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 optimization and applications of the implicit function theorem to NN models, we develop a differentiable PDE-constrained layer that can be incorporated into a NN. Inspired by dictionary learning, 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 accurately 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), $\mathcal{F}_\phi(u) = \mathbf{0}$. $\mathcal{F}_\phi$ is a differential operator, and the PDE parameters ϕ and solution u are functions over a domain $\mathcal{X}$. Let Φ be a distribution of PDE-parameter functions ϕ. 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 functions ϕ sampled from Φ, the NN solves the feasibility problem,

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

Training such a model requires solving highly nonlinear feasibility problems in the NN parameter space, 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 1 by considering the relaxation,

$$\min_\theta \mathbb{E}_{\phi\sim\Phi} \|\mathcal{F}_\phi(u_\theta(\phi))\|_2^2. \tag{2}$$

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 (however, note that when solution data is available, we can easily add a data fitting loss to improve training). To solve Equation 1, we design a PDE-constrained layer for NNs that maps 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 differentiable layer, which we call *PDE-Constrained-Layer* or PDE-CL. We make the PDE-CL differentiable using implicit differentiation, thereby allowing us to train our model with gradient-based optimization methods. This layer allows us to find the optimal linear combination of functions in a learned basis, given the PDE constraint.
- At inference time, our model only requires finding the optimal linear combination of the fixed basis functions. When the PDE is linear, this amounts to solving a linear system. We can use computationally efficient, optimized linear solvers for this system at inference time. After using a small number of sampled points to fit this linear combination, we can evaluate the model on a much higher resolution grid.
- Our method controls the trade-off between speed and accuracy through two parameters: the number of functions in the learned basis, and the number of sampled points used to fit the linear combination.
- We provide empirical validation of our method on two problems representing different classes of PDEs. The 1D convection problem is a hyperbolic PDE on a spatiotemporal domain, and the 2D Darcy Flow problem is an elliptic PDE on a stationary (steady-state) spatial 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. Compared to the soft constraint approach, our approach takes fewer iterations to converge to the correct solution, and also requires less training time.
- Our method relies on solving batches of linear systems. It will strongly benefit from future improvements in linear solvers on hardware accelerators (e.g., GPUs, TPUs).

2 Background and Related Work

The layer we design solves a constrained optimization problem corresponding to a PDE constraint. 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 that represents the data accurately. Each datapoint is then represented by combining 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 a parametric over-complete functional basis, which is continuous and differentiable with regard to both its inputs and its parameters.

NNs and structural constraints. Using NNs to solve scientific modeling problems has gained interest in recent years (Willard et al., 2020). NN architectures can also 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). It is sometimes possible to directly parameterize Gaussian processes (Lange-Hegermann, 2018; Jidling et al., 2017) or NNs (Hendriks et al., 2020) to satisfy PDEs, and fit some desired loss function. However, in the PDEs we study, we cannot have a closed-form parameterization for solutions to the PDE. 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. 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 (Krantz & Parks, 2002), 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 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 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 for differentiating through physics simulators (Degrave et al., 2019; Schoenholz & Cubuk, 2020; Um et al., 2020). 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 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. We consider well-posed PDEs, following previous work exploring NNs and PDEs (Raissi et al., 2019; Li et al., 2020; Wang et al., 2021). Let $\mathcal{F}_\phi$ be a functional operator such that for all PDE parameter functions ϕ sampled from Φ, the solution $u(\phi)$ satisfies $\mathcal{F}_\phi(u(\phi)) = \mathbf{0}$. The inputs to our NN vary depending on the domain of interest and the PDE parameters. In the simplest case, the

