Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Wilhelm-Schickard-Institut für Informatik

Master Thesis Machine Learning

Uncertainty Quantification in Physics-informed Neural Networks using Laplace Approximations

Thomas Albrecht

08.02.2023

Reviewers

Prof. Dr. Philipp Hennig
(Methods of Machine Learning)
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen

Prof. Dr. Jakob Macke
(Machine Learning in Science)
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen

Albrecht, Thomas:
Uncertainty Quantification in Physics-informed Neural Networks using Laplace Approximations
Master Thesis Machine Learning
Eberhard Karls Universität Tübingen
Thesis period: 08.08.2022 - 08.02.2023

Abstract

Physics-informed neural networks (PINN) have shown great success in multiple applications in solving PDE problems [Karniadakis et al., 2021] and have become a well established part scientific machine learning. One ongoing area of research is quantifying the different sources of uncertainty arising in different scientific machine learning methods. This work investigated whether we can capture the numerical error of a PINN solution for a PDE problem using a probabilistic approch in a setting with noise free input data. To make the computations tractable and efficient we used Laplace approximations to infer the posterior and get uncertainties over the outputs of our network. We begin by modifying the PINN approach [Raissi et al., 2019] and constructing a probabilistic formulation of the PINN problem. For that we defined a corresponding prior over the weights and likelihood functions. But computing the posterior $p(\theta \mid \mathcal{D})$ analytically is usually intractable. We employ a Laplace approximation to efficiently approximate the posterior by a Gaussian around the mode of our posterior θ_{MAP}. We improve our approximation by tuning the hyperparameters of our probabilistic model (the variances of the prior and likelihood) using the Laplace-approximated marginal likelihood. To get uncertainty estimates over the PINN solution we approximate the posterior over the network outputs $p(u(x) \mid x, \mathcal{D})$ by linearizing the neural network around our MAP estimate θ_{MAP}.

We then show that out method is able to accurately solve a PDE problem and analyze the uncertainties returned by our posterior. The uncertainties are reasonably well calibrated and capture the numerical error of our method.

Zusammenfassung

So genannte physics-informed neural networks (PINNs) haben in der nahen Vergangenheit in verschiedenen Anwendungen großen Erfolg im Lösen partieller Differentialgleichungen(PDE) gezeigt [Karniadakis et al., 2021]. Ein laufendes Forschungsgebiet ist die Quantifizierung der verschiedenen Quellen von Unsicherheiten, die in verschiedenen Methoden im Forschungsbereich von SciML auftreten. Diese Arbeit untersucht die Möglichkeit, den numerischen Fehler einer PINN-Lösung für ein PDE-Problem mit Hilfe eines probabilistischen Ansatzes in einer Umgebung mit rauschfreien Eingabedaten zu bestimmen. Dieser probabilistische Ansatz ist oft analytisch nicht lösbar oder sehr rechenintensiv. Um die Berechnung der a-posteriori-Wahrscheinlichkeit effizient zu gestalten, haben wir Laplace-Approximationen verwendet, um diese Wahrscheinlichkeit zu schätzen und Unsicherheiten über die Ausgaben unseres Netzwerks zu erhalten. Zunächst formulieren wir den PINN-Ansatz [Raissi et al., 2019] probabilistisch um. Dazu definieren wir eine entsprechende a-priori-Wahrscheinlichkeit über die Gewichte und die Likelihood-Funktionen. Die Berechnung der a-posteriori-Wahrscheinlichkeit $p(\theta \mid \mathcal{D})$ ist jedoch in der Regel analytisch nicht lösbar. Wir verwenden eine Laplace-Approximation, um die a-posteriori-Wahrscheinlichkeit durch eine Normalverteilung um den Modus dieser a-posteriori-Wahrscheinlichkeit θ_{MAP} effizient zu approximieren. Um Unsicherheiten über die PINN-Lösung zu erhalten, approximieren wir die Wahrscheinlichkeit über die Ausgaben $p(u(x) \mid x, \mathcal{D})$ durch Linearisierung des neuronalen Netzes um unsere MAP-Schätzung θ_{MAP}.

In unseren Experimenten zeigen wir, dass unsere Methode in der Lage ist, ein PDE-Problem gut zu lösen, und analysieren die Unsicherheiten, die von der Methode ausgegeben werden.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Partial differential equations (PDEs) have played an important role in modelling and predicting the behaviour of many complex phenomena occuring in the real world. They can be used to describe how physical, biological, mathematical or engineering systems change over time and space. In physics, they are used to model the behaviour of fluids, e.g. in the famous Navier-Stokes equations. In engineering and material science they can be used to model and describe structures and to analyze their properties. There are many different approaches to solving PDEs, using both analytical methods and numerical algorithms. PDE problems can often be very difficult to solve, as they involve multiple variables and partial derivatives and often do not have a closed-form solution.

In recent years researchers have explored many approaches to modeling and forecasting physical systems using various machine learning methods. In this field of scientific machine learning, methods attempt to incorporate knowledge of physical systems into e.g. the learning process of a neural network. One approach to incorporate prior knowledge of the physical principles of a dynamical system into deep learning algorithms are so called physics-informed neural networks (PINNs) [Raissi et al., 2019]. Here the underlying dynamics of a PDE are encoded into the neural network training process using a modified loss term and automatic-differentiation. This loss term then enforces the physical principles at a chosen set of collocation points. PINNs have been shown to have perform well on various types of PDE problems [Karniadakis et al., 2021]. The PINN method can be thought of as a numerical algorithm using collocation points. It returns an approximate solution to a true underlying solution of the PDE problem. Usually the true underlying solution to a PDE problem is not known. Thus, in addition to the solution, we may also be interested in the quality of our solution, i.e. how

certain we can be about our solution. Sources that influence the uncertainty of the solution in scientific machine learning are the architecture of the neural network, the choice of hyperparameters, the chosen discretization of the data grid, as well as the uncertainty inference algorithm itself [Psaros et al., 2022]. But depending on the setting, the uncertainty might also be influenced by noisy data or stochasticity in the PDE. Traditional numerical solvers for partial differential equations try to find a solution to the problem, but it is often just an approximation with an unknown numerical error. To apply the numerical solution (as the true underlying solution to the PDE problem is usually unknown) an estimate or guarantee of the magnitude and distribution of the error is useful. In a probabilistic model we can represent uncertain quantities like the measurements and by extension the measurement errors as random variables and probability distributions. This approach is similar to the one usually employed in probabilistic numerical modeling. Probabilistic numerics is a field of research that tries to pose numerical algorithms and methods as problems of Bayesian inference [Hennig et al., 2022]. The PINN approach is also comparable to a collocation based numerical algorithm for solving PDEs. In this thesis we introduce a probabilistic approach for PINNs in a setting with noise free inputs and analyze the resulting uncertainty estimates. We define a probabilistic model for the PINN problem and use Laplace approximations to make the computation of the uncertainty estimates tractable. We chose Laplace approximations to infer the posterior, because they give good uncertainty estimates while still being computationally efficient. They also do not infer with the neural network training process and can be applied to all network architectures. In addition to that, in contrast to other posterior inference algorithms, we are able to approximate the full covariance of the posterior, and can subsequently work with this covariance matrix. While other work was mostly concerned with uncertainty quantification for noisy observations or stochastic partial differential equations [Yang et al., 2020, Zhang et al., 2019, Psaros et al., 2022], in this work we instead want to analyze the uncertainty estimates in a with noise-free inputs and compare those estimates to the numerical error of the PINN solution. Next we introduce some of the past approaches to quantify the uncertainty in PINNs, both with Laplace approximations and other posterior inference methods.

Related Work

In [Yang et al., 2021], the authors introduce Bayesian PINNs (B-PINNs) by representing the PINN formulation in the Bayesian framework and defining an appropriate prior and likelihood function. They then explore different posterior sampling approaches to sample from the Bayesian posterior. In that work the setting is different, as it mainly considers noisy inputs as sources of uncertainty. The meth-

ods for posterior inference are also different, in the work they used Hamiltonian Monte-Carlo and variational inference to estimate the uncertainties.

In the study [Psaros et al., 2022] the authors analyzed a wide range of posterior inference methods (including Laplace approximations) for a wide range of scientific machine learning settings. They considered many cases of uncertainty, including noisy input data and stochastic PDEs. But they did not analyze the noise free input setting we use in this work here. Their posterior inference method using Laplace approximations is also less refined.

While [Izzatullah et al., 2022] use Laplace approximations to make the computations in Bayesian PINNs fast and efficient, the work only tries to solve one specific PDE problem. And while their method is similar to ours, they did not analyze the performance of the method itself.

1.2 Outline

Chapter 2 introduces the two underlying concepts used in this work, physics-informed neural networks and Laplace approximation. Next, Chapter 3 introduces the method we have developed. Here we define the general problem formulation and the underlying probabilistic model we constructed. The evaluation of the our methods and the empirical results are presented in Chapter 4. The experiments not only include the general evaluation of the effectiveness of the method but also investigate the behaviour of the method for different models and collocation grids. Finally we discuss the our findings and possible extensions in Chapter 5.

Chapter 2

Foundations

In this work we want to quantify the numerical error of a PDE solution returned by a physics-informed neural network. To this end we apply Laplace approximations to physics-informed neural networks in setting with noise-free inputs. We then analyse the uncertainty estimates of the posterior predictive distribution. In our setting, this posterior should reflect other sources of uncertainty, coming from model misspecification in the form of the architecture and hyperparameters of the model, as well as the discretization used in the method itself.

This chapter first introduces and defines partial differential equations in Section 2.1. The rest of the chapter then describes the two foundations this work is based on, physics-informed neural networks [Raissi et al., 2019] in Section 2.2 and Laplace approximations and some extensions in Section 2.3.

2.1 Partial Differential Equations

Partial differential equations (PDEs) are mathematical equations that involve at least two variables and some of their partial derivatives. In the PDE literature partial derivatives are usually denoted using a subscript notation:

$$u_x := \frac{\partial u(\cdot)}{\partial x}, \tag{2.1}$$

where $u(\cdot)$ denotes the underlying solution function.

A general PDE is an equation of the form

$$\mathcal{N}\left(x, t, u, u_t, u_{x_1}, u_{x_2}, u_{x_1 x_2}, \dots\right) = 0, \quad (x, t) \in \Omega, \tag{2.2}$$

where $u = u(t, x)\colon \Omega \to \mathbb{R}^m$ is the unknown function that we are trying to solve for, and the partial derivatives u_x, u_t, u_{xx}, etc. represent the rates of change of u with respect to the variables x and t in the spatio-temporal domain Ω (see Borthwick [2017]). The operator $\mathcal{N}$ represents the specific form of the PDE, which may involve constants or other terms that depend on the problem at hand. The operator is often also denoted in the following way:

$$\mathcal{N}[u](t, x). \tag{2.3}$$

We need to further constraint the problem by imposing boundary conditions or initial conditions (or both). They restrict the PDE at the boundary of the domain or impose the starting configuration. Boundary conditions are commonly given in the following form:

$$au + b\frac{\partial u}{\partial x} = c, \forall x \in \partial\Omega. \tag{2.4}$$

They are called Neumann (a=0), Dirichlet (b=0), or Robin (c=0) boundary conditions Jaun et al. [1999]. Another common form are periodic boundary conditions, given like

$$u(x_l) = u(x_r), \{x_l, x_r\} \in \partial\Omega. \tag{2.5}$$

Sometimes initial conditions need to be imposed:

$$u(t = 0, x) = u_0(x), \forall x \in \Omega. \tag{2.6}$$

Solving the equation entails finding a solution function $u(\cdot, \cdot)$ that satisfies the equation 2.2 and the initial or boundary conditions, if any are given. We call the combination of a PDE and its corresponding boundary and/or initial conditions a PDE problem.

Example PDE problem We introduce a simple illustrative example PDE problem, adapted from [Barba and Forsyth, 2019]: One fundamental PDE problem is the heat equation, in two-dimensions it is defined in the following way:

$$\frac{\partial u}{\partial t} - (\nu \frac{\partial^2 u}{\partial x_1^2} + \nu \frac{\partial^2 u}{\partial x_2^2}) = 0 \tag{2.7}$$

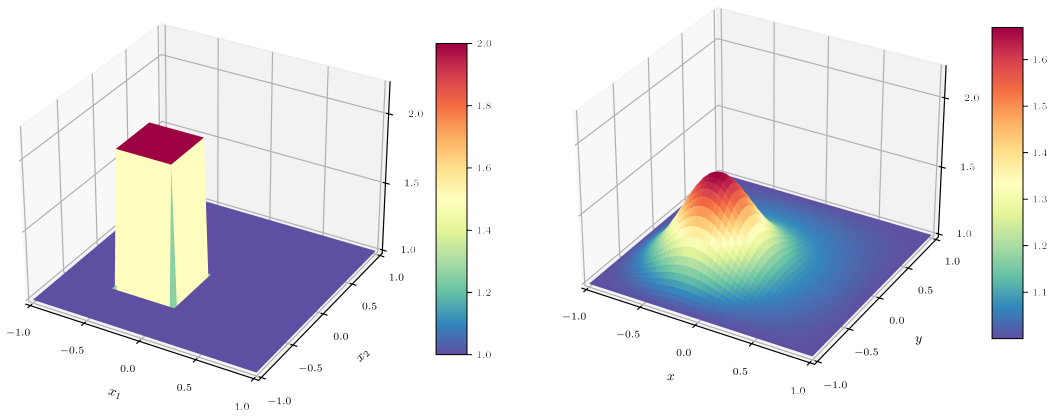
The heat equation (also known as diffusion equation) models how heat transfers and spreads in different materials and is important in many fields of science, including physics, engineering, mathematics, and finance. In addition to the partial differential equation we also impose Dirichlet boundary conditions:

$$u(t, x) = 1 \text{ for } x_1, x_2 \in \{-1, 1\} \tag{2.8}$$

and the initial condition

$$u(t = 0, x_1, x_2) = \begin{cases} 2 \text{ for } x_1, x_2 \in [-0.5, 0] \\ 1 \text{ else} \end{cases} \tag{2.9}$$

In Figure 2.1 we show both the initial condition of this PDE problem, as well as a solution obtained after running a numerical solver to simulate 50 time steps.



