DOI: 10.14464/gammas.v7i1.813 GAMMAS 2025, Vol. 7, No. 1, p. 31–45

International Association
of Applied Mathematics and Mechanics
– Archive for Students –

Experiences with Physics-Informed Neural Networks for Optimal Control Problems

Johannes Wagner[a,⋆], Evelyn Herberg[b], Roland Herzog[b]

[a] Professur für Numerik partieller Differentialgleichungen, Goethe University Frankfurt, 60054 Frankfurt am Main, Germany
[b] Interdisciplinary Center for Scientific Computing, Heidelberg University, 69120 Heidelberg, Germany

received 30.12.2024, accepted 04.04.2025, published 22.05.2025

⋆ corresponding author: jwagner@math.uni-frankfurt.de

Abstract: *Neural networks can be used to parametrize solutions of partial differential equations (PDEs). Specifically, physics-informed neural networks (PINNs) have gained popularity since they do not require numerically simulated or experimental data for training. It is straightforward to use PINNs to represent not only a single PDE solution but parameter dependent families of solutions. On the other hand, PINNs are reported to experience limited accuracy as well as convergence issues during training. In this work, we give an account of our experience in using PINNs to solve a simple family of PDE-constrained optimal control problems dependent on the control cost parameter. We employ a straightforward approach by minimizing the pointwise residuals of the optimality system at random collocation points. In our experiments, we observe severe convergence problems during training, even in the fixed parameter setting, and attribute them to a bias in the multi-objective PINN training. We report on two techniques to overcome this issue: dynamic loss weights and hard-coding certain parts of the optimality system. The combination of these measures enables the PINN model to learn solutions to reasonable accuracy for individual parameter values within a range of several orders of magnitude. In an attempt to learn the entire parametric family of solutions, we increase the network size yet are unable to consistently achieve the same accuracy as in the fixed parameter setting. This indicates that loss-weighting algorithms cannot completely overcome training bias.*

Keywords: optimal control, physics-informed neural networks, adaptive loss weighting

1 Introduction

The rise of machine learning has produced a dynamic field of research employing deep neural networks to solve complex problems. Recently, it has been proposed in [29] that their differentiable structure makes them suitable candidates to solve problems modeled by partial differential equations (PDEs). In particular, it has been discovered that Algorithmic Differentiation (AD) [12], the technique that enabled training deep neural networks with gradient based methods [30], can be used to efficiently compute partial derivatives of neural network outputs w.r.t. the network's parameters and the network's input. This procedure can be used to inform the training process of so-called physics-informed neural networks (PINNs) as follows. Consider a PDE of the

general form

$$\begin{aligned} \mathscr{D}[z] &= 0 \quad \text{in } \Omega, \\ \mathscr{B}[z] &= 0 \quad \text{on } \partial\Omega, \end{aligned} \tag{1.1}$$

where $\mathscr{D}$ represents a single or a finite collection of linear or nonlinear differential operators acting on the unknown solution z defined on a bounded domain $\Omega \subseteq \mathbb{R}^D$. Moreover, $\mathscr{B}$ represents associated boundary conditions.

Following the PINN approach, we approximate the PDE solution's point evaluation map $x \mapsto z(x)$ using a neural network z_θ with parameters θ and input x. Since we focus on stationary PDEs, x denotes any point in $\overline{\Omega}$ but an extension to time-dependent PDEs is straightforward. The spatial derivatives of $x \mapsto z_\theta(x)$ appearing in $\mathscr{D}$ and $\mathscr{B}$ can be conveniently evaluated using AD. First, we sample interior collocation points $x_i^I \in \Omega$, $i = 1, \dots, m_I$ and boundary collocation points $x_i^B \in \partial\Omega$, $i = 1, \dots, m_B$. Next, we define the loss function for PINN training of (1.1) in case of a single scalar-valued PDE and boundary condition as

$$\mathscr{L}(\theta) = \frac{1}{2}\frac{1}{m_I}\sum_{i=1}^{m_I}\left[\mathscr{D}[z_\theta](x_i^I)\right]^2 + \frac{1}{2}\frac{1}{m_B}\sum_{i=1}^{m_B}\left[\mathscr{B}[z_\theta](x_i^B)\right]^2. \tag{1.2}$$

During PINN training, this loss function is approximately minimized using a gradient descent scheme or one of its stochastic variants. For reasonably low loss values we can expect that $z_\theta(x)$ is a good approximation of the solution at the chosen collocation point sets and, under continuity assumptions, also elsewhere within $\overline{\Omega}$. PINNs can be easily implemented with AD-enabled deep learning libraries, and their great flexibility makes them applicable to a wide range of problems.

Recent research, however, suggests that PINNs are unlikely to provide runtime or accuracy improvements over traditional finite element based solvers whenever these are available [13]. On the other hand, an additional advantage of PINNs is that they can approximate the solution of a family of parameter-dependent PDEs in a one-shot approach, where the parameter is treated as an additional network input. In theory, a point evaluation $z_\theta(x)$ of the solution for a fixed parameter value can then be obtained in a single forward pass.

In this work, we study the capability of PINNs to approximate solutions to optimal control problems for PDEs. Specifically, we consider a simple model problem, i. e., distributed optimal control of the Poisson equation:

$$\begin{aligned} &\text{Minimize} \quad J(y,u) := \frac{1}{2}\|y - y_d\|^2_{L^2(\Omega)} + \frac{\gamma}{2}\|u\|^2_{L^2(\Omega)} \\ &\text{subject to} \quad \begin{cases} -\Delta y = u & \text{in } \Omega, \\ y = 0 & \text{on } \partial\Omega \end{cases} \end{aligned} \tag{1.3}$$

on a bounded Lipschitz domain $\Omega \subseteq \mathbb{R}^2$. We refer to the functions y and u as state and control variables, respectively. The first term in J can be thought of as a fidelity term, describing how well the so-called "desired state" y_d is met by y. Large values of the control u are penalized by the second term, weighted by the regularization or control cost parameter $\gamma > 0$. By introducing the so-called "adjoint variable" p, solutions of (1.3) can be classified by a system of PDEs. Namely, a control $u \in L^2(\Omega)$ together with the corresponding state and adjoint variables $y, p \in H_0^1(\Omega)$ solve (1.3) if and only if they satisfy the following system of equations:

$$\begin{cases} -\Delta y = u & \text{in } \Omega, \\ y = 0 & \text{on } \partial\Omega, \end{cases} \quad \text{(state eq.)} \tag{1.4a}$$

$$\gamma u - p = 0 \quad \text{in } \Omega, \quad \text{(gradient eq.)} \tag{1.4b}$$

$$\begin{cases} -\Delta p = -(y - y_d) & \text{in } \Omega, \\ p = 0 & \text{on } \partial\Omega. \end{cases} \quad \text{(adjoint eq.)} \tag{1.4c}$$

For a derivation of these well-known necessary and sufficient optimality conditions we refer the reader to, for instance, [32, Theorem 2.25].

Our goal is to employ a PINN to solve the above system of equations, first for fixed, then for variable control cost parameter γ. We report on difficulties and limited success we experience while studying this problem and provide some remedies for commonly encountered caveats in PINN training.

Our presentation is organized as follows. In section 2, we present a common failure mode of PINN training and discuss mitigation techniques that we have used to ensure convergence during training of our models for the fixed parameter setting, i. e., solving (1.3) for a fixed value of γ, using the ADAM algorithm [20]. Once we establish a setup such convergence is achieved for a range of control costs, we introduce γ as an additional input to the network. We report on numerical results in section 3 and conclude with a discussion in section 4.

2 Analyzing Failure Modes of PINN Training

Following the standard PINN approach, we use a neural network with outputs y_θ, u_θ and p_θ to model state, control and adjoint variables. Next, we use (1.4) to formulate a loss-function according to (1.2), i. e., a loss

consisting of the sum of the following five terms:

$$\mathscr{L}_1(\theta) = \frac{1}{2}\frac{1}{m_I}\sum_{i=1}^{m_I}\left[-\Delta y_\theta(x_i^I) - u_\theta(x_i^I)\right]^2 \tag{2.1a}$$

$$\mathscr{L}_2(\theta) = \frac{1}{2}\frac{1}{m_B}\sum_{i=1}^{m_B}\left[y(x_i^B)\right]^2 \tag{2.1b}$$

$$\mathscr{L}_3(\theta) = \frac{1}{2}\frac{1}{m_I}\sum_{i=1}^{m_I}\left[\gamma\, u_\theta(x_i^I) - p(x_i^I)\right]^2 \tag{2.1c}$$

$$\mathscr{L}_4(\theta) = \frac{1}{2}\frac{1}{m_I}\sum_{i=1}^{m_I}\left[-\Delta p_\theta(x_i^I) + y_\theta(x_i^I) - y_d(x_i^I)\right]^2 \tag{2.1d}$$

$$\mathscr{L}_5(\theta) = \frac{1}{2}\frac{1}{m_B}\sum_{i=1}^{m_B}\left[p(x_i^B)\right]^2. \tag{2.1e}$$

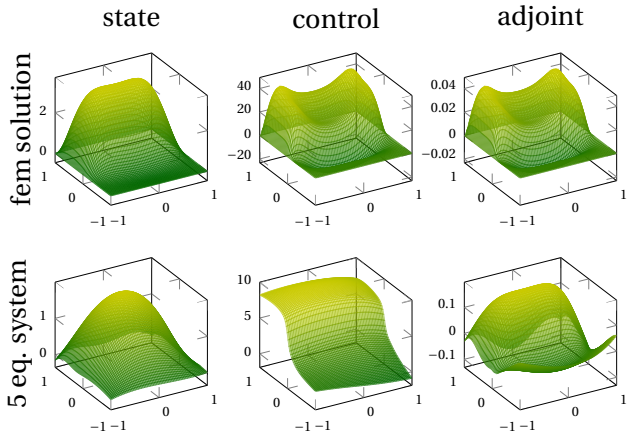

Figure 2.1 – Comparison of a finite element baseline solution and standard PINN solution of (1.4) with $\gamma = 10^{-3}$. The PINN does not converge to a good solution. Detailed setup in table 3.1, ID: 1a.

