POLITECNICO DI MILANO

School of Industrial and Information Engineering
Master of Science in Mathematical Engineering

Bayesian Physics-Informed Neural Networks for Inverse Uncertainty Quantification problems in Cardiac Electrophysiology

Supervisor: Prof. Andrea Manzoni
Co-Supervisor: Dott. Stefano Pagani

Thesis of:
Daniele Ceccarelli
Student ID: 919795

Academic Year 2019-2020

Ai miei nonni Giuseppe e Nicolò,
a mio zio Antonio.

Abstract

In this Thesis, we address Inverse Uncertainty Quantification problems related to partial differential equations (PDEs). These problems are very challenging from a numerical point of view since classical approaches require to numerically approximate multiple times a PDE to infer input parameters and their distributions.

To solve this issue, we rely on Physics-Informed Neural Networks (PINN), a novel technique in Scientific computing which combines a data-driven approach with the knowledge of physics-based models. In particular, we adopt a variant of PINN called Bayesian Physics-Informed Neural Network (B-PINN), which approximates the posterior distribution of the unknown parameters starting from some prior knowledge. From the resulting distribution, the uncertainty on the parameters estimates is quantified using suitable reliability intervals.

We focus on the Laplace equation to validate our method, and on the (nonlinear) Eikonal equation to address a challenging problem in cardiac electrophysiology: the reconstruction of the activation times' pattern in the cardiac tissue from noisy sparse measurements. We compare B-PINNs using different sampling methodologies varying both the number of data and the noise level.

The final application consists of estimating the posterior distribution of both the activation times and the conduction velocities (from which we extract the mean and the standard deviation) by using noisy data coming from sparse surface measurements. In this respect, we carry out some experiments relying on synthetic datasets in simple domains and we assess the accuracy of the implemented B-PINN technique and its computational performances. Also, new techniques like Active Sampling and Transfer Learning in subdomains are applied to improve accuracy in the activation time reconstruction and to speed up the training process.

Sommario

In questa tesi abbiamo affrontato problemi inversi di quantificazione dell'incertezza governati da equazioni alle derivate parziali. Questi problemi sono molto dispendiosi da un punto di vista numerico, dal momento che gli approcci classici richiedono di risolvere una PDE molte volte per trovare i parametri di input.
Per risolvere questi problemi, abbiamo utilizzato una nuova metodologia recentemente introdotta nel calcolo scientifico, le reti neurali fisicamente informate (PINN), che uniscono un approccio guidato dai dati con la conoscenza di un modello fisico. In particolare, abbiamo utilizzato una variante dei PINN in un contesto Bayesiano, le reti neurali Bayesiane fisicamente informate (B-PINN). Partendo da una conoscenza a priori dei parametri, si costruisce una distribuzione a posteriori e si quantifica la variabilità delle stime usando intervalli di credibilità.
Ci siamo concentrati in particolare sull'equazione di Laplace, per validare la nostra metodologia, e sull'equazione Eikonale (non lineare), che è stata invece usata per risolvere un difficile problema nell'elettrofisiologia cardiaca: la ricostruzione della mappa di attivazione nel tessuto cardiaco partendo da misurazioni sparse e affette da errore.
Abbiamo comparato il metodo B-PINNs usando diversi metodi di sampling variando sia il numero di dati che il livello di errore. Il nostro obiettivo finale è quello di stimare la distribuzione a posteriori sia dei tempi di attivazione che delle velocità di conduzione (dai quali calcoliamo media e deviazione standard) usando dati sparsi e affetti da rumore ottenuti a partire da mappe di attivazione cardiaca.
A questo proposito, abbiamo condotto diversi esperimenti usando dataset sintetici in domini semplici e verificando l'accuratezza delle metodologie implementate e le loro performance computazionali. Oltre a questo, nuove tecniche come Active Learning e Transfer Learning sono state applicate in questo campo per migliorare l'accuratezza e velocizzare la fase di training.

Acknowledgments

First of all, I would like to thank my thesis supervisors, Prof. Andrea Manzoni and Dott. Stefano Pagani, for their guidance throughout this thesis work and for all their advice. Their support and knowledge have encouraged me in all my research activity for this work.
I wish to extend my special thanks to all my friends and classmates for their company during these years and all the time shared together.
Finally, I want to show all my appreciation to my family for their support during these five years. I am grateful for the opportunity they gave to me. Without their endless encouragement I would not have achieved all of this.

D. C.

Contents

List of Figures

List of Tables

List of abbreviations

PDE	Partial Differential Equation
PDF	Probability Density Function
NN	Neural Network
BNN	Bayesian Neural Network
PINN	Physics-Informed Neural Network
B-PINN	Bayesian Physics-Informed Neural Network
MCMC	Markov Chain Monte Carlo
HMC	Hamiltonian Monte Carlo
SVGD	Stein Variational Gradient Descent
TL	Transfer Learning

Chapter 1

Introduction

In this Thesis we aim at solving Inverse Uncertainty Quantification (UQ) problems governed by partial differential equation (PDE) using a recently proposed methodology, called Bayesian Physics-Informed Neural Networks (B-PINNs), with special emphasis on the Eikonal equation for cardiac electrophysiology, that can be used to describe the activation times of the cardiac tissue. In the next sections of this introduction, we show how we can define this Inverse problem in Cardiac Electrophysiology and why it is important.

1.1 Inverse Uncertainty Quantification Problem

Inverse Uncertainty Quantification (from now on, Inverse UQ) problems involve the need of estimating input parameters of a given PDE giving some (sparse, noisy) measurements of the solution. These parameters can be, e.g., physical coefficients or geometrical parameters. These problems are very challenging from a numerical point of view, since with a classical approaches multiple solutions of the forward PDE problem are required when relying on usual sampling-based approaches, thus implying huge computational times and resources [2].
In this Thesis we are going to analyze a method to solve Inverse UQ problems for a generic (stationary, either linear or nonlinear) PDE, with a particular focus on cardiac applications, without implying the need of multiple solutions of the forward PDE model.

1.1.1 Inverse Problem

Let us start from a rigorous mathematical description of a generic Inverse UQ problem and classical methods to solve it [3],[4],[5].
Given Ω a bounded and regular domain and a differential operator $\mathscr{N}(\cdot;\boldsymbol{\theta}) : \mathbb{R}^d \to \mathbb{R}$, where

$d = 1,2,3$ denotes the spatial dimension, depending on a set of unknown parameters $\boldsymbol{\theta} \in \Theta \subset \mathbb{R}^J$, $J \geq 1 \in \mathbb{N}$, we can find a suitable function $u(\mathbf{x}) : \Omega \subset \mathbb{R}^d \to \mathbb{R}$ satisfying the following abstract PDE problem:

$$\begin{aligned} \mathscr{N}(u(\mathbf{x};\boldsymbol{\theta})) &= 0, & \mathbf{x} \in \Omega, \\ u(\mathbf{x};\boldsymbol{\theta}) &= BC(\mathbf{x}), & \mathbf{x} \in \partial\Omega, \end{aligned} \tag{1.1}$$

where $BC(\mathbf{x})$ is the boundary condition on $\partial\Omega$. The inverse problem consists in finding the parameters $\boldsymbol{\theta}$ given some (noisy) measurements of the PDE solution $u(\mathbf{x}) : \{u_i\}_{i=1}^n$ in some given locations $\{\mathbf{x}_i\}_{i=1}^n \subset \Omega$. Inverse problems are usually ill-posed and strongly dependent on the measurements of the solution: multiple solutions of $\boldsymbol{\theta}$ can be found and measurement noise could lead to unstable solutions [6].
In the standard approach, this problem can be formulated as a minimization problem over the parameter space for the mismatch between measurements of $\{u_i\}_{i=1}^n$ and $\{u(\mathbf{x}_i;\hat{\boldsymbol{\theta}})\}_{i=1}^n$, solution of (1.1) solving the forward model with a particular choice of parameter input vector $\hat{\boldsymbol{\theta}}$. After we have defined a suitable mismatch metric between the solutions, that depends on the choice of $\boldsymbol{\theta}$, for instance a mean quadratic loss :

$$\mathscr{L}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}(u(\mathbf{x}_i;\boldsymbol{\theta}) - u_i)^2, \tag{1.2}$$

we look for $\boldsymbol{\theta}^*$ such that:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}\in\Theta} \mathscr{L}(\boldsymbol{\theta}). \tag{1.3}$$

In order to find the best approximation $\boldsymbol{\theta}^*$, we generate M samples from the parameter space Θ and look for the one that minimize the loss. This approach has two limitations, as explained before: first, we need to solve the forward model for every samples in $\{\hat{\boldsymbol{\theta}}_i\}_{i=1}^M \subset \Theta$ we have generated from the parameter space in order to compute the solution. This implies to solve numerically a PDE multiple times, with a huge consumption of computational times and resources. The second issue is related with the noisy measurements: since only a point estimate of the true parameter vector $\boldsymbol{\theta}$ is found with this approach, we cannot take into account the propagation of the noise on the measurements on that estimation.
In this thesis we tackle both these issues using a new methodologies called Bayesian Physics-Informed Neural Networks [7]. Concerning the latter problem, a Bayesian approach (recalled in the next section) will be considered, whereas we will rely on a novel technique involving Neural Networks, Physics-Informed Neural Networks (PINNs) [8], to avoid the multiple solution of the given PDE system.

1.1.2 Bayesian Inverse UQ

Given $\boldsymbol{\theta} \in \Theta \subset \mathbb{R}^J$, input parameter vector, we define the forward problem as follows: find $u(\mathbf{x}) : \Omega \subset \mathbb{R}^d \to \mathbb{R}$ solution of the following PDE:

$$\begin{aligned} \mathscr{N}(u(\mathbf{x}); \boldsymbol{\theta}) &= 0, & \mathbf{x} \in \Omega, \\ u(\mathbf{x}; \boldsymbol{\theta}) &= BC(\mathbf{x}), & \mathbf{x} \in \partial\Omega. \end{aligned} \tag{1.4}$$

The solution $u(\mathbf{x})$ depends on the parameters $\boldsymbol{\theta}$. We can define this relation as:

$$u(\mathbf{x}) = f(\mathbf{x}, \boldsymbol{\theta}), \tag{1.5}$$

for a suitable function $f(\mathbf{x}, \boldsymbol{\theta}) : \Omega \subset \mathbb{R}^d \times \Theta \to \mathbb{R}$. In an Inverse UQ problem we start from n (noisy) measurements of the solution, namely $\mathbf{u}^{noisy} = \{u_i\}_{i=1}^n$, in some sparse locations $\{\mathbf{x}_i\}_{i=1}^n \subset \Omega$ to estimate the Probability Density Function (PDF) of the model parameters $\boldsymbol{\theta}$. The measurements could be affected by noise (as for instance in a real applications, where measurements are acquired from sensors). A reasonable assumption is that we have a white noise, comes from a Normal distribution, and it is independent from every measurements:

$$u_i = f(\mathbf{x}_i, \boldsymbol{\theta}) + \varepsilon_i, \;\; \varepsilon_i \sim \mathscr{N}(0, \sigma^2) \;\; \forall i = 1, \dots, n, \tag{1.6}$$

where σ^2 is the noise variance.
Given this model for the noise measurements, and after collecting the set of measurements $\mathbf{u}^{noisy} = \{u_i\}_{i=1}^n$, the problem of estimating input parameters $\boldsymbol{\theta}$ can be formulated in a Bayesian framework: find the posterior PDF of $\boldsymbol{\theta}$ conditioned to the measurements $\mathbf{u}^{noisy}$. To compute the posterior distribution of $\boldsymbol{\theta}$ we recall the Bayes Theorem:

$$\mathscr{P}(\boldsymbol{\theta} | \mathbf{u}^{noisy}) = \frac{p(\mathbf{u}^{noisy} | \boldsymbol{\theta}) \mathscr{P}(\boldsymbol{\theta})}{\mathscr{P}(\mathbf{u}^{noisy})}, \tag{1.7}$$

where we have to define the likelihood of $\mathbf{u}^{noisy}$ conditioned to $\boldsymbol{\theta}$, $p(\mathbf{u}^{noisy}|\boldsymbol{\theta})$, which encodes the probability that the measurements comes from a model with $\boldsymbol{\theta}$ as parameters, the prior distribution of $\boldsymbol{\theta}$, $\mathscr{P}(\boldsymbol{\theta})$, which is our prior knowledge of the parameters, and the denominator $\mathscr{P}(\mathbf{u}^{noisy})$. The latter, that involves the computation of a multidimensional integral on $\Theta \subset \mathbb{R}^J$, is equal to:

$$\mathscr{P}(\mathbf{u}^{noisy}) = \int_{\boldsymbol{\theta} \in \Theta} p(\mathbf{u}^{noisy} | \boldsymbol{\theta}) \mathscr{P}(\boldsymbol{\theta}) d(\boldsymbol{\theta}), \tag{1.8}$$

and plays the role of a normalizing constant in the equation (1.7). This integral could be intractable from a computational point of view, and since it is just a normalizing constant, its evaluation can be avoided.

The likelihood function of our measurements $\mathbf{u}^{noisy}$ conditioned on $\boldsymbol{\theta}$ could be easily computed from the noise model we have defined in (1.6):

$$\mathscr{P}(\mathbf{u}^{noisy}|\{\mathbf{x}_i\}_{i=1}^n, \boldsymbol{\theta}) \sim \mathscr{N}(\{f(x_i, \boldsymbol{\theta})\}_{i=1}^n, \Sigma) \tag{1.9}$$

where $\Sigma = \sigma^2\mathbb{I}$ with the same σ^2 defined for noise model.
For the prior distribution of the parameters $\boldsymbol{\theta}$ we have to select a distribution. For instance, imagine to know that these parameters follow a Normal distribution with zero mean and a variance γ^2:

$$\mathscr{P}(\boldsymbol{\theta}) \sim \mathscr{N}(\mathbf{0}, \Gamma), \tag{1.10}$$

where $\Gamma = \gamma^2\mathbb{I}$. This is a "non informative" prior [9], since we are providing a vague information for the parameters. Sometimes we can provide a more informative prior distributions for the parameters, driven by some physical knowledge or previous experiment, to have better results.

1.1.3 MCMC and estimators

The computation of the posterior PDF of $\boldsymbol{\theta}$ from the Bayes theorem could be intractable in a case where $J > 1$ or in a non-Gaussian models. This fact also makes the evaluation of a point estimators, as the maximum a posteriori estimator (MAP):

$$\boldsymbol{\theta}_{MAP} = \arg\max_{\boldsymbol{\theta}} \mathscr{P}(\boldsymbol{\theta}|\mathbf{u}^{noisy}), \tag{1.11}$$

or the conditional mean (CM):

$$\boldsymbol{\theta}_{CM} = \mathbb{E}[\boldsymbol{\theta}|\mathbf{u}^{noisy}] = \int_{\boldsymbol{\theta}\in\Theta} \boldsymbol{\theta}\mathscr{P}(\boldsymbol{\theta}|\mathbf{u}^{noisy})d(\boldsymbol{\theta}), \tag{1.12}$$

computationally intensive.
For the sake of Uncertainty Quantification, we also need to assess the variability of the posterior, for instance evaluating the conditioned standard deviation or building suitable confidence region.
When the parameter dimension J is really big, also a direct computation of the previous estimators could lead to intractable integrals: suitable sampling methods as Markov Chain Monte Carlo (MCMC) techniques are required to efficiently sample from the posterior distribution without exactly knowing it. MCMC methods explore the posterior distribution of $\boldsymbol{\theta}$ and collect a set of samples $\{\hat{\boldsymbol{\theta}}_i\}_{i=1}^M$ that approximates the posterior PDF.
In this thesis two different sampling methods are employed: a MCMC method called Hamiltonian Monte Carlo (HMC) and Stein Variational Gradient Descent (SVGD). Both these

methods allow us to sample from posterior distribution of our parameters $\boldsymbol{\theta}$ without directly computing it.
After collecting a set of samples $\{\hat{\boldsymbol{\theta}}_i\}_{i=1}^M$ we have an approximation of the posterior PDF, and we can also compute numerically the previous estimators. The evaluation of MAP estimator now becomes:

$$\boldsymbol{\theta}^*_{MAP} \approx \arg \max_{\hat{\boldsymbol{\theta}}_i,\, i=1,\dots,M} \mathscr{P}(\hat{\boldsymbol{\theta}}_i | \mathbf{u}^{noisy}), \tag{1.13}$$

while the conditional mean estimator:

$$\boldsymbol{\theta}^*_{CM} \approx \frac{1}{M}\sum_{i=1}^{M} \hat{\boldsymbol{\theta}}_i. \tag{1.14}$$

All these methods require multiple computations of forward PDE model: at every iteration of a MCMC, $u(\mathbf{x};\hat{\boldsymbol{\theta}}) = f(\mathbf{x},\hat{\boldsymbol{\theta}})$ is computed in order to evaluate the likelihood function. Since $u(\mathbf{x};\hat{\boldsymbol{\theta}})$ is the solution of (1.1) with a fixed parameters $\hat{\boldsymbol{\theta}}$, we compute a solution of a PDE multiple times, with a consumption of time and computational resources. In order to avoid a multiple solve of the PDE model (1.1), in this thesis we replace the numerical approximation of PDEs came out with classical methods (such as the finite element method) with Physics-Informed Neural Networks, which are Neural Networks involving a PDE constraint that are able to approximate the problem solution without having to recomputing it (but just updating the parameters in the NN).

1.2 Cardiac Electrophysiology

In this Thesis we will focus on a particular PDE model, called Eikonal Equation. This model could be used to describe, among a lot of different physical phenomena [10],[11],[12], also the propagation of electrical signal in the heart. In this context, we can imagine to solve an Inverse Problem starting from activation times measurements on the surface of the heart and reconstruct the physical parameters, that are the conduction velocities in every area of the heart tissue.
This might be of key importance to help clinicians to find low conduction velocities regions, that are usually related to electrical disorders. In the following sections we will go through a brief presentation of Cardiovascular diseases and present the Inverse problem of Cardiac Electrophysiology.

1.2.1 Arrhythmias

Cardiovascular diseases (CVDs) are a group of diseases of both heart and blood vessels. According to World Health Organization (WHO), these types of disease are the number

1 cause of death globally. In 2016, 17.9 million people died from CVDs, more than any other cause. Among these deaths, 85% are caused by heart attack and stroke. Cardiac Electrophysiology is the science that study the electrical activity in heart. In this thesis we will go through a specific subset of all Cardiovascular diseases, that are the ones related with Cardiac Electrophysiology diseases. Cardiac Electrophysiology focuses on the electrical activity of the heart, which triggers the cardiac contraction and so the movement of blood inside and between them. The diseases related to the disordered propagation of electrical signals in the heart are cardiac arrhythmias. With Arrhythmias we refers to any change from the normal sequence of electrical impulses. There are different types of them, like for instance Atrial Fibrillation, Ventricular Fibrillation, Tachycardia and Brachycardia.
To treat these types of problems, invasive methods like Radiofrequency catheter ablation (RFCA) could be needed. These procedures work directly on the surface tissue of the heart, using radiofrequency energy to burn some small parts of the tissue that cause irregular heart beat or any other problems.
The goal is to identify where to ablate, i.e. to find the region that cause the irregular heart beat. For this reason, before the RFCA procedure, the clinicians perform a map of cardiac activation times, to understand better how the electrical signal spreads in the heart and where it shows some improper behaviours.
3D electroanatomic mapping systems [13] are used to collect this data on the surface and construct the map. Our method might support the construction of accurate activation maps, with an Uncertainty Quantification, as explained in the following section.

1.2.2 Inverse Problem in Cardiac Electrophysiology

Clinical procedures for the treatment of electrical disorder phenomena usually relies on a few sparse noisy data of electro-anatomic activation maps, recorded on the cardiac surface using electrodes. These data are processed with deterministic interpolation methods to create an anatomical activation times map of the whole cardiac surface. From this measures, we are interested to compute the conductivity velocity properties of the tissue, to identify slow conduction areas that can be possibly related to arrhythmias.
To obtain these conduction velocity maps, we have to rely on the activation times map, using data usually affected by noise, due to the complicated measurement process. In addition to this, the classical methods developed so far do not take into account the physics behind the phenomena, that is a propagation of electric signal in an anisotropic medium. This is a standard data-driven approach, that usually works well when we have a lot of data, which requires a long acquisition time.

Looking at the physical modeling approach, the first model we can use to describe the electrical activity in heart is the so-called Bidomain model [14],[5]. To reduce the complexity of the former, the Monodomain model [15] was proposed. A further reduction can be done by using the Eikonal equation [16]. In this thesis we will use the latter.
The forward model consists of the Eikonal equation, providing the activation time in the domain, and receiving as input the conductivity tensor. On the other hand, the inverse problem - much harder to be solved - aims at reconstructing, starting from a set of measured activation times, the anisotropic conductivity tensor. As shown in the previous Section, we are interested not only to reconstruct a best approximation of the conduction velocities, but also in their posterior PDFs. In this application we obtain a measure of fidelity, i.e. how much we can trust our findings.

1.3 Thesis Organization

This thesis is organized as follows. In Chapter 2 we introduce the main building blocks of the framework we are using: Neural Networks, Bayesian Neural Networks and Physics-Informed Neural Networks.
In Chapter 3 we detail the methods we are going to use, namely Bayesian Physics-Informed Neural Networks. We deeply explain the Bayesian Model, and present the algorithms used: Hamiltonian Monte Carlo and Stein Variational Gradient Descent. We introduce also an Active Learning sampler, to compute the next location where we have to take a measurement in order to reduce the Uncertainty, and a new algorithm that uses Transfer Learning in subdomains, to obtain accurate results in a smaller parts of the domain without having to retrain our network.
In Chapter 4 we present some numerical experiments using the Laplace equation; in particular we validate our results against the true posterior distribution in a case where this latter can be computed in a closed form.
Chapter 5 introduces first the Cardiac application of this methods, based on Eikonal model. Then we propose some numerical experiments for both isotropic and anisotropic models, using different geometries. We then test our Active Learning sampler and Transfer Learning model.
Finally, our conclusions and future developments are reported in Chapter 6.

Chapter 2

Physics-Informed Neural Networks

2.1 Neural Networks

An Artificial Neural Network (ANN) is a computing system that reproduces in theory how a biological neural network works in our brain. The Perceptron [17] is one of the first and simple Neural Network and can be seen as the basic brick to build more complex Neural Network architectures, in the same way as our brain is build by a complex network of a very big number of single neurons.
The ingredients to build a Perceptron are the following ones:

- $\mathbf{x} \in \mathbb{R}^n$ input vector;
- $\mathbf{W} \in \mathbb{R}^n$ weights vector;
- $b \in \mathbb{R}$ scalar bias;
- $\sigma : \mathbb{R} \to \mathbb{R}$ activation function;
- $\hat{y} \in \mathbb{R}$ output.

Then, the relation that maps input into the output is defined as:

$$\hat{y} = \sigma(a) = \sigma\left(\mathbf{W}^T\mathbf{x} + b\right) = \sigma\left(\sum_{i=1}^{n}(w_i x_i) + b\right). \tag{2.1}$$

Since the activation function can be a non-linear function, the Perceptron maps the input onto the output in a non-linear way. The natural extension is the so-called Feed Forward Neural Network (FFNN), where we use neurons organized in layers (one for the Single Layer FFNN or more in Deep FFNN), and every output of a layer becomes the input of the following layer, in a cascade from the initial layer to the last one.

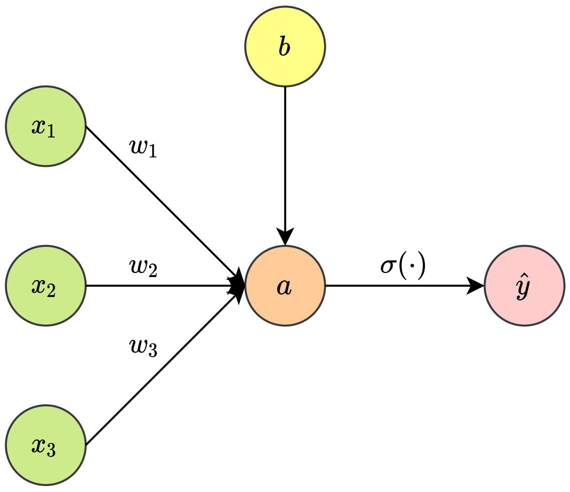


Figure 2.1: Example of Perceptron Architecture

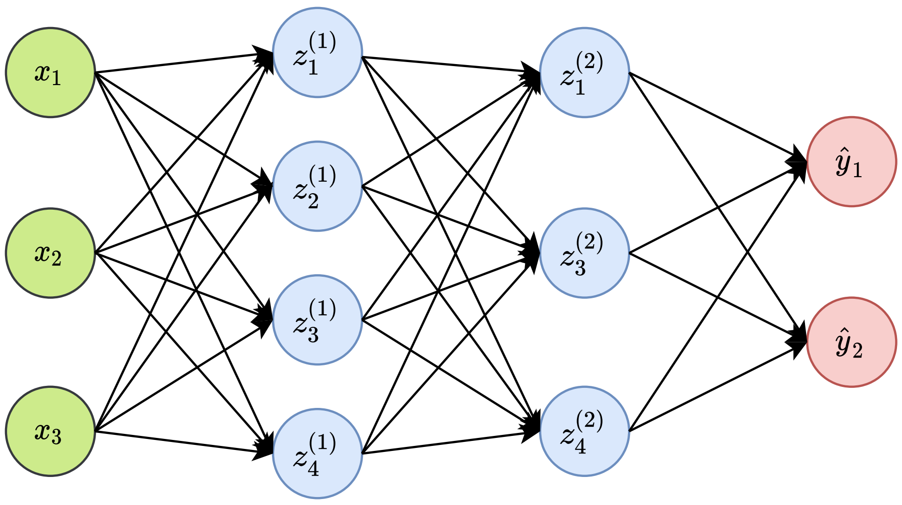


Figure 2.2: Example of a FFNN Architecture

We present in the following an example of FFNN. Given n the input vector dimension and d output vector dimension, we define L as the number of hidden layers of our FFNN. For every hidden layer $i = 1, \dots, L$ we define the number of neurons in each layer as l_i. In figure 2.2 an example of Feed Forward NN with 2 hidden layers, $l_1 = 4$ and $l_2 = 3$, $n = 3$ and $d = 2$.