(a) Initial Condition

(b) Solution after iterating simple numerical solver for 50 time-steps

Figure 2.1: Simple diffusion equation in 2D

Partial differential equations are present in many applications and have been thoroughly studied (see e.g. Evans [2010], Boyce et al. [2021]). Many approaches to solve PDE problems analytically or to find properties of possible solutions have been proposed. But those analytical approaches for solving PDEs are often not

feasible or tractable, and there are multiple different numerical approaches to solve them approximately [Morton and Mayers, 2005]. With the emergence of the field of scientific machine learning, several approaches have been developed to leverage the recent success of deep learning methods to solve PDEs. One such approach, which tries to harness the power of neural networks as universal function approximators to solve PDE problems, are so called physics-informed neural networks.

2.2 Physics-informed Neural Networks

The main idea of physics-informed neural networks [Raissi et al., 2019] is to incorporate knowledge about the dynamics of a system (i.e. the partial differential equation) in addition to training data (e.g. measurements). The dynamics are incorporated into the training of the neural network in the form of an additional loss term, which can be thought of as a soft constraint. This enables us to learn a potentially high-dimensional nonlinear function despite possibly using only a few input data points. Notably the PDE constraints are imposed at a set of collocation points that can be selected freely, not unlike some traditional approaches to numerically solve PDE problems.

Problem setup

We first introduce the problem setup for the so called forward problem in the "continuous time model" case as described by Raissi et al. [2019]. The goal is to compute a solution $u(t, x)$ to a deterministic PDE problem:

$$u_t + \mathcal{N}[u] = 0,\ x \in \Omega,\ t \in [0, T]. \tag{2.10}$$

Here $\mathcal{N}$ denotes a general (non-)linear differential operator, $u(t, x)$ is the solution function to the PDE and $\Omega \subset \mathbb{R}^m$ the problem domain. Some PDE problems will also require corresponding boundary and initial conditions to be fullfilled, which are given separately depending on the problem at hand. A more general problem setup will be defined in 3.1. Our goal is to infer a solution $u(t, x)$ that satisfies Equation 2.10 and the boundary or initial conditions, if any are given. We define $f(t, x)$ as the left-hand side in Equation 2.10:

$$f(t, x) := u_t + \mathcal{N}[u]. \tag{2.11}$$

In the PINN aproach we approximate the latent solution to the PDE problem $u(t, x)$ with a deep neural network $u_\theta(t, x)$ with parameters θ. We now call $f_\theta(t, x)$ the physics-informed neural network, as by applying the chain rule and differentiating $u_\theta(t, x)$ using automatic differentiation with regard to the inputs, we again

obtain a neural network that shares its parameters with $u_\theta(t, x)$. By constructing an adequate loss function we can learn the shared parameters θ of both neural networks that minimize this loss simultaneously. In the actual implementation we will employ only one neural network $u_\theta(t, x)$ and train it, the second network $f_\theta(t, x)$ is only encoded in the loss term. We can learn the shared parameters θ of the two neural networks by minimizing the following sum of two mean-squared error loss terms:

$$MSE = MSE_u + MSE_f \tag{2.12}$$

with

$$MSE_u := \frac{1}{N_u} \sum_{i=1}^{N_u} |u(t_u^i, x_u^i) - u^i|^2 \tag{2.13}$$

and similarly

$$MSE_f := \frac{1}{N_f} \sum_{i=1}^{N_f} |f(t_f^i, x_f^i) - 0|^2 = \frac{1}{N_f} \sum_{i=1}^{N_f} |f(t_f^i, x_f^i)|^2. \tag{2.14}$$

Here $\{t_u^i, x_u^i, u^i\}$ denote boundary training data and also initial training data (if available) and $\{t_f^i, x_f^i\}$ denote the collocation points employed on $f_\theta(t, x)$. Note that MSE_f is penalizing the deviation of f_θ from zero, as enforced by the PDE in equation 2.10. We can think of the loss MSE_u as enforcing the target values u^i at the boundary and initial points, while the loss MSE_f enforces the dynamics of the PDE system or tries to limit the set of possible solutions to the ones satisfying the PDE constraints at the collocation points. This combined loss term can now be optimized using a gradient-based optimizer (neural-network training).

After learning, the resulting neural network $u_\theta(t, x)$, our surrogate to the solution function $u(t, x)$ can easily be queried with unseen data points and interpolates on the whole domain.

2.3 Laplace Approximation for Neural Networks

In recent years deep neural networks have shown great potential for various different learning tasks. But in the most straightforward deep learning approach we can not know how reliable the predictions of the model are. Bayesian neural networks try to mitigate this by also estimating the uncertainty over the parameters of the network. The following introduction is primarily derived from the exposition in MacKay et al. [2003] and Daxberger et al. [2021].

A neural network can be defined as a function $f_\theta(x)$ that maps an input x to an output y. This mapping is non-linear and the function is parametrized by weights θ. Standard neural network training corresponds to maximizing the likelihood $p(\mathcal{D} \mid \theta)$ at a set of data points $\mathcal{D}$. In addition we also define a prior $p(\theta)$ over the weights. By Bayes rule this yields the posterior

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})}. \tag{2.15}$$

Computing the evidence term $p(\mathcal{D}) = \int_\theta p(\mathcal{D}|\theta)p(\theta)d\theta$ is usually intractable and we have to find an approximate solution. One simple approximation is maximum a-posteriori (MAP) estimation with the data likelihoods over the dataset $\mathcal{D}$ and a prior over the parameters θ. The maximum a-posteriori estimate of a function is defined as:

$$\theta_{MAP} = \underset{\theta}{\operatorname{argmax}} \ \log p(\theta \mid \mathcal{D}) = \underset{\theta}{\operatorname{argmax}} \ \log p(\mathcal{D} \mid \theta)p(\theta). \tag{2.16}$$

This estimate returns the single most likely parameter θ for the dataset and prior. But this does not give us uncertainty estimates. Several methods have been developed to approximate the full posterior distribution. A common approach to do this are Markov chain Monte Carlo (MCMC) methods. In these methods we want to generate multiple Monte Carlo samples of the parameter θ using a Markov chain with $p(\theta \mid \mathcal{D})$ as the stationary distribution. MCMC methods can be relatively easy to implement and are possible to apply to any network architecture and shape of target distribution. However, they can be computationally expensive, and working with the full posterior distribution might not be straightforward or possible. Another popular approach is variational inference. Here we approximate the posterior $p(\theta \mid \mathcal{D})$ with another parameterized distribution $q(\theta \mid \mathcal{D})$ (this is often a Gaussian). We learn the parameters of this distribution by optimizing them wrt. the marginal likelihood to try to match the distributions. This process is iterative and might be computationally expensive, but we are able to get a good approximation.

Another approach are Laplace approximations [MacKay, 1992]. The goal of using the Laplace approximation is to make the computation of the full posterior

over the parameters θ tractable, by approximating the target distribution (the posterior $p(\theta \mid \mathcal{D})$) with a Gaussian approximate posterior distribution $q(\theta \mid \mathcal{D})$. Some key advantages of the Laplace approximation are its ease of implementation and its ability to approximate a wide range of distributions. In addition, it can be applied post-hoc, after the neural network has been fully trained and can therefore be applied to pre-trained models. Compared to sampling-based approaches or variational inference, the Laplace approximation is computationally lightweight and can be implemented without modifying the neural network training procedure or model architecture. The Laplace approximation for approximating the posterior also returns an approximate full covariance matrix, which allows for a simpler Bayesian interpretation of the method. Since the approximate posterior is Gaussian, many computations reduce to linear algebra and we can analyze and try to interpret the uncertainties by examining the covariance of the Gaussian. A drawback of the Laplace approximation is the potentially bad estimates, since we are always only approximating the posterior with a Gaussian distribution. For some target posterior distributions this local approximation can be arbitrarily bad.

To apply the Laplace approximation we first train a deep neural network to obtain a maximum a-posteriori estimate. We then set the mean of the approximate posterior Gaussian density as the mode of our unnormalized posterior distribution θ_{MAP}, this way the approximate posterior will match the value of the target distribution at its mode.

We introduce short-hand notation for the numerator and denominator of our posterior:

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{\int p(\mathcal{D}|\theta)p(\theta)d\theta} =: \frac{h(\theta)}{Z}. \tag{2.17}$$

Our first step is to approximate $\log h(\theta)$ by a second-order Taylor expansion around θ_{MAP}:

$$\log h(\theta) \approx \log h(\theta_{MAP}) - \frac{1}{2}(\theta - \theta_{MAP})^T \Lambda (\theta - \theta_{MAP}), \tag{2.18}$$

with $\Lambda := -\nabla^2 \log h(\theta)|_{\theta_{MAP}}$ Note that the first order term $-\nabla \log h(\theta)(\theta - \theta_{MAP})$ is equal to zero at the mode θ_{MAP}.

Next we can approximate the evidence term by applying our approximation

for $h(\theta)$ from equation 2.18 to the definition of Z:

$$Z = \int p(\mathcal{D}|\theta)p(\theta)d\theta = \int \exp(\log h(\theta))d\theta \tag{2.19}$$

$$\approx \exp(\log h(\theta_{MAP})) \int \exp\left(-\frac{1}{2}(\theta - \theta_{MAP})^T \Lambda (\theta - \theta_{MAP})\right) d\theta \tag{2.20}$$

$$= h(\theta_{MAP}) \frac{(2\pi)^{\frac{d}{2}}}{(\det \Lambda)^{\frac{1}{2}}}. \tag{2.21}$$

We can now combine both of those approximations and write down the full approximate posterior $q(\theta \mid \mathcal{D})$:

$$p(\theta \mid \mathcal{D}) = \frac{1}{Z} h(\theta) \approx \frac{(\det \Lambda)^{\frac{1}{2}}}{(2\pi)^{\frac{d}{2}}} \exp\left(-\frac{1}{2}(\theta - \theta_{MAP})^T \Lambda (\theta - \theta_{MAP})\right) =: q(\theta \mid \mathcal{D}). \tag{2.22}$$

This approximate posterior has the form of a Gaussian density around the mode of our target distribution:

$$q(\theta \mid \mathcal{D}) = \mathcal{N}(\theta_{MAP}, \Lambda^{-1}). \tag{2.23}$$

The Laplace approximation thus approximates our target distribution $p(\theta \mid \mathcal{D})$ with a Gaussian, defined by the mode given by the MAP estimate θ_{MAP} obtained by training our neural network, and the covariance matrix given by the inverse Hessian Λ^{-1} computed at the MAP estimate.

2.3.1 The Hessian and Hessian approximations

To approximate the posterior using a Laplace approximation we need to compute the Hessian of the prior and likelihood:

$$\Lambda := -\nabla^2_\theta \log(p(\mathcal{D} \mid \theta)p(\theta)) = \nabla^2_\theta \mathcal{L}(\mathcal{D}, \theta). \tag{2.24}$$

We have to compute this Hessian with regard to all parameters θ, which in principle is possible using automatic differentiation. But computing this Hessian and its determinant is usually computationally expensive for many neural network architectures as we need to compute the second derivative with regard to all network parameters. Instead we can approximate the Hessian in various different ways:

Fisher Information Matrix/Gauss-Newton Matrix A commonly used approximation of the Hessian Λ is the so called Fisher information matrix [Daxberger

et al., 2021]. Here we approximate the Hessian using only first-order information, i.e. the first-derivative of our loss terms.

$$F := \sum_{n=1}^{N} \mathbb{E}_{\hat{y}\sim p(y|f_\theta(x_n))} \left[(\nabla_\theta \log p(\hat{y} \mid f_\theta(x_n))|_{\theta_{MAP}})(\nabla_\theta \log p(\hat{y} \mid f_\theta(x_n))|_{\theta_{MAP}})^T \right]. \tag{2.25}$$

We can also use the generalized Gauss-Newton matrix to approximate the Hessian Λ:

$$G := \sum_{n=1}^{N} J(x_n) \left(\nabla_f^2 \log p(y_n \mid f)|_{f=f_{\theta_{MAP}}(x_n)} \right) J(x_n)^T, \tag{2.26}$$

with the Jacobian of the neural network $J(x_n) := \nabla_\theta f_\theta(x_n)|_{\theta_{MAP}}$. For many commonly used likelihood functions those two approximations are equivalent.

If even this approximation is too computationally expensive we can use a block-diagonal factorization of the Hessian like Kronecker-factored approximate curvature [Ritter et al., 2018] or even an extremely simplified diagonal approximation to the Hessian [Daxberger et al., 2021].

Last-Layer Laplace Both in the case of the full Hessian and the Fisher information matrix/Gauss-Newton matrix we still need to invert a square matrix of the size of the parameters. If the neural network has many parameters, i.e. if the dimensionality of θ is large, those approximations might also be computationally expensive. As an alternative we can make the computation tractable by only considering uncertainty on a subset of the models weights, in particular only the weights in the last-layer of the model, while using the MAP estimate for the other weights [Kristiadi et al., 2020]. This can severely reduce the number of parameters we have to consider. If the full Hessian is still not feasible when estimating the uncertainty over the weights in the last layer we again can use one of the approximations discussed above.

2.3.2 Hyperparameter optimization

Instead of just setting the variances to a fixed value a-priori, our method actually allows us to choose hyperparameters that improve the performance of our method. To improve the accuracy of the approximation it can be useful to optimize the hyperparameters of the model [Immer et al., 2021a].

Suppose we have already obtained a θ_{MAP} estimate by training a neural network, and have already calculated our first approximation $q(\theta \mid \mathcal{D})$, with a corresponding Hessian Λ^{-1}. In our model we have some hyperparameters that we have implicitly assumed to be chosen in some way, in particular we can often choose

the covariance of our prior and likelihoods. We optimize these hyperparameters post-hoc, without retraining the neural network $u_\theta(x)$. One way to do this is to maximize the marginal likelihood of the Laplace-approximated posterior. One big advantage of this method is that it doesn't require additional validation data or a hold-out dataset [Immer et al., 2021a].

