Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

INTERACTIVE GEOMETRY LAB

Physics-Informed Neural Networks for Structural Concrete Engineering

Master's Thesis
Rafael Bischof

Supervision

Dr. Michael Kraus
Prof. Dr. Olga Sorkine-Hornung
Prof. Dr. Walter Kaufmann

September 2021

Contents

Summary

Neural Networks are universal function approximators [21] and have shown impressive results on a variety of different tasks, especially when problems involve unstructured data and an abundant amount of training samples [1][28][30][31]. However, collecting a dataset of sufficient quality and size in order to train a reliable Neural Network can be prohibitively expensive or even impossible in some fields.

Physics-Informed Neural Networks leverage well known physical laws, generally in the form of partial differential equations (PDE), by including them into their loss function [34]. This prior knowledge guides and facilitates the training process, enabling the model to quickly approximate the right solution, even when little to no training examples are available. Their continuity, flexibility and the perspective of solving arbitrary instances of a given PDE in a single forward pass, might turn them into a valuable alternative to established, mesh-based numerical techniques like the finite elements method [2][23].

However, PINNs are currently too inefficient and unreliable to be seriously taken into consideration in practice. The stochasticity arising from the random weights initialisation in combination with their reliance on gradient descent means that PINNs can produce unsatisfactory solutions due to local minima, imbalanced losses or insufficient modelling power in the architecture.

This thesis is concerned with making PINNs more reliable and studying the effectiveness of various state-of-the-art extensions to vanilla PINNs on the Kirchhoff plate bending problem. In a first step, we address the issue of gradient pathologies arising because of terms with different units of measurements and thus of varying magnitudes in the loss function [44]. To this end, we compare an Adaptive Loss Balancing algorithm proposed in the context of PINNs - Learning Rate Annealing [44] - to two Multi-Objective Balancing techniques originating from Computer Vision - GradNorm [9] and SoftAdapt [20]. Furthermore, we propose our own adaptation of the aforementioned methods, Relative Loss Balancing, that we found to be particularly effective at solving Kirchhoff's equation. To assess their generalisability, we test the four algorithms on three benchmark problems for PINNs: the Helmholtz, Burgers and Kirchhoff PDEs. The results produced by automated scaling heuristics consistently outperform a manually tuned baseline. Additionally, we show that the scalings generated by Relative Loss Balancing provide valuable insight into the training process and can be used to take informed decisions in order to improve the framework.

In a second step, we conduct an extensive parameter study on Kirchhoff's plate bending equation, where we test several network architectures, activation functions, sampling techniques and loss balancing algorithms in order to consolidate knowledge about this problem. We analyse the results in perspective with important properties from structural engineering, highlight hyperparameters and extensions which consistently improve the model's performance and identify a number of trade-offs that need to be addressed on a per-task basis. On many problems, our frameworks are able to achieve promising results, often exceeding the threshold of minimum accuracy for use in practice. However, we also identify important limitations and failure modes that need to be addressed before PINNs can be used in real-world applications.

PINNs are currently most often used as so-called representation networks, meaning that they represent specific PDEs with given boundary- and initial conditions. This is a major limiting factor, as any slight change of those conditions requires the network to be retrained from scratch, making the use of PINNs computationally expensive. In a last step of this thesis, we therefore propose the use of hypernetworks [18] in order to solve this issue. Hypernetworks are used to translate an

embedding of context variables to weights and biases of another, main network. In the context of PDEs, such a hypernetwork could receive the boundary conditions and other parameters and parameterise the function by generating a corresponding PINN. We demonstrate on Helmholtz' and Kirchhoff's equations, that hypernetworks can be used to solve these tasks and indeed produce qualitatively high results in a single forward pass, therefore turning PINNs extremely valuable for fast prototyping in many practical applications.

Abstract

This thesis is concerned with making Physics-Informed Neural Networks (PINN) more reliable and studying the effectiveness of various state-of-the-art extensions to vanilla PINNs on the Kirchhoff plate bending problem. In a first step, we address the issue of gradient pathologies arising because of terms with different units of measurements and thus of varying magnitudes in the loss function. To this end, we compare an Adaptive Loss Balancing algorithm proposed in the context of PINNs - Learning Rate Annealing - to two Multi-Objective Balancing techniques originating from Computer Vision - GradNorm and SoftAdapt. Furthermore, we propose our own adaptation of the aforementioned methods, Relative Loss Balancing, that we found to be particularly effective at solving Kirchhoff's equation. To assess their generalisability, we test the four algorithms on three benchmark problems for PINNs: the Helmholtz, Burgers and Kirchhoff PDEs.

In a second step, we conduct an extensive parameter study on Kirchhoff's plate bending equation, where we test several network architectures, activation functions, sampling techniques and loss balancing algorithms in order to consolidate knowledge about this problem. We analyse the results in perspective with important properties from structural engineering, highlight hyperparameters and extensions which consistently improve the model's performance and identify a number of trade-offs that need to be addressed on a per-task basis. On many problems, our frameworks are able to achieve promising results, often exceeding the threshold of minimum accuracy for use in practice. However, we also identify important limitations and failure modes that need to be addressed before PINNs can be used in real-world applications.

Finally, we propose to use hypernetworks in order to parameterise PDEs with any parameters or constraints by generating the weights and biases of the corresponding PINNs. This framework is able to compute predictions in a single forward pass, therefore turning PINNs extremely valuable for fast prototyping in many practical applications.

Introduction

1.1 Neural Networks

Neural Networks are inspired by the nervous system in human brains and their building blocks are called neurons. Neurons get as input a set of signals $x \in \mathbb{R}^N$ scaled by weights $w \in \mathbb{R}^N$ as well as shifted by a bias $b \in \mathbb{R}^1$. The signals are aggregated, typically by taking the sum, and passed through a non-linear function σ.

$$\hat{y} = \sigma\left(\sum_{i=1}^{N} w_i x_i + b\right), \quad \hat{y} \in \mathbb{R}^1 \tag{1.1}$$

In Machine Learning, this algorithm is known as Perceptron [35]. Neural Networks consist of an arbitrary number of Perceptrons wired in parallel as well as in sequence, therefore earning them the name of Multi-Layered Perceptrons (MLP). In vector notation, one can write a two layered network with weights and biases of layer one $W^{(1)} \in \mathbb{R}^{M\times N}$, $b^{(1)} \in \mathbb{R}^M$ and layer two $W^{(2)} \in \mathbb{R}^{1\times M}$, $b^{(2)} \in \mathbb{R}^1$ with the following equation:

$$\hat{y} = W^{(2)}\sigma\left(W^{(1)}x + b^{(1)}\right) + b^{(2)} \tag{1.2}$$

Weights and biases are tipically represented by θ and a network F parameterised by $F(x;\theta)$.

The performance of a Neural Network is measured by a loss function $\mathcal{L}$ comparing the predicted output $\hat{y}$ to the real solution y from a batch of samples $\{(x_1, y_1), \dots, (x_b, y_b)\}$. A common choice of loss function is the Mean Squared Error:

$$\mathcal{L}(\hat{y}, y) = \frac{1}{b}\sum_{i=1}^{b}\|\hat{y} - y\|_2^2 \tag{1.3}$$

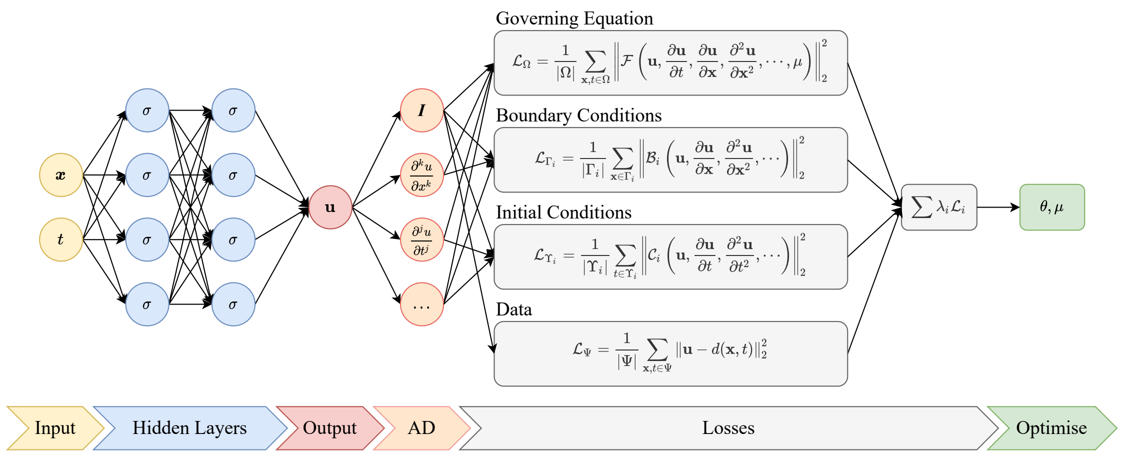
The network's weights are updated using gradient descent. Therefore, all non-linearities used within the network, as well as the loss function, must be differentiable.

$$\theta^{(t)} = \theta^{(t-1)} - \eta\nabla_\theta\mathcal{L}(F(x;\theta^{(t-1)}), y) \tag{1.4}$$

Neural Networks are universal function approximators [21] and have shown impressive results on a variety of different tasks, especially when problems involve unstructured data and an abundant amount of training samples [1][28][30][31]. However, collecting a dataset of sufficient quality and size in order to train a reliable Neural Network can be prohibitively expensive or even impossible in some fields. For example, Machine Learning has seen many difficulties setting foot in Civil Engineering due to the great costs of acquiring data involving physical structures.

1.2 Physics-Informed Neural Networks

Physics-Informed Neural Networks (PINN) provide a way of incorporating prior knowledge into the optimisation in order to facilitate and guide the training process. They leverage well-established


Governing Equation
$\mathcal{L}_\Omega = \frac{1}{|\Omega|}\sum_{\mathbf{x},t\in\Omega}\left\|\mathcal{F}\left(\mathbf{u},\frac{\partial\mathbf{u}}{\partial t},\frac{\partial\mathbf{u}}{\partial\mathbf{x}},\frac{\partial^2\mathbf{u}}{\partial\mathbf{x}^2},\cdots,\mu\right)\right\|_2^2$
Boundary Conditions
$\mathcal{L}_{\Gamma_i} = \frac{1}{|\Gamma_i|}\sum_{\mathbf{x}\in\Gamma_i}\left\|\mathcal{B}_i\left(\mathbf{u},\frac{\partial\mathbf{u}}{\partial\mathbf{x}},\frac{\partial^2\mathbf{u}}{\partial\mathbf{x}^2},\cdots\right)\right\|_2^2$
Initial Conditions
$\mathcal{L}_{\Upsilon_i} = \frac{1}{|\Upsilon_i|}\sum_{t\in\Upsilon_i}\left\|\mathcal{C}_i\left(\mathbf{u},\frac{\partial\mathbf{u}}{\partial t},\frac{\partial^2\mathbf{u}}{\partial t^2},\cdots\right)\right\|_2^2$
Data
$\mathcal{L}_\Psi = \frac{1}{|\Psi|}\sum_{\mathbf{x},t\in\Psi}\|\mathbf{u}-d(\mathbf{x},t)\|_2^2$
$\boldsymbol{x}$
t
σ
$\mathbf{u}$
$\boldsymbol{I}$
$\frac{\partial^k u}{\partial x^k}$
$\frac{\partial^j u}{\partial t^j}$
$\dots$
$\sum \lambda_i \mathcal{L}_i$
θ, μ
Input
Hidden Layers
Output
AD
Losses
Optimise

Note: θ: weights/biases of network; μ: unknown PDE parameters; λ_i: scaling factor for loss term i

Figure 1.1: Schematic of a Physics-Informed Neural Network (PINN): A fully-connected feed-forward neural network with space and time coordinates $(\mathbf{x}, t)$ as inputs, approximating a solution $\hat{\mathbf{u}}(\mathbf{x}, t)$. Derivatives of $\mathbf{u}$ w.r.t. inputs are computed by automatic differentiation (AD) and then incorporated into residuals of the governing equations as the loss function, which is composed of multiple terms weighted by different coefficients. Parameters of the FCNN θ and the unknown PDE parameters μ may be optimised simultaneously by minimising the loss function.

physical laws by including them into the loss functions [34]. These physical laws generally come in the form of a general partial differential equation (PDE):

$$\begin{aligned} &\mathcal{N}_x[u; k] - f(x) = 0, \;\; x \in \Omega \\ &u(x) = g(x), \;\; x \in \partial\Omega \end{aligned} \tag{1.5}$$

where $\mathcal{N}_x[\cdot]$ is a nonlinear differential operator parameterized by coefficient k, f is an arbitrary (e.g., forcing) function, g is the boundary condition, Ω and $\partial\Omega$ are subsets of $\mathbb{R}^d$ representing the domain and its boundary.

The function u(x) is approximated by a neural network $U(x, \theta)$ and its loss defined as follows:

$$\begin{aligned} \mathcal{L}_\Omega(\theta) &= \frac{1}{|\Omega|}\sum_{x\in\Omega}\|\mathcal{N}_x[U(x,\theta); k] - f(x)\|_2^2 \\ \mathcal{L}_{\partial\Omega}(\theta) &= \frac{1}{|\partial\Omega|}\sum_{x\in\partial\Omega}\|U(x,\theta) - g(x)\|_2^2 \end{aligned} \tag{1.6}$$

where Ω and $\partial\Omega$ are finite sets of collocation points on the domain and the boundary respectively. Note that including U into $\mathcal{N}_x$ might require multiple higher order derivatives of the network (Sobolev-Training), which is possible through automatic differentiation and the use of appropriate activation functions [11][16].

PINNs are currently predominantly used as so-called representation networks, meaning that they parameterise specific PDEs with given boundary- and initial conditions. However, any slight change of those conditions requires the neural network to be retrained from scratch, making the use of PINNs computationally expensive and raising concerns as to their effectiveness compared to established, numerical techniques like the finite elements method (FEM). Recent advances showed the potential of models incorporating PINNs into their pipelines and thereby solving PDEs in a single forward pass, which would make them extremely valuable for fast prototyping in many practical applications [38].

However, until the point where PINNs find their place in practice, further research remains to be conducted in order to identify and solve their current failure modes, one of which is the imbalance between the two objectives in Equation 1.6, which are generally trained jointly.

$$\mathcal{L}(\theta) = (\mathcal{L}_{\Omega}(\theta), \mathcal{L}_{\partial\Omega}(\theta))^T \tag{1.7}$$

Since PINNs can be simultaneously trained on multiple PDEs, or the boundary conditions be composed of several different terms, Equation 1.7 often contains considerably more than two objectives. All of these objectives can have different units of measurements and therefore also varying magnitudes, manifesting as imbalanced losses and consequently also as imbalances between backpropagated gradients. The resulting pathologies were shown to impede proper training and often result in unsatisfactory solutions [44]. It is therefore paramount to find a way of balancing the contributions of the various terms during optimisation.