When training PINNs using this simple setup, we were initially surprised to find the quality of our solutions heavily depending on γ: the solution can be well approximated by a PINN, whenever γ is close to one, however the quality of approximation worsens dramatically when $\gamma = 10^{-3}$; see fig. 2.1. These convergence failures persisted when increasing training time, network size and manually tuning optimization parameters like the learning rate. Notably, other researchers also report convergence problems when training PINNS using this five-term loss function [6]. Our observations fall in line with a growing number of publications on PINN training reporting difficulties to converge to good solutions [3, 9, 33, 38]. Indeed, since we aim to minimize the residuals of all equations simultaneously, PINN training can be seen as a multi-objective optimization problem. Summing up all residuals to obtain a naive scalarization of the multi-dimensional loss function has some major theoretical drawbacks:

(1) From a mathematical perspective, residuals should be measured in suitable dual Sobolev-norms on function spaces supported on Ω and $\partial\Omega$. These norms usually contain derivatives of the functions under consideration. The (squared) pointwise residuals lack this information.

(2) When physical phenomena are modeled, as is the case for PDE and PDE-constrained optimal control problems, the residual expressions carry physical units which should match before being added up in a single loss function. By default, this is not the case, e. g., when adding $\mathscr{L}_1, \ldots, \mathscr{L}_5$.

From a more practical point of view, we implicitly assume that all residual terms equally contribute to the update of the network parameters. Otherwise, training via first order methods is biased towards minimizing a particular residual. This is a well-known challenge for multi-objective optimization with competing terms; see, e. g., [8, Chapter 1]. In case of our optimal control problem (1.3), we search a state y approximating the desired state y_d, while imposing homogeneous Dirichlet conditions on the boundary. This introduces a competition between objectives, e. g., whenever y_d has nonzero boundary values. Competing objectives can be decoupled "in space", by using multiple gradient descent [1], or "in time", by alternating the training of state and adjoint variables [39], or by enforcing chronological training in time-dependent PDEs [35].

In this work, we instead follow a substantial body of research developing techniques to:

(1) rescale objectives with suitable weights λ_k, so they contribute evenly to the network parameter updates.

(2) eliminate objective terms by designing the network architecture such that some of the residuals are inherently satisfied. In the present study, we achieve this by exactly imposing the Dirichlet boundary conditions of (1.4a) and (1.4c), as well as the gradient equation (1.4b); see fig. 2.2.

We emphasize, that only a combination of the two approaches yields stable convergence for a significant range of γ values (fig. 3.2).

2.1 Adaptive Loss Weighting Strategies

A straightforward way to reduce bias during training is to introduce weights $\lambda_k > 0$ into the loss function, i. e., for a system consisting of a total of N_I interior and N_B boundary equations:

$$\mathscr{L}(\theta) = \sum_{k=1}^{N} \lambda_k \mathscr{L}_k(\theta) \tag{2.2}$$

with $N := N_I + N_B$. This allows to For specific problems, theoretically optimal weights can be estimated analytically [33] or empirically, by selecting those that produce the lowest value of the optimal control objective J in (1.3) [26]. In general, since training is a dynamic optimization process, there is little hope to find fixed weight values that ensure convergence over the whole course of training. Instead, a large body of literature proposes dynamic weighting strategies that update weights during optimization, for instance [1, 2, 16, 23, 33, 36, 37]. In the following we discuss three such strategies and later compare them for problem (1.3).

Inverse Dirichlet Weighting

When training is performed using (a variant of) gradient descent, the influence of the objective $\mathscr{L}_k$ on a network parameter θ_ℓ in each training step is proportional to $\frac{\partial \mathscr{L}_k}{\partial \theta_\ell}$. The influence that $\mathscr{L}_k$ has on all weights, in a particular training step, can be visualized using a histogram of $\nabla_\theta \mathscr{L}_k$; see fig. 3.4. Treating $\nabla_\theta \mathscr{L}_k$ as a distribution of partial derivatives, we expect distributions with large variance to have more influence on the network parameters [11, 36]. Having made this observation, the straightforward approach is thus to introduce weights λ_k which rescale distributions to have similar variances. This is in essence what the authors of [23] have termed inverse Dirichlet weighting. The particular update rule they propose is

$$\widehat{\lambda}_k(t) = \sqrt{\frac{\max_{i=1,\dots,N} \mathrm{Var}[\nabla_\theta \mathscr{L}_i(\theta)]}{\mathrm{Var}[\nabla_\theta \mathscr{L}_k(\theta)] + \varepsilon}}, \tag{2.3a}$$

$$\lambda_k(t) = \alpha \lambda_k(t-1) + (1-\alpha)\, \widehat{\lambda}_k(t), \tag{2.3b}$$

where $\alpha \in [0,1]$ is an exponential smoothing parameter and $\theta = \theta(t)$ are the network weights in iteration $t \in \mathbb{N}$. Here we omit the argument for readability. We also added a small positive constant $\varepsilon = 10^{-7}$, chosen by hand, in the denominator of (2.3b), to avoid division by zero.

The authors of [23] argue that this weighting procedure leads to the variances of each objective being approximately equal, i. e.,

$$\mathrm{Var}[\lambda_k \nabla_\theta \mathscr{L}_k(\theta)] \approx \max_{i=1,\dots,N} \mathrm{Var}[\nabla_\theta \mathscr{L}_i(\theta)] \tag{2.4}$$

for $k = 1, \dots, N$. Indeed, without smoothing and numerical correction, i. e., $\alpha = \varepsilon = 0$, we see

$$\begin{aligned}
&\mathrm{Var}[\lambda_k \nabla_\theta \mathscr{L}_k(\theta)] \\
&\quad = \mathrm{Var}[\widehat{\lambda}_k \nabla_\theta \mathscr{L}_k(\theta)] = \widehat{\lambda}_k^2 \mathrm{Var}[\nabla_\theta \mathscr{L}_k(\theta)] \\
&\quad = \left(\frac{\max_{i=1,\dots,N} \mathrm{Var}[\nabla_\theta \mathscr{L}_i(\theta)]}{\mathrm{Var}[\nabla_\theta \mathscr{L}_k(\theta)]} \right) \mathrm{Var}[\nabla_\theta \mathscr{L}_k(\theta)] \\
&\quad = \max_{i=1,\dots,N} \mathrm{Var}[\nabla_\theta \mathscr{L}_i(\theta)],
\end{aligned}$$

where the first equality follows by standard calculus rules for the variance. When $\alpha \in [0,1]$ and $\varepsilon > 0$ is small, we expect this equation to be true in an approximate sense, i. e., (2.4) to hold.

Relative Loss Balancing with Random Lookback

The next adaptive weighting strategy we implement is called Relative Loss Balancing with Random Lookback (ReLoBraLo) and has been proposed in [1]. It aims to combine the advantages of three existing approaches [2, 16, 36] and the authors report performance improvements on model problems. ReLoBraLo operates solely on loss statistics. Weight updates are exponentially smoothed and updates are based on the relative progress of each objective. In particular, the authors of [1] propose the update rule

$$\lambda_k^{\mathrm{bal}}(t,t') = N \cdot \frac{\exp\left(\frac{\mathscr{L}_k(t)}{\tau \mathscr{L}_k(t') + \varepsilon} - \mu(t,t') \right)}{\sum_{i=1}^m \exp\left(\frac{\mathscr{L}_i(t)}{\tau \mathscr{L}_i(t') + \varepsilon} - \mu(t,t') \right)},$$

$$\lambda_k^{\mathrm{hist}}(t) = \rho \lambda_k(t-1) + (1-\rho)\, \lambda_k^{\mathrm{bal}}(t,0),$$

$$\lambda_k(t) = \alpha \lambda_k^{\mathrm{hist}} + (1-\alpha)\, \lambda_k^{\mathrm{bal}}(t,t-1),$$

for $k = 1, \dots, N$, where N denotes the number of additive objective terms. They add a lookback with probability represented by a Bernoulli distributed random variable ρ, where $\mathbb{E}[\rho]$ is close to 1. The parameter α is the exponential smoothing rate, like in inverse Dirichlet weighting, and τ is used to control the homogenizing effect of the softmax function: large values of τ lead to uniform loss weights, while with $\tau \to 0$, the softmax approaches an argmax function. To avoid division by zero we have added a small number to the denominator, i. e., $\varepsilon = 10^{-15}$. The quantity

$$\mu(t,t') := \max_{k=1,\dots,N} \frac{\mathscr{L}_k(t)}{\mathscr{L}_k(t')}$$

is subtracted to prevent numerical overflow. Note that it is not present in the original formulation but we have added it in our implementation, inspired by [27].

NTK Loss Weighting

Some recent progress in the analysis of neural network training dynamics has been made by [17] using a concept called the neural tangent kernel (NTK). The authors of [37] have extended these results to PINN training, use them to derive a measure of average convergence rate and propose loss weights aimed to homogenize this convergence rate over all objectives. Training a neural network z_θ using a loss function $\mathscr{L}(\theta)$ with gradient descent can be understood as a forward Euler approximation of the time-dependent gradient flow system [14].

$$\frac{\mathrm{d}\theta(t)}{\mathrm{d}t} = -\nabla_\theta \mathscr{L}(\theta(t)). \tag{2.5}$$

Here, by a slight abuse of notation, we denoted the continuous "time" variable also with $t \in \mathbb{R}$.