The general marginal likelihood of our training data given a model $\mathcal{M}$ is defined as follows:

$$p(\mathcal{D} \mid \mathcal{M}) = \int p(u(x) \mid \mathcal{D}, \mathcal{M}) p(\theta \mid \mathcal{M}) \, d\theta, \tag{2.27}$$

where $\mathcal{M}$ encompasses all model hyperparameters (in particular the variance of our prior and likelihood) and potentially other parts of the model such as the architecture of the neural network.

We want to find the model hyperparameters that maximize the probability of our training data:

$$\mathcal{M}_* = \underset{\mathcal{M}}{\operatorname{argmax}} \; p(\mathcal{D} \mid \mathcal{M}) \tag{2.28}$$

The Laplace-approximated (log-)marginal likelihood [Immer et al., 2021a] is given by:

$$\log p(\mathcal{D} \mid \mathcal{M}) \approx \log q(\mathcal{D} \mid \mathcal{M}) := \log p(\mathcal{D} \mid \mathcal{M}) - \frac{1}{2} \log |\frac{1}{2\pi} \mathbf{H}_{\theta_{MAP}}| \tag{2.29}$$

$$= \log h(\theta_{MAP}) - \frac{1}{2} (\theta - \theta_{MAP})^T \Lambda (\theta - \theta_{MAP}) - \frac{1}{2} \log |\frac{1}{2\pi} \mathbf{H}_{\theta_{MAP}}|. \tag{2.30}$$

The log-determinant of the Hessian follows from inserting the Laplace approximated joint distribution $p(\mathcal{D}, \theta \mid \mathcal{M})$ back into the marginal likelihood in Equation 2.27 and also serves as a regularization term to favor a Hessian with smaller eigenvalues.

We can now optimize the marginal log-likelihood with a gradient-based optimizer like Adam [Kingma and Ba, 2014] until convergence. We do not need to compute or invert the Hessian in each iteration, but we do need to compute a log-determinant in each iteration of our optimization. This way the optimization of the hyperparameters is still cheap computationally and needs to be done only once. After computing the hyperparameters we can infer a predictive distribution to query the network for new data points and get uncertainty estimates for them.

2.4 Predictive Distribution

Now that we approximated $p(\theta \mid \mathcal{D})$ and obtained our Laplace-approximated posterior $q(\theta \mid \mathcal{D})$, we next want to approximate the predictive distribution

$p(u(x) \mid x, \mathcal{D})$ for potentially new and unseen data-points x.

A simple way to approximate the posterior predictive $p(u \mid x, \mathcal{D})$ is to integrate over K Monte Carlo samples from our approximated posterior $q(\theta \mid \mathcal{D})$:

$$p(u \mid x, \mathcal{D}) = \mathbb{E}_{q(\theta)}\left[p(u \mid u_\theta(x)\right] \tag{2.31}$$

$$= \int_\theta p(u \mid u_\theta(x))\, \mathcal{N}(\theta; \theta_{MAP}, \Lambda^{-1}) d\theta \tag{2.32}$$

$$\approx \frac{1}{K} \sum_{i=1}^{K} p(u \mid u_{\theta_i}(x)), \quad \theta_i \sim \mathcal{N}(\theta; \theta_{MAP}, \Lambda^{-1}). \tag{2.33}$$

This approach is common, as it can be applied in many cases and is simple to implement. But this approach may be computationally expensive and might require many samples to return an accurate posterior.

Alternatively we can approximate the posterior with a linearized model and exploit some of the additional structure we obtained from our choice of approximation [Immer et al., 2021b]: We first linearize the neural network ouput around θ_{MAP} and obtain the following approximation:

$$u_\theta(x) \approx u_{\theta_{MAP}}(x) + J(x)^T(\theta - \theta_{MAP}), \tag{2.34}$$

where $J(x) := \nabla_\theta u_\theta(x)\big|_{\theta_{MAP}}$ is the Jacobian matrix of the neural network outputs.

We want to find the marginal predictive distribution

$$p(u(x) \mid x, \mathcal{D}) = \int p(u(x) \mid x, \theta) p(\theta \mid \mathcal{D}) d\theta. \tag{2.35}$$

By using the approximation in Equation 2.34 we can compute this integral in closed form:

$$p(u(x) \mid \mathcal{D}) = \int \delta(u(x) - u_\theta(x)) q(\theta \mid \mathcal{D}) d\theta \tag{2.36}$$

$$\approx \mathcal{N}(u(x) \mid u_{\theta_{MAP}}(x), J(x)^T \Lambda^{-1} J(x)) \tag{2.37}$$

Because the approximate posterior is Gaussian the marginal distribution over the output is again Gaussian.

We have now shown how to approximated a posterior over the outputs $p(u(x) \mid x, \mathcal{D})$ for a general deep neural network. With this posterior we can estimate the uncertainty of our network predictions along the whole problem domain.

Chapter 3

Method

Our goal is to efficiently compute the uncertainty of the predictive distribution over the network outputs $p(u(x) \mid \mathcal{D})$. We can then compare the uncertainty estimate of the predictive distribution with the numerical error of the PINN solution. Ideally we want to reflect the magnitude and distribution of the numerical error of the PINN solution in the posterior uncertainty estimate. Our first step now is to rephrase the PINN problem statement (2.10) in a probabilistic way. With this probabilistic model we can then infer the predictive uncertainty by estimating the posterior predictive distribution. We employ a Laplace approximation to efficiently estimate the posterior, as computing it exactly is usually not tractable.

3.1 Problem Statement

We consider a general partial differential equation of the following form:

$$\mathcal{F}[u](x) = 0, \quad x \in \Omega, \tag{3.1}$$
$$\mathcal{B}[u](x) = 0, \quad x \in \partial\Omega, \tag{3.2}$$

where $u\colon \Omega \to \mathbb{R}^m$ denotes an arbitrary solution function to the PDE, $\Omega \in \mathbb{R}^d$ specifies a bounded domain and $\partial\Omega$ denotes the boundary of the domain. $\mathcal{F}$ is a general differential operator and $\mathcal{B}$ is a boundary condition operator. The goal is to find a solution function $u(x)$ that satisfies the two operators on the given domain. In the following we use PINNs to solve the PDE problem, similar to a numerical solver, by using a neural network $u_\theta(x)$ with parameters θ to learn a solution function that satisfies the PDE dynamics given by the operators in Equations 3.1 and 3.2 at a discrete grid of collocation points $\{x_i^f\}_{i=1}^{N_f}$ and $\{x_i^b\}_{i=1}^{N_b}$.

Enforcing the operators at the collocation points amounts to the following:

$$\begin{aligned} \mathcal{F}[u_\theta](x_i^f) &= 0, \\ \mathcal{B}[u_\theta](x_i^b) &= 0. \end{aligned} \tag{3.3}$$

The discretization can be chosen arbitrarily, we can select the points both on the boundary and inside the domain at which to enforce the corresponding operators. This approach is very flexible and similar to some traditional numerical methods for solving PDE problems. But a good discretization and a sufficient number of collocation points is important for a good approximation to the PDE solution. With this approach it is also possible to add more collocation points (e.g. in areas of the domain where the estimated error is high) and adaptively refine the collocation grid this way.

The problem setup introduced here is a little more general than the one introduced in Section 2.2 as the formulation of the differential operator $\mathcal{F}[u](x)$ is less restrictive. We also explicitly include the initial and boundary conditions in the problem statement.

3.2 Overview of the Method

Next we give a short overview of the full method and how it corresponds to the one given in Raissi et al. [2019].

In this work, in addition to approximating the solution function $u(x)$ we are also interested in computing a posterior distribution on the outputs of the solution function $u(x)$, given the PDE and boundary information $\mathcal{D}$:

$$p(u(x) \mid \mathcal{D}). \tag{3.4}$$

In Section 3.4 we will introduce in more detail the way we construct this information from the operators $\mathcal{F}$ and $\mathcal{B}$.

This posterior over the outputs enables us to estimate the uncertainty at the outputs at a point x. Ideally, we can construct a posterior distribution that accurately captures the uncertainty of the PINN solution. In the setting we consider we have no noise on the observations and no stochasticity in the PDE, so the posterior should reflect other sources of uncertainty. In particular we are interested in how the estimated uncertainty reflects the numerical error of the PINN solution. Sources that influence the error of the PINN solution are the choice of hyperparameters, the architecture of the neural network, the chosen discretization of the data grid, as well as the posterior inference algorithm itself [Psaros et al., 2022].

To compute the PINN solution the neural network is trained similarly to the approach used in the original PINN paper. In our implementation of the method

we compute the derived neural networks $\mathcal{F}[u_\theta](x)$ and $\mathcal{B}[u_\theta](x)$ using an automatic differentiation framework and train a neural network with the usual deep learning tools. We will use a short-hand notation to denote those neural networks:

$$f_\theta(x) := \mathcal{F}[u_\theta](x), \quad b_\theta(x) := \mathcal{B}[u_\theta](x). \tag{3.5}$$

After training we want to compute the posterior over the parameters $p(\theta \mid \mathcal{D})$. We use Bayes' theorem:

$$p(\theta \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \theta)p(\theta)}{p(\mathcal{D})}. \tag{3.6}$$

Since computing the posterior is usually intractable we approximate this posterior using a Laplace approximation (see Section 2.3). Next we can infer $p(u(x) \mid x, \mathcal{D})$ using the Laplace-approximated posterior $q(\theta \mid \mathcal{D})$ to compute uncertainty estimates on the outputs of the predictive of the solution function. To this end we then linearize the neural network and approximate a Gaussian posterior $p(u(x) \mid x, \mathcal{D})$. This way the distribution over the network outputs is again Gaussian and we can get variance values for each neural network output.

3.3 The prior

To do Bayesian inference over the network u_θ we need to define a prior $p(\theta)$ over the weights of the network. In this work we consider a Gaussian prior

$$p(\theta) \sim \mathcal{N}(0, \sigma_*^2 I). \tag{3.7}$$

The basic idea of the method does not depend on this particular choice of prior and it would also be possible to use a different prior over the weights if needed, but the hessian of the different prior might not have such a nice, closed-form solution.

3.4 The likelihood model

So far the information about our PDE problem is still only encoded in the two operators $\mathcal{F}$ and $\mathcal{B}$. To do Bayesian inference we also need to define a data likelihood $p(\mathcal{D} \mid \theta)$.

We first discretize our domain by choosing data points $\{x_i^f\}$ and $\{x_i^b\}$, the two sets of collocation points for the two operators respectively. In our Bayesian setting we

then relax the hard constraints (3.3) at the collocation points and instead model them using zero-mean Gaussian distributions:

$$\begin{aligned} \mathcal{F}[u_\theta](x_i^f) &= z_i^f, \\ \mathcal{B}[u_\theta](x_i^b) &= z_i^b, \end{aligned} \tag{3.8}$$

with

$$\begin{aligned} z_i^f &= f_\theta(x_i^f) + \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(0, \sigma_f^2 I), \\ z_i^b &= b_\theta(x_i^b) + \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(0, \sigma_b^2 I). \end{aligned} \tag{3.9}$$

We can think of the sets

$$\begin{aligned} \mathcal{D}_f &= \{x_i^f, z_i^f\}_{i=1}^{N_f}, \\ \mathcal{D}_b &= \{x_i^b, z_i^b\}_{i=1}^{N_b}, \end{aligned} \tag{3.10}$$

as tuples of (input, output) training data for our neural network. When learning the network we set all

$$z_i^{\{f,b\}} = 0. \tag{3.11}$$

Note that by construction $\mathcal{D}_f$ and $\mathcal{D}_b$ are independent and the single data points are identically and independently distributed. In the following we will denote the full data set as

$$\mathcal{D} = \mathcal{D}_f \cup \mathcal{D}_b. \tag{3.12}$$

For the differential operator f_θ we then get the likelihood:

$$p(z_i^f \mid x_i^f, \theta) = \mathcal{N}(f_\theta(x_i^f), \sigma_f^2 I) \tag{3.13}$$

$$= \frac{1}{\sigma_f \sqrt{2\pi}} \exp\left(-\frac{(f_\theta(x_i^f) - z_i^f)^T I (f_\theta(x_i^f) - z_i^f)}{2\sigma_f^2} \right) \tag{3.14}$$

The full likelihood for the data set $\mathcal{D}_f$ can now be written as

$$p(D_f \mid \theta) = \prod_{i=1}^{N} \frac{1}{\sigma_f \sqrt{2\pi}} \exp\left(\frac{\|(f_\theta(x_i^f) - z_i^f)\|^2}{2\sigma_f^2} \right), \tag{3.15}$$

because the data points are independent and identically distributed. We construct a similar likelihoods for the boundary condition operator b_θ:

$$p(z_i^b \mid x_i^b, \theta) = \mathcal{N}(b(x_i^b), \sigma_b^2 I), \tag{3.16}$$

$$p(D_b \mid \theta) = \prod_{i=1}^{N} \frac{1}{\sigma_b \sqrt{2\pi}} \exp\left(\frac{\|(b_\theta(x_i^b) - z_i^b)\|^2}{2\sigma_b^2} \right) \tag{3.17}$$

Because we assume independent and identically distributed data points we can write down the total likelihood that encompasses both operators in the following way:

$$p(\mathcal{D} \mid \theta) \overset{iid}{=} p(\mathcal{D}_f \mid \theta)\, p(\mathcal{D}_b \mid \theta) \tag{3.18}$$

3.5 Maximum Likelihood Estimation

Our next step is now to maximize this data likelihood $p(\mathcal{D} \mid \theta)$. The maximum likelihood estimate for a parameter θ is defined as

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \ \log p(\mathcal{D} \mid \theta). \tag{3.19}$$

