Constraint-Guided PINNs: A Constrained Optimization Approach

Wout Rombouts[1,2,3,*], Quinten Van Baelen[1,2,3] and Peter Karsmakers[1,2,3]

[1]KU Leuven, Dept. of Computer Science, Kleinhoefstraat 4, B-2440 Geel, Belgium
[2]Leuven.AI - KU Leuven Institute for AI
[3]Flanders Make @ KU Leuven, B-3000 Leuven, Belgium

Abstract

Physics-Informed Neural Networks (PINNs) have emerged as a powerful tool for solving Partial Differential Equations (PDEs) by integrating physical laws into the learning process. However, PINNs often struggle with training instability and the challenge of balancing multiple loss terms, which typically requires extensive hyperparameter tuning. In this paper, we introduce Constraint Guided Physics-Informed Neural Networks (CGPINNs), a novel approach that leverages Constraint Guided Gradient Descent (CGGD) to train PINNs. CGPINN reframes the learning problem as a constrained optimization task, replacing complex hyperparameter balancing with more intuitive, semantically meaningful parameters. We also propose to add two sets of constraints derived from the PDE at the initial and boundary conditions, which prevent the model from converging to trivial solutions when using CGGD. Our experiments on a simulated heat diffusion problem demonstrate that CGPINN offers a more stable and robust training procedure, effectively learning the underlying physics without the need for expensive hyperparameter searches.

Keywords

Physics-Informed Neural Networks, Constraint-Guided Gradient Descent, Neuro-Symbolic AI, Constrained Optimization, Partial Differential Equations

1. Introduction

Neuro-symbolic AI aims to combine deep learning with knowledge-based systems, bridging the gap between statistical learning and symbolic reasoning. Physics-Informed Neural Networks (PINNs) fall under this paradigm, as they integrate prior knowledge, in the form of physically inspired differential equations, into the learning process. In PINNs, this physical knowledge is encoded through Partial Differential Equations (PDEs) and is embedded into the neural network training, enforcing physical laws or prior knowledge during the learning process into the model weights. This enables PINNs to learn solutions that are consistent not only with observational data but also with the governing physics of the problem.

PDEs play a fundamental role in modeling a wide range of physical phenomena across science and engineering, including fields like fluid dynamics, heat transfer and others [1, 2]. Traditional numerical methods, such as the finite element method, have long been the standard for solving PDEs [3]. While these methods are robust and well-established, they often face challenges when extended to high-dimensional problems and can suffer from high computational cost and very slow inference [4]. In contrast, PINNs represent a novel approach to solving PDEs by leveraging the expressive power and fast inference of deep neural networks [5].

Despite their potential, they are not without limitations. It is generally known that PINNs can be hard to train. A common challenge is the balancing of different loss components, such as the data-fitting term and the PDE residual term, which often requires careful and costly hyperparameter tuning, as an imbalance in these terms can lead to slow convergence or a failure to learn a good solution [6].

To address these challenges, we introduce Constraint Guided Physics-Informed Neural Networks (CGPINNs), which reformulates the training of PINNs as a constrained optimization problem. To solve this, we propose a novel training methodology based on Constraint Guided Gradient Descent (CGGD) [7]. CGGD is a learning framework that enables the training of deep learning models by minimizing an objective function while explicitly satisfying a set of constraints, including those involving continuous variables. This approach allows constraints to be enforced directly during training, thereby eliminating the need to manually tune weighting hyperparameters for balancing multiple loss terms.

This article is organised as follows. In Section 2, we discuss related work on addressing PINN training difficulties, including adaptive weighting strategies and architectural modifications. Section 3 details our methodology, starting with an introduction to the CGGD algorithm, followed by the new CGPINN method, its learning objective and the inclusion of initial and boundary condition constraints. Section 4 outlines heat diffusion experiments, covering the setup and physical configurations, data generation, evaluation metrics, network architecture, and the training process. Section 5 presents and discusses the results, comparing CGPINN's performance against a vanilla PINN baseline. Finally, section 6 provides the conclusion and outlines future work.

2. Related Work

The challenge of effectively training PINNs is widely recognized in the scientific machine learning community [8, 9, 10]. One of the core difficulties originates from the multi-objective nature of the standard training process, which relies on minimizing a composite loss function. This loss typically combines a data-fitting term with a physics-based residual term that enforces a PDE. These different objectives often result in a difficult training process, as the corresponding loss terms can have vastly different scales and gradient magnitudes. This "gradient pathology" [11] can cause the optimization process to be dominated by one objective, leading to slow convergence or a failure to find a physically meaningful solution.