1.3 Multi-Objective Optimisation

Multi-Objective Optimisation (MOO) is concerned with simultaneously optimising a set of $m > 1$, potentially conflicting objectives [7].

$$\mathcal{L}(\theta) = (\mathcal{L}_1(\theta), \dots, \mathcal{L}_m(\theta))^T \tag{1.8}$$

Many problems in engineering or economics can be formulated as multi-objective optimisations and generally require trade-offs, meaning that there's no solution that losslessly satisfies all objectives [8]. Therefore, we're generally interested in finding so-called Pareto optimal solutions according to the following definitions [37]:

Definition 1. *A solution $\hat{\theta} \in \Omega$ Pareto dominates solution θ (denoted $\hat{\theta} \prec \theta$) if and only if $\mathcal{L}_i(\hat{\theta}) \le \mathcal{L}_i(\theta), \forall i \in \{1, \dots, m\}$ and $\exists j \in \{1, \dots, m\}$ such that $\mathcal{L}_j(\hat{\theta}) < \mathcal{L}_j(\theta)$.*

Definition 2. *A solution $\hat{\theta} \in \Omega$ is said to be Pareto optimal if $\forall \theta \in \Omega, \hat{\theta} \preceq \theta$. The set of all Pareto optimal points is called the Pareto set and the image of the Pareto set in the loss space is called the Pareto front.*

A multi-objective optimisation can be turned into a single objective through linear scalarisation:

$$\mathcal{L}(\theta) = \sum_{i=1}^{m} \lambda_i \mathcal{L}_i(\theta), \quad \lambda_i \in \mathbb{R}_{>0} \tag{1.9}$$

In theory, a Pareto optimal solution θ is independent of the scalarisation [36]. However, when using neural networks for MOO, the solution space becomes highly non-convex. Thus, although neural networks are universal function approximators [21], we're not guaranteed to find the optimal solution through first-order gradient-based optimisation. Scaling the loss space therefore gives us the option of guiding the gradients into having an a priori deemed desirable property. However, manually finding optimal λ requires laborious grid search and becomes intractable as m gets large. Furthermore, one might want to let λ evolve over time. This raises the need for an automated heuristic to dynamically choose the scalings λ.

Adaptive Loss Balancing

In a first step, we investigate different methods aiming at balancing the various terms in a Multi-Objective Optimisation. To this end, we compare the effectiveness of Learning Rate Annealing [44], proposed in the context of PINNs, to two approaches originating from Computer Vision applications: GradNorm [9] and SoftAdapt [20]. Furthermore, we present our own variation of these techniques that we found to be effective and test the algorithms on three benchmark problems for PINNs: the Helmholtz, Burger and Kirchhoff PDEs.

2.1 Related Work

2.1.1 Learning Rate Annealing

Wang et al. conducted a study on gradients in PINNs and identified pathologies that explained some failure modes [44]. One pathology is gradient stiffness in the boundary conditions caused by the imbalance amongst the different loss terms. As a remedy, they propose to adaptively scale the loss using gradient statistics.

$$\begin{aligned}
\hat{\lambda}_i^{(t)} &= \frac{\max\{|\nabla_\theta \mathcal{L}_\Omega^{(t)}|\}}{\overline{|\nabla_\theta \mathcal{L}_{\partial\Omega_i}^{(t)}|}},\ i \in \{1, \dots, m\} \\
\lambda_i^{(t)} &= \alpha \lambda_i^{(t-1)} + (1-\alpha)\hat{\lambda}_i^{(t-1)} \\
\theta^{(t+1)} &= \theta^{(t)} - \eta \nabla_\theta \mathcal{L}_\Omega + \eta \sum_{i=1}^{m} \lambda_i^{(t)} \nabla_\theta \mathcal{L}_{\partial\Omega_i}^{(t)}
\end{aligned} \tag{2.1}$$

where $\overline{|\nabla_\theta \mathcal{L}_{\Gamma_i}^{(t)}|}$ is the mean of the gradient w.r.t. the parameters θ and a hyperparameter α with a value $\alpha = 0.9$ recommended by the authors.

Whenever the maximum value of $|\nabla_\theta \mathcal{L}_\Omega^{(t)}|$ grows considerably larger than the average value in $|\nabla_\theta \mathcal{L}_{\partial\Omega_i}^{(t)}|$, the scalings $\hat{\lambda}^{(t)}$ correct for this discrepancy such that all gradients have similar magnitudes. The authors also employ exponential decay to smoothen the balancing and avoid drastic changes of the loss space between optimisation steps.

This procedure induces a few issues. Its unboundedness means it can up- or downscale terms by several orders of magnitude. The upscaling in particular can cause problems similar to the effect of choosing too large of a learning rate, potentially causing the network to overshoot the objective. Furthermore, scaling all terms to have the same magnitude throughout training can incite the network to optimize for the "low hanging fruit". A term, whose loss decreased considerably in the last optimisation step, will see its contribution to the total gradient scaled back up to the same magnitude to match the other terms. Therefore, the network might focus on the objectives that are easiest to optimize for.

2.1.2 GradNorm

Chen et al. take a different approach and make the scalings λ trainable [9]. The updates on these trainable scalings are chosen such that all terms improve at the same relative rate w.r.t. their initial loss and are performed by a separate optimizer. A term that improved at a higher rate since the beginning of training compared to the other terms, gets a weaker scaling until all terms have made the same percentual progress. Therefore, one could argue that they weakly enforce each optimisation-step to Pareto dominate (Defninition 1) its predecessor.

The loss for updating the scalings is calculated the following way:

$$\mathcal{L}(t;\lambda) = \sum_{i=1}^{m} \left| G_\theta^{(i)}(t) - \overline{G}_\theta(t) \times [r_i(t)]^\alpha \right|_1 \tag{2.2}$$

where $G_\theta^{(i)}(t) = \|\nabla_\theta \lambda_i \mathcal{L}_i(t)\|_2$ is the L_2 norm of the gradient w.r.t. the network's parameters θ for the scaled loss of objective $i \in \{1, \dots, m\}$, $\overline{G}_\theta(t) = \frac{1}{m}\sum_{i=1}^{m} G_\theta^{(i)}(t)$ is the average of all gradient norms, $r_i(t) = \mathcal{L}_i(t)/(\mathcal{L}_i(0) \cdot \overline{\mathcal{L}}(t))$ defines the rate at which term i improved so far and α is a hyperparameter representing the strength of the restoring force which pulls tasks back to a common training rate. Note that $\overline{G}_\theta(t) \times [r_i(t)]^\alpha$ is the desirable value that $G_\theta^{(i)}(t)$ should take on, so it is kept constant such that no can gradients flow through it.

The final loss for updating the network's parameters is then simply a linear scalarisation with the scalings that were previously updated:

$$\mathcal{L}(t;\theta) = \sum_{i=1}^{m} \lambda_i(t)\mathcal{L}_i(t) \tag{2.3}$$

This algorithm is fairly evolved and, despite solving some issues that learning rate annealing has, it still requires a separate backward-pass for each objective, which becomes prohibitively expensive as m gets large. Furthermore, it relies on two separate optimisation rounds at each step: one for adapting the scalings λ and another for updating the weights θ. To get back to our definition in Equation 1.8, we can formulate GradNorm as a MOO:

$$\mathcal{L}(t) = (\mathcal{L}(t;\theta), \mathcal{L}(t;\lambda))^T \tag{2.4}$$

which in turn requires empirical hyperparameter search (learning rate, initialisation, etc.) to keep the system balanced - exactly the problem we are actually seeking to solve by using adaptive loss balancing.

2.1.3 SoftAdapt

Similar to GradNorm, SoftAdapt leverages the ansatz of relative progress in order to balance the loss terms. However, the authors relax it by only considering the previous time-step $\mathcal{L}_i(t-1)$. The scalings are then normalised by using a softmax function:

$$\lambda_i^{(t)} = \frac{exp\left(\frac{\mathcal{L}_i^{(t)}}{\mathcal{L}_i^{(t-1)}}\right)}{\sum_{j=1}^{m} exp\left(\frac{\mathcal{L}_j^{(t)}}{\mathcal{L}_j^{(t-1)}}\right)}, \quad i \in \{1, \dots, m\} \tag{2.5}$$

where $\mathcal{L}_i^(t)$ is the loss of term i at optimisation step t.

SoftAdapt also differs from GradNorm in that it does not require gradient statistics, thus eliminating the need of performing separate backward passes for each objective. Instead, it makes use of the fact that magnitudes in the gradients directly depend on the magnitudes of the terms in the loss function and therefore aims at achieving the balance solely through loss statistics. Of course, this is only true if the same loss function is used for every objective (e.g. the L_2 loss). However, this setting generalises to a vast majority of applications for PINNs.

2.1.4 Relative Loss Balancing with random lookback

Drawing inspiration from all these techniques, we propose our own implementation for balancing the various objectives in the loss function:

- We employ SoftAdapt's balancing method using the rate of change between consecutive training steps and normalizing them through a softmax function.

- Similar to learning rate annealing, we update the scalings using an exponential decay in order to utilise loss statistics from more than just one training step in the past. This has the further benefit of smoothening the training progress and therefore allowing the network to make faster progress.

- In addition, we introduce a random lookback into the exponential decay, where a Bernoulli random variable ρ decides whether to use the previous steps' loss statistics to calculate the scalings, or whether to look all the way back until the start of training $\mathcal{L}_i^{(0)}$.

$$\begin{aligned}
\hat{\lambda}_i^{(t;t')} &= m \cdot \frac{exp\left(\frac{\mathcal{L}_i^{(t)}}{\mathcal{T}\mathcal{L}_i^{(t')}}\right)}{\sum_{j=1}^{m} exp\left(\frac{\mathcal{L}_j^{(t)}}{\mathcal{T}\mathcal{L}_j^{(t')}}\right)}, \quad i \in \{1, \dots, m\} \\
\lambda_i^{(t)} &= \alpha(\rho\lambda_i^{(t-1)} + (1-\rho)\hat{\lambda}_i^{(t;0)}) + (1-\alpha)\hat{\lambda}_i^{(t;t-1)}
\end{aligned} \tag{2.6}$$

where ρ is a Bernoulli random variable with $\mathbb{E}[\rho]$ close to 1.

This method is an attempt at combining the best attributes of the aforementioned approaches into a new heuristic. First and foremost, it still weakly enforces every training step to Pareto dominate its predecessor, which is an important property in physical applications. It also avoids using gradient statistics, making it considerably more efficient than learning rate annealing and GradNorm. Furthermore, it reduces drastic changes in the loss space by using exponential decay and can easily be adapted to use more or fewer information of past optimisation steps by tuning the parameter α. One can think of α as being the model's ability to remember the past, with a high alpha giving lots of weight to past loss statistics, while a lower alpha increases stochasticity. Setting $\alpha = 1$ results in each term's relative progress being calculated w.r.t. the initial loss $\mathcal{L}_i^{(0)}$. However, we found this to be too restrictive, since it causes the model to stop making progress as soon as one term reaches a local minimum. We chose values between 0.9 and 0.999 and report the effects of varying this hyperparameter in Section 2.3.

Choosing the value of α also requires to make a trade-off: a high value means the model will remember potential deteriorations of certain terms for longer and therefore leave a longer time-frame in order to compensate them. However, it also induces a latency between a term starting to deteriorate and the scalings λ reacting accordingly. We therefore study the effect of introducing the Bernoulli random variable ρ that causes the model to occasionally look back until the start of training. This allows to set a lower value for α, thus making the model more flexible while still "reminding" it of the progress made since the start of training. Furthermore, the random lookback can give occasional new impulses and let the model escape local minima by changing the loss space, as well as inciting it to explore more of the parameter space. In case the impulse would turn out to have a negative effect on the accuracy, one can still choose to roll back and reset to the previous settings.

The last hyperparameter is the so-called temperature $\mathcal{T}$. Setting $\mathcal{T} \to \infty$ recalibrates the softmax in order to output uniform values and thus all $\hat{\lambda}_i^{(t)} = 1$. On the other hand, $\mathcal{T} \to 0$ essentially turns the softmax into an argmax function, with the scaling $\hat{\lambda}_i^{(t)} = m$ resulting for the term with the lowest relative progress and $\hat{\lambda}_i^{(t)} = 0$ for all others.

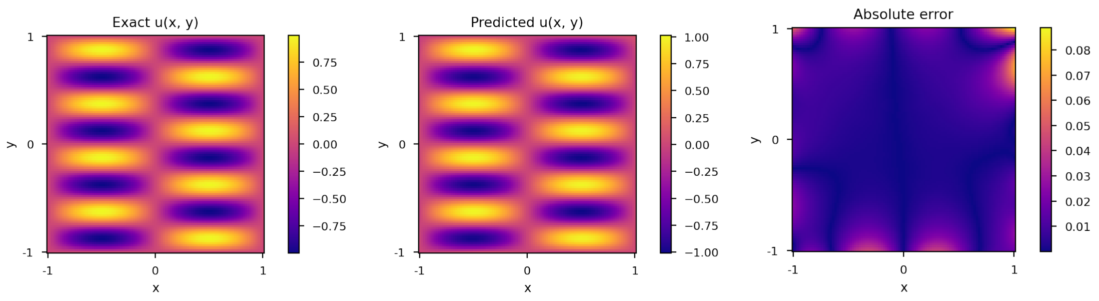


Figure 2.1: Helmholtz equation predicted with a fully-connected network consisting of two layers and 128 nodes each.

2.2 Experiments

2.2.1 Helmholtz equation

The Helmholtz equation represents a time-independent form of the wave equation and arises in many physical problems like acoustics and electromagnetism [40].

$$\Delta u + k^2(u) = f(x, y), \quad (x, y) \in [-1, 1]^2 \tag{2.7}$$

This represents a common problem to benchmark PINNs and has an analytical solution in combination with Dirichlet boundaries:

$$\begin{aligned} &f(x, y) = (-\pi^2 - (4\pi)^2 + k^2) sin(\pi x) sin(4\pi y) \\ &u(x, y) = sin(\pi x) sin(4\pi y) \\ &u(-1, y) = u(1, y) = u(x, -1) = u(x, 1) = 0 \end{aligned} \tag{2.8}$$

when $k(u) = 1$.

Both the x and y input variables are bounded below by -1 and bounded above by 1. Therefore, the boundary conditions add four terms to the loss function:

$$\mathcal{L}^{(t)} = \lambda_0 \sum_{(x,y)\in\Omega} \|U(x, y; \theta) - f(x, y)\|_2^2 + \sum_{i=1}^{4} \lambda_i \sum_{(x,y)\in\partial\Omega_i} \|U(x, y; \theta)\|_2^2 \tag{2.9}$$

2.2.2 Burgers' equation

Burgers' equation is a one-dimensional Navier-Stokes equation and is used i.a. to model shock waves, gas dynamics or traffic flow [4]. Again using Dirichlet boundary conditions, the PDE takes the following form:

$$\begin{aligned} &\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu \frac{\partial^2 u}{\partial x^2} = 0, \quad x \in [-1, 1], \quad t \in [0, 1] \\ &u(0, x) = -sin(\pi x) \\ &u(t, -1) = u(t, 1) = 0 \end{aligned} \tag{2.10}$$

where we choose $\nu = \frac{1}{100\pi}$.

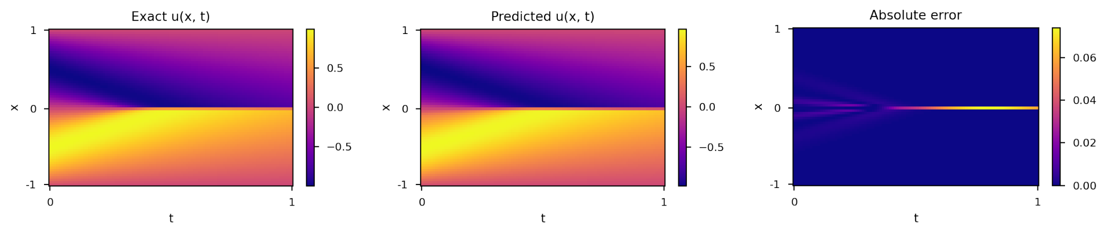


Figure 2.2: Burgers' equation predicted with a fully-connected network consisting of two layers and 128 nodes each. The exact solution was calculated using the finite elements method.