For every layer ($i = 1, \dots, (L+1)$, we consider also the output layer) we define a weights matrix $\mathbf{W}^{(i)} \in \mathbb{R}^{l_{i-1} \times l_i}$ for the passage between the layer (i-1) to the layer (i) (where $l_0 = n$ input dimension, $l_{L+1} = d$ output dimension), a bias vector $\mathbf{b}^{(i)} \in \mathbb{R}^{l_i}$ and an activation function $\sigma^{(i)}\left(\mathbf{z}^{(i)}\right) : \mathbb{R}^{l_i} \to \mathbb{R}^{l_i}$. The input-output equation for a FFNN then reads as follows:

$$\begin{aligned}
\mathbf{z}^{(1)} =& \sigma^{(1)}\left(\mathbf{W}^{(1)}\mathbf{x}+\mathbf{b}^{(1)}\right),\\
\mathbf{z}^{(2)} =& \sigma^{(2)}\left(\mathbf{W}^{(2)}\mathbf{z}^{(1)}+\mathbf{b}^{(2)}\right),\\
&\dots\\
\mathbf{z}^{(L)} =& \sigma^{(L)}\left(\mathbf{W}^{(L)}\mathbf{z}^{(L-1)}+\mathbf{b}^{(L)}\right),\\
\hat{\mathbf{y}} =& \sigma^{(L+1)}\left(\mathbf{W}^{(L+1)}\mathbf{z}^{(L)}+\mathbf{b}^{(L+1)}\right).
\end{aligned} \tag{2.2}$$

In the following, to indicate the computation of the prediction vector $\hat{\mathbf{y}}$ using the input vector $\mathbf{x}$ (called Forward Propagation in a NN), we will use the following notation:

$$\hat{\mathbf{y}}(\mathbf{x}) = NN(\mathbf{x}), \tag{2.3}$$

where with $NN(\cdot)$ we consider all the passages in (2.2). Suppose now to have a dataset of couples $\mathbf{D} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$, where the vector $\mathbf{x}_i$ is the vector of the features, while $\mathbf{y}_i$ is the vector of the (real) outputs. For each $\{\mathbf{x}_i\}_{i=1}^N$, we use our FFNN to compute a prediction of the output $\mathbf{y}$, namely $\{\hat{\mathbf{y}}(\mathbf{x}_i)\}_{i=1}^N$, computed as:

$$\hat{\mathbf{y}}(\mathbf{x}_i) = NN(\mathbf{x}_i), \quad i = 1,\dots,N. \tag{2.4}$$

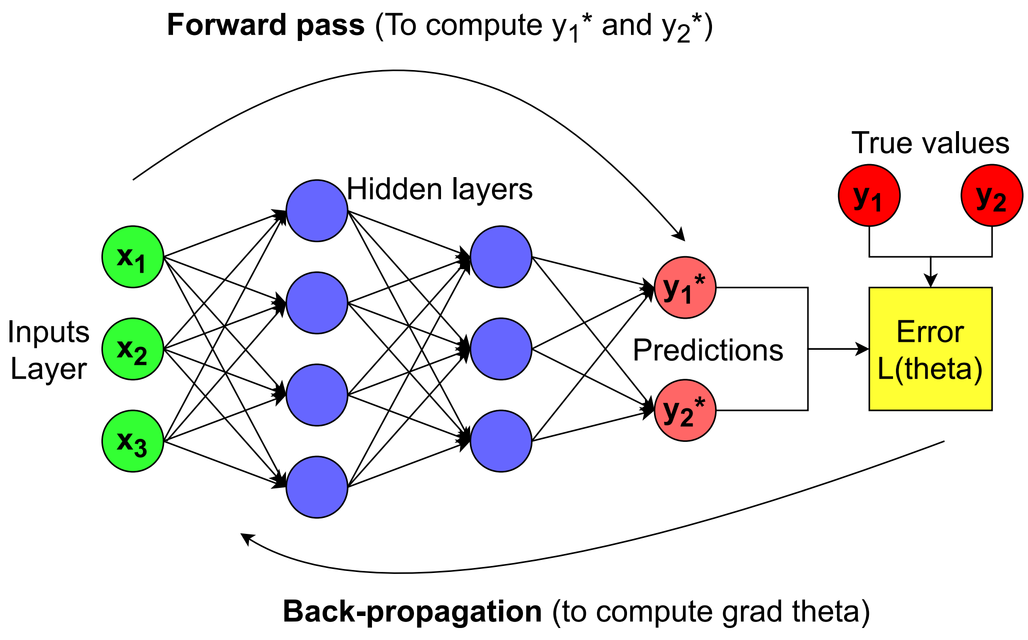
The final aim is to make the predictions $\{\hat{\mathbf{y}}(\mathbf{x}_i)\}_{i=1}^N$ sufficiently close to to the real outputs $\{\mathbf{y}_i\}_{i=1}^N$. To this goal we define a loss that measures the distance between $\hat{\mathbf{y}}$ and $\mathbf{y}$, which enable to define a measure of discrepancy that can be used to find the parameters in the network (all the matrices $\mathbf{W}$ and bias vectors $\mathbf{b}$ for every layer) in order to minimize this loss. Let $\boldsymbol{\theta}$ be all the parameters in our network, namely $\boldsymbol{\theta} = \{\boldsymbol{W}^{(1)}, \boldsymbol{b}^{(1)}, \dots, \boldsymbol{W}^{(L+1)}, \boldsymbol{b}^{(L+1)}\}$, where L is the number of hidden layers. We define the classical Mean Square Error (MSE) as follows:

$$\mathscr{L}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} ||\mathbf{y}^{\boldsymbol{\theta}}(\mathbf{x}_i) - \mathbf{y}_i||_2^2, \tag{2.5}$$

where with $\mathbf{y}^{\boldsymbol{\theta}}(\mathbf{x}_i)$ we indicate the output prediction on the input vector $\mathbf{x}_i$ using the NN with parameters $\boldsymbol{\theta}$ (previously indicated with $\hat{\mathbf{y}}(\mathbf{x}_i)$, but now we want to highlight the dependence from the parameters $\boldsymbol{\theta}$), namely:

$$\mathbf{y}^{\boldsymbol{\theta}}(\mathbf{x}_i) = NN^{\boldsymbol{\theta}}(\mathbf{x}_i), \quad i = 1,\dots,N. \tag{2.6}$$

The training procedure consists in updating the parameters in order to minimize $\mathscr{L}(\boldsymbol{\theta})$ using the Back-Propagation method [18], to compute the gradients of $\mathscr{L}(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$. After having collected the gradients (as shown in figure 2.3) we use an optimizer to iteratively update the parameters, i.e. the Gradient Descent [19]. The algorithm works as follows:



Forward pass (To compute y1* and y2*)
True values
y1
y2
Hidden layers
x1
x2
x3
Inputs Layer
y1*
Predictions
y2*
Error L(theta)
Back-propagation (to compute grad theta)

Figure 2.3: Forward pass and Back propagation

```
Algorithm 1: Gradient Descent
Initialization: Initialize $\boldsymbol{\theta}^0$ using a NN initializer, choose a number of total epochs
  E and a learning rate $\varepsilon$;
for epochs $e=1,\ldots,E$ do
    Compute $\mathbf{y}^{\boldsymbol{\theta}^{e-1}}(\mathbf{x}_i) = NN^{\boldsymbol{\theta}^{e-1}}(\mathbf{x}_i), \quad i = 1,\ldots,N$;
    Compute $\mathscr{L}(\boldsymbol{\theta}^{e-1})$ using (2.5);
    Compute $\nabla_{\boldsymbol{\theta}}\mathscr{L}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{e-1}}$ using Back-Propagation;
    Update $\boldsymbol{\theta}^e \leftarrow \boldsymbol{\theta}^{e-1} - \varepsilon\nabla_{\boldsymbol{\theta}}\mathscr{L}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{e-1}}$;
end
```

where ε is a parameter that measure the step size (also called learning rate) and $\nabla_{\boldsymbol{\theta}}\mathscr{L}(\boldsymbol{\theta})$ is the gradients of the loss with respect to $\boldsymbol{\theta}$ computed through Back-Propagation.

We have to repeat these steps (forward pass to compute the predictions, than compute the loss for all the couples in the dataset $\boldsymbol{D}$, then back propagation and updating parameters) for a fixed number of epochs (iterations) E in our algorithm, or until a convergence condition properly defined, with the final aim to find an approximation of the best parameters value $\boldsymbol{\theta}^*$ in its paramater space Θ:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}\in\Theta} \mathscr{L}(\boldsymbol{\theta}). \tag{2.7}$$

2.2 Bayesian Neural Network

When we are dealing with data, affected by noise, we are interested also in estimating how likely our model outcomes are, by estimating the uncertainty on the prediction. This process is called Uncertainty Quantification (UQ) of our results. We need to compute the posterior distribution of our estimates (and parameters $\boldsymbol{\theta}$), that provides us a bigger and complete information on the output.
However, neural network cannot provide any uncertainty estimation on its outputs, just a prediction. Several methods were proposed to provide an interval prediction also in NNs, such as a drop-out methods [20],[21]. However, a straightforward way to perform UQ in neural network is to use a Bayesian approach: Bayesian Neural Networks (BNNs) provide a natural extension of the NN framework, where we provide a prior distribution for every parameters in the network $\boldsymbol{\theta}$, and update these distribution with the knowledge of data using Bayes's rule, computing then the posterior distribution of $\boldsymbol{\theta}$. For each input now we can compute a distribution of the output, not only a point estimate which has its own uncertainty in terms of variance and standard deviation.
Even more than in the previous Bayesian setting, due to the huge dimension of the NN parameter set, computing directly a posterior distribution is impossible, since it involves some integral computations; for this reason we can use a Markov Chain Monte Carlo (MCMC) method to directly sample from the posterior distribution.
Hence, Bayesian Neural Networks compute the posterior PDF of their network parameters $\boldsymbol{\theta}$, not only the best approximation $\boldsymbol{\theta}^*$ that satisfies (2.7). According to the Bayes Rule, the posterior distribution of $\boldsymbol{\theta}$ depend on the likelihood functions of our data $\boldsymbol{D} = \{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$ and prior distribution over $\boldsymbol{\theta}$:

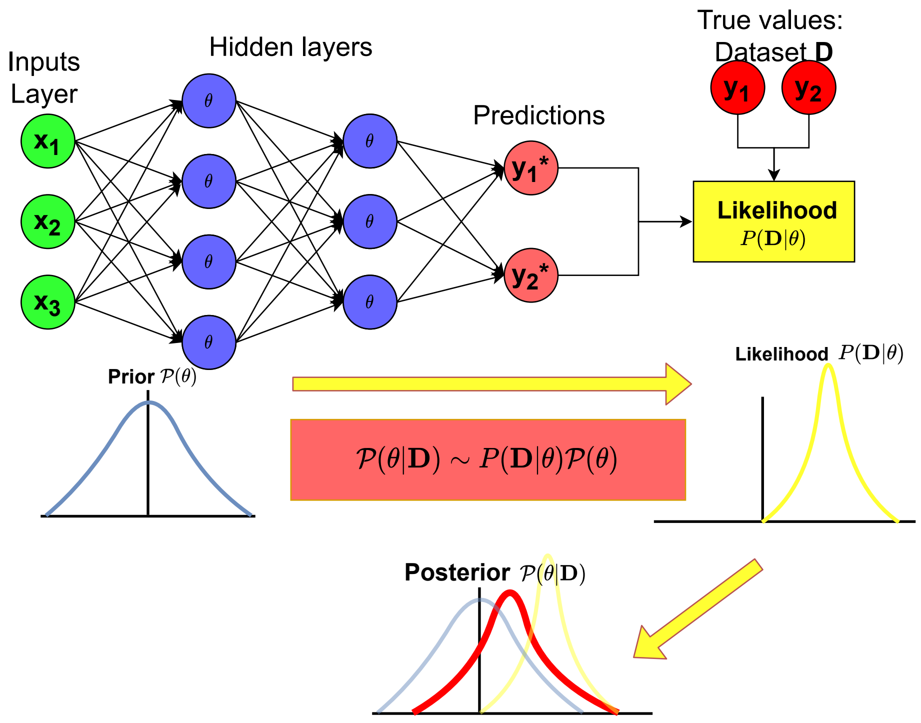
$$\mathscr{P}(\boldsymbol{\theta}|\mathbf{D}) \sim P(\mathbf{D}|\boldsymbol{\theta})\mathscr{P}(\boldsymbol{\theta}). \tag{2.8}$$

In (2.8) we update our prior beliefs on $\boldsymbol{\theta}$, that are the prior distribution $\mathscr{P}(\boldsymbol{\theta})$, with the likelihood of the data $\mathbf{D}$ we have, conditioning to the value of $\boldsymbol{\theta}$, that is $P(\mathbf{D}|\boldsymbol{\theta})$. A classical choice for the prior distribution for $\boldsymbol{\theta}$ could be:

$$\mathscr{P}(\boldsymbol{\theta}) \sim \mathscr{N}(\mathbf{0}, \mathbf{I}). \tag{2.9}$$

For the likelihood we have to first introduce the prediction of the BNN with parameters $\boldsymbol{\theta}$ on the features of the dataset $\boldsymbol{D}$, namely the set $\{\mathbf{x}_i\}_{i=1}^N$. We define these predictions through the BNN as shown before for the NN in (2.6). We define the set of these predictions as $\boldsymbol{D}^{\boldsymbol{\theta}} = \{\mathbf{x}_i, \mathbf{y}^{\boldsymbol{\theta}}(\mathbf{x}_i)\}_{i=1}^N$. Going back to the model, we can still use for the likelihood a Normal distribution over our model prediction $\boldsymbol{D}^{\boldsymbol{\theta}}$, namely:

$$\mathscr{P}(\boldsymbol{D}|\boldsymbol{\theta}) \sim \mathscr{N}(\boldsymbol{D}^{\boldsymbol{\theta}}, \sigma_D^2\mathbf{I}), \tag{2.10}$$



Inputs Layer
Hidden layers
Predictions
True values: Dataset D
x_1
x_2
x_3
θ
y_1^*
y_2^*
y_1
y_2
Likelihood $P(\mathbf{D}|\theta)$
Prior $\mathcal{P}(\theta)$
$\mathcal{P}(\theta|\mathbf{D}) \sim P(\mathbf{D}|\theta)\mathcal{P}(\theta)$
Likelihood $P(\mathbf{D}|\theta)$
Posterior $\mathcal{P}(\theta|\mathbf{D})$

Figure 2.4: Bayesian Neural Network update

with $\sigma_D > 0$ fixed standard deviation. (In the previous equation we should have written $\boldsymbol{D}^{[2]}$ and $\boldsymbol{D}^{\boldsymbol{\theta}[2]}$ since the likelihood is just on the $\mathbf{y}$, that is the second column in the datasets, but we keep them without the $[2]$ index for notation simplicity).
Figure 2.4 shows a simple scheme of how we compute the posterior distribution of our parameters $\boldsymbol{\theta}$ in a BNN.

Relying on a MCMC method, such as the Hamiltonian Monte Carlo, or variational inference methods such as the Stein Variational Gradient Descent, we can collect a set of M samples from the posterior distribution of our parameters $\boldsymbol{\theta}$, namely:

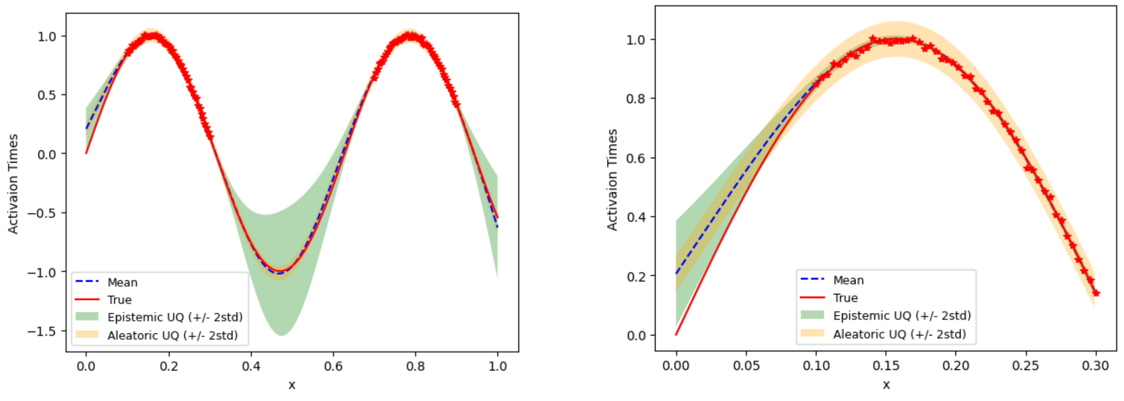
$$\{\boldsymbol{\theta}_i\}_{i=1}^M, \quad \boldsymbol{\theta}_i \sim \mathscr{P}(\boldsymbol{\theta}|\mathbf{D})\ i = 1,\dots,M, \tag{2.11}$$

which directly approximate the posterior distribution of $\boldsymbol{\theta}$. We can compute some estimators such as the sample mean and the sample variance, as:

$$\mu(\boldsymbol{\theta}|\mathbf{D}) \approx \frac{1}{M}\sum_{i=1}^{M}\boldsymbol{\theta}_i, \tag{2.12}$$

$$var(\boldsymbol{\theta}|\mathbf{D}) \approx \frac{1}{M}\sum_{i=1}^{M}(\boldsymbol{\theta}_i - \mu(\boldsymbol{\theta}|\mathbf{D}))^2. \tag{2.13}$$

This set of samples $\{\boldsymbol{\theta}_i\}_{i=1}^M$ can be also used to compute the posterior predictive distribution of our model $\boldsymbol{y}^{\theta}(\mathbf{x}) = NN^{\boldsymbol{\theta}}(\boldsymbol{x})$. For instance, we can compute the posterior predictive mean



((**a**)) Full view in domain (0,1) ((**b**)) Zoom in portion (0,0.3)

Figure 2.5: Aleatoric and Epistemic UQ in a simple regression task

and variance of $\boldsymbol{y}$ giving a location $\boldsymbol{x}^*$ as [22]:

$$\mathbb{E}_{\mathscr{P}(\boldsymbol{\theta}|\mathbf{D})}[\boldsymbol{y}|\mathbf{x}^*,\mathbf{D}] \approx \frac{1}{M}\sum_{i=1}^{M} \boldsymbol{y}^{\boldsymbol{\theta}_i}(\boldsymbol{x}^*), \tag{2.14}$$

$$Var_{\mathscr{P}(\boldsymbol{\theta}|\mathbf{D})}[\boldsymbol{y}|\mathbf{x}^*,\mathbf{D}] \approx \frac{1}{M}\sum_{i=1}^{M} \left(\boldsymbol{y}^{\boldsymbol{\theta}_i}(\boldsymbol{x}^*) - \mathbb{E}_{\mathscr{P}(\boldsymbol{\theta}|\mathbf{D})}[\boldsymbol{y}|\mathbf{x}^*,\mathbf{D}]\right)^2 + \sigma_D^2, \tag{2.15}$$

where $\boldsymbol{y}^{\boldsymbol{\theta}_i}(\boldsymbol{x}^*)$ is the prediction of our network with parameters $\boldsymbol{\theta}_i$, namely $\boldsymbol{y}^{\boldsymbol{\theta}_i}(\boldsymbol{x}^*) = NN^{\boldsymbol{\theta}_i}(\boldsymbol{x}^*)$. As shown by in the formula (2.15) the posterior predictive variance can be naturally divided in two different parts: the first part is called "Epistemic" uncertainty, while the second is called "Aleatoric" uncertainty. The former is determined by the variability among our predictions, in terms of samples generated with our MCMC algorithm, while the latter is more closely related to the noise of data we have.
To visualize and explain better the difference between these two parts, let analyze a simple regression problem in which the function to find is:

$$T(x) = sin(10x), \quad for\, x \in [0,1], \tag{2.16}$$

given a dataset $\mathbf{D}$ composed by 100 noisy evaluations grouped in the intervals [0.1,0.3] and [0.7,0.9] only. In figure 2.5 the differences between the two types of uncertainties can be highlighted. The Epistemic UQ bounds is strictly dependent on the distance among the points groups: if we are far from the exact points, the uncertainty is bigger and also the bound is larger, while in proximity of the data clouds (and also inside them) the uncertainty is very low.
The Aleatoric uncertainty in these models is determined by the variance parameter in the likelihood [22], σ_D, that in this example constant along all the domain.

2.3 Physics-Informed Neural Networks

Physics-Informed Neural Networks (PINNs) are a new powerful method introduced by Raissi, Karniadiakis and Perdikaris in [8] to solve both forward and inverse problems related with PDEs using neural networks. The idea behind PINNs is to create a bridge between data-driven and physics based modeling, encapsulating some knowledge of the system equations in the loss function to be minimized during the training stage.
Data-driven modeling and machine learning have experienced a big growth in popularity in the last few years, and Neural Networks are one of the main characters. Thanks to their universal function approximators property [23], their use in Scientific Computing has became massive, especially when "big data" are easily available. However, when the cost of data acquisition is bigger, neural networks fail to give good results in a "small data" regime. This is the case, e.g, of living system, where we need some invasive procedure to get the data from the patient.
On the other hand, physics-driven modeling is the classical way to study phenomena in real world through PDEs that well describes the physical behaviour of the system. In this case, we do not need any data acquisition, but the result rely on the knowledge of fundamental ingredients of PDE, like boundary conditions and initial conditions.
It seems natural to require a method that can use both our physical knowledge of the phenomena and both the few data that we have; PINN are neural networks that are trained to solve supervised learning tasks while respecting any given law of physics described by general nonlinear partial differential equations.

2.3.1 Physics-Informed Neural Network

Let us consider a generic stationary, nonlinear PDE, under the (abstract) form:

$$\begin{aligned} \mathscr{N}[u(\mathbf{x});\boldsymbol{\lambda}(\mathbf{x})] &= 0, & \mathbf{x} \in \Omega, \\ u(\mathbf{x}) &= BC(\mathbf{x}), & \mathbf{x} \in \partial\Omega, \end{aligned} \tag{2.17}$$

where $u = u(\mathbf{x})$ is the solution, $\mathscr{N}$ is a non-linear differential operator, $\boldsymbol{\lambda} = \boldsymbol{\lambda}(\mathbf{x})$ is a parametric field, Ω is the domain and $\partial\Omega$ is its boundary (where we have chosen to use just Dirichlet boundary conditions for simplicity). Thanks to PINNs, we can solve both forward problems, whose final goal is to find the solution $u(\mathbf{x})$ given $\boldsymbol{\lambda}(\mathbf{x})$, and inverse problems, whose goal is to reconstruct $\boldsymbol{\lambda}(\mathbf{x})$ from a set of (noisy) measurements of u. In this work we will focus on this latter class of problems.
In the forward problem, we aim at finding the solution $u(\mathbf{x})$ that is coherent with the PDE

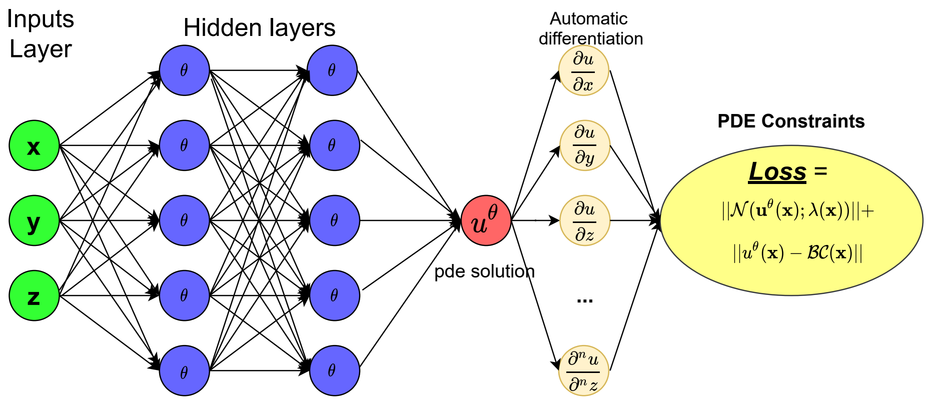

Inputs Layer
Hidden layers
Automatic differentiation
x
y
z
θ
u^θ
pde solution
$\frac{\partial u}{\partial x}$
$\frac{\partial u}{\partial y}$
$\frac{\partial u}{\partial z}$
...
$\frac{\partial^n u}{\partial^n z}$
PDE Constraints
Loss =
$||\mathcal{N}(\mathbf{u}^\theta(\mathbf{x}); \lambda(\mathbf{x}))||+$
$||u^\theta(\mathbf{x}) - \mathcal{BC}(\mathbf{x})||$

Figure 2.6: Physics-Informed Neural Network: forward problem scheme

defined in (2.17) using Neural Networks. In this case we have a dataset of boundary condition (BC) measurements (we do not need to know the analytical form of the boundary data BC, like in the classical formulation of PDEs), and we know all the coefficients that describe the PDE (in this case $\boldsymbol{\lambda}(\mathbf{x})$ is fixed and known), that means we know the operator $\mathcal{N}(\cdot)$. In Figure 2.6 we report a sketch of how PINNs work in this case: after computing the prediction $u^{\boldsymbol{\theta}}(\mathbf{x}) = NN^{\boldsymbol{\theta}}(\mathbf{x})$ as forward pass of our NN, as shown before in 2.6, Automatic Differentiation [24] is employed to compute exact partial derivatives (taking advantage of the input-output relationship, that is the neural network itself), and the loss of PDE constraints (2.17). Automatic differentiation in neural networks is a powerful method to compute derivatives: we can compute the exact derivative of the output given the input, instead of a numerical approximation as in common numerical differentiation (e.g., finite difference). This is possible because in a NN the relation that goes from outputs to inputs is made by the composition of an activation function and a linear combination using weight matrices and bias vectors, as shown in (2.2): we can compute the true derivative of all these steps (if we suppose to know the activation function derivative). More details on Automatic Differentiation can be found in Appendix A.
We consider two datasets:

- $\mathbf{R} \in \mathbb{R}^{n_r \times (d+1)}$: collocations points (dimension = n_r), that are points inside the domain Ω and where we impose the validity of PDE; for these points we don't know the actual value of $u(\mathbf{x})$, we know only that the PDE must holds, so we impose that the residual is zero in every points; $\mathbf{R}$ is a dataset composed by couples $(\mathbf{x}_r, 0)$, $\forall r = 1, \ldots, n_r$ (the zero in second position is just to indicate that, if we use the true solution $u(\mathbf{x})$, the residual of the PDE equation must be zero);

- $\mathbf{B} \in \mathbb{R}^{n_b \times (d+1)}$: boundary points (dimension = n_b), that are the points on the boundary $\partial\Omega$ and where we know the values of u; $\mathbf{B}$ is a dataset composed by couples = $(\mathbf{x}_b, BC(\mathbf{x}_b))$, $\forall b = 1, \ldots, n_b$.

The final goal is that the NN will map the an input vector $\mathbf{x}$ in the NN output (with parameters $\boldsymbol{\theta}$) $u^{\boldsymbol{\theta}}(\mathbf{x})$ that is a good approximation of the PDE solution $u(x)$, solution of the forward problem (2.17). The key passage with PINNs is that, beyond the fact that we use the boundary conditions in $\mathbf{B}$ to enforce that $u^{\boldsymbol{\theta}}$ will be close to the solution u in the boundary points, we are requiring the NN to solve the PDE in all collocation points in $\mathbf{R}$: inside the domain Ω, where we do not have any measurements of the real solution u, the only constraint that drives the prediction $u^{\boldsymbol{\theta}}(\mathbf{x})$ close to the PDE solution $u(\mathbf{x})$ is the PDE constraint itself.
For every point $\mathbf{x}$ in $\mathbf{R}$ and $\mathbf{B}$ we compute the relative prediction of the solution u with our NN (parameters $\boldsymbol{\theta}$), namely $u^{\boldsymbol{\theta}}(\mathbf{x})$ with a forward pass in the network (as shown in (2.6)); then, just for the collocation points, we also need all the derivatives (with Automatic Differentiation) to compute the PDE constraint that we want to minimize, namely the operator $\mathscr{N}$ residual:

$$R^{\boldsymbol{\theta}}(\mathbf{x}_r) = \mathscr{N}\left[u^{\boldsymbol{\theta}}(\mathbf{x}_r); \boldsymbol{\lambda}(\mathbf{x}_r)\right], \quad \mathbf{x}_r \in \boldsymbol{R}. \tag{2.18}$$

Finally, the loss (where $\boldsymbol{\theta}$, as before, represent all the parameters in our network) can be defined as:

$$\mathscr{L}(\boldsymbol{\theta}) = \frac{1}{n_r}\sum_{r=1}^{n_r} ||R^{\boldsymbol{\theta}}(\mathbf{x}_r)||^2 + \frac{1}{n_b}\sum_{b=1}^{n_b} ||u^{\boldsymbol{\theta}}(\mathbf{x}_b) - BC(\mathbf{x}_b))||^2. \tag{2.19}$$

By minimizing the loss function (2.19), the NN will update his parameters $\boldsymbol{\theta}$ in order to approximate the true solution $u(\mathbf{x})$ with its prediction $u^{\boldsymbol{\theta}}(\mathbf{x})$, since it has to satisfy the PDE inside the domain and the boundary conditions. We recall that, differently from the second term in the loss where we are requiring that the prediction of u in boundary points $u^{\boldsymbol{\theta}}(\mathbf{x}_b)$ should be close to the real boundary value $BC(\mathbf{x}_b)$, the first term is requiring that the operator $\mathscr{N}$ applied to $u^{\boldsymbol{\theta}}(\mathbf{x}_r)$ in all the collocation points must be close to zero. This is true because, if in the end the prediction with NN $u^{\boldsymbol{\theta}}(\mathbf{x})$ is equal to the real solution $u(\mathbf{x})$, the operator $\mathscr{N}$ applied to it must be zero, as shown in (2.17).