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

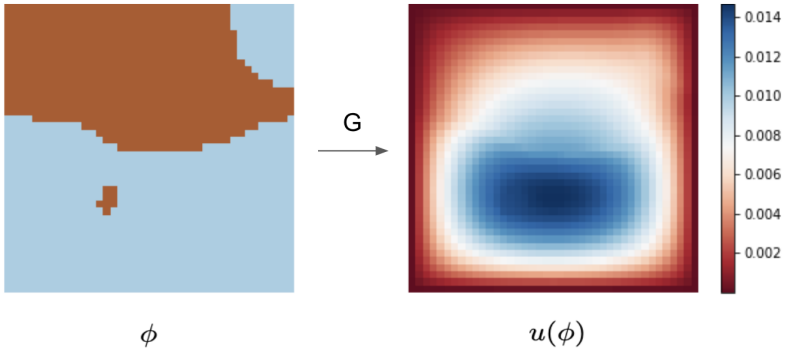


Figure 1: **Mapping PDE parameters ϕ to PDE solutions $u(\phi)$.** The goal of our model is to learn a mapping $G : \phi \mapsto u(\phi)$, without access to solution data. As an example, we study the Darcy Flow PDE, which describes the chemical engineering problem of fluid flow through a porous medium (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 $u(x, y) = 0, \ \forall (x, y) \in \partial(0,1)^2$.

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 $u_\theta(\phi)$, for a given x. We want to learn the mapping,

$$G : \underbrace{\phi}_{\text{PDE parameters}} \mapsto \underbrace{u(\phi)}_{\text{PDE solutions}}. \tag{3}$$

We show an example of such a mapping 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 parameterized by θ, denoted by f_θ. The NN f_θ takes the inputs described in Section 3.1 and outputs a vector in $\mathbb{R}^N$. The output dimension N is the number of functions in our basis, and is a hyperparameter.

The second component of our model, the PDE-constrained layer or PDE-CL, is our main design contribution. We implement a layer that performs a linear combination of the N outputs from the first component, such that the linear combination 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 of f_θ. We now describe the forward and backward pass of the PDE-CL.

Forward pass of the differentiable constrained layer. Our layer is a differentiable root-finder for PDEs; here we focus on linear and affine PDEs, such that solving the PDE constraint can be done by efficiently solving linear systems. As an example, we describe a system based on an affine PDE, where $\mathcal{F}_\phi$ is an affine operator, which depends on ϕ. In our experiments, we study a homogeneous system in Section 4.1 and an inhomogeneous system in Section 4.2. The operator $\mathcal{G}_\phi$ 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,

$$\mathcal{G}_\phi(u + \lambda v) = \mathcal{G}_\phi(u) + \lambda \mathcal{G}_\phi(v). \tag{4}$$

The operator $\mathcal{F}_\phi$ is affine when there exists a function b such that the shifted operator $\mathcal{F}_\phi - b$ is linear. Let $\mathcal{G}_\phi$ be the linear part of the operator: $\mathcal{G}_\phi = \mathcal{F}_\phi - b$. 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 linear equation system

$$\forall j = 1, \ldots, n, \quad \mathcal{G}_\phi\left(\sum_{i=1}^{N} \omega_i f_\theta^i\right)(x_j) = b(x_j) \iff \sum_{i=1}^{N} \omega_i \mathcal{G}_\phi(f_\theta^i)(x_j) = b(x_j). \tag{5}$$

This linear system is a discretization of the PDE $\mathcal{F}_\phi(u_\theta) = 0$; we aim to enforce the PDE at the sampled points $x_1, \ldots, x_n$. The linear system has n constraints and N variables. These are both hyperparameters, that can be chosen to maximize performance. Note that once N is fixed, it cannot be changed. On the other hand, n can be changed at any time.

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 $g(\omega, A, b) = A\omega - b = \mathbf{0}$ 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 $\frac{\partial \omega}{\partial b}$,

$$\forall i, k \in 1, \ldots, N, \quad \forall j \in 1, \ldots, n, \quad 0 = \frac{\partial \omega_i}{\partial A_{jk}} = \left(\omega_k + A_j^\top \frac{\partial \omega}{\partial A_{jk}}\right) \mathbb{1}_{i=j}, \tag{6}$$

$$\forall i \in 1, \ldots, N, \quad \forall j \in 1, \ldots, n, \quad 0 = \frac{\partial g_i}{\partial b_j} = A_i^\top \frac{\partial \omega}{\partial b_j} - \mathbb{1}_{i=j}, \tag{7}$$