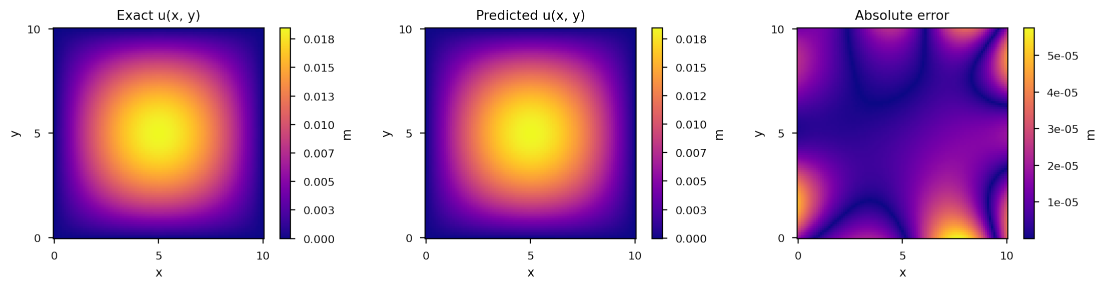


Figure 2.3: Kirchhoff equation predicted with a fully-connected network consisting of three layers and 128 nodes each.

2.2.3 Kirchhoff equation

The Kirchhoff bending problem is a classical fourth-order problem and its mechanical behaviour described by a fourth-order partial differential equation:

$$\begin{aligned} &\nabla^4 u(x,y) - \frac{p(x,y)}{D} = 0, \quad (x,y) \in \mathbb{R}^2_{>0} \\ &D = \frac{Eh^3}{12(1-\nu^2)} \end{aligned} \tag{2.11}$$

where $p(x,y)$ is the load acting on the plate at coordinates (x,y), D is the plate's flexural stiffness calculated with Young's modulus E, the plate's thickness h and Poisson's ratio ν. One can infer an analytical solution by applying a sinusoidal load:

$$\begin{aligned} &p(x,y) = p_0 \sin\left(\frac{x\pi}{a}\right)\sin\left(\frac{y\pi}{b}\right) \\ &u(x,y) = \frac{p_0}{\pi^4 D(\frac{1}{a^2}+\frac{1}{b^2})^2}\sin\left(\frac{x\pi}{a}\right)\sin\left(\frac{y\pi}{b}\right) \end{aligned} \tag{2.12}$$

where we set plate width $a = 10$, plate height $b = 10$, base load $p_0 = 0.015$, $E = 30'000$, $h = 0.2$ and $\nu = 0.2$ and once again choose to use the simply supported edge boundary conditions.

$$\begin{aligned} &u(0,y) = u(a,y) = u(x,0) = u(x,b) = 0 \\ &m_x(0,y) = m_x(a,y) = m_y(x,0) = m_y(x,b) = 0 \end{aligned} \tag{2.13}$$

where m_x and m_y are moments calculated as $m_x = -D\left(\frac{\partial^2 u}{\partial x^2} + \nu\frac{\partial^2 u}{\partial y^2}\right)$ and $m_y = -D\left(\nu\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)$.

In total, there are 12 boundary conditions and therefore 13 terms in the loss function, making this a challenging task for balancing the contributions of the various objectives.

2.3 Results

We tested the different balancing techniques on fully-connected networks with *tanh* activations and varying depth and width. Training was limited to 10^5 steps of gradient descent using the Adam optimiser [26] and an initial learning rate of 0.001. Additionally, we reduced the learning rate by a multiplicative factor of 0.1 whenever the optimisation stopped making progress for over 3'000 optimisation steps and finally used early stopping in case of 9'000 steps without improvement.

Figure 2.4 shows the scaling factors λ of our Relative Loss Balancing method with varying hyperparameters. As can be expected, a larger value for α leads to a smoother curve because past loss statistics are dragged on longer, thus countering drastic changes that might arise at every optimisation step. On the other hand, the temperature $\mathcal{T}$ influences the magnitude of the scalings.

The relatively small variances across training runs suggest that the optimisation progress follows similar patterns, even when varying depth and width of the network. Therefore, these values can provide valuable insight into the training and help identifying possibilities of improving the model. F.ex. the fact that the scaling for the governing equation has the largest value after 50,000 epochs, indicates that it was the first term to stop making progress. On the other hand, Figure 2.8 shows that the opposite is true for Helmholtz' and Kirchhoff's equations, where the boundary conditions have more difficulties making progress. This knowledge can help taking informed decisions to improve the framework, e.g. by adapting the activation functions, the loss function or the model's architecture accordingly.

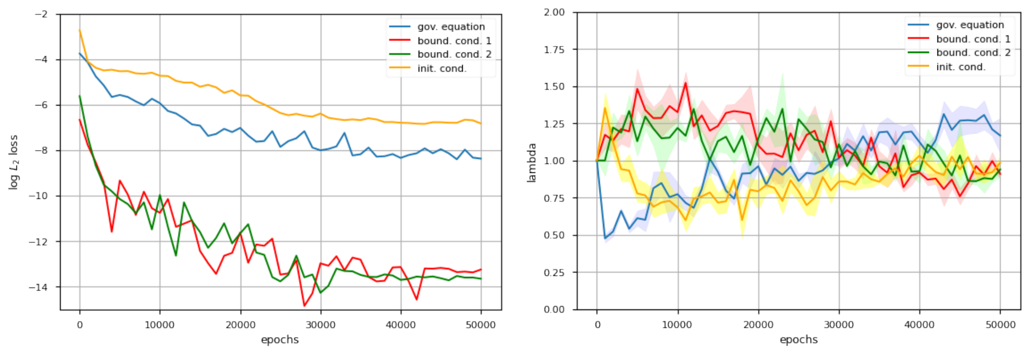
Table 2.1 shows the performances of the different balancing techniques against a baseline where we manually chose the optimal scalings λ through grid search. As can be observed, the adaptive scaling techniques consistently outperform the baseline, with only a few exceptions. By choosing either one of these methods, one can therefore greatly reduce the amount of work required for hyperparameter search while still achieving good results with high probability.

PDE		Manual	GradNorm	LR anneal.	SoftAdapt	Relative
Helmholtz	train f	$1.4\cdot10^{-2}$	$7.1\cdot10^{-2}$	$2.7\cdot10^{-1}$	$4.9\cdot10^{-1}$	$4.7\cdot10^{-3}$
	val u	$7.1\cdot10^{-2}$	$5.6\cdot10^{-6}$	$1.4\cdot10^{-5}$	$8.4\cdot10^{-4}$	$1.4\cdot10^{-4}$
	val std u	$8.1\cdot10^{-3}$	$1.9\cdot10^{-5}$	$7.6\cdot10^{-5}$	$7.3\cdot10^{-3}$	$8.2\cdot10^{-4}$
Burgers	train f	$5.5\cdot10^{-4}$	$6.6\cdot10^{-4}$	$9.9\cdot10^{-4}$	$4.5\cdot10^{-4}$	$1.7\cdot10^{-4}$
	val u	$1.2\cdot10^{-3}$	$2.0\cdot10^{-3}$	$1.6\cdot10^{-4}$	$1.8\cdot10^{-3}$	$5.4\cdot10^{-4}$
	std val u	$5.7\cdot10^{-4}$	$2.0\cdot10^{-2}$	$2.3\cdot10^{-4}$	$3.2\cdot10^{-2}$	$1.4\cdot10^{-3}$
Kirchhoff	train f	$1.2\cdot10^{-8}$	$5.3\cdot10^{-7}$	$9.1\cdot10^{-9}$	$2.0\cdot10^{-8}$	$6.0\cdot10^{-9}$
	val u	$1.3\cdot10^{-9}$	$1.7\cdot10^{-8}$	$2.7\cdot10^{-9}$	$4.2\cdot10^{-9}$	$4.7\cdot10^{-10}$
	std val u	$3.9\cdot10^{-9}$	$2.2\cdot10^{-7}$	$1.0\cdot10^{-6}$	$4.7\cdot10^{-7}$	$7.7\cdot10^{-10}$

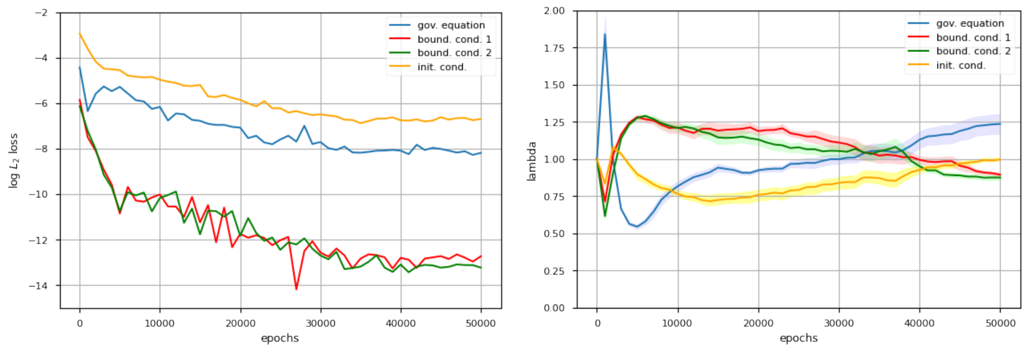
Table 2.1: Comparison of the L_2 training and validation loss, averaged over four independent runs with identical settings. Additionally, we report the standard deviation over the runs of the best performing model on the validation loss.

On Kirchhoff's equation, our proposed Relative Balancing method outperforms all other approaches in the training- and validation loss, as well as the standard deviation between independent runs. Besides its effectiveness, Relative Balancing is also more efficient. As can be inferred from Table 3.3, the computational overhead it induces is relatively low compared to a static loss function. On the other hand, the methods relying on gradient statistics need about six times more time to compute 1,000 optimisation steps. Therefore, Relative Balancing presents a viable option for problems where the number of objectives m is high.

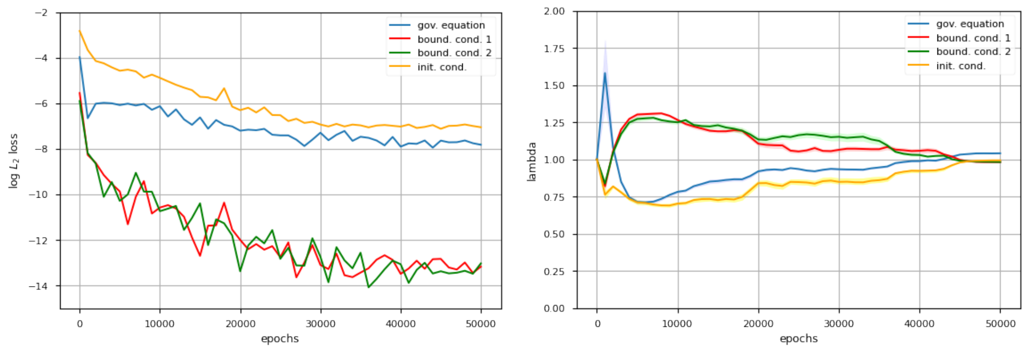
However, the Helmholtz equation shows a limitation of our approach. GradNorm and learning rate annealing both achieve impressive results and substantially outperform the baseline, as well as our Relative Balancing. This is likely due to the enormous initial difference in magnitudes between the governing equation and the boundary conditions. Furthermore, the high values of α induce a latency between the increase of a term's loss until the scaling λ reacts accordingly. Figure 2.6 shows



(a) $\alpha = 0.9$, $\mathcal{T} = 0.1$

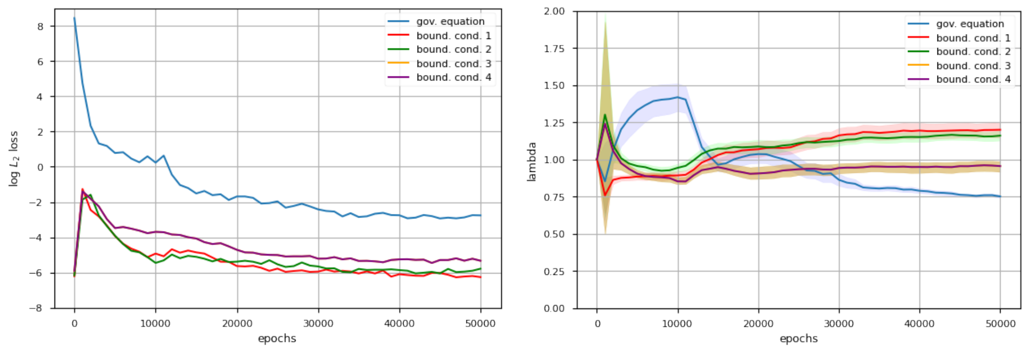


(b) $\alpha = 0.999$, $\mathcal{T} = 0.1$

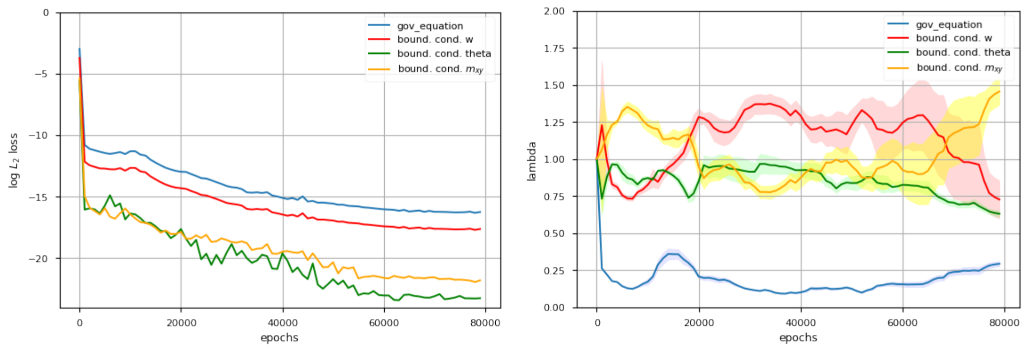


(c) $\alpha = 0.999$, $\mathcal{T} = 1$

Figure 2.4: Median of the log L_2 loss over multiple training runs (left) of Burgers' equation and the mean and variance of the corresponding scaling factors λ (right) calculated with our Relative Loss Balancing method.



(a) Helmholtz, $\alpha = 0.999$, $\mathcal{T} = 0.1$

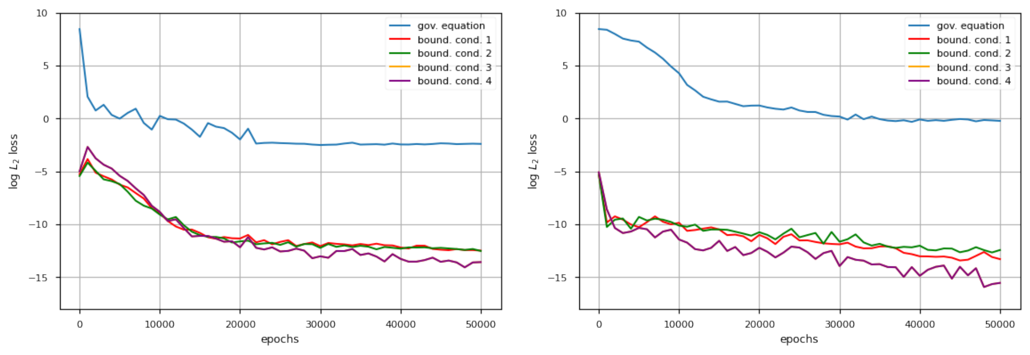


(b) Kirchhoff, $\alpha = 0.999$, $\mathcal{T} = 0.1$

Figure 2.5: Median of the log L_2 loss over multiple training runs (left) of the Helmholtz and Kirchhoff equations and the mean and variance of the corresponding scaling factors λ (right) calculated with our Relative Loss Balancing method.

method	Helmholtz	Burgers	Kirchhoff
Classic	4.8	3.7	17.3
GradNorm	10.4	14.3	128.6
LR annealing	6.7	7.2	139.7
SoftAdapt	5.0	4.1	20.2
Relative	5.2	4.3	22.5

Table 2.2: Median execution times (in s) per 1'000 optimisation steps for different balancing methods.