Optimizing this is equivalent to minimizing the negative log-likelihood

$$\mathcal{L}(\theta) = -\log p(\mathcal{D} \mid \theta). \tag{3.20}$$

For the likelihoods defined for our problem setup this results in the following:

$$\begin{aligned}
\underset{\theta}{\operatorname{argmax}} \ p(\mathcal{D} \mid \theta) &= \underset{\theta}{\operatorname{argmin}} \ -\log p(\mathcal{D} \mid \theta) \\
&= \underset{\theta}{\operatorname{argmin}} \ -\left(\log\left[p(\mathcal{D}_f \mid \theta)\right] + \log\left[p(\mathcal{D}_b \mid \theta)\right]\right) \\
&= \underset{\theta}{\operatorname{argmin}} \ -\log\left[\prod_{i=1}^{N} \frac{1}{\sigma_f\sqrt{2\pi}} \exp\left(\frac{\|(f_\theta(x_i^f))\|^2}{2\sigma_f^2}\right)\right] \\
&\quad -\log\left[\prod_{i=1}^{N} \frac{1}{\sigma_b\sqrt{2\pi}} \exp\left(\frac{\|(b_\theta(x_i^b))\|^2}{2\sigma_b^2}\right)\right] \\
&= \underset{\theta}{\operatorname{argmin}} \ \sum_{i=1}^{N_f} \left(f_\theta(x_i^f)\right)^2 + \sum_{i=1}^{N_b} \left(b_\theta(x_i^b)\right)^2 \\
&= \underset{\theta}{\operatorname{argmin}} \ \frac{1}{N_f}\sum_{i=1}^{N_f} \left(f_\theta(x_i^f)\right)^2 + \frac{1}{N_b}\sum_{i=1}^{N_b} \left(b_\theta(x_i^b)\right)^2
\end{aligned}$$

In the case of $\sigma_f = \sigma_b = 1$ this exactly corresponds to the PINN loss formulation in Equation 2.10:

$$MSE := \frac{1}{N_f}\sum_{i=1}^{N_f} \left(f_\theta(x_i^f)\right)^2 + \frac{1}{N_b}\sum_{i=1}^{N_b} \left(b_\theta(x_i^b)\right)^2 \tag{3.21}$$

3.6 MAP Inference

Our overarching goal is computing the full posterior distribution $p(\theta \mid \mathcal{D})$, including the prior $p(\theta)$ this posterior is of the form

$$p(\theta \mid D) = \frac{p(\mathcal{D} \mid \theta)p(\theta)}{\int_\theta p(\mathcal{D} \mid \theta)p(\theta)} \propto p(\mathcal{D} \mid \theta)p(\theta). \tag{3.22}$$

As a first step we can find the most likely parameter under the posterior. This is called the maximum a-posteriori (MAP) estimate of a function and is defined as:

$$\theta_{MAP} = \underset{\theta}{\operatorname{argmax}} \ \log p(\theta \mid \mathcal{D}) = \underset{\theta}{\operatorname{argmax}} \ \log p(\mathcal{D} \mid \theta)p(\theta) \tag{3.23}$$

For our choice of likelihoods and prior the MAP can be computed in the following:

$$\begin{aligned}
\underset{\theta}{\operatorname{argmax}} \ p(\mathcal{D} \mid \theta) &= \underset{\theta}{\operatorname{argmin}} \ -\log[p(\mathcal{D} \mid \theta)p(\theta)] \\
&= \underset{\theta}{\operatorname{argmin}} \ -\log p(\mathcal{D}_f \mid \theta) - \log p(\mathcal{D}_b \mid \theta) - \log p(\theta) \\
&= \underset{\theta}{\operatorname{argmin}} \ \sum_{i=1}^{N_f} \left(f_\theta(x_i^f)\right)^2 + \sum_{i=1}^{N_b} \left(b_\theta(x_i^b)\right)^2 + \frac{1}{2}\sigma_*^{-2}\|\theta\|^2.
\end{aligned}$$

The choice of our Gaussian prior on the parameters θ above corresponds to the weight decay regularizer $r(\theta)$, widely used for neural training:

$$r(\theta) = \frac{1}{2}\sigma_*^{-2}\|\theta\|^2 = -\log p(\theta). \tag{3.24}$$

3.7 Laplace Approximation for PINNs

Section 2.3 introduced the general approach of the Laplace approximation to approximate a target distribution with a Gaussian. In the next step we apply the discussed method, adapted to our problem setup.

We are interested in computing the posterior

$$p(\theta \mid D) = \frac{p(\mathcal{D} \mid \theta)p(\theta)}{\int_\theta p(\mathcal{D} \mid \theta)p(\theta)} := \frac{1}{Z}h(\theta). \tag{3.25}$$

Computing this posterior exactly is often impossible, as solving the integral

$$p(\mathcal{D}) = \int_\theta p(\mathcal{D} \mid \theta)p(\theta) \tag{3.26}$$

is intractable. We now approximate the posterior using a Laplace approximation (as described in Section 2.3) after optimizing and obtaining a MAP estimate θ_{MAP} for $p(\theta \mid \mathcal{D})$. We first use a second order Taylor expansion around the mode of our target distribution θ_{MAP}:

$$\log h(\theta) \approx \log h(\theta_{MAP}) - \frac{1}{2}(\theta - \theta_{MAP})^T \Lambda (\theta - \theta_{MAP}) \tag{3.27}$$

$$= \log \left[p(\mathcal{D}_f \mid \theta_{MAP}) p(\mathcal{D}_b \mid \theta_{MAP}) p(\theta_{MAP})\right] - \frac{1}{2}((\theta - \theta_{MAP})^T \Lambda (\theta - \theta_{MAP})) \tag{3.28}$$

$$= \left[\log p(\mathcal{D}_f \mid \theta_{MAP}) + \log p(\mathcal{D}_b \mid \theta_{MAP}) + \log p(\theta_{MAP})\right] \tag{3.29}$$

$$- \frac{1}{2}((\theta - \theta_{MAP})^T \Lambda (\theta - \theta_{MAP})), \tag{3.30}$$

with the Hessian

$$\Lambda = -\nabla^2 \left[\log p(\mathcal{D}_f \mid \theta_{MAP}) + \log p(\mathcal{D}_b \mid \theta_{MAP}) + \log p(\theta_{MAP})\right] |_{\theta_{MAP}} \tag{3.31}$$

$$= -\nabla^2 \log p(\mathcal{D}_f \mid \theta_{MAP})|_{\theta_{MAP}} - \nabla^2 \log p(\mathcal{D}_b \mid \theta_{MAP})|_{\theta_{MAP}} - \nabla^2 \log p(\theta_{MAP})|_{\theta_{MAP}}, \tag{3.32}$$

and

$$\log h(\theta_{MAP}) = \sum_{i=1}^{N} \log \left(\frac{1}{\sqrt{2\pi\sigma_f^2}} \right) + \left(\frac{f_{\theta_{MAP}}(x_i^f)^T f_{\theta_{MAP}}(x_i^f)}{2\sigma_f^2} \right)$$
$$+ \sum_{i=1}^{N} \log \left(\frac{1}{\sqrt{2\pi\sigma_b^2}} \right) + \left(\frac{b_{\theta_{MAP}}(x_i^b)^T b_{\theta_{MAP}}(x_i^b)}{2\sigma_b^2} \right)$$
$$+ \frac{1}{2}\sigma_*^{-2} \|\theta\|^2.$$

The Hessian for our combined likelihood can be computed as a sum of Hessians of the individual likelihood terms. Applying this approximation to the evidence term Z results in:

$$Z \approx \exp(\log h(\theta_{MAP})) \int \exp\left(-\frac{1}{2}(\theta - \theta_{MAP})^T \Lambda (\theta - \theta_{MAP})\right) d\theta \tag{3.33}$$

$$= h(\theta_{MAP}) \frac{(2\pi)^{\frac{d}{2}}}{(\det \Lambda)^{\frac{1}{2}}}. \tag{3.34}$$

With these two results we can now write down the full approximate posterior:

$$p(\theta \mid \mathcal{D}) = \frac{1}{Z} h(\theta) \approx \frac{(\det \Lambda)^{\frac{1}{2}}}{(2\pi)^{\frac{d}{2}}} \exp\left(-\frac{1}{2}(\theta - \theta_{MAP})^T \Lambda (\theta - \theta_{MAP})\right). \tag{3.35}$$

This is the Gaussian density $q(\theta \mid \mathcal{D}) := \mathcal{N}(\theta; \theta_{MAP}, \Lambda^{-1})$.

We compute the Hessian $-\nabla^2 \left[\log h(\theta_{MAP})\right] |_{\theta_{MAP}}$ using automatic differention. If that is intractable we can use one of the approximations discussed in 2.3.1. The Hessian of the Gaussian prior $p(\theta) = \mathcal{N}(\theta; 0, \sigma_*^2 I)$ has a nice closed form solution: $\nabla^2 p(\theta) = -\sigma_*^2 I$.

3.8 Optimizing the hyperparameters

As we have already shown the proposed method corresponds exactly to the definition of PINNs used in Raissi et al. [2019] for the case of $\sigma_f^2 = \sigma_b^2 = 1$. Instead of just setting the variances to a fixed value a-priori, we need to select different hyperparameters that improve the performance of our method and make the uncertainty estimates more accurate. We have already obtained a θ_{MAP} estimate by training a neural network, and have already calculated a first approximation $q(\theta \mid \mathcal{D})$, with the corresponding Hessian Λ^{-1}. Now want to optimize the prior hyperparameters, i.e. the prior variance σ_*^2, and the variance used in the two likelihood functions for the discretized differential operators σ_f^2 and σ_b^2. Note that since we treat the MAP optimization of θ and the Laplace approximation as two separate steps we can choose any prior variance for the Laplace approximation, independent of the weight decay parameter employed in neural network training. We already introduced the Laplace-approximated (log-)marginal likelihood in Equation 2.29. For our problem setup we get the following log-marginal likelihood:

$$\log p(\mathcal{D} \mid \mathcal{M}) \approx \log q(\mathcal{D} \mid \mathcal{M}) := \log p(\mathcal{D} \mid \mathcal{M}) - \frac{1}{2} \log |\frac{1}{2\pi} \mathbf{H}_{\theta_{MAP}}| \tag{3.36}$$

$$= \log h(\theta_{MAP}) - \frac{1}{2}(\theta - \theta_{MAP})^T \Lambda (\theta - \theta_{MAP}) - \frac{1}{2} \log |\frac{1}{2\pi} \mathbf{H}_{\theta_{MAP}}|, \tag{3.37}$$

with

$$\begin{aligned}
\log h(\theta_{MAP}) &= \log\left[p(\mathcal{D}_f \mid \theta_{MAP})p(\mathcal{D}_b \mid \theta_{MAP})p(\theta_{MAP})\right] \\
&= \sum_{i=1}^{N_f} \log\left(\frac{1}{\sqrt{2\pi\sigma_f^2}}\right) + \left(-\frac{f_{\theta_{MAP}}(x_i^f)^T f_{\theta_{MAP}}(x_i^f)}{2\sigma_f^2}\right) \\
&\quad + \sum_{i=1}^{N_b} \log\left(\frac{1}{\sqrt{2\pi\sigma_b^2}}\right) + \left(-\frac{b_{\theta_{MAP}}(x_i^b)^T b_{\theta_{MAP}}(x_i^b)}{2\sigma_b^2}\right) + \frac{1}{2}\sigma_*^{-2}\|\theta\|^2 \\
&= N_f(-\log\left(\sqrt{2\pi\sigma_f^2}\right)) - \frac{1}{2\sigma_f^2}MSE_f + N_b(-\log\left(\sqrt{2\pi\sigma_b^2}\right)) \\
&\quad - \frac{1}{2\sigma_b^2}MSE_b + \frac{1}{2}\sigma_*^{-2}\|\theta\|^2. \qquad (3.38)
\end{aligned}$$

This is our training objective for the hyperparameter optimization for the three parameters σ_*^2, σ_f^2 and σ_b^2. In our offline, post-hoc setting we can now iteratively optimize those hyperparameters by minimizing this negative log-marginal distribution until convergence.

Notably, in our setup optimizing the hyperparameters σ_f and σ_b is comparable to choosing weights w_f, w_b for a weighted MSE loss function:

$$MSE = \frac{w_f}{N_f}\sum_{i=1}^{N_f}|f(t_f^i, x_f^i)|^2 + \frac{w_b}{N_b}\sum_{i=1}^{N_b}|b(t_b^i, x_b^i)|^2. \qquad (3.39)$$

Multiple papers [Wang et al., 2021, 2022] proposed adaptive weighting techniques to mitigate different problems in PINN training. We can think of the hyperparameter optimization by maximizing the marginal likelihood as another approach to choose the weights.

3.9 Summary of the full Method

The full method now entails first training a neural network using a gradient based optimizer (e.g. the widely used optimizer Adam [Kingma and Ba, 2014]). Next we use a Laplace approximation to get our approximate posterior $q(\theta \mid \mathcal{D})$. We can then tune our models hyperparameters σ_*^2, σ_f^2 and σ_b^2 by minimizing the negative Laplace-approximated marginal log-likelihood until convergence. Lastly we linearize the neural network to compute the posterior over the network outputs $p(u(x) \mid x, \mathcal{D})$. Simplified pseudocode for the full method is described in Algorithm 1.

