example, when trained without the data-fit term for the interface position, the predicted solution for the interface position is way off the analytical one as seen in Fig 43, which would obviously lead to incorrect predictions for temperatures and compositions.

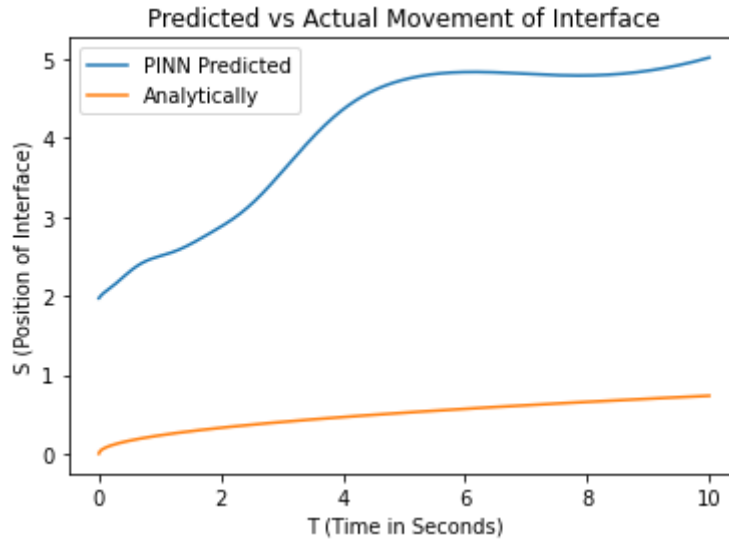


Fig 43: Incorrect prediction for evolution of interface position

The adaptive weighting or learning rate annealing algorithm (Fig 41) is useful in automatically learning the weight for the data-fit term in comparison with the PDE residual. Mostly, it increases the focus on the data-fitting part without performing much worse on the residual constraints or vice-versa, thus training can be performed until there is a reversal in the progress on loss minimisation. As a result, it maximises the learning from available information. When this algorithm was applied despite the incorrect interface predictions as above, the model tried to learn to match the analytical profiles (Fig 44). Thus, it is expected that once the evolution of interface position is modelled to a good extent, applying this algorithm can be sensible.

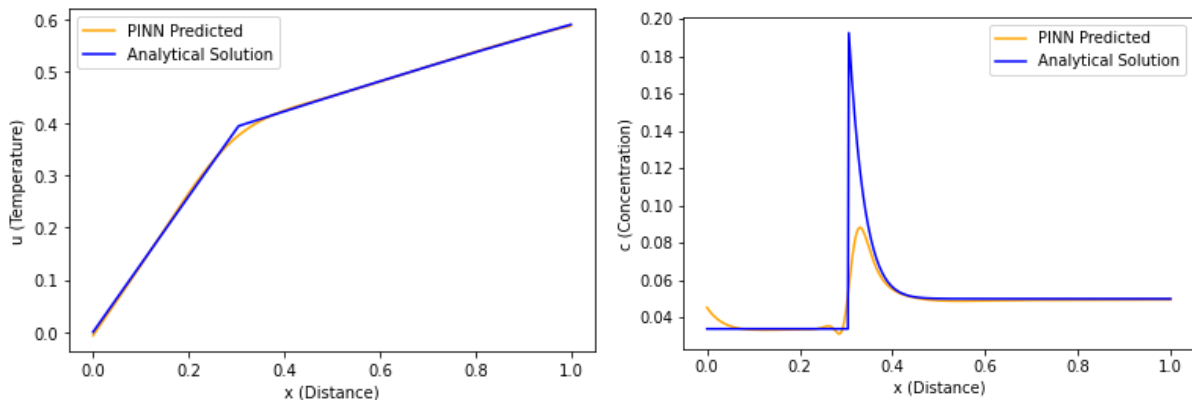


Fig 44: Predicted profiles $c(x)$ & $u(x)$ at $t = 1.725s$ matching the analytical despite incorrect s

This necessitated the application of the causal training algorithm (Fig 40) to model the temporal evolution of the variables and improve the loss convergence. Now, for enforcing the boundary conditions and residual constraints, the loss terms are computed at all discrete timesteps with temporal weights as defined in Fig 40. Training for future timesteps proceeds only when loss at previous timesteps is minimised enough.

Causal training was employed for 720 epochs with a learning rate = 0.01 and a single fixed causality parameter $\varepsilon = 1$. The results are shown in Fig 45.