In case of PINN training we have a general system

$$\begin{aligned} \mathscr{D}_k[z] &= 0 \quad \text{in } \Omega \qquad \text{for } k = 1,\dots,N_I, \\ \mathscr{B}_k[z] &= 0 \quad \text{on } \partial\Omega \quad \text{for } k = 1,\dots,N_B \end{aligned} \tag{2.6}$$

with N_I scalar PDE components $\mathscr{D}_k$ and N_B boundary conditions $\mathscr{B}_k$. Similar to the previous discussions we sample collocation points $x_i^I \in \Omega$, $i = 1,\dots,m_I$ and boundary points $x_i^B \in \partial\Omega$, $i = 1,\dots,m_B$ and formulate the general squared loss function

$$\begin{aligned} \mathscr{L}(\theta) = &\frac{1}{2}\sum_{k=1}^{N_I}\sum_{i=1}^{m_I}\big[\mathscr{D}_k[z_\theta](x_i^I)\big]^2 \\ &+ \frac{1}{2}\sum_{k=1}^{N_B}\sum_{i=1}^{m_B}\big[\mathscr{B}_k[z_\theta](x_i^B)\big]^2. \end{aligned} \tag{2.7}$$

Note that unlike before, in this method it is customary not to divide by m_I and m_B. We rewrite the loss function (2.7) using the vectorized notation

$$\mathbf{D}_k(\theta) := \begin{pmatrix} \mathscr{D}_k[z_\theta(x_1^I)] \\ \vdots \\ \mathscr{D}_k[z_\theta(x_{m_I}^I)] \end{pmatrix} \in \mathbb{R}^{m_I},$$

$$\mathbf{B}_k(\theta) := \begin{pmatrix} \mathscr{B}_k[z_\theta(x_1^B)] \\ \vdots \\ \mathscr{B}_k[z_\theta(x_{m_B}^B)] \end{pmatrix} \in \mathbb{R}^{m_B},$$

as

$$\mathscr{L}(\theta) = \frac{1}{2}\sum_{k=1}^{N_I}\|\mathbf{D}_k(\theta)\|_2^2 + \frac{1}{2}\sum_{k=1}^{N_B}\|\mathbf{B}_k(\theta)\|_2^2, \tag{2.8}$$

where the $\|\cdot\|_2$ is the Euclidean norm on $\mathbb{R}^{m_I}$ and $\mathbb{R}^{m_B}$ respectively.

The following lemma analyzes the evolution of the residuals associated with each of the differential and boundary operators under the gradient flow (2.5).

Lemma 2.1. *Consider the gradient flow (2.5) for the loss function (2.8). Then there exists a symmetric and positive semi-definite matrix $K(t) \in \mathbb{R}^{n\times n}$ of dimension $n := N_I m_I + N_B m_B$, called the neural tangent kernel, such that the evolution of the residuals is described by the following system of ordinary differential equations (ODEs):*

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix} \mathbf{D}_1 \\ \vdots \\ \mathbf{D}_{N_I} \\ \mathbf{B}_1 \\ \vdots \\ \mathbf{B}_{N_B} \end{pmatrix} = -K(t)\begin{pmatrix} \mathbf{D}_1 \\ \vdots \\ \mathbf{D}_{N_I} \\ \mathbf{B}_1 \\ \vdots \\ \mathbf{B}_{N_B} \end{pmatrix} \tag{2.9a}$$

with

$$K(t) = \begin{pmatrix} \mathscr{J}_\theta\mathbf{D}_1 \\ \vdots \\ \mathscr{J}_\theta\mathbf{D}_{N_I} \\ \mathscr{J}_\theta\mathbf{B}_1 \\ \vdots \\ \mathscr{J}_\theta\mathbf{B}_{N_B} \end{pmatrix}\begin{pmatrix} \mathscr{J}_\theta\mathbf{D}_1 \\ \vdots \\ \mathscr{J}_\theta\mathbf{D}_{N_I} \\ \mathscr{J}_\theta\mathbf{B}_1 \\ \vdots \\ \mathscr{J}_\theta\mathbf{B}_{N_B} \end{pmatrix}^{\mathrm{T}}. \tag{2.9b}$$

Here, $\mathscr{J}_\theta\mathbf{D}_k$ is the Jacobian of $\mathbf{D}_k(\theta(t))$ with respect to θ.

The proof is rather elementary and essentially amounts to rearranging terms. For two objectives it can be found in the appendix of [37].

Proof. We start by calculating one entry of the vector on the left-hand side of (2.9). The chain rule and (2.5) yield

$$\begin{aligned} \left(\frac{\mathrm{d}\mathbf{D}_\ell(\theta(t))}{\mathrm{d}t}\right)_j &= \frac{\mathrm{d}\mathscr{D}_\ell[y_{\theta(t)}](x_j^I)}{\mathrm{d}t} \\ &= \left(\nabla_\theta\mathscr{D}_\ell[y_{\theta(t)}](x_j^I), \frac{\mathrm{d}\theta(t)}{\mathrm{d}t}\right) \\ &= -\left(\nabla_\theta\mathscr{D}_\ell[y_{\theta(t)}](x_j^I), \nabla_\theta\mathscr{L}(\theta(t))\right) \end{aligned} \tag{2.10}$$

for all $\ell = 1,\dots,N_I$ and $j = 1,\dots,m_I$ Here $(\cdot,\cdot)$ denotes the Euclidean inner product.

Next, we need to compute $\nabla_\theta\mathscr{L}(\theta)$. Starting out with a single partial derivative, we obtain

$$\begin{aligned} \frac{\partial\mathscr{L}(\theta)}{\partial\theta_\ell} &= \frac{\partial}{\partial\theta_\ell}\Big[\frac{1}{2}\sum_{k=1}^{N_I}\sum_{i=1}^{m_I}\big[\mathscr{D}_k[y_\theta](x_i^I)\big]^2 \\ &\qquad + \frac{1}{2}\sum_{k=1}^{N_B}\sum_{i=1}^{m_B}\big[\mathscr{B}_k[y_\theta](x_i^B)\big]^2\Big] \\ &= \sum_{k=1}^{N_I}\sum_{i=1}^{m_I}\mathscr{D}_k[y_\theta](x_i^I)\frac{\partial}{\partial\theta_\ell}\mathscr{D}_k[y_\theta](x_i^I) \\ &\qquad + \sum_{k=1}^{N_B}\sum_{i=1}^{m_B}\mathscr{B}_k[y_\theta](x_i^B)\frac{\partial}{\partial\theta_\ell}\mathscr{B}_k[y_\theta](x_i^B), \end{aligned}$$

which means that

$$\nabla_\theta \mathscr{L}(\theta) = \sum_{k=1}^{N_I} \sum_{i=1}^{m_I} \mathscr{D}_k[y_\theta](x_i^I) \nabla_\theta \mathscr{D}_k[y_\theta](x_i^I) + \sum_{k=1}^{N_B} \sum_{i=1}^{m_B} \mathscr{B}_k[y_\theta](x_i^B) \nabla_\theta \mathscr{B}_k[y_\theta](x_i^B)$$

holds. Inserting into (2.10), we get

$$\left(\frac{\mathrm{d}\mathbf{D}_\ell(\theta(t))}{\mathrm{d}t}\right)_j \tag{2.11}$$
$$= -\sum_{k=1}^{N_I} \sum_{i=1}^{m_I} \mathscr{D}_k[y_{\theta(t)}](x_i^I) \left(\nabla_\theta \mathscr{D}_\ell[y_{\theta(t)}](x_j^I), \nabla_\theta \mathscr{D}_k[y_{\theta(t)}](x_i^I)\right)$$
$$- \sum_{k=1}^{N_B} \sum_{i=1}^{m_B} \mathscr{B}_k[y_{\theta(t)}](x_i^B) \left(\nabla_\theta \mathscr{D}_\ell[y_{\theta(t)}](x_j^I), \nabla_\theta \mathscr{B}_k[y_{\theta(t)}](x_i^B)\right).$$

Next, we observe that

$$\begin{aligned}\left(\nabla_\theta \mathscr{D}_\ell[y_{\theta(t)}](x_j^I), \nabla_\theta \mathscr{D}_k[y_{\theta(t)}](x_i^I)\right) &= \sum_{n=1}^{N_\theta} \frac{\partial \mathscr{D}_\ell[y_{\theta(t)}](x_j^I)}{\partial \theta_n} \cdot \frac{\partial \mathscr{D}_k[y_{\theta(t)}](x_i^I)}{\partial \theta_n} \\ &= \left[\mathscr{J}_\theta \mathbf{D}_\ell(\theta(t)) \left(\mathscr{J}_\theta \mathbf{D}_k(\theta(t))\right)^{\mathrm{T}}\right]_{ji}\end{aligned}$$

holds, where $\mathscr{J}_\theta \mathbf{D}_k$ is the Jacobian of $\mathbf{D}_k$ with respect to the network parameters θ, and N_θ denotes the number of parameters. Inserting into (2.11), we obtain

$$\begin{aligned}&\left(\frac{\mathrm{d}\mathbf{D}_\ell(\theta(t))}{\mathrm{d}t}\right)_j \\ &= -\sum_{k=1}^{N_I} \underbrace{\sum_{i=1}^{m_I} \mathscr{D}_k[y_{\theta(t)}](x_i^I) \left[\mathscr{J}_\theta \mathbf{D}_\ell(\theta(t)) \left(\mathscr{J}_\theta \mathbf{D}_k(\theta(t))\right)^{\mathrm{T}}\right]_{ji}}_{j\text{-th entry of } \mathscr{J}_\theta \mathbf{D}_\ell(\theta(t)) (\mathscr{J}_\theta \mathbf{D}_k(\theta(t)))^{\mathrm{T}} \mathbf{D}_k} \\ &\quad - \sum_{k=1}^{N_B} \underbrace{\sum_{i=1}^{m_B} \mathscr{B}_k[y_{\theta(t)}](x_i^B) \left[\mathscr{J}_\theta \mathbf{D}_\ell(\theta(t)) \left(\mathscr{J}_\theta \mathbf{D}_k(\theta(t))\right)^{\mathrm{T}}\right]_{ji}}_{j\text{-th entry of } \mathscr{J}_\theta \mathbf{D}_\ell(\theta(t)) (\mathscr{J}_\theta \mathbf{B}_k(\theta(t)))^{\mathrm{T}} \mathbf{B}_k},\end{aligned}$$

which means

$$\begin{aligned}\left(\frac{\mathrm{d}\mathbf{D}_\ell}{\mathrm{d}t}\right)_j &= -\sum_{k=1}^{N_I} \left[\mathscr{J}_\theta \mathbf{D}_\ell \left(\mathscr{J}_\theta \mathbf{D}_k\right)^{\mathrm{T}} \mathbf{D}_k\right]_j \\ &\quad - \sum_{k=1}^{N_B} \left[\mathscr{J}_\theta \mathbf{D}_\ell \left(\mathscr{J}_\theta \mathbf{B}_k\right)^{\mathrm{T}} \mathbf{B}_k\right]_j.\end{aligned}$$