2.3.2 PINNs for the Inverse Problem

So far we have sketched the idea exploited by PINNs in order to approximate the PDE solution, given its parameters $\boldsymbol{\lambda}(\mathbf{x})$ and the boundary conditions.
When dealing with the inverse problem, we suppose to know some sparse and noisy measurements of the solution $u(\mathbf{x})$ and we want to reconstruct the parametric field $\boldsymbol{\lambda} = \boldsymbol{\lambda}(\mathbf{x})$, as well as the solution $u(\mathbf{x})$ of the PDE.
To identify the unknown field $\boldsymbol{\lambda}(\mathbf{x})$, we rely on an additional dataset $\mathbf{D} \in \mathbb{R}^{n_d \times (d+1)}$ of n_d

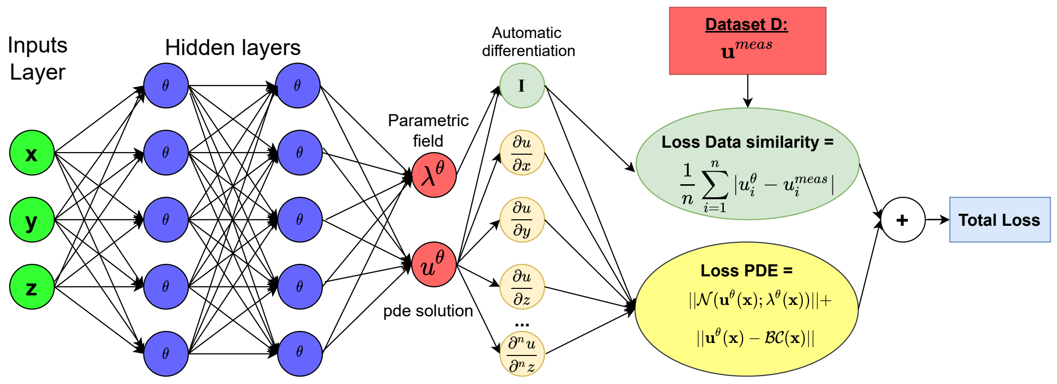

Inputs Layer
Hidden layers
Automatic differentiation
Dataset D: $\mathbf{u}^{meas}$
Parametric field
pde solution
Loss Data similarity = $\frac{1}{n}\sum_{i=1}^{n}|u_i^\theta - u_i^{meas}|$
Loss PDE = $||\mathcal{N}(\mathbf{u}^\theta(\mathbf{x});\lambda^\theta(\mathbf{x}))||+$ $||\mathbf{u}^\theta(\mathbf{x}) - \mathcal{BC}(\mathbf{x})||$
Total Loss

Figure 2.7: Physics-Informed Neural Network: inverse problem scheme

measurements of u, $\boldsymbol{D} = \{\mathbf{x}_d, u_d\}_{d=1}^{n_d}$.
After a forward pass in the network, we collect the prediction of $u(\mathbf{x})$ and $\boldsymbol{\lambda}(\mathbf{x})$ as outputs of the NN, namely $u^{\boldsymbol{\theta}}(\mathbf{x}) = NN_u^{\boldsymbol{\theta}}(\mathbf{x})$ and $\boldsymbol{\lambda}^{\boldsymbol{\theta}}(\mathbf{x}) = NN_{\boldsymbol{\lambda}}^{\boldsymbol{\theta}}(\mathbf{x})$ (extending the notation presented in (2.6)), and then we use the automatic differentiation to compute the derivatives required, such as $\frac{\partial u}{\partial x}$ and $\frac{\partial u}{\partial y}$. After computing the derivatives forming the operator $\mathcal{N}$ for each collocation points, we end up with the same physics-informed neural network loss as before, but now including both the data-driven loss and physics-driven loss. We compute the prediction for the residual $\mathbf{R}^{\boldsymbol{\theta}}$ on the collocation points as shown before, but now using also the prediction for $\boldsymbol{\lambda}$: $R^{\boldsymbol{\theta}}(\mathbf{x}_r) = \mathcal{N}\left[u^{\boldsymbol{\theta}}(\mathbf{x}_r); \boldsymbol{\lambda}^{\boldsymbol{\theta}}(\mathbf{x}_r)\right], \quad r = 1, \dots, n_r.$
After we have introduced two additional weights α_{data} and α_{pde}, we can define our loss as:

$$\mathcal{L}(\boldsymbol{\theta}) = \alpha_{data}\frac{1}{n_d}\sum_{d=1}^{n_d}||u^{\boldsymbol{\theta}}(\mathbf{x}_d) - u_d||^2 + \alpha_{pde}\left(\frac{1}{n_r}\sum_{r=1}^{n_r}||R^{\boldsymbol{\theta}}(\mathbf{x}_r)||^2 + \frac{1}{n_b}\sum_{b=1}^{n_b}||u^{\boldsymbol{\theta}}(\mathbf{x}_b) - BC(\mathbf{x}_b))||^2\right). \tag{2.20}$$

Solving the PDE and matching the measured data can then be done at the same time, by minimizing the loss function, using a suitable numerical optimization procedure (such as Gradient Descent or the Adam optimizer), to find an approximation of $\boldsymbol{\theta}^*$:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}). \tag{2.21}$$

After having selected the best approximation $\boldsymbol{\theta}^*$, we compute our final prediction for both solution the $u(\mathbf{x})$ and $\boldsymbol{\lambda}(\mathbf{x})$ as a simple forward pass in our network, obtaining $u^{\boldsymbol{\theta}^*}(\mathbf{x})$ and $\boldsymbol{\lambda}^{\boldsymbol{\theta}^*}(\mathbf{x})$, respectively.
Figure 2.7 shows the scheme related to the solution of an inverse problem with PINNs, that is similar to the one reported in figure 2.6, with in addition the output $\boldsymbol{\lambda}$ (in the figure considered as scalar field for simplicity) for the NN and a data similarity part in the loss computation. We want to highlight the importance of the physical knowledge in this example: using just the datasets $\boldsymbol{D} = \{\mathbf{x}_d, u_d\}_{d=1}^{n_d}$ and $\boldsymbol{B} = \{\mathbf{x}_b, BC(\mathbf{x}_b)\}_{b=1}^{n_b}$ we could compute an approximation

of u inside the domain Ω using a simple Neural Network. In this case a NN would find a classic interpolation between the measurements of u we have and the boundary conditions. After having collected this classic interpolation $u^{interp}(\mathbf{x})$, in a second stage, we could infer the values of $\boldsymbol{\lambda}(\mathbf{x})$ using the PDE constraint (2.17). The main advantage of PINNs is that we use the collocation points in $\boldsymbol{R}$ to enforce the validity of the PDE model (2.17). In this way our prediction $u^{\boldsymbol{\theta}}(\mathbf{x})$ is not a simple interpolation between the measurements of u we have in $\boldsymbol{D}$ and $\boldsymbol{B}$, but this is also solution of the PDE: $u^{\boldsymbol{\theta}}(\mathbf{x})$ and $\boldsymbol{\lambda}^{\boldsymbol{\theta}}(\mathbf{x})$ are now computed using the data we have $\boldsymbol{D}$ and $\boldsymbol{B}$ but also solving the (2.17).

Chapter 3

Bayesian Physics-Informed Neural Networks

As Neural Networks, also Physics-Informed Neural Networks can be extended within a Bayesian framework, leading to the so-called Bayesian Physics-Informed Neural Networks (B-PINNs) [7]. This is achieved by using a Bayesian Neural Network instead of a Neural Network. B-PINNs compute the posterior distribution of the network parameters $\boldsymbol{\theta}$ conditioning to the available datasets, in this case measurements of the PDE solutions included in $\boldsymbol{D}$ and collocation points $\boldsymbol{R}$ where we enforce the PDE constraint, as explained in the previous Chapter. As for BNNs, we rely on the Bayes theorem, defining a prior distribution for $\boldsymbol{\theta}$ and a likelihood functions for both measurements in $\boldsymbol{D}$ and collocation points in $\boldsymbol{R}$.
To sample from the posterior, we wil finally use MCMC methods, since its analytical computation could be intractable.

3.1 Model

In this section we define the Bayesian model underlying B-PINN; in particular, we specify likelihood functions on our data and the prior distribution of our parameters (all the network weights $\boldsymbol{\theta}$, and optionally also the variances in the likelihoods, if we consider them as parameters). Then we show how to derive the posterior distribution.
In the following we consider the same PDE model described in the previous Chapter in (2.17), namely:

$$\begin{aligned} \mathscr{N}[u(\mathbf{x});\boldsymbol{\lambda}(\mathbf{x})] &= 0, \qquad && \mathbf{x} \in \Omega, \\ u(\mathbf{x}) &= BC(\mathbf{x}), && \mathbf{x} \in \partial\Omega. \end{aligned} \tag{3.1}$$

3.1.1 Datasets and Likelihood functions

In the training phase of the B-PINN, we use a small dataset of noisy evaluations of the PDE solution. Let us denote this dataset as a matrix $\mathbf{D}$ of n_d exact sparse measurements u and their relative locations $\mathbf{x}$, $\mathbf{D} = \{\mathbf{x}_i, u_i\}_{i=1}^{n_d}$, that will be used to partially reconstruct the solution (for notation simplicity we do not consider the previously introduced dataset $\boldsymbol{B}$ of boundary conditions. We can always consider it as part of the dataset $\boldsymbol{D}$ of measurements, where we choose some locations on the boundary).

For the sake of testing, we assume that $\mathbf{D}$ is given by measurements of the exact solution $\mathbf{D}^*$ plus a white noise, that can be described as a normally distributed error $\boldsymbol{\varepsilon}$ of mean 0 and standard deviation σ (noise level):

$$u_i = u^*(\mathbf{x}_i) + \varepsilon(i), \;\; where\; \varepsilon(i) \sim \mathcal{N}(0, \sigma^2), \;\; i = 1, \dots, n_d. \tag{3.2}$$

Denoting by:

$$\mathbf{D}^{\boldsymbol{\theta}} = \{\mathbf{x}_i, u^{\boldsymbol{\theta}}(\mathbf{x}_i)\}_{i=1}^{n_d} \tag{3.3}$$

the approximation through the NN with parameters $\boldsymbol{\theta}$ (as shown in (2.6)) of the noisy data $\mathbf{D}$, we set a likelihood distribution of:

$$\mathscr{P}(\mathbf{D}|\boldsymbol{\theta}, \sigma_D) \sim \mathcal{N}(\mathbf{D}^{\boldsymbol{\theta}}, \Sigma_D) \tag{3.4}$$

where $\Sigma_D = \sigma_D^2 \mathbf{I}$, and σ_D can be a fixed value or an hyperparameter.

In addition to the dataset $\mathbf{D}$, we know that the underlying relation between the solution $u(\mathbf{x})$ and parameters $\boldsymbol{\lambda}(\mathbf{x})$ is given by the PDE equation we are going to solve. Let us introduce a dataset of n_r collocation points, selected in the spatial domain, where we impose that the NN must fulfill the PDE constraint. Hence, we denote the evaluation of the residual of the equation as a dataset $\mathbf{R} = \{\mathbf{x}_r, 0\}_{r=1}^{n_r}$ of n_r collocation points, where we force the NN to satisfy the PDE constraint (the zeros in the second position, as explained in the previous Chapter, indicate that using the true PDE solution the residual is equal to zero). We can compute the residual of collocation points with the NN, $\mathbf{R}^{\boldsymbol{\theta}}$, starting from $u^{\theta}(\mathbf{x})$ and $\boldsymbol{\lambda}^{\boldsymbol{\theta}}(\mathbf{x})$ and using the operator $\mathcal{N}$ of the PDE as:

$$\mathbf{R}^{\boldsymbol{\theta}} = \{\mathbf{x}_r, \mathcal{N}(u^{\boldsymbol{\theta}}(\mathbf{x}_r); \boldsymbol{\lambda}^{\boldsymbol{\theta}}(\mathbf{x}_r))\}_{r=1}^{n_r}. \tag{3.5}$$

As previously explained, the goal is to make $\boldsymbol{R}^{\boldsymbol{\theta}}$ (considering of course just the second column, namely $\mathcal{N}(u^{\boldsymbol{\theta}}(\mathbf{x}_r); \boldsymbol{\lambda}^{\boldsymbol{\theta}}(\mathbf{x}_r))$ close to $\boldsymbol{R}$ (that, considering here the second column, is a vector of zeros): we want that our approximation with the NN of u and $\boldsymbol{\lambda}$, namely $u^{\boldsymbol{\theta}}$ and $\boldsymbol{\lambda}^{\boldsymbol{\theta}}$, satisfy (or, at least, be very close to satisfy) the PDE equation 3.1, hence the residual should be close to zero.

Going back to the Bayesian model, we can suppose also in this case a Normal likelihood of the dataset $\mathbf{R}$ over our prediction $\boldsymbol{R^{\theta}}$ with an additive error:

$$\mathscr{P}(\mathbf{R}|\boldsymbol{\theta},\sigma_R) \sim \mathscr{N}(\mathbf{R}^{\boldsymbol{\theta}},\Sigma_R) \tag{3.6}$$

where $\Sigma_R = \sigma_R^2\mathbf{I}$, and also in this case σ_R can be fixed or treated as a further parameter.

3.1.2 Prior distributions

Let $\boldsymbol{\theta} \in \mathbb{R}^J$ be the vector of all the weights of our NN. We define a prior distribution for every entry $\theta_j,\ j = 1,\dots,J$ of the vector $\boldsymbol{\theta}$. Since we do not have any hint on the distribution of these parameters (except that they should be initialized near zero to prevent instabilities [25]), we use a Normal distribution, with zero mean and fixed variance $\Gamma = \gamma^2\mathbf{I}$:

$$\mathscr{P}(\theta_j) \sim N(0,\gamma^2), \quad \forall j = 1,\dots,J, \tag{3.7}$$

or a T-student distribution, with zero mean and n fixed parameter:

$$\mathscr{P}(\theta_j) \sim StudentT(n), \quad \forall j = 1,\dots,J. \tag{3.8}$$

If we consider the variances of likelihoods σ_D^2 and σ_R^2 as hyperparameters, we also need to define a prior for them. In this case, we have some prior knowledge on these parameters:

- the prior $\mathscr{P}(\sigma_D^2)$ on σ_D^2 employed knowledge on the measurements error, for instance the specific error of the sensor. In a synthetic experiment, where we add a noise ε to our measurements in the dataset $\boldsymbol{D}$ as seen before in (3.2), a reasonable choice is to give a prior to σ_D^2 centered in a value close to the variance σ^2 of ε;
- the prior $\mathscr{P}(\sigma_R^2)$ on σ_R^2 could be seen as a way to measure the preliminary knowledge of the physical model. If we believe that our PDE explain well the physical phenomena we are interested in, we should set a prior for σ_R^2 that takes small values.

As priors we can use an Inverse Gamma with fixed parameters for both:

$$\mathscr{P}(\sigma_D^2) \sim Inv-Gamma(\alpha_1,\beta_1), \tag{3.9}$$

$$\mathscr{P}(\sigma_R^2) \sim Inv-Gamma(\alpha_2,\beta_2), \tag{3.10}$$

where we choose $\alpha_1 = \alpha_2 = 2$, while β_1 and β_2 encode our uncertainties on both the data we have collected and the equation behind the physical problem, as explained before.

3.1.3 Posterior distributions

Let us finally compute the posterior distribution of our parameters ($\boldsymbol{\theta}$, as well as σ_D and σ_R if we choose to consider them as hyperparameters) using the exact noisy dataset $\mathbf{D}$ and the residual on the collocation points $\mathbf{R}$, and employing the likelihoods and priors defined before. We can compute the posterior distribution for $\boldsymbol{\theta}$, σ_D and σ_R simply following the Bayes Rule, under the hypothesis of independence between σ_D and σ_R:

$$\mathscr{P}(\boldsymbol{\theta}, \sigma_D, \sigma_R | \mathbf{D}, \mathbf{R}) \propto \mathscr{P}(\mathbf{D}|\boldsymbol{\theta}, \sigma_D)\, \mathscr{P}(\mathbf{R}|\boldsymbol{\theta}, \sigma_R)\, \mathscr{P}(\boldsymbol{\theta})\, \mathscr{P}(\sigma_D)\, \mathscr{P}(\sigma_R) \tag{3.11}$$

This is the posterior distribution of the network parameters $\boldsymbol{\theta}$. After computing it, we can also compute posterior predictive distributions for both the PDE solution $u(\mathbf{x})$ and the parametric field $\boldsymbol{\lambda}(\mathbf{x})$. Given an input position $\mathbf{x}^*$, the posterior predictive distributions for both $u(\mathbf{x}^*)$ and $\boldsymbol{\lambda}(\mathbf{x}^*)$ are:

$$\begin{aligned}
\mathscr{P}(u^*|\mathbf{x}^*, \mathbf{D}, \mathbf{R}) &= \int_{\boldsymbol{\theta}, \sigma_D} \mathscr{P}(u^*|u^{\boldsymbol{\theta}}(\mathbf{x}^*), \boldsymbol{\theta}, \sigma_D)\mathscr{P}(\boldsymbol{\theta}, \sigma_D|\mathbf{D}, \mathbf{R}) d\boldsymbol{\theta} d\sigma_D, \\
\mathscr{P}(\boldsymbol{\lambda}^*|\mathbf{x}^*, \mathbf{D}, \mathbf{R}) &= \int_{\boldsymbol{\theta}, \sigma_R} \mathscr{P}(\boldsymbol{\lambda}^*|\boldsymbol{\lambda}^{\boldsymbol{\theta}}(\mathbf{x}^*), \boldsymbol{\theta}, \sigma_R)\mathscr{P}(\boldsymbol{\theta}, \sigma_R|\mathbf{D}, \mathbf{R}) d\boldsymbol{\theta} d\sigma_R,
\end{aligned} \tag{3.12}$$

where:

$$\begin{aligned}
\mathscr{P}(u^*|u^{\boldsymbol{\theta}}(\mathbf{x}^*), \boldsymbol{\theta}, \sigma_D) &\sim \mathscr{N}(u^{\boldsymbol{\theta}}(\mathbf{x}^*), \sigma_D^2), \\
\mathscr{P}(\boldsymbol{\lambda}^*|\boldsymbol{\lambda}^{\boldsymbol{\theta}}(\mathbf{x}^*), \boldsymbol{\theta}, \sigma_R) &\sim \mathscr{N}(\boldsymbol{\lambda}^{\boldsymbol{\theta}}(\mathbf{x}^*), \sigma_R^2 \mathbf{I}).
\end{aligned} \tag{3.13}$$

As before, we use Markov Chain Monte Carlo methods to sample from this posterior distribution. In this framework, using a MCMC method, we can collect a set of M samples of our parameters:

$$\{\boldsymbol{\theta}_m, \sigma_{Dm}, \sigma_{Rm}\}_{m=1}^M, \quad \boldsymbol{\theta}_m, \sigma_{Dm}, \sigma_{Rm} \sim \mathscr{P}(\boldsymbol{\theta}, \sigma_D, \sigma_R|\mathbf{D}, \mathbf{R}), \; m = 1, \dots, M, \tag{3.14}$$

and also then the posterior predictive distribution as:

$$\begin{aligned}
\mathscr{P}(u^*|\mathbf{x}^*, \mathbf{D}, \mathbf{R}) &\approx \frac{1}{M} \sum_{m=1}^M \mathscr{P}(u^*|u^{\boldsymbol{\theta}_m}(\mathbf{x}^*), \boldsymbol{\theta}_m, \sigma_{Dm}, \sigma_{Rm}), \\
\mathscr{P}(\boldsymbol{\lambda}^*|\mathbf{x}^*, \mathbf{D}, \mathbf{R}) &\approx \frac{1}{M} \sum_{m=1}^M \mathscr{P}(\boldsymbol{\lambda}^*|\boldsymbol{\lambda}^{\boldsymbol{\theta}_m}(\mathbf{x}^*), \boldsymbol{\theta}_m, \sigma_{Dm}, \sigma_{Rm}).
\end{aligned} \tag{3.15}$$

3.2 Methods

In all the algorithms that follow, we need to compute the log posterior probability that can be easily derived from the posterior distribution written above:

$$\begin{aligned} L(\boldsymbol{\theta}) = log(Posterior) = & log\left(\mathscr{P}\left(\boldsymbol{D}|\boldsymbol{\theta},\sigma_D\right)\right) + log\left(\mathscr{P}\left(\boldsymbol{R}|\boldsymbol{\theta},\sigma_R\right)\right) + \\ & log\left(\mathscr{P}\left(\boldsymbol{\theta},\sigma_D,\sigma_R\right)\right) \end{aligned} \tag{3.16}$$

where:

$$log\left(p\left(\boldsymbol{D}|\boldsymbol{\theta},\sigma_D\right)\right) \propto \left(-\frac{1}{2{\sigma_D}^2}\sum_{d=1}^{n_d}\left(D^{\boldsymbol{\theta}}(\mathbf{x}_d)-D_d\right)^2 + \frac{n_d}{2}log\left(\frac{1}{{\sigma_D}^2}\right)\right) \tag{3.17}$$

and

$$log\left(p\left(\boldsymbol{R}|\boldsymbol{\theta},\sigma_R\right)\right) \propto \left(-\frac{1}{2{\sigma_R}^2}\sum_{r=1}^{n_r}\left(R^{\boldsymbol{\theta}}(\mathbf{x}_r)-0\right)^2 + \frac{n_r}{2}log\left(\frac{1}{{\sigma_R}^2}\right)\right). \tag{3.18}$$

We can also weight the three components of this log posterior (3.16) (log likelihood of measurements data, log likelihood of PDE constraint and log priors) using three different parameters $\alpha_{data}, \alpha_{pde}, \alpha_{prior} > 0$, ending up with a modified log posterior:

$$\begin{aligned} L(\boldsymbol{\theta}) = & \alpha_{data} log\left(\mathscr{P}\left(\boldsymbol{D}|\boldsymbol{\theta},\sigma_D\right)\right) + \alpha_{pde} log\left(\mathscr{P}\left(\boldsymbol{R}|\boldsymbol{\theta},\sigma_R\right)\right) + \\ & \alpha_{prior} log\left(\mathscr{P}\left(\boldsymbol{\theta},\sigma_D,\sigma_R\right)\right). \end{aligned} \tag{3.19}$$

The final goal of our methods will be the sampling from this distribution and the evaluation of all the statistics of the outputs.

3.2.1 Hamiltonian Monte Carlo

The first method that we have implemented is a classical MCMC method, the Hamiltonian Monte Carlo (HMC) method [26].

The idea behind HMC is to generate samples following an Hamiltonian dynamics, in contrast to previous and simpler MCMC methods, like Metropolis-Hastings [27], where we generate samples randomly from a proposal distribution. In high dimensional cases, the idea of following this specific dynamics instead of random sampling will make HMC much more efficient than Metropolis-Hastings.

Consider the log posterior distribution defined in (3.19) (we consider as parameters only $\boldsymbol{\theta}$ for simplicity). We define the potential $U(\boldsymbol{\theta})$ as the opposite of log posterior, namely:

$$U(\boldsymbol{\theta}) = -L(\boldsymbol{\theta}). \tag{3.20}$$

HMC method introduce an auxiliary momentum variable $\mathbf{r}$ to mimic the Hamiltonian dynamic, building the following Hamiltonian system:

$$H(\boldsymbol{\theta},\mathbf{r}) = U(\boldsymbol{\theta}) + \frac{1}{2}\mathbf{r}^T\mathbf{M}^{-1}\mathbf{r} \tag{3.21}$$

where **M** is an additional mass matrix that can be set, for simplicity, equal to the identity times a fixed positive constant. We end up with a joint distribution of $\boldsymbol{\theta}$ and $\mathbf{r}$, under the form:

$$\pi(\boldsymbol{\theta},\mathbf{r}) \sim \exp(-H(\boldsymbol{\theta},\mathbf{r})), \tag{3.22}$$

from which we can sample different values of $\boldsymbol{\theta}$ discarding $\mathbf{r}$, that belong to the following dynamical system:

$$\begin{cases} d\boldsymbol{\theta} = \mathbf{M}^{-1}\mathbf{r}dt \\ d\mathbf{r} = -\nabla U(\boldsymbol{\theta})dt. \end{cases} \tag{3.23}$$

We numerically approximate (3.23) using a "leap frog" method [28], leading to the following time-step for each iteration k from 1 to a fixed L (length of trajectory) with a step-size dt (length of every step):

$$\begin{cases} \mathbf{r}_{t_k} = \mathbf{r}_{t_k} - \frac{dt}{2}\nabla U(\boldsymbol{\theta}_{t_k}) \\ \boldsymbol{\theta}_{t_{k+1}} = \boldsymbol{\theta}_{t_k} + dt\mathbf{M}^{-1}\mathbf{r}_{t_k} \\ \mathbf{r}_{t_{k+1}} = \mathbf{r}_{t_k} - \frac{dt}{2}\nabla U(\boldsymbol{\theta}_{t_{k+1}}). \end{cases} \tag{3.24}$$

Considering $\boldsymbol{\theta} = (\boldsymbol{\theta}, \sigma_D, \sigma_R)$ for simplicity, if they are trainable too, and choosing $\mathbf{M} = \mathbf{I}$ as the identity, the HMC algorithm works as follows [7]:

Algorithm 2: Hamiltonian Monte Carlo

```
Initialization: Initialize $\boldsymbol{\theta}^{t_0}$, fix the number of iterations N, the number of samples
  M to collect (after burn-in), the number of leap-frog steps L and the step size $dt$;
for k in 1,...,N do
    Sample $\mathbf{r}^{t_{k-1}} \sim \mathcal{N}(\mathbf{0},\mathbf{I})$;
    Set $(\boldsymbol{\theta}_0,\mathbf{r}_0) = (\boldsymbol{\theta}^{t_{k-1}},\mathbf{r}^{t_{k-1}})$;
    for i in 0,...,(L-1) do
        $\mathbf{r}_i = \mathbf{r}_i - \frac{dt}{2}\nabla U(\boldsymbol{\theta}_i)$ ;
        $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + dt\mathbf{r}_i$ ;
        $\mathbf{r}_{i+1} = \mathbf{r}_i - \frac{dt}{2}\nabla U(\boldsymbol{\theta}_{i+1})$;
    end
    Sample $p \sim Uniform(0,1)$;
    Compute $\alpha = \min(1,\ \exp(H(\boldsymbol{\theta}_L,\mathbf{r}_L) - H(\boldsymbol{\theta}^{t_{k-1}},\mathbf{r}^{t_{k-1}})))$;
    if $p \geq \alpha$ then
        $\boldsymbol{\theta}^{t_k} = \boldsymbol{\theta}_L$;
    else
        $\boldsymbol{\theta}^{t_k} = \boldsymbol{\theta}^{t_{k-1}}$;
    end
end
Collect the last M samples: $\{\boldsymbol{\theta}^{t_i}\}_{i=N-M+1}^{N}$.
```

We remark that, while sampling the parameters $\boldsymbol{\theta}^{t_i}$ with the HMC method, at the same time we are also train our NN to approximate both $u(\mathbf{x})$ and $\boldsymbol{\lambda}(\mathbf{x})$: we can expect that at the first iteration, that is when $\boldsymbol{\theta} = \boldsymbol{\theta}^{t_0}$, our approximations of $u(\mathbf{x})$ and $\boldsymbol{\lambda}(\mathbf{x})$ through the NN of parameters $\boldsymbol{\theta}^{t_0}$, $u^{\boldsymbol{\theta}^{t_0}}(\mathbf{x})$ and $\boldsymbol{\lambda}^{\boldsymbol{\theta}^{t_0}}(\mathbf{x})$, are very far from the true solutions. Advancing in the algorithm, the following samples of $\boldsymbol{\theta}$ would provide some better approximation of both $u(\mathbf{x})$ and $\boldsymbol{\lambda}(\mathbf{x})$ (this is also the reason why we discard the first N-M samples we generate: we expect that these samples do not come from the posterior distribution of $\boldsymbol{\theta}$).
The training stage of the NN is hidden inside algorithm 2. At every iteration k we compute $\nabla U(\boldsymbol{\theta})$, that means:

1. we first compute the prediction of both u and $\boldsymbol{\lambda}$ using the NN with parameters $\boldsymbol{\theta}^{t_{k-1}}$ in all the locations we need, that are:

    - $u^{\boldsymbol{\theta}^{t_{k-1}}}(\mathbf{x}_d)$, $d = 1,\dots,n_d$;
    - $u^{\boldsymbol{\theta}^{t_{k-1}}}(\mathbf{x}_r)$, $r = 1,\dots,n_r$;

- $\boldsymbol{\lambda}^{\boldsymbol{\theta}^{t_{k-1}}}(\mathbf{x}_d)$, $r = 1, \dots, n_r$;

2. we then compute $\boldsymbol{D}^{\boldsymbol{\theta}^{t_{k-1}}}$ and $\boldsymbol{R}^{\boldsymbol{\theta}^{t_{k-1}}}$ using (3.3) and (3.5) (taking advantage of Automatic Differentiation for the computation of $\mathbf{R}^{\boldsymbol{\theta}^{t_{k-1}}}$);
3. we compute $log\left(p\left(\boldsymbol{D}|\boldsymbol{\theta}^{\boldsymbol{\theta}^{t_{k-1}}}, \sigma_D^{\boldsymbol{\theta}^{t_{k-1}}}\right)\right)$ and $log\left(p\left(\boldsymbol{R}|\boldsymbol{\theta}^{\boldsymbol{\theta}^{t_{k-1}}}, \sigma_R^{\boldsymbol{\theta}^{t_{k-1}}}\right)\right)$ using (3.17) and (3.18) in order to collect $U(\boldsymbol{\theta}^{t_{k-1}}) = -L(\boldsymbol{\theta}^{t_{k-1}})$;
4. finally we compute $\nabla U(\boldsymbol{\theta})$ employing the Back Propagation for NN.

Every time we update $\boldsymbol{\theta}$, we are also changing the parameters in the NN used to compute the approximation of u and $\boldsymbol{\lambda}$.

Parameters choice of HMC

One of the big challenges in HMC is the choice of its hyper-parameters [29], namely dt, L and $\mathbf{M}$. These parameters are strictly related to the Hamiltonian dynamics, and inadequate choices might compromise all the useful properties of Hamiltonian and ultimately lead to wrong results.
There is not a simple rule to choose them, but we have to manually fine tune in order to achieve accurate results by maintaining an high acceptance/rejection rate α. Let start from dt and L, that are strictly related one to each other. L is the number of steps we do in a single iteration, while dt is the step size. An intuitive choice would be to impose a relation between them, for instance fixing the total distance K:

$$K = L\,dt. \tag{3.25}$$

Hence, if we take a bigger step size dt, we need a smaller number of steps L, because we want to maintain fixed the distance the Hamiltonian walks. Moreover we do not know a-priori how long our dynamics should be, since K is another parameter we have to choose by fine-tuning. For this reason, the relation (3.25) does not provide any real constraint between dt and L but just a new parametrization. There is also an issue related with efficiency when tuning these parameters: the bigger L, the bigger the required computational time. Walking a lot of steps with a very small step size is computationally more expensive than doing just a few steps with a bigger step size. The ideal choice would be the one that minimizes L, but still with a good accuracy and high acceptance/rejection rate.
We also have to take into account the choice of batch size when selecting L: if we want to implement a mini-batch training, in the algorithm we loop over L steps but also on $n_{batch} = \frac{n_r}{n_{batchsize}}$, so that we perform overall a total of Ln_{batch} steps.