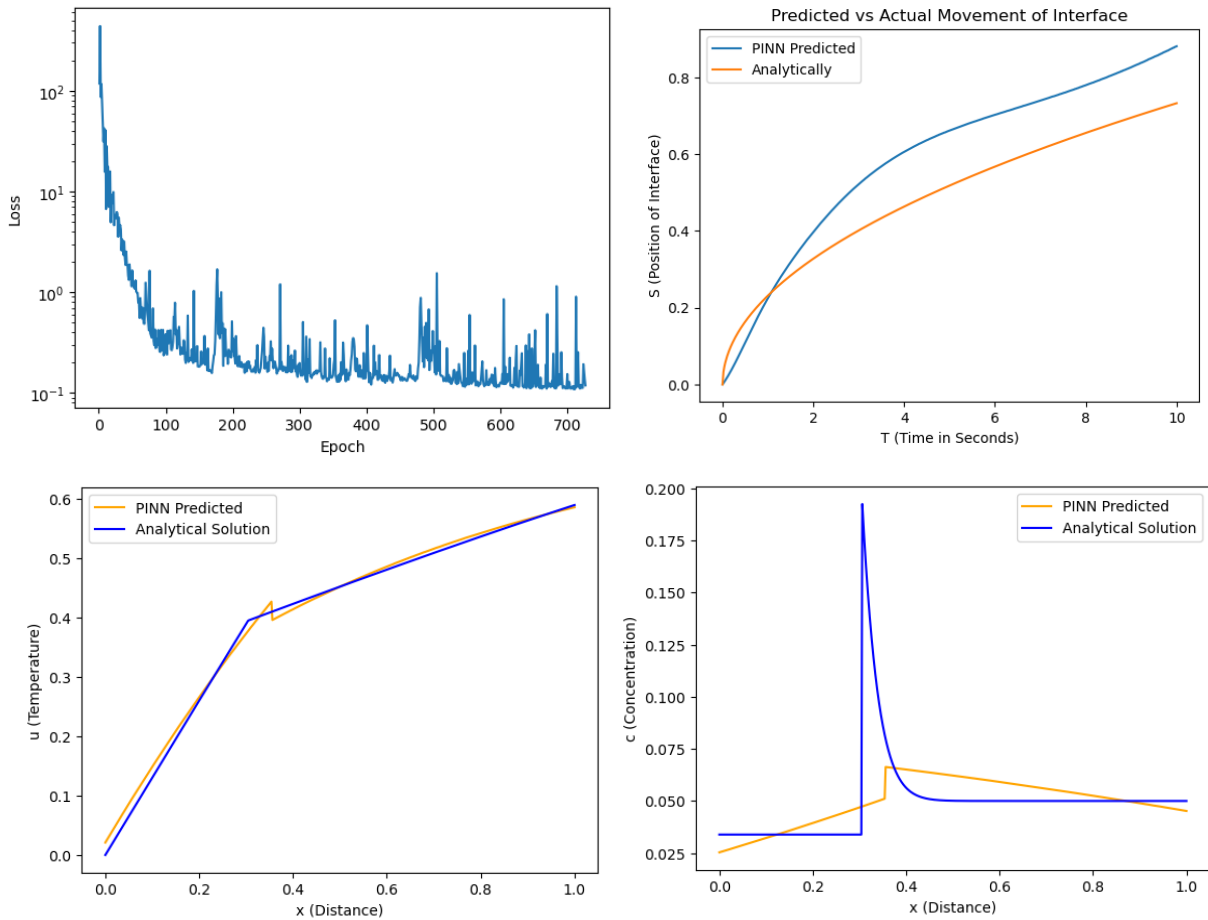


Fig 45: Loss Curve and Predicted Profiles $S(t)$, $U(x)$, $C(x)$ at $t = 1.725s$ for training with $\varepsilon = 1$

The evolution of the interface is predicted better now (still inaccurate) and a good temperature fit is observed but the composition profile isn't learned much.

When trained with a gradually increasing causality parameter i.e. 200 epochs each for epsilon in [0.01, 0.1, 1, 10, 100], the following results are observed (Fig 46). The predictions for the interface evolution and the composition profile are much better.