A similar calculation shows

$$\begin{aligned}\left(\frac{\mathrm{d}\mathbf{B}_\ell}{\mathrm{d}t}\right)_j &= -\sum_{k=1}^{N_I} \left[\mathscr{J}_\theta \mathbf{B}_\ell \left(\mathscr{J}_\theta \mathbf{D}_k\right)^{\mathrm{T}} \mathbf{D}_k\right]_j \\ &\quad - \sum_{k=1}^{N_B} \left[\mathscr{J}_\theta \mathbf{B}_\ell \left(\mathscr{J}_\theta \mathbf{B}_k\right)^{\mathrm{T}} \mathbf{B}_k\right]_j.\end{aligned}$$

With these two equations we can finally assemble the entire system

$$\frac{\mathrm{d}}{\mathrm{d}t} \begin{pmatrix} \mathbf{D}_1 \\ \vdots \\ \mathbf{D}_{N_I} \\ \mathbf{B}_1 \\ \vdots \\ \mathbf{B}_{N_B} \end{pmatrix} = - \underbrace{\begin{pmatrix} \mathscr{J}_\theta \mathbf{D}_1 \\ \vdots \\ \mathscr{J}_\theta \mathbf{D}_{N_I} \\ \mathscr{J}_\theta \mathbf{B}_1 \\ \vdots \\ \mathscr{J}_\theta \mathbf{B}_{N_B} \end{pmatrix} \begin{pmatrix} \mathscr{J}_\theta \mathbf{D}_1 \\ \vdots \\ \mathscr{J}_\theta \mathbf{D}_{N_I} \\ \mathscr{J}_\theta \mathbf{B}_1 \\ \vdots \\ \mathscr{J}_\theta \mathbf{B}_{N_B} \end{pmatrix}^{\mathrm{T}}}_{=:K(t)} \begin{pmatrix} \mathbf{D}_1 \\ \vdots \\ \mathbf{D}_{N_I} \\ \mathbf{B}_1 \\ \vdots \\ \mathbf{B}_{N_B} \end{pmatrix}.$$

The symmetry and positive semi-definiteness of $K(t)$ are evident. □

The authors of [37] argue that (under certain assumptions) the NTK $K(t)$ remains approximately constant and the training dynamics (2.9) of PINNs are well-described by the spectrum of the NTK, where components parallel to eigenvectors corresponding to large eigenvalues of the NTK decay faster. For this reason, for a positive semi-definite kernel matrix $K \in \mathbb{R}^{n\times n}$, they define the average convergence rate $c(K)$ as the mean of its eigenvalues, i. e.,

$$c(K) = \frac{\operatorname{trace}(K)}{n}. \tag{2.12}$$

For fully connected neural networks, eigenvectors that belong to large eigenvalues have been conjectured to represent lower frequencies of the solution. Hence, fully connected neural networks tend to learn lower frequencies faster — a phenomenon termed spectral bias [28]. The authors of [37] further observe that eigenvalues of the NTK of PINNs decay fast and thus conclude that PINNs also suffer from spectral bias. The gradient flow (2.5) is therefore a stiff ODE system. To calculate weights that homogenize the convergence rate in the sense of (2.12), i. e., to remove spectral bias, they propose the following. First, define the diagonal entries of the block matrix $K = K(t)$ from (2.9) as

$$\begin{aligned}K_k^{\mathbf{D}} &:= \mathscr{J}_\theta \mathbf{D}_k (\mathscr{J}_\theta \mathbf{D}_k)^{\mathrm{T}}, \quad k = 1, \dots, N_I, \\ K_k^{\mathbf{B}} &:= \mathscr{J}_\theta \mathbf{B}_k (\mathscr{J}_\theta \mathbf{B}_k)^{\mathrm{T}}, \quad k = 1, \dots, N_B.\end{aligned}$$

Noting that K has block form, one obtains

$$\operatorname{trace}(K) = \sum_{k=1}^{N_I} \operatorname{trace}(K_k^{\mathbf{D}}) + \sum_{k=1}^{N_B} \operatorname{trace}(K_k^{\mathbf{B}}).$$

Then, we introduce weights λ_k^I, λ_k^B into (2.8)

$$\mathscr{L}(\theta) = \frac{1}{2} \sum_{k=1}^{N_I} \lambda_k^I \|\mathbf{D}_k(\theta)\|_2^2 + \frac{1}{2} \sum_{k=1}^{N_B} \lambda_k^B \|\mathbf{B}_k(\theta)\|_2^2$$

with the choices

$$\lambda_k^I := \frac{c(K)}{c(K_k^{\mathbf{D}})} = \frac{\operatorname{trace}(K)}{\operatorname{trace}(K_k^{\mathbf{D}})},$$
$$\lambda_k^B := \frac{c(K)}{c(K_k^{\mathbf{B}})} = \frac{\operatorname{trace}(K)}{\operatorname{trace}(K_k^{\mathbf{B}})}.$$

These weights rescale the trace of the NTK and lead to a homogeneous convergence rate in the sense of (2.12). Although the NTK is assumed to be constant for the theoretical analysis, we follow [36] and update the NTK loss weights in regular intervals over the course of training.

2.2 Hard-Constraining the Boundary and Gradient Equations

In our experiments, despite a hyperparameter search, none of the weighting schemes described above is successful in guaranteeing convergence to a good solution of the optimality system (1.4) for $\gamma = 10^{-3}$ using the five objective terms (2.1). In particular, the boundary conditions and the linear relation between control and adjoint variable (1.4a) could not be recovered (see fig. 3.2). Although the hyperparameter search was not exhaustive, these findings are in line with [6] who report similar difficulties.

Thus, we now modify the network architecture in order to eliminate objective terms such that some of the residuals are inherently satisfied. As in [31], we impose the Dirichlet boundary condition exactly. This technique has been used to solve stochastic PDEs [18], optimal control problems [39], and it can be extended to other forms of boundary conditions [31, 22]. Since we face homogeneous Dirichlet conditions, we multiply the neural network predictions for state and adjoint y_θ, p_θ with D, a distance-like function to the boundary $\partial\Omega$:

$$\overline{y}_\theta(x) := y_\theta(x)\, D(x), \quad \overline{p}_\theta(x) := p_\theta(x)\, D(x). \tag{2.13}$$

Because $D(x) = 0$ holds on the boundary, the model exactly satisfies the homogeneous Dirichlet conditions independently of the values of the network's parameters θ. The neural network then only learns the deviations of the true solutions from D. The exact distance function usually has some discontinuities in the first derivative, which is why we use a smooth approximation. Such approximations can be obtained in a variety of ways even for complex domains, for instance, using R-functions and mean value potentials [31], or by solving heat transfer problems [4].

The next design step exploits the specific structure of the problem (1.3), namely, that the linear gradient equation (1.4b) can also be explicitly imposed in the network architecture. This is done by simply replacing the network's output representing the control variable u_θ with $\overline{u}_\theta := \gamma^{-1}\overline{p}_\theta$. This idea has also been described in [6] and it can be easily extended to problems with pointwise bound constraints on the control. A disadvantage of this architecture is that it requires the gradient equation to be explicitly solvable w.r.t. the control variable u as a function of y and p. The authors of [6] propose that this hard constraint could be replaced with a small separate neural network to approximate nonlinear control-adjoint relations, as they appear for more complicated optimal control problems. However, since they did not provide any numerical results it remains to be seen how well this works in practice.

The resulting architecture for the optimal control problem (1.3) with fixed control cost γ is visualized in fig. 2.2. Using this architecture together with suitable loss weights for the two remaining objective terms

$$\mathscr{L}_{\text{state}}(\theta) = \frac{1}{2}\frac{1}{m_I}\sum_{i=1}^{m_I}\left[-\Delta\overline{y}_\theta(x_i^I) - \gamma^{-1}\overline{p}_\theta(x_i^I)\right]^2,$$
$$\mathscr{L}_{\text{adjoint}}(\theta) = \frac{1}{2}\frac{1}{m_I}\sum_{i=1}^{m_I}\left[-\Delta\overline{p}_\theta(x_i^I) + \overline{y}_\theta(x_i^I) - y_d(x_i^I)\right]^2,$$

we are now able to approximate the solution of (1.3) for $\gamma = 10^{-3}$ to reasonable accuracy (see bottom row of fig. 3.2 and fig. 3.6).

