LEARNING DIFFERENTIABLE SOLVERS FOR SYSTEMS WITH HARD CONSTRAINTS

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce a practical method to enforce linear partial differential equation (PDE) constraints for functions defined by neural networks (NNs), up to a desired tolerance. By combining methods in differentiable physics and applications of the implicit function theorem to NN models, we develop a differentiable PDE-constrained NN layer. During training, our model learns a family of functions, each of which defines a mapping from PDE parameters to PDE solutions. At inference time, the model finds an optimal linear combination of the functions in the learned family by solving a PDE-constrained optimization problem. Our method provides continuous solutions over the domain of interest that exactly satisfy desired physical constraints. Our results show that incorporating hard constraints directly into the NN architecture achieves much lower test error when compared to training on an unconstrained objective.

1 INTRODUCTION

Methods based on neural networks (NNs) have shown promise in recent years for physics-based problems (Raissi et al., 2019; Li et al., 2020; Lu et al., 2021a; Li et al., 2021). Consider a parameterized partial differential equation (PDE), $\mathcal{F}_\phi(u) = 0$, where $\mathcal{F}_\phi$ is a differential operator, and the PDE parameters ϕ and solution u are functions over a domain $\mathcal{X}$. The goal is to solve the following feasibility problem by training a NN with parameters $\theta \in \mathbb{R}^p$, i.e., find θ such that, for all ϕ describing PDEs in the considered family,

$$\mathcal{F}_\phi(u_\theta(\phi)) = 0. \tag{1}$$

Training such a model requires solving highly non-linear feasibility problems, even when $\mathcal{F}_\phi$ describes a linear PDE.

Current NN methods use two main training approaches to solve Equation 1. The first approach is strictly supervised learning, and the NN is trained on PDE solution data using a regression loss (Lu et al., 2021a; Li et al., 2020). In this case, the feasibility problem only appears through the data; it does not appear explicitly in the training algorithm. The second approach Raissi et al. (2019) aims to solve the feasibility problem by considering the relaxation,

$$\min_\theta \|\mathcal{F}_\phi(u_\theta)\|_2^2, \tag{2}$$

where the NN typically takes both PDE parameters and coordinates of the spatial and/or temporal domain of interest as input. Due to the self-supervised training procedure, this second approach does not require access to any PDE solution data. These two approaches have also been combined by having both a data fitting loss and the PDE residual loss (Li et al., 2021).

However, both of these approaches come with major challenges. The first approach requires potentially large amounts of PDE solution data, which may need to be generated through expensive numerical simulations or experimental procedures. It can also be challenging to generalize outside the training data, as there is no guarantee that the NN model has learned the relevant physics. For the second approach, recent work has highlighted that in the context of scientific modeling, the relaxed feasibility problem in Equation 2 is a difficult optimization problem (Krishnapriyan et al., 2021; Wang et al., 2021; Edwards, 2022). There are several reasons for this, including gradient imbalances in the loss terms (Wang et al., 2021) and ill-conditioning Krishnapriyan et al. (2021),

as well as only *approximate* enforcement of physical laws. In numerous scientific domains including fluid mechanics, physics, and materials science, systems are described by well-known physical laws, and breaking them can often lead to nonphysical solutions. Indeed, if a physical law is only approximately constrained (in this case, "soft constrained," as with popular penalty-based optimization methods), then the system solution may behave qualitatively differently or even fail to reach an answer.

In this work, we propose a method to overcome these challenges by solving the PDE-constrained problem in Equation 1 directly. We only consider the data-starved regime, i.e., we do not assume that any solution data is available on the interior of the domain. Of course, the method we develop can easily have a data fitting loss when more solution data are available. To solve Equation 1, we design a PDE-constrained NN layer to map PDE parameters to their solutions, such that the PDE constraints are enforced as "hard constraints." Once our model is trained, we can take new PDE parameters and solve for their corresponding solutions, while still enforcing the correct constraint.

In more detail, our main contributions are the following.

- We propose a method to enforce hard PDE constraints by creating a NN layer, which we call *PDE-Constrained-Layer* or PDE-CL. We make the PDE-CL differentiable by using implicit differentiation, thereby allowing us to train our model with popular gradient-based optimization methods. We explicitly write our solution function $u_\theta(\phi) = f_\theta(\phi)^\top \omega$, where the linear combination weighting ω is the output from our layer, and f_θ is a NN representing a functional basis. In this paper, we focus specifically on linear PDEs. In this setting, the vector ω solves a linear system which enforces the PDE constraint, over a given discretization of the domain, and up to a desired tolerance.
- After training a model, the functions f_θ in the basis are fixed. At inference time, our model then outputs a solution to a new problem by simply finding the optimal linear combination of the basis functions. To do so, we solve a linear system, on a lower resolution grid. We can then add fast optimized linear solvers for this system into our model at inference time, and also evaluate the model on a much higher resolution grid. This follows previous work in operator learning, and it does not require retraining for each new instance (Lu et al., 2021a; Li et al., 2020).
- We provide an empirical validation of our method on two problems representing different classes of PDEs. The Darcy Flow problem is an elliptic PDE on a stationary (steady-state) domain; and the convection problem is a hyperbolic PDE on a spatiotemporal domain. In both situations, we have no access to solution data on the interior spatial and/or temporal domain. We show that our approach has lower error than the soft constraint approach when predicting solutions for new, unseen test cases. Our approach also takes fewer iterations to converge to the correct solution.

2 Background and Related Work

The layer we design solves a constrained optimization problem corresponding to a PDE constraint. Here, we outline some relevant lines of work.

Dictionary learning. The problem we study can be seen as PDE-constrained dictionary learning. Dictionary learning Mairal et al. (2009) aims to learn an over-complete basis which can represent data well. Each datapoint will then be represented by a sparse subset of the learned basis. Since dictionary learning is a discrete method, it is not directly compatible with learning solutions to PDEs, as we need to be able to compute partial derivatives for the underlying learned functions. NNs allow us to do exactly this, as we can learn an overcomplete functional basis, represented by a NN, which is continuous and differentiable with regard to both its inputs and its parameters. We can therefore constrain the *dictionary* to satisfy the desired PDEs.