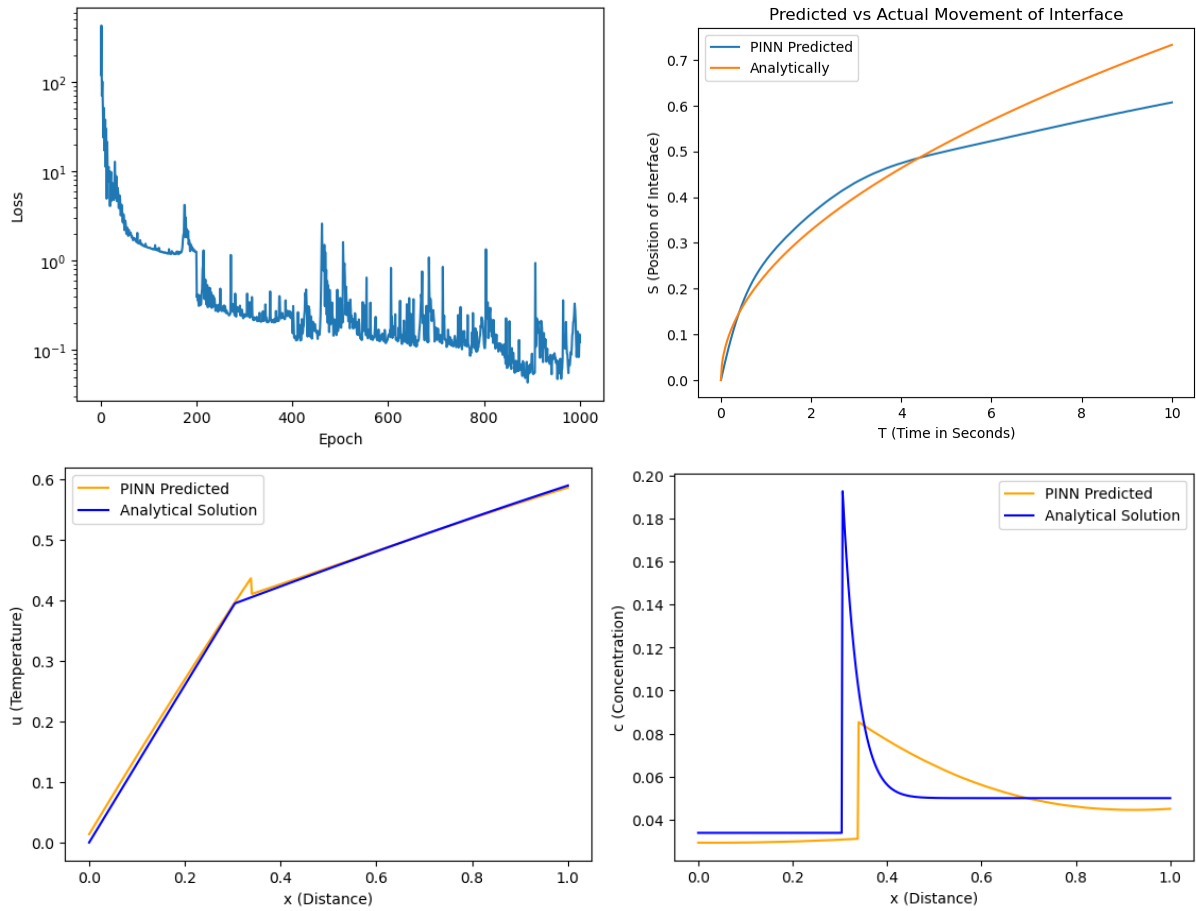


Fig 46: Loss Curve and Predicted Profiles $S(t)$, $U(x)$, $C(x)$ at $t = 1.725s$ (multiple ϵ case)

Several experiments were run with different strategies and varying learning rate schedules. It was observed that, in most cases, for a single epsilon, loss decreased consistently till 150-200 epochs after which oscillations occurred, and the last 2 epsilons (i.e. 10 and 100) were proving to be very difficult for optimisation with loss minimisation ineffective compared to the first 3 epsilons. Use of multiple networks for each phase was also tried. In order to improve the predicted composition profile, the coefficients of the composition loss terms were increased but this seemed to confuse the model and it could neither model $U(x,t)$ nor $C(x,t)$ good-enough.

After gaining insights from many experiments, the final procedure involves first approximately predicting the constant solid composition and then finding a good enough fit for interface movement. This is done during causal training (multiple epsilons) which meanwhile learns modelling $U(x,t)$ good enough but not $C(x,t)$. Then, for getting a better temperature profile and good composition profile, adaptive weighting and causal training can be alternatively applied (with decreasing learning rate) to help focus both on data-fitting and physics.

Ultimately, causal training was employed (loss inclusive of the data-fit term for interface position L_{m3}) with a learning rate = 0.005. The algorithm was trained with epsilon in [0.01, 0.1, 1, 10, 100] for 200 epochs each. An error of 10^{-4} was used as a stopping condition for training the solid composition parameter C_s i.e. whenever the value of trainable variable C_s is offset from the analytical value by less than 10^{-4} , C_s is no more trainable and training proceeds only for the weights of the 3 networks. This resulted in the predicted value of C_s to be 0.03390457 \approx 0.034 which is correct.

After training for 1000 epochs, the loss curve was plotted (Fig 47 L). The weights of the model were restored to that iteration wherein the least error was recorded (826th). The model still suffers from improper convergence but the predictions are better than earlier. A good fit was observed in the evolution of the interface position (Fig 47 R). This paved the way for the application of adaptive weighting. The NN model for predicting the interface movement $s(t)$ was now freezed and this algorithm was employed for 1000 epochs with a learning rate of 0.0001. This improved the predictions for the temperature (Fig 48) and composition profiles (Fig 49). A great improvement occurs specifically in the composition profile after adaptive weighting.