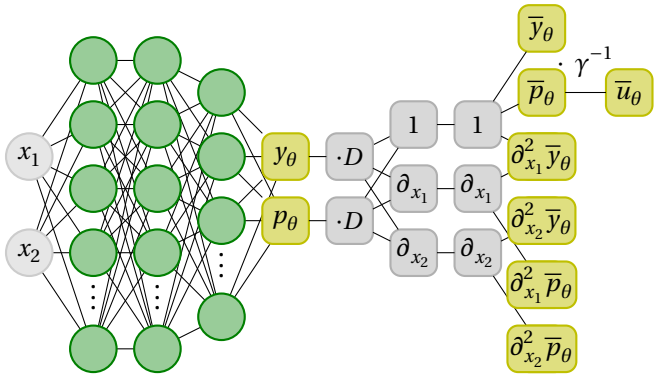

Figure 2.2 – Schematic network architecture. Outputs of the network are multiplied by the approximate distance function D. The calculation of derivatives is depicted as differentiation layers. The output of the control variable $\overline{u}_\theta$ is hard-coded as $\gamma^{-1}\overline{p}_\theta$.

2.3 Loss Weights for Variable Control Cost

Having obtained a PINN model that can be trained to reasonable accuracy for a range of fixed values of the control cost parameter γ including $\gamma = 10^{-3}$, we turn to solving the family of parameter-dependent problems by a single PINN. By treating γ as an additional input to the network and including loss terms for various sample

values of γ, we may approximate solutions to the family of problems in a single training run. After the reduction steps described in the last paragraph, only state and adjoint equation remain which we equip with weights λ_{state} and λ_{adjoint}. Note however, that it is necessary to choose γ-dependent weights, as the influence of γ on gradient scaling made individually weighting the loss terms necessary in the first place fig. 3.4. Given samples γ_j, $j = 1, \dots, N_\gamma$, we thus derive the loss function

$$\mathscr{L}(\theta) = \sum_{j=1}^{N_\gamma} \lambda_{\text{state}}(\gamma_j)\, \mathscr{L}_{\text{state}}(\theta, \gamma_j) + \sum_{j=1}^{N_\gamma} \lambda_{\text{adjoint}}(\gamma_j)\, \mathscr{L}_{\text{adjoint}}(\theta).$$

Unfortunately, this leads to $2N_\gamma$ independently weighted terms in the loss function. Increasing the number of γ-samples to reduce interpolation error simultaneously increases the number of objectives. This reinforces the problem's multi-objective nature, making it arguably harder to solve.

3 Numerical Results

We first give a general overview about the experimental setup, and discuss our results in detail afterwards.

Experimental Setup

Our numerical experiments have been performed on a square domain $\Omega = (-1,1)^2$ and we have chosen the desired state in (1.3) as the step function $y_d(x_1, x_2) = 3 \cdot \mathbb{1}_{x_1 \geq 0}$. As ground truth state and control variables we use a finite element solution that was obtained by L-BFGS minimization of the reduced form of (1.3), cf. [10, 15], where Euclidean gradients were calculated using the DOLFIN-ADJOINT library [25]. To obtain a ground truth for the adjoint variable, we multiplied the calculated finite element control by γ. The experiments were performed on a regular 64-by-64 triangle mesh. To assess the quality of the approximations we calculate relative error measures

$$E^{L^2}_{\text{state}} = \frac{\|\overline{y}_\theta - y_{\text{fem}}\|_{L^2(\Omega)}}{\|y_{\text{fem}}\|_{L^2(\Omega)}},$$

where we approximate the integrals using a two-dimensional trapezoidal rule on the mesh vertices $x_1, \dots, x_n$ for $n = 64^2$, as well as

$$E^{L^\infty}_{\text{state}} = \frac{\max\limits_{i=1,\dots,n} |\overline{y}_\theta(x_i) - y_{\text{fem}}(x_i)|}{\max\limits_{i=1,\dots,n} |y_{\text{fem}}(x_i)|}.$$

We consider solutions for $\gamma \in [10^{-5}, 1]$. This choice is motivated as follows. On the one hand, $\gamma \gg 1$ would lead to solutions y very far from the desired state y_d, i. e., the control problem loses its meaning. On the other hand, for $\gamma \leq 10^{-5}$, the finite element solution is no longer a reliable ground truth on the chosen grid.

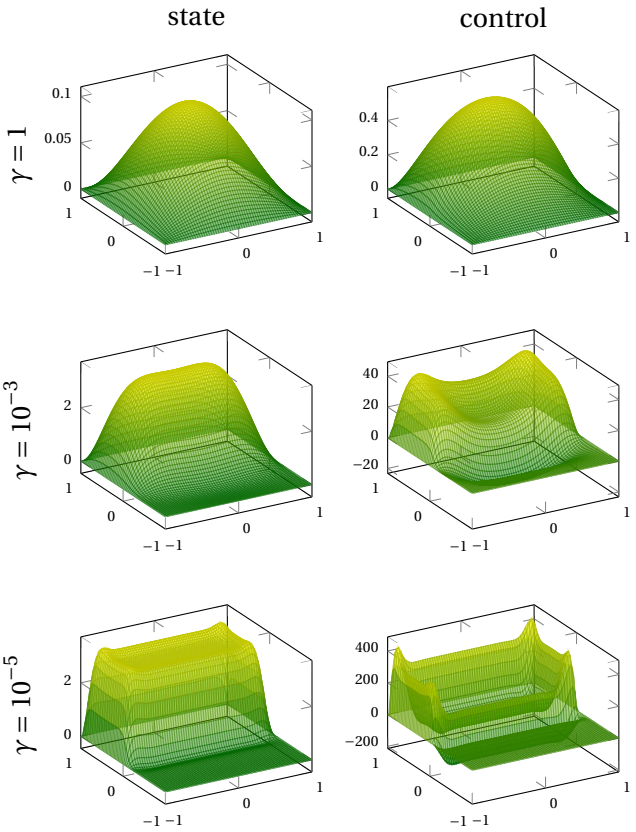

Figure 3.1 – Finite element solutions for several values of γ. While the state variable values are roughly bounded between 0 and 3, the values for the control variable increase in magnitude for smaller values of γ.

With our finite element ground truth solution established, we perform a range of experiments for a range of γ values. Our experiments indicate that the unweighted PINN model using the five-term loss function (2.1) struggles to learn solutions with fixed $\gamma \leq 10^{-3}$. To establish which of the proposed measures enable model training to good accuracy below this critical value of γ, we first test combinations of architectures and NTK loss weighting in section 3.1. Next, we train models without adaptive loss weighting for a range of fixed values of γ and record their gradient histograms. We find that increasing imbalances in gradient distributions are a plausible explanation for the decay of approximation quality; see section 3.2. After that, in section 3.3, we return to fixed $\gamma = 10^{-3}$ and examine the capabilities of the proposed loss weighting schemes to counteract these imbalances. Finally, we turn to approximating the

ID	hidden layers	system size	loss weights	γ	N_γ	m_B	m_I	epochs	sect.
1a	40,40,20	full	None	$\gamma = 10^{-3}$	1	500	1000	10000	3.1
1b	40,40,20	full	NTK	$\gamma = 10^{-3}$	1	500	1000	10000	3.1
1c	40,40,20	reduced	None	$\gamma = 10^{-3}$	1	0	1000	10000	3.1
1d	40,40,20	reduced	NTK	$\gamma = 10^{-3}$	1	0	1000	10000	3.1
2	40,40,20	reduced	None	$\gamma = 10^{-5}, \dots, 10^{0}$	1	0	1000	10000	3.2
3a	40,40,20	reduced	ReLoBraLo	$\gamma = 10^{-3}$	1	0	1000	10000	3.3
3b	40,40,20	reduced	InvDirichlet	$\gamma = 10^{-3}$	1	0	1000	10000	3.3
3c	40,40,20	reduced	NTK	$\gamma = 10^{-3}$	1	0	1000	10000	3.3
4a	40,40,40,40	reduced	InvDirichlet	$\gamma \in [10^{-4}, 1]$	20	0	1000	30000	3.4
4b	40,40,20	reduced	InvDirichlet	$\gamma = 10^{-4}, \dots, 10^{0}$	1	0	1000	10000	3.4
4c	40,40,40,40	reduced	NTK	$\gamma \in [10^{-4}, 1]$	20	0	1000	30000	3.4
4d	40,40,20	reduced	NTK	$\gamma = 10^{-4}, \dots, 10^{0}$	1	0	1000	10000	3.4

Table 3.1 – Details of architecture and training setups. ID: 4a and ID: 4c examine the variable γ setting. System size "full", refers to training with all five-term loss terms (2.1), while "reduced" refers to training with hard-constrained gradient equation and boundary conditions as described in section 2.2. All networks use hyperbolic tangent activation functions. We perform full-batch ADAM optimization with a learning rate of 10^{-3}. During training, weight updates are performed every 100 epochs.

γ-dependent family of solutions in section 3.4.

In all of our experiments we use hyperbolic tangent activation and train the networks using full-batch ADAM optimization with a learning rate of 10^{-3}. Whenever we employ loss weighting algorithms, weight updates are performed every 100 epochs. The detailed setup for each experiment can be found in table 3.1.

3.1 Experiment 1: Comparing Combinations of Loss Weighting and Architecture Constraints

As mentioned in section 2, preliminary experiments suggest that the unweighted PINN formulation using the 5-term loss function (2.1) struggles to learn a solution of (1.4) for fixed $\gamma \leq 10^{-3}$ (cf. second row of fig. 3.2). To investigate which measures are sufficient to enable the network to learn the solution below this critical value, we train networks using combinations of the measures described in section 2.1 and section 2.2.

When introducing NTK loss weights, the accuracies for state and adjoint variable approximations improve, but the gradient equation is still poorly satisfied (cf. third row of fig. 3.2). The other weighting schemes lead to equally poor results (not shown). Next, we hard-constrain the gradient equation and explicitly impose boundary conditions using the following polynomial approximation to the distance-to-the-boundary function

$$D(x_1, x_2) = (x_1 - 1)(x_1 + 1)(x_2 - 1)(x_2 + 1),$$

for the square domain $\Omega = (-1, 1)^2$, which has also been successfully employed in [39]. Training this reduced two-equation system without adaptive loss weighting does not meaningfully improve the convergence (cf. fourth row in fig. 3.2). Only a model trained using both of the two measures is able to successfully learn the solution in our training setup (cf. bottom row in fig. 3.2).