NNs and structural constraints. Using NNs to solve scientific modeling problems has gained interest in recent years Willard et al. (2020). NN architectures also can be designed such that they are tailored to a specific problem structure, e.g. local correlations in features LeCun et al. (1998); Bronstein et al. (2017); Hochreiter & Schmidhuber (1997), symmetries in data Cohen & Welling

(2016), convexity Amos et al. (2017), or monotonicity Sill (1997) with regard to input. This reduces the class of models to ones that enforce the desired structure exactly. For scientific problems, NN generalization can be improved by incorporating domain constraints into the ML framework, in order to respect the relevant physics. Common approaches have included adding PDE terms as part of the optimization loss function Raissi et al. (2019), using NNs to learn differential operators in PDEs such that many PDEs can be solved at inference time Li et al. (2020); Lu et al. (2021a), and incorporating numerical solvers within the framework of NNs Um et al. (2020). When an operator G is known such that $G(u)$ satisfies a given linear PDE for any function u, it is possible to directly parameterize Gaussian processes Lange-Hegermann (2018; 2021); Jidling et al. (2017) or neural networks Hendriks et al. (2020) to satisfy the PDE, and fit some desired loss function. Our work considers the setting where such an operator G is not known, and aims to find it algorithmically. Other previous work in PDE-solving has tried to enforce hard constraints by enforcing boundary conditions Lu et al. (2021b). We instead enforce the PDE constraint on the interior domain.

Implicit layers. We start by giving some background on implicit layers, i.e., NN layers made differentiable by application of the implicit function theorem (see Krantz & Parks (2002)). A deep learning layer is a differentiable, parametric function defined as $f_\theta : x \mapsto y$. For most deep learning layers, the two Jacobians $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial \theta}$ are computed using the chain rule. For these *explicit* layers, f_θ is usually defined as the composition of elementary operations for which the Jacobians are known. On the other hand, *implicit* layers create an implicit relationship between the inputs and outputs by computing the Jacobian using the implicit function theorem, rather than the chain rule. Specifically, if the layer has input $x \in \mathbb{R}^{d_{\text{in}}}$, output $y \in \mathbb{R}^{d_{\text{out}}}$ and parameters $\theta \in \mathbb{R}^p$, we suppose that y solves the following nonlinear equation $g(x, y, \theta) = 0$ for some g. Under mild assumptions, this defines an implicit function $f_\theta : x \mapsto y$. In our method, the forward function solves a constrained optimization problem. When computing the Jacobian of the layer, it is highly memory inefficient to differentiate through the optimization algorithm (i.e., all the steps of the iterative solver). Instead, by using the implicit function theorem, a set of linear systems can be solved to obtain the required Jacobians (see Amos & Kolter (2017); Barratt (2018); Blondel et al. (2021); Agrawal et al. (2019) and the Deep Implicit Layer NeurIPS 2021 tutorial[1] for more details). Implicit layers have been leveraged in many applications, including solving ordinary differential equations (ODEs) (Chen et al., 2018), optimal power flow Donti et al. (2021), and rigid many-body physics (de Avila Belbute-Peres et al., 2018).

Differentiable physics. In a related (but somewhat different) setting, recent work has aimed to make physics simulators differentiable. The adjoint method (Pontryagin et al., 1962) is classically used in PDE-constrained optimization, and it has been incorporated into NN training (Chen et al., 2018; Zhang et al., 2019). In this case, the assumption is that discrete samples from a function satisfying an unknown ODE are available. The goal is to learn the system dynamics from data. A NN model is used to approximate the ODE, and traditional numerical integration methods (Euler, Runge-Kutta) are applied on the output of the NN to get the function evaluation at the next timestep. The adjoint method is used to compute gradients with regard to the NN parameters through the obtained solution. The adjoint method also allows differentiating through physics simulators Degrave et al. (2019); Schoenholz & Cubuk (2020); Um et al. (2020). Access to the gradient information enables more efficient training of control and reinforcement learning models. Our setup is different. In our case, the underlying physical law(s) are known and the NN is used to approximate the solutions, under the assumption that there is no observational data in the interior of the solution domain.

3 METHODS

We describe the ML problem and the details of our method for enforcing PDE constraints within a NN.

3.1 PROBLEM SETUP

Our goal is to learn a mapping between a PDE parameter function $\phi : \mathcal{X} \to \mathbb{R}$ and the corresponding PDE solution $u(\phi) : \mathcal{X} \to \mathbb{R}$, where the domain $\mathcal{X}$ is an open subset of $\mathbb{R}^d$ for some d. The PDE parameters ϕ could be parameter functions such as initial condition functions, boundary condition functions, forcing functions, and/or physical properties such as wavespeed, diffusivity, and viscosity.

[1] http://implicit-layers-tutorial.org/

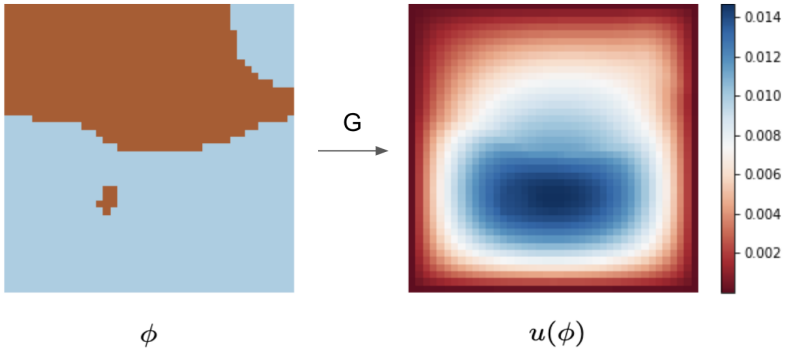


Figure 1: **Mapping PDE parameters ϕ to PDE solutions $u(\phi)$.** The goal of our model is to learn a mapping $G : \phi \mapsto u(\phi)$, without access to solution data. As an example, we study the Darcy Flow PDE, which describes the chemical engineering problem of fluid flow through a porous medium Darcy (1856). The system is composed of two materials in a given spatial domain $\mathcal{X} = (0,1)^2$, each with specific diffusion coefficients which depend on the position. The left figure shows ϕ, which encodes the locations and diffusion properties of the two materials. The right figure shows the corresponding solution $u(\phi)$. The function u is a solution of the Darcy Flow PDE with diffusion coefficients ϕ if, for all $(x, y) \in (0,1)^2$, it satisfies $-\nabla \cdot (\phi(x, y)\nabla u(x, y)) = 1$. The boundary condition is $u(x, y) = 0,\ \forall (x, y) \in \partial(0,1)^2$.