Much of the existing research on multi-objective opti-

ANSyA 2025: 1st International Workshop on Advanced Neuro-Symbolic Applications, co-located with ECAI 2025.
*Corresponding author.
wout.rombouts@kuleuven.be (W. Rombouts); quinten.vanbaelen@kuleuven.be (Q. Van Baelen); peter.karsmakers@kuleuven.be (P. Karsmakers)
0009-0009-9612-3948 (W. Rombouts); 0000-0003-2863-4227 (Q. Van Baelen); 0000-0001-8119-6823 (P. Karsmakers)

mization, though not specific to PINNs, has focused on addressing this issue through adaptive weighting strategies [12]. These methods aim to dynamically adjust the relative importance of each loss component during training, in an effort to achieve a more balanced and effective optimization process. Examples include approaches such as GradNorm [13] that adjust weights based on the norm of the gradients of each loss term, attempting to ensure that all objectives contribute meaningfully to the weight updates. Other techniques, such as Learning Rate Annealing [11], assign different learning rates to different parts of the loss function and anneal them over time. While often effective, these methods can introduce new hyperparameters that require careful, and often expensive, tuning.

Another line of research has explored architectural modifications to improve PINN performance. Some studies have demonstrated that using specialized architectures or adaptive elements can enhance the network's ability to approximate complex solutions [14, 11]. Others have incorporated techniques like Fourier feature mappings [15] to help the network learn high-frequency components that are common in physical phenomena but are notoriously difficult for standard MLPs to capture. While beneficial, these architectural changes do not fundamentally alter the underlying training challenge of balancing competing loss objectives.

Our work takes a different path by reframing the PINN training problem from a multi-objective optimization task to a constrained optimization task. Instead of balancing competing objectives, we treat the physical laws as constraints that the solution must satisfy.

3. Methodology

At the core of our proposed methodology for training CGPINNs lies the CGGD algorithm [7]. We begin with a brief introduction to CGGD before detailing the CGPINN framework.

3.1. Constraint Guided Gradient Descent (CGGD)

CGGD is an optimization framework that enhances traditional gradient descent by incorporating hard inequality constraints into the training process. Unlike conventional approaches that rely solely on minimizing data-driven or multi-objective loss functions, CGGD introduces a mechanism to enforce a set of constraints throughout the training.

At each iteration, the method checks whether the current prediction is feasible, i.e., whether it belongs to the set of predictions that can be obtained from models satisfying all constraints on the training set. We refer to this set as the Feasible Region (FR). If the constraints are satisfied, the update proceeds as in standard gradient descent, optimizing the loss without modification. Consider at training iteration j the update of a set of model weights W_j during training. When constraints are violated, the update is guided not only by the gradient of the loss function but also by a corrective direction dir_C that steers the model towards the FR. Before combining these vectors, the constraint direction is matched to the gradient loss. The constraint direction is then scaled by a factor greater than 1 to ensure it dominates the update step. By default, this rescale factor is set to 1.5, although any value greater than 1 is sufficient. This approach guarantees that the updated model moves closer to the FR [7]. An illustration of this process for a two-weight update is shown in Fig. 1, where the loss gradient and constraint direction are shown in red and blue, respectively.

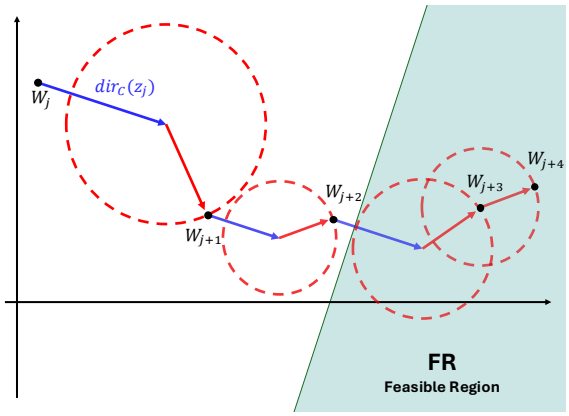

Figure 1: Graphical representation of CGGD update step. The red vector represents the loss-driven component of the update step, i.e., the standard loss gradient. The blue vector represents the constraint-driven component, pointing toward the FR and rescaled to have a norm 1.5 times that of the red vector. The red circle, with radius equal to the loss gradient norm, illustrates all possible update positions after the step.