Algorithm 1 Bayesian Physics-informed Neural Networks

Require:
Neural network u_θ; number of epochs T_0 for MAP calculation

1: Initialize θ_0
2: Generate training set $\mathcal{D}$ by choosing collocation points for $\mathcal{F}[u], \mathcal{B}[u]$;
3: Train the neural network $u_\theta(x)$ for T_0 iterations, by optimizing $\mathcal{L}(\mathcal{D}; \theta)$
4: Laplace Approximation: $q(\theta \mid \mathcal{D}) \approx \mathcal{N}(\theta_t, (\nabla^2 \mathcal{L}(\mathcal{D}; \theta)|_{\theta_t})^{-1})$
5: Optimize the log-marginal likelihood $\log p(\mathcal{D} \mid \sigma_*^2, \sigma_f^2, \sigma_b^2)$ until convergence
6: Approximate posterior: $p(u(x) \mid x, \mathcal{D}) \approx \mathcal{N}(u(x) \mid u_{\theta_{MAP}}(x), J(x)^T \Lambda^{-1} J(x))$
7: **return** θ_{MAP}; $q(u(x) \mid x, \mathcal{D})$ ▷ Return the MAP estimate and the posterior

In this algorithm the loss function used training the neural network is

$$\mathcal{L}(\mathcal{D}; \theta) = \sum_{i=1}^{N_f} \left(f_\theta(x_i^f) \right)^2 + \sum_{i=1}^{N_b} \left(b_\theta(x_i^b) \right)^2 + \frac{1}{2} \sigma_*^{-2} \|\theta\|^2. \tag{3.40}$$

And the loss function for optimizing the hyperparameters is the Laplace approximated log-marginal likelihood

$$\log p(\mathcal{D} \mid \sigma_*^2, \sigma_f^2, \sigma_b^2) = \log h(\theta_{MAP}) - \frac{1}{2}(\theta - \theta_{MAP})^T \Lambda (\theta - \theta_{MAP}) - \frac{1}{2} \log |\frac{1}{2\pi} \mathbf{H}_{\theta_{MAP}}|, \tag{3.41}$$

with $\log h(\theta)$ as in Equation 3.38.

Chapter 4

Experimental Results

In this section we present the results and interpretation of different experiments. We evaluate the method we developed in Chapter 3 on the Burgers' equation, a non-linear hyperbolic partial differential equation. We analyse the quality of the PINN solution for the PDE, as well as the ability of the posterior variance to capture the numerical error and compare those results for different densities of collocation grid points. Next we analyze the ability of the posterior to capture sources of uncertainty in the PINN solution, like missing data in large regions of the domain or temporal extrapolation. Our aim is to show that our method is able to accurately solve a PDE problem and return reasonable uncertainty estimates over the outputs.

In Section 4.1 we first introduce the evaluation metrics used to asses the performance of the solution and the calibration of the returned uncertainties. Section 4.2 defines the PDE problem we analyzed in our experiments. Next, Section 4.3 gives some background on the technical details of our implementation of the method. In Section 4.4 we analyse the performance of our method in solving the PDE problem. Section 4.5 then presents the uncertainties returned by our method and examines some properties of them. Next, in Section 4.6 we showcase some experiments where we apply our method in a setting where large regions of the domain are not covered with collocation points. We do not expect our method to be able to accurately solve the PDE problem in this setting, but we want to verify that our posterior captures the uncertainty. Finally, Section 4.7 summarizes our results.

4.1 Evaluation metrics

4.1.1 Performance metrics

To evaluate the accuracy of the PINN solution, we will compare the approximate PINN solution $u_\theta(x)$ returned by our method with a reference solution on a dense grid of evaluation points $\{x_i\}_{i=1}^N$. The evaluation grid we use has 25600 data points. In scientific machine learning many different metrics can be employed [Takamoto et al., 2022]. We use the root-mean-square-error (RMSE) as a measure to capture the global performance of the accuracy of our method. For the Burgers' equation a large part of the domain is fairly easy to solve, but the shock at $x \approx 0$ is very difficult to model, and must be solved very accurately, as small deviations can lead to large errors.

$$RMSE := \sqrt{\frac{1}{N}\sum_{i=1}^{N}(u_\theta(x_i) - u_i^*)^2} \tag{4.1}$$

4.1.2 Calibration

Having obtained a posterior $p(u(x) \mid x, \mathcal{D})$ we are now interested in quantifying the accuracy of our posterior, as well as comparing the predicted error estimates to the actual numerical error of the PINN solution.

To asses how well our predictive posterior captures the numerical error of our PINN solution we need to analyze the uncertainties returned by the posterior. Calibration measures how well the predicted distribution captures the true underlying data distribution.

In our model the true values of the solution of the PDE problem u_i^* are Gaussian distributed around the predicted solution of our model $u_\theta(x_i)$:

$$u_i^* \sim \mathcal{N}(u_\theta(x_i), \sigma_i^2), \tag{4.2}$$

where σ_i^2 is the variance at the data point x_i given by the approximate posterior:

$$p(u(x_i) \mid \mathcal{D}) = \mathcal{N}(u_\theta(x_i); J(x_i)^T \Lambda^{-1} J(x_i)) = \mathcal{N}(u_\theta(x_i); \sigma_i^2 I). \tag{4.3}$$

With this, the deviation of the solution is a zero-mean Gaussian:

$$(u_i^* - u_\theta(x_i)) \sim \mathcal{N}(0, \sigma_i^2). \tag{4.4}$$

Next we normalizing with the standard deviation to get the standard normal distribution:

$$\frac{(u_i^* - u_\theta(x_i))}{\sigma_i} \sim \mathcal{N}(0, 1). \tag{4.5}$$

Then by definition the square of this standard normal random variable is distributed chi-squared with one degree of freedom:

$$Q := \frac{(u_\theta(x_i) - u_i^*)^2}{\sigma_i^2} \sim \chi_1^2. \tag{4.6}$$

This is also the squared error of our solution, normalized with the variance of the posterior.

We will use this quantity to analyze how well the uncertainty of our posterior is calibrated. If our posterior is perfectly calibrated, i.e. our model captures the data uncertainty well, the quantity Q should be exactly at the mean of the χ^2 distribution, which corresponds to the degrees of freedom, in our case $\mathbb{E}(\chi_1^2) = 1$. However, we can assume that our predicted solution has some numerical error, so this will almost never be the case. If $Q << 1$, we say that our posterior is underconfident in this region, since the squared error of the PINN solution is much smaller than the estimated error of the posterior. On the other hand, if $Q >> 1$, we say that our posterior is overconfident in this region because the squared error of the PINN solution is much larger than the estimated error. For values of $Q \approx 1$ we assume that our posterior is reasonably well calibrated. As an indication of how close to the mean we are, we can use the confidence intervals of the chi-squared distribution.

We can compute confidence intervals for different confidence levels and then check whether the quantity Q lies within them for the points in our evaluation grid. At points where Q is less than the left endpoint of the interval we call our posterior underconfident, and at points where Q is greater than the right endpoint, we call the posterior overconfident.

4.2 Problem description: Burgers' equation

To illustrate the problem statement and to introduce one of the problems considered in the experiments, we introduce the Burgers' equation.

This is a nonlinear convection-diffusion equation . The Burgers' equation is a simplification of the incompressible Navier-Stokes equation that omits the pressure term and inherits many properties of the Navier-Stokes equation [Frisch and Bec, 2001]. When the parameter ν is very small this equation becomes numerically challenging to solve because a small ν can lead to shocks. Because of its challenging nature, its similarities to the incompressible Navier-Stokes equations and possible analytical solution it is still a widely used equation for testing and benchmarking numerical algorithms.

In its most general form the Burgers' equation in one spatial dimension is defined as follows:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}. \tag{4.7}$$

Here x denotes the space and t the time dimension. The solution function $u(x,t)$ denotes the speed of the fluid along the space dimensions at each time point. The parameter ν denotes the viscosity of the fluid.

In the following we will consider the following PDE problem, i.e. the Burgers' equation with $\nu = \frac{0.01}{\pi}$, $u(0,x) = sin(\pi x)$ as the initial state at $t = 0$ and with Dirichlet boundary conditions:

$$\begin{aligned} &u_t + uu_x - (0.01/\pi)u_{xx} = 0,\ x \in [-1,1],\ t \in [0,1], \\ &u(0,x) = -sin(\pi x), \\ &u(t,-1) = u(t,1) = 0. \end{aligned}$$

In figure 4.1 we show a reference solution to the Burgers' equation with the above parameterization.

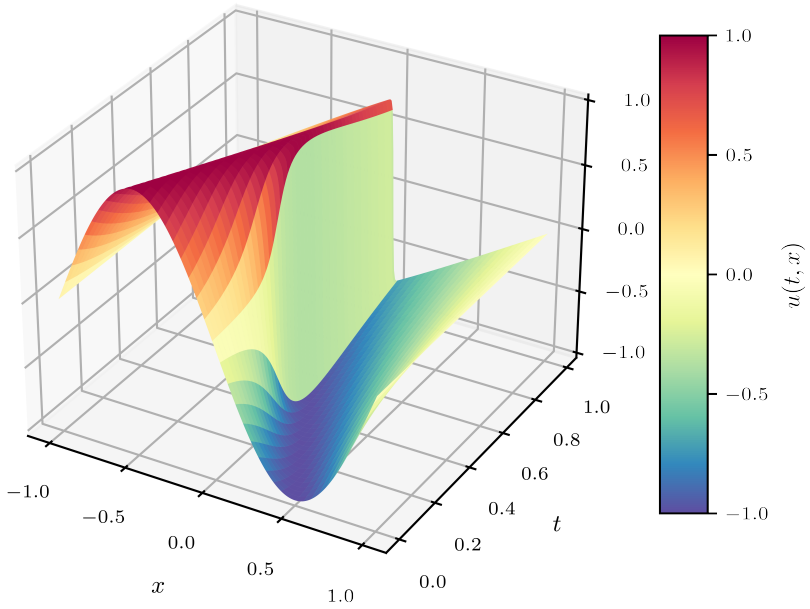


Figure 4.1: *Burgers' Equation.* Each point (t,x) on this grid represents the reference solution u(t,x).

4.3 Implementation details

We implemented [1] the method discussed in Chapter 3 in Python using PyTorch [Paszke et al., 2019] for our neural network and as a auto-differentiation framework. All experiments are reproducible with the provided code. A very simple neural network architecture is sufficient for this setup. The model we used has three hidden layers with 16 nodes each. We used hyperbolic tangent activation functions, in our tests other activation functions performed much worse. The training scheme was optimized for the case of $N = 2^{15}$ collocation points. The model was trained for 50000 iterations using the Adam optimizer [Kingma and Ba, 2014] with a fixed learning rate $\alpha = 10^{-3}$ and a weight decay parameter of $r = 10^{-2}$. We computed the Hessians for our likelihood terms using ASDL [Osawa, 2020], for our simple experimental setup we did not need to use any of the approximations discussed in Section 2.3.1. However, our implementation also features the Fisher information matrix as an approximation to the Hessian. We had to use a pseudo-inverse to invert the Hessian in the Laplace approximation because the sum of the Hessians of the likelihoods was ill-conditioned. We optimized the hyperparameters of our likelihoods and priors by optimizing the marginal likelihood until convergence, again using Adam.

4.4 Performance of PINN solution

In this section, we evaluate the performance of our PINN approach for solving PDE problems. We compare the accuracy of the solution of our PINN method and a reference solution to the Burgers' equation, which we introduced in 4.2. For each experiment, we set up an equidistant grid over the entire domain. We use different numbers of grid points to test the performance of the method for different discretizations. In both traditional numerical solvers and general neural network training we would expect our performance to improve with more grid points. We run our method and train a neural network with the setup described in Section 4.3 for each grid.

Figure 4.2 shows the RMSE of the PINN solution compared to the reference solution for the different grids. As expected, the error of the solution decreases with a denser grid of collocation points. As can be seen, the PINN solution for $N_f = 2^{13}$ has the lowest RMSE for this choice of grid and training setup. The increased error for more grid points beyond this point can be attributed to insufficient training, as the training setup was optimized for $N_f = 2^{13}$ collocation points. The quality and convergence of the solution also depends on the random initialization of the

[1] https://github.com/aken0/laplacePINNs

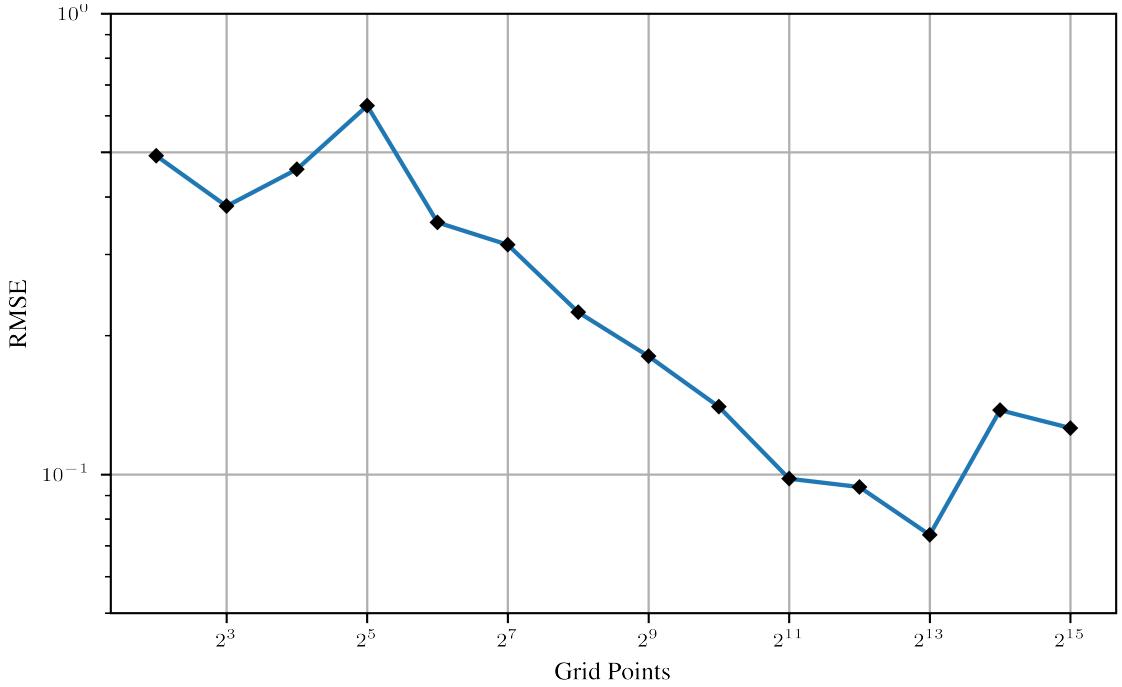


Figure 4.2: *RMSE for different equidistant grids.*

network weights. While the absolute value of the error does not seem to decrease much (only by about a factor of 6), the quality of the solutions improves a lot. This can be attributed to the properties of the Burgers' equation, as propagating the initial condition along the whole time dimension already achieves a reasonable RMSE.