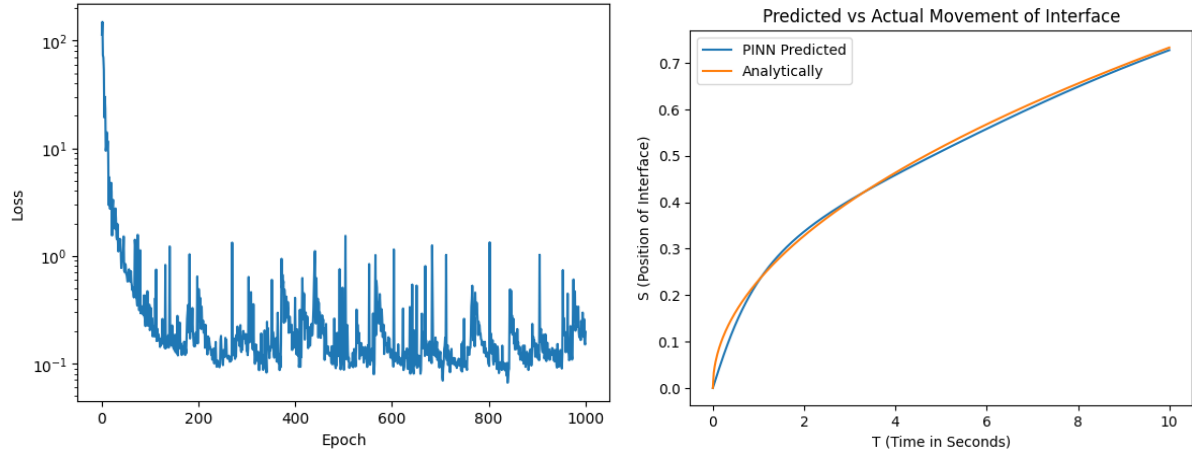


Fig 47: Loss vs Epochs curve (L) and Movement of interface $s(t)$ (R) after causal training

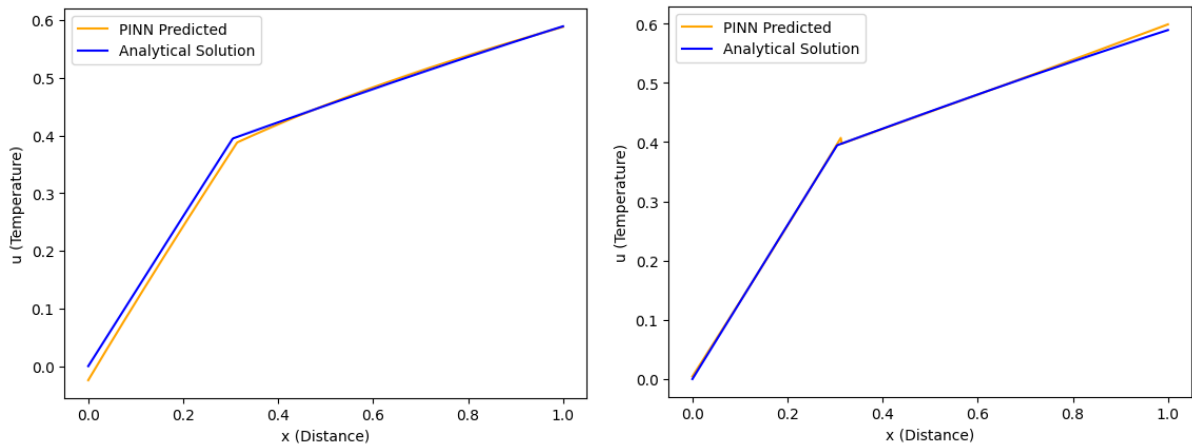


Fig 48: Predicted vs Analytical Temperature Profile $U(x)$ at $t = 1.725s$ before (L) & after (R)

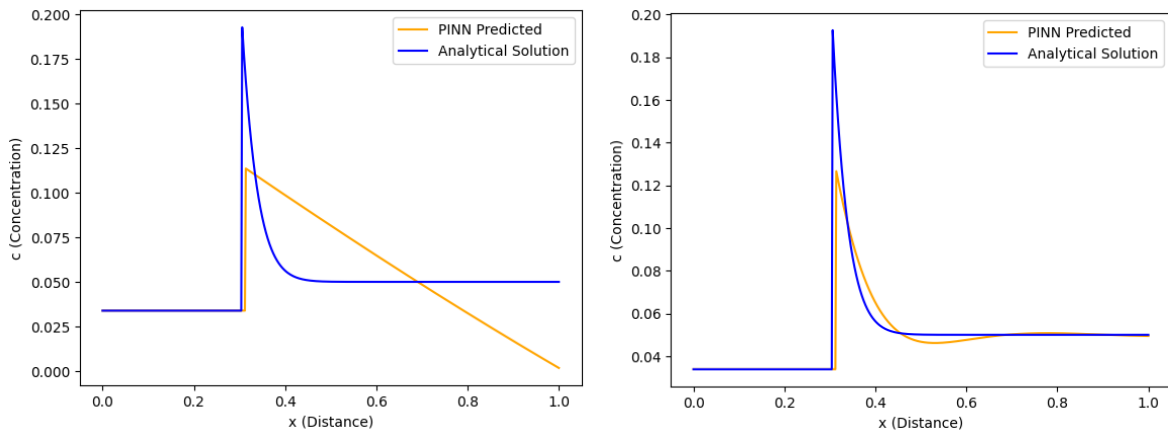


Fig 49: Predicted vs Analytical Composition Profile $C(x)$ at $t = 1.725s$ before (L) & after (R)

Again, causal training was employed for 150 epochs (least error in 76th) with learning rate = 0.0005 and fixed $\varepsilon = 1$, followed by another round of adaptive weighting (same as earlier). Fig 50 shows the final temperature and composition profiles which now indicate a closer match between the predicted and the analytical. Fig 51 shows the evolution of temperature and composition with time at a particular location. Fig 52 shows the 3D surface plots of the final model predictions and their analytical solutions for $U(x,t)$ (top) and $C(x,t)$ (bottom) over a domain grid of 500 x 500 points.