3.2. Constraint-Guided PINN (CGPINN)

PINNs offer a framework for incorporating physical laws, expressed as differential equations, directly into the training of neural networks. The vanilla training objective of a PINN [5] is defined by

$$\begin{aligned}\underset{\boldsymbol{W}}{\operatorname{argmin}} \quad & \alpha \cdot L(\boldsymbol{\Phi}(\boldsymbol{x}_{obs}, \boldsymbol{W}), \boldsymbol{y}_{obs}) \\ & + (1-\alpha) \cdot \mathrm{PDE}(\boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{W}))\end{aligned}$$

where the loss function L measures the difference, typically by considering the Mean Squared Error (MSE), of the observations $\boldsymbol{y}_{obs}$ corresponding to the observed inputs $\boldsymbol{x}_{obs}$ with the predictions of the neural network $\boldsymbol{\Phi}$ with learnable weights $\boldsymbol{W}$, PDE measures how well the differential equation is obeyed by the neural network $\boldsymbol{\Phi}$ for (typically) both the observations and the unobserved collocation samples, in this work named $\boldsymbol{x}$. The smaller its value, the better it is obeyed. To balance both terms properly, a hyperparameter α is present which needs tuning as indicated before. Note that the PDE can also be an ordinary differential equation. This methodology is visualized in Fig. 2.

Instead of relying on the manually tuned weighting parameter α, CGPINNs reframes the training objective as a constrained optimization problem, where the governing PDE is enforced directly through explicit constraints. The new training objective of CGPINN is therefore defined as

$$\begin{aligned}\underset{\boldsymbol{W}}{\operatorname{argmin}} \quad & L\left(\boldsymbol{\Phi}\left(\boldsymbol{x}_{obs}, \boldsymbol{W}\right), \boldsymbol{y}_{obs}\right) \\ \text{s.t.} \quad & \mathrm{PDE}\left(\boldsymbol{\Phi}\left(\boldsymbol{x}, \boldsymbol{W}\right)\right) \leq \varepsilon,\end{aligned}$$

where the constraint tolerance ε specifies the allowable deviation of the neural network's predictions from the governing PDE. In this work, we initialize ε at a relatively large value (0.1), corresponding to a high FR, and progressively

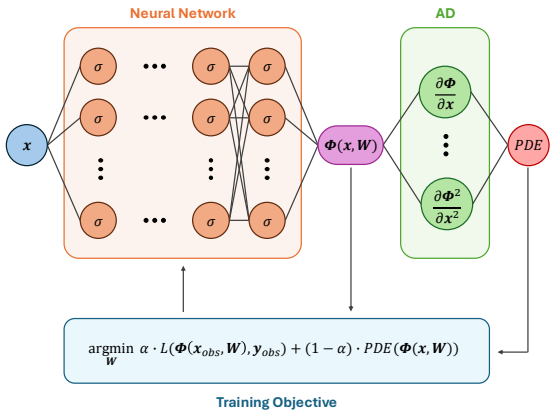

Figure 2: Visual of vanilla PINN network architecture. A network $\boldsymbol{\Phi}$ with weights $\boldsymbol{W}$ predicts outputs $\boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{W})$ from the inputs $\boldsymbol{x}$. Automatic Differentiation (AD) is used to calculate gradients required to compute the PDE residual. The learning objective is to minimize the weighted sum of the loss L and PDE residuals, with the weighting controlled by a parameter α.

reduce it during training using an exponential decay schedule ($\varepsilon_{t+1} = 0.9 \cdot \varepsilon_t$). This decay process continues until the constraints can no longer be fulfilled. The final value of ε, together with the fraction of constraints satisfied, characterizes how well the model adheres to the governing physical equations. In this way, ε attains a meaningful interpretation as part of a joint measure of the model's physical consistency.

Because this newly formulated PINN approach uses CGGD to solve the training objective, it can guarantee that the model converges to a solution that satisfies all constraints. However, the generic heat diffusion PDE (equation 1) still allows for a trivial solution to be found. If both gradients in this PDE are zero, the equation is satisfied, but the result is meaningless. In other words, as long as a trivial solution is part of a constraint, it can happen that the model will converge to it. Therefore, we must prevent this from happening to ensure a meaningful result is found.

To avoid converging to a trivial solution, where all derivatives computed via AD are zero, we introduce two additional sets of constraints that address special cases of the governing PDE. To illustrate these constraints, we consider the example of diffusion in a one-dimensional rod with Dirichlet boundary conditions, as presented in [16]. While this example is used for clarity, the proposed technique is general and not limited to any specific type of PDE. For the diffusion case under consideration, the governing PDE is given by

$$\frac{\partial}{\partial t} u(x,t) = \kappa \frac{\partial^2}{\partial x^2} u(x,t), \tag{1}$$

where κ is the thermal diffusivity coefficient. For the Dirichlet boundary conditions, the Initial Conditions (ICs) and the Boundary Conditions (BCs) are given by

$$\text{IC}: \quad u(x,0) = \sin\left(\frac{\pi}{L}x\right), \qquad \text{for } x \in [0, L], \tag{2}$$

$$\text{BC}: \quad u(0,t) = u(L,t) = 0, \qquad \text{for } t \in [0, T]. \tag{3}$$

The first additional set of constraints is obtained by substituting the function (2) into the right-hand side of (1). This yields

$$\frac{\partial}{\partial t} u(x,0) = \kappa \frac{-\pi^2}{L^2} \sin\left(\frac{\pi}{L}x\right), \quad \text{for } t = 0, x \in [0, L].$$

In other words, by combining the spatial partial derivatives of the initial conditions with the governing PDE, we derive an additional constraint. If the spatial derivatives are zero, this constraint will be violated. Consequently, the FR will exclude models that produce zero spatial derivatives, effectively preventing such trivial solutions. This set of constraints will be referred to from now on as ICCon.