Let $\mathcal{F}_\phi$ be a functional operator such that for all PDE parameter functions ϕ, the solution $u(\phi)$ satisfies $\mathcal{F}_\phi(u(\phi)) = 0$. We search for a vector $\theta \in \mathbb{R}^p$ which parameterizes the NN u_θ.

The inputs to our NN u_θ vary depending on the domain of interest and the PDE parameters. In the simplest case, the input is a pair $(x, \phi(x))$, where $x \in \mathcal{X}$ and $\phi(x)$ is the value of the PDE parameter at x. The output of the NN is the value of the corresponding approximated solution $u_\theta(\phi)$, taken at x.

We show the mapping $G : \phi \mapsto u(\phi)$ that we want to learn in Figure 1.

Importantly, we consider only the unsupervised learning setting, where solution data in the interior domain of the PDE is not available for training the model. In this setting, the training is done by only enforcing the PDE, and the initial and/or boundary conditions.

3.2 A DIFFERENTIABLE CONSTRAINED LAYER FOR ENFORCING PDES

There are two main components to our model. The first component is a NN, denoted by f_θ, where θ are the parameters that are to be learned. Given the inputs described in Section 3.1 to f_θ, the NN then outputs a vector in $\mathbb{R}^N$. Note that the output dimension of f_θ is N, a hyperparameter.

The second component of our model, the PDE-constrained layer or PDE-CL, is our main design contribution. We implement a NN layer that performs a linear combination of the N outputs from the first component, such that the linear combination exactly satisfies the PDE on all points x_i in the discretized domain. Specifically, let ω be the weights in the linear combination given by the PDE-CL. The output of our system is $u_\theta = \sum_{i=1}^{N} \omega_i f_\theta^i$, where f_θ^i is the i-th coordinate output by the NN f_θ. We now describe the training details of the PDE-CL.

Forward pass of the differentiable constrained layer. In this paper, we focus on *linear* PDEs, such that solving the PDE constraint can be done efficiently by solving linear systems. Here, we describe a homogeneous system, where $\mathcal{F}_\phi$ is a linear operator. Our method can also handle non-homogeneous systems where $\mathcal{F}_\phi$ is an affine operator (we describe such a system in Section 4.1). $\mathcal{F}_\phi$ is linear when for any two functions u, v from $\mathcal{X}$ to $\mathbb{R}$, and any scalar $\lambda \in \mathbb{R}$, we have that,

$$\mathcal{F}_\phi(u + \lambda v) = \mathcal{F}_\phi(u) + \lambda \mathcal{F}_\phi(v). \tag{3}$$

Then, we define the PDE-CL to find the optimal linear weighting ω of the N 1D functions encoded by the first NN component, over the set of sampled inputs $x_1, \ldots, x_n$. The vector $\omega \in \mathbb{R}^N$ solves the following linear equation system,

$$\forall j = 1, \ldots, n, \quad \mathcal{F}_\phi\left(\sum_{i=1}^{N} \omega_i f_\theta^i\right)(x_j) = 0 \iff \sum_{i=1}^{N} \omega_i \mathcal{F}_\phi(f_\theta^i)(x_j) = 0. \tag{4}$$

This linear system is a discretization of the PDE $\mathcal{F}_\phi(u_\theta) = 0$; we enforce the PDE exactly at the sampled points $x_1, \ldots, x_n$. The linear system has n constraints and N variables, and so the chosen value of N must be greater than n. This is because we want the system to be underdetermined to allow finding nontrivial solutions.

We consider well-posed PDEs, following previous work exploring NNs and PDEs Raissi et al. (2019); Li et al. (2020); Wang et al. (2021). However, the optimization problem for PDE-CL should always have a solution because we suppose that $N > n$, where N is the number of basis functions and n is the number of interior points. In cases where there are initial/boundary conditions, this allows us to have non-trivial solutions becuase if $n < N$, then the zero solution is feasible for the linear PDE constraint since the initial and boundary conditions are only enforced through the constrained objective in the quadratic problem in the PDE-CL. When there is only a linear constraint, if $n < N$, our PDE-CL would output a solution to the least-squares system $\min_\omega \|A\omega - b\|^2$. In this case, it would be beneficial to keep the PDE loss in the overall objective function, to guide the basis functions of the NN, so that b is in the span of A. Therefore, even if $n < N$, our model may still output meaningful solutions after training.

Backward pass of the differentiable constrained layer. To incorporate the PDE-CL into an end-to-end differentiable system that can be trained by first-order optimization methods, we need to compute the gradients of the full model using an autodiff system (Bradbury et al., 2018). To do this, we must compute the Jacobian of the layer.

The PDE-CL solves a linear system of the form $g(\omega, A, b) = A\omega - b$ in the forward pass, where $A \in \mathbb{R}^{n \times N}$, $\omega \in \mathbb{R}^N$, $b \in \mathbb{R}^n$. Differentiating g with respect to A and b using the chain rule gives the following linear systems, which must be satisfied by the Jacobians $\frac{\partial \omega}{\partial A}$ and $\frac{\partial \omega}{\partial b}$,

$$\forall i, k \in 1, \ldots, N, \quad \forall j \in 1, \ldots, n, \quad 0 = \frac{\partial \omega_i}{\partial A_{jk}} = \left(\omega_k + A_j^\top \frac{\partial \omega}{\partial A_{jk}}\right) \mathbb{1}_{i=j}, \tag{5}$$

$$\forall i \in 1, \ldots, N, \quad \forall j \in 1, \ldots, n, \quad 0 = \frac{\partial g_i}{\partial b_j} = A_i^\top \frac{\partial \omega}{\partial b_j} - \mathbb{1}_{i=j}, \tag{6}$$

where $\mathbb{1}_{i=j}$ is 1 when $i = j$ and 0 otherwise. Given the size and and conditioning of our problems, we cannot directly solve the linear system. Therefore, we use an indirect solver (such as conjugate gradient (Hestenes & Stiefel, 1952) or GMRES (Saad & Schultz, 1986)) for both the forward system $A\omega = b$ and the backward system given by Equation 5 and Equation 6. We use the JAX autodiff framework (Bradbury et al., 2018; Blondel et al., 2021) to implement the full model. We include an ablation study in B.2 to evaluate the quality of functions in our basis. We also include an analysis and additional information on the enforcement of the hard constraints in A.