To gain an overview over the quality and differences of the solutions we take a look at the solution functions returned by our method. In Figure 4.3 we show the PINN solution and underlying grid points for four selected grids, with $2^2, 2^4, 2^6$ and 2^{10} points respectively. The solution for $N_f = 2^4$ is already quite good, it is able to capture the general dynamics of the system. In Figure 4.4 we also show the squared error of the PINN solution and our reference solution for the same grids. Note that the scale of colorbars is different for the different solutions. For all employed grids the solution struggles to accurately model the shock at $x \approx 0$, even the best solutions had some error there. But the solutions with more grid points achieve a better approximation to the reference solution. In our tests, we also observed that a more powerful neural network architecture with a correspondingly modified training setup showed even better performance.

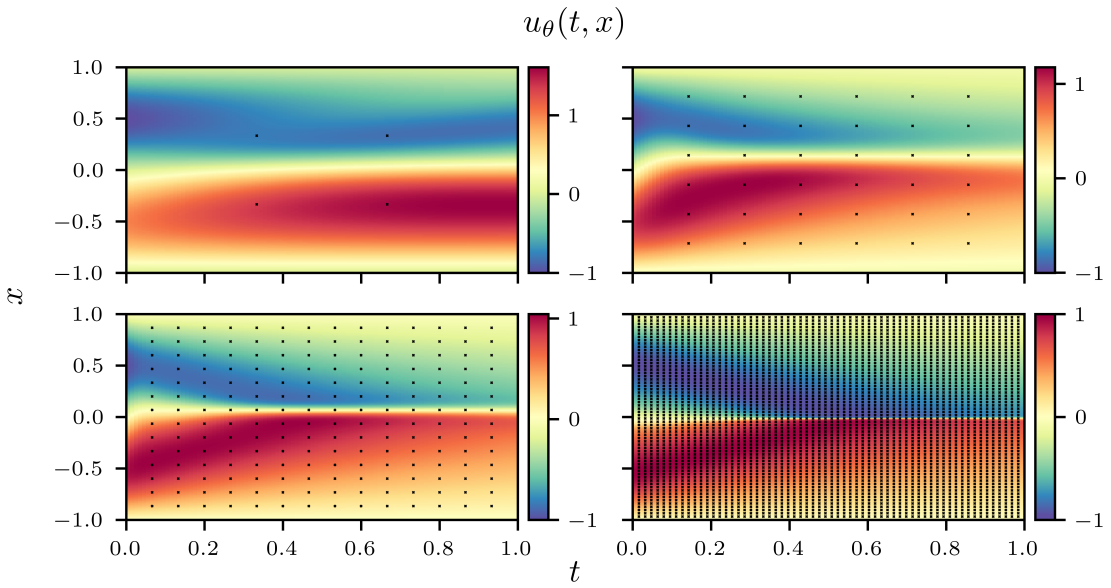


Figure 4.3: *Solutions for equidistant grid.* The grids have $2^2, 2^4, 2^6$ and 2^{10} points respectively. The solution on the top-left fails to capture the PDE dynamics. The solution on the bottom right is a good approximation to the reference solution.

4.5 Calibration of uncertainty

In this section we evaluate the quality and calibration of our predicted uncertainty estimate using the metric introduced in 4.1.

Figure 4.5 displays the squared error of the PINN solution and the estimated posterior variance for the same grid. The distribution or shape of the error is reflected in the posterior, i.e. the variance is large in regions of the domain where the error is also large. At the same time the variance is low in areas where the numerical error is also low. Our posterior captures the large error at the shock for values of $x \approx 0$. But the scale of posterior variance is different from the error. This scale mostly affects the values at $x \approx 0$.

4.5.1 Confidence intervals

In Section 4.1.2 we introduced the chi-squared distributed quantity Q. To measure how well the posterior variance captures the numerical error we now analyse how Q relates to the chi-square confidence intervals for each point in our evaluation grid. We calculate the quantity Q_i for each point x_i in our evaluation grid, using the PINN solution $u_\theta(x_i)$ at that point and the corresponding reference solution u_i^*. We then determine for each point x_i whether Q_i lies below, above or within

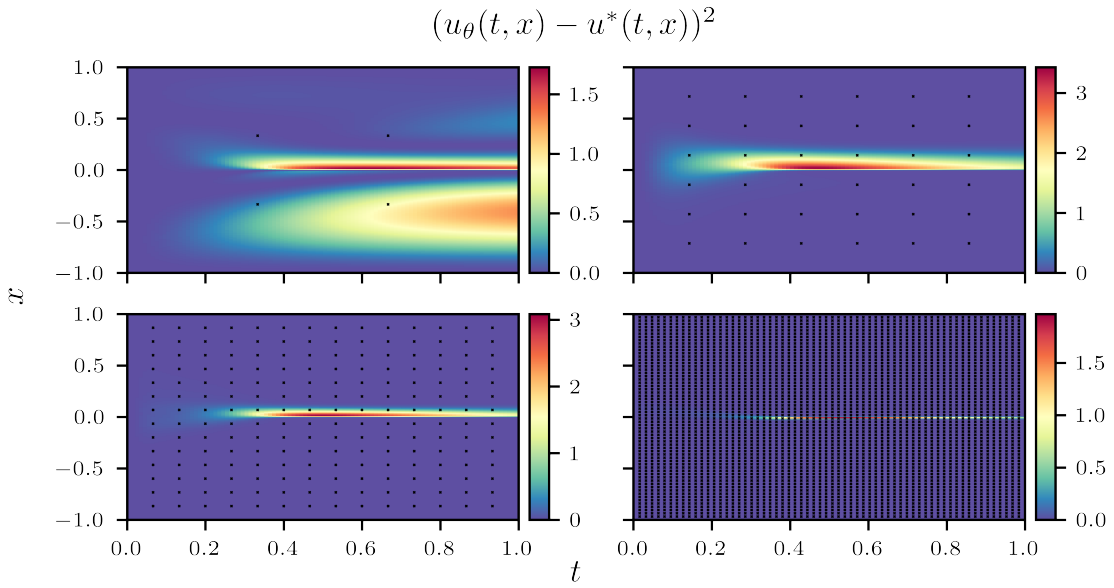


Figure 4.4: *Squared Errors for equidistant grid.* The grids have $2^2, 2^4, 2^6$ and 2^{10} points.

the confidence interval. Points at which Q_i is greater than the right endpoint of the interval are shown in red. In those areas the posterior is overconfident. If Q_i is smaller than the left endpoint of the interval, we show the point in blue, here the posterior is underconfident. We first inspect the 95% confidence interval. For the chi-square distribution with one degree of freedom it is

$$CI_{95\%} = [9.82 \times 10^{-4}, 5.024]. \tag{4.8}$$

In Figure 4.6 we show the quantity Q_i for each point of the evaluation grid. We use the same four training grids used in Section 4.4. For $N_f = 2^2$ the posterior is overconfident almost everywhere. While the PINN solution for $N_f = 2^4$ was already quite good, the posterior at the same grid is still overconfident in a large area of the domain. For $N_f = 2^{10}$ the posterior is overconfident mainly along the shock of the Burgers' equation. It is also underconfident in areas close to the boundary. Next we can also calculate the 99% confidence interval for the same grids. The interval is given as

$$CI_{99\%} = [3.93 \times 10^{-5}, 7.879]. \tag{4.9}$$

Compared to the 95% interval less regions of the posterior are outside the interval, for all grids. With $N_f \geq 2^9$ grid points our posterior looks well calibrated. While the experiments with less grid points were already able to approximate the

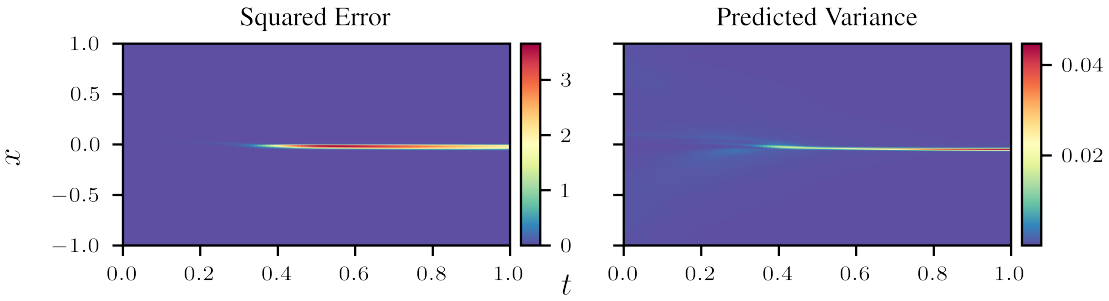


Figure 4.5: *Squared Error and posterior variance for selected grid.* Left: The squared error of the PINN solution. Right: The posterior variance returned by our method for the same grid. The distribution of the error is similar, but the scale is different.

underlying solution function to a good degree, adding more collocation points further improved the calibration of the posterior. This behaviour is inversed at some point, as further collocation points caused the posterior to be underconfident in larger regions of the domain. To further quantify how well calibrated our method is can use a different type of calibration plot.

4.5.2 Calibration plots

If we take a look at the confidence interval plots in the last section we notice that our predictions are both overconfident and underconfident in different regions of the same posterior. This is fine because even if our model would perfectly coincide with the χ^2 distribution, some points might lie outside the confidence interval. For example, if our posterior is well calibrated, at a 95% confidence level roughly 95% of the points in the posterior grid should lie within the confidence interval. We now calculate the area of the region of the predicted posterior uncertainty that lies outside the confidence interval to see how much we deviate from this. We calculate this area for multiple confidence levels, i.e. for all percentiles.

We can contrast the expected area outside any confidence region and the actually observed area outside this confidence region. The closer those two quantities are, the better the calibration of our posterior. In Figure 4.8 on the left we first plot the expected confidence level compared to the actually observed confidence level (represented by the blue line). Ideally these values coincide, this is represented by the black line. On the right in Figure 4.8 we represent the difference of the observed and expected confidence level as a heatmap. This makes it easier to compare the calibration plots for different experiment setups. Instead we could also use the area under the curve and compress this heatmap into a single number, but that way some information is lost. In Figure 4.9 we construct those calibra-

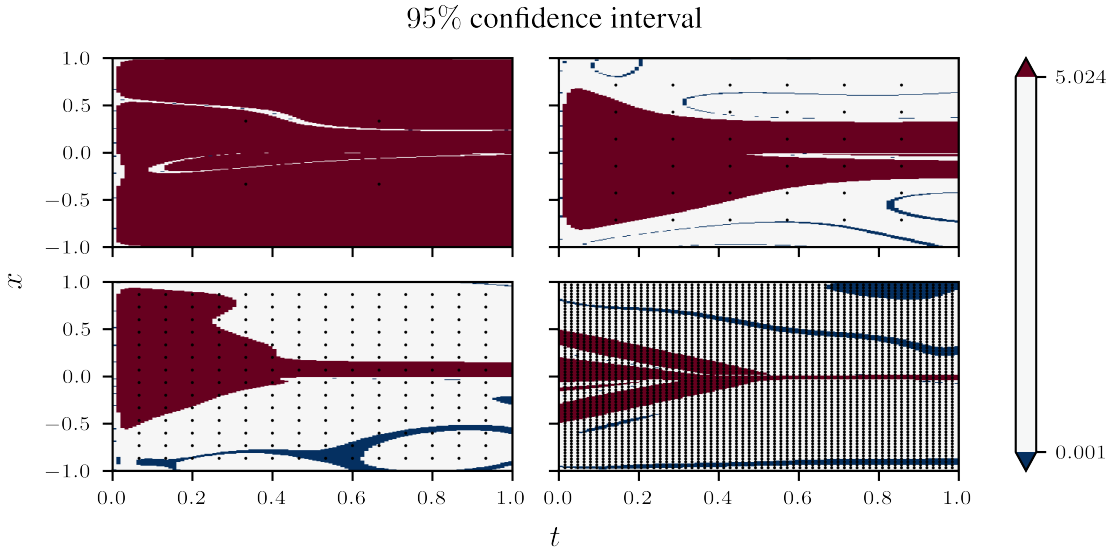


Figure 4.6: 95% *confidence interval for equidistant grids.* The grids have $2^2, 2^4, 2^6$ and 2^{10} points. At points in red the posterior is overconfident, at points in blue underconfident.

tion heatmaps for the 14 equidistant grids used earlier. For the grids with up to $N_f \leq 2^5$ collocation points the calibration is very bad, as was already noted in the section above. The posterior has the best calibration for $N_f = 2^7$ and $N_f = 2^{14}$ collocation points.

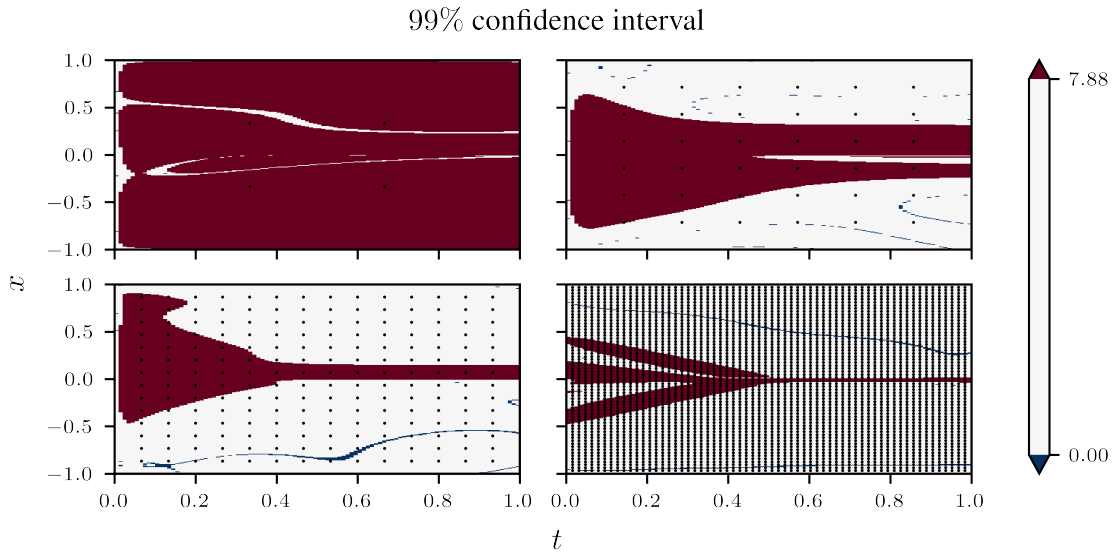


Figure 4.7: 99% *confidence interval for equidistant grids.* The grids have $2^2, 2^4, 2^6$ and 2^{10} points. At points in red the posterior is overconfident, at points in blue underconfident.