Similarly, the second set of constraints is obtained by substituting the function in (3) into the left-hand side of (1). This yields

$$0 = \kappa \frac{\partial^2}{\partial x^2} u(x,t), \quad \text{for } x \in \{0, L\}, t \in [0, T].$$

As with the PDE constraint, a slack variable defined by ε, is introduced that will allow some tolerance on the constraint. To summarize, the optimization objective of CG-PINNs is defined by

$$\begin{aligned} \underset{\boldsymbol{W}}{\text{argmin}} \quad & L\left(\boldsymbol{\Phi}\left(\boldsymbol{x}_{obs}, \boldsymbol{W}\right), \boldsymbol{y}_{obs}\right) \\ \text{s.t.} \quad & \text{PDE}\left(\boldsymbol{\Phi}\left(\boldsymbol{x}, \boldsymbol{W}\right)\right) \leq \varepsilon, \\ & \text{ICCon}\left(\boldsymbol{\Phi}\left(\boldsymbol{x}_{obs}, \boldsymbol{W}\right)\right) \leq \varepsilon, \\ & \text{BCCon}\left(\boldsymbol{\Phi}\left(\boldsymbol{x}_{obs}, \boldsymbol{W}\right)\right) \leq \varepsilon. \end{aligned}$$

This constrained optimization problem can be solved directly by using CGGD [7]. In this work, the loss function L is defined as the MSE between the boundary samples and its ground truth values. The function PDE $\left(\boldsymbol{\Phi}\left(\boldsymbol{x}, \boldsymbol{W}\right)\right)$ internally uses the required derivatives of the network output with respect to the input variables, computed using AD. The resulting residual quantifies how well the PDE is satisfied at the sampled collocation points $\boldsymbol{x}$. The direction dir_C of the constraints is computed by calculating the derivative of PDE $\left(\boldsymbol{\Phi}\left(\boldsymbol{x}, \boldsymbol{W}\right)\right)$. To ensure balanced influence, this direction vector is scaled to have the same norm as the corresponding loss gradient, an approach analogous to the multi-head factor scaling used in [17]. A similar procedure is applied to the remaining constraint terms.

4. Experiments

To validate our method and compare it against a vanilla PINN, we implement a heat diffusion experiment based on [16]. The setup models one-dimensional heat diffusion in a rod of length L, capturing the temperature distribution $u(x,t)$ over time t. The governing physical process is described by the following PDE:

$$\frac{\partial u(x,t)}{\partial t} = \kappa \frac{\partial^2 u(x,t)}{\partial x^2}$$

Following [16], we explore several configurations of the rod length L and thermal diffusivity coefficient κ. Specifically, we investigate the parameter pairs $(L, \kappa) \in \{(5, 0.04), (5, 1), (1, 1), (1, 25)\}$. These configurations cover a broad range of physical regimes, from slow to fast diffusion with varying rod length L. To enable meaningful comparisons across these settings, each simulation is run over a time horizon defined by the diffusive time scale $\tau = \frac{L^2}{\kappa}$. This characteristic time scale allows us to normalize the simulation duration relative to the physical properties of each setup and ensure that the dynamics are compared over equivalent stages of diffusion.

The analytical solution of the PDE is

$$u(x,t) = \sin\left(\frac{\pi}{L}x\right) \cdot e^{-\frac{\pi^2}{\tau}t},$$

and is used to generate the training, validation, and test sets.

For the initial condition, spatial points are randomly sampled from the domain, where $u(x,0) = \sin\left(\frac{\pi}{L}x\right)$. Boundary values are sampled along the temporal domain at the Dirichlet boundaries $x = 0$ and $x = L$, with fixed temperatures $u(0,t) = u(L,t) = 0$. To enforce the PDE, collocation points are sampled within the spatiotemporal domain using Latin Hypercube Sampling (LHS) [18]. Fig. 3 provides a visual representation of the domain and the sampled points.

The training dataset consists of 128 initial and 128 boundary condition samples along with 1024 sampled collocation points. The validation and test datasets consists of 1024 collocation samples each. While the training and validation sets are re-sampled at each iteration to improve generalization, the test set is randomly sampled once to evaluate the model's final performance.