These graphs highlight the success of the scheme employed and its potential once trained for more iterations with emphasis still needed on addressing convergence.

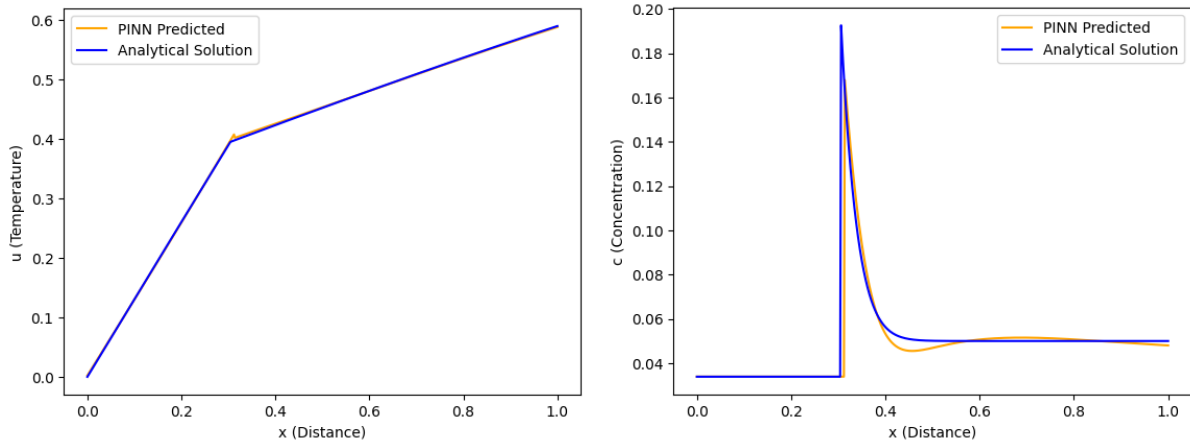


Fig 50: Final temperature and composition profiles $U(x)$ (L) & $C(x)$ (R) at $t = 1.725s$

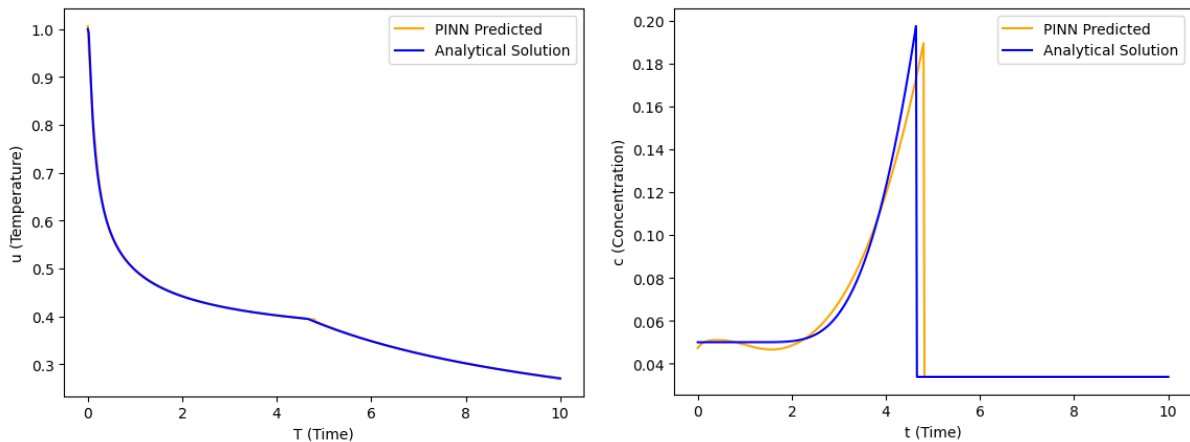


Fig 51: Evolution of temperature and composition with time $U(t)$ (L) & $C(t)$ (R) at $x = 0.5$

Final errors over the domain grid of 500 x 500 points:

MSE between the predicted and analytical solution for temperature $U(x,t) = 2.32E-05$

MSE between the predicted and analytical solution for composition $C(x,t) = 2.51E-04$

MSE between the predicted and analytical positions of interface $s(t) = 1.26E-04$

Note: It was found that using `@tf.function` decorator before the function definition of `training_step` redefines the function as a tensorflow graph, significantly speeding up computation. As a result, 1000 epochs of causal training took ~86 mins (earlier it used to take >24 hours for the same) and 1000 epochs of adaptive weighting took ~3 mins.