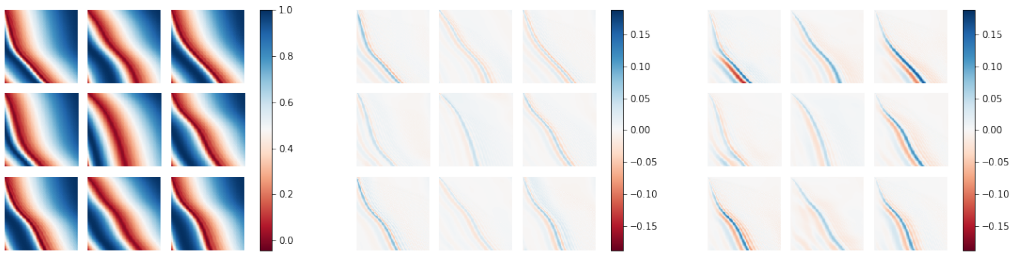
where $\mathbb{1}_{i=j}$ is 1 when $i = j$ and 0 otherwise. Given the size and conditioning of our problems, we cannot directly solve the linear system. Thus, we use an indirect solver (such as conjugate gradient (Hestenes & Stiefel, 1952) or GMRES (Saad & Schultz, 1986)) for both the forward system $A\omega = b$ and the backward system given by Equation 6 and Equation 7. 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 D.1 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 C.

Loss function. Our goal is to obtain a NN parameterized function which verifies the PDE over the whole domain. The PDE-CL only guarantees that we verify the PDE over the sampled points. In the case where $N > n$, the residual over the sampled points $x_1, \ldots, x_n$ is zero, up to numerical error of the linear solver used in our layer. It is preferable to not use this residual for training the NN as it may not be meaningful, and an artifact of the chosen linear solver and tolerances. Instead, we sample new points $x'_1, \ldots, x'_{n'}$ and build the corresponding linear system A', b'. Our loss value is $\|A'\omega - b'\|_2^2$, where ω comes from the PDE-CL and depends on A and b, not A', b'. We compute gradients of this loss function using the Jacobian described above. Another possibility is to use $n > N$. In this case, the residual $\|A\omega - b\|_2^2$ will be non-zero, and while the "hard constraints" will not be satisfied during training, we can minimize this residual loss directly. Let $\mathcal{U}(\mathcal{X})$ denote the uniform distribution over our (bounded) domain $\mathcal{X}$. Formally, our goal is to solve the bilevel optimization problem

$$\begin{aligned} &\min_\theta \ \mathbb{E}_{\phi \sim \Phi} \mathbb{E}_{(x_1, \ldots, x_n), (x'_1, \ldots, x'_{n'}) \sim \mathcal{U}(\mathcal{X})} \|A'(\phi, x'_1, \ldots, x'_{n'}; \theta)\omega(\phi, x_1, \ldots, x_n; \theta) - b'(\phi, x'_1, \ldots, x'_{n'}; \theta)\|_2^2 \\ &\text{s.t. } \omega = \arg\min_w \|A(\phi, x_1, \ldots, x_n; \theta)w - b(\phi; x_1, \ldots, x_n; \theta)\|_2^2. \end{aligned} \tag{8}$$

We approximate this problem by replacing the expectations by sums over finite samples. The matrices A, A', and vectors b, b' are built by applying the differential operator $\mathcal{F}_\phi$ to each function in our basis f_θ^i, using the sampled gridpoints.

Inference procedure. At inference, when given a new PDE parameter test point ϕ, the weights θ are fixed as our function basis is trained. Suppose that we want the values of u_θ over the (new) points $x_1^{\text{test}}, \ldots, x_{n^{\text{test}}}^{\text{test}}$. If $n^{\text{test}} < N$, we can fit ω in the PDE-CL using all of the test points. This guarantees that our model satisfies the PDE on all test points: the linear system in the PDE-CL



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

Figure 2: **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 $\mathbf{1.32\%} \pm \mathbf{0.02\%}$ relative error and $\mathbf{9.84} \pm \mathbf{2.15}$ PDE residual test loss. Our model achieves $\mathbf{49\%}$ less error than the soft-constrained model. In contrast, the soft-constrained model only reaches $\mathbf{2.59\%} \pm \mathbf{0.15\%}$ relative error and $\mathbf{774} \pm \mathbf{1.2}$ PDE residual 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.