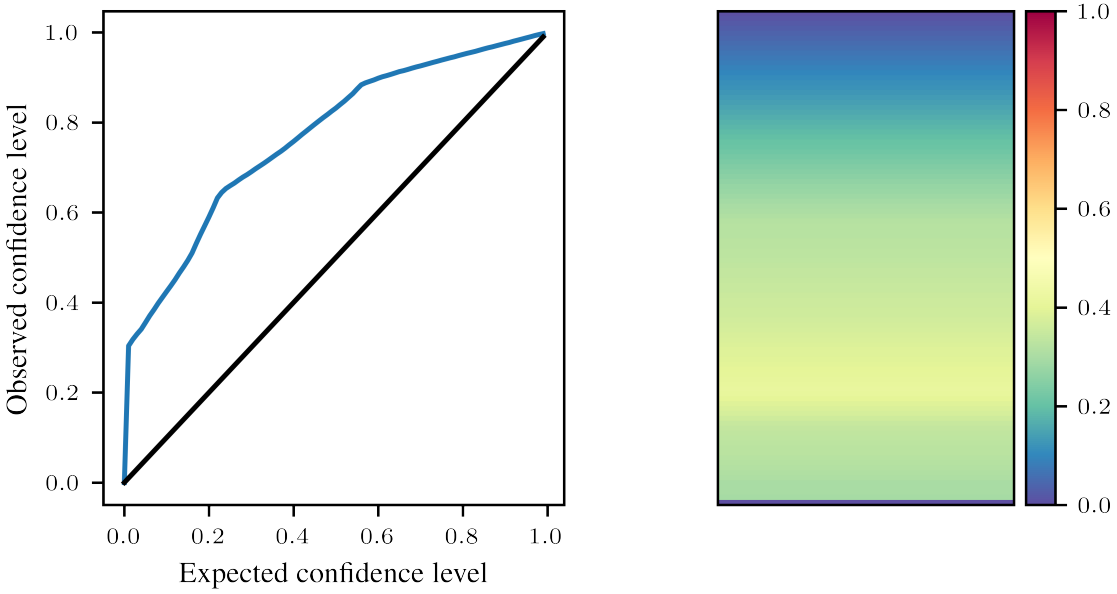


Figure 4.8: *Calibration Plot.* Observed area outside the confidence interval compared to the expected area for all percentile confidence levels (equidistant grid with 2^{14} collocation points).

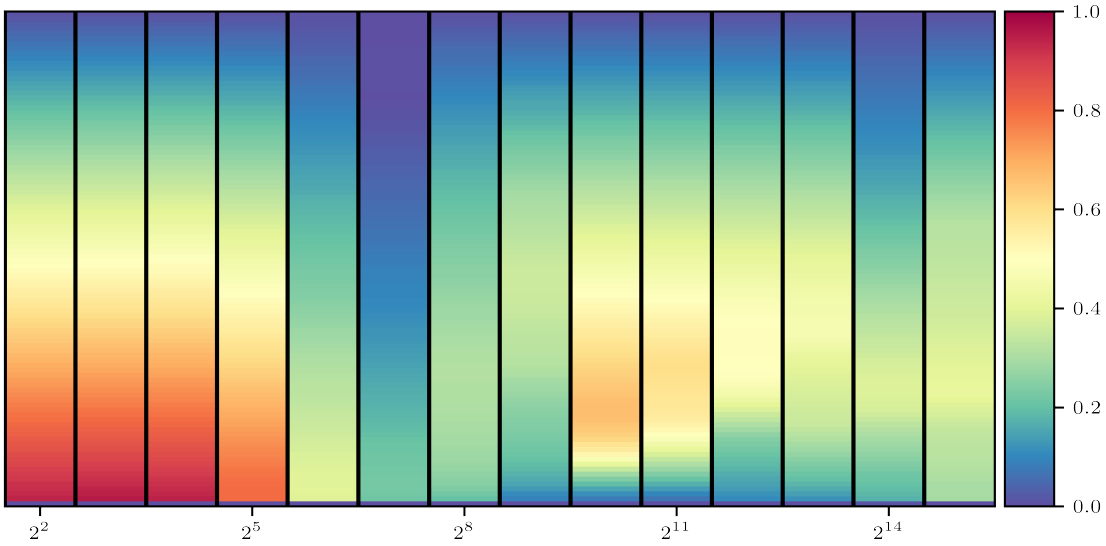


Figure 4.9: *Calibration heatmap for different grids.* For the 14 equidistant grids we showcase the calibration heatmap introduced above.

4.6 Missing collocation data

If we enforce the dynamics of the system given by our PDE problem only on a partial area of the domain we cannot expect good results. But if our posterior is well calibrated we would expect it to reflect the uncertainty that arises from missing data in large areas of the problem domain. To test this behaviour we can restrict the area of our equidistant discrete grid of collocation points in both dimensions of the Burgers' equation we defined.

4.6.1 Missing collocation points in space

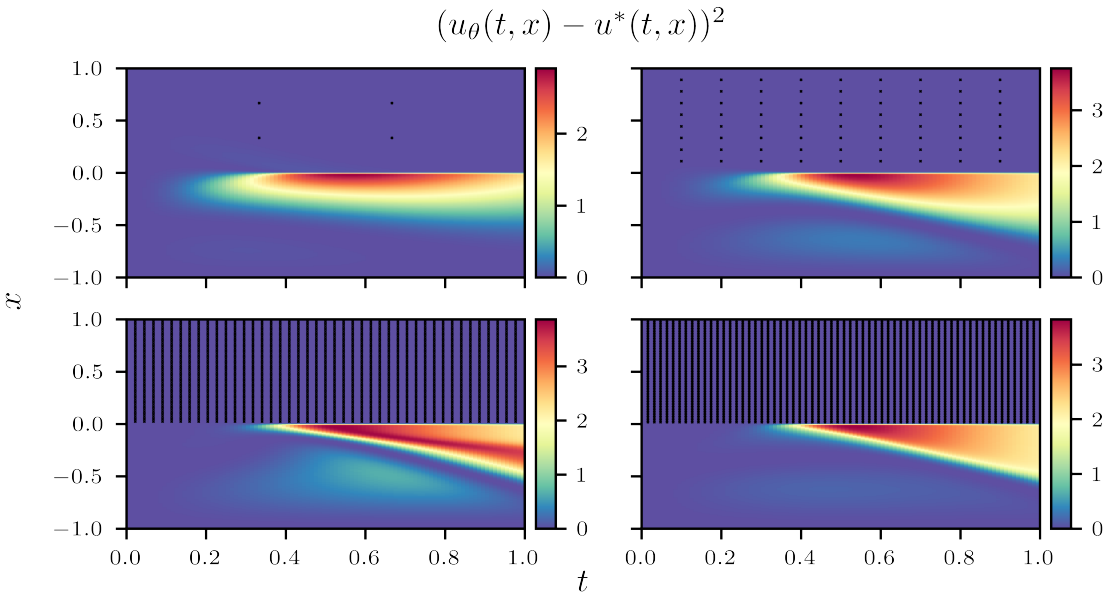


Figure 4.10: *Squared Error for partial spatial grid.* The grids have $2^2, 2^5, 2^9$ and 2^{10} points.

We first examine the setting where there is missing data in the spatial dimension. For this we restrict the collocation grid used for learning the network to values $x > 0$. We expect the model to learn the solution accurately in the area with grid points, and expect the model to perform much worse in areas without training data. Note that we still use the boundary data for $x = -1$ for this experiment. In Figure 4.10 we show four grids with missing collocation points in the spatial dimension and the squared error of the PINNs trained on those grids. In parts of the domain with grid points the PINN is able to solve the PDE problem. But in areas with missing points the error is large, even if we employ many grid

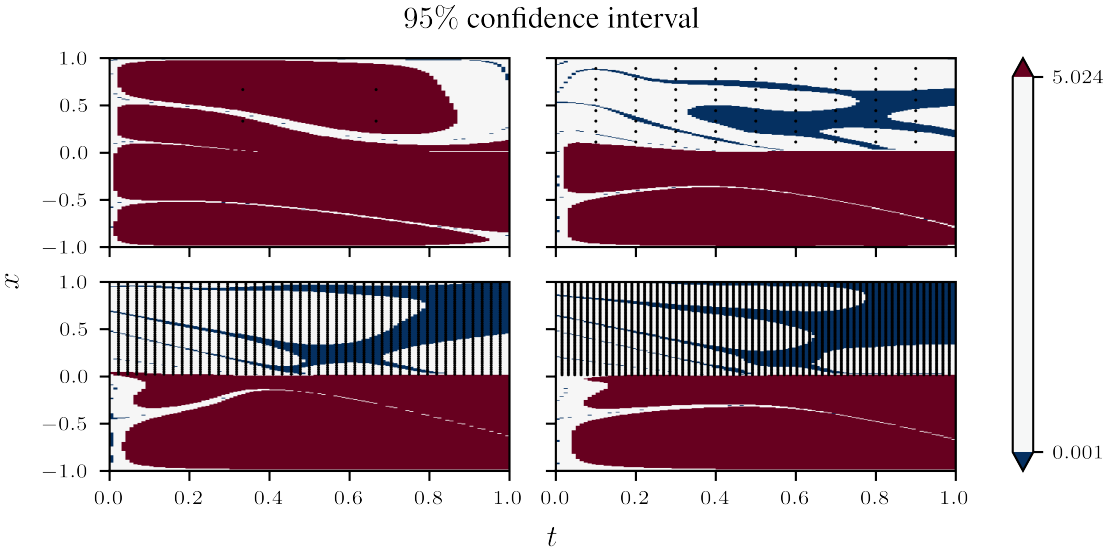


Figure 4.11: 95% *confidence interval for partial spatial grid.* The grids have $2^2, 2^5, 2^9$ and 2^{10} points. At points in red the posterior is overconfident, at points in blue underconfident. For $N_f \geq 2^5$ we see clear structure in the plots, we are underconfidents in areas with data, and overconfident in the others.

points. Figure 4.11 shows the Q_i for the same grids and an underlying 95% confidence interval. Again, areas where Q_i is smaller than the left interval endpoint are colored blue, areas where it is bigger than the right endpoint are red. We can see that for $N_f = 2^2$ grid most of the posterior is overconfident. But already in the case of $N_f = 2^5$ collocation points, we see some clear structure in the plot: In the region of the domain where we had training data the posterior is underconfident, while the region without data is almost entirely overconfident. Our method is able to return some sensible uncertainty estimates. [text]

4.6.2 Extrapolation in Time

Next we also examine the setting where we limit the collocation data in the time dimension, $t < 0.5$. We still use the boundary data for $t > 0.5$ for this experiment. This is significantly harder to solve for PDE problems in general, and the Burgers' equation in particular. The most difficult part to solve in the Burgers' equation is the shock close to $x = 0$ for $t \to 1$. Without data our method need to extrapolate this region, which is not possible. Without collocation points in that area we cannot expect our method to solve the problem accurately, but we still want to verify whether the posterior reflects this uncertainty.

In Figure 4.12 we show four grids with missing collocation points in the time

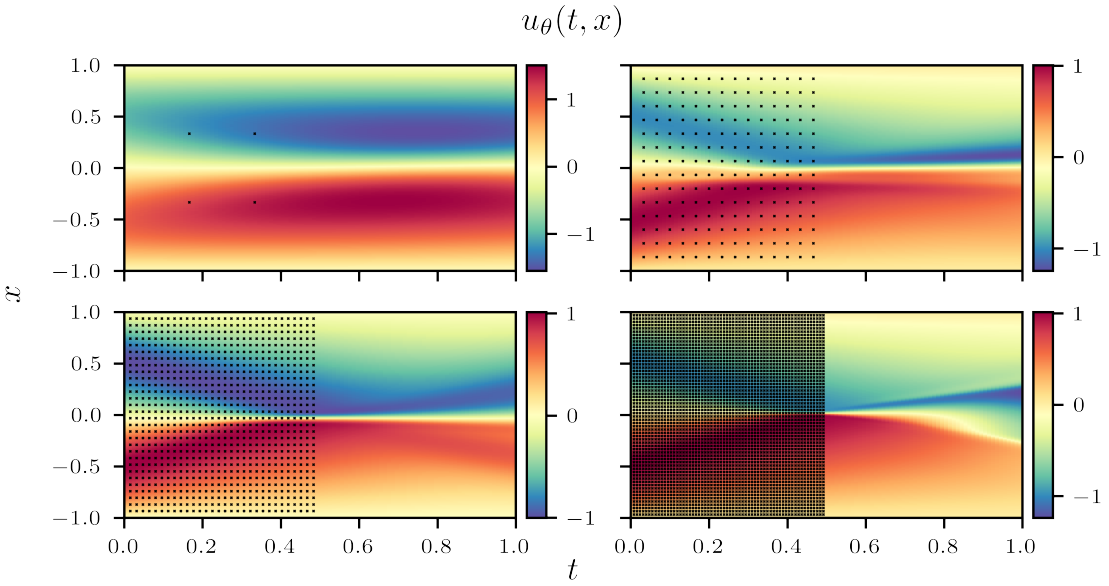


Figure 4.12: *Solution for partial time grid.* The grids have $2^2, 2^6, 2^8$ and 2^{10} points.

dimension and the solution returned by our PINN. In parts of the domain with grid points the PINN is able to solve the PDE problem to a similar degree as in the full grid setting in Section 4.4. This is also reflected in the squared error for the same grids in Figure 4.13. But in the region of the domain without collocation points our method fails to extrapolate and the method is unable to model the PDE dynamics.