(a) GradNorm (b) LR annealing

Figure 2.6: Median of the log L_2 loss over multiple training runs on the Helmholtz equation.

that both GradNorm and learning rate annealing focus on improving the boundary conditions right from the beginning of training, whereas Relative Balancing counters the initial deterioration, but eventually "forgets" the neglection of the boundary conditions and instead focuses on the more dominant governing equation. This is also reflected in the discrepancy between the training and validation loss: GradNorm and learning rate annealing have a higher training loss than Relative Balancing, but still exceed at approximating the underlying function. Furthermore, Figure 2.7a shows that Relative Loss Balancing performs best on Helmholtz if the temperature parameter $\mathcal{T}$ is small, while the hyperparameter for "remembering" α has to be rather high. The small $\mathcal{T}$ causes the scalings' to sheer out, thus countering the large difference of magnitudes between the terms. On the other hand, the high α makes the model remember longer, which somewhat counters the initial neglection of the boundary conditions.

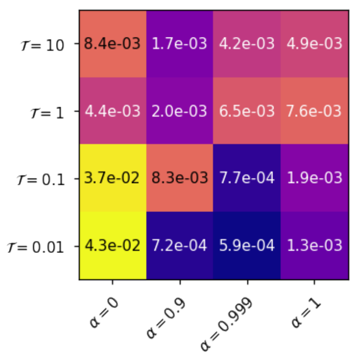
2.3.1 Relative Balancing with Random Lookback

The observations made in the previous section raise the need of extending Relative Balancing such that the identified limitation can be mitigated. As described in Section 3.6.2, we studied the effects of introducing an occasional lookback to the initial loss $\mathcal{L}_i^{(0)}$.

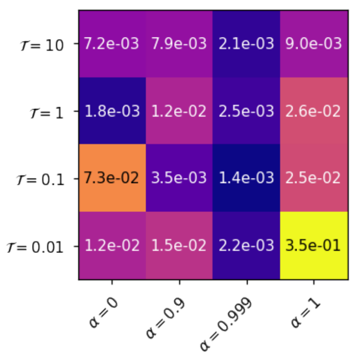
$\mathbb{E}[\rho]$	Helmholtz	Burgers	Kirchhoff
0.99	$1.4{\cdot}10^{-4}$	$2.2{\cdot}10^{-3}$	$1.1{\cdot}10^{-09}$
0.999	$1.5{\cdot}10^{-4}$	$1.1{\cdot}10^{-3}$	$9.1{\cdot}10^{-10}$
0.9999	$2.0{\cdot}10^{-4}$	$4.6{\cdot}10^{-3}$	$8.7{\cdot}10^{-10}$
1	$2.9{\cdot}10^{-4}$	$5.5{\cdot}10^{-4}$	$4.7{\cdot}10^{-10}$

Table 2.3: Validation loss when varying the expected value of ρ. Reported values are the median over three independent runs.

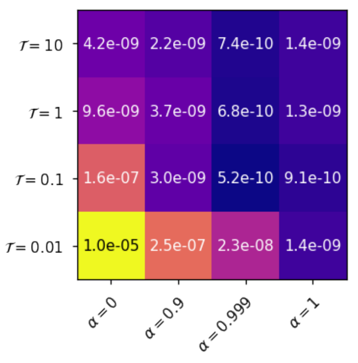
Table 2.3 shows that random lookback indeed improves the validation loss on Helmholtz'



(a) Helmholtz



(b) Burgers



(c) Kirchhoff

Figure 2.7: Ablation of the model's performance when varying $\mathcal{T}$ and α. The reported values are the median of the log L_2 loss over multiple training runs (without random lookback).

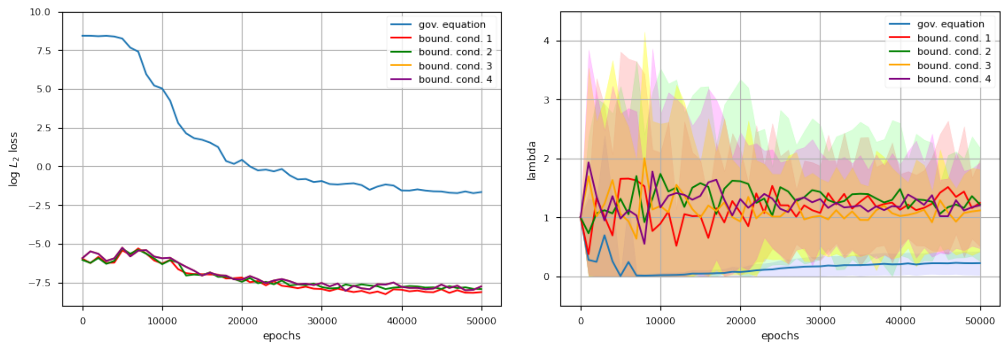


Figure 2.8: Median of the log L_2 loss over multiple training runs (left) and the mean and variance of the corresponding scaling factors λ (right) calculated with our Relative Loss Balancing method on Helmholtz, $\alpha = 0.999$, $\mathcal{T} = 0.1$, $\mathbb{E}[\rho] = 0.9999$.

equation by increasing the contribution of the boundary conditions. However, the benefits are limited to the Helmholtz equation and do not apply to Burgers' and Kirchhoff. We conclude that random lookback can be useful in certain applications. But since it also hurts performance on certain other problems, it is use must be carefully evaluated on a per-task basis.

PINN Variants for Bending Analysis of Kirchhoff Plates

In the following section, we focus on applying PINNs to the Kirchhoff plate bending problem and study the effects of a wide selection of state-of-the-art techniques proposed in the context of PINNs. More specifically, we test several network architectures, activation functions and sampling techniques, as well as two adaptive loss balancing algorithms introduced in Chapter 2: Learning Rate Annealing and Relative Loss Balancing.

3.1 Motivation

The mechanical behaviour of the Kirchhoff plate bending problem is described by a fourth-order partial differential equation. The function is $\mathcal{H}_2$ regular, i.e. globally $\mathcal{C}_1$ continuous but piecewise $\mathcal{C}_2$ continuous, which poses difficulties for existing, mesh-based numerical methods, such as the finite element method [2][23] and the boundary element method [25].

However, according to the universal approximation theorem [21], any continuous function can be approximated arbitrarily well by a feed-forward neural network with a single hidden layer, therefore offering a new possibility of approaching Kirchhoff plate bending problems.

3.2 Training method

Solving the Kirchhoff-Love plate bending problem with neural networks the classical way would require a large amount of paired data to map from forces p to the plate displacement w. However, since only very few realistic instances have an analytical solution and other approaches like the finite elements method are costly, such a dataset is hard to come by. Instead, Guo et al. proposed to use physics-informed neural networks (PINNs) [17][33]. PINNs are different from classical networks in that they are optimised on a higher-order derivative using Sobolev-Training and automatic differentiation [11][16]. For Kirchhoff-Love, instead of directly training a function U to map from coordinates x, y to the displacement w, the loss is formulated such that the fourth order derivative $\nabla^4 U((x,y),\theta)$ maps to $p(x,y)/D$ (in accordance with Eq. 2.11). By approximating this governing equation and given the appropriate boundary conditions, the network will implicitly learn to model w without ever needing a paired sample $(x,y) \rightarrow w$.

More formally, the learning objective $\mathcal{L}$ is composed of the following terms:

$$\mathcal{L}_\Omega = \frac{1}{|N_\Omega|} \sum_{(x,y)\in\Omega} \|\nabla^4 U(x,y) - \frac{p(x,y)_\Omega}{D}\|^2 \tag{3.1}$$

$$\mathcal{L}_{\Gamma_1} = \frac{1}{|N_\Gamma|} \sum_{(x,y)\in\Gamma} \|U(x,y) - \hat{w}(x,y)_\Gamma\|^2 \tag{3.2}$$

$$\mathcal{L}_{\Gamma_2} = \frac{1}{|N_\Gamma|} \sum_{(x,y)\in\Gamma} \|\nabla U(x,y) - \hat{\theta}(x,y)\|^2 \tag{3.3}$$

$$\mathcal{L}_{\Gamma_3} = \frac{1}{|N_\Gamma|} \sum_{(x,y)\in\Gamma} \|(M_x + M_y) - \hat{\mathbf{M}}(x,y)\|^2 \tag{3.4}$$

where N_Ω is a set of collocation points inside the physical domain and N_Γ is a set of points on the boundaries. Here, the term $\mathcal{L}_\Omega$ incites the network G to approximate the governing equation, while $\mathcal{L}_\Gamma$ will enforce the boundary conditions and thus ensure the network's convergence towards the correct solution. Finally:

$$\mathcal{L} = \lambda_0\mathcal{L}_\Omega + \lambda_1\mathcal{L}_{\Gamma_1} + \lambda_2\mathcal{L}_{\Gamma_2} + \lambda_3\mathcal{L}_{\Gamma_3} \tag{3.5}$$

Since there is an infinite number of correct solutions to the governing equation, the hyperparameters λ_i must be carefully chosen for the network to respect the boundary conditions. If set too high or too low, the network could be led to neglect either one of the terms, thus leading to insufficiently accurate results.

3.3 Architecture

3.3.1 Fully-Connected Networks

The original PINNs consist of a variable number of fully-connected layers and *tanh* activations. This architecture has some flexibility in the width and depth of the network, while the nature of the *tanh* activation in theory allows to calculate infinite-order derivatives, making it well-suited for this task. We investigate the network's performance using different depths and widths as well as the effects of replacing *tanh* with *sigmoid* activations.

3.3.2 Multiplicative Residual Connections

PINNs with fully-connected layers were shown to be prone to gradient pathologies like vanishing and/or unbalanced gradients [44], which predominantly affects deeper architectures. While, in theory, a single hidden layer with finite width is sufficient to form a universal function approximator [21], it is widely understood that deeper networks learn features at various levels of abstraction and will generalise better in practice [42]. Therefore, inspired by the great success of ResNets [19], Wang et al. introduced a specific architecture for PINNs with residual connections and multiplicative operations, thus improving the gradient flow during backpropagation.

3.4 Activation functions

Activation functions have been thoroughly studied in the context of neural networks, some of the most popular choices being *ReLU*, *tanh* or *sigmoid*. While any of these activations have pros and cons, they all share the property that their first derivatives are well-behaved (with the exception of *ReLU* at $x = 0$). This is, however, no longer true in PINNs, where multiple higher order derivatives might be needed. Since the derivative of *ReLU* is constant and its Laplacian zero everywhere, this particular activation function can not be taken into consideration in its classical form. On the other hand, while there is in theory no limit to the number of derivatives of *tanh* and *sigmoid*, they become increasingly ill-behaved in higher orders as the derivatives change shapes at every level and their amplitudes vary considerably (see Fig. 3.1).

This property can be detrimental to performance, as the network's ability to model the terms in the PDE might differ, depending on the level of differentiation and the corresponding shape of the activation function. Furthermore, none of the higher order shapes resemble the state of the art activations used in regular neural networks, thus suggesting that they might be sub-obtimal.

Periodic activation functions as proposed by [38] could be remedy to this problem. More specifically, the authors use the *sine* function in combination with a specific weight-initialisation and demonstrate on a number of tasks that their so-called SIREN are able to fit higher order

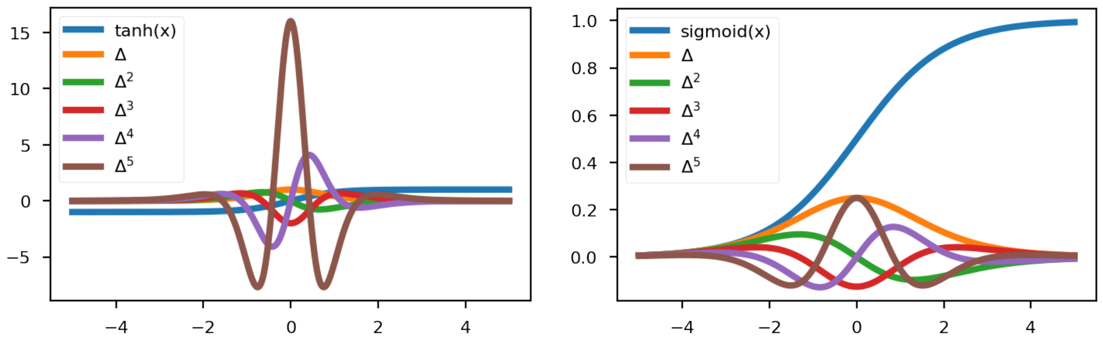


Figure 3.1: Multiple higher order derivatives for *tanh* and *sigmoid*

derivatives faster and with greater detail. While periodic functions might seem like a surprising choice of activations due to their repetitiveness, they have the property that their derivatives are again periodic. This particularity ensures that they keep the same modelling capabilities across all levels of differentiation.

3.5 Points sampling

While the sampling procedure plays a role in numerical methods like FEM as well, PINNs offer more flexibility in this regard, making the sampling a hyperparameter of great importance. In addition to the total number of points, the ratio between internal and boundary points and the structure (random, grid), we can also opt for dynamically resampling the points after every optimisation step.

3.5.1 Adversarial sampling

A further extension to dynamical sampling is employing an adversarial procedure, where the loss in the previous step acts as probability distribution. In this manner, points are more likely to be sampled in areas with higher errors.

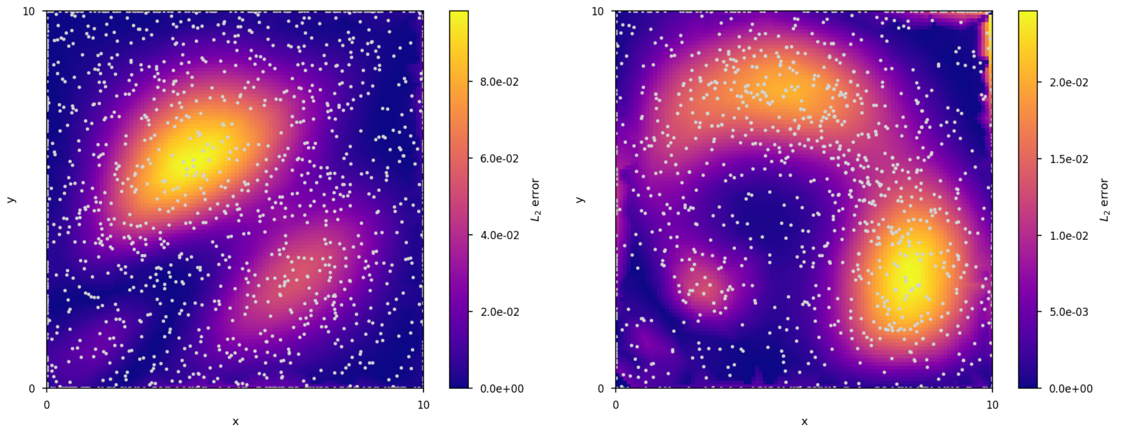


Figure 3.2: Random sampling (left) and andversarial sampling (right)

In adversarial sampling, the points and corresponding errors are taken from the previous optimisation step. Then, categorical sampling with replacement is performed where the errors act as logits for the probability distribution. Similar as in softmax, a temperature parameter T can be employed to create a more uniform or more discretised distribution by scaling the logits.