is underdetermined. In practice, n^{test} is often larger than N. In this case, we can sample $J \subset \{1, \ldots, n^{\text{test}}\}$, $|J| < N$, and fit the PDE-CL over these points. Over the subset $\{x_j^{\text{test}}, j \in J\}$, the PDE will be satisfied. Over the other points, the residual may be non-zero. Another option is to fit the PDE-CL using all the points $x_1^{\text{test}}, \ldots, x_{n^{\text{test}}}^{\text{test}}$, in which case the residual may be non-zero for all points, but is minimized on average over the sampled points by our PDE-CL. **Our method controls the trade-off between speed and accuracy**: choosing a larger N results in larger linear systems, but also larger sets of points on which the PDE is enforced. A smaller N allows for faster linear system solves. Once ω is fit, we can query u_θ for any point in the domain. In practice, we can choose $|J|$ to be much smaller than n^{test}; here, the linear system we need to solve is much smaller than the linear system required by a numerical solver.

4 Experimental Results and Implementation

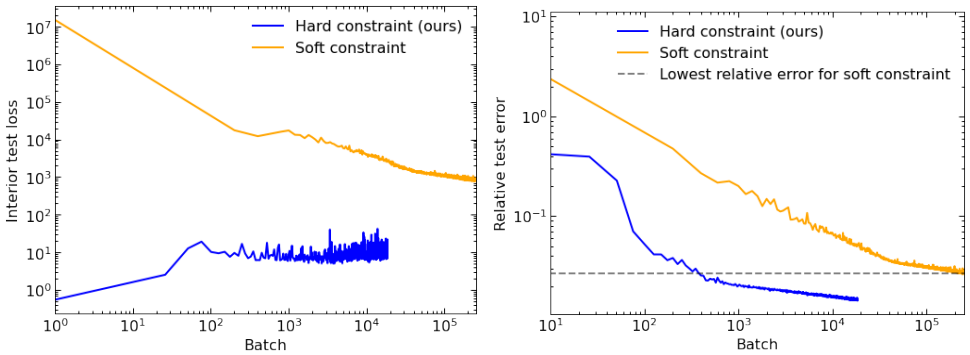
We test the performance of our model on two different scientific problems: 1D convection (Section 4.1) and 2D Darcy Flow (Section 4.2). In each case, the model is trained without access to any solution data in the interior solution domain. The training set contains 1000 PDE parameters ϕ. The model is 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]

4.1 1D Convection

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

$$\begin{aligned} &\frac{\partial u(x,t)}{\partial t} + \beta(x)\frac{\partial u(x,t)}{\partial x} = 0, && x \in (0,1), t \in (0,1), \\ &h(x) = \sin(\pi x), && x \in (0,1) \\ &g(t) \ = \sin\left(\frac{\pi}{2}t\right), && t \in (0,1). \end{aligned} \tag{9}$$

[2]We used a single Titan RTX GPU for each run in our experiments.



(a) PDE residual loss on test set

(b) Relative error on test set

Figure 3: **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. We observe that the NN starts by fitting the IC/BC regression loss during training, which explains why the PDE residual loss seems to go up initially. (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 grey, dashed line shows that the hard-constrained model achieves the same relative error as the soft-constrained model in over 100x fewer iterations, and ultimately achieves lower relative error. Wall-time comparison figures are given in Appendix A.

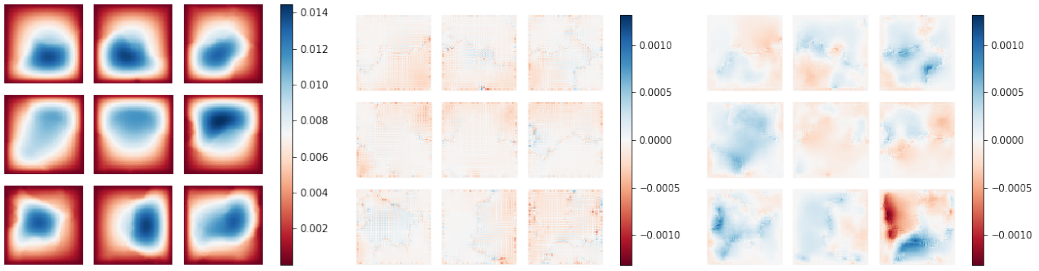
Here, $h(x)$ is the initial condition (at $t = 0$), $g(t)$ is the boundary condition (at $x = 0$), and $\beta(x)$ represents the variable coefficients (denoted by ϕ in Section 3). Given a set of variable coefficients, $\beta(x)$, and spatiotemporal points (x_i, t_i), the goal is to predict the correct 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. 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 for a fair comparison. We provide more details on our setup and experiments in Appendix A.