Another important choice is how to set the mass matrix $\mathbf{M}$: setting $\mathbf{M} = \mathbf{I}$ is usually the simplest choice, however it might lead to very poor results and need some more complicated guess. The first modification, still very simple, could be a scaling of the form:

$$\mathbf{M} = \eta^2 \mathbf{I} \tag{3.26}$$

for a parameter $\eta > 0$ large enough. This new mass matrix acts as a regularization for the problem, with η properly selected.

More complicated mass matrix can be proposed, for instance knowing in advance how the posterior distribution would be (in analytical cases). The procedure is explained for instance in [30], where the authors make use of both covariance matrices of prior and likelihood distribution. Imagine to know also the relationship between the input $\mathbf{x}$ and the output $\mathbf{y}$: $\mathbf{y} = \mathbf{G}\mathbf{x}$, and called $\mathbf{C}_p$ the prior covariance and $\mathbf{C}_l$ the likelihood covariance, we can compute the mass matrix $\mathbf{M}$ as:

$$\mathbf{M} = \mathbf{G}^T \mathbf{C}_l^{-1} \mathbf{G} + \mathbf{C}_p^{-1}. \tag{3.27}$$

However, there is no simple and universal way to set all these parameters, so that most of them need to be tuned with a trial and error process, like in other machine learning algorithms.

Some modifications to the classical HMC algorithm make these choices easier. One of them is the famous NUTS (No U-Turn Sampler) method [31], that dinamically selects the best step length L. The idea behind it is to avoid the Hamiltonian to turn back to states already explored, so that it stops the random walk when we are starting to go in the opposite direction. NUTS thus only require to set *dt* and $\mathbf{M}$.

3.2.2 Stein Variational Gradient Descent

Since the sampling from the posterior with standard MCMC methods can become sometimes infeasible, we also employ the Stein Variational Gradient Descent algorithm (SVGD) [32] with a finite number of neural networks (called “particles"), for example N=30, that will approximate the posterior distribution (of course the larger N, the better the approximation, however, implying a growing computational time).

The idea behind SVGD algorithm is to iteratively transport a finite number of particles toward the target distribution, with the goal of minime the Kullback-Leibler divergence [33] between the particles and the distribution itself. The SVGD algorithm works as follows, as shown in [34]:

Algorithm 3: Stein Variational Gradient Descent

```
Initialization: Initialize N neural networks weights $\boldsymbol{\theta}^i$, $\sigma_D^i$ and $\sigma_R^i$, $i = 1,\dots,N$, set
  the number of epochs E and the step size $\varepsilon$;
for epoch e=1,...,E do
    Compute the log posterior $L(\boldsymbol{\theta}^i)$ $\forall i = 1\dots,N$ ;
    Compute the gradients $\nabla_{\boldsymbol{\theta}^i} L(\boldsymbol{\theta}^i)$ by back-propagation ;
    Compute
        $$\phi(\boldsymbol{\theta}^i) = \frac{1}{N}\sum_{j=1}^{N}\left[k(\boldsymbol{\theta}^i,\boldsymbol{\theta}^j)\nabla_{\boldsymbol{\theta}^i} L(\boldsymbol{\theta}^i) + \nabla_{\boldsymbol{\theta}^i} k(\boldsymbol{\theta}^i,\boldsymbol{\theta}^j)\right] \quad \forall i = 1\dots,N; \qquad (3.28)$$
    Update $\boldsymbol{\theta}^i = \boldsymbol{\theta}^i + \varepsilon\phi(\boldsymbol{\theta}^i)$ or use a Stochastic Gradient Descent method,
      $\forall i = 1,\dots,N$;
    Update $\sigma_D^i$ and $\sigma_R^i$ (simply through back-propagation or use the same SVGD
      passages as for $\boldsymbol{\theta}$), $\forall i = 1,\dots,N$;
end
```

where $k(x_i,x_j)$ is the Radial Basis Function (RBF) kernel [35], namely:

$$k(x_i,x_j) = exp(-\frac{1}{h}\|x_i - x_j\|^2)$$

for a constant bandwidth $h > 0$.
The key passage in this method is provided by equation (3.28), where we modify the gradients of $\boldsymbol{\theta}^i$, $i = 1,\dots,N$ to approximate the posterior distribution. In particular, the first term of the sum drives the particles towards the high probability areas of $L(\boldsymbol{theta})$ by following a smoothed gradient direction, which is the weighted sum of the gradients of all the particles weighted by the kernel function $k(x_i,x_j)$; the second terms (gradients of k) acts as a repulsive force that prevents all particles to collapse to a single point, namely the MAP of the posterior distribution.
As we have said before, the critical choice in SVGD is the number of particles we are going to use. A very small number N of particles will result in a fast computation but yielding worse results in terms of posterior approximation. On the contrary, a larger values of N will result in a bigger ability to approximate the posterior distribution, however implying a bigger cost in terms of computational time (in this case we need also a RAM with enough memory to store a high number of neural networks at the same time).
As explained before for the HMC method, also with SVGD we are training the NN simultaneously to the algorithm. Every time we update the parameters $\boldsymbol{\theta}^i, \quad i = 1,\dots,N$, we are

updating also the N neural networks we use to approximate the solutions.

3.3 Active Learning

When dealing with a small dataset, it is important that data are sampled at significant locations to train a good model. In a situation where acquiring new data has a big cost, knowing where to find a new observations a priori plays a key role. Active Learning methods are algorithms that guide the data selection process. In Bayesian Neural Network this can be done by adding an observation where the posterior predictive distribution over our outputs has a bigger variability. In our case, when solving an Inverse UQ problem for a PDE, we will add a new measurement $u(\mathbf{x}_{\mathbf{new}})$ of the PDE solution at a position $\mathbf{x}_{\mathbf{new}}$ where the posterior predictive standard deviation is bigger. The standard deviation of the posterior distribution of the PDE solution u evaluated at the n_r collocation points, using M samples of $\boldsymbol{\theta}$ trained with B-PINN, $\{\boldsymbol{\theta}_m\}_{m=1}^{M}$, is:

$$std(u^{\boldsymbol{\theta}}(\mathbf{x}_r)) = \sqrt{\frac{1}{M}\sum_{m=1}^{M} \|u^{\boldsymbol{\theta}_m}(\mathbf{x}_r) - \mathbb{E}\left[u^{\boldsymbol{\theta}}(\mathbf{x}_r)\right]\|_2^2} \quad \forall r = 1,\dots,n_r, \tag{3.29}$$

where $\mathbb{E}\left[u^{\boldsymbol{\theta}}(\mathbf{x}_r)\right] = \frac{1}{M}\sum_{m=1}^{M}(u^{\boldsymbol{\theta}_m}(\mathbf{x}_r))$ at every collocation points.

The idea of Active Learning in this setting is the following: starting from a small set of PDE solution measurements $[u]_0^{ex} = \{u_d\}_{d=1}^{n_d}$, add K new measurements at those locations of maximum uncertainty. We end up with this active sampling technique:

Algorithm 4: Active sampler B-PINN

```
Data: Measurements: $[u]_0^{ex} = \{u_d\}_{d=1}^{n_d}$
Collocation points: $\{\boldsymbol{x}_r\},\ r = 1,\dots,n_r$.
Algorithm: Active Sampling for Inverse UQ problems with B-PINN;
Train a B-PINN with measurements $[u]_0^{ex}$;
for $k = 1,\dots,K$ do
    compute $std(u^{\boldsymbol{\theta}}(\mathbf{x}_r))$ for every $\boldsymbol{x}_r,\ r = 1,\dots,n_r$, using (3.29);
    $[u]_k^{ex} = [u]_{k-1}^{ex} \cup \{u_{r^*},\ r^* = \arg\max_{r=1,\dots,n_r} std(u^{\boldsymbol{\theta}}(\mathbf{x}_r))\}$;
    train a B-PINN with measurements $[u]_k^{ex}$;
```

3.4 Transfer Learning in subdomains

There are some situations where measurements are all concentrated in a small portion of the domain. This is the case for instance when we use a grid of sensors, like in the case where observations mimic data acquires through a catheter of a Cardiac Activation mapping system (see Chapter 5).
In these cases, we can expect a good accuracy of our solutions in those portion of the domain where we have a lot of measurements, and worse results in all the other portions of Ω. Some new grids of measurements are required also in the other areas to improve results in all the domain.
After collecting this new set of measurements, we can avoid to train from scratch our B-PINN, relying on Transfer Learning approaches. In the following sections we explain our methods that apply Transfer Learning concepts in subdomains of Ω.

3.4.1 Transfer Learning

Transfer Learning (TL) is a Machine Learning method that uses previous knowledge to solve a problem, trying to get better results than starting the training from scratch [1],[36]. In our case, we can use the knowledge gained with a previous set of measurements. To be effective, we need a proper way to share the knowledge: in Neural Networks this is usually done by sharing weights parameters. There are two different ways to use a Pre-Trained Network [37] in Transfer Learning:

- using the previous weights as initial guess for our Neural Network and train for a few epochs;
- fix the first layers with the previous weights and train only the last few layers.

In this work we use the first approach. Some new approaches specifically set for Bayesian Networks are presented in [38], however they have not been considered here.
There are three types of performance improvement we can expect applying a Transfer Learning approach, as shown in Figure 3.1:

- higher start;
- higher slope;
- higher asymptote.

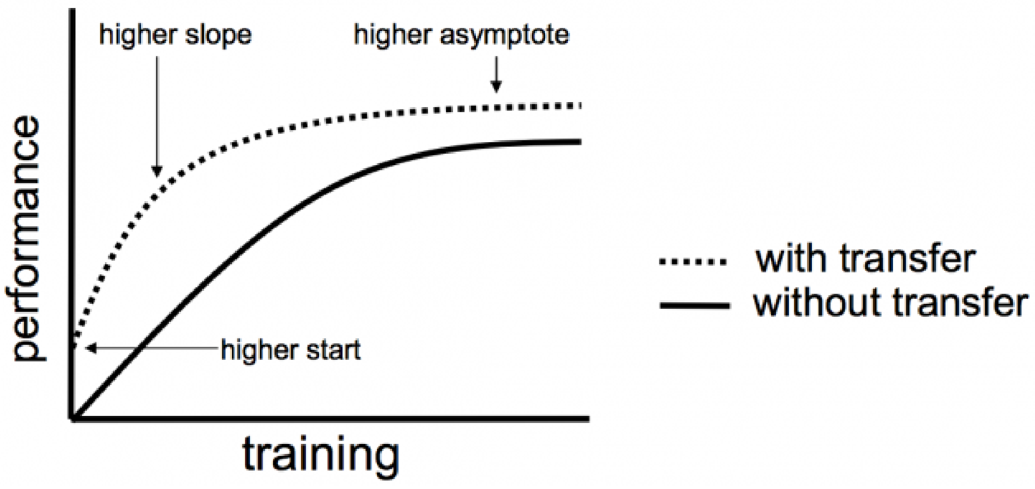


Figure 3.1: Transfer Learning: three types of performance improvement [1]

We could expect to find at least one of them, but sometimes all the three improvements are found at the same time. For the purpose of our algorithm, presented in the next section, we look for higher start and a higher slope in the first epochs: this will make possible to repeat the same faster training on some small portions of the domain and still get accurate results with just few epochs.

3.4.2 Domain decomposition and TL

Assume that the domain Ω is split into K subdomains: $\Omega_i, \quad i = 1, \dots, K$. For each subdomain we have a dataset $\mathbf{R}_i$ of collocation points and $\mathbf{D}_i$ of noisy measurements. For instance, in the cardiac application we are going to describe later on, we can imagine that every $\mathbf{D}_i$ is a grid of measurements of a multi electrode catheter at a single location. The idea behind this domain decomposition strategy is simple: after a full training on the first subdomain Ω_1, where we train our B-PINN with its datasets $\mathbf{D}_1$ and $\mathbf{R}_1$, we can apply the Transfer Learning approach to all the other subdomains. Figure 3.2 shows the process on a 2D domain:

- first, we train our B-PINN with N epochs on Ω_1;
- then, we reuse the final parameters vector $\boldsymbol{\theta}_{\Omega_1}$ as initial guess (instead of a random guess like Glorot or He initializer [39]) for every Ω_i, $i = 2, \dots, K$, using just a fraction of the total epochs N, such as $\frac{N}{10}$.

Getting a good and accurate result with just $\frac{N}{10}$ epochs will be very challenging for this task, but the performance boost we get from Transfer Learning will help the convergence. If this process is accurate enough, we can save a lot of computational time and resources, since the most of time is spent on Ω_1, while in the other subdomains we spend just 10% of it.

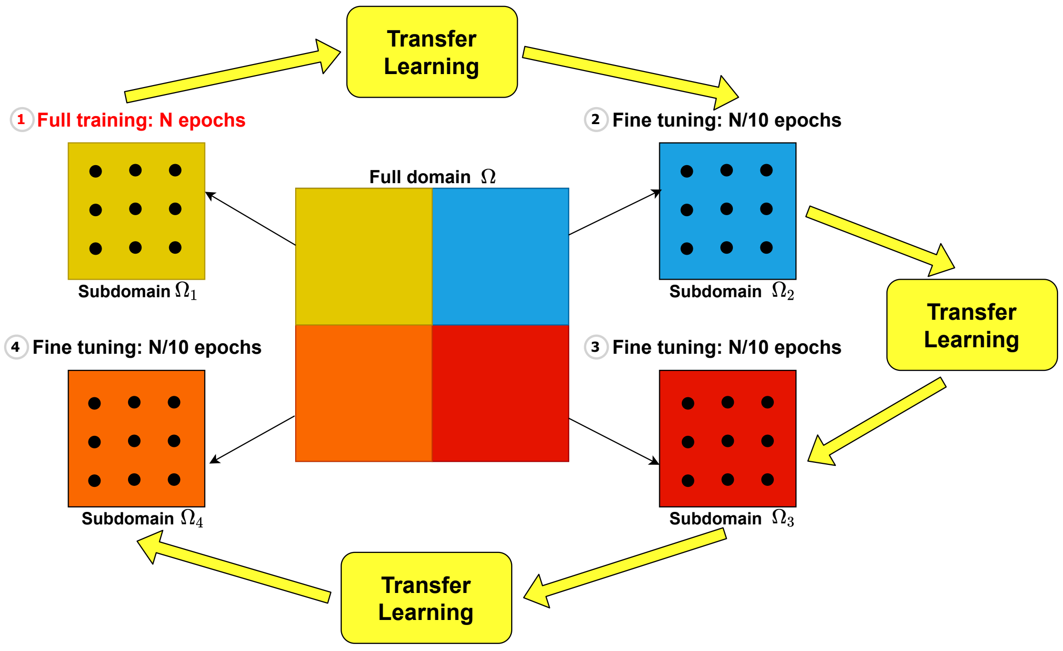


Figure 3.2: Transfer Learning in subdomains algorithm on a 2D domain example

Chapter 4

Numerical Validation

We present here some initial numerical results to validate Bayesian PINNs for the solution of Inverse UQ problems. In this chapter we focus on a linear Elliptic problem, since its linearity enables the calculation of solution of the Inverse UQ problem in a closed form. In a first experiment we reconstruct a single scalar parameter: in this case, using Normal distributions, we are able to compute the real posterior distribution by hand and then assess our method. In a second experiment, we reconstruct all the right-hand side $f(x)$ of a monodimensional Poisson problem and check the posterior distribution on both the PDE solution $u(x)$ and the datum $f(x)$.

4.1 Parametric 1D Poisson problem, single parameter

We firstly consider an Inverse UQ problem involving a single scalar parameter of a PDE. The parameter $v \in \mathbb{R}$ is on the right hand side of a one-dimensional Poisson problem, of the form:

$$\begin{cases} -\dfrac{\partial^2 u(x)}{\partial x^2} &= v f(x), \quad x \in \Omega = (0,8), \\ u(0) &= v, \\ u(8) &= v cos(8). \end{cases} \tag{4.1}$$

Given $f(x) = cos(x)$, the parametric solution of the previous equation is $u(x) = vcos(x)$, with $v \in \mathbb{R}$. In this Inverse UQ problem we suppose to know n_d noisy measurements of the solution $\{u_d\}_{d=1}^{n_d}$ in the locations $\{\mathbf{x}_d\}_{d=1}^{n_d}$ and we want to compute the posterior PDF of the parameter v.

Recasting this problem in the Bayesian framework, we suppose to have a dataset $\boldsymbol{D} = \{x_d, u_d\}_{d=1}^{n_d}$ of measurements, perturbed by a gaussian noise:

$$u_d = v\cos(x_d) + \varepsilon_d, \quad \varepsilon_d \sim \mathcal{N}(0, \sigma^2), \quad d = 1, \ldots, n_d, \tag{4.2}$$

for a fixed σ^2. In addition to $\boldsymbol{D}$ we also have the dataset of collocation points $\boldsymbol{R} = \{x_r, 0\}_{r=1}^{n_r}$ where we enforce the validity of the PDE (4.1).
The choices for Likelihood functions for $\boldsymbol{D}$ and $\boldsymbol{R}$ and prior distributions for the neural network weights $\boldsymbol{\theta}$ are the ones described in the previous chapter:

- Likelihood for $\boldsymbol{D}$: $\mathscr{P}(\boldsymbol{D}|\boldsymbol{\theta}) \sim \mathscr{N}(\boldsymbol{D}^{\boldsymbol{\theta}}, \sigma_D^2 \mathbb{I})$,
- Likelihood for $\boldsymbol{R}$: $\mathscr{P}(\boldsymbol{R}|\boldsymbol{\theta}) \sim \mathscr{N}(\boldsymbol{R}^{\boldsymbol{\theta}}, \sigma_R^2 \mathbb{I})$,
- Prior for $\boldsymbol{\theta}$: $\mathscr{P}(\boldsymbol{\theta}) \sim \mathscr{N}(\boldsymbol{0}, \gamma^2 \mathbb{I})$,

where $\sigma_D = \sigma$ is the same as in (4.2), σ_R represent our uncertainty on the PDE equation (in this experiment we have set $\sigma_R = \sigma_D$) and γ large enough (for instance $\gamma = 1$).
The only difference here is that our parametric field $\lambda(x)$ now is a scalar parameter $\lambda(x) = v > 0$; for this reason it is added to the hyperparameters.
We have to define also a prior distribution for v. We also use for the prior of v a Normal distribution, in order to make the computation of the actual posterior easy. Give a mean v_0 and variance σ_0^2, the prior distribution for v is:

$$\mathscr{P}(v) \sim \mathscr{N}(v_0, \sigma_0^2), \tag{4.3}$$

where we select in particular $v_0 = 0$ and $\sigma_0 = 1$. We conduct an experiment varying the number of measurements of the solution (10 and 100) and the noise level ($\sigma = 0.01, 0.02, 0.05$). The analysis of the posterior distributions of v in all four cases, approximated by our method using B-PINN (both with HMC and SVGD), are compared against the true posterior distribution in a classical Inverse UQ problem, computed in the next section.

4.1.1 True posterior of v

In this case, using Normal distributions for both likelihood and prior, and thanks to the linearity of the input-to-observation map, we can compute in a closed form the true posterior distribution. Instead of having another datset $\boldsymbol{R}$ where enforcing the validity of PDE (4.2), suppose to know the solution map $\mathscr{F} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, defined as:

$$u(x; v) = \mathscr{F}(x, v) = v cos(x). \tag{4.4}$$

We then define the likelihood for the dataset $\boldsymbol{D}$ conditioned to v as:

$$\mathscr{P}(\boldsymbol{D}|v) \propto \prod_{d=1}^{n_d} \left(\frac{1}{\sqrt{2\pi\sigma_D^2}} \exp\left(-\frac{(u_d - v cos(x_d))^2}{2\sigma_D^2} \right) \right). \tag{4.5}$$

We need to define a prior distribution for the parameter v, so that applying the Bayes Rule, we can finally compute the posterior distribution as:

$$\mathscr{P}(v|\boldsymbol{D}) \propto \mathscr{P}(\boldsymbol{D}|v)\mathscr{P}(v), \tag{4.6}$$

that in this case is equal to:

$$\mathscr{P}(v|\boldsymbol{D}) \propto \left[\left(\frac{1}{\sqrt{2\pi\sigma_D^2}}\right)^{n_d} \prod_{d=1}^{n_d} \exp\left(-\frac{(u_d - vcos(x_d))^2}{2\sigma_D^2}\right)\right] \frac{1}{\sqrt{2\pi\sigma_0^2}} \exp\left(-\frac{(v_0 - v)^2}{2\sigma_0^2}\right). \tag{4.7}$$

Since the posterior is a product of Normal distributions, it is Normal too:

$$\mathscr{P}(v|\boldsymbol{D}) \sim \mathscr{N}(v_{post}, \sigma_{post}^2), \tag{4.8}$$

with mean v_{post} and variance σ_{post}^2, defined by the following equations:

$$\begin{aligned} v_{post} &= \frac{\sum_{d=1}^{n_d}(u_d \cos(x_d))\sigma_0^2 + v_0\sigma_D^2}{\sum_{d=1}^{n_d}(\cos^2(x_d))\sigma_0^2 + \sigma_D^2} \\ \sigma_{post}^2 &= \frac{\sigma_0^2\sigma_D^2}{\sum_{d=1}^{n_d}(\cos^2(x_d))\sigma_0^2 + \sigma_D^2}. \end{aligned} \tag{4.9}$$

Appendix B shows the details for the computation of the formulas above.

4.1.2 Results

We run six different experiments, with different number of measurements (10 or 100) and different noise level on them (0.01, 0.02 and 0.05). True posterior means and variances are computed with the two formulas (4.9) defined above. Both HMC and SVGD have been tested: we used for both methods a NN architecture of 2 Hidden layers with 50 neurons each. As activation function in the NN we have used for all the layers a Swish activation function (we use this particular activation function in all the experiments of this thesis). HMC runs for N=3000 iterations, with a length of trajectory L=100. We collect the last M=2500 samples. It takes approximately 2 sec for iteration, resulting in approximately 6000 sec in total. On the other hand, for SVGD we use n=30 neural networks (“particles") and 10000 epochs, that takes 3000 sec in total.

In Figure 4.1 we plot the true posterior distribution of v in the case with 100 data, noise level = 0.02, against the posterior distribution predicted with the HMC method and the SVGD (where we can see the n=30 particles positions drawn in blue in the lower part of the figure). Numerical results are shown in table 4.1, where we show the mean and standard deviation for all the cases. Both the methods approximate the mean of the true posterior distribution with

n. data	noise	True (μ,σ)	HMC (μ,σ)	SVGD (μ,σ)
10	0.01	(1.0008, 4.186e-3)	(0.9999, 2.819e-3)	(0.9974, 9.685e-3)
10	0.02	(1.0015, 8.371e-3)	(1.0092, 1.083e-2)	(0.9933, 1.966e-2)
10	0.05	(1.0036, 2.092e-2)	(1.0147, 8.472e-3)	(0.9735, 4.908e-2)
100	0.01	(0.9995, 1.349e-3)	(0.9997, 1.229e-3)	(0.9998, 2.362e-3)
100	0.02	(0.9990, 2.699e-3)	(1.0011, 3.057e-3)	(0.9998, 4.654e-3)
100	0.05	(0.9976, 6.748e-3)	(1.0014, 3.443e-3)	(0.9994, 1.170e-2)

Table 4.1: 1D Poisson Parametric experiment, mean and standard deviation, n.data 10,100 and noise 0.01,0.02,0.05. Comparison between HMC and SVGD against true result

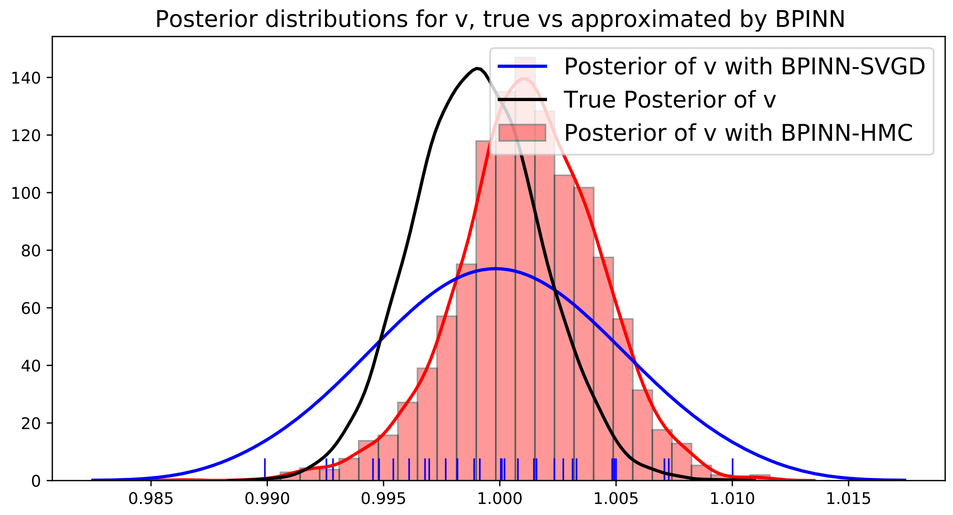


Figure 4.1: Posterior distributions for v with 100 data, 0.02 noise, 1D Poisson parametric experiment. Comparison between HMC and SVGD against true result

a maximum error of 0.03, while only HMC is able to approximate well also the posterior standard deviation.

SVGD tends to overestimate the variance: out of 30 particles, some of them are very far from the real distribution. For this reason, the distribution constructed with these samples has always a bigger variance respect to the true one. This problem could be fixed using additional particles, but this will increase the computational cost and times, making useless this methods that approximate a distribution with just few of them.
To compare the distribution, we can use the Kullback-Leibler divergence [33], which is a measure of difference between two distributions. Given two distributions $q : \mathscr{X} \to \mathbb{R}^+$, $p : \mathscr{X} \to \mathbb{R}^+$ on the same Probability space, the Kullback-Leibler divergence of p given q

is:

$$D_{KL}(p||q) = \int_{x\in\mathscr{X}} p(x)\log_2\left(\frac{p(x)}{q(x)}\right)dx. \tag{4.10}$$

A lower KL divergence means a lower distance between the two distributions: this type of measure can be seen as a measure of information loss between the proposed (p) and the target (q) distribution. The results shown in table 4.2 confirm our beliefs: HMC provides always a lower KL divergence compared to SVGD.

n. data	noise	D_{KL} HMC	D_{KL} SVGD
10	0.01	0.1452	1.6675
10	0.02	0.5024	1.8838
10	0.05	0.6259	2.4378
100	0.01	0.0191	0.4974
100	0.02	0.3195	0.4857
100	0.05	0.4628	0.4899

Table 4.2: 1D Poisson Parametric experiment, KL Divergence results

4.2 Parametric 1D Poisson problem, parametric field

In this second example, we consider the whole right hand side as our parametric field $\lambda(x) = f(x)$:

$$\begin{cases} -\dfrac{\partial^2 u(x)}{\partial x^2} =f(x), \quad x\in\Omega=(0,8) \\ u(0) =1, \\ u(8) =cos(8). \end{cases} \tag{4.11}$$

Also in this case we use the solution $u(x) = cos(x)$, when $f(x) = cos(x)$. To compute the posterior distribution of $f(x)$ we generate the samples $\{\boldsymbol{\theta}_m\}_{m=1}^M$ from the posterior distribution of the network parameters $\boldsymbol{\theta}$. We collect M samples from the posterior of $f(x)$, indicated with $\{f^{\boldsymbol{\theta}_m}(x)\}_{m=1}^M$ for every x in our domain, where each $f^{\boldsymbol{\theta}_m}(x)$ is simply the prediction of our network in x using the parameters $\boldsymbol{\theta}_m$.