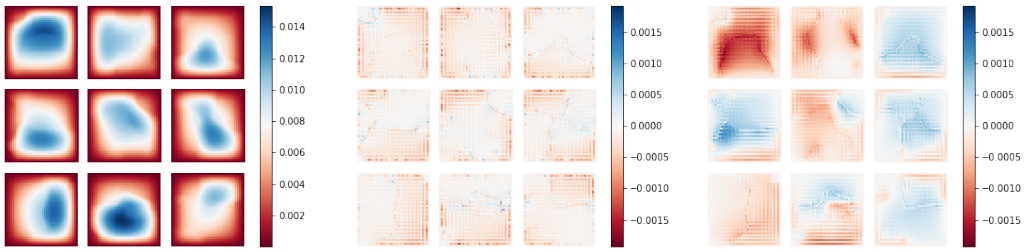
Inference procedure. After training our model and amortizing the computation, at inference time, we map new test sample PDE parameters to PDE solution.

Notes about how interpolation can be done on a much finer grid after the function is fit.

4 Experimental results and implementation

We test the performance of our model on two different scientific problems: 2D Darcy Flow (Section 4.1); and 1D convection (Section 4.2). In each case, the model is trained without access to any solution data in the interior solution domain. The training set contains 1000 PDE parameters ϕ. The model is then evaluated on a separate test set with $M = 50$ PDE parameters ϕ that are not seen during training. We compare model results on the test set using two metrics: relative L_2 error $\frac{1}{M}\sum_{i=1}^{M} \frac{\|u_\theta(\phi_i) - u(\phi_i)\|_2}{\|u(\phi_i)\|_2}$; and the PDE residual loss $\frac{1}{M}\sum_{i=1}^{M} \|\mathcal{F}_{\phi_i}(u_\theta)\|^2$, which measures how well the PDE is enforced on the interior domain. We demonstrate that our constrained NN architecture generalizes much better on the test set than the comparable unconstrained model.[2]

[2]We used a single Titan RTX GPU for each run in our experiments, for less than 15h.



(a) Target (b) Hard-constrained difference (c) Soft-constrained difference

Figure 2: **Heatmaps of Darcy Flow example test set predictions.** We compare our hard-constrained model and the baseline soft-constrained model on a test set of new diffusion coefficients ν. The NN architectures are the same except for our additional PDE-CL in the hard-constrained model. (a) Target solutions of a subset of PDEs in the test set. (b) Difference between the predictions of our hard-constrained PDE-CL model and the target solution. (c) Difference between the predictions of the baseline soft-constrained model and the target solution. Over the test dataset, our model achieves $\mathbf{3.39\% \pm 0.03\%}$ relative error and $\mathbf{0.0070 \pm 0.0002}$ interior domain test loss. In contrast, the soft-constrained model only reaches $\mathbf{3.83\% \pm 0.21\%}$ relative error and $\mathbf{0.3233 \pm 0.0192}$ interior domain test loss. While the heatmaps show a subset of the full test set, the standard deviation across the test set for our model is very low, indicating that these results are consistent across test samples.

4.1 2D Darcy Flow

We look at the steady-state 2D Darcy flow problem, which describes, for example, fluid flow through porous media. In this section, the PDE parameter (denoted by ϕ in the previous sections as a general variable) is $\nu \in L^\infty((0,1)^2; \mathbb{R}_+)$, a diffusion coefficient. The problem is formulated as follows:

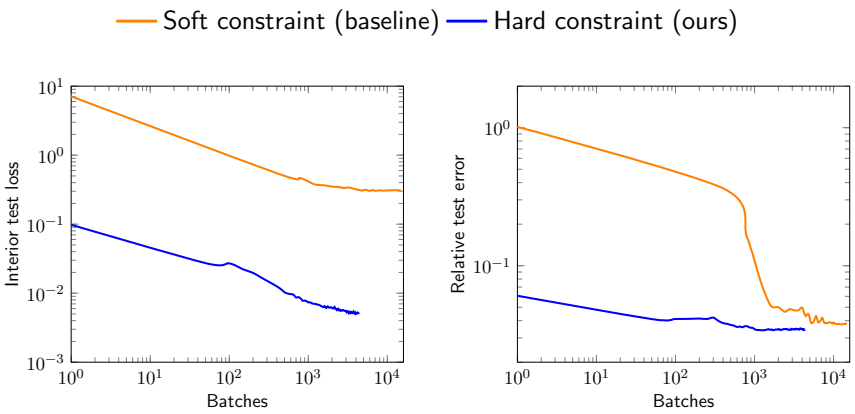
$$\begin{aligned} -\nabla \cdot (\nu(x)\nabla u(x)) &= f(x), \qquad \forall x \in (0,1)^2, \\ u(x) &= 0, \qquad \forall x \in \partial(0,1)^2. \end{aligned} \tag{7}$$

Here, f is the forcing function ($f \in L^2((0,1)^2; \mathbb{R})$, and $u(x) = 0$ is the boundary condition. The differential operator from the general set up is then $F_\nu(u) = -\nabla \cdot (\nu(x)\nabla u(x))$. Given a set of variable coefficients ν, the goal is to predict the correct solution u. The $\nu(x)$ values are generated from a Gaussian and then mapped to two values, corresponding to the two different materials in the system (such as the fluid and the porous medium). We follow the data generation procedure from Li et al. (2020).