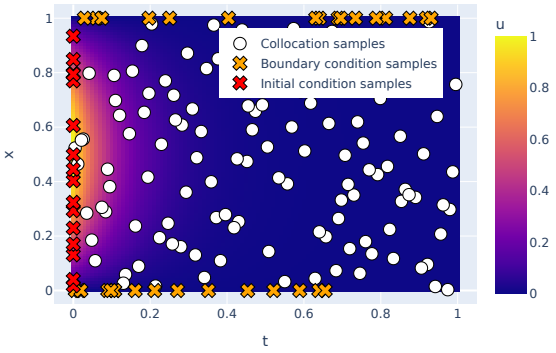

Figure 3: Representation of simulation data. Boundary samples indicated with crosses and collocation samples with dots. Background presents the ground truth solution of the heat diffusion PDE.

4.1. Evaluation Metric

The primary performance indicator for all models is the test loss, computed as the mean squared (prediction) error (MSE) on an unseen test set of 1024 points sampled with LHS from the spatio-temporal domain. The test loss reflects how well the model generalizes to unseen data and provides a direct measure of predictive accuracy. It is chosen as the main metric because it quantifies the discrepancy between the learned solution and the true analytical solution in a data-driven and interpretable way.

4.2. Network Architecture

A standard Multilayer Perceptron (MLP) model is used for the experiments. The network architecture consists of 4 hidden layers, each comprising 50 neurons. The hyperbolic tangent activation function is applied in all hidden layers, a common choice in both the PINN and regression literature due to its smoothness and its capacity to support higher-order derivatives, which are crucial for accurately modeling and solving PDEs.

The network is designed to approximate the solution $u(x,t)$ of the PDE. It takes a two-dimensional input vector, consisting of the spatial coordinate x and the temporal coordinate t, and produces a single scalar output representing the predicted temperature u.

The network's forward pass is augmented to not only compute the output u, but also to leverage AD to calculate with it the partial derivatives that are required to enforce the PDE, namely $\frac{\partial u}{\partial x}$, $\frac{\partial u}{\partial t}$, and $\frac{\partial^2 u}{\partial x^2}$. These derivatives are used to compute the PDE and boundary residuals, which are needed to enforce the physical constraints during training. Furthermore, to improve training stability, the input coordinates (x,t) are scaled to a normalized range before being processed by the network. The derivatives are calculated with respect to the original non-normalized input coordinates so that gradients and constraints can be formulated in the original physical or domain-specific units, and the PDE captures the relation between the original physical quantities.

4.3. Training Process

To ensure reproducibility, pseudo-random seeds are set for Python's built-in random module, as well as NumPy and PyTorch. Deterministic behavior is also enabled where applicable. A base seed, provided via the configuration or script arguments, serves as the foundation from which individual seeds are derived for each component that requires one. This is to ensure that independence between random number generators is maintained, avoiding unintended correlations while preserving reproducibility across runs.

We use the ADAM optimizer [19] in combination with an exponential learning rate scheduler, following a similar setup used in [16]. The initial learning rate is set to 10^{-3} and decays exponentially by a factor of 0.9. If no improvement occurs within 1000 consecutive epochs, the learning rate is reduced according to the predefined schedule.

The constraint tolerance parameter ε defines the allowable error for considering a constraint as satisfied. It is initially set to 1 and is progressively reduced when a constraint satisfaction rate of 95% is achieved. This dynamic adjustment enables automatic tuning of the tolerance for each experiment and helps prevent the use of overly strict tolerances, which could otherwise degrade the performance of CGGD. Whenever the tolerance is dynamically reduced, the learning rate is reset to the original value to allow the model to restart learning from a better initialization.

The training objective depends on the executed experiment. For the vanilla PINN, the objective is a direct minimization of the loss, which is a weighted sum of MSEs of the boundary data and the collocation data based on hyperparameter α:

$$\begin{aligned}\underset{\boldsymbol{W}}{\operatorname{argmin}} \quad & \alpha \cdot \frac{1}{N}\sum_{i=1}^{N}\left(\boldsymbol{x}_{\mathrm{obs},i} - \boldsymbol{\Phi}\left(\boldsymbol{x}_{\mathrm{obs},i}, \boldsymbol{W}\right)\right)^2 \\ & + (1-\alpha)\cdot\frac{1}{M}\sum_{j=1}^{M}\left(\mathrm{PDE}\left(\boldsymbol{\Phi}\left(\boldsymbol{x}_j, \boldsymbol{W}\right)\right)\right)^2\end{aligned}$$