4.2.1 Results

To measure the accuracy of our method in the reconstruction of the parametric field $f(x)$, and also of the whole solution $u(x)$, we compare the sample means of the M predictions (using the M sampled parameters $\{\boldsymbol{\theta}_m\}_{m=1}^M$) and the analytical solutions, $u^{true}(x)$ and $f^{true}(x)$. We define the sample means as:

$$\begin{aligned}\mathbb{E}\left[u^{\boldsymbol{\theta}}(x)\right] =& \frac{1}{M}\sum_{m=1}^{M}(u^{\boldsymbol{\theta}_m}(x)),\\ \mathbb{E}\left[f^{\boldsymbol{\theta}}(x)\right] =& \frac{1}{M}\sum_{m=1}^{M}(f^{\boldsymbol{\theta}_m}(x)).\end{aligned} \tag{4.12}$$

We use an euclidean relative error to assess the accuracy of our prediction in reconstruct the analytical solution, defined as:

$$\begin{aligned}l^2_{rel\ err}(u^{\boldsymbol{\theta}}) =& \frac{||\mathbb{E}\left[u^{\boldsymbol{\theta}}\right] - u^{true}||_2^2}{||u^{true}||_2^2} = \frac{\sum_{i=1}^{n_{dom}}(\mathbb{E}\left[u^{\boldsymbol{\theta}}(x_i)\right] - u^{true}(x_i))^2}{\sum_{i=1}^{n_{dom}}(u^{true}(x_i))^2}\\ l^2_{rel\ err}(f^{\boldsymbol{\theta}}) =& \frac{||\mathbb{E}\left[f^{\boldsymbol{\theta}}\right] - f^{true}||_2^2}{||f^{true}||_2^2} = \frac{\sum_{i=1}^{n_{dom}}(\mathbb{E}\left[f^{\boldsymbol{\theta}}(x_i)\right] - f^{true}(x_i))^2}{\sum_{i=1}^{n_{dom}}(f^{true}(x_i))^2}\end{aligned} \tag{4.13}$$

using all the points in our domain (dimension of n_{dom}), while for the sake of UQ we first compute (numerically) the standard deviation on every input x_i, $i = 1,\dots,n_{dom}$ of our predictions:

$$\begin{aligned}std(u^{\boldsymbol{\theta}}(x_i)) =& \sqrt{\frac{1}{M}\sum_{m=1}^{M}(u^{\boldsymbol{\theta}_m}(x_i) - \mathbb{E}\left[u^{\boldsymbol{\theta}}(x_i)\right])^2} \quad \forall i = 1,\dots,n_{dom},\\ std(f^{\boldsymbol{\theta}}(x_i)) =& \sqrt{\frac{1}{M}\sum_{m=1}^{M}(f^{\boldsymbol{\theta}_m}(x_i) - \mathbb{E}\left[f^{\boldsymbol{\theta}}(x_i)\right])^2} \quad \forall i = 1,\dots,n_{dom}.\end{aligned} \tag{4.14}$$

We then compute the standard deviation of the prediction of both u and f, averaged all over the domain and the maximum standard deviation in all the domain:

$$\begin{aligned}ST^{mean}(u^{\boldsymbol{\theta}}) =& \frac{1}{n_{dom}}\sum_{i=1}^{n_{dom}} std(u^{\boldsymbol{\theta}}(x_i))\\ ST^{max}(u^{\boldsymbol{\theta}}) =& \max_{i=1,\dots,n_{dom}} std(u^{\boldsymbol{\theta}}(x_i))\\ ST^{mean}(f^{\boldsymbol{\theta}}) =& \frac{1}{n_{dom}}\sum_{i=1}^{n_{dom}} std(f^{\boldsymbol{\theta}}(x_i))\\ ST^{max}(f^{\boldsymbol{\theta}}) =& \max_{i=1,\dots,n_{dom}} std(f^{\boldsymbol{\theta}}(x_i)).\end{aligned} \tag{4.15}$$

The results, using both HMC and SVGD, are shown in Tables 4.3 and 4.4. The tests are performed varying the number of exact data (10 and 100) and the noise level on them (0.01, 0.05 and 0.10). Also in this case SVGD can reach a good approximation in a smaller amount of time with respect to HMC, namely 400 sec vs 2000 sec in this setting.

As it can be seen in the tables, the relative error on f is always quite large: looking at the figures 4.2 and 4.3 (where we plot the analytical solutions for both u and f against the samples

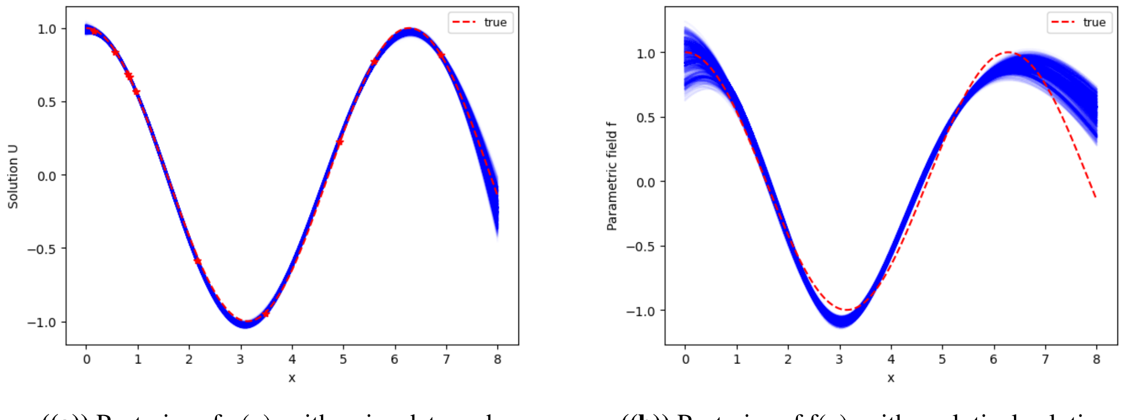
n. data	noise	l^2 rel err (u,f)	ST mean (u,f)	ST max (u,f)
10	0.01	(0.035,0.233)	(0.013,0.037)	(0.085,0.135)
10	0.05	(0.152,0.387)	(0.049,0.117)	(0.158,0.583)
10	0.10	(0.409,0.564)	(0.107,0.211)	(0.325,0.988)
100	0.01	(0.020,0.201)	(0.008,0.026)	(0.022,0.112)
100	0.05	(0.031,0.240)	(0.013,0.045)	(0.035,0.294)
100	0.10	(0.053,0.232)	(0.026,0.065)	(0.067,0.349)

Table 4.3: 1D Poisson UQ analysis with HMC, n.data 10,100 and noise 0.01,0.05,0.10

n. data	noise	l^2 rel err (u,f)	ST mean (u,f)	ST max (u,f)
10	0.01	(0.167,0.275)	(0.026,0.076)	(0.173,0.303)
10	0.05	(0.197,0.394)	(0.022,0.064)	(0.133,0.284)
10	0.10	(0.236,0.478)	(0.025,0.066)	(0.113,0.238)
100	0.01	(0.019,0.166)	(0.012,0.055)	(0.066,0.246)
100	0.05	(0.032,0.180)	(0.012,0.056)	(0.070,0.231)
100	0.10	(0.053,0.192)	(0.012,0.055)	(0.073,0.222)

Table 4.4: 1D Poisson UQ analysis with SVGD, n.data 10,100 and noise 0.01,0.05,0.10

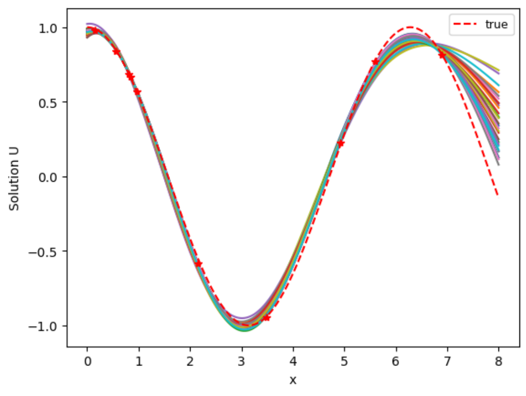
we generate with HMC and SVGD), we can highlight that most of the distance between the true field and our samples is on the right part of the domain, close to the boundary value $x = 8$.



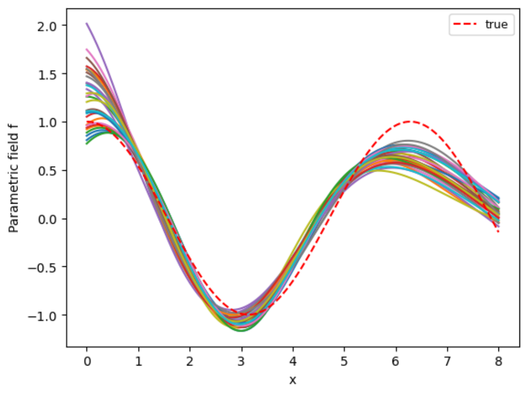
((a)) Posterior of u(x), with noisy data and analytical solution

((b)) Posterior of f(x), with analytical solution

Figure 4.2: 1D Poisson experiment results with HMC, 10 exact data, noise level = 0.01



((a)) Posterior of u(x), with noisy data and analytical solution



((b)) Posterior of f(x), with analytical solution

Figure 4.3: 1D Poisson experiment results with SVGD, 10 exact data, noise level = 0.01

Even if SVGD provides some better results in terms of relative error (5.11) for the parametric field $f(x)$, we can see that the averaged and the maximum standard deviations only depend on the number of data. This is not true in HMC, where the standard deviation increases with the noise and decrease with more data, as expected. Looking at the two figures 4.2 and 4.3, where we have performed the same experiment with the same noisy data, we notice that SVGD tends to explain mostly what in a BNN is called the Epistemic part of the Uncertainty (see section 2.2 for more details). This means that our particles (30 NNs in this setting) are all very close to the data points we have provided, while they spread out when they are far from any exact measurement. This is true also in HMC, but the latter offers, in addition to SVGD, the ability to explain better also the noise on the data. In table 4.3 we can see that both ST^{mean} and ST^{max} (for both u and f) increase with the noise level.
We can state that HMC provides better results than SVGD also in this second experiment, however requiring at least 5 times the computational time of SVGD to produce such results.

Chapter 5

Application to cardiac electrophysiology

Heart contraction, responsible of blood dynamics, are triggered by an electrical signal, that "activate" the cardiac muscle; this electrical activity is produced by the cardiac cells and induced by different concentration of ionic species.
The electrical activity in heart has been extensively studied in the literature in the last decades, and different models of increasing complexity were proposed: the Eikonal equation, the Monodomain model and the Bidomain model [16],[15], [14].
In this thesis we focus on the former model, the Eikonal equation, considering different options. Then, we present some numerical tests related with the Inverse UQ problems using BPINN, as well as some example of Active Sampling and Transfer Learning in subdomains applied to the Eikonal model.

5.1 The Eikonal model

The Eikonal Equation is a simplified model for cardiac electrophysiology, which describes the activation times, that is the times at which a depolarization front reaches a given point in the heart tissue; in this case, the propagation of the electrical signal is modeled as a wave in an anisotropic media. In the next subsection we are going to describe different Eikonal models we can define.

5.1.1 The Anisotropic Eikonal equation

Given $\Omega \in \mathbb{R}^d$, $d = 1,2,3$, a compact domain, let us denote by $T : \mathbb{R}^d \to \mathbb{R}$ the activation time and $\mathbf{M} : \mathbb{R}^d \to \mathbb{R}^{d\times d}$ conductivity tensor (symmetric and positive-definite). The Eikonal equation reads as follows:
given the conductivity tensor $\mathbf{M}(\mathbf{x})$ and the localization of the sources $\{\hat{\mathbf{x}}_i\}_{i=1}^{N_S}$, find $T(\mathbf{x})$

such that

$$\begin{cases} \sqrt{\nabla T(\mathbf{x})^T \mathbf{M}(\mathbf{x}) \nabla T(\mathbf{x})} = 1 & in\ \Omega, \\ T(\hat{\mathbf{x}}_i) = 0 \quad \forall i = 1, \dots, N_S. \end{cases} \tag{5.1}$$

In the forward problem above, given the conductivity tensor $\mathbf{M}(\mathbf{x})$ and the localization of the sources, we have to find the activation times map $T(\mathbf{x})$. In the Inverse problem (which is indeed our final goal) we are given the activation times map $T(\mathbf{x})$ at some locations $\mathbf{x}_j$, $j = 1, \dots, N$ (collected by sensors and potentially affected by noise) and the aim is to reconstruct all the entries of the conductivity tensor $\mathbf{M}(\mathbf{x})$.
Accordingly to the form of the conductivity tensor (in dimension $d > 1$), we can distinguish two different forms of the Eikonal equation, the Anisotropic and the Isotropic one. The conductivity tensor $\mathbf{M}(\mathbf{x})$, that has to be symmetric and positive-definite, represents the anisotropy in conduction velocity in all the domain. We can define it using the cartesian coordinates (x, y, z) or using local basis direction, namely, the fiber direction, tangent direction and normal direction (only in the case of three dimensional domains). Going from one description to another is easy since it only requires a change of coordinates. In the first case $\mathbf{M}(\mathbf{x})$ can be written as (for instance in dimension $d = 3$):

$$\mathbf{M}(\mathbf{x}) = \begin{bmatrix} a(x,y,z) & -d(x,y,z) & -e(x,y,z) \\ -d(x,y,z) & b(x,y,z) & -f(x,y,z) \\ -e(x,y,z) & -f(x,y,z) & c(x,y,z) \end{bmatrix} \tag{5.2}$$

for some suitable functions a, b, c, d, e, f, while in the latter, using the local vector basis $\mathbf{a}_f(\mathbf{x})$, $\mathbf{a}_s(\mathbf{x})$ and $\mathbf{a}_n(\mathbf{x})$, denoting the basis of fiber direction, sheet direction and normal direction at each point $\mathbf{x} \in \Omega$, $\mathbf{M}(\mathbf{x})$ is defined as:

$$\mathbf{M}(\mathbf{x}) = V_f^2 \mathbf{a}_f \otimes \mathbf{a}_f + V_s^2 \mathbf{a}_s \otimes \mathbf{a}_s + V_n^2 \mathbf{a}_n \otimes \mathbf{a}_n \tag{5.3}$$

5.1.2 The Isotropic Eikonal equation

In the particular case where $\mathbf{M}(\mathbf{x}) = v(\mathbf{x})^2 \mathbb{I}$, for a suitable function $v : \mathbb{R}^d \to \mathbb{R}$, with $d \geq 2$, equation (5.1) becomes:

$$\begin{cases} \|\nabla T(\mathbf{x})\| = \dfrac{1}{v(\mathbf{x})} & in\ \Omega, \\ T(\hat{\mathbf{x}}_i) = 0 & \forall i = 1, \dots, N_S, \end{cases} \tag{5.4}$$

and is called Isotropic Eikonal equation, since the conduction velocity is the same along all the directions.

5.1.3 The Eikonal-Diffusion equation

A first different version of the previous equation is the Eikonal-Diffusion equation [40]. Compared to the classical Eikonal equation, this model features an additional diffusive term, that makes the propagation speed influenced by the tissue surrounding the wavefront. The Eikonal-Diffusion equation can be formulated as:

$$\begin{cases} \sqrt{\nabla T(\mathbf{x})^T \mathbf{M}(\mathbf{x}) \nabla T(\mathbf{x})} - \dfrac{1}{c_0} \nabla \cdot (\mathbf{M}(\mathbf{x}) \nabla T(\mathbf{x})) = 1 & in \ \Omega \\ T(\widehat{\mathbf{x}}_i) = 0 \quad \forall i = 1, \ldots, N_S \end{cases} \tag{5.5}$$

where we add the planar velocity c_0 of the wave in the fiber direction to keep the equation balanced from a measurement system point of view.

5.1.4 The Factored Eikonal equation

By defining the slowness as $S(\mathbf{x}) = \frac{1}{v(\mathbf{x})}$, the Isotropic Eikonal Equation (5.4) can be rewritten as:

$$\|\nabla T(\mathbf{x})\| = S(\mathbf{x}) \ \ in\, \Omega. \tag{5.6}$$

Factorizing both $T(\mathbf{x})$ and $S(\mathbf{x})$ as:

$$\begin{aligned} S(\mathbf{x}) &= S_0(\mathbf{x}) \alpha(\mathbf{x}), \\ T(\mathbf{x}) &= T_0(\mathbf{x}) \tau(\mathbf{x}), \end{aligned}$$

such that:

$$\|\nabla T_0(\mathbf{x})\| = S_0(\mathbf{x}) \ \ in\, \Omega,$$

we recast the problem of finding $T(\mathbf{x})$ fulfilling (5.6) into the following Factored Eikonal equation [41], that is, to find $\tau(\mathbf{x})$ such that:

$$\begin{cases} T_0^2(\mathbf{x}) \|\nabla \tau(\mathbf{x})\|^2 + 2T_0(\mathbf{x}) \tau(\mathbf{x}) \nabla T_0(\mathbf{x}) \cdot \nabla \tau \mathbf{x} + \left[\tau^2(\mathbf{x} - \alpha^2(\mathbf{x}))\right] S_0^2(\mathbf{x}) = 0 & in \ \Omega \\ \tau(\widehat{\mathbf{x}}_i) = 0 \quad \forall i = 1, \ldots, N_S. \end{cases} \tag{5.7}$$

Compared to the Isotropic Eikonal Equation (5.4), equation (5.6) shows some better numerical properties, as we can put all the singularities in $T(\mathbf{x})$ that could be present in $T_0(\mathbf{x})$, while $\tau(\mathbf{x})$ remains a smooth function so that the accuracy of the forward solution, through a numerical solver, can be further improved.

In the following numerical tests we will use the first two models, namely the Anisotropic Eikonal equation and the Isotropic Eikonal equation.

5.2 Numerical Experiments with the Eikonal equation

In this section we test our methodologies by carrying out some numerical experiments the with Eikonal Equation. All the following experiments are performed with the HMC method. A numerical comparison with SVGD is described in section 5.2.6.

5.2.1 1D Isotropic Eikonal Equation

The first experiment deals with a 1D Eikonal equation in the $\Omega = (0,1)$ domain, with an exponential behaviour of the activation times and a source located at $x = 0$. In this case the analytical solution is:

$$\begin{aligned} T(x) &= 1 - e^{-2x} \\ v(x) &= \frac{1}{2}e^{2x}. \end{aligned} \tag{5.8}$$

We can easily prove that with these choices for the activation time T(x) and the conduction velocity v(x) we satisfy the Eikonal equation, that in this case is simply:

$$\left|\frac{\partial T(x)}{\partial x}\right| = \frac{1}{v(x)}. \tag{5.9}$$

We collect a dataset $\boldsymbol{D}$ of measurements of the activation times T on n_d randomly selected locations (inside the domain Ω). We the perturb our n_d measurements with an additive noise ε, coming from a Gaussian distribution with zero mean and variance σ^2 (where σ is the noise level), as shown in the previous chapters, namely:

$$T_d = T^{true}(x_d) + \varepsilon_d, \;\; \varepsilon_d \sim N(0,\sigma^2), \;\; d = 1,\ldots,n_d. \tag{5.10}$$

We randomly select n_r collocation points in the domain Ω where we enforce the Isotropic Eikonal equation. Finally, we choose n_{dom} points equispaced in Ω to evaluate the final results. The goal is to reconstruct the posterior distribution of the conduction velocity field $v(x)$, plus the whole solution $T(x)$, using the n_d noisy measurements of T collected in $\boldsymbol{D}$. We analyze this case varying just the number of data (n_d) and the noise level (σ). We consider:

- architecture: 2 hidden layers, 50 neurons each;
- dataset: 1000 collocation points, 10, 20 or 40 measurements data, noise level $\sigma = 0.01, 0.05, 0.1$, no mini-batch training;
- $\sigma_D = \sigma_R$, both equal to the noise level σ;

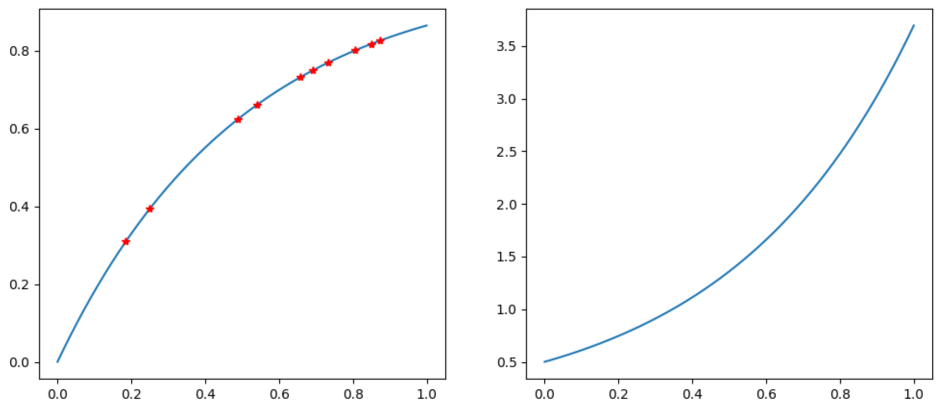


Figure 5.1: Analytical solution of activation times $T(x)$ (showing also 10 measurements without noise) and conduction velocity (right) in the 1D Eikonal experiment.

- HMC with N=3000, M=2500, L=10 and dt=$5 \cdot 10^{-4}$.

For this experiment the computational times to generate all the N=3000 samples with HMC is 3000 sec. In Figure 5.1, we report the analytical results used to generate the dataset (where we considered 10 exact data for T). The noise level in this experiment is quite large (in particular when $\sigma = 0.10$) compared to the range of T, that in this case goes from 0 to 0.86, with a mean close to 0.5. This means that we are adding to the activation times a noise level of 2%, 10% and 20%, respectively.
We use the last M=2500 samples of parameters $\boldsymbol{\theta}$, generated with HMC, to compute the M different predictions of T and v in all the domain points. We compute the euclidean relative error between the sample mean predictions and the analytical solution, as:

$$\begin{aligned} l^2_{rel\ err}(T^{\boldsymbol{\theta}}) &= \frac{||\mathbb{E}\left[T^{\boldsymbol{\theta}}\right] - T^{true}||_2^2}{||T^{true}||_2^2} = \frac{\sum_{i=1}^{n_{dom}} (\mathbb{E}\left[T^{\boldsymbol{\theta}}(x_i)\right] - T^{true}(x_i))^2}{\sum_{i=1}^{n_{dom}} (T^{true}(x_i))^2}, \\ l^2_{rel\ err}(v^{\boldsymbol{\theta}}) &= \frac{||\mathbb{E}\left[v^{\boldsymbol{\theta}}\right] - v^{true}||_2^2}{||v^{true}||_2^2} = \frac{\sum_{i=1}^{n_{dom}} (\mathbb{E}\left[v^{\boldsymbol{\theta}}(x_i)\right] - v^{true}(x_i))^2}{\sum_{i=1}^{n_{dom}} (v^{true}(x_i))^2}, \end{aligned} \tag{5.11}$$

where:

$$\begin{aligned} \mathbb{E}\left[T^{\boldsymbol{\theta}}(x)\right] &= \frac{1}{M}\sum_{m=1}^{M} (T^{\boldsymbol{\theta}_m}(x)), \\ \mathbb{E}\left[v^{\boldsymbol{\theta}}(x)\right] &= \frac{1}{M}\sum_{m=1}^{M} (v^{\boldsymbol{\theta}_m}(x)), \end{aligned} \tag{5.12}$$

to assess the accuracy of the reconstruction of the analytical solution, while for the sake of UQ (as already explained in the previous Chapter 4; we rewrite here the formulas) we first compute (numerically) the standard deviation on every input x_i, $i = 1, \dots, n_{dom}$ of our predictions:

$$\begin{aligned} std(T^{\boldsymbol{\theta}}(x_i)) =& \sqrt{\frac{1}{M}\sum_{m=1}^{M}(T^{\boldsymbol{\theta}_m}(x_i) - \mathbb{E}\left[T^{\boldsymbol{\theta}}(x_i)\right])^2} \quad \forall i = 1,\dots,n_{dom}, \\ std(v^{\boldsymbol{\theta}}(x_i)) =& \sqrt{\frac{1}{M}\sum_{m=1}^{M}(v^{\boldsymbol{\theta}_m}(x_i) - \mathbb{E}\left[v^{\boldsymbol{\theta}}(x_i)\right])^2} \quad \forall i = 1,\dots,n_{dom}. \end{aligned} \tag{5.13}$$

We then compute the standard deviation of the prediction of both T and v, averaged all over the domain and the maximum standard deviation in all the domain:

$$\begin{aligned} ST^{mean}(T^{\boldsymbol{\theta}}) =& \frac{1}{n_{dom}}\sum_{i=1}^{n_{dom}} std(T^{\boldsymbol{\theta}}(x_i)) \\ ST^{max}(T^{\boldsymbol{\theta}}) =& \max_{i=1,\dots,n_{dom}} std(T^{\boldsymbol{\theta}}(x_i)) \\ ST^{mean}(v^{\boldsymbol{\theta}}) =& \frac{1}{n_{dom}}\sum_{i=1}^{n_{dom}} std(v^{\boldsymbol{\theta}}(x_i)) \\ ST^{max}(v^{\boldsymbol{\theta}}) =& \max_{i=1,\dots,n_{dom}} std(v^{\boldsymbol{\theta}}(x_i)). \end{aligned} \tag{5.14}$$

Results are reported in table 5.1: the relative error decreases with a smaller noise and more data (at least in the case of the reconstructed T, not always for v), while the uncertainty increases with noise and decreases with more data.

n_d	noise level (σ)	l^2 rel err (T,v)	ST mean (T,v)	ST max (T,v)
10	0.01	(0.019, 0.145)	(0.006, 0.064)	(0.026, 0.185)
10	0.05	(0.045, 0.120)	(0.033, 0.323)	(0.127, 0.864)
10	0.10	(0.095, 0.207)	(0.055, 0.563)	(0.148, 1.206)
20	0.01	(0.007, 0.067)	(0.004, 0.065)	(0.006, 0.208)
20	0.05	(0.026, 0.079)	(0.019, 0.210)	(0.044, 0.729)
20	0.10	(0.042, 0.083)	(0.035, 0.323)	(0.083, 0.916)
40	0.01	(0.005, 0.063)	(0.002, 0.039)	(0.006, 0.180)
40	0.05	(0.021, 0.102)	(0.014, 0.181)	(0.034, 0.573)
40	0.10	(0.039, 0.064)	(0.027, 0.317)	(0.060, 1.002)

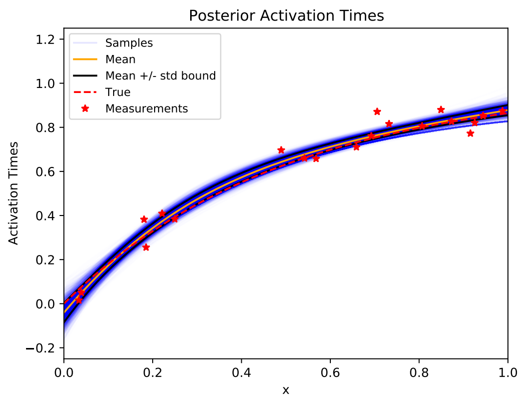
Table 5.1: 1D Eikonal UQ analysis, n.data 10,20,40 and noise 0.01,0.05,0.10.

Analyzing deeper the results in table 5.1, we see that the l^2 relative error for T directly depend on data and noise, while the l^2 relative error for v seems to depend more on the number of data than on the noise level σ.
An interesting behaviour can be seen looking at ST^{mean}, in all the nine experiments we have

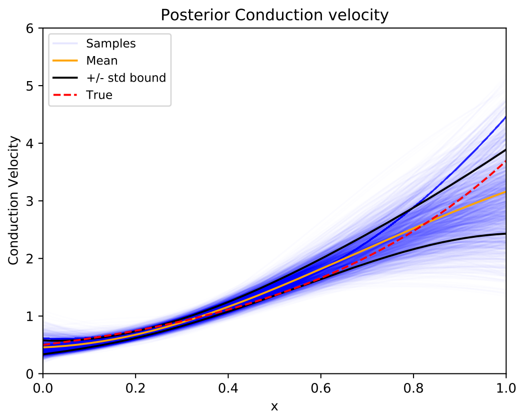
performed, for both T and v. It can be stated a behavior as the following:

$$\begin{aligned} ST^{mean}(T^{\boldsymbol{\theta}}) &= \frac{1}{n_{dom}} \sum_{i=1}^{n_{dom}} std(T^{\boldsymbol{\theta}}(\mathbf{x}_i)) \approx K_T \frac{\sigma}{\sqrt{n_d}}, \\ ST^{mean}(v^{\boldsymbol{\theta}}) &= \frac{1}{n_{dom}} \sum_{i=1}^{n_{dom}} std(v^{\boldsymbol{\theta}}(\mathbf{x}_i)) \approx K_v \frac{\sigma}{\sqrt{n_d}}, \end{aligned} \tag{5.15}$$

where $K_T \approx 1.72$ and $K_v \approx 18$ are constants computed through Least Squares approximation. This behaviour is reasonable since the standard deviation depends linearly on the noise level and depends on $\frac{1}{\sqrt{n_d}}$, as it is reasonable in posterior distribution. In Figure 5.2 and 5.3 two cases of this analysis (20 data, noise 0.05 and 0.10): we show the analytical solution for both T and v, defined in (5.8), and the noisy measurements for T, against the samples (in blue) generated by the method. Also the sample mean (in yellow) with an interval of width equal to the sample standard deviation (in black), defined in (5.12) and (5.13), is reported. As we can see, even in the case where we have a considerable noise level, we are able to both reconstruct the posterior distribution of the solution $T(x)$ with just 20 data, but more important we are able to reconstruct the posterior distribution of the conduction velocity $v(x)$, as we can see on the right of Figure 5.3, since the B-PINN actually solve the Isotropic Eikonal Equation (5.4) in all the collocation points, in addition to use the noisy measurements of T. Figure 5.4 shows the behaviour of ST^{mean} for both T and v, varying the noise level and the number of data, and the formulas (5.15) using the selected values for K_T and K_v.