Experiment setup and implementation details. We use the Fourier Neural Operator (FNO) (Li et al., 2020) architecture trained using a PDE residual loss as the baseline model ("soft-constrained"). Our model uses the FNO architecture and adds our PDE-CL ("hard-constrained"). The domain $(0,1)^2$ is discretized over $n_x \times n_y$ points. For each point on this grid, the model takes as input the coordinates, $x \in (0,1)^2$, and the corresponding values of the diffusion coefficients, $\nu(x)$. The boundary condition is satisfied by using a mollifier function Li et al. (2020), and so we only require fitting the PDE. We use a constant forcing function f equal to 1. Fitting the NN requires finding NN parameters, θ, which solve,

$$\begin{aligned} &\min_\theta \sum_\nu \|\mathcal{F}_\nu(u_\theta) - f(x)\|_2^2 \\ &\text{s.t. } \forall \nu,\ \mathcal{F}_\nu(u_\theta) = f(x). \end{aligned} \tag{8}$$

By design, the objective function for feasible parameters θ is zero. While numerical issues may prevent exact feasibility, solving the equality constrained problem by additionally minimizing the PDE residual helps the training procedure. We include an ablation study in B looking at this. The soft-constrained training method is trained only by minimizing the PDE residual. We train the FNO model using the same hyperparameters as Li et al. (2021). We denote the FNO model part of the



(a) PDE residual loss on test set

(b) Relative error on test set

Figure 3: **2D Darcy Flow: Error on test set during training.** We train a NN architecture with the PDE residual loss function ("soft constraint" baseline), and the same NN architecture with our PDE-CL ("hard constraint"). During training, we track error on the test set, which we plot on a log-log scale. (a) PDE residual loss on the test set, during training. This loss measures how well the PDE is enforced. (b) Relative error on the test set, during training. This metric measures the distance between the predicted solution and the target solution obtained via finite differences. Both measures show that our hard-constrained PDE-CL model starts at a much lower error (over an order of magnitude lower) on the test set at the very start of training, and continues to decrease as training proceeds. This is particularly visible when tracking the PDE residual test loss. See Figure 2 caption and text for the final error values.

architecture as f_θ. The PDE-CL constrains the output of the FNO model by solving the linear system,

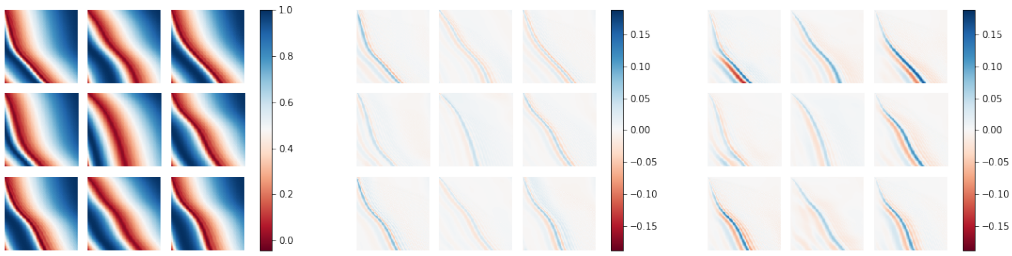
$$\forall \nu,\ \mathcal{F}_\nu(f_\theta)^\top \omega = f(x). \tag{9}$$

To train our model, we compute the Jacobian of this layer via implicit differentiation with respect to this linear system equation.

Results. We plot example heatmaps from the test set in Figure 2. We compare visually how close our hard-constrained model is to the target solution (Figure 2b), and how close the soft-constrained baseline model is to the target solution (Figure 2c). Our hard-constrained model is much closer to the target solution, as indicated by the difference plots mostly being white (corresponding to zero difference).

During the training procedure for both hard- and soft-constrained models, we track error on an unseen test set of PDE solutions with different PDE parameters from the training set. We show these error plots in Figure 3. In Figure 3a, our model starts at a PDE residual test loss value two orders of magnitude smaller than the soft constraint baseline. The PDE residual test loss continues to decrease as training proceeds, remaining significantly lower than the baseline. Similarly, in Figure 3b, we show the curves corresponding to the relative error and the PDE residual loss metric on the test dataset. Our model starts at a much smaller relative error immediately and continues to decrease, achieving a final lower test relative error.

On the test set, our model achieves $\mathbf{3.39\%} \pm \mathbf{0.03\%}$ relative error, versus $\mathbf{3.83\%} \pm \mathbf{0.21\%}$ for the soft-constrained baseline model. Our model also achieves $\mathbf{0.007} \pm \mathbf{0.0002}$ for the PDE residual test loss, versus $\mathbf{0.3233} \pm \mathbf{0.0192}$. Our model has almost two orders of magnitude lower PDE residual test loss, and it has significantly lower standard deviation. Note that while the example heatmaps show a subset of the full test set, the standard deviation across the test set for our model is very low, indicating that these results are consistent across test samples.



(a) Target (b) Hard-constrained difference (c) Soft-constrained difference

Figure 4: **Heatmaps of 1D convection example test set predictions.** We compare our hard-constrained model and the baseline soft-constrained model on a test set of new wavespeed parameters β. Both architectures are the same, except for our additional PDE-CL in the hard-constrained model. (a) Target solutions of a subset of PDEs in the test set. (b) Difference between the predictions of our hard-constrained model and the target solution. (c) Difference between the predictions of the baseline soft-constrained model and the target solution. Over the test dataset, our model achieves $\mathbf{1.77}\% \pm \mathbf{0.11}\%$ relative error and $\mathbf{0.399} \pm \mathbf{0.068}$ interior domain test loss. In contrast, the soft-constrained model only reaches $\mathbf{2.59}\% \pm \mathbf{0.15}\%$ relative error and $\mathbf{7.6} \pm \mathbf{1.2}$ interior domain test loss. The errors in both models are concentrated around the "sharp" features in the solution, but these errors have higher magnitude in the soft-constrained model.

4.2 1D CONVECTION

We now study a 1D convection problem, describing transport phenomena. The problem can be formulated as follows:

$$\begin{aligned} &\frac{\partial u(x,t)}{\partial t} + \beta(x)\frac{\partial u(x,t)}{\partial x} = 0, && x \in (0,1), t \in (0,1), \\ &h(x) = \sin(\pi x), && x \in (0,1) \\ &g(t) \ = \sin\left(\frac{\pi}{2}t\right), && t \in (0,1). \end{aligned} \tag{10}$$

Here, $h(x)$ is the initial condition (at $t = 0$), $g(t)$ is the boundary condition (at $x = 0$), and $\beta(x)$ represents the variable coefficients (denoted by ϕ in Section 3). Given a set of variable coefficients, $\beta(x)$, and spatiotemporal points (x_i, t_i), the goal is to predict the correct solution $u(x,t)$. The $\beta(x)$ values are generated in the same manner as in Wang et al. (2021) via $\beta(x) = v(x) - \min_x v(x) + 1$, where $v(x)$ is generated from a Gaussian random field with a length scale of 0.2.