3.2 Experiment 2: Examining Gradient Imbalances as a Proxy for Optimization Bias

We observe a broad trend in worse performance for smaller values of γ, so we perform a systematic study using fixed, logarithmically spaced γ-values in the interval of interest $[10^{-5}, 1]$. Training the reduced system consisting of objective terms $\mathscr{L}_{\text{state}}$ and $\mathscr{L}_{\text{adjoint}}$ without loss weighting, the model does not converge for small values of γ; see fig. 3.3.

As mentioned in section 2, PINN training failures have been attributed to biased multi objective optimization. To gain further insight, we follow previous authors who proposed to examine the histograms of objective-specific gradients [11, 36]. We have combined

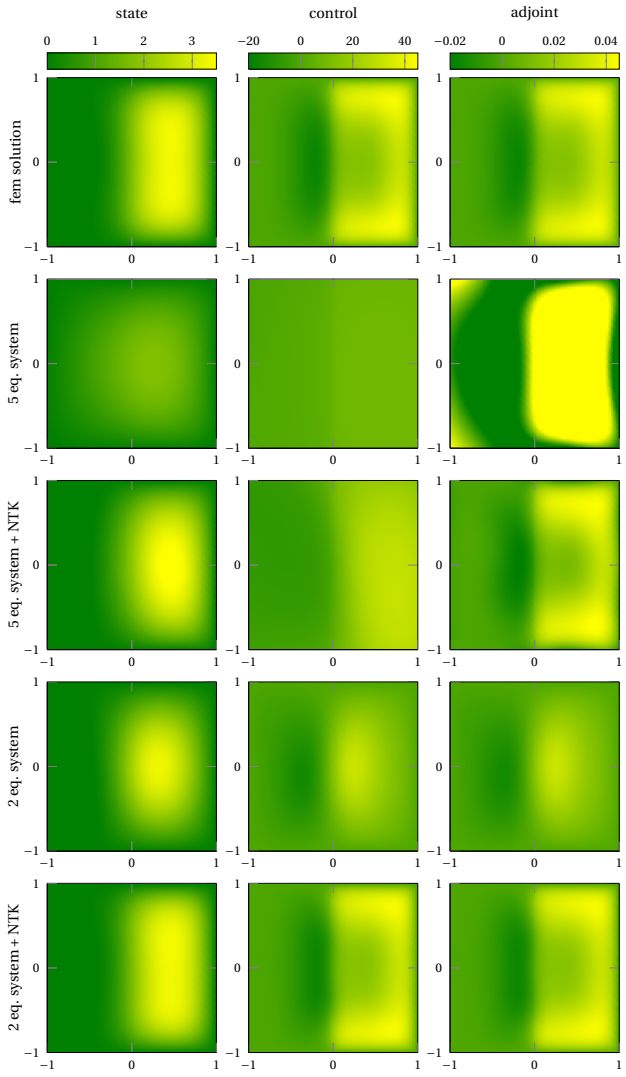

Figure 3.2 – Overview of solutions for (1.3) obtained with different setups for fixed $\gamma = 10^{-3}$. Top row: finite element ground truth solution. Second row: training is not successful using the loss terms (2.1) without loss weights. Notice that the color values for the adjoint variable are truncated. Third row: introduction of NTK weights improves state and adjoint accuracy, while the control value is still poorly captured. Fourth row: with the architecture from section 2.2 state and adjoint still show qualitatively different behavior from the ground truth. Fifth row: with NTK loss weights, we obtain a favorable solution. Detailed setup in table 3.1, ID: 1a–1d.

the objective-specific gradients of 10 training runs and show the corresponding histograms in fig. 3.4. Indeed we find similar distributions in the gradients of state and adjoint equation when approximation quality is good (i. e., for $\gamma \approx 1$), which become increasingly imbalanced as γ decreases. To visualize this trend for decreasing γ, we calculate the quotient of the variances of these

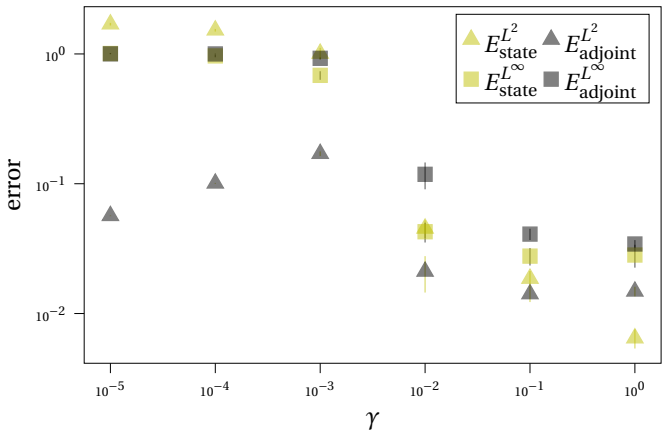

Figure 3.3 – Evolution of final prediction errors of the unweighted reduced model versus γ, averaged over 10 training runs, with vertical lines indicating one standard deviation. Quality of fits decreases significantly for γ smaller than 10^{-3}. Detailed setup in table 3.1, ID: 2.

two distributions over the course of training; see fig. 3.5. While subject to randomness, this relation of gradient variances remains approximately constant over the course of training. Large values, indicating optimization bias, correspond to large approximation errors. We can thus attribute the difficulties in convergence to imbalanced gradient distributions.

3.3 Experiment 3: Comparison of Loss Weighting Algorithms for the Case of Fixed γ

Having seen that the convergence failure is caused by an optimization bias, we investigate the capacity of the proposed loss weighting schemes to counteract this bias. We compare all adaptive weighting schemes introduced in section 2.1, by training 10 models each for the previously critical value $\gamma = 10^{-3}$; see fig. 3.6. Convergence is only improved for NTK and InvDirichlet weights, while ReLoBraLo weights yield no advantage over the unweighted model. We suspect that tuning of the hyperparameters in the ReLoBraLo model can improve convergence because in experiments with a different desired state $y_d \equiv 3$, ReLoBraLo weights have led to improved convergence accuracy (results not shown). Errors for the improved models are of order 10^{-2} with $E^{L^\infty}_{\text{adjoint}}$ of order 10^{-1} being the exception. Our attempts to lower these relatively high errors by tweaking (hyper-)parameters like the number of network parameters, learning rate and training time, were unsuccessful.

Next, we decrease γ beyond 10^{-3}. NTK and InvDirichlet weights reduce model accuracy to reason-

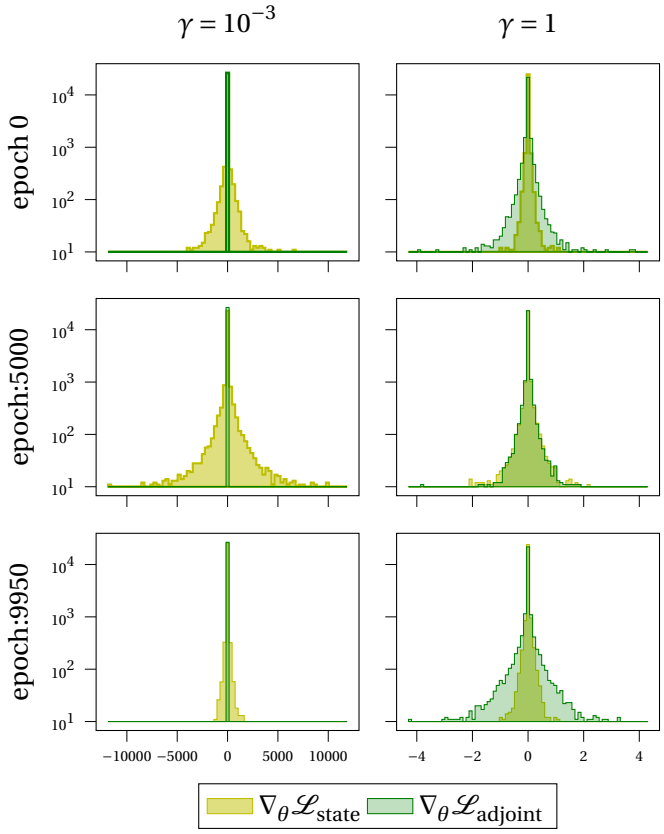

Figure 3.4 – Histograms of objective-specific gradients. Each panel shows the distribution of partial derivatives $\frac{\partial}{\partial \theta_i}$ accumulated over 10 training runs. For $\gamma = 1$, partial derivatives of state and adjoint equations show similar distributions (right column), while the variance of state equation derivatives is much larger for $\gamma = 10^{-3}$, indicating increasing optimization bias. Detailed setup in table 3.1, ID: 2.

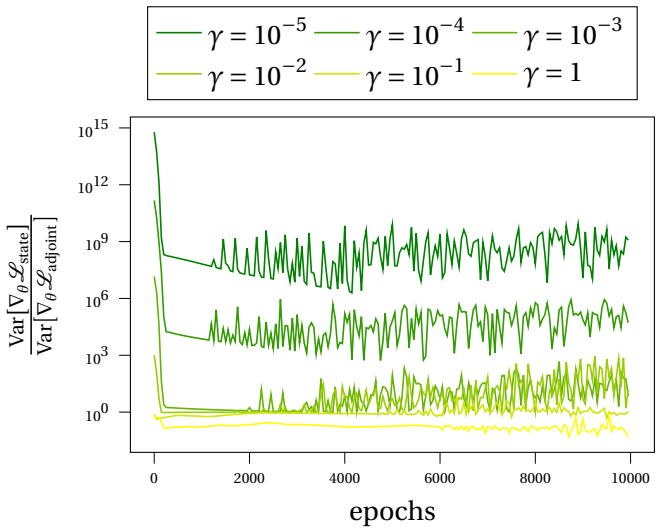

Figure 3.5 – Relation of objective-specific gradient variances for different values of γ. Large quotients indicate training imbalances, which explain the poor approximation quality for $\gamma \leq 10^{-3}$, confirming the behavior in fig. 3.3. Detailed setup in table 3.1, ID: 2.