((a)) Activation times

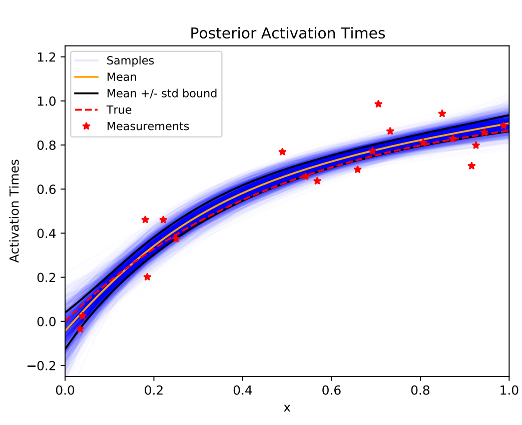


((b)) Conduction velocity

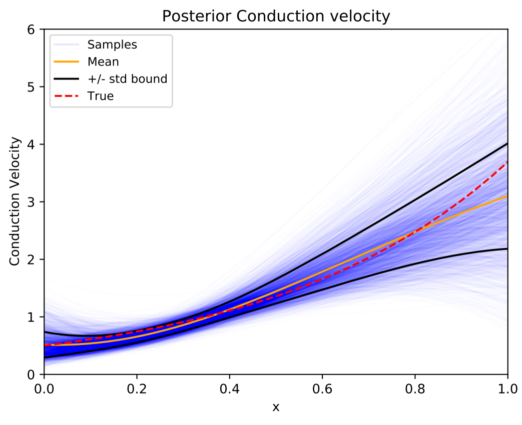
Figure 5.2: 1D Eikonal experiment, results with 20 exact data, noise level = 0.05

5.2.2 1D Isotropic Eikonal with σ_D and σ_R as hyperparameters

In all the previous experiments we fixed $\sigma_D = \sigma_R = \sigma$, where σ is the noise level. This is reasonable since we are assuming the standard deviation parameter of our data likelihood



((**a**)) Activation times



((**b**)) Conduction velocity

Figure 5.3: 1D Eikonal experiment, results with 20 exact data, noise level = 0.10

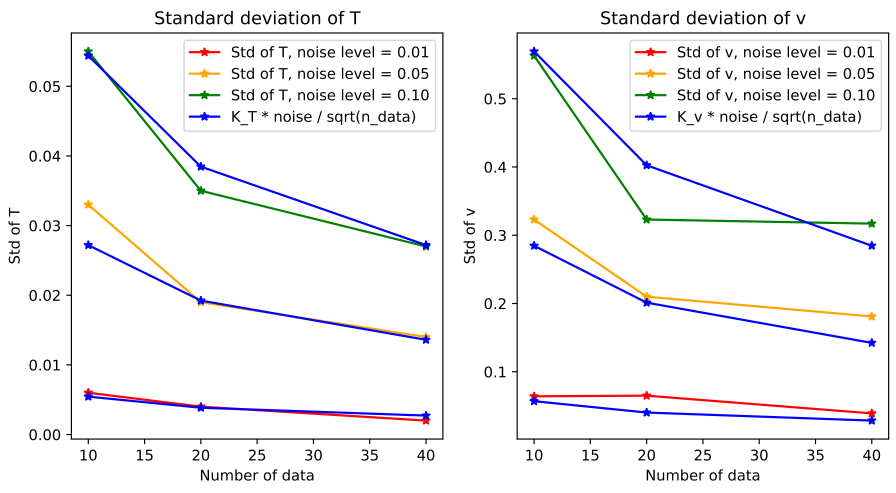


Figure 5.4: Behaviour of ST mean for T and v, varying noise level and n.data, and their approximation using formulas (5.15), 1D Eikonal experiment

(σ_D) to be the same as the noise we are adding to our measurements (the noise level σ), and then we are considering the same value for σ_R. The choice of σ_R is done only for a practical reason: a choice of $\sigma_R \ll \sigma_D$ for instance leads our B-PINN to local minimum, since in the log posterior the component of PDE residual likelihood is much bigger than the Data one. But in principle we are free to choose the value of σ_R without considering the noise level σ, that represents our uncertainty on the PDE model.
The B-PINN could also estimate these two parameters if we do not fix them a priori. For instance, we can specify a prior distribution on them, as shown in the previous chapter, and compute the posterior distribution using the Bayes rule.
We perform two experiments (with the same setting as the previous 1D Isotropic Eikonal

experiment 5.2.1) in which we assign to both σ_D and σ_R a prior distribution with mean 0.01, as:

$$\begin{aligned}\mathscr{P}(\sigma_D^2) &\sim Inv-Gamma(2,10^{-4}),\\ \mathscr{P}(\sigma_R^2) &\sim Inv-Gamma(2,10^{-4}),\end{aligned} \tag{5.16}$$

but we use for the noise level σ values of 0.05 in the first experiment and 0.10 in the second. We can expect that the posterior distribution for σ_D will be close to the noise level value σ, while σ_R should be independent of the noise level. Figures 5.5 and 5.6 confirm this: we can see that the posterior distributions of σ_D (in red) includes the noise level σ values (blue line) in both cases, while this is not true for the posterior σ_R (in yellow), since its posterior distribution (in both the experiment) is centered in smaller values.

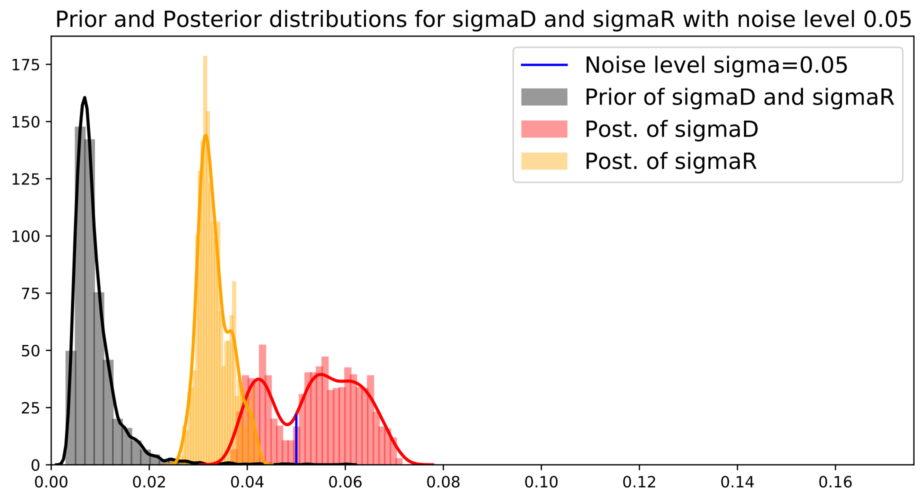


Figure 5.5: Prior and Posterior of σ_D and σ_R for noise level = 0.05

5.2.3 2D Anisotropic Eikonal Equation

The second experiment deals with the 2D Anisotropic Eikonal equation (5.1) in $\Omega = (0,1)^2$, with a source located at $(x,y) = (0.5,0.5)$. The analytical solution in this case is:

$$\begin{aligned}T(x,y) &= 1-\exp(-\sqrt{2(x-0.5)^2+2(x-0.5)(y-0.5)+(y-0.5)^2})\\ a(x,y) &= \frac{1}{\exp(-2\sqrt{2(x-0.5)^2+2(x-0.5)(y-0.5)+(y-0.5)^2})}\\ b(x,y) &= \frac{2}{\exp(-2\sqrt{2(x-0.5)^2+2(x-0.5)(y-0.5)+(y-0.5)^2})}\\ c(x,y) &= \frac{1}{\exp(-2\sqrt{2(x-0.5)^2+2(x-0.5)(y-0.5)+(y-0.5)^2})}\end{aligned} \tag{5.17}$$

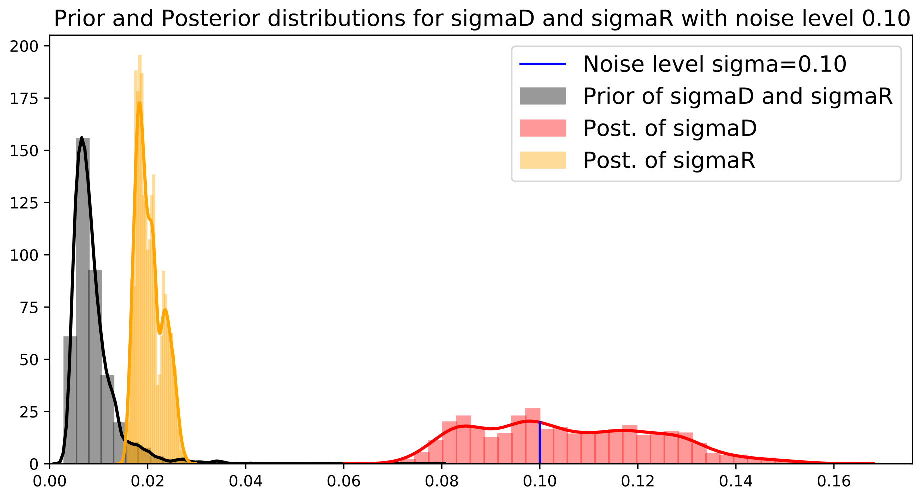


Figure 5.6: Prior and Posterior of σ_D and σ_R for noise level = 0.1

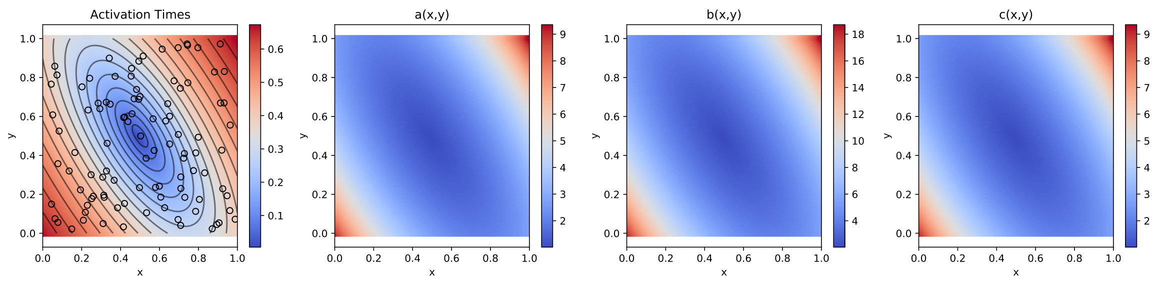


Figure 5.7: Analytical result for T, a, b and c in 2D Anisotropic experiment

with the conduction velocity tensor, as shown in (5.2), given by:

$$\mathbf{M}^2(\mathbf{x}) = \begin{bmatrix} a(x,y) & -c(x,y) \\ -c(x,y) & b(x,y) \end{bmatrix}.$$

Figure 5.7 shows the plot of analytical solutions of T, a, b and c in (5.17).
Since here we are in an anisotropic case, we have to equip the algorithm also with a relation between the entries of the matrix $\mathbf{M}$, that in this case is:

$$\begin{cases} b(x,y) = 2a(x,y) \\ c(x,y) = a(x,y). \end{cases} \tag{5.18}$$

Here we work under the simplifying assumption of homogeneous coefficients. We impose these constraints in a "weak" sense (instead of a "strong" sense that could be outputs only $a(x,y)$ and then compute b and c using the relations (5.18)), imposing the residual of both the relations in our likelihood of residual (in addition to the PDE residual).

Figures 5.8 and 5.9 show the results for $T(x,y)$ and $a(x,y)$ with 100 measurements data, randomly selected inside the domain, affected by a noise level equal to 0.05. For each figure we plot the true field (on the left), the sample mean with the M samples generate by the BPINN, as defined in (5.12) (in the middle), and the sample standard deviations defined in (5.13) (on the right). We can remark that the reconstruction of $a(x,y)$ (and similarly for $b(x,y)$ and $c(x,y)$) is affected by the complexity of the anisotropic equation, in particular close to the boundary and to the center of Ω: the anisotropic problem is more difficult than the isotropic one because of the relations between the three components of M.
Figure 5.10 shows the results along the line $y = x$, in order to visualize all the samples against the true fields. Table 5.2 displays the errors and uncertainty results (defined in Section 5.2.1) for $T(x,y)$ and $a(x,y)$.

n. data	noise	l^2 rel err (T,a)	ST mean (T,a)	ST max (T,a)
100	0.01	(0.013, 0.176)	(0.008, 0.343)	(0.076, 0.949)
100	0.05	(0.042, 0.221)	(0.008, 0.449)	(0.031, 4.029)

Table 5.2: 2D Anisotropic Eikonal UQ analysis, n.data 100 and noise 0.01,0.05.

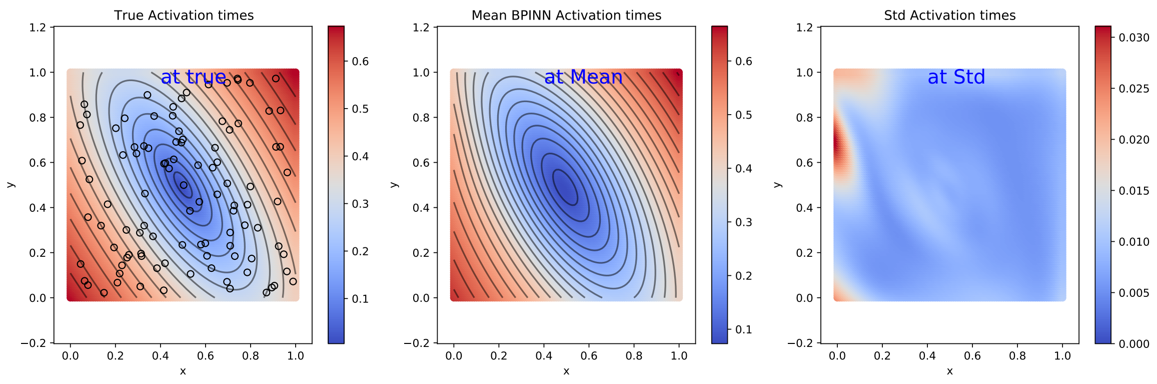


Figure 5.8: Activation times in 2D Anisotropic experiment, 100 data and 0.05 noise

5.2.4 Influence of Physics-Informed formulation on errors and uncertainty in Activation Times

Knowing a physical model in these methods also works as a regularization technique in addition to the possibility of reconstructing the conduction velocities; indeed, it helps us to build a reasonable activation map even if we rely on a very small dataset of noisy measurements of activation times.

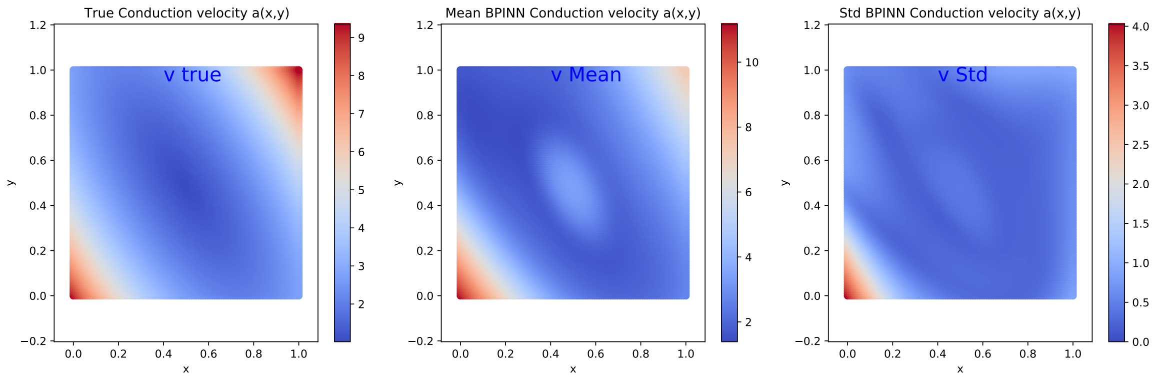
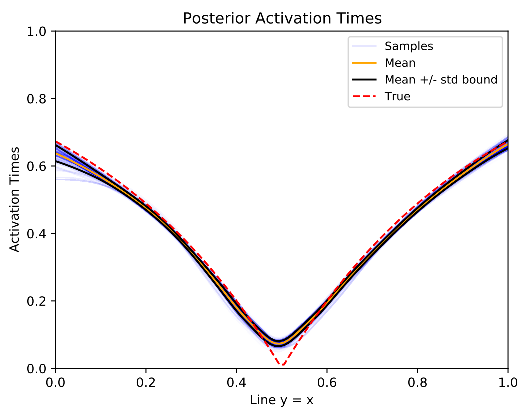
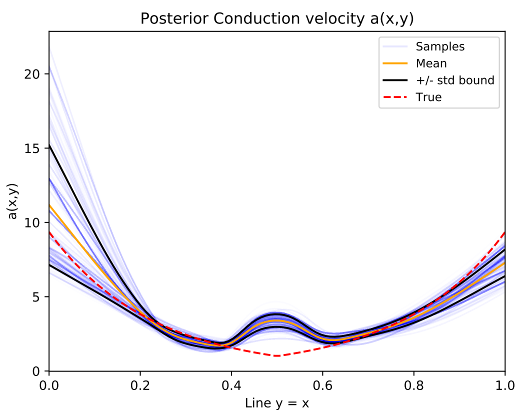


Figure 5.9: a(x,y) in 2D Anisotropic experiment, 100 data and 0.05 noise



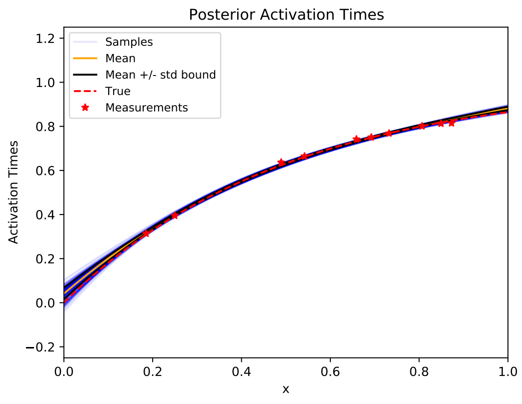
((**a**)) Activation times



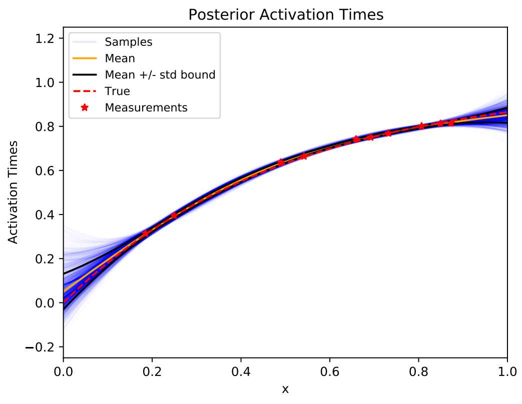
((**b**)) Conduction velocity

Figure 5.10: Activation times and a(x,y) in 2D Anisotropic experiment along the axis $y = x$

To see this, we have tried to run our code without the “Physics-Informed" part, to estimate just T, with a noisy and sparse dataset. We have performed the same training as in the 1D example, with just 10 data, but this time we have set $\alpha_{pde} = 0$, so that our log posterior is built only by two components: the log prior of theta and the log likelihood of activation times data. The results in Figures 5.11, 5.12 and 5.13 show that knowing the physics behind the phenomena also helps in reconstructing T substantially, since the physics informed cases show always better results than the other ones (and the difference increases with noise level). This means that, even if we are not interested in reconstructing the conduction velocities, knowing the physics behind the phenomena from which we have collected the data is useful and allow us to achieve better results: BPINNs in this example outperforms a simple Bayesian Neural Network, that is performing a simple regression task with the activation time measurements data.

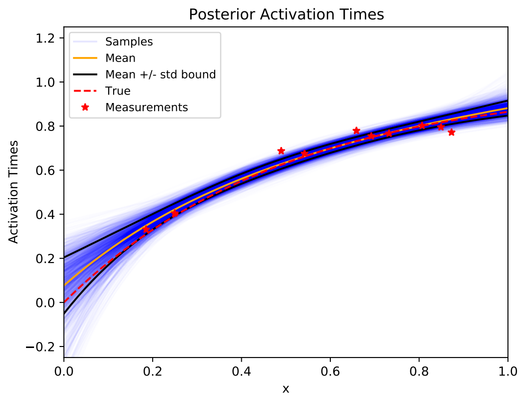


((**a**)) With Eikonal

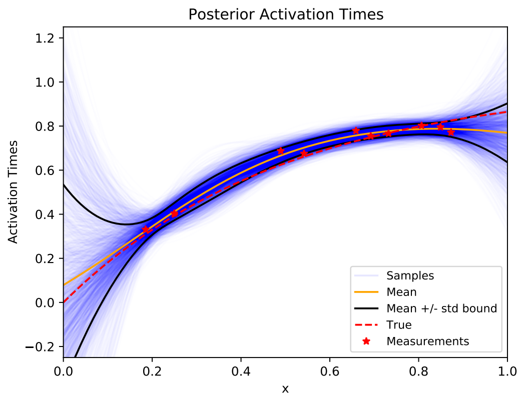


((**b**)) Without Eikonal

Figure 5.11: Comparison of the posterior of Activation times with 10 exact data, noise_lv = 0.01, with Eikonal (left) and without Eikonal (right)

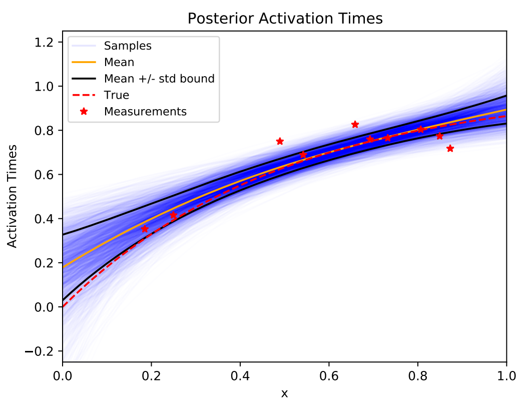


((**a**)) With Eikonal

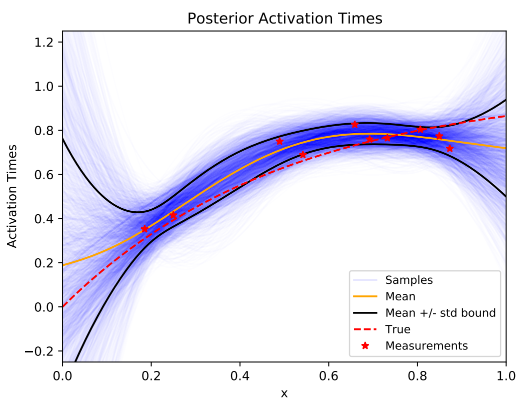


((**b**)) Without Eikonal

Figure 5.12: Comparison of the posterior of Activation times with 10 exact data, noise_lv = 0.05, with Eikonal (left) and without Eikonal (right)



((**a**)) With Eikonal



((**b**)) Without Eikonal

Figure 5.13: Comparison of the posterior of Activation times with 10 exact data, noise_lv = 0.10, with Eikonal (left) and without Eikonal (right)

5.2.5 3D Isotropic Eikonal Equation

We have tested our methodologies also on a 3D Isotropic case, in a prolate geometry, for which the Forward Problem was solved with the Pykonal library [42]. This prolate geometry (see figure 5.14) represents an idealized ventricle.

We have initialized the forward problem by setting a local source at the apex of the prolate, and providing a conduction velocity of 0.5 in the half below, 1.0 in the upper half. The real activation times now range from 0 to more than 28. Here we consider a noise level of 1.0. We use 80'000 collocation points and 100, 400 and 800 measurements data, randomly selected inside the domain. For this experiment we use a bigger network, indeed we use an architecture of 5 hidden layers, 100 neurons each. In Figures 5.14 and 5.15 the results (true, sample mean and sample standard deviation) with 400 data and noise level 1.0, while in figures 5.16 and 5.17 with 800 data and same noise level are reported. We can clearly see that we can achieve better results (especially in the reconstruction of the conduction velocity v) with more data. In Figure 5.18 we display the relation between uncertainty (in terms of standard deviation) in activation times and localization of exact data: we can appreciate the fact that the uncertainty seems bigger in those portions of the domain where only few measurements are selected. (All these 3D plots are made using the library Mayavi, `https://docs.enthought.com/mayavi/mayavi/`). Table 5.3 shows the results (relative error, ST^{mean} and ST^{max}) for this 3D experiment.

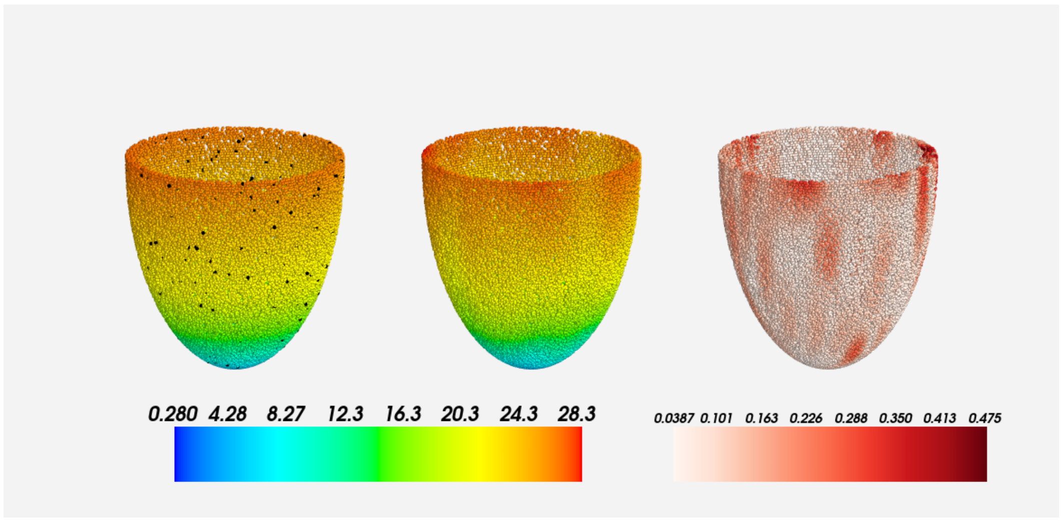


Figure 5.14: Activation times (True, mean BPINN and std) with 400 exact values in 3D Eikonal experiment

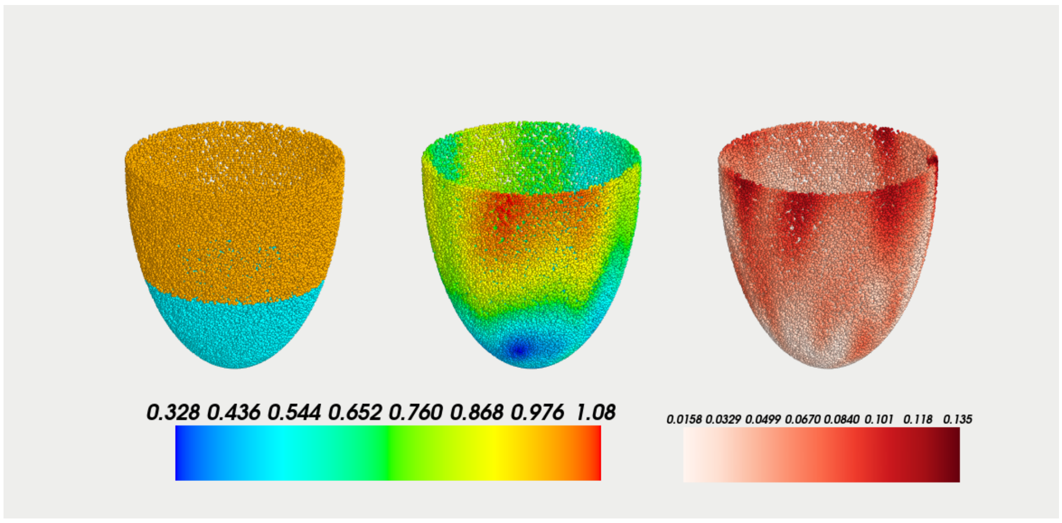


Figure 5.15: Conduction velocity (True, mean BPINN and std) with 400 exact values in 3D Eikonal experiment

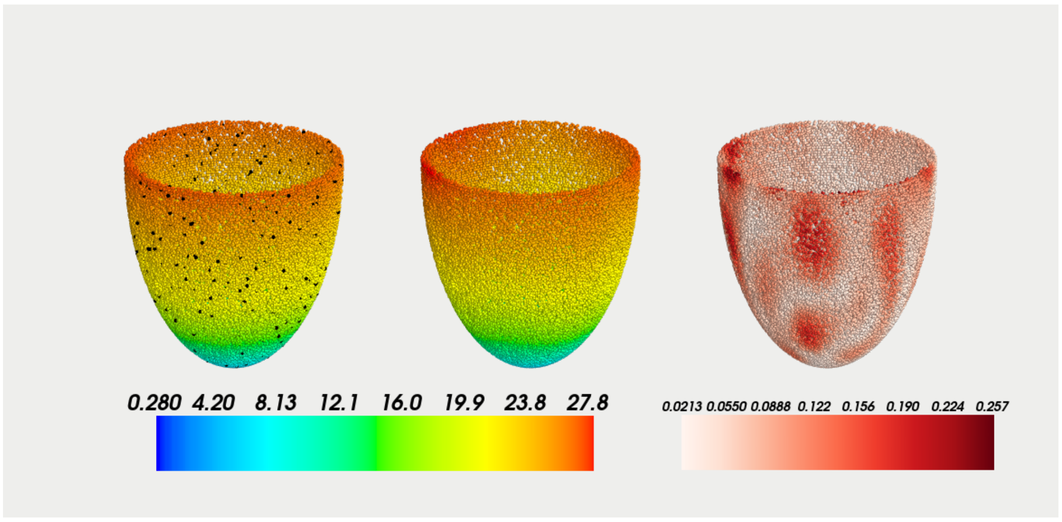


Figure 5.16: Activation times (True, mean BPINN and std) with 800 exact values in 3D Eikonal experiment

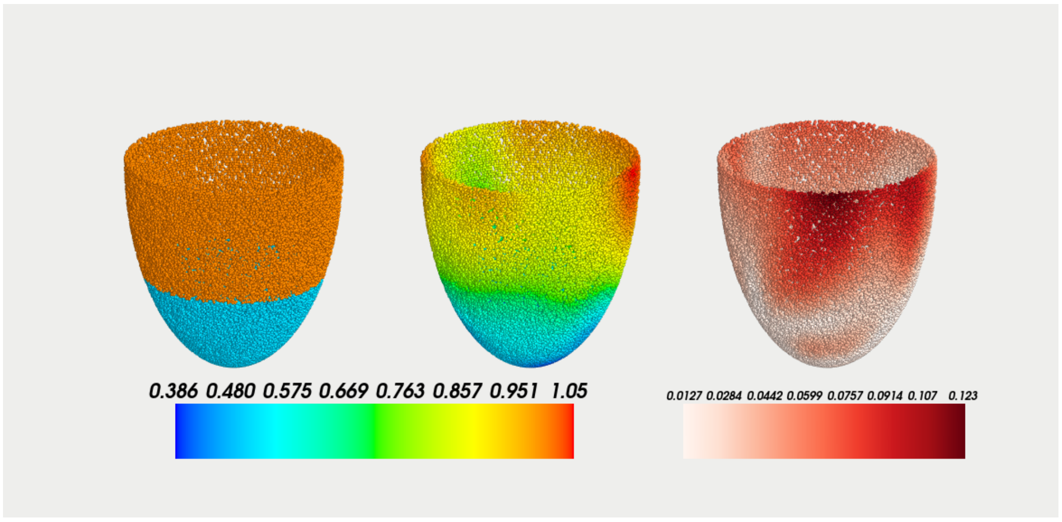


Figure 5.17: Conduction velocity (True, mean BPINN and std) with 800 exact values in 3D Eikonal experiment

n. data	noise	L^2 rel err (T,v)	ST mean (T,v)	ST max (T,v)
100	1.0	(0.078, 0.480)	(0.292, 0.063)	(1.481, 0.223)
400	1.0	(0.030, 0.249)	(0.110, 0.046)	(0.475, 0.135)
800	1.0	(0.019, 0.159)	(0.074, 0.035)	(0.257, 0.123)

Table 5.3: 3D Eikonal UQ analysis, n.data 100,400,800 and noise 1.0.