For our CGPINNs method, the objective is determined by CGGD and is to satisfy the constraints while simultaneously minimizing the data loss:

$$\underset{\boldsymbol{W}}{\operatorname{argmin}} \ \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_{obs,i} - \boldsymbol{\Phi}\left(\boldsymbol{x}_{obs,i}, \boldsymbol{W}\right))^2$$

$$\text{s.t.} \quad \begin{cases} \text{PDE}\left(\boldsymbol{\Phi}\left(\boldsymbol{x}, \boldsymbol{W}\right)\right) \leq \varepsilon, \\ \text{ICCon}\left(\boldsymbol{\Phi}\left(\boldsymbol{x}, \boldsymbol{W}\right)\right) \leq \varepsilon, \\ \text{BCCon}\left(\boldsymbol{\Phi}\left(\boldsymbol{x}, \boldsymbol{W}\right)\right) \leq \varepsilon. \end{cases}$$

To facilitate the practical application of the CGGD algorithm, we rely on our custom-developed Python package Congrads[1]. This package encapsulates the specialized logic required to define learning objectives that include constraints. In particular, it provides a flexible and user-friendly interface for specifying constraints on selected parts of the neural network, and includes integrated checkpointing and logging mechanisms. By abstracting away the low-level implementation details, Congrads enables researchers and practitioners to focus on modeling and experimentation, making constraint-guided training accessible and efficient in practice.

Metrics were accumulated each epoch and averaged over 10-epoch intervals for logging and plot generation. Checkpointing and other scheduling functionalities, however, continued to rely on per-epoch metrics directly.

4.4. Implementation Details

The experiments are implemented in Python using PyTorch [20]. In the experiments, gradients are computed using PyTorch's AD functionality. Although it is not strictly necessary to calculate them in this way, AD provides a straightforward and reliable method, simplifying the implementation of gradient-based procedures and reducing the potential for manual errors in derivative calculations. This approach allows for efficient experimentation without compromising flexibility, as alternative gradient computation methods could also be employed if desired.

In PyTorch, gradients can only be computed with respect to leaf tensors. Consequently, it is not possible to compute the gradient of the constraints directly with respect to the model output, since output tensors are typically non-leaf nodes in the computational graph. To address this, we create a leaf tensor filled with ones, of shape $[batch\_size, 1]$, with gradient tracking enabled. This tensor is element-wise multiplied with the output to produce a new tensor for which gradients of the constraints can be computed. The resulting gradient can then be transformed back to the gradient with respect to the original output by dividing by the output values.

5. Results & Discussion

We compare the performance of our CGPINN approach against a standard vanilla PINN across four different physical configurations, varying the rod length L and thermal diffusivity κ. For the vanilla PINN, we report the test set MSE loss for five different values of the weighting hyperparameter α to showcase its sensitivity. For CGPINN, no such hyperparameter is needed and experiments were repeated over 3 different random seeds. In this case, we report the mean and standard deviation of the test set MSE loss.

[1]Congrads: A Python Toolbox for constraint-guided deep learning. Available at https://github.com/ML-KULeuven/congrads.

5.1. Baseline Performance

The results in Table 2 highlight the sensitivity of the vanilla PINN to the choice of the hyperparameter α. For each physical configuration, the performance varies significantly across different α values. For instance, in the $(L = 5, \kappa = 0.04)$ case, the Test MSE changes by an order of magnitude depending on α. The optimal value of α is inconsistent across different configurations; $\alpha = 0.5$ is best for $(L = 1, \kappa = 1)$, but it performs poorly for the high-diffusivity case $(L = 1, \kappa = 25)$. This demonstrates that finding the right balance requires a costly, problem-specific hyperparameter search. In the challenging high-diffusivity scenario, the vanilla PINN fails to converge to a good solution for any tested α, with MSE values several orders of magnitude higher than in other cases.

5.2. CGPINN Performance

The CGPINN framework demonstrates robust and stable performance without the need for manual loss weighting. As shown in the final column of Table 2, CGPINN achieves a relatively low MSE across all configurations. Most notably, in the two more challenging setups, $(L = 5, \kappa = 0.04)$ and $(L = 1, \kappa = 25)$, CGPINN significantly outperforms the best vanilla PINN configuration. In the high-diffusivity case where $\kappa = 25$, where the baseline failed, CGPINN successfully converges to an accurate solution with a test MSE of 7.53×10^{-9}. This good performance and stability can be attributed to two factors. First, by treating physics as explicit constraints, CGPINN avoids the delicate balancing act of multiple loss terms. Second, the inclusion of the 'ICCon' and 'BCCon' constraints helps guide the model away from trivial or physically inconsistent solutions, and may offer advantages in more challenging configurations. While for many of the configurations the vanilla PINN with a carefully tuned α achieves a lower mean error, CGPINN remains competitive while offering a much easier and more reliable training process.