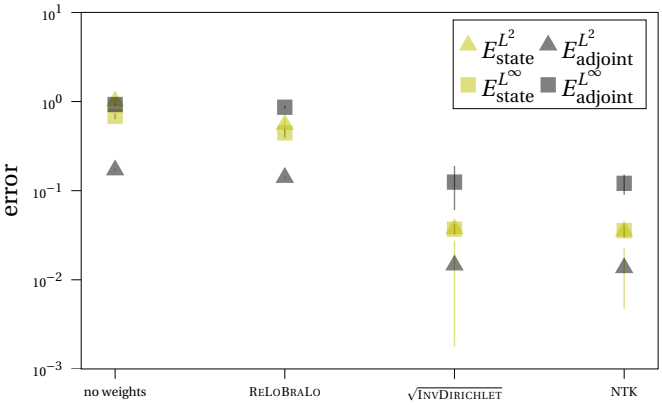

Figure 3.6 – Final prediction errors for the critical value $\gamma = 10^{-3}$ versus different loss weighting schemes averaged over 10 training runs, with vertical lines indicating one standard deviation. While ReLoBraLo weights provide no advantage over the unweighted model, NTK and InvDirichlet are able to reduce errors. Detailed setup table 3.1, ID: 2, 3a–3c.

able levels for $\gamma = 10^{-4}$. For $\gamma = 10^{-5}$ however, none of the proposed weighting schemes ensures convergence to a good solution in 10 000 training epochs (not shown). Again, we have been unable to find hyperparameter settings which lead to lower errors. This suggests a limit in the ability of loss weights to compensate the increasing optimization bias. As a consequence, we decide to restrict γ to $[10^{-4}, 1]$ for the parameter-dependent models.

3.4 Experiment 4: Approximation of the γ-Dependent Family of Solutions

Pivoting to the variable-γ setting, our goal was to explore whether the accuracy of fixed parameter models can be achieved by one large model trained in a one-shot approach. To this end, we add a third input unit and increase the network's hidden dimensions to 4 layers of 40 neurons each. Then we train a one-shot model for $\gamma \in [10^{-4}, 1]$, sampling $N_\gamma = 20$ logarithmically spaced values. Training is again performed using full batch ADAM, meaning that in each epoch, the network was trained on all $N_\gamma \cdot m_I = 20 \cdot 1000$ combinations of values of γ and collocation points. As discussed in section 2.3, the loss weights are γ-dependent, since fig. 3.4 indicates a difference of several orders of magnitude in gradient variances when training fixed parameter models for dif-

ferent values of γ.

For reference, we train models with fixed $\gamma = 10^{-4}, \ldots, 10^{0}$ utilizing the same training setup as in section 3.3 (details in table 3.1, ID: 4b and 4d). While ReLoBraLo weights lead to numerical overflow, results after 30 000 epochs using InvDirichlet and NTK weights are shown in fig. 3.7. Like in the fixed parameter regimen, approximation errors increase for smaller γ-values. For InvDirichlet weights, the solution accuracy is similar to that of the fixed-γ models when $\gamma \in [10^{-3}, 1]$. While for small values of γ, errors of the NTK-weighted model are comparable to errors of corresponding fixed-γ model, there is a sharp increase for values $\gamma \geq 10^{-1}$.

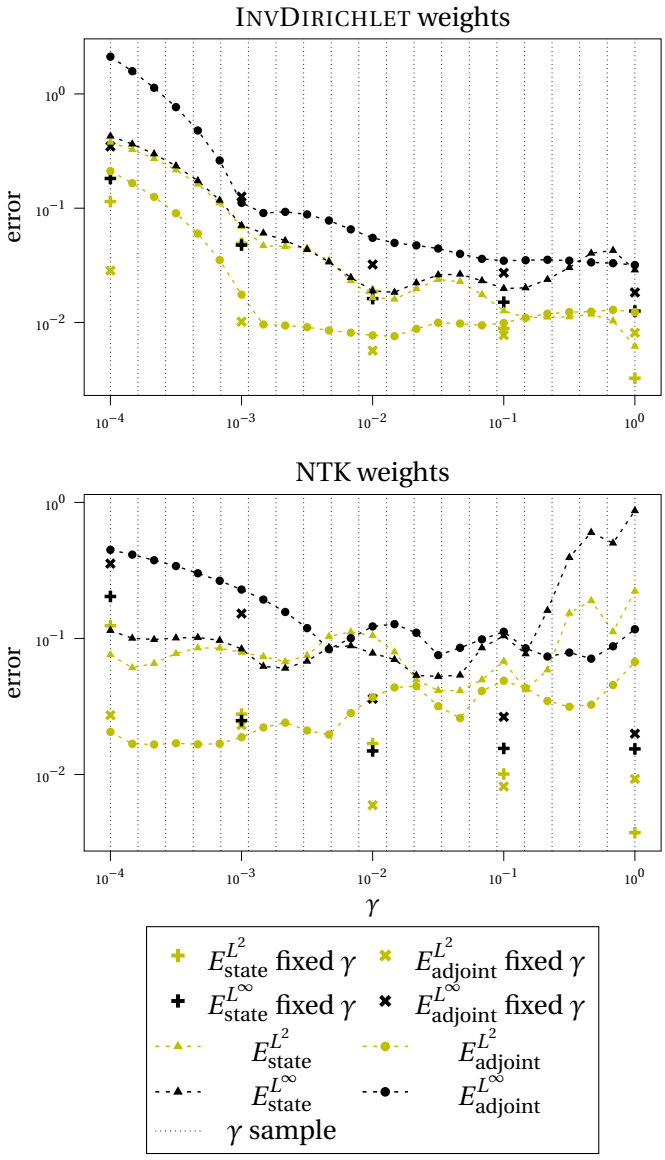

Figure 3.7 – Results of training with variable values of γ for 30 000 epochs using InvDirichlet (top panel) and NTK weights (bottom panel), together with prediction errors of fixed-γ models for comparison. Vertical dots indicate γ values used for training. When training with InvDirichlet weights, the model reaches comparable accuracy for $\gamma \geq 10^{-3}$. NTK-weighted training leads to generally larger errors and a pronounced discrepancy for $\gamma \geq 10^{-1}$. Detailed setup table 3.1, ID: 4a–4d .

4 Discussion

Although the PINN approach is easy to implement and very flexible, we encounter major difficulties when employing it to solve the simple optimal control model problem (1.3) for fixed $\gamma \leq 10^{-3}$. We attribute these difficulties to converge to good solutions to a well-documented phenomenon in general multi-objective optimization and specifically in PINN training, that is imbalanced objective-specific gradient distributions, resulting in an optimization bias. Following the literature we have tried to reduce this bias using several loss weighting schemes and have found some of them improving convergence, however satisfactory accuracy has only been achieved after reducing the number of equations from five to two, by designing a network architecture such that the gradient equation as well as state and adjoint boundary conditions are inherently satisfied. This leads to reasonable accuracy for some of the weighting algorithms and $\gamma \in [10^{-4}, 1]$. In case of InvDirichlet weights, similar accuracy can be achieved when approximating the family of parameter-dependent solutions in a one-shot approach for $\gamma \in [10^{-3}, 1]$. This is not the case for NTK weights, where the one-shot model exhibits overall worse accuracy Let us remark that we did not conduct a systematic hyperparameter tuning and used moderate training times due to limited computational resources. Alternatively, a reason for this poor accuracy could be the increase in number of objectives mentioned in section 1 and a limited capacity of the proposed schemes to reduce optimization bias for a high numbers of loss terms. We conclude that the model used in this work can not be used reliably to solve the considered parameter-dependent optimal control problem.

Another potential reason for the observed convergence issues could be that the learning setup we use aims to find strong solutions of the partial differential equations. Although there exists literature establishing error estimates for this approach in case the PDE solution is smooth [5], numerical experiments suggest problems when approximating solutions with less regularity [24]. This bears similarity to the perceived increase in difficulty training models for small values of γ, since the optimal control of problem eq. (1.3) loses regularity as

$\gamma \searrow 0$.

A number of improvements have been proposed ranging from variational loss functions [7, 19], to learning the solution operator itself [21, 34]. Nevertheless, we conclude that the classical PINN formulation cannot be used as an "off-the-shelf" approach to obtain reliable solutions even for a simple optimal control problem of the form (1.3).

CRediT Author Statement

Johannes Wagner: Software, Validation, Formal analysis, Investigation, Writing - Original Draft, Visualization
Evelyn Herberg: Conceptualization, Methodology, Writing - Review and Editing, Supervision, Project administration
Roland Herzog: Conceptualization, Methodology, Writing - Review and Editing, Supervision, Project administration

References

[1] R. Bischof; M. Kraus. *Multi-objective loss balancing for physics-informed deep learning*. 2021. arXiv: 2110.09813 (cit. on pp. 33 sq.).

[2] Z. Chen; V. Badrinarayanan; C.-Y. Lee; A. Rabinovich. "GradNorm: gradient normalization for adaptive loss balancing in deep multitask networks". *Proceedings of the 35th International Conference on Machine Learning*. Ed. by J. Dy; A. Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm, Sweden: PMLR, 2018, pp. 794–803. arXiv: 1711.02257 (cit. on p. 34).

[3] P.-Y. Chuang; L. Barba. "Experience report of physics-informed neural networks in fluid simulations: pitfalls and frustration". *Proceedings of the 21st Python in Science Conference*. SciPy. SciPy, 2022, pp. 28–36. DOI: 10.25080/majora-212e5952-005. arXiv: 2205.14249 (cit. on p. 33).