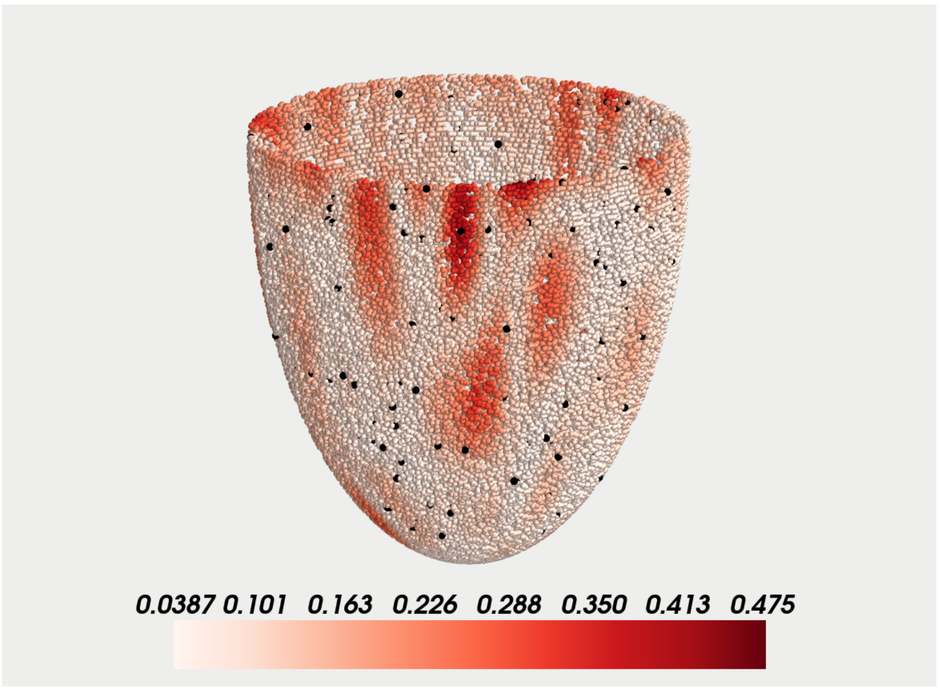


Figure 5.18: Uncertainty (standard deviation) and exact data positions in Activation times in 3D Eikonal experiment

5.2.6 Comparison with SVGD

Using the SVGD algorithm, we found good results in terms of errors in the reconstruction of T and v (especially with a small noise), but weaker results in terms of Uncertainty Quantification. We compare the findings of 1D-case, with 10 data, noise level 0.01 and 0.05, obtained with the HMC method and with a similar setting employed with the SVGD method. We use here 30 NNs to approximate the posterior: choosing this parameter is the most critical part of the algorithm, since we must ensure a suitable trade-off between computational efficiency and accuracy. In addition to this, having 30 NNs stored at the same time can cause a memory allocation problem.

From the results in table 5.4, we can conclude that the SVGD method underestimates the standard deviations more than HMC: this problem can be motivated by the small number of

Method	n. data	noise	L^2 (T,v)	std_{mean} (T,v)	std_{max} (T,v)
HMC	10	0.01	(0.019,0.145)	(0.006,0.064)	(0.026,0.187)
HMC	10	0.05	(0.045,0.120)	(0.033,0.323)	(0.127,0.864)
SVGD	10	0.01	(0.034,0.066)	(0.0021,0.021)	(0.018,0.089)
SVGD	10	0.05	(0.0717,0.354)	(0.0078,0.139)	(0.055,0.489)

Table 5.4: 1D Isotropic Eikonal analysis, comparison between HMC and SVGD, n.data 10,20,40 and noise 0.01,0.05,0.10

NNs, too few to compute effectively a numerical standard deviation. We could increase that number but, as said before, we could also occur in problems of memory allocation, even if using more NNs, we might increase computational time.
Also in this case, as shown in the previous example, the HMC method achieves better results but taking a bigger time, since it requires twice the computational time of SVGD in this example.

5.3 Active Sampling

In this section we test the Active Sampling method described in Section 3.3, to actively sample new data in order to minimize the uncertainty on the predicted value of T. We will show that this method of adding new data outperforms both random sampling or uniform sampling and proves to be useful when acquiring new data is very expensive or time-consuming.

5.3.1 1D Isotropic Eikonal Equation, Active Sampling with HMC

To test the Active Sampling algorithm, we run an 1D Eikonal experiment using the HMC method with 5 data randomly selected, noise level of 0.05 and we use the Active Sampling algorithm to iteratively sample other 4 points. The setting is the same as the previous 1D UQ Eikonal experiment of Section 5.2.1, with the analytical solution (5.8). The initial data positions are $\{0.185,0.541,0.872,0.732,0.806\}$. The Active Sampling algorithm iteratively sample four new data: the first two are close to the left boundary, 0 and 0.001; the third, 0.269, falls in the biggest region without any data between 0.185 and 0.541; the last, 0.968, is close to the right boundary.

As shown in Figures 5.19 and 5.20, the results with the new 4 sampled data improve considerably: indeed, the variance on T seems to be almost constant along all the domain. Also the ability to reconstruct V improves. The new data have been located in a suitable way:

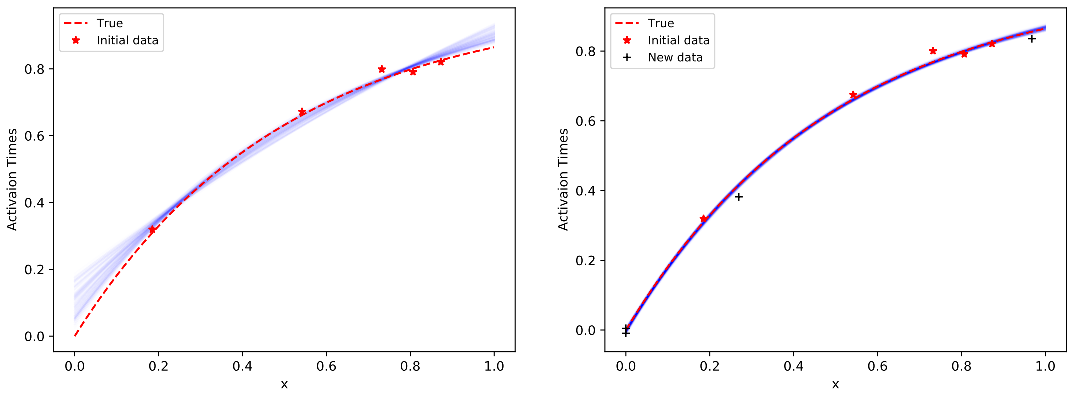


Figure 5.19: Activation times, before (left) and after (right) Active Sampling

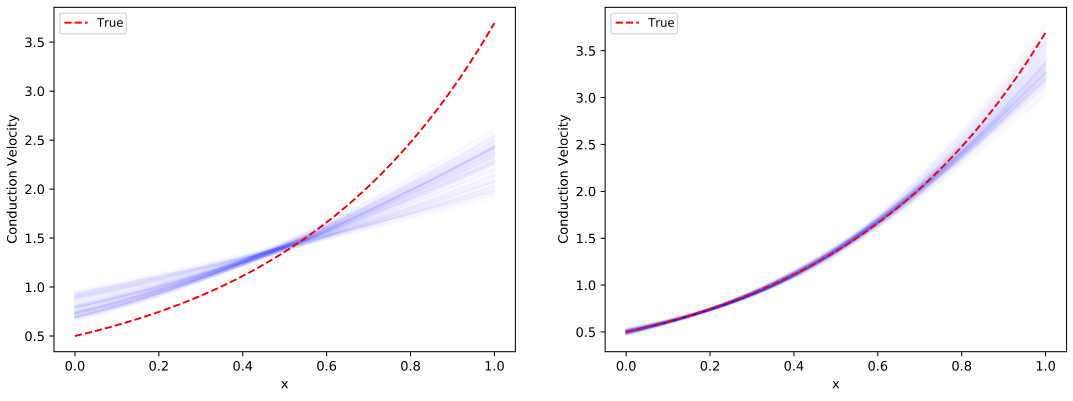


Figure 5.20: Conduction velocity, before (left) and after (right) Active Sampling

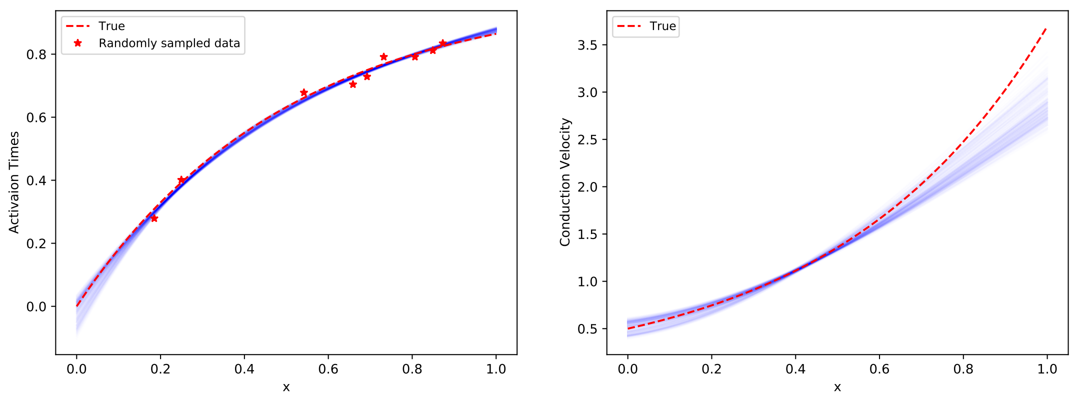


Figure 5.21: Comparison result with 9 randomly sampled data

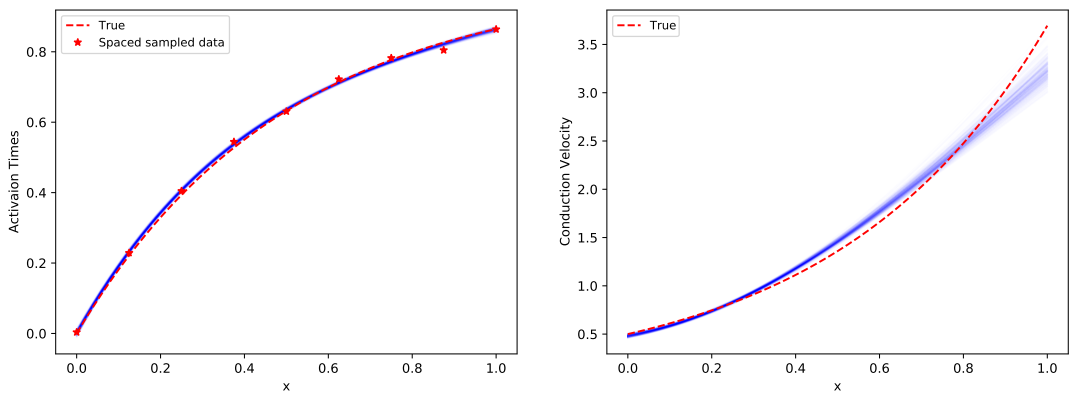


Figure 5.22: Comparison result with 9 sampled data on a regular grid

the big variability of the predictions for T on the left part of the domain has required two new points, $x = 0$ and $x = 0.001$. Then, the algorithm finds a big variability of the predictions for T in the middle of the domain, where the distance from the other measurements was the largest. Finally, we require also an evaluation on the right boundary.
To measure the effectiveness of the Active Sampling algorithm, we can compare these results with experiments run by sampling 9 data sampled in a classical way, for instance randomly or uniformly on the domain.

Figures 5.21 and 5.22 show the corresponding results: it seems clear that in these cases we are getting a lower ability to reconstruct both the activation time and the conduction velocity field. Using data acquired with Active Sampling yields more accurate solutions, however requiring about 5 times the computational time of the other methods: every time we add a new data we need to retrain the network. This can be useful in situations where the main bottleneck is the cost of data acquisition, not time consumption.

5.3.2 2D Isotropic Eikonal Equation, Active Sampling with SVGD

In this second example, we show the ability of Active Sampling on a 2D Eikonal experiment to improve the identification task. The exact solutions of this example are shown in Figure 5.23 [43]. In this example we are going to use the SVGD method, that, as we have shown

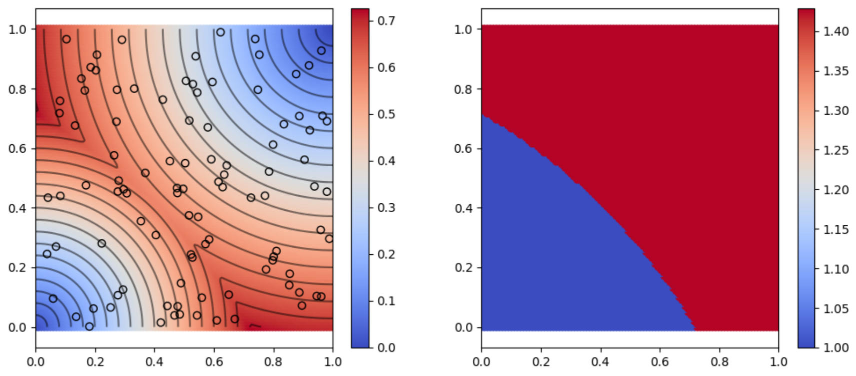


Figure 5.23: Analytical solution of activation times (left) and conduction velocity (right) in 2D Eikonal Active Sampling example

before, entails a bigger uncertainty on the regions with less data. We initially use 100 measurements with a noise level of 0.01. We then add a single new sampled measurement in the position of maximum uncertainty, using our Active Learning algorithm. The results are shown in Figure 5.24, where we plot the posterior standard deviation on Activation Times and the position of the measurements, before and after Active Sampling. The new added point is able to remove the region of bigger uncertainty we had. After Active Sampling, with just a single new measurement over the 100 data we already had, we end up with a more homogeneous posterior standard deviation all over the domain.

5.4 Transfer Learning in subdomains

In this section we test a new method to split the training stage in subdomains and taking advantage of Transfer Learning, as described in Section 3.4, and we explain why this new methodology can be useful for cardiac applications. We test this algorithm both on a 2D and a 3D experiments. We start by highlighting that in a real clinical application, a grid of electrodes is used to record activation times on heart chambers surfaces [44],[45]. A multi electrode catheter is applied all over the surface in order to map the activation times in the chamber. An example of multi electrode cathether and its use on a heart chamber is shown in Figure 5.25. A multi electrode catheter has a grid of sensors (16 in the figure) and is able

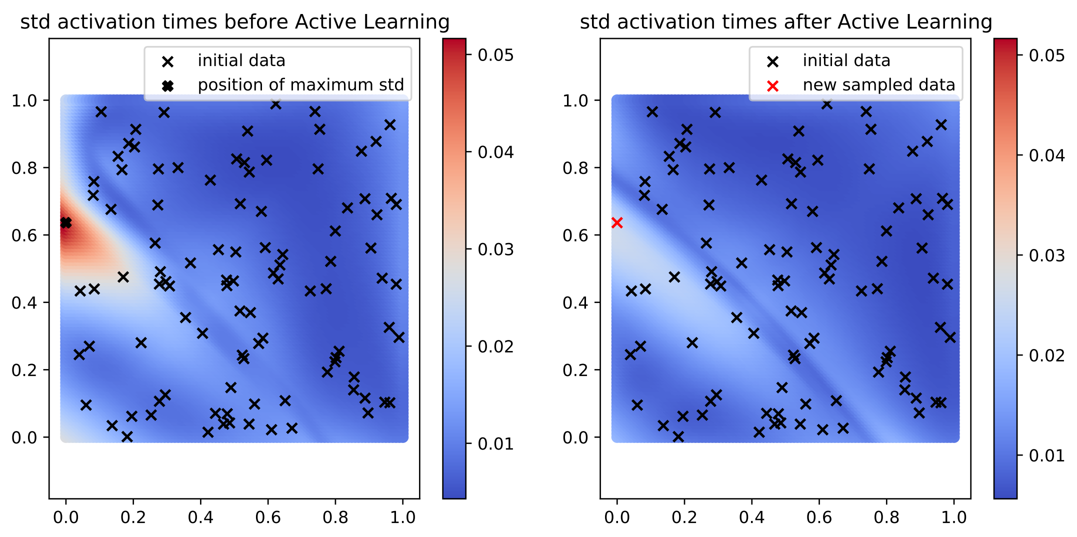


Figure 5.24: Posterior standard deviation for Activation times before (left) and after (right) a new single data was added with Active Sampling

to measure the activation times in those locations in real time: step by step the electrode is moved along the surface to map all the heart tissue. For this reason, the mapping procedure at every step is accurate in that particular region (that coincides with the multi electrode catheter area). The domain Ω is split in some smaller subdomains, which can be mapped sequentially. Inspired by this fact, we have exploited the idea of subdomain splitting also in our B-PINN procedure, ending up with a new algorithm, called "Patch B-PINN", which takes advantage of both the domain splitting and the concept of Transfer Learning in a Neural Network [36].

5.4.1 2D test

Let us start from a 2D Isotropic test case, in a domain $\Omega = (0,1)^2$, subdivided in 4 smaller subdomains $\Omega_1 = (0,0.5) \times (0.5,1)$, $\Omega_2 = (0.5,1) \times (0.5,1)$, $\Omega_3 = (0.5,1) \times (0,0.5)$ and $\Omega_4 = (0,0.5) \times (0,0.5)$. In this simple example all the subdomains have their own conduction velocity, as shown in Figure 5.26, where we plot the true solution for this example.
For this test case, we employ a SVGD Bayesian PINN implementation with 15 particles for simplicity. We set the number of epochs N equal to 100, so in the other subdomains except the first we train just for 10 epochs. The number of collocation points and exact measurements are 10000 and 100 respectively, divided in 25000 and 25 for each subdomain. The experiment is carried out as follows: we first train all the 4 subdomains for N=100 epochs separately, for comparison purpose. Then we apply our Patch B-PINN algorithm:

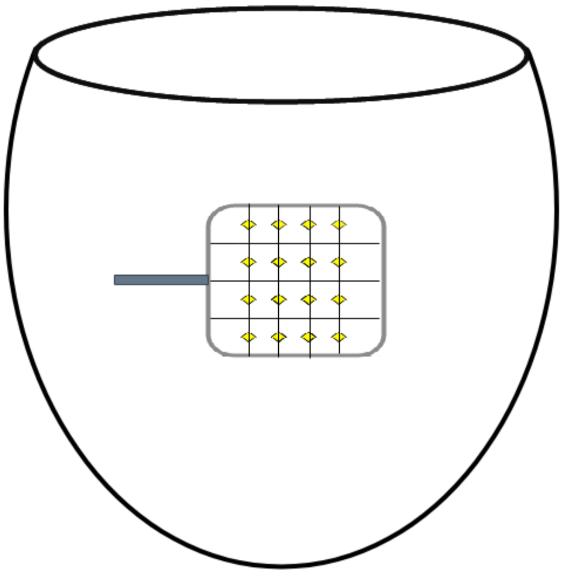


Figure 5.25: Multi electrode catheter used to measure activation times on heart tissues

after collecting $\boldsymbol{\theta}_1$, we use it as initial guess for $\boldsymbol{\theta}_i$, $i = \{2,3,4\}$ (parameters for NN in the subdomains Ω_2, Ω_3 and Ω_4) and train all the $\boldsymbol{\theta}_i$ separately for just 10 epochs. An additional training with Transfer Learning and N=100 epochs in all the subdomains is done in order to compare also the TL performance boost with a longer training. Results are compared with the total loss, defined as a sum of the quadratic error on T on exact data ($\mathbf{D}_i$) and the quadratic error on residuals ($\mathbf{R}_i$) using the samples mean:

$$Tot.\ loss_i = \frac{1}{n^i_{ex}} \sum_{j=1}^{n^i_{ex}} |\mathbb{E}\left[T^{\boldsymbol{\theta}}_{i,j}\right] - T_{i,j}|^2 + \frac{1}{n^i_{coll}} \sum_{l=1}^{n^i_{coll}} |\mathbb{E}\left[R^{\boldsymbol{\theta}}_{i,l}\right] - 0|^2, \quad i = 2,3,4. \tag{5.19}$$

Results are shown in Figures 5.27, 5.28 and 5.29.

For what concerns Ω_2 and Ω_4, 10 epochs with TL seems to be an acceptable training time: in Ω_2 Transfer Learning is so effective that with just 10 epochs we can get a lower loss than with a full training, while in Ω_4 with 10 epochs and TL we get the same result we would obtain with about 40 epochs without TL. On the other hand, on Ω_3 the transfer learning efficacy is so low that with just 10 epochs we cannot see a big difference with respect to the standard case. This problem could be caused by the particular form of activation times T in Ω_3 (lower right quadrant), as shown in Figure 5.26. Compared to the other subdomains, in Ω_3 we have the collision of two different wave fronts, that produces that particular shape of

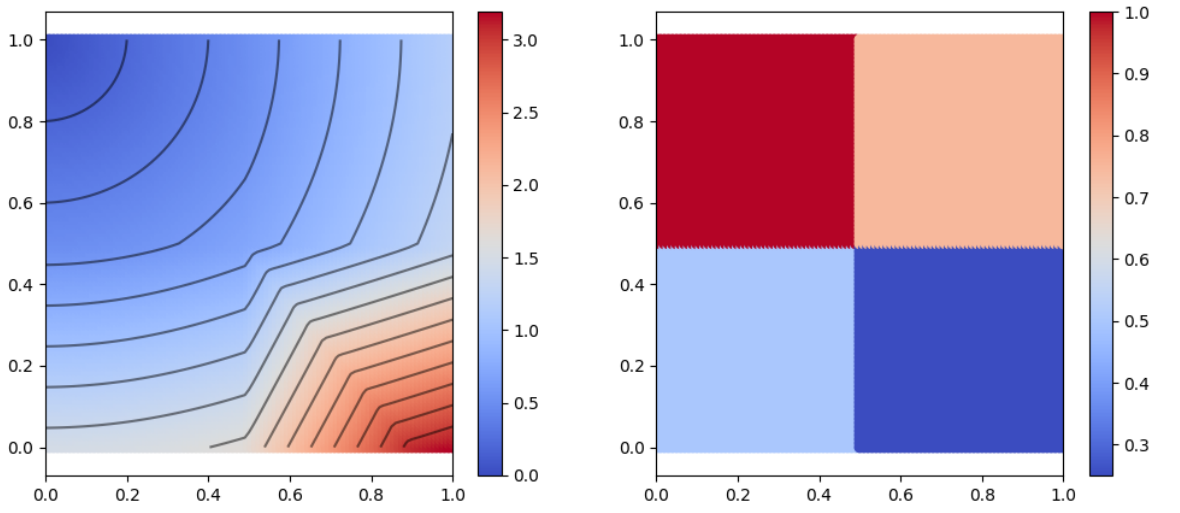


Figure 5.26: Exact solution for activation times and conduction velocity in Ω, 2D Isotropic Eikonal Patch BPINN example

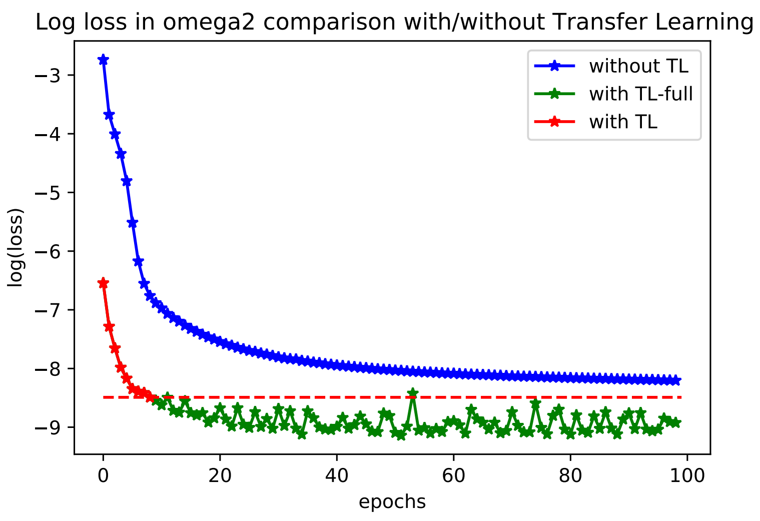


Figure 5.27: Log(Total loss) in Ω_2 with and without Transfer Learning, 2D Isotropic Eikonal Patch B-PINN example

T. This problem can be easily overcome redefining the activation maps in every subdomains, for instance using always a source localization in the upper left vertex of the subdomain.
Finally we show that, considering all the 100 epochs, Transfer learning always provides a performance boost. In particular, we found all the three types of performance improvement defined in the previous section in Ω_2 and Ω_4, while only two (a better intercept and a better slope) in Ω_3.

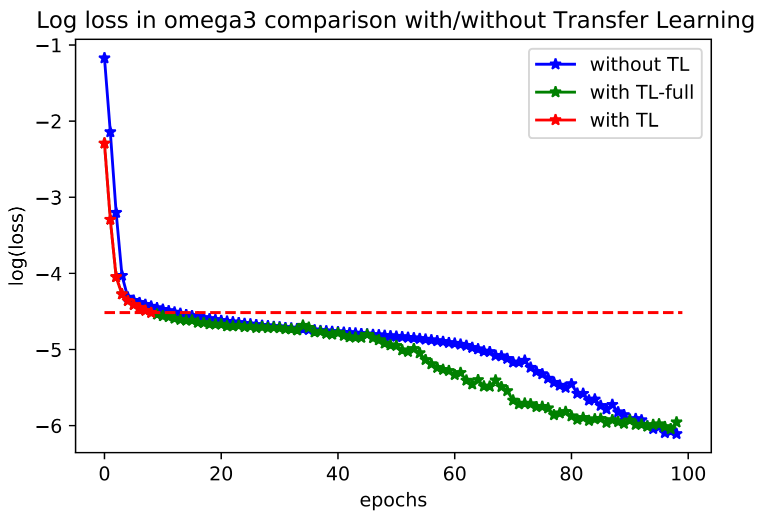


Figure 5.28: Log(Total loss) in Ω_3 with and without Transfer Learning, 2D Isotropic Eikonal Patch B-PINN example

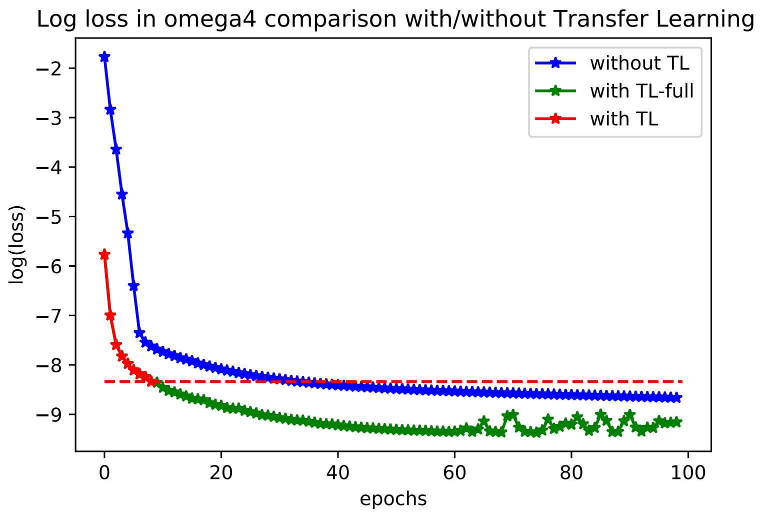


Figure 5.29: Log(Total loss) in Ω_4 with and without Transfer Learning, 2D Isotropic Eikonal Patch B-PINN example

5.4.2 3D test

As a final example of this methodology, we consider a 3D prolate geometry. Figure 5.30 shows the exact solutions for both activation times and conduction velocity in the domain Ω, plus the division in subdomains. In this case we have a conduction velocity equal to 1 in all the domain, except for a low conduction velocity region (where $v = 0.5$ or 0.75, respectively). This can represent a real clinical application, in which we are interested in find low conduction areas.