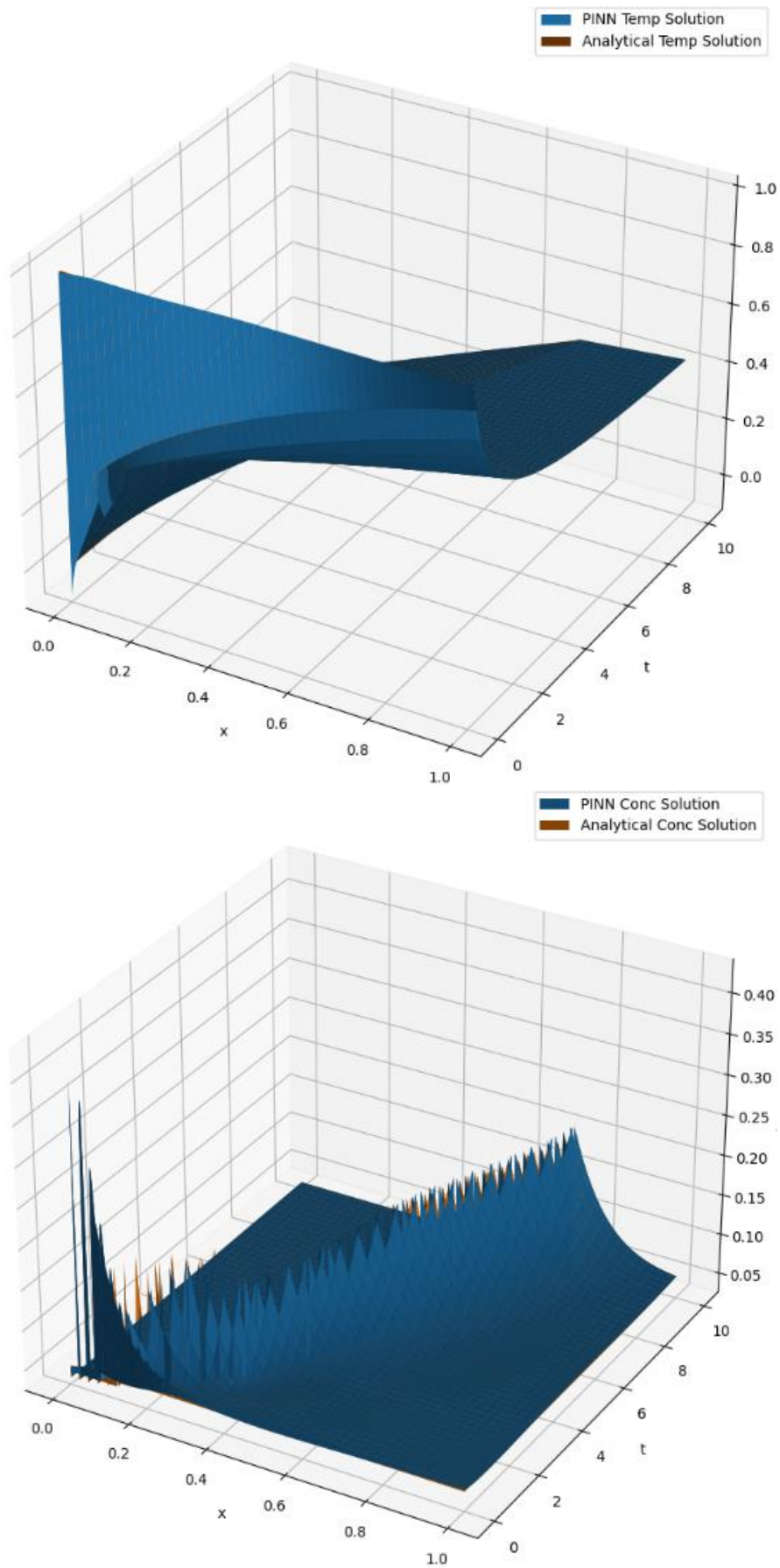


Fig 52: Predicted vs Analytical surface plots for temperature $U(x,t)$ and composition $C(x,t)$

Conclusion

Physics-Informed Neural Networks provide a useful framework to build data-driven and physics-informed models. This research delved into the design aspects of PINNs with the aim to understand whether a unique PINN structure exists for a particular problem so that when a PDE is provided a solution can be readily proposed. Hyperparameters were identified and model tuning was performed to solve several PDEs with the linear convection-diffusion in focus. A lasso regression model was fit to express the number of layers as a function of the polynomial features over variables such as the number of operations, order and degree of the PDE which had a significant strength with $R^2 = 0.953$ indicating that its possible to predict the number of layers required for a given PDE. However, it was concluded from another exercise that knowing the number of layers alone isn't enough to best solve a PDE. A smarter investigation perhaps could be to predict the PINN design from the loss function that gets formed for a system under consideration.