In order to keep discovering new error "islands" that might arise, we always combine adversarial with random sampling. We found that choosing half of the points adversarially and the other half randomly was a reasonable split. See Algorithm 1 for more details.

Algorithm 1 Adversarial sampling

Arguments: N number of points, E number of epochs, a, b width and height of plate, θ parameters of network, T temperature parameter, σ very small value for standard deviation (e.g. 10^{-4})

```
1: $x \leftarrow \mathcal{U}(0,a)^N$
2: $y \leftarrow \mathcal{U}(0,b)^N$
3: **for** $t \leftarrow 1, ..., E$ **do**
4:     $i \leftarrow Categorical(\frac{\mathcal{L}(x,y;\theta_t)}{T})^{\frac{N}{2}}$
5:     $x_{adv} \leftarrow x[i] + \mathcal{N}(0,\sigma)^{\frac{N}{2}}$
6:     $y_{adv} \leftarrow y[i] + \mathcal{N}(0,\sigma)^{\frac{N}{2}}$
7:     $x \leftarrow x_{adv} \parallel \mathcal{U}(0,a)^{\frac{N}{2}}$
8:     $y \leftarrow y_{adv} \parallel \mathcal{U}(0,b)^{\frac{N}{2}}$
9: **end for**
```

Since adversarial sampling induces a computational overhead, one can choose to perform Algorithm 1 only every nth optimisation step (e.g. every second, 10th, 20th, etc.) and keep dynamically sampling the non-adversarial points at every iteration. We chose to adversarially sample every 10th iteration.

3.6 Gradient balancing

During the early stages of training, the learning signals originating from $\mathcal{L}_i, i \in \{\Omega, \Gamma_1, \Gamma_2, \Gamma_3\}$ can vary considerably in strength, sometimes differing in several orders of magnitude. This can be due to different units of measurements and leads to the network potentially neglecting weaker terms. To mitigate this issue, the scalings λ_i must be chosen appropriately. This can either be done manually through extensive grid search, or an adaptive balancing algorithm can be employed.

We chose to compare our manual baseline to two adaptive balancing techniques: Learning Rate Annealing and Relative Loss Balancing.

3.6.1 Learning Rate Annealing

[44] proposed to adaptively scale the loss using gradient statistics, thus reducing the laborious tuning of these hyperparameters.

$$\begin{aligned} \hat{\lambda}_i &= \frac{\max\{|\nabla_\theta \mathcal{L}_\Omega|\}}{\overline{|\nabla_\theta \mathcal{L}_{\Gamma_i}|}}, \ i \in \{1,2,3\} \\ \lambda_i^{(t)} &= \alpha \lambda_i^{(t-1)} + (1-\alpha)\hat{\lambda}_i \\ \theta &= \theta - \eta \nabla_\theta \mathcal{L}_\Omega + \eta \sum_{i=1}^{3} \lambda_i \nabla_\theta \mathcal{L}_{\Gamma_i} \end{aligned} \tag{3.6}$$

where $\overline{|\nabla_\theta \mathcal{L}_{\Gamma_i}|}$ is the mean of the gradient w.r.t. the parameters θ and a value $\alpha = 0.9$ recommended by the authors. If $\max\{|\nabla_\theta \mathcal{L}_\Omega|\}$ grows considerably larger than $\overline{|\nabla_\theta \mathcal{L}_{\Gamma_i}|}$, the scalings $\hat{\lambda}(t)$ correct for this discrepancy and ensure that all gradients have similar magnitudes. In addition, exponential decay is employed in order to smoothen the balancing and avoid drastic changes of the loss space between optimisation steps.

3.6.2 Relative Loss Balancing

Relative Loss Balancing keeps track of each term's relative progress and adapts the scalings such that all terms improve at the same rate. This is achieved by comparing the rate of change between

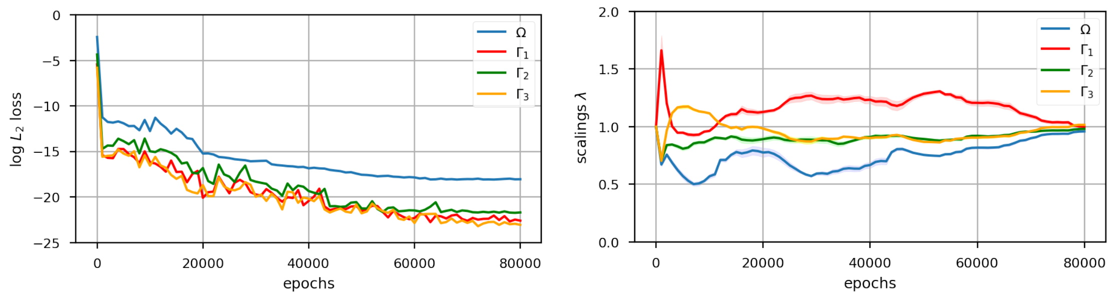


Figure 3.3: Median of the log L_2 loss over multiple training runs (left) and the mean and variance of the corresponding scaling factors λ (right), calculated with Relative Loss Balancing.

timestep t and $t-1$: $\mathcal{L}_i^{(t)}/\mathcal{L}_i^{(t-1)}$. The resulting rates are then normalised through a softmax function. Similar to Learning Rate Annealing, exponential decay is employed to store statistics from more than just one timestep and to smoothen the scalings over time. Figure 3.3 shows how the scalings react to changes in the loss.

$$
\begin{aligned}
\hat{\lambda}_i^{(t)} &= \frac{exp\left(\frac{\mathcal{L}_i^{(t)}}{T\mathcal{L}_i^{(t-1)}}\right)}{\sum_{j=1}^{m} exp\left(\frac{\mathcal{L}_j^{(t)}}{T\mathcal{L}_j^{(t-1)}}\right)}, \quad i \in \{0,1,2,3\} \\
\lambda_i^{(t)} &= \alpha\lambda_i^{(t-1)} + (1-\alpha)\hat{\lambda}_i^{(t)}
\end{aligned}
\tag{3.7}
$$

where $\mathcal{L}_i^{(t)}$ is the loss of term i at optimisation step t, α is a hyperparameter controlling the model's ability to "remember" past statistics, and T is a hyperparameter defining whether the resulting distribution should resemble more a uniform distribution ($T \to \infty$) or the discrete *argmax* function ($T \to 0$). We achieved the best performances when choosing $\alpha = 0.999$ and $T = 0.1$.

3.7 Results

In this type of optimisation, a separate neural network (so-called neural representation networks) is trained for every instance of a plate and its boundary conditions. Furthermore, the number and positions of collocation points can be freely chosen beforehand. As such, the networks do not necessarily have to generalise and can be trained indefinitely long, something that in other circumstances would be considered as "overfitting".

We limited our training to 10^5 steps of gradient descent using the Adam optimiser [26] and an initial learning rate of 0.001. Additionally, we reduced the learning rate by a multiplicative factor of 0.3 whenever the optimisation stopped making progress for over 3000 optimisation steps and finally employed early stopping in case of 9000 steps without improvement.

We tested the different architectures and optimisation methods on a simply-supported square plate under a distributed load with an analytical solution (Figure 3.4).

$$
p = p_0 \sin\left(\frac{x\pi}{a}\right) \sin\left(\frac{y\pi}{b}\right) \tag{3.8}
$$

where a, b denote the length of the plate. Using Eq. (2.11), the displacement w in this problem is

$$
w = \frac{p_0}{\pi^4 D \left(\frac{1}{a^2} + \frac{1}{b^2}\right)^2} \sin\left(\frac{x\pi}{a}\right) \sin\left(\frac{y\pi}{b}\right) \tag{3.9}
$$

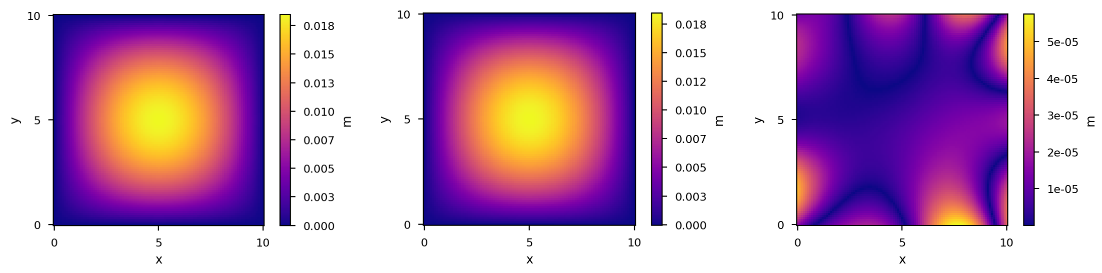


Figure 3.4: Kirchhoff equation predicted with a fully-connected network consisting of three layers and 128 nodes each. Exact solution (left), prediction (center), absolute error (right).

3.7.1 Sampling method

Table 3.1 shows the mean- and maximum squared errors when sampling the points once at the beginning of training and then keeping them fixed (*fixed*), when resampling them randomly at every iteration (*dynamic*) and when sampling them adversarially as described in Section 3.5.1 (*adversarial*).

method	MSE	max sq. err.	exec. time (s) (1'000 steps)
fixed	$3.55{\cdot}10^{-7}$	$1.31{\cdot}10^{-5}$	24.4
dynamic	$4.02{\cdot}10^{-7}$	$9.73{\cdot}10^{-6}$	26.0
adversarial	$4.81{\cdot}10^{-7}$	$4.58{\cdot}10^{-6}$	29.6

Table 3.1: Comparison of the mean- and maximum squared error when the points are sampled using different methods. Reported numbers are the median over all performed tests.

The results indicate that there is a trade-off between average accuracy and the maximum error: while training the model on a fixed set of collocation points yields the best accuracy averaged over all points, it also has the highest maximum error amongst the three methods. On the other hand, an adversarially trained model performs slightly worse on the mean squared error, while it achieves the lowest maximum squared error. In absolute values, the benefits of adversarial sampling are more significant, probably making it the best option for applications in practice since reducing maximum errors is crucial for safety reasons. However, requirements might differ and should therefore be determined on a per-task basis.

Another hyperparameter, one of great concern in FEM as well, is the ratio between the number of points in the domain and on the boundaries. Table 3.2 shows that the optimal ratio sits at 66% of the points inside the domain and 33% on the boundaries. In order to keep the number of performed experiments at a manageable size, all the hereafter reported results were achieved using this finding, i.e. with two thirds of the points inside the domain and one third on the boundaries.

ratio $\|\Omega\|$, $\|\Gamma\|$	mean squared error	max. squared error
75%, 25%	$1.92{\cdot}10^{-7}$	$4.05{\cdot}10^{-6}$
66%, 33%	$1.44{\cdot}10^{-7}$	$2.85{\cdot}10^{-6}$
50%, 50%	$2.00{\cdot}10^{-7}$	$3.48{\cdot}10^{-6}$

Table 3.2: Comparison of the mean- and maximum squared error when varying the ratio between number of points inside the domain and on the boundaries with fixed batch size.

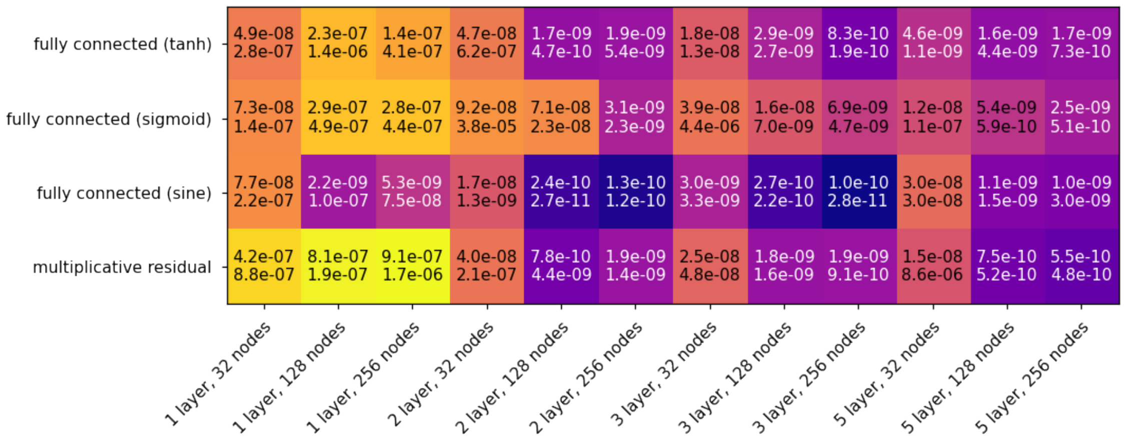


Figure 3.5: MSE in meters between the prediction and the exact solution of different networks with varying depth and width (darker is better). Reported values are the median and standard deviation over five experiments conducted with the given settings.

3.7.2 Architecture

Figure 3.5 shows the effects of varying the depth and width of networks with different architectures and activation functions. The results indicate that a certain depth and width is necessary to achieve good results. The difference in accuracy between one and two layers is considerable. A third layer boosts the performance by another small margin. However, this trend seems to stop after three layers, which might be due to pathologies arising in deeper networks, like vanishing gradients.

layers	nodes	exec. time (s) (1'000 steps)
1	32	5.4
	128	6.0
	256	8.2
2	32	10.3
	128	15.0
	256	27.1
3	32	15.9
	128	25.5
	256	50.1
5	32	28.6
	128	48.0
	256	94.5

Table 3.3: Comparison of the execution time when varying depth and width of the network. Reported numbers are the median over all experiments with the given settings.

Table 3.3 shows that additional layers induce a costly increase in computation time. Given that the best results are already close to the floating point precision error, gaining a few percent in accuracy is most likely not worth the extra effort of going beyond 5 layers or wider than 256 nodes. Furthermore, the precision at which these networks operate, will in practice already be outweighed by uncertainties in the materials like the plate rigidity. Therefore, a shallow network with just two layers and 128 nodes will most likely yield results that are accurate enough for most real-world applications while still being fairly efficient to train.

Concerning the activation functions and architecture of the network, all studied types achieved satisfactory results, with *sigmoid* being slightly below average. On the other hand, *sine* is particularly well-suited for this type of problems.

3.7.3 Adaptive loss balancing

In this section, we compare the performances of the two loss balancing techniques introduced in Section 3.6 against a baseline where we manually searched for the optimal scalings λ through grid-search.

method	MSE w	MSE m
Manual	$1.01{\cdot}10^{-10}$	$1.37{\cdot}10^{-7}$
LR Annealing	$3.42{\cdot}10^{-10}$	$1.03{\cdot}10^{-6}$
Relative	$1.25{\cdot}10^{-10}$	$1.22{\cdot}10^{-7}$

Table 3.4: Comparison of the performances when using different loss balancing techniques. Reported numbers are the median mean squared error against the exact solution of the displacements (w) in meters as well as the moments (m) in mega Newton-meters over five experiments using the same hyperparameters.

Table 3.4 indicates that carefully and manually chosen scalings outperform both adaptive balancing algorithms when compared against the exact displacements w. With a standard deviation of $7.20{\cdot}10^{-8}$ over all experiments conducted with manual balancing, it also achieves very consistent results. On the other hand, Relative Balancing has a slightly higher stochasticity at $2.82{\cdot}10^{-7}$ and using LR Annealing results in a few instances with high errors, driving the standard deviation up to $5.11{\cdot}10^{-4}$. Despite the higher standard deviation between runs, the errors of Relative Loss Balancing are in such a small range that the reduced workload when employing a balancing algorithm will most likely overweigh the slight difference in performance, as discussed in the previous section.