Figure 4.14 shows the Q_i for the same grids and an underlying 95% confidence interval. We can see that for $N_f = 2^2$ grid most of the posterior is overconfident. Opposed to the experiments with missing collocation data in the spatial dimension our posterior does not show such clear structure. For the grids with $N_f > 2^5$ collocation points more regions of the posterior are outside the confidence interval, but those regions do not correspond to the ones where data is missing.

From these two experiments we can conclude that our method is able to return sensible uncertainty estimates in some cases, but it failed in the case where the method has to extrapolate in time. But this is usually not relevant for time dependant PDE problems.

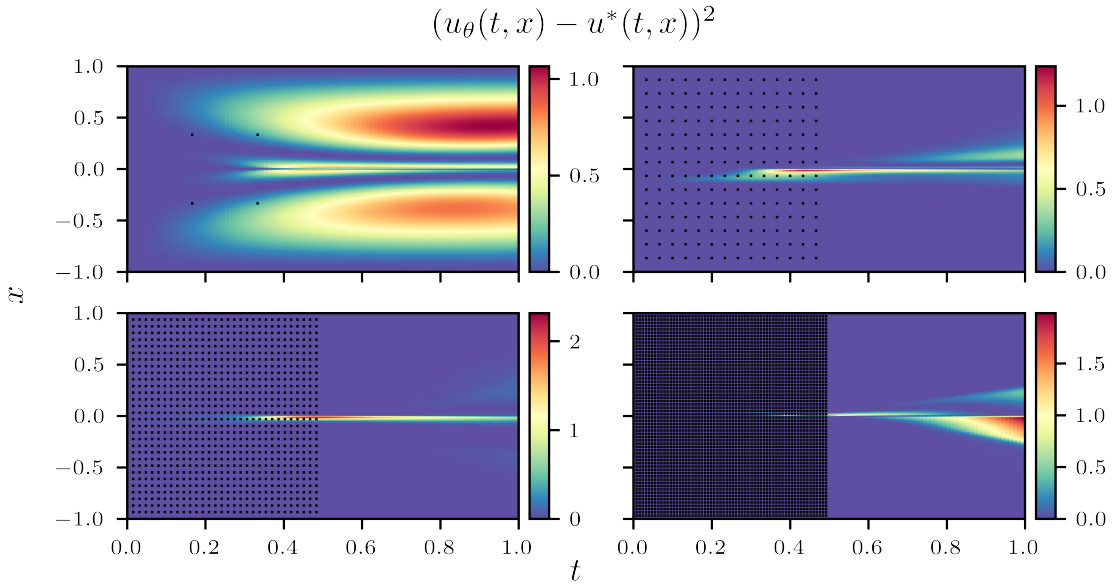


Figure 4.13: *Squared Error for partial time grid.* The grids have $2^2, 2^6, 2^8$ and 2^{10} points.

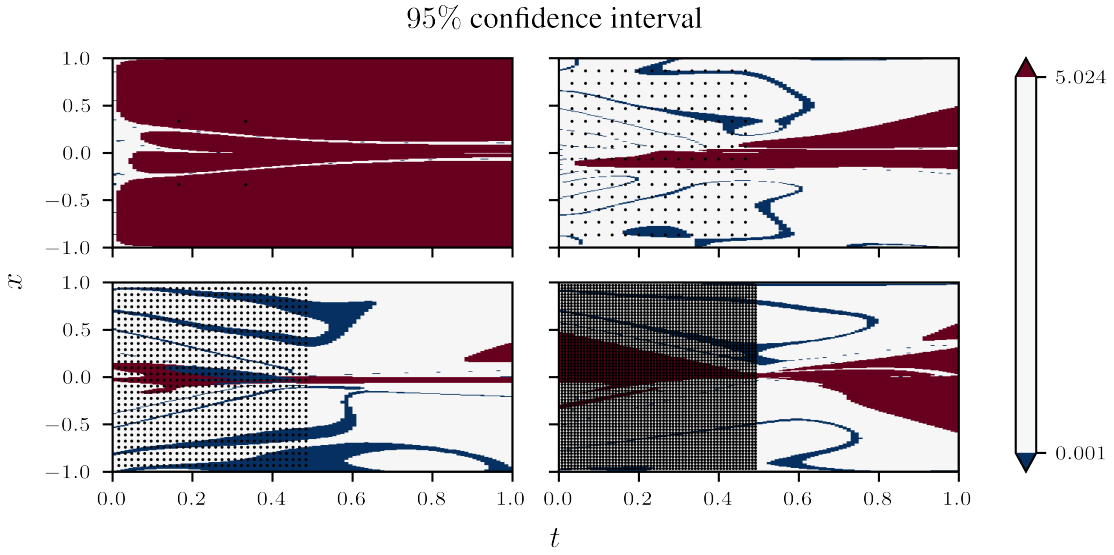


Figure 4.14: 95% *confidence interval for partial time grid.* The grids have $2^2, 2^6, 2^8$ and 2^{10} points. At points in red the posterior is overconfident, at points in blue underconfident. Opposed to the case with missing spatial data, no clear structure is visible.

4.7 Summary

We showed that our method is able to solve the Burgers' equation. By construction it is able to interpolate the solution function trained on a set of colloation points. The PINN learning part of the method can be further refined by using more refined neural network architectures or other enhancements for neural network training. The general method does not interfere with the neural network training and is still able to return uncertainty estimates for more complicated architectures.

We also analyzed the uncertainty estimates of our posterior. In our tests the method is able to provide reasonable uncertainty estimates if a sufficient number of training points are used. The method is also able to correctly model uncertainties resulting from missing data in large regions of the domain. Thus, the entire method is capable of providing both an accurate approximation and reasonable uncertainty estimates over the model outputs.

The reported experiments were all based around equidistant grids. But this choice of grid may not be optimal , in some of our experiments modified grids performed much better, e.g. grids with more points close to the shock of the Burgers' equation. But in general, it can be difficult to choose a good grid for a PDE problem, especially when the underlying solution function is unknown.

Chapter 5

Discussion and Conclusion

5.1 Discussion

Physics-informed neural networks (PINN) have shown great success in multiple applications [Karniadakis et al., 2021] and have become a well established part scientific machine learning. An ongoing area of research is quantifying the different sources of uncertainty that arise in different scientific machine learning methods. In this work, we investigated whether we can capture the numerical error of a PINN solution for a PDE problem using a probabilistic approach in a setting with noise free input data. To make the computation tractable and efficient, we used Laplace approximations to infer the posterior and obtain uncertainties over the outputs of our network. We begin by modifying the PINN approach [Raissi et al., 2019] and constructing a probabilistic formulation of the PINN problem. To do this, we defined a prior over the weights and appropriate likelihood functions. However, computing the posterior $p(\theta \mid \mathcal{D})$ analytically is usually intractable. We use a Laplace approximation to efficiently approximate the posterior by a Gaussian around the mode of our posterior θ_{MAP}. We improve our approximation by tuning the hyperparameters of our probabilistic model (the variance of the prior and the likelihood) using the Laplace-approximated marginal likelihood. To obtain uncertainty estimates over the PINN solution we approximate the posterior over the network outputs $p(u(x) \mid x, \mathcal{D})$ by linearizing the neural network around our MAP estimate θ_{MAP}.

We then tested our method to find out how well it performs in solving PDE problems. In our experiments the PINN solution of our method is able to accurately solve the PDE with a simple network architecture. Next we examined the uncertainty estimates returned by our method. We show that our posterior is reasonably well calibrated and that the returned uncertainty estimates are sensible. To do this we analyzed both the uncertainties in a normal learning setting that one

might use to solve the PDE, as well as in settings where large regions of the domain are missing data. The uncertainty due to missing data in the spatial dimension in large areas of the domain is well reflected in our posterior. We conclude that our method is both able to solve PDE problems accurately, and return sensible uncertainty estimates over the solution.

In our construction the uncertainty inference does not hinder the performance of the PINN solution. Our proposed method does not require any modifications to the neural network learning process itself. In this way, possible improvements in the learning process and more powerful architectures can be exploited without having to modify the posterior inference algorithm. With our method, we obtain a good approximate solution to the PDE problem, and get computationally cheap uncertainty estimates of the numerical error on top of it.

5.2 Extensions and Future Work

PINNs have been shown to be have great performance in many real-world applications [Karniadakis et al., 2021]. We believe that incorporating cheap, reasonable uncertainty estimates, that can be computed on top of these solutions without modifying the training process, can add further utility to these methods. Since in real-world PDE problems the underlying solution function is usually unknown, an estimate of the numerical error can improve confidence in the result. In the reported experiments, we used only a simple neural network architecture and fixed hyperparameters. To fully evaluate the method, we also need to run the method on more powerful neural network architectures. We can also compare the quality of our uncertainty estimates and the efficiency of our method with other recent developments in scientific machine learning for solving PDE problems, by applying these methods to our noise free input setting. While the PDE problem we analyzed in this work is not straightforward to solve (see Section 4.2), we also need to test the method on different PDE problems. In particular, higher-dimensional PDE problems are of interest, as we might need to make further modifications to accurately solve them. If we try to solve more complicated PDE problems, we may also to employ a more powerful neural network structure, as well as find hyperparameters for the training procedure. We can use the Laplace-approximated marginal likelihood to help us choose a fitting model [Immer et al., 2021a], by comparing the likelihood functions for different choices of model architecture and hyperparameters. If we have a system that provides observations of the PDE at certain time steps, e.g. sensor data in a real world PDE, we can also incorporate this data into the PINN learning process with an additional data loss term. Since we can efficiently compute uncertainty estimates with the Laplace approximation we can also incorporate these uncertainties into the learning process. In this active

learning setting we could adaptively add collocation points during the training loop in regions of the domain where the posterior shows high uncertainty. This could improve the performance of the PINN solution and lead to a better calibrated posterior.

Bibliography

Lorena Barba and Gilbert Forsyth. Cfd python: the 12 steps to navier-stokes equations. *Journal of Open Source Education*, 2(16):21, 2019. URL https://doi.org/10.21105/jose.00021.

David Borthwick. *Introduction to partial differential equations*. Springer, 2017.

William E Boyce, Richard C DiPrima, and Douglas B Meade. *Elementary differential equations and boundary value problems*. John Wiley & Sons, 2021.

Erik Daxberger, Agustinus Kristiadi, Alexander Immer, Runa Eschenhagen, Matthias Bauer, and Philipp Hennig. Laplace redux-effortless bayesian deep learning. *Advances in Neural Information Processing Systems*, 34:20089–20103, 2021.

Lawrence C Evans. *Partial differential equations*, volume 19. American Mathematical Soc., 2010.

Uriel Frisch and Jérémie Bec. Burgulence. In *New trends in turbulence Turbulence: nouveaux aspects*, pages 341–383. Springer, 2001.

Philipp Hennig, Michael A Osborne, and Hans P Kersting. *Probabilistic Numerics*. Cambridge University Press, 2022.

Alexander Immer, Matthias Bauer, Vincent Fortuin, Gunnar Rätsch, and Khan Mohammad Emtiyaz. Scalable marginal likelihood estimation for model selection in deep learning. In *International Conference on Machine Learning*, pages 4563–4573. PMLR, 2021a.

Alexander Immer, Maciej Korzepa, and Matthias Bauer. Improving predictions of bayesian neural nets via local linearization. In *International Conference on Artificial Intelligence and Statistics*, pages 703–711. PMLR, 2021b.

Muhammad Izzatullah, Isa Eren Yildirim, Umair Bin Waheed, and Tariq Alkhalifah. Laplace hypopinn: Physics-informed neural network for hypocenter localization and its predictive uncertainty. *arXiv preprint arXiv:2205.14439*, 2022.

André Jaun, J Hedin, and T Johnson. Numerical methods for partial differential equations. *Swedish Netuniversity*, 1999.

George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Agustinus Kristiadi, Matthias Hein, and Philipp Hennig. Being bayesian, even just a bit, fixes overconfidence in relu networks. In *International conference on machine learning*, pages 5436–5446. PMLR, 2020.

David JC MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

David JC MacKay, David JC Mac Kay, et al. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

Keith W Morton and David Francis Mayers. *Numerical solution of partial differential equations: an introduction*. Cambridge university press, 2005.

Kazuki Osawa. Asdl: Automatic second-order differentiation library for pytorch. https://github.com/kazukiosawa/asdl, 2020.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Apostolos F Psaros, Xuhui Meng, Zongren Zou, Ling Guo, and George Em Karniadakis. Uncertainty quantification in scientific machine learning: Methods, metrics, and comparisons. *arXiv preprint arXiv:2201.07766*, 2022.

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.

Hippolyt Ritter, Aleksandar Botev, and David Barber. A scalable laplace approximation for neural networks. In *6th International Conference on Learning Representations, ICLR 2018-Conference Track Proceedings*, volume 6. International Conference on Representation Learning, 2018.

Makoto Takamoto, Timothy Praditia, Raphael Leiteritz, Dan MacKinlay, Francesco Alesiani, Dirk Pflüger, and Mathias Niepert. Pdebench: An extensive benchmark for scientific machine learning. *arXiv preprint arXiv:2210.07182*, 2022.

Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021.

Sifan Wang, Xinling Yu, and Paris Perdikaris. When and why pinns fail to train: A neural tangent kernel perspective. *Journal of Computational Physics*, 449: 110768, 2022.

Liu Yang, Dongkun Zhang, and George Em Karniadakis. Physics-informed generative adversarial networks for stochastic differential equations. *SIAM Journal on Scientific Computing*, 42(1):A292–A317, 2020.

Liu Yang, Xuhui Meng, and George Em Karniadakis. B-pinns: Bayesian physics-informed neural networks for forward and inverse pde problems with noisy data. *Journal of Computational Physics*, 425:109913, 2021.

Dongkun Zhang, Lu Lu, Ling Guo, and George Em Karniadakis. Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems. *Journal of Computational Physics*, 397:108850, 2019.

Selbständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind. Diese Masterarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Ort, Datum Unterschrift