Experiment setup and implementation details. We use a physics-informed DeepONet baseline model (Wang et al., 2021), trained with the PDE residual loss. Our hard-constrained model is composed of stacked dense layers and our PDE-CL, which allows a fair comparison. In this setting, the inputs to the models are a set of $((x_1, t_1), \ldots, (x_n, t_n))$ sampled points within the domain $\mathcal{X}$ (interior points) and the corresponding $\beta(x)$ values. We also require a set $((x_{n+1}, t_{n+1}), \ldots, (x_{n+n'}, t_{n+n'}))$ of sampled points on the initial condition ($t = 0$) and boundary condition ($x = 0$).

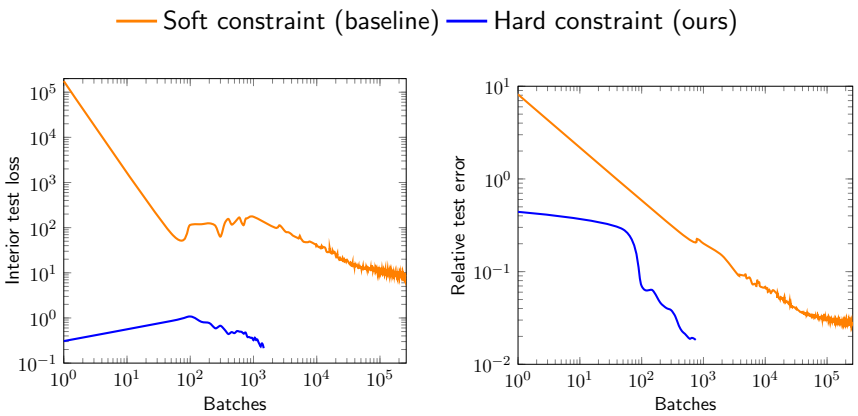
The training optimization problem is formulated as follows:

$$\min_{\theta} \sum_{\beta} \mathcal{L}_\beta(u_\theta) + \|\mathcal{F}_\beta(u_\theta)\|_2^2 \ \text{ s.t. } \forall \beta,\ \mathcal{F}_\beta(u_\theta) = 0, \tag{11}$$

with,

$$\mathcal{L}_\beta(u_\theta) = \frac{1}{2}\sum_{i=1}^{n'} (u_\theta(\beta, x_{n+i}, t_{n+i}) - u(\beta, x_{n+i}, t_{n+i}))^2,$$

$$\mathcal{F}_\beta(u_\theta) = \left(\frac{\partial u_\theta(\beta, x_1, t_1)}{\partial t} + \beta(x_1)\frac{\partial u_\theta(\beta, x_1, t_1)}{\partial x} \quad \ldots \quad \frac{\partial u_\theta(\beta, x_n, t_n)}{\partial t} + \beta(x_n)\frac{\partial u_\theta(\beta, x_n, t_n)}{\partial x}\right)^\top,$$



(a) PDE residual loss on test set

(b) Relative error on test set

Figure 5: **1D convection: Error on test set during training.** We train a NN with the PDE residual loss function ("soft constraint" baseline) and the same NN architecture with our PDE-CL ("hard constraint"). During training, we track error on the test set, which we plot on a log-log scale. (a) PDE residual loss on the test set, during training. (b) Relative error on the test set, during training. Both measures show that our hard-constrained model starts at a much lower error on the test set at the very start of training. The relative error decreases in many fewer iterations for the hard-constrained model.

where θ corresponds to parameters of the NN, $u(x,t)$ is the solution at the initial and boundary conditions, and $\mathcal{F}(u_\theta)$ is the PDE constraint that must be satisfied. The loss term $\mathcal{L}(u_\theta)$ is a regression loss over the initial and boundary conditions. Similar to the Darcy Flow example, we penalize the PDE residual in the loss. Note that while for any feasible point θ, the residual is zero, we found this to help training even in the hard-constrained model. Since the equality constraint is only enforced up to a given tolerance, the θ parameters may not be exactly feasible.

The forward pass of the PDE-CL solves the following equality constrained problem,

$$\min_{\omega} \ \mathcal{L}(f_\theta^\top \omega) \quad \text{s.t.} \quad \mathcal{F}_\beta(f_\theta)^\top \omega = 0, \tag{12}$$

where f_θ refers to the outputs of the base NN, on which we stack the PDE-CL. Since the initial/boundary condition regression loss uses a quadratic penalty, this equality constrained problem is in fact a convex equality constrained quadratic problem (EqQP), which is equivalent to a linear system. We solve this linear system using GMRES (Saad & Schultz, 1986). Similar to the Darcy flow case, during the training procedure, we compute the Jacobian via implicit differentiation with respect to Equation 12.

Results. We plot example heatmaps from the test set in Figure 4. We visually compare how close our hard-constrained model is to the target solution (Figure 4b), and similarly for the soft-constrained baseline model (Figure 4c). The solution found by our hard-constrained model is much closer to the target solution than the solution found by the baseline model, and our model captures "sharp" features in the solution visibly better than the baseline model.

During the training procedure for both hard- and soft-constrained models, we track error on an unseen test set of PDE solutions with different PDE parameters from the training set. We show these error plots in Figure 5. In Figure 5a, the PDE residual loss for the hard-constrained model starts close to six orders of magnitude lower than for the soft-constrained model, and it continues to remain low. In Figure 5b, we track the the relative error with respect to the target solution obtained via a Lax-Wendroff scheme. Similarly, we see that our model starts at much smaller relative error immediately and also continues to decrease. Our model achieves $\mathbf{1.77\% \pm 0.11\%}$ relative error and $\mathbf{0.399 \pm 0.068}$ PDE residual test loss, versus $\mathbf{2.59\% \pm 0.15\%}$ and $\mathbf{7.6 \pm 1.2}$ for the soft-constrained baseline model. On the relative error metric, our model achieves a 32% improvement over the soft-constrained model. Once again, the standard deviations of our model are very low, indicating that error is consistently low across test samples, and that the difference between the two models is significant.

5 CONCLUSIONS

In this work, we study the problem of training NNs to map PDEs within a given parameterized family to their corresponding solutions. We consider the unsupervised setting, where no solution data is available on the interior domain during training. We develop a method to enforce hard PDE constraints when training a NN. Considering linear PDEs specifically, we design a differentiable PDE-constrained layer (PDE-CL). This layer can be added to any NN architecture to enforce PDE constraints exactly. The full model can then be trained end-to-end. We evaluate our proposed model on two problems representing different physical settings: a 2D Darcy Flow problem, which describes fluid flow; and a 1D convection problem, which describes transport phenomena. Compared to the baseline soft-constrained model, our model can be trained in fewer iterations, achieves lower PDE residual error (measuring how well the PDE is enforced on the interior domain), and lower relative error with respect to target solutions generated by numerical solvers.