While a high accuracy of the predicted displacements is of top priority in structural engineering, a good performance on the moments is also necessary to calculate the bending of a plate. In this regard, it is worth noting that Relative Balancing achieves the best accuracy when compared to the exact moments m. One could imagine that additional tuning of the manually chosen scalings could lead to more equalised predictions between displacements and moments in Manual balancing. However, Relative Balancing captures this property rather well out-of-the-box.

The benefits of Relative Balancing come with just a slight computational overhead (Table 3.5).

method	exec. time (s) (1'000 steps)
Manual	18.5
LR Annealing	40.4
Relative	20.3

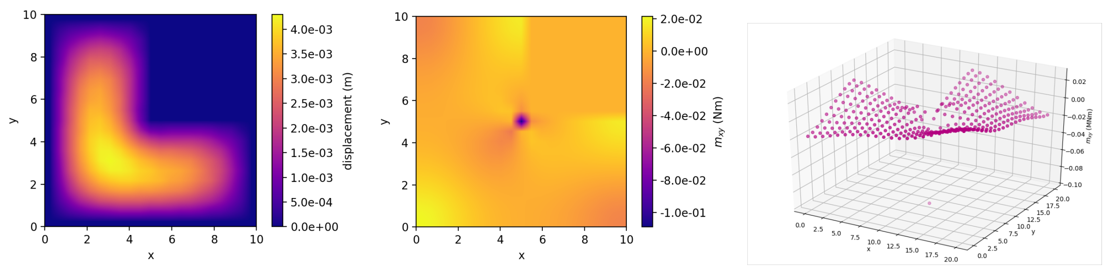
Table 3.5: Comparison of the execution time when using different loss balancing techniques. Reported numbers are the median over all experiments with the given settings.

PINN for bending of L-shaped plates

4.1 Limitations of vanilla PINN

Physics-Informed Neural Networks have shown great results at predicting the Kirchhoff bending of simply supported square plates (Section 3). In fact, the best performing models achieve accuracies that go well beyond the threshold necessary to be used in practice. Guo et al. [17] also demonstrated the PINN's ability to approximate clamped and round plates. However, these examples represent simple instances of the plate bending problem, as they are all globally C_2 continuous.

To test the network's generalisation to more complex examples, we applied PINNs to an L-shaped plate under uniform load. The L-shape introduces a discontinuity in the second derivative,

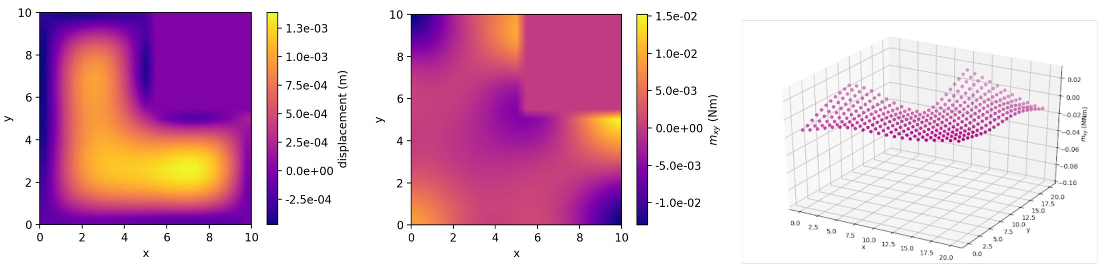


(a) Displacement w (m) (b) Moments m_{xy} (MNm) (c) Moments m_{xy} (MNm) in 3D

Figure 4.1: Displacements and moments within an L-shaped plate under uniform load. Results calculated using the finite element method.

resulting in a singularity at the center of the plate. While Neural Networks are universal function approximators [21], capturing sharp edges still poses a significant challenge.

In fact, despite using specialised hardware (high-end GPUs) and all the methodologies studied in the previous sections, we were unable to train a network capable of approximating the L-shape sufficiently well. As can be seen in Figure 4.3, the network does not capture the singularity. The lacking accuracy in the moments manifests as underdefined L-knee in the displacements, which in turn has a negative effect on the boundaries by lifting them up.



(a) Displacement w (m) (b) Moments m_{xy} (MNm) (c) Moments m_{xy} (MNm) in 3D

Figure 4.2: Displacements and moments within an L-shaped plate under uniform load. Results predicted with a PINN.

4.2 Domain decomposition with XPINN

In order to tackle this problem, we draw inspiration from FEM by decomposing the domain into multiple subdomains and training a separate PINN on each of these subdomains. This type of optimisation is called XPINN [24]. While this introduces additional boundaries between the subdomains that need to be carefully tuned, the complexity of the problem is split over various networks. Furthermore, when dividing the plate into three parts - one in each of the corners - an inductive bias is introduced by placing the boundaries exactly where the discontinuities in the second derivatives occur. The continuity on the two boundaries between subdomains can be added

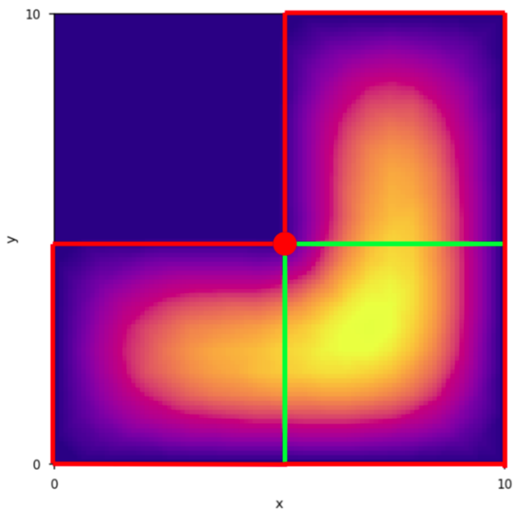


Figure 4.3: Decomposition of an L-shaped plate into three subdomains. The red lines denote the areas where the boundary conditions apply. In addition, continuity conditions must be enforced on the green lines. In an XPINN, a separate neural network is used for each subdomain.

to the loss function in the form of additional terms and scalings. The number of new terms depends on the level of differentiation where the continuity should be enforced, i.e. C_1, C_2 or C_3 continuity. For example, the loss function enforcing continuity on a single level can be denoted the following way:

$$\mathcal{L} = \lambda_0\mathcal{L}_\Omega + \lambda_1\mathcal{L}_{\Gamma_1} + \lambda_2\mathcal{L}_{\Gamma_2} + \lambda_3\mathcal{L}_{\Gamma_3} + \lambda_4\mathcal{L}_{\Gamma_1|\Gamma_2} + \lambda_5\mathcal{L}_{\Gamma_1|\Gamma_3} \tag{4.1}$$

Given the effectiveness of Relative Loss Balancing introduced in Section 3.6.2, one could be tempted to apply the algorithm to the scalings λ_i, $i \in \{0, \ldots, 5\}$. Unfortunately, the premise of improving all terms at the same rate drives the model into a failure state. Since all networks are initialised around zero, the errors between neighboring subdomains are relatively small at the start of training. However, in order to make progress, the networks need to be allowed to divert for approximating the vertical displacement. If the continuity is enforced too strictly right from the start, then the continuity conditions will be regarded as boundary conditions and the values on the boundaries between subdomains set to zero.

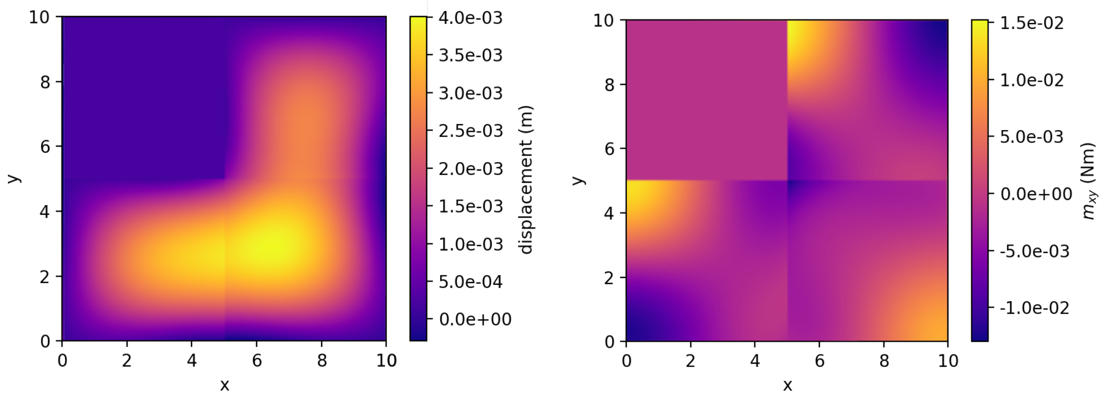
The problem of letting the networks explore new states while also enforcing continuity is known as Exploration-Exploitation trade-off and arises often in the context of Reinforcement Learning [10][45]. A basic approach to control this trade-off is by sporadically adding and removing the continuity conditions from the loss function. Similar to the Random Lookback in Section 3.6.2, we introduce a set of Bernoulli random variables ρ_i, $i \in \{0, \dots, m'\}$, where m' denotes the number of continuity conditions. In the case of C_1 continuity with three subdomains, Equation 4.1 can be extended the following way:

$$\mathcal{L} = \lambda_0 \mathcal{L}_\Omega + \lambda_1 \mathcal{L}_{\Gamma_1} + \lambda_2 \mathcal{L}_{\Gamma_2} + \lambda_3 \mathcal{L}_{\Gamma_3} + \rho_0 \lambda_4 \mathcal{L}_{\Gamma_1|\Gamma_2} + \rho_1 \lambda_5 \mathcal{L}_{\Gamma_1|\Gamma_3} \tag{4.2}$$

In this manner, the model is free to explore new states whenever the corresponding random variable evaluates to zero, while it will focus on continuity if the opposite is the case. $\mathbb{E}[\rho_i]$ is a hyperparameter and can be chosen arbitrarily in [0, 1]. We achieved the best results when starting with a relatively low value (e.g. 0.2) and slowly increasing it as training progressed. However, $\mathbb{E}[\rho_i]$ should only be set to exactly one at the end of training. During optimisation, we found the range [0.2, 0.9] to be most effective. Also, the scalings of continuity conditions should not be controlled by Relative Loss Balancing, even with the introduction of the Bernoulli random variables. We suggest searching the optimal scalings either manually through grid-search, or by employing Learning Rate Annealing (Section 3.6.1). On the other hand, scalings of the governing equation and the boundary conditions can still be chosen by using any of the proposed techniques.

4.3 Results

Figure 4.4 shows the prediction of an XPINN trained with the Exploration-Exploitation trade-off, Relative Balancing over the scalings of the governing equation and boundary conditions, as well as Learning Rate Annealing over the scalings of the continuity conditions.



(a) Displacement w (m) (b) Moments m_{xy} (MNm)

Figure 4.4: Displacements and moments within an L-shaped plate under uniform load. Results predicted with an XPINN using the Exploration-Exploitation trade-off and enforcing C_1 continuity.

The XPINN clearly improves the performance compared to a vanilla PINN. The displacements are closer to the results calculated with FEM and the "knee" of the L is more distinct. While the predictions are still far from being as accurate as the predictions for the square plate, additional research could bring further improvements and see PINNs conquering even this particularly challenging task. It is also worth noting that the singularity problem is less acute in practice than it is in theory, as the properties of materials have a counter-acting effect. Therefore, one might argue that perfectly approximating the discontinuities is not necessary in order to get predictions for real-world applications. However, this would have to be carefully studied and goes beyond the scope of this thesis.

Analysis of hidden representations in PINNs

Neural Networks are often regarded as black-box models due to the difficulty of comprehending their decision-making [6]. This uncertainty, along with the costs associated with gathering high quality datasets, has long been a hindering factor for the introduction of Deep Learning in Structural Engineering [27].

PINNs have the ability of modelling an underlying function in an unsupervised fashion by approximating the governing equation and respecting boundary- and initial conditions. This eliminates the need for large amounts of labelled data samples and is a great advantage over classical Neural Networks. Concerning their interpretability, little effort has been put into investigating the inner workings and theory behind PINNs. However, a few simple experiments are enough to reveal that their behaviour is fundamentally different from what we would expect in classical Deep Learning.

5.1 Experiments

It is widely understood that depth in Neural Networks allows modelling features at various levels of abstraction and increases the model's capability to generalise in practice [13][42]. In the case of Kirchhoff's plate bending problem, one would expect a deep network to find symmetries and cluster coordinates that evaluate to the same displacement. Such a clustering can be detected by feeding the network with a set of input coordinates and then visualising the internal (hidden) representations using a technique like t-SNE [43].

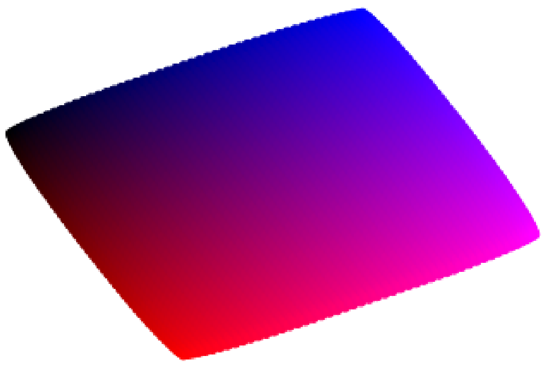
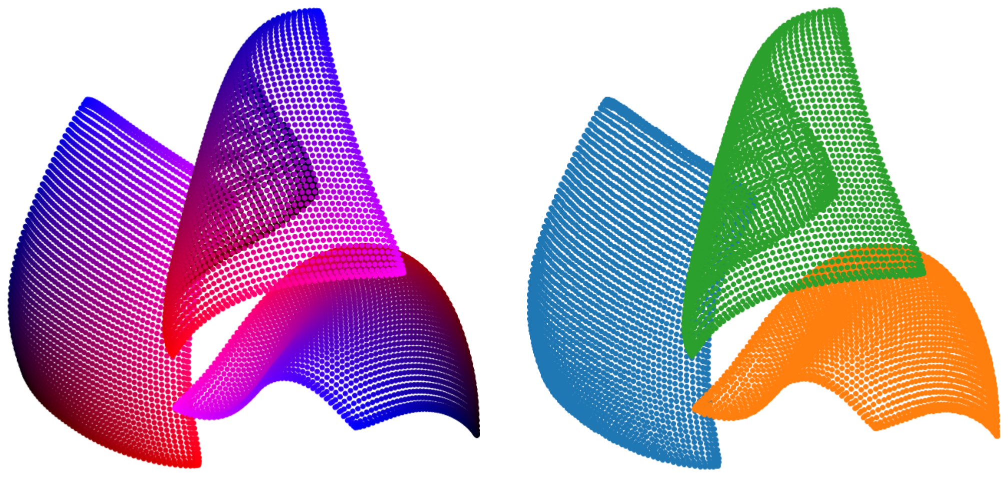


Figure 5.1: t-SNE visualisation of hidden representations in the last layer of a PINN with three layers. A higher x-value in the input coordinate results in a higher red value in the pixel color, while a higher y-value results in more blue (i.e. coordinate (0, 0) is in the top left corner, (1, 1) in the bottom right corner).

As can be seen in Figure 5.1, this is not what happens in PINNs. The complete lack of clustering and mixing leads to an internal representation that still resembles a two-dimensional plate. The experiment gets even more surprising when visualising the representations in all three layers at the same time (Figure 5.2). Not only do the coordinates not mix, but even the layers all



(a) x value of input contributes to red pixel value, y to blue.