[4] K. Crane; C. Weischedel; M. Wardetzky. "Geodesics in heat: a new approach to computing distance based on heat flow". *ACM Transactions on Graphics* 32.5 (2013), pp. 1–11. DOI: 10.1145/2516971.2516977. arXiv: 1204.6216 (cit. on p. 37).

[5] T. De Ryck; S. Mishra. "Generic bounds on the approximation error for physics-informed (and) operator learning". *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo; S. Mohamed; A. Agarwal; D. Belgrave; K. Cho; A. Oh. Vol. 35. Curran Associates, Inc., 2022, pp. 10945–10958. arXiv: 2205.11393 (cit. on p. 42).

[6] N. Demo; M. Strazzullo; G. Rozza. "An extended physics informed neural network for preliminary analysis of parametric optimal control problems". *Computers & Mathematics with Applications* 143 (2023), pp. 383–396. DOI: 10.1016/j.camwa.2023.05.004. arXiv: 2110.13530 (cit. on pp. 33, 37).

[7] W. E; B. Yu. "The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems". *Communications in Mathematics and Statistics* 6.1 (2018), pp. 1–12. DOI: 10.1007/s40304-018-0127-z. arXiv: 1710.00211 (cit. on p. 43).

[8] M. Ehrgott. *Multicriteria Optimization*. Second. Springer, Berlin, 2005. DOI: 10.1007/3-540-27659-9 (cit. on p. 33).

[9] O. Fuks; H. A. Tchelepi. "Limitations of physics informed machine learning for nonlinear two-phase transport in porous media". *Journal of Machine Learning for Modeling and Computing* 1.1 (2020), pp. 19–37. DOI: 10.1615/jmachlearnmodelcomput.2020033905 (cit. on p. 33).

[10] S. W. Funke. *dolfin-adjoint Documentation, Optimal control of the Poisson equation*. Accessed: 2024-12-26 (cit. on p. 38).

[11] X. Glorot; Y. Bengio. "Understanding the difficulty of training deep feedforward neural networks". *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Y. W. Teh; M. Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 2010, pp. 249–256 (cit. on pp. 34, 39).

[12] A. Griewank; A. Walther. *Evaluating Derivatives*. 2nd ed. Principles and techniques of algorithmic differentiation. Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM), 2008. DOI: 10.1137/1.9780898717761 (cit. on p. 31).

[13] T. G. Grossmann; U. J. Komorowska; J. Latz; C.-B. Schönlieb. "Can physics-informed neural networks beat the finite element method?" *IMA Journal of Applied Mathematics* 89.1 (2024), pp. 143–174. DOI: 10.1093/imamat/hxae011. arXiv: 2302.04107 (cit. on p. 32).

[14] E. Haber; L. Ruthotto. “Stable architectures for deep neural networks”. *Inverse Problems* 34.1 (2017), p. 014004. DOI: 10.1088/1361-6420/aa9a90. arXiv: 1705.03341 (cit. on p. 35).

[15] R. Herzog; K. Kunisch. “Algorithms for PDE-constrained optimization”. *GAMM Reports* 33.2 (2010), pp. 163–176. DOI: 10.1002/gamm.201010013 (cit. on p. 38).

[16] A. A. Heydari; C. A. Thompson; A. Mehmood. *SoftAdapt: techniques for adaptive loss weighting of neural networks with multi-part loss functions*. 2019. arXiv: 1912.12355 (cit. on p. 34).

[17] A. Jacot; F. Gabriel; C. Hongler. “Neural tangent kernel: convergence and generalization in neural networks”. *Advances in Neural Information Processing Systems*. Ed. by S. Bengio; H. Wallach; H. Larochelle; K. Grauman; N. Cesa-Bianchi; R. Garnett. Vol. 31. 2018. arXiv: 1806.07572 (cit. on p. 35).

[18] S. Karumuri; R. Tripathy; I. Bilionis; J. Panchal. “Simulator-free solution of high-dimensional stochastic elliptic partial differential equations using deep neural networks”. *Journal of Computational Physics* 404 (2020), p. 109120. DOI: 10.1016/j.jcp.2019.109120. arXiv: 1902.05200 (cit. on p. 37).

[19] E. Kharazmi; Z. Zhang; G. E. Karniadakis. *Variational physics-informed neural networks for solving partial differential equations*. 2019. arXiv: 1912.00873 (cit. on p. 43).

[20] D. P. Kingma; J. Ba. “Adam: a method for stochastic optimization”. *3rd International Conference on Learning Representations, ICLR 2015*. Ed. by Y. Bengio; Y. LeCun. 2015. arXiv: 1412.6980 (cit. on p. 32).

[21] L. Lu; P. Jin; G. E. Karniadakis. *DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators*. 2019. arXiv: 1910.03193 (cit. on p. 43).

[22] L. Lu; R. Pestourie; W. Yao; Z. Wang; F. Verdugo; S. G. Johnson. “Physics-informed neural networks with hard constraints for inverse design”. *SIAM Journal on Scientific Computing* 43.6 (2021), B1105–B1132. DOI: 10.1137/21m1397908. arXiv: 2102.04626 (cit. on p. 37).

[23] S. Maddu; D. Sturm; C. L. Müller; I. F. Sbalzarini. “Inverse Dirichlet weighting enables reliable training of physics informed neural networks”. *Machine Learning: Science and Technology* 3.1 (2022), p. 015026. DOI: 10.1088/2632-2153/ac3712. arXiv: 2107.00940 (cit. on p. 34).

[24] S. Mishra; R. Molinaro. “Estimates on the generalization error of physics-informed neural networks for approximating a class of inverse problems for PDEs”. *IMA Journal of Numerical Analysis* 42.2 (2021), pp. 981–1022. DOI: 10.1093/imanum/drab032. arXiv: 2006.16144 (cit. on p. 42).

[25] S. Mitusch; S. Funke; J. Dokken. “dolfin-adjoint 2018.1: automated adjoints for FEniCS and Firedrake”. *Journal of Open Source Software* 4.38 (2019), p. 1292. DOI: 10.21105/joss.01292 (cit. on p. 38).

[26] S. Mowlavi; S. Nabi. “Optimal control of PDEs using physics-informed neural networks”. *Journal of Computational Physics* 473 (2023), p. 111731. DOI: 10.1016/j.jcp.2022.111731. arXiv: 2111.09880 (cit. on p. 34).

[27] *NVIDIA Docs, NVIDIA Modulus Sym (Latest Release) Advanced Schemes and Tools*. Accessed: 2024-12-18 (cit. on p. 34).

[28] N. Rahaman; A. Baratin; D. Arpit; F. Draxler; M. Lin; F. A. Hamprecht; Y. Bengio; A. Courville. “On the spectral bias of neural networks”. *Proceedings of the 36th International Conference on Machine Learning*. Ed. by K. Chaudhuri; R. Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2018, pp. 5301–5310. arXiv: 1806.08734 (cit. on p. 36).

[29] M. Raissi; P. Perdikaris; 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 (2019), pp. 686–707. DOI: 10.1016/j.jcp.2018.10.045 (cit. on p. 31).

[30] D. E. Rumelhart; G. E. Hinton; R. J. Williams. “Learning representations by back-propagating errors”. *Nature* 323.6088 (1986), pp. 533–536. DOI: 10.1038/323533a0 (cit. on p. 31).

[31] N. Sukumar; A. Srivastava. “Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks”. *Computer Methods in Applied Mechanics and Engineering* 389 (2022), p. 114333. DOI: 10.1016/j.

cma . 2021 . 114333. arXiv: 2104 . 08426 (cit. on p. 37).

[32] F. Tröltzsch. *Optimal Control of Partial Differential Equations*. Vol. 112. Graduate Studies in Mathematics. Providence: American Mathematical Society, 2010. DOI: 10 . 1090/gsm/112 (cit. on p. 32).

[33] R. van der Meer; C. W. Oosterlee; A. Borovykh. "Optimally weighted loss functions for solving PDEs with neural networks". *Journal of Computational and Applied Mathematics* 405 (2022), p. 113887. DOI: 10 . 1016/j . cam . 2021 . 113887. arXiv: 2002 . 06269 (cit. on pp. 33 sq.).

[34] S. Wang; M. A. Bhouri; P. Perdikaris. *Fast PDE-constrained optimization via self-supervised operator learning*. 2021. arXiv: 2110 . 13297 (cit. on p. 43).

[35] S. Wang; S. Sankaran; P. Perdikaris. "Respecting causality for training physics-informed neural networks". *Computer Methods in Applied Mechanics and Engineering* 421 (2024), p. 116813. DOI: 10 . 1016/j . cma . 2024 . 116813. arXiv: 2203 . 07404 (cit. on p. 33).

[36] S. Wang; Y. Teng; P. Perdikaris. *Understanding and mitigating gradient pathologies in physics-informed neural networks*. 2020. arXiv: 2001 . 04536 (cit. on pp. 34, 37, 39).

[37] S. Wang; X. Yu; P. Perdikaris. "When and why PINNs fail to train: a neural tangent kernel perspective". *Journal of Computational Physics* 449 (2022), p. 110768. DOI: 10 . 1016/j . jcp . 2021 . 110768. arXiv: 2007 . 14527 (cit. on pp. 34 sqq.).

[38] C. L. Wight; J. Zhao. "Solving Allen-Cahn and Cahn-Hilliard equations using the adaptive physics informed neural networks". *Communications in Computational Physics* 29.3 (2021), pp. 930–954. DOI: 10 . 4208/cicp . oa - 2020 - 0086. arXiv: 2007 . 04542 (cit. on p. 33).

[39] P. Yin; G. Xiao; K. Tang; C. Yang. "AONN: an adjoint-oriented neural network method for all-at-once solutions of parametric optimal control problems". *SIAM Journal on Scientific Computing* 46.1 (2024), pp. C127–C153. DOI: 10 . 1137/22m154209x. arXiv: 2302 . 02076 (cit. on pp. 33, 37, 39).