Our approach for enforcing PDE hard constraints for NNs is tailored to linear PDEs. Future work could study extending our method to nonlinear PDEs, which would require developing an approach to implicitly differentiate a nonlinearly constrained optimization problem. Additionally, scaling our method to larger numbers of interior points would be an interesting future direction. There are known methods for fast linear system solving (for example, preconditioning Amos & Kolter (2017) and sketching Gower & Richtárik (2015)) that could be used to assist in this.

REFERENCES

Akshay Agrawal, Brandon Amos, Shane T. Barratt, Stephen P. Boyd, Steven Diamond, and J. Zico Kolter. Differentiable convex optimization layers. In NeurIPS, 2019.

Brandon Amos and J. Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In ICML, 2017.

Brandon Amos, Lei Xu, and J. Zico Kolter. Input convex neural networks. In Doina Precup and Yee Whye Teh (eds.), Proceedings of the 34th International Conference on Machine Learning, volume 70 of Proceedings of Machine Learning Research, pp. 146–155. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/amos17b.html.

Shane T. Barratt. On the differentiability of the solution to convex optimization problems. arXiv: Optimization and Control, 2018.

Mathieu Blondel, Quentin Berthet, Marco Cuturi, Roy Frostig, Stephan Hoyer, Felipe Llinares-Lopez, Fabian Pedregosa, and Jean-Philippe Vert. Efficient and modular implicit differentiation. ArXiv, abs/2105.15183, 2021.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur D. Szlam, and Pierre Vandergheynst. Geometric deep learning: Going beyond euclidean data. IEEE Signal Processing Magazine, 34: 18–42, 2017.

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Kristjanson Duvenaud. Neural ordinary differential equations. ArXiv, abs/1806.07366, 2018.

Taco Cohen and Max Welling. Group equivariant convolutional networks. In ICML, 2016.

Henry Darcy. Les fontaines publiques de la ville de Dijon : exposition et application des principes à suivre et des formules à employer dans les questions de distribution d'eau. Victor Dalmont, 1856.

Filipe de Avila Belbute-Peres, Kevin Smith, Kelsey Allen, Josh Tenenbaum, and J Zico Kolter. End-to-end differentiable physics for learning and control. Advances in neural information processing systems, 31:7178–7189, 2018.

Jonas Degrave, Michiel Hermans, Joni Dambre, and Francis wyffels. A differentiable physics engine for deep learning in robotics. Frontiers in Neurorobotics, 13, 2019. ISSN 1662-5218. doi: 10.3389/fnbot.2019.00006. URL https://www.frontiersin.org/article/10.3389/fnbot.2019.00006.

Priya L Donti, David Rolnick, and J Zico Kolter. Dc3: A learning method for optimization with hard constraints. arXiv preprint arXiv:2104.12225, 2021.

Chris Edwards. Neural networks learn to speed up simulations. Communications of the ACM, 65 (5):27–29, 2022.

Robert M Gower and Peter Richtárik. Randomized iterative methods for linear systems. SIAM Journal on Matrix Analysis and Applications, 36(4):1660–1690, 2015.

Johannes N. Hendriks, Carl Jidling, Adrian G. Wills, and Thomas Bo Schön. Linearly constrained neural networks. ArXiv, abs/2002.01600, 2020.

Magnus R. Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear systems. Journal of research of the National Bureau of Standards, 49:409–435, 1952.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural Computation, 9:1735–1780, 1997.

Carl Jidling, Niklas Wahlstrom, Adrian G. Wills, and Thomas Bo Schön. Linearly constrained gaussian processes. In NIPS, 2017.

Steven George Krantz and Harold R Parks. The implicit function theorem: history, theory, and applications. Springer Science & Business Media, 2002.

Aditi Krishnapriyan, Amir Gholami, Shandian Zhe, Robert Kirby, and Michael W Mahoney. Characterizing possible failure modes in physics-informed neural networks. Advances in Neural Information Processing Systems, 34, 2021.

Markus Lange-Hegermann. Algorithmic linearly constrained gaussian processes. In NeurIPS, 2018.

Markus Lange-Hegermann. Linearly constrained gaussian processes with boundary conditions. ArXiv, abs/2002.00818, 2021.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. Proc. IEEE, 86:2278–2324, 1998.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. arXiv preprint arXiv:2010.08895, 2020.

Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, and Anima Anandkumar. Physics-informed neural operator for learning partial differential equations. arXiv preprint arXiv:2111.03794, 2021.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. Nat. Mach. Intell., 3:218–229, 2021a.

Lu Lu, Raphael Pestourie, Wenjie Yao, Zhicheng Wang, Francesc Verdugo, and Steven G Johnson. Physics-informed neural networks with hard constraints for inverse design. SIAM Journal on Scientific Computing, 43(6):B1105–B1132, 2021b.

Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online dictionary learning for sparse coding. In Proceedings of the 26th Annual International Conference on Machine Learning, ICML '09, pp. 689–696, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585161. doi: 10.1145/1553374.1553463. URL https://doi.org/10.1145/1553374.1553463.

Lev Semenovich Pontryagin, EF Mishchenko, VG Boltyanskii, and RV Gamkrelidze. The mathematical theory of optimal processes. 1962.

M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics, 378:686–707, 2019. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.10.045. URL https://www.sciencedirect.com/science/article/pii/S0021999118307125.

Youcef Saad and Martin H. Schultz. Gmres: a generalized minimal residual algorithm for solving nonsymmetric linear systems. Siam Journal on Scientific and Statistical Computing, 7:856–869, 1986.

Samuel S. Schoenholz and Ekin D. Cubuk. Jax m.d. a framework for differentiable physics. In Advances in Neural Information Processing Systems, volume 33. Curran Associates, Inc., 2020. URL https://papers.nips.cc/paper/2020/file/83d3d4b6c9579515e1679aca8cbc8033-Paper.pdf.

Joseph Sill. Monotonic networks. In M. Jordan, M. Kearns, and S. Solla (eds.), Advances in Neural Information Processing Systems, volume 10. MIT Press, 1997. URL https://proceedings.neurips.cc/paper/1997/file/83adc9225e4deb67d7ce42d58fe5157c-Paper.pdf.