(b) Distinguished layers: layer 1 (blue), layer 2 (orange), layer 3 (green)

Figure 5.2: t-SNE visualisation of hidden representations in a PINN with three layers.

live in a distinct space on the manifold. This observation also holds for different types of plates (e.g. the L-shaped plate) and even for completely different PDEs, like the Helmholtz equation.

5.2 Implications

There is undoubtedly a mathematical explanation for this phenomenon and finding it might open up a vast number of new possibilities for improving the performance of PINNs. For example, it would allow us to build specialised architectures that favor this kind of internal representations. Furthermore, the clear shapes in the hidden representations could give hints as to how PINNs make their predictions and potentially reveal weak points and failure states, which would be a great boost to their interpretability and thus also to their adequacy for use in practice.

Hypernetworks

Since PINNs utilise physical laws to estimate their accuracy and do not require paired input-output data, they can be regarded as Unsupervised Learning methods and have therefore sparked great interest in fields which see themselves regularly confronted with problems in the low data regime [3][12][17]. Furthermore, the continuous parameterisation of PDEs is a major advantage of PINNs compared to discrete mesh-based methods like the finite element method [23]. It allows for exact predictions anywhere on the physical domain and is limited only by the capacity of the underlying network instead of the grid resolution. However, PINNs are so-called implicit representation networks [38], meaning that they model specific PDEs with given boundary- and initial conditions. Note that the constraints can not be passed to the network as inputs, since they are not directly included in the underlying physical laws. The same goes for hyperparameters in the PDE, like the plate thickness, Poisson ratio or Young's modulus in the case of Kichhoff's plate bending problem. This in turn means that the network needs to be retrained whenever the constraints or hyperparameters change. If classical neural networks are finding their way into many real-time applications like autonomous driving [22], it is because, once trained, they are able to make predictions in a single forward pass. In order to turn PINNs into a viable option compared to other methods like FEM, they must be able to make predictions for more than just one set of parameters and constraints.

The time a network requires to adapt to new conditions can be greatly reduced by using techniques like transfer-learning [47] or meta-learning [14][32]. However, these techniques still require at least one backward pass to update the model's weights, thus greatly increasing the computing time. Instead, we draw inspiration from other fields where representation networks have been studied for a long time in order to tackle this issue.

Representation networks are commonly used in Computer Vision to parameterise images [38], 3D shapes [15] or entire scenes [39] as an efficient form of storing and rendering information. These implementations all suffer from the same, aforementioned limitation. As a solution, Sitzmann et al. proposed to leverage hypernetworks [18] in order to learn priors over the space of functions that define them.

A hypernetwork converts input embeddings to weights and biases of an other, main network. By passing the hypernetwork a task-specific context embedding, one can therefore generate the weights of a specialised network for the given task in a zero-shot learning manner [46]. Hypernetworks eliminate the need for a backward pass at inference time and thus greatly reduce computation time compared to transfer learning or meta learning. While not as accurate and reliable as other, established techniques, PINNs generated by hypernetworks could become a valuable tool for fast prototyping by providing engineers with instant predictions in a single forward pass.

Applied to PINNs of the form $U(\mathbf{x};\theta_U)$ parameterised by weights θ_U, one can define a hypernetwork $H(\mathbf{z};\theta_H)$ and reformulate the training procedure:

$$\mathcal{L}_F = \frac{1}{|N_F|} \sum_{\mathbf{x} \in N_F} \|F(\mathbf{x}, U(\mathbf{x}; H(\mathbf{z}; \theta_H)), \nabla_{\mathbf{x}} U(\mathbf{x}; H(\mathbf{z}; \theta_H)), \dots) - 0\|^2 \tag{6.1}$$

$$\mathcal{L}_{B_m} = \frac{1}{|N_{B_m}|} \sum_{\mathbf{x} \in N_{B_m}} \|B_m(\mathbf{x}, U(\mathbf{x}; H(\mathbf{z}; \theta_H)), \nabla_{\mathbf{x}} U(\mathbf{x}; H(\mathbf{z}; \theta_H)), \dots) - 0\|^2 \tag{6.2}$$

$$\theta_H^{(t)} = \theta_H^{(t-1)} - \eta \nabla_{\theta_H} \left(\lambda_0 \mathcal{L}_F + \sum_{m=1}^{M} \lambda_m \mathcal{L}_{B_m} \right) \tag{6.3}$$

6.1 Method

The pipeline for generating PINNs with hypernetworks is divided into three parts:

- An **Encoder** E transforms a set of input features $\mathbf{z}$ to a latent representation. The input features can be any variables which, if modified, would require an original PINN to be retrained. In the case of Kirchhoff's plate bending problem, this could be the load applied on the plate, the boundary conditions, the plate rigidity, etc. Depending on the nature of the input features, the Encoder can consist of fully-connected layers, convolutional layers, recurrent layers or a combination thereof.
- The **Generator** G transforms the latent context representation into weights and biases for the PINN. It consists of a variable number of fully-connected layers. Note that the Generator's own weights and biases should be initialised with care, as their value directly influences the weights in the PINN. If the generated weights have a large standard deviation, exploding gradients might impede proper training.
- The **PINN** P receives as input the generated weights as well as the variables $\mathbf{x}$ modelled by the approximated PDE, e.g. position, time, or both. It then emulates a fully-connecte feed-forward network by applying vector multiplication and addition between the input variables and the weights.

6.1.1 Partial PINN generation

Albeit PINNs being relatively small networks compared to state of the art architectures in other fields like image recognition [41] or natural language processing [5], the best performing models still comprise multiple thousands of weights (see Results in Chapter 3). Since the number of weights grows linearly with the depth and quadratically with the width of a network, naively parametrising a PINN requires a large hypernet which can quickly become impractical [29].

To circumvent this issue, we investigate if generating only parts of the PINN is enough for them to still accurately adapt to new tasks. More specifically, we give innate, trainable weights to the first, as well as the last layer in the PINN and generate the weights and biases only for the remaining hidden layers.

6.1.2 Learnable Dictionary

Another method of addressing the curse of dimensionality in hypernetworks is by using bayesian inference [29]. We provide the network with a trainable dictionary of weights for PINNs and employ an attention mechanism to create new PINNs from a linear combination of stored values. The trainable dictionary contains pairs of keys (vectors of the same dimensionality as the context embedding created by the Encoder) and weights (of the same dimensionality as the weights created by the Generator). The cosine similarity between the context embedding and the stored keys then defines how much a particular entry in the dictionary contributes to the generated PINN. Finally, the weights issued by the linear combination of stored PINNs are added to the values generated by the Generator (Figure 6.1).

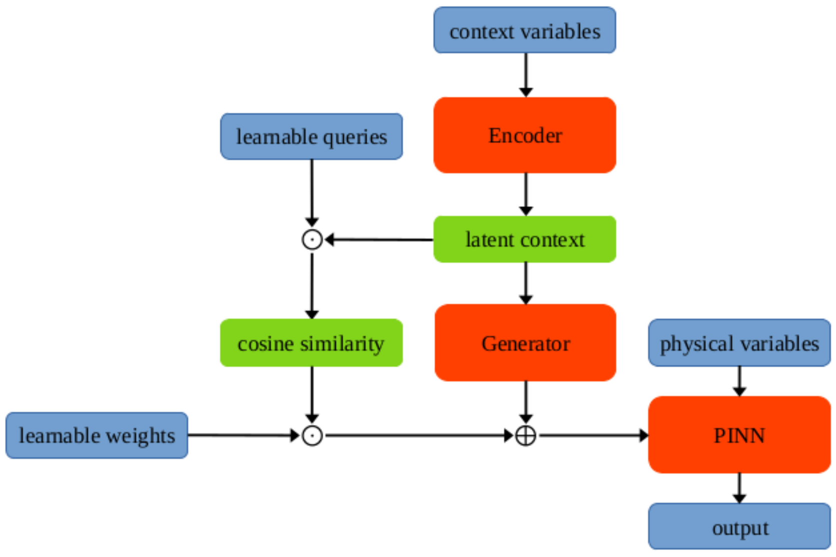


Figure 6.1: Structure of a hypernet with learnable dictionary. The blue cases represent input/output nodes, the green cases are latent representations and the red cases define modules that can themselves contain one or more hidden layers.

More formally, the PINN's weights θ_P when using a dictionary are computed using the following formula:

$$\theta_P = G(E(\mathbf{z})) + \mathbf{V}^T \text{softmax}(\mathbf{K}E(\mathbf{z})) \tag{6.4}$$

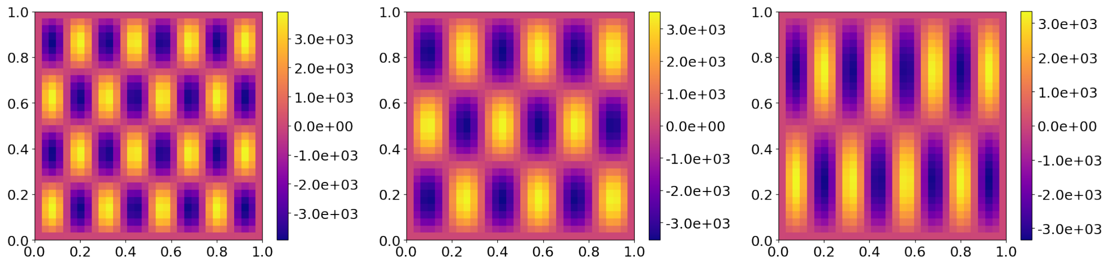
where $\theta_P \in \mathbb{R}^p$ are the PINN's weights, $E(\mathbf{z}) \in \mathbb{R}^d$ is the context embedding, $K \in \mathbb{R}^{n \times d}$ are the learnable keys, $V \in \mathbb{R}^{n \times p}$ are the learnable weights and n is a hyperparameter that denotes the number of entries in the dictionary.

6.2 Experiments

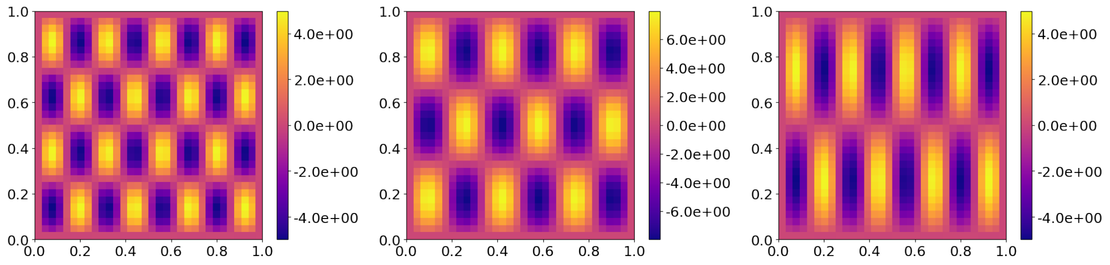
We tested hypernetworks on the Helmholtz and Kirchhoff equations introduced in Chapter 2. Since the idea of hypernetworks is to make predictions for more than just one instance of a PDE, we first had to construct a dataset. We again choose the function $u(x, y) = i \sin(j\pi x) sin(k\pi y)$, $i \in \mathbb{R}$, $j, k \in \mathbb{Z}^+$ as Helmholtz equation and resample i, j and k at every iteration in order to generate a synthetic dataset. We proceed similarly for the Kirchhoff equation. A selection of samples for both functions is presented in Figures 6.2 and 6.3.

The pipeline is defined as follows: the governing equation $f(x, y)$ is passed to the hypernetwork (for Kirchhoff's equation, we additionally pass the flexural rigidity D) and let it predict $u(x, y)$. By taking the appropriate derivatives of the PINN within the hypernetwork w.r.t. the coordinates x, y we receive the predicted governing equation $f'(x, y)$ and can calculate the mean sqared error in order to evaluate the performance.

The collocation points are chosen on a 32×32 grid in order to be able to embed the contextual features using a convolutional network. Note that despite this choice, the generated PINN is still a continuous function and can be evaluated in-between the collocation points. The width of the latent context representation, the depth of the hypernetwork and PINN as well as their width and activation functions are hyperparameters. In addition, we can choose the option of generating only the hidden layers of the PINN (partial PINN) or to add the dictionary.

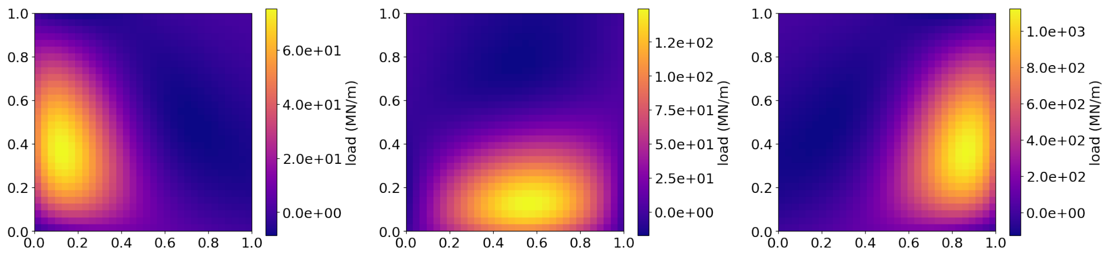


(a) Governing equation $F(x, y)$

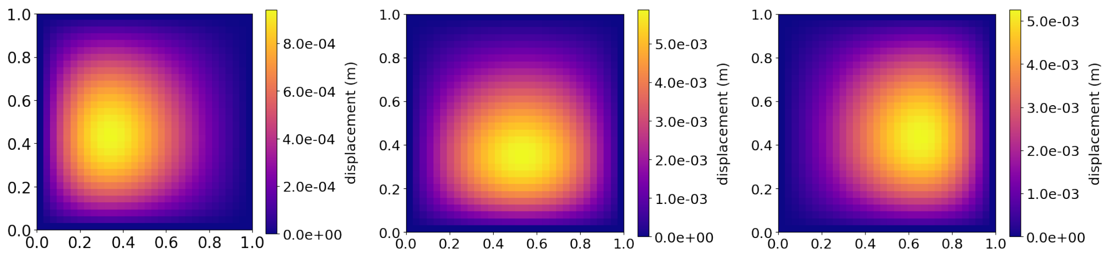


(b) Analytical solution $u(x, y)$

Figure 6.2: Different generated samples with analytical solution for the Helmholtz equation.

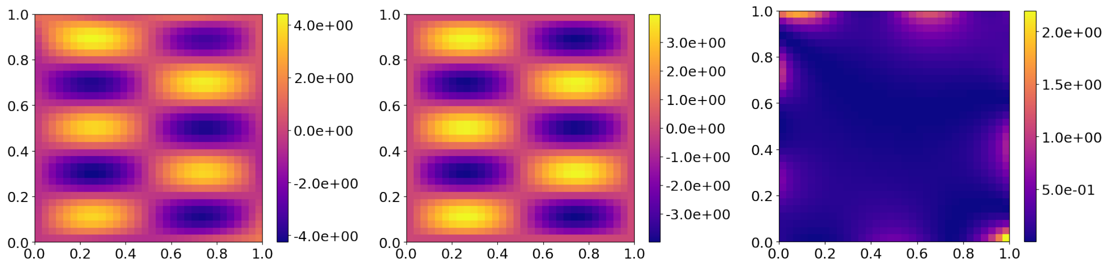


(a) Governing equation $F(x, y)$

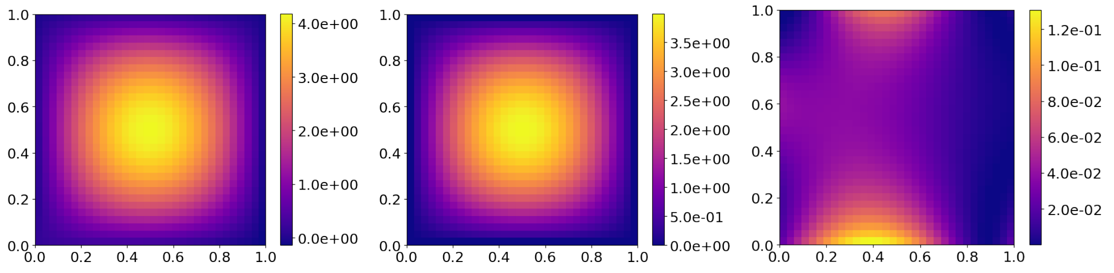


(b) Analytical solution $u(x, y)$

Figure 6.3: Different generated samples with analytical solution for the Kirchhoff equation.