Additionally, Table 1 presents the final constraint tolerances achieved by our CGPINN method across various experimental configurations. For each setting of L and κ, the reported value of ε indicates the tolerance within which 95% of the test samples satisfy all imposed constraints. This demonstrates the method's ability to consistently enforce constraints across different parameter regimes.

Table 1
Final achieved constraint tolerances ε of our CGPINN method for different experimental parameters. The values indicate that 95% of samples had all constraints satisfied with the tolerance ε.

Parameters		CGPINN
L	κ	ε
5	0.04	1.01×10^{-5}
5	1	1.34×10^{-4}
1	1	3.17×10^{-3}
1	25	1.78×10^{-1}

While CGPINNs show promising results, the current implementation is limited to a one-dimensional PDE. Therefore, further validation on higher-dimensional problems is needed for a more complete comparison, as this may introduce additional computational complexity and constraint formulation challenges.

Table 2
Test losses of baseline compared with CGPINN for different experimental parameters. For the baseline (vanilla PINN), results are shown for five different hyperparameter configurations to demonstrate sensitivity. CGPINN results are shown as a single representative value per configuration.

Parameters		Vanilla PINN					CGPINN
L	κ	$\alpha = 0.10$	$\alpha = 0.25$	$\alpha = 0.5$	$\alpha = 0.75$	$\alpha = 0.90$	
5	0.04	2.77×10^{-7}	2.95×10^{-7}	5.72×10^{-6}	2.13×10^{-5}	2.28×10^{-4}	$\mathbf{4.80 \times 10^{-8}}$
5	1	1.62×10^{-9}	3.68×10^{-10}	$\mathbf{3.10 \times 10^{-10}}$	1.79×10^{-9}	1.10×10^{-8}	2.36×10^{-8}
1	1	3.67×10^{-7}	2.78×10^{-7}	1.94×10^{-7}	1.98×10^{-9}	$\mathbf{1.11 \times 10^{-9}}$	3.00×10^{-8}
1	25	1.44×10^{-2}	1.49×10^{-2}	1.95×10^{-4}	6.18×10^{-5}	1.66×10^{-5}	$\mathbf{7.53 \times 10^{-9}}$

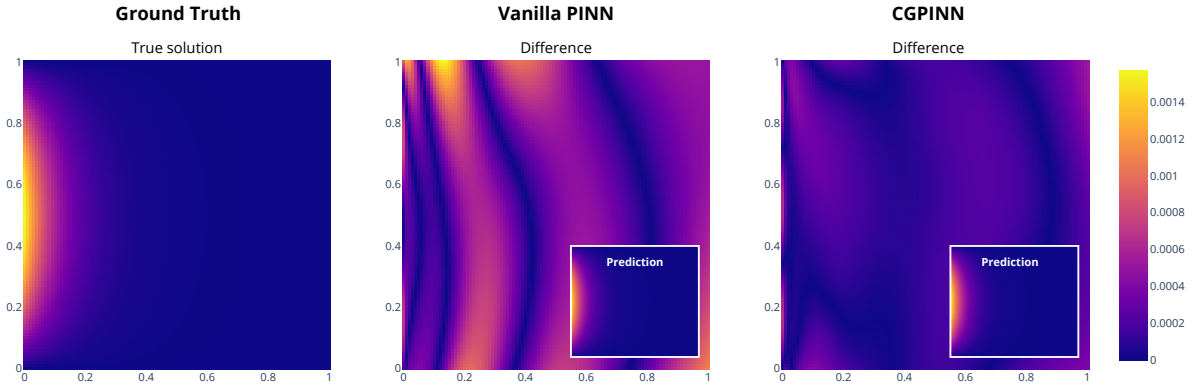

Figure 4: Visualization of predictions and differences to the ground truth for experiment with $L = 1$ and $\kappa = 1$. The left panel shows the analytical solution of the PDE; the center panel displays the vanilla PINN prediction (small) with hyperparameter $\alpha = 0.5$ and the corresponding difference from the analytical solution (large); the right panel presents the same results for our CGPINN method.

6. Conclusion & Future work

In this paper, the framework CGPINNs is introduced for training PINNs using constrained optimization. By leveraging CGGD, the proposed approach eases the need for delicate and costly hyperparameter tuning associated with balancing multiple loss terms in traditional PINNs. Two novel sets of constraints are proposed, ICCon and BCCon, which are derived from the governing PDE at the domain boundaries that prevent the model from learning trivial solutions.

The experiments on a 1D heat diffusion problem demonstrate that CGPINN provides a more stable and robust training procedure. It achieves good performance, and in challenging cases superior to, a well-tuned vanilla PINN, without requiring any sensitive weighting hyperparameters. This makes the process of developing and training PINNs simpler and more reliable.