Kiwon Um, Robert Brand, Yun Raymond Fei, Philipp Holl, and Nils Thuerey. Solver-in-the-loop: Learning from differentiable physics to interact with iterative pde-solvers. Advances in Neural Information Processing Systems, 33:6111–6122, 2020.

Sifan Wang, Hanwen Wang, and Paris Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed deeponets. Science Advances, 7(40), 2021.

Jared Willard, Xiaowei Jia, Shaoming Xu, Michael Steinbach, and Vipin Kumar. Integrating physics-based modeling with machine learning: A survey. arXiv preprint arXiv:2003.04919, 1(1):1–34, 2020.

T. Zhang, Z. Yao, A. Gholami, K. Keutzer, J. Gonzalez, G. Biros, and M. W. Mahoney. ANODEV2: A coupled Neural ODE evolution framework. Technical Report Preprint: arXiv:1906.04596, 2019.

A Appendix

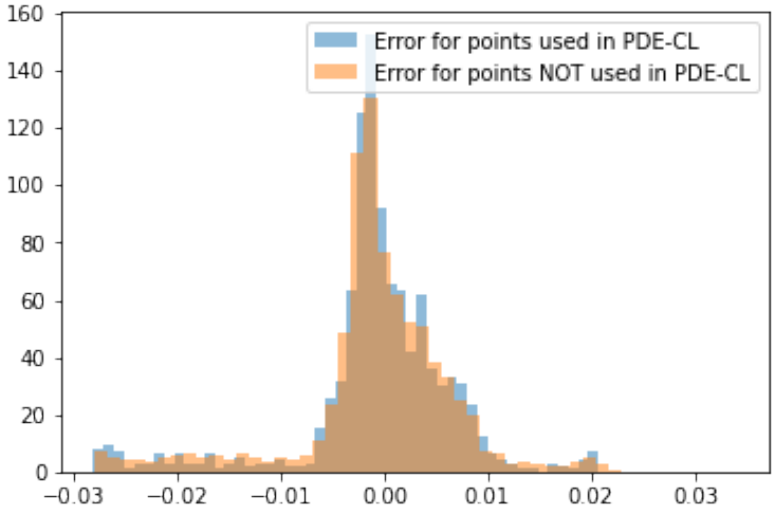


Figure A.1: **Histogram of errors, for points used in PDE-CL vs not used.** We consider a trained model, and perform inference on a random PDE instance. In this plot, we consider the 1D advection setting. The histogram shows that errors for points used in the PDE-CL (1000 points) and are significantly the same as errors for points not used for the PDE-CL (9000 points). This shows that we do not need to fit the PDE-CL on all points of the grid.

A HARD CONSTRAINTS BOUND

To provide an evaluation of how hard the hard constraints are, we conduct an additional study where we look at the error in prediction for points sampled and used to fit the PDE-CL vs. points that were not sampled. With a trained hard-constrained model for 1D convection, we sample a batch of points and create a density plot showing the error as a function of points used for fitting the PDE-CL and points not used for fitting the PDE-CL. The histogram in Figure A.1 shows that the errors are qualitatively the same between points used for fitting the linear combination and those not used. There are 1000 points used for fitting the PDE-CL, and 9000 points not used. Our results show that our model achieves low error, even outside of the points used for fitting the PDE-CL.

B ABLATION STUDIES

B.1 PDE LOSS TERM

We include an ablation study looking at the addition of the PDE residual loss term for the 1D convection example described in Section 10.

We first note that we compute the PDE residuals for our PDE-CL layer, and backpropagate through it when backpropagating through the PDE-CL. Therefore, it does not add computational cost to also include the PDE residual in the loss function. This term depends on the magnitude of the QP solver's tolerance.

The results are as follows: Without PDE residual loss: 38.6% relative error, 4.26 interior PDE loss. With PDE residual loss (included in the paper): 1.77% relative error, 0.399 interior PDE loss

We observe that removing the PDE residual loss hurts the performance of the model. Solving the QP to a non-zero tolerance seems to help the model's training. Our interpretation is that this gives the learned parameters some space to fit to a solution.

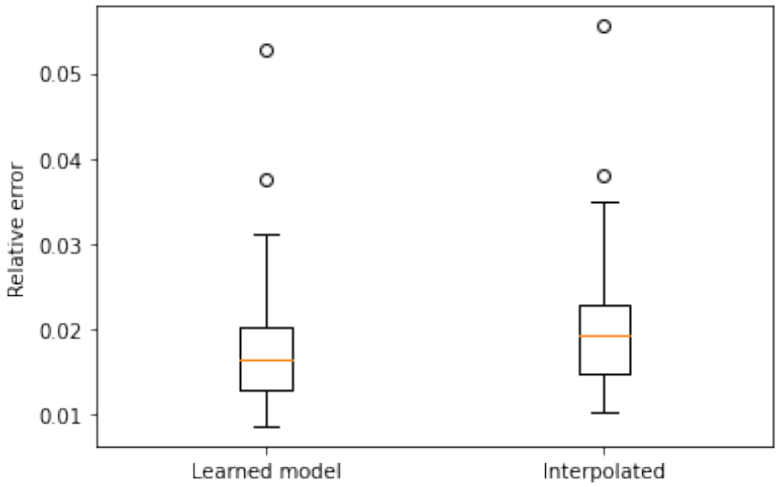


Figure B.1: **Boxplot of errors over the testing dataset, using interpolation from the hard-constrained points vs our model's prediction.**

B.2 Evaluating the quality of the learned basis functions

We add an experiment to evaluate the quality (and the advantage) of our learned basis functions. We want to understand whether our learned functions are useful outside of the points used for the constrained problem.

We start with a trained model. We study a 100x100 grid (10,000 points total). We sample 1000 points from the grid, and solve the corresponding PDE-CL, which gives us a candidate PDE solution. We interpolate from the solution we find on these 1000 points to the 10,000 points using `scipy.optimize`'s cubic interpolation. This is our "baseline."

We compare the above results with the output of our trained hard-constrained model: once the linear combination weights are fit (same as the baseline), we now use our learned basis functions to perform inference over all 10,000 points.

We plot our results over the test dataset in Figure B.1. The figure shows using our model reduces the error vs using interpolation on the hard-constrained points.

C Constrained optimization solver procedure