(a) Helmholtz



(b) Kirchhoff

Figure 6.4: Predictions made by a hypernetwork (left), generating all weights of a PINN with 1 hidden layer and a width of 32 nodes, compared to the analytical solution (center) and the squared error (right).

6.3 Results

Figure 6.4 shows examples of predictions made by a hypernetwork. As can be seen, it captures the underlying physics very accurately. Training a PINN with similar accuracy would require approximately one minute for the Helmholtz equation and two to three minutes for the Kirchhoff equation on a high-end GPU (as per Chapter 3), whereas the predictions made by the hypernetwork incur almost no latency and can be computed on any average CPU. One can also infer from Figure 6.4 that the errors lay mainly on the borders. While we employed Relative Loss Balancing introduced in Chapter 2, a further decrease of the temperature parameter might be necessary in order to get smaller errors on the boundaries and a more uniformly distributed error overall.

PDE	partial PINN	dictionary	L_2 training error	L_2 val error
Helmholtz	False	False	1.848	2.007
	True	False	0.498	0.476
	False	True	0.148	0.150
	True	True	0.114	0.115
Kirchhoff	False	False	2.314	3.283
	True	False	3.835	4.380
	False	True	2.283	3.724
	True	True	2.932	2.876

Table 6.1: Comparison of median results when ablating partial PINN generation and dictionary.

The results in Table 6.1 offer no conclusive answer as to the effectiveness of the partial PINN generation or the use of a dictionary. While both methods greatly improve the performance on the Helmholtz equation, they have little effect on the Kirchhoff equation. It's possible that Kirchhoff's fourth order differential equation requires radically different networks across tasks and therefore sharing parameters could reduce the hypernetwork's flexibility to generate new outputs. However, more tests are necessary in order to prove or disprove this hypothesis, e.g. by varying the number of trainable layers in the partial PINN generation and by testing dictionaries with more learnable entries.

Module	depth	width	L_2 training error	L_2 val error
Generator	2	64	17.428	17.492
	2	128	0.624	0.583
	2	256	13.491	21.509
	3	128	1.184	1.233
	3	256	27.344	33.296
PINN	1	32	2.195	2.240
	1	64	0.280	0.279
	2	32	0.624	0.583
	2	64	0.130	0.101

Table 6.2: Comparison of median results when varying the depth and width of the Generator or PINN on the Helmholtz equation.

The effects of choosing different architectures for the Generator and PINN are reported in Table 6.2. In Chapter 3, the extensive tests of different PINN architectures for the Kirchhoff equation suggested that more depth and width tended to improve the networks' performance. It is interesting to note that the same effect appears when using hypernetworks to generate the PINNs, despite the exponential increase in dimensionality of the output. Also, a rather shallow and narrow fully connected Generator seems to be enough to create these PINNs. It is however likely that PINNs with more than two layers and 64 nodes will also require larger hypernetworks. Note that the reported numbers stem from hypernetworks that could not be trained to full convergence due to resource limitations. Continuing to train these networks for longer on a broader domain would further improve their predictions and capability to generalise.

It is also important to note the great variance in performance when modifying the width of the hypernetwork. A width of 128 nodes appears to be particularly well-suited to the problem, while 64 or 256 nodes perform considerably worse. This discrepancy highlights the hypernetworks' sensitivity to changes in hyperparameters. Currently, these models require a lot of manual tuning to avoid divergence during training. While the effort of tuning a hypernetwork is more rewarding than seeking the optimal PINN, because hypernetworks can generate an infinite amount of the latter, it is still foreseeable that additional research will seek to reduce the workload by means of automated heuristics, as was done for the automated loss balancing in Chapter 2, e.g. by finding a way of automatically selecting the appropriate standard deviation of generated weights.F

Conclusion

Physics-Informed Neural Networks have shown great potential in many applications due to their ability to leverage well known physical laws and thus greatly reducing or even eliminating the need for large, annotated datasets. Their continuity, flexibility and the perspective of potentially solving arbitrary instances of a given PDE in a single forward pass, might turn them into a valuable alternative to established, mesh-based numerical techniques like the finite elements method (FEM) [2][23].

However, contrary to the impression one might get when studying the literature about Physics-Informed Deep Learning, PINNs have not yet reached the maturity to be seriously taken into consideration in practice. The stochasticity arising from the random weights initialisation in combination with their reliance on gradient descent means that PINNs can produce unsatisfactory solutions due to local minima, imbalanced losses or insufficient modelling power in the architecture. Furthermore, training a PINN to the accuracy reported in Section 3 takes about 10min on a high-end GPU and is therefore also considerably more computationally expensive than FEM.

Our work on Adaptive Loss Balancing and more specifically the Relative Loss Balancing technique is a contribution towards reducing the workload necessary to successfully train a PINN by dynamically balancing the loss function. We showed that, despite the stochasticity of the random initialisation, the training progress with Relative Loss Balancing follows similar patterns across multiple independent runs, thus reducing the probability of a failure mode. Furthermore, inspecting the adaptively chosen scalings can provide valuable insight into the training process and allows to take informed decisions in order to improve the framework.

In our parameter study, we then focused on applying a wide selection of state-of-the-art techniques, proposed in the context of PINNs, to the Kirchhoff plate bending equation with the goal of consolidating knowledge for this problem. More specifically, we tested several network architectures, activation functions and sampling techniques, as well as two adaptive loss balancing algorithms. We analysed the results in perspective with important properties from structural engineering, highlighted hyperparameters and extensions which improve the model's performance, and identified a number of trade-offs that need to be addressed on a per-task basis. Finally, we noted that the best performing models achieved accuracies exceeding minimum requirements for practical applications.

Finally, we successfully implemented hypernetworks in charge of generating task-specialised PINNs for multiple instances of PDEs. This eliminates the need for retraining the networks at every change of constraints or context variables. Instead, hypernetworks are able to make predictions in a single forward pass, thus greatly boosting the PINNs utility as a tool for fast prototyping.

Some problems, however, remain to be solved. Experiments on the bending of an L-shaped plate revealed that the good performance of PINNs on square plates does not generalise to all types of plates. Furthermore, the peculiar internal representation of PDEs in PINNs indicates that PINNs might require radically different architectures and optimisation techniques compared to classical Neural Networks.

Outlook

The interest for Physics-Informed Neural Network amongst the researching community is impressive to the point where it is difficult to keep up with the newest advancements. Researchers from various different fields experiment with PINNs as a new toolbox for solving physical problems. With adaptive loss balancing and hypernetworks, we made a significant step towards more robust and efficient PINNs. However, the potential of these approaches is by far not fully tapped. Especially hypernetworks have been introduced in this thesis as a bare proof of concept. Further research could go towards bayesian optimisation for hyperparameter search, regularisation of the generated PINNs for sparcity, inspection of latent representations for interpretation purposes, or making the generating process more flexible by employing state of the art methods like PonderNet, which would allow a single hypernetwork to generate PINNs of variable depth and width.

In a broader context, PINNs can become a valuable tool, not only for fast prototyping, but as means of optimisation. Since they are differentiable parameterisations of physical laws, they could be used to optimise designs of cars, planes, buildings or other objects subject to differential equations. Many processes, which currently rely on laborious grid-search over hyperparameters as well as the intuition and experience of the engineer, could be optimised much more efficiently through gradient descent.

Bibliography

[1] B. Alipanahi, A. Delong, M. T. Weirauch, and B. J. Frey. Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning. *Nat Biotechnol*, 33(8):831–838, Aug 2015.

[2] K.J. Bathe. *Finite Element Procedures.* Number pt. 2 in Finite Element Procedures. Prentice Hall, 1996.

[3] Alex Bihlo and Roman O. Popovych. Physics-informed neural networks for the shallow-water equations on the sphere. *arXiv e-prints*, page arXiv:2104.00615, April 2021.

[4] Mayur P. Bonkile, Ashish Awasthi, C. Lakshmi, Vijitha Mukundan, and V. S. Aswin. A systematic literature review of burgers' equation with recent advances. *Pramana*, 90(6):69, Apr 2018.

[5] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. *arXiv e-prints*, page arXiv:2005.14165, May 2020.

[6] Vanessa Buhrmester, David Münch, and Michael Arens. Analysis of Explainers of Black Box Deep Neural Networks for Computer Vision: A Survey. *arXiv e-prints*, page arXiv:1911.12116, November 2019.

[7] Rich Caruana. Multitask learning. *Machine Learning*, 28, 07 1997.

[8] Kuang-Hua Chang. Chapter 17 - design optimization. In Kuang-Hua Chang, editor, *e-Design*, pages 907–1000. Academic Press, Boston, 2015.

[9] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. GradNorm: Gradient Normalization for Adaptive Loss Balancing in Deep Multitask Networks. *arXiv e-prints*, page arXiv:1711.02257, November 2017.

[10] Wang Chi Cheung. Exploration-Exploitation Trade-off in Reinforcement Learning on Online Markov Decision Processes with Global Concave Rewards. *arXiv e-prints*, page arXiv:1905.06466, May 2019.

[11] Wojciech Marian Czarnecki, Simon Osindero, Max Jaderberg, Grzegorz Swirszcz, and Razvan Pascanu. Sobolev training for neural networks. In *Advances in Neural Information Processing Systems*, volume 2017-Decem, pages 4279–4288, 2017.

[12] Hamidreza Eivazi, Mojtaba Tahani, Philipp Schlatter, and Ricardo Vinuesa. Physics-informed neural networks for solving Reynolds-averaged Navier-Stokes equations. *arXiv e-prints*, page arXiv:2107.10711, July 2021.

[13] Ronen Eldan and Ohad Shamir. The Power of Depth for Feedforward Neural Networks. *arXiv e-prints*, page arXiv:1512.03965, December 2015.

[14] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. *arXiv e-prints*, page arXiv:1703.03400, March 2017.

[15] Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. Implicit Geometric Regularization for Learning Shapes. *arXiv e-prints*, page arXiv:2002.10099, February 2020.

[16] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *arXiv e-prints*, page arXiv:1502.05767, February 2015.

[17] Hongwei Guo, Xiaoying Zhuang, and Timon Rabczuk. A Deep Collocation Method for the Bending Analysis of Kirchhoff Plate. *arXiv e-prints*, page arXiv:2102.02617, February 2021.

[18] David Ha, Andrew Dai, and Quoc V. Le. HyperNetworks. *arXiv e-prints*, page arXiv:1609.09106, September 2016.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv e-prints*, page arXiv:1512.03385, December 2015.

[20] A. Ali Heydari, Craig A. Thompson, and Asif Mehmood. SoftAdapt: Techniques for Adaptive Loss Weighting of Neural Networks with Multi-Part Loss Functions. *arXiv e-prints*, page arXiv:1912.12355, December 2019.

[21] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Networks*, 3(5):551–560, 1990.

[22] Yu Huang and Yue Chen. Autonomous Driving with Deep Learning: A Survey of State-of-Art Technologies. *arXiv e-prints*, page arXiv:2006.06091, June 2020.

[23] T.J.R. Hughes. *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Dover Civil and Mechanical Engineering. Dover Publications, 2012.

[24] Ameya Jagtap and George Karniadakis. Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. *Communications in Computational Physics*, 28:2002–2041, 11 2020.

[25] John T. Katsikadelis, editor. *The Boundary Element Method for Engineers and Scientists*. Academic Press, Oxford, second edition edition, 2016.

[26] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv e-prints*, page arXiv:1412.6980, December 2014.

[27] M. A. Kraus and M. Drass. Artificial intelligence for structural glass engineering applications — overview, case studies and future potentials. *Glass Structures & Engineering*, 5(3):247–285, Nov 2020.

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 25, 01 2012.

[29] David Krueger, Chin-Wei Huang, Riashat Islam, Ryan Turner, Alexandre Lacoste, and Aaron Courville. Bayesian Hypernetworks. *arXiv e-prints*, page arXiv:1710.04759, October 2017.

[30] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[31] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

[32] Alex Nichol, Joshua Achiam, and John Schulman. On First-Order Meta-Learning Algorithms. *arXiv e-prints*, page arXiv:1803.02999, March 2018.

[33] Maziar Raissi, Paris Perdikaris, and George 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.

[34] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations. *arxiv.org*, 2017.

[35] F. Rosenblatt. The perceptron - a perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, Ithaca, New York, January 1957.

[36] Michael Ruchte and Josif Grabocka. Efficient multi-objective optimization for deep learning. *ArXiv*, abs/2103.13392, 2021.

[37] Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[38] Vincent Sitzmann, Julien N. P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit Neural Representations with Periodic Activation Functions. *arXiv e-prints*, page arXiv:2006.09661, June 2020.

[39] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations. *arXiv e-prints*, page arXiv:1906.01618, June 2019.

[40] Arnold Sommerfeld. *Partial differential equations in physics*. Academic press, 1949.

[41] Mingxing Tan and Quoc V. Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *arXiv e-prints*, page arXiv:1905.11946, May 2019.

[42] Matus Telgarsky. Benefits of depth in neural networks. *arXiv e-prints*, page arXiv:1602.04485, February 2016.

[43] Laurens van der Maaten and Geoffrey Hinton. Viualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 11 2008.

[44] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient pathologies in physics-informed neural networks. *arXiv e-prints*, page arXiv:2001.04536, January 2020.

[45] Huasen Wu, Xueying Guo, and Xin Liu. Adaptive Exploration-Exploitation Tradeoff for Opportunistic Bandits. *arXiv e-prints*, page arXiv:1709.04004, September 2017.

[46] Yongqin Xian, Bernt Schiele, and Zeynep Akata. Zero-Shot Learning – The Good, the Bad and the Ugly. *arXiv e-prints*, page arXiv:1703.04394, March 2017.

[47] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A Comprehensive Survey on Transfer Learning. *arXiv e-prints*, page arXiv:1911.02685, November 2019.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Interactive Design Lab
Prof. Dr. Olga Sorkine-Hornung

Title of work:

Physics-Informed Neural Networks for Structural Concrete Engineering

Thesis type and date:

Master's Thesis, September 2021

Supervision:

Dr. Michael Kraus
Prof. Dr. Olga Sorkine-Hornung
Prof. Dr. Walter Kaufmann

Student:

Name:	Rafael Bischof
E-mail:	rabischof@student.ethz.ch
Legi-Nr.:	16-816-654
Semester:	FS21

Statement regarding plagiarism:

By signing this statement, I affirm that I have read and signed the Declaration of Originality, independently produced this paper, and adhered to the general practice of source citation in this subject-area.

Declaration of Originality:

`http://www.ethz.ch/faculty/exams/plagiarism/confirmation_en.pdf`

Zürich, 23. 11. 2021:
Rafael Bischof

[illegible]