Later, PINNs are used to solve several problems of increasing complexity such as the nonlinear Burger's Equation (involving shocks), the 2-phase Stefan's problem (involving moving interface and missing boundary conditions), the directional solidification problem and ultimately a binary alloy solidification as a benchmark. This problem involved predicting the temperature and composition of the two phases and tracking the movement of the interface separating them. There was a discontinuity, high gradient and non-linear curve in the composition profile. Strategies such as multiple parallel networks and algorithms like causal training and adaptive weighting were adopted from the literature and after insights from several experiments, a scheme was employed which solved the problem to a good extent as highlighted by the figures presented. Although conventional numerical solvers (such as finite difference or finite element methods) typically perform well for such problems, PINNs and its advanced variants can be beneficial in solving problems where coupled and moving boundary systems exist.

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," *J Comput Phys*, vol. 378, pp. 686–707, Feb. 2019, doi: 10.1016/j.jcp.2018.10.045.
- [2] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, "Physics-informed machine learning," *Nature Reviews Physics*, vol. 3, no. 6. Springer Nature, pp. 422–440, Jun. 01, 2021. doi: 10.1038/s42254-021-00314-5.
- [3] I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Trans Neural Netw*, vol. 9, no. 5, pp. 987–1000, 1998, doi: 10.1109/72.712178.
- [4] S. Cuomo, V. S. di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli, "Scientific Machine Learning through Physics-Informed Neural Networks: Where we are and What's next," Jan. 2022, [Online]. Available: <http://arxiv.org/abs/2201.05624>
- [5] T. Kadeethum, T. M. Jørgensen, and H. M. Nick, "Physics-informed neural networks for solving nonlinear diffusivity and Biot's equations," *PLoS One*, vol. 15, no. 5, May 2020, doi: 10.1371/journal.pone.0232683.
- [6] S. Wang, S. Sankaran, and P. Perdikaris, "Respecting causality is all you need for training physics-informed neural networks," Mar. 2022, [Online]. Available: <http://arxiv.org/abs/2203.07404>
- [7] N. Sukumar and A. Srivastava, "Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks," *Computer Methods in Applied Mechanics and Engineering*, vol. 389, Feb. 2022, doi: 10.1016/j.cma.2021.114333.
- [8] A. A. Ramabathiran and P. Ramachandran, "SPINN: Sparse, Physics-based, and partially Interpretable Neural Networks for PDEs," *J Comput Phys*, vol. 445, Nov. 2021, doi: 10.1016/j.jcp.2021.110600.
- [9] S. A. Niaki, E. Haghighat, T. Campbell, A. Poursartip, and R. Vaziri, "Physics-Informed Neural Network for Modelling the Thermochemical Curing Process of Composite-Tool Systems During Manufacture," *Computer Methods in Applied Mechanics and Engineering*, vol. 384, Oct. 2021, doi: 10.1016/j.cma.2021.113959.
- [10] W. H. Lim, S. Sfarra, and Y. Yao, "A Physics-Informed Neural Network Method for Defect Identification in Polymer Composites Based on Pulsed Thermography," *Engineering Proceedings*, vol. 8, no. 1, 2021, doi: 10.3390/engproc2021008014.

- [11] R. Pu and X. Feng, "Physics-Informed Neural Networks for Solving Coupled Stokes–Darcy Equation," *Entropy*, vol. 24, no. 8, Aug. 2022, doi: 10.3390/e24081106.
- [12] L. Lu, P. Jin, and G. E. Karniadakis, "DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators," Oct. 2019, doi: 10.1038/s42256-021-00302-5.
- [13] C. Lin, Z. Li, L. Lu, S. Cai, M. Maxey, and G. E. Karniadakis, "Operator learning for predicting multiscale bubble growth dynamics," *Journal of Chemical Physics*, vol. 154, no. 10, Mar. 2021, doi: 10.1063/5.0041203.
- [14] P. Escapil-Inchauspé and G. A. Ruz, "Hyper-parameter tuning of physics-informed neural networks: Application to Helmholtz problems," May 2022, [Online]. Available: <http://arxiv.org/abs/2205.06704>
- [15] J. Blechschmidt and O. G. Ernst, "Three ways to solve partial differential equations with neural networks — A review," *GAMM Mitteilungen*, vol. 44, no. 2, Jun. 2021, doi: 10.1002/gamm.202100006.
- [16] S. Cai, Z. Wang, S. Wang, P. Perdikaris, G. E. Karniadakis "PINNs for Heat Transfer Problems", *J. Heat Transfer*, Jun. 2021, vol. 143, no. 6, doi: 10.1115/1.4050542.
- [17] J. A. Dantzig and M. Rappaz, *Solidification*. in Engineering sciences. Taylor & Francis Group, 2009. [Online]. Available: <https://books.google.co.in/books?id=yBDWQAAACAAJ>
- [18] S. Wang, Y. Teng, and P. Perdikaris, "Understanding and mitigating gradient pathologies in physics-informed neural networks," Jan. 2020, [Online]. Available: <http://arxiv.org/abs/2001.04536>
- [19] A. D. Jagtap, K. Kawaguchi, and G. E. Karniadakis, "Locally adaptive activation functions with slope recovery term for deep and physics-informed neural networks," Sep. 2019, doi: 10.1098/rspa.2020.0334.
- [20] M. A. Nabian, R. J. Gladstone, and H. Meidani, "Efficient training of physics-informed neural networks via importance sampling," Apr. 2021, doi: 10.1111/mice.12685.
- [21] P. Baheti, "Activation Functions in Neural Networks [12 Types & Use Cases]", V7Labs, May 2021, <https://www.v7labs.com/blog/neural-networks-activation-functions>, last accessed 20/06/2023.
- [22] "Deep Learning: The Swish Activation Function", Lazy Programmer, Oct. 2017, <https://lazyprogrammer.me/deep-learning-the-swish-activation-function/>, last accessed 20/06/2023.
- [23] A Ananthaswamy, "Latest Neural Nets Solve World's Hardest Equations Faster Than Ever Before", *Quantamagazine*, Apr. 2021, <https://www.quantamagazine.org/latest-neural-nets-solve-worlds-hardest-equations-faster-than-ever-before-20210419/> last accessed 20/06/2023.