One possible direction for future work is to apply CGPINN to more complex, multi-dimensional PDEs. Another line of follow-up research is the comparison of CGPINN to existing state-of-the-art techniques like [11, 13] and further investigate the interplay between constraint tolerance scheduling and learning rate scheduling to further improve convergence speed and solution accuracy.

Acknowledgments

This research was supported by the DTF-PINN SBO project of Flanders Make, the strategic research centre for the manufacturing industry of Flanders, Belgium and received funding from the Flemish Government (AI Research Program).

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT, Microsoft Copilot and Gemini in order to: Drafting content, Paraphrase and reword, Improve writing style, Grammar and spelling check. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] S. Cai, Z. Mao, Z. Wang, M. Yin, G. E. Karniadakis, Physics-informed neural networks (pinns) for fluid mechanics: a review, Acta Mechanica Sinica 37 (2021) 1727–1738. doi:10.1007/s10409-021-01148-1.

[2] S. Cai, Z. Wang, S. Wang, P. Perdikaris, G. E. Karniadakis, Physics-informed neural networks for heat transfer problems, Journal of Heat Transfer 143 (2021). doi:10.1115/1.4050542.

[3] M. Baccouch, A brief summary of the finite element method for differential equations, in: M. Baccouch (Ed.), Finite Element Methods and Their Applications, IntechOpen, London, 2021. doi:10.5772/intechopen.95423.

[4] T. G. Grossmann, U. J. Komorowska, J. Latz, C.-B. Schönlieb, Can physics-informed neural networks beat the finite element method?, 2023. doi:10.48550/ARXIV.2302.04107.

[5] M. Raissi, P. Perdikaris, G. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Jour-

nal of Computational Physics 378 (2019) 686–707. doi:10.1016/j.jcp.2018.10.045.

[6] S. Basir, Investigating and mitigating failure modes in physics-informed neural networks (pinns), 2022. doi:10.48550/ARXIV.2209.09988.

[7] Q. Van Baelen, P. Karsmakers, Constraint guided gradient descent: Training with inequality constraints with applications in regression and semantic segmentation, Neurocomputing 556 (2023) 126636. doi:10.1016/j.neucom.2023.126636.

[8] P. Rathore, W. Lei, Z. Frangella, L. Lu, M. Udell, Challenges in training pinns: A loss landscape perspective, 2024. doi:10.48550/ARXIV.2402.01868.

[9] A. S. Krishnapriyan, A. Gholami, S. Zhe, R. M. Kirby, M. W. Mahoney, Characterizing possible failure modes in physics-informed neural networks, 2021. doi:10.48550/ARXIV.2109.01050.

[10] A. Farea, O. Yli-Harja, F. Emmert-Streib, Understanding physics-informed neural networks: Techniques, applications, trends, and challenges, AI 5 (2024) 1534–1557. doi:10.3390/ai5030074.

[11] S. Wang, Y. Teng, P. Perdikaris, Understanding and mitigating gradient flow pathologies in physics-informed neural networks, SIAM Journal on Scientific Computing 43 (2021) A3055–A3081. doi:10.1137/20m1318043.

[12] R. Bischof, M. A. Kraus, Multi-objective loss balancing for physics-informed deep learning, Computer Methods in Applied Mechanics and Engineering 439 (2025) 117914. doi:10.1016/j.cma.2025.117914.

[13] Z. Chen, V. Badrinarayanan, C.-Y. Lee, A. Rabinovich, Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks, 2017. doi:10.48550/ARXIV.1711.02257.

[14] H. Bi, T. D. Abhayapala, Point neuron learning: a new physics-informed neural network architecture, EURASIP Journal on Audio, Speech, and Music Processing 2024 (2024). doi:10.1186/s13636-024-00376-0.

[15] M. Tancik, P. P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. T. Barron, R. Ng, Fourier features let networks learn high frequency functions in low dimensional domains, 2020. doi:10.48550/ARXIV.2006.10739.

[16] F. M. Rohrhofer, S. Posch, C. Gößnitzer, B. C. Geiger, Data vs. physics: The apparent pareto front of physics-informed neural networks, IEEE Access 11 (2023) 86252–86261. doi:10.1109/ACCESS.2023.3302892.

[17] Y. Tefera, Q. Van Baelen, M. Meire, S. Luca, P. Karsmakers, Constraint-guided learning of data-driven health indicator models: An application on the pronostia bearing dataset, 2025. doi:10.48550/ARXIV.2503.09113.

[18] M. D. McKay, R. J. Beckman, W. J. Conover, A comparison of three methods for selecting values of input variables in the analysis of output from a computer code, Technometrics 21 (1979) 239. doi:10.2307/1268522.

[19] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014. doi:10.48550/ARXIV.1412.6980.

[20] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, Pytorch: An imperative style, high-performance deep learning library, 2019. doi:10.48550/ARXIV.1912.01703.