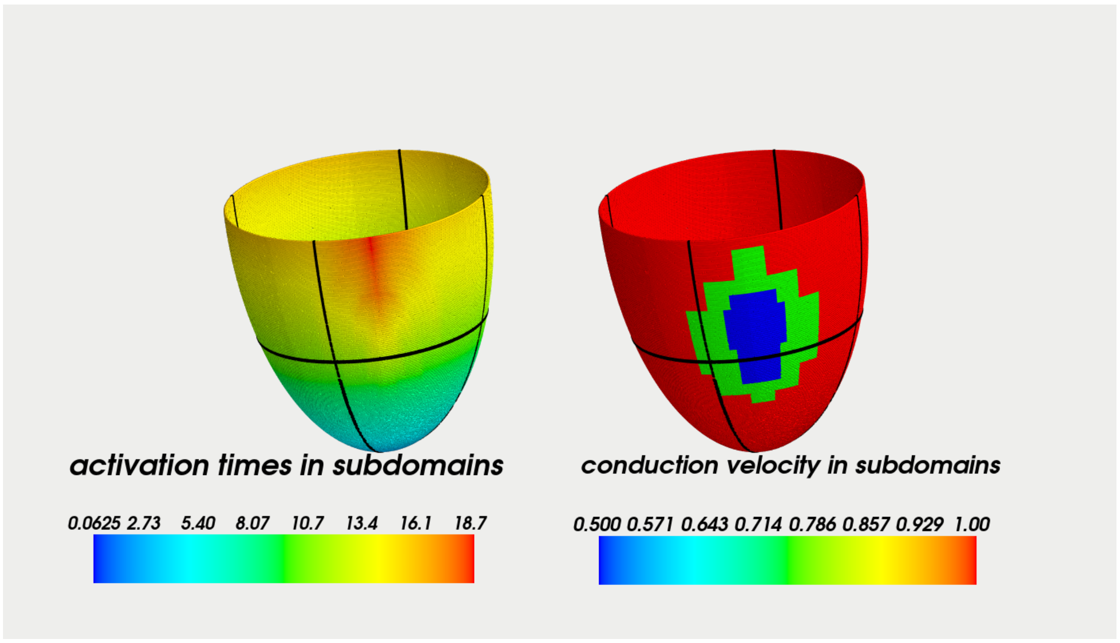


Figure 5.30: Exact solution of activation times and conduction velocity in Ω, 3D Isotropic Eikonal Patch B-PINN example

For this test, we have used a SVGD B-PINN implementation, employing just 5 NNs with an architecture of 5 hidden layers, 100 neurons each. The full domain Ω is divided in 8 subdomains, as shown in Figure 5.30. We first train each subdomain separately for 100 epochs. Then we select one of them (Ω_1) and use it for Transfer Learning on the other subdomains, trained for just 20 epochs this time. As Ω_1 we have choose a subdomain where $v = 1$ always.

We compute the Euclidean relative error (using the formulas (5.11)) for each subdomain for both T and v, and we use the mean between the subdomains error to measure the total error. With a full training we obtain a mean Euclidean relative error of 0.0167 for T, 0.190 for v; while using the Transfer Learning with just 20 epochs we obtain a mean Euclidean rel. error of 0.0198 for T and 0.204 for v.

As we can seen from the results above, with the Transfer Learning approach we have less accurate results, but remarkable computational savings: with the full approach we are run the training for a total of 800 epochs (100 epochs for each subdomain), with a computational time of approximately 4000 sec; on the other hand, with the TL approach, we are training for just a total of 240 epochs (100 epoch for the first subdomain, then just 20 for all the others), 1200 sec. This big speed up could justify a slightly less accurate results, especially during a real clinical mapping of a patient.

In Figure 5.31 we compare the true conduction velocity against the mean BPINN results in Ω_3, that is the subdomain where we have most of the low conduction region. We can see

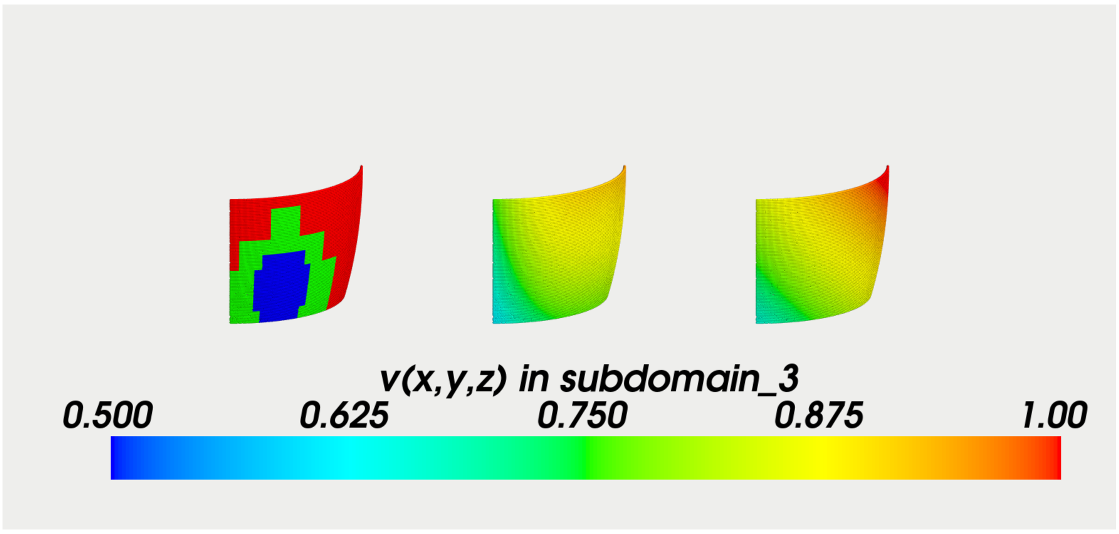


Figure 5.31: Conduction velocity in Ω_3, 3D Isotropic Patch BPINN example.
True (left), mean of BPINN with full training (center), mean of BPINN with Transfer Learning (right)

that both the approaches (Full training and Transfer Learning) recognize a lower conduction velocity region in the lower left part of the domain, even if they are not able to completely reconstruct the exact solution.

Chapter 6

Conclusions

In this thesis we have demonstrated the ability of Bayesian Physics-Informed Neural Network to solve the Inverse UQ problems governed by stationary PDEs, by applying the Physics-Informed Neural Networks (PINNs) paradigm in a Bayesian framework. In the last few years, PINNs have been used to tackle several challenges in different scientific areas with extremely promising results. Indeed, solving an Inverse UQ problem for PDEs is a challenging task from a computational standpoint, which can be effectively tackled by PINNs. Despite an in-depth numerical analysis of PINNs in terms of convergence and accuracy is still lacking, some preliminary results (see, e.g., [46]) might be helpful in order to assess a priori the performance of a PINN method, at least in simplified contexts.

Bayesian PINNs (B-PINNs) represent an extremely recent extension of PINNs, for which very few examples of applications, and no theoretical results, are available. The results provided in Chapter 4 can be seen as one of the first attempts of validation of the B-PINN technique, in the case of simple parametric Elliptic equations. Indeed, the possibility to compare our results with the true posterior distribution, available in a closed form in the case of a one-dimensional Elliptic PDE with measurements acquired on the solution, has allowed us to validate these methods.

In Chapter 5 we have shown the capability of BPINNs to solve the Inverse UQ problems related with the Eikonal equation in cardiac electrophysiology. We have addressed many test cases of increasing complexity, ranging from one- to three-dimensional problems. In a one-dimensional case, we have performed a comprehensive UQ analysis using the Hamiltonian Monte Carlo method (HMC) to explore the posterior distribution of the parameters, showing the impact of the number of measurements and the noise level on the obtained results. Then, we have turned to a two-dimensional anisotropic Eikonal model, and to a three-dimensional case set on a prolate geometry, that can be seen as a simple model of a ventricle geometry. In all these cases, even if the HMC method provides better performances in terms of accuracy, it requires an expensive tuning stage to reach convergence; on the other hand, the Stein Variational Gradient Descent (SVGD) method is more robust and faster, however featuring possible memory allocation issues (when storing additional NNs at the same time), and possibly providing weaker results in terms of the estimated (conduction velocity) fields.

In Section 5.4, we have shown the effectiveness of Active Sampling in reducing uncertainty and improving results adding just few new data. This algorithm can be very useful in all those cases where we need to rely on small datasets, because of the high costs entailed by the acquisition of new measurements. Finally, we have proposed a new method that takes advantage of Transfer Learning in those situations where we have grids of measurements. For the cases considered, we have assumed to be able to collect activation times measurements coming from a grid of sensors, as it happens when using cathethers for electrical mapping in the clinical practice. After collecting a new grid of measurements, instead of retrain the network from scratch, we use Transfer Learning to reach a substantial speed up during the training stage, and improve the accuracy in the identified quantities since we focus on smaller areas. As we have shown in Section 5.4, Transfer Learning can recognize small conduction velocity areas in a three-dimensional geometry. To reach this goal, we have used just five networks within the SVGD method, and only 100 epochs for the sake of computational time. Using a bigger number of networks and more epochs could potentially yield even more accurate results. We also highlight that Transfer Learning is more suitable in the case the SVGD method, rather than the HMC method, is employed for sampling from the posterior distribution.

To conclude, we have shown that B-PINNs can greatly enhance the reconstruction of uncertain quantities or fields from scattered measurements. Indeed, trying to estimate activation times using neural network regression or interpolation without involving the physical model, and then try to compute the conduction velocity, can lead to very poor results. On the other hand, computing both quantities at the same time, as we can do using B-PINNs, and taking advantage of the knowledge of a physical model for the quantities being observed, acts as a goal-oriented regularization technique and greatly enhances results in terms of uncertainty reduction.

Appendix A

Automatic Differentiation

By Automatic Differentiation (AD) [24] we indicate all the techniques to evaluate the derivative of a function by computer programs applying the chain rule. The basic idea behind these methods is that a computer evaluates a function, providing an input, by a composition of basic functions that use elementary arithmetic operations and elementary functions.
Since the forward pass from input to compute the output is a chain of basic operations, we can compute derivatives using this chain backward, employing the chain rule for derivative, knowing only the basic derivatives (derivative of the sum, product by a constant, polynomials, exponential, trigonometric functions and so on). In this way, we can achieve better results than every other approximate numerical differentiation formula, since here we are computing the exact derivative.
These methods are widely used in every Machine Learning methods, and so also in Neural Networks, mostly for computing back-propagation derivatives of parameters $\boldsymbol{\theta}$, in according to loss functions being minimized. In this way, we can easily update the parameters $\boldsymbol{\theta}$ after having collected $\nabla_{\boldsymbol{\theta}}\mathscr{L}(\boldsymbol{\theta})$ through automatic differentiation, using an optimizer like the Gradient Descent method and Adam optimizer.
In addition to this classical use of Automatic Differentiation, in this thesis (as well as in every work of the field "Scientific Machine Learning") we employ AD to compute derivatives of output with respect to input data (collocation points), and not only with respect to parameters. This allows us to compute for instance $\frac{\partial T}{\partial x}$, $\frac{\partial T}{\partial y}$ and $\frac{\partial T}{\partial z}$, in order to have $\nabla T(\mathbf{x})$ and compute the residual of the PDE (for instance the Eikonal Equation) representing the physical model at hand.
Let us analyze more in detail how this works in our case. In order to keep everything simple, suppose we are in the case of 1D isotropic Eikonal equation (see Section 5.2.1 for a detailed introduction), with only 2 hidden layers, each of them having 5 neurons. Figure A.1 shows the resulting architecture, where for completeness we have drawn also the bias vectors (in

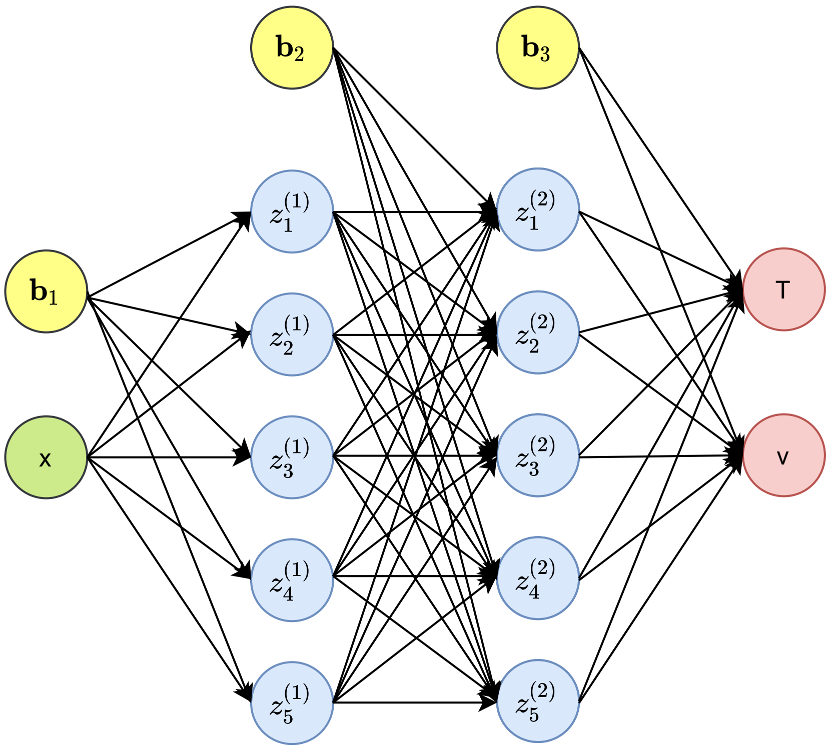


Figure A.1: Architecture in 1D isotropic Eikonal

yellow).

After our 1D input (that is simply the first neuron x), to map x in the first layer we use weights matrix $\mathbf{W}^{(1)} \in \mathbb{R}^{1\times 5}$ and bias vector $\mathbf{b}^{(1)} \in \mathbb{R}^5$, the same in the second layer with $\mathbf{W}^{(2)} \in \mathbb{R}^{5\times 5}$ and $\mathbf{b}^{(2)} \in \mathbb{R}^5$, and finally $\mathbf{W}^{(3)} \in \mathbb{R}^{5\times 2}$ and $\mathbf{b}^{(3)} \in \mathbb{R}^2$ to get outputs T and v. The simple chain of computations that goes from input x to output (T, v) is therefore:

$$\begin{aligned}
\mathbf{a}^{(\mathbf{1})} &= \mathbf{W}^{(1)} x + \mathbf{b}^{(1)}, \\
\mathbf{z}^{(\mathbf{1})} &= \sigma^{(1)}(\mathbf{a}^{(\mathbf{1})}), \\
\mathbf{a}^{(\mathbf{2})} &= \mathbf{W}^{(2)} \mathbf{z}^{(\mathbf{1})} + \mathbf{b}^{(2)}, \\
\mathbf{z}^{(\mathbf{2})} &= \sigma^{(2)}(\mathbf{a}^{(\mathbf{2})}), \\
\mathbf{a}^{(\mathbf{3})} &= \mathbf{W}^{(3)} \mathbf{z}^{(\mathbf{2})} + \mathbf{b}^{(3)}, \\
(T, v) &= (a_1^{(3)}, a_2^{(3)}),
\end{aligned} \tag{A.1}$$

where $\sigma^{(l)}$ are our activation functions, that in this case are equal to the Swish activation function

$$\sigma^{(l)}(x) = x\, sigmoid(x) = \frac{x}{1+e^{-x}} \quad \forall l = 1, 2$$

and equal to identity function for $l = 3$ (since it is equal to identity we do not explicitly indicate it). Using the chain rule to compute the derivative is simple, indeed:

$$\frac{\partial T}{\partial x} = \frac{\partial a_1^{(3)}}{\partial \mathbf{z}^{(2)}} \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{a}^{(2)}} \frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{z}^{(1)}} \frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{a}^{(1)}} \frac{\partial \mathbf{a}^{(1)}}{\partial x}, \tag{A.2}$$

and knowing the exact form of derivative of swish activation function:

$$\partial \sigma(x) = \sigma(x) + sigmoid(x)(1 - \sigma(x))$$

we can compute the derivative of T with respect to x directly using the architecture of the network.

Appendix B

True posterior distribution in 1D Elliptic parametric computation

Let us now derive the formulas to compute v_{post} and σ_{post} in (4.9).
We are in a 1D Elliptic parametric problem setting, and we know that the exact solution of the equation (4.1) is $u(x) = vcos(x)$. We have already shown that the posterior of v, conditioned to the dataset of n_d noisy measurements $\boldsymbol{D}$, can be computed as:

$$\mathscr{P}(v|\boldsymbol{D}) \propto \left[\left(\frac{1}{\sqrt{2\pi\sigma_D^2}}\right)^{n_d} \prod_{d=1}^{n_d} \exp\left(-\frac{(u_d - vcos(x_d))^2}{2\sigma_D^2}\right)\right] \frac{1}{\sqrt{2\pi\sigma_0^2}} \exp\left(-\frac{(v_0 - v)^2}{2\sigma_0^2}\right).$$

We will now consider only the exponential for simplicity. The next steps are:

$$\begin{aligned}
\mathscr{P}(v|\boldsymbol{D}) &\propto \left[\exp\left(-\frac{1}{2\sigma_D^2}\sum_{d=1}^{n_d}(u_d - vcos(x_d))^2\right)\right] \frac{1}{\sqrt{2\pi\sigma_0^2}} \exp\left(-\frac{(v_0 - v)^2}{2\sigma_0^2}\right) \\
&\propto \exp\left(-\frac{1}{2\sigma_D^2}\left(\sum_{d=1}^{n_d} u_d^2 - 2\left(\sum_{d=1}^{n_d} u_d cos(x_d)\right)v + v^2\sum_{d=1}^{n_d} cos(x_i)^2\right) - \frac{1}{2\sigma_0^2}(v^2 - 2v_0 v + v_0^2)\right) \\
&\propto \exp(-\frac{1}{2\sigma_D^2\sigma_0^2}\{\sigma_0^2\sum_{d=1}^{n_d} u_d^2 + v_0^2\sigma_D^2 - 2\left[\left(\sum_{d=1}^{n_d} u_d cos(x_d)\right)\sigma_0^2 + v_0\sigma_D^2\right]v + \\
&\left(\sum_{d=1}^{n_d} cos(x_d)^2\sigma_0^2 + \sigma_D^2\right)v^2\}) \\
&\propto \exp\left(-\frac{1}{2\frac{\sigma_D^2\sigma_0^2}{\sum_{d=1}^{n_d}(cos(x_d)^2\sigma_0^2)+\sigma_D^2}}\left(v - \frac{\left(\sum_{d=1}^{n_d} u_d cos(x_d)\right)\sigma_0^2 + v_0\sigma_D^2}{\sum_{d=1}^{n_d} cos(x_d)^2\sigma_0^2 + \sigma_D^2}\right)^2\right).
\end{aligned}$$

Hence, we have proved that the posterior is still a Normal distribution:

$$\mathscr{P}(v|\boldsymbol{D}) \sim \mathscr{N}(v_{post}, \sigma_{post}^2),$$

of mean v_{post} and variance σ^2_{post}, defined by the following equations:

$$\begin{aligned}
v_{post} &= \frac{\sum_{d=1}^{n_d}(u_d \cos(x_d))\sigma_0^2 + v_0\sigma_D^2}{\sum_{d=1}^{n_d}(\cos^2(x_d))\sigma_0^2 + \sigma_D^2} \\
\sigma^2_{post} &= \frac{\sigma_0^2\sigma_D^2}{\sum_{d=1}^{n_d}(\cos^2(x_d))\sigma_0^2 + \sigma_D^2}.
\end{aligned}$$

Bibliography

[1] Yusuf Aytar and Andrew Zisserman. Tabula rasa: Model transfer for object category detection. *2011 International Conference on Computer Vision*, pages 2252–2259, 2011.

[2] Zhilong Fang, Curt Da Silva, Rachel Kuske, and Felix J. Herrmann. Uncertainty quantification for inverse problems with weak partial differential equation constraints. *GEOPHYSICS*, 83(6):R629–R647, 2018.

[3] Joseph B. Nagel. *Bayesian techniques for inverse uncertainty quantification*. PhD thesis, ETH Zurich, Zürich, 2017.

[4] Andrew M. Stuart. Inverse problems: A bayesian perspective. *Acta Numerica*, 19:451–559, 2010.

[5] Alfio Quarteroni, Andrea Manzoni, and Christian Vergara. The cardiovascular system: Mathematical modelling, numerical algorithms and clinical applications. *Acta Numerica*, 26:365–590, 05 2017.

[6] Finbarr O'Sullivan. A statistical perspective on ill posed problems. *Statistical Science*, 1(4):502–518, 11 1986.

[7] 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.

[8] 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.

[9] Robert Tibshirani. Noninformative priors for one parameter of many. *Biometrika*, 76(3):604–608, 09 1989.

[10] Anna R. Bruss. The eikonal equation: some results applicable to computer vision. *Journal of Mathematical Physics*, 23(5):890–896, 1982.

[11] Qi Hao and Tariq Alkhalifah. An acoustic eikonal equation for attenuating orthorhombic media. *GEOPHYSICS*, 82(4):WA67–WA81, 2017.

[12] Xingguo Huang and Stewart Greenhalgh. Linearized formulations and approximate solutions for the complex eikonal equation in orthorhombic media and applications of complex seismic traveltime. *GEOPHYSICS*, 83(3):C115–C136, 2018.

[13] Natasja M.S. de Groot, Martin J. Schalij, Katja Zeppenfeld, Nico A. Blom, Enno T. Van der Velde, and Ernst E. Van der Wall. Voltage and activation mapping. *Circulation*, 108(17):2099–2106, 2003.

[14] Henriquez Craig S. Simulating the electrical behavior of cardiac tissue using the bidomain model. *Critical reviews in biomedical engineering*, 21, 1993.

[15] Mark Potse, Bruno Dube, Jacques Richer, Alain Vinet, and Ramesh M. Gulrajani. A comparison of monodomain and bidomain reaction-diffusion models for action potential propagation in the human heart. *IEEE Transactions on Biomedical Engineering*, 53(12):2425–2435, 2006.

[16] Piero Colli Franzone, Lucia Guerri, and Sergio Rovida. Wavefront propagation in an activation model of the anisotropic cardiac tissue: asymptotic analysis and numerical simulations. *Journal of Mathematical Biology*, 28:121–176, 1990.

[17] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.

[18] Hecht-Nielsen Robert. *Theory of the backpropagation neural network*. Academic Press, 1989.

[19] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv*, 2017.

[20] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *arXiv*, 2016.

[21] Yarin Gal. *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge, 2016.

[22] Stefan Depeweg, José Miguel Hernández-Lobato, Finale Doshi-Velez, and Steffen Udluft. Decomposition of uncertainty in bayesian deep learning for efficient and risk-sensitive learning. *arXiv*, 2018.

[23] Tianping Chen and Hong Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks*, 6(4):911–917, 1995.

[24] Atilim Baydin, Barak Pearlmutter, Alexey Radul, and Jeffrey Siskind. Automatic differentiation in machine learning: A survey. *Journal of Machine Learning Research*, 18:1–43, 04 2018.

[25] Siddharth Krishna Kumar. On weight initialization in deep neural networks. *arXiv*, 2017.

[26] Michael Betancourt. A conceptual introduction to hamiltonian monte carlo. *arXiv*, 2018.

[27] Siddhartha Chib and Edward Greenberg. Understanding the metropolis-hastings algorithm. *The American Statistician*, 49(4):327–335, 1995.

[28] Robert Leonard Nelson and Hercule Kwan. "leapfrog" methods for numerical solution of differential equations, sinewave generation, and magnitude approximation. *Conference Record of The Twenty-Ninth Asilomar Conference on Signals, Systems and Computers*, 2:802–806 vol.2, 1995.

[29] Radford M. Neal. Mcmc using hamiltonian dynamics. *arXiv*, 2012.

[30] Andreas Fichtner, Andrea Zunino, and Lars Gebraad. A tutorial introduction to the hamiltonian monte carlo solution of weakly nonlinear inverse problems. *eartharxiv*, 06 2018.

[31] Matthew D. Homan and Andrew Gelman. The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo. *Journal of Machine Learning Research*, 15(1), 2014.

[32] Qiang Liu. Stein variational gradient descent as gradient flow. *arXiv*, 2017.

[33] Tim van Erven and Peter Harremos. Rényi divergence and kullback-leibler divergence. *IEEE Transactions on Information Theory*, 60(7):3797–3820, 2014.

[34] Luning Sun and Jian-Xun Wang. Physics-constrained bayesian neural network for fluid flow reconstruction with sparse and noisy data. *arXiv preprint arXiv:2001.05542*, 2020.

[35] Mohamad T. Musavi, Wahid Ahmed, Koon Ho Chan, Kathleen B. Faris, and Donald M. Hummels. On the training of radial basis function classifiers. *Neural Networks*, 5(4):595–603, 1992.

[36] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[37] Dong Yu and Michael Seltzer. Improved bottleneck features using pretrained deep neural networks. *Interspeech 2011, 12th Annual Conference of the International Speech Communication Association*, pages 237–240, 01 2011.

[38] Roger Luis, L. Enrique Sucar, and Eduardo F. Morales. Transfer learning for bayesian networks. *Advances in Artificial Intelligence – IBERAMIA 2008*, pages 93–102, 2008.

[39] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. *Proceedings of the 30th International Conference on Machine Learning*, 28(3):1139–1147, 17–19 Jun 2013.

[40] Ender Konukoglu, Jatin Relan, Ulas Cilingir, Bjoern Menze, Phani Chinchapatnam, Amir Jadidi, Hubert Cochet, Mélèze Hocini, Hervé Delingette, Pierre Jais, Michel Haïssaguerre, Nicholas Ayache, and Maxime Sermesant. Efficient probabilistic model personalization integrating uncertainty on data and parameters: Application to eikonal-diffusion models in cardiac electrophysiology. *Progress in biophysics and molecular biology*, 107:134–46, 07 2011.

[41] Sergey Fomel, Songting Luo, and Hongkai Zhao. Fast sweeping method for the factored eikonal equation. *Journal of Computational Physics*, 228(17):6440 – 6455, 2009.

[42] Malcolm C. A. White, Hongjian Fang, Nori Nakata, and Yehuda Ben-Zion. Pykonal: A python package for solving the eikonal equation in spherical and cartesian coordinates using the fast marching method. *Seismological Research Letters*.

[43] Daniel Hurtado, Francisco Costabal, Yibo Yang, Paris Perdikaris, and Ellen Kuhl. Physics-informed neural networks for cardiac activation mapping. *Frontiers of Physics*, 8, 02 2020.

[44] John K. Triedman, Kathy J. Jenkins, Steven D. Colan, J. Philip Saul, and Edward P. Walsh. High-density endocardial activation mapping of the right atrium in three dimensions by composition of multielectrode catheter recordings. *Journal of Electrocardiology*, 29:234–240, 1996.

[45] Cory M. Tschabrunn, Sebastien Roujol, Nicole C. Dorman, Reza Nezafat, Mark E. Josephson, and Elad Anter. High-resolution mapping of ventricular scar. *Circulation: Arrhythmia and Electrophysiology*, 9(6):e003841, 2016.

[46] Yeonjong Shin. On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type pdes. *Communications in Computational Physics*, 28(5):2042–2074, Jun 2020.

[47] Jorge Romero, Florentino Lupercio, David Goodman-Meza, Juan Carlos Ruiz, David F. Briceno, John D. Fisher, Jay Gross, Kevin Ferrick, Soo Kim, Luigi Di Biase, Mario J. Garcia, and Andrew Krumerman. Electroanatomic mapping systems (carto/ensite navx) vs. conventional mapping for ablation procedures in a training program. *Journal of Interventional Cardiac Electrophysiology*, 2016.

[48] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv*, 2017.

[49] Laurent Valentin Jospin, Wray Buntine, Farid Boussaid, Hamid Laga, and Mohammed Bennamoun. Hands-on bayesian neural networks – a tutorial for deep learning users. *arXiv*, 2020.

[50] Eric Mjolsness and Dennis DeCoste. Machine learning for science: State of the art and future prospects. *Science*, 293(5537):2051–2055, 2001.

[51] Yibo Yang and Paris Perdikaris. Adversarial uncertainty quantification in physics-informed neural networks. *Journal of Computational Physics*, 394:136 – 152, 2019.

[52] Luning Sun and Jian-Xun Wang. Physics-constrained bayesian neural network for fluid flow reconstruction with sparse and noisy data. *Theoretical and Applied Mechanics Letters*, 10(3):161 – 169, 2020.

[53] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient pathologies in physics-informed neural networks. *arXiv*, 2020.

[54] Vikram Mullachery, Aniruddh Khera, and Amir Husain. Bayesian neural networks. *arXiv*, 2018.