Results. We plot example heatmaps from the test set in Figure 2. We compare how close our hard-constrained model is to the target solution (Figure 2b), and similarly for the soft-constrained baseline model (Figure 2c). 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 3. In Figure 3a, 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 3b, we track 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 $\mathbf{1.32\% \pm 0.02\%}$ relative error and $\mathbf{9.84 \pm 2.15}$ PDE residual test loss, versus $\mathbf{2.59\% \pm 0.15\%}$ and $\mathbf{774 \pm 1.2}$ for the soft-constrained baseline model. On the relative error metric, our model achieves a $\mathbf{49\%}$ **improvement** over the soft-constrained model. The standard deviation of our model for the relative error metric over the test dataset is also small. Our model trains faster than the soft-constrained method, due to much higher gains in accuracy per batch, even though each batch is slower (see Figure A.1).

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

Figure 4: **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 $\mathbf{1.82\% \pm 0.04\%}$ relative error and $\mathbf{0.0457 \pm 0.0021}$ interior domain test loss. In contrast, the soft-constrained model only reaches $\mathbf{3.86\% \pm 0.3\%}$ relative error and $\mathbf{1.1355 \pm 0.0433}$ interior domain test loss. Our model improves the relative test error by $\mathbf{71\%}$. While the heatmaps show a subset of the full test set, the standard deviation across the test set for our model is very low, as shown by the box plot in Appendix B.

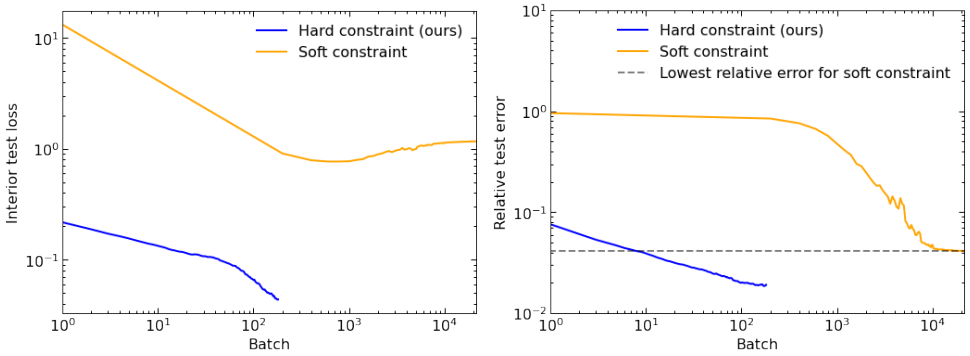
a general variable) is $\nu \in L^\infty((0,1)^2; \mathbb{R}_+)$, a diffusion coefficient. The problem is formulated as follows:

$$\begin{aligned} -\nabla \cdot (\nu(x)\nabla u(x)) &= f(x), \qquad && \forall x \in (0,1)^2, \\ u(x) &= 0, && \forall x \in \partial(0,1)^2. \end{aligned} \tag{10}$$

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 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). 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 the only term in the loss function is the PDE residual. We use a constant forcing function f equal to 1. We provide more details on our setup and implementation in Appendix B.

Results. We plot example heatmaps from the test set in Figure 4. We compare visually how close our hard-constrained model is to the target solution (Figure 4b), and how close the soft-constrained baseline model is to the target solution (Figure 4c). 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 5. In Figure 5a, 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 5b, 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.



(a) PDE residual loss on test set

(b) Relative error on test set

Figure 5: **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.