- [24] F Nogueira, "Bayesian Optimization: Open source constrained global optimization tool for Python", Github, 2014
<https://github.com/fmfn/BayesianOptimization> last accessed 20/06/2023
- [25] Glinowiecka-Cox, Małgorzata B., "Analytic Solution of 1D Diffusion-Convection Equation with Varying Boundary Conditions", University Honors Theses, Paper 1182, 2022, <https://doi.org/10.15760/honors.1224>
- [26] S. Koga, M. Diagne, M. Krstic, "Control and State Estimation of the One-Phase Stefan Problem via Backstepping Design", IEEE Transactions on Automatic Control., Feb. 2019, vol. 64, no. 2, doi: 10.1109/TAC.2018.2836018
- [27] Jedelbrock, Wikipedia Image, Mar. 2016, last accessed 20/06/2023
https://commons.wikimedia.org/wiki/File:Directional_Solidification.png

Appendix

On Hyperparameter Tuning – Few assumptions to fasten exps despite stochasticity

The hyperparameter values and PINN loss do not form a function. Different hyperparameter combinations might result in loss of the same order but I haven't seen a particular hyperparameter combination resulting in losses of completely different orders in different runs. This is why the idea of using Bayesian optimisation is to search for those combinations which result in lower losses. When written in code, a 'python function' can be defined with input as hyperparameter values, PINN can be trained in function definition with the return value (output) as the loss. I allow real numbers for hyperparameter inputs, which are then rounded-off to integers. This enables BO to perform PINN training with same hyperparameters (integers) multiple times. Additionally, it is a convention to use BO for hyperparameter tuning of neural networks when training is time-consuming so that instead of trying out each and every combination, a smaller set is utilised. However, yes 15 iterations might be too less to search up the best combination. With the conditions being the same for all PDEs and the number of IC and BC points being just 50 compared to the 10,000 collocation points, I assumed the loss to be dominated by the PDE residuals. The PDE residual loss term is same as the error as the actual residual is zero, whether we use collocation data or analytical solution. Further, here the loss is computed on the validation set, which aren't the datapoints used for training. For the 8 PDEs which I had tuned, I got 0 as the optimal L2 weight penalty 5 times, 2 times 10^{-6} (5 layers*256 neurons and 11 layers*40 neurons), 1 time 10^{-4} (16 layers*248 neurons) which seemed to be aligned with my intention of higher penalty for high complexity network (a greater number of NN parameters) to avoid overfitting. Ideally, large number of BO iterations should be used (these would include realisations of same hyperparameters in-built) before I proceed to explore and utilise any relationship between the layers in the tuned model and the given PDE, when considered relative to other PDEs.

On Stefan's Problem – I or the authors didn't enforce additional conditions in PINN

PINNs are claimed to be useful because they provide an industrially-relevant framework which can help to solve real-world problems with imperfect data or missing boundary conditions. As highlighted in [16], "in real heat-transfer applications such as power electronics or nuclear reactor, the thermal boundary conditions are never precisely known as it would require enormous and complex instrumentation which is not feasible in industrial set-ups. This leads to an ill-posed boundary value problem which cannot be solved by any sophisticated CFD method." This is why using whatever known conditions and some collected data, PINN learns from the data while enforcing the known-constraints. The utility of PINNs is not just to solve differential equations but to embed physics into the data-driven framework. Here, the labelled data used for fitting is just 200 fixed points in x, t (roughly 14×14 grid) compared to the collocation data which are 10000 points randomly sampled in each iteration of PINN training where the PDE residuals need to be minimised to 0. The analytical solution is used as a representation of collected data. By its defined nature of learning itself, PINN (my case + authors) actually converged to the analytical solution despite missing conditions, which essentially was an attempt to verify the earlier stated claim.