On the test set, our model achieves $\mathbf{1.82\%} \pm \mathbf{0.04\%}$ relative error, versus $\mathbf{3.86\%} \pm \mathbf{0.3\%}$ for the soft-constrained baseline model. Our model also achieves $\mathbf{0.0457} \pm \mathbf{0.0021}$ for the PDE residual test loss, versus $\mathbf{1.1355} \pm \mathbf{0.0433}$. Our model has almost two orders of magnitude lower PDE residual test loss, and it has significantly lower standard deviation. 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. This indicates that the results are consistent across test samples.

5 Conclusions

We have considered the problem of mapping PDEs to their corresponding solutions, in particular in the unsupervised setting, where no solution data is available on the interior of the domain during training. For this situation, we have developed a method to enforce hard PDE constraints, when training NNs, by designing a differentiable PDE-constrained layer (PDE-CL). We can add our layer to any NN architecture to enforce PDE constraints accurately, and then train the whole system end-to-end. Due to the procedure by which we sample our points and pick the number of basis functions, our method controls a trade-off between speed and accuracy. We evaluate our proposed method on two problems representing different physical settings: a 1D convection problem, which describes transport phenomena; and a 2D Darcy Flow problem, which describes fluid flow. 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 achieves 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. However, future work could consider extending our method to nonlinear PDEs. This would require developing an approach to implicitly differentiate through solvers for nonlinearly constrained optimization problems. Our method also opens exciting avenues for combining numerical methods and NNs more generally. In particular, our method would immediately benefit from improvements from two areas: better neural architectures for representing functions; and better linear solvers on GPUs. Localized neural architectures such as MoE (Shazeer et al., 2017; Wang et al., 2022) would allow making the linear system in our layer sparse. Leveraging this structure when solving the linear system in PDE-CL would improve our method, in terms of both speed and accuracy.

REFERENCES

Akshay Agrawal, Brandon Amos, Shane T. Barratt, Stephen P. Boyd, Steven Diamond, and J. Zico Kolter. Differentiable convex optimization layers. In NeurIPS, 2019. 3

Brandon Amos and J. Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In ICML, 2017. 3

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. 3

Shane T. Barratt. On the differentiability of the solution to convex optimization problems. arXiv: Optimization and Control, 2018. 3

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. 3, 5

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. 5

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. 2

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Kristjanson Duvenaud. Neural ordinary differential equations. ArXiv, abs/1806.07366, 2018. 3

Taco Cohen and Max Welling. Group equivariant convolutional networks. In ICML, 2016. 2

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. 4

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. 3

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. 3

Priya L Donti, David Rolnick, and J Zico Kolter. Dc3: A learning method for optimization with hard constraints. arXiv preprint arXiv:2104.12225, 2021. 3

Chris Edwards. Neural networks learn to speed up simulations. Communications of the ACM, 65 (5):27–29, 2022. 1

Johannes N. Hendriks, Carl Jidling, Adrian G. Wills, and Thomas Bo Schön. Linearly constrained neural networks. ArXiv, abs/2002.01600, 2020. 3

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. 5

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural Computation, 9:1735–1780, 1997. 2

Carl Jidling, Niklas Wahlstrom, Adrian G. Wills, and Thomas Bo Schön. Linearly constrained gaussian processes. In NIPS, 2017. 3

Steven George Krantz and Harold R Parks. The implicit function theorem: history, theory, and applications. Springer Science & Business Media, 2002. 3

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. 1

Markus Lange-Hegermann. Algorithmic linearly constrained gaussian processes. In NeurIPS, 2018. 3

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. Proc. IEEE, 86:2278–2324, 1998. 2

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. 1, 3, 8

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. 1, 15

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. 1, 3

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. 3

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. 2

Lev Semenovich Pontryagin, EF Mishchenko, VG Boltyanskii, and RV Gamkrelidze. The mathematical theory of optimal processes. 1962. 3

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. 1, 3

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. 5, 14

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. 3

Noam M. Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. ArXiv, abs/1701.06538, 2017. 9

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. 3

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. 3

Peihao Wang, Zhiwen Fan, Tianlong Chen, and Zhangyang Wang. Neural implicit dictionary learning via mixture-of-expert training. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), Proceedings of the 39th International Conference on Machine Learning, volume 162 of Proceedings of Machine Learning Research, pp. 22613–22624. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/wang22d.html. 9

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. 1, 3, 7

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. 2

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. 3