Trusted Predictive Modelling for Engineering Systems with Large Scale Data



Suraj A. Tailor

Supervisors:

Professor Andrew R. Mills
Professor Visakan Kadirkamanathan
Dr William R. Jacobs

A report submitted in partial fulfillment of the requirements
for the award of the degree of
Doctor of Philosophy

Sheffield 2026

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my parents for their unwavering support throughout my life. They have always been patient with me and have consistently encouraged my pursuit of education. Through their example, they have taught me resilience and perseverance, without which I would not have been able to undertake this PhD.

I am extremely grateful to my team of supervisors. I would first like to thank Andy. Through his mentorship, I have grown tremendously, both academically and as a person. He gave me the freedom to pursue my ideas, while always being there to challenge my thinking and offer me a broader perspectives. I hope to continue receiving his guidance far into the future.

I would also like to acknowledge Visakan, who has always been a great example of leadership throughout my PhD. His clarity of thought and breadth of knowledge during our meeting was always inspiring.

My thanks also go to Will who began supervising at an crucial juncture when the project lacked momentum. His enthusiasm gave the impetus to get the project moving again. Through our discussions, I have learned so much.

I have been fortunate to have made wonderful friends in the UTC. In particular, I would like to thank Panos, Andrew, Hadriano and Felipe. At different stages of my PhD they have supported me in the technical aspects of my work, but most importantly have made the office a joy to work in.

Finally, I would like to thank my brother, Arun, who has constantly pushed me to aim higher and strive to achieve great things.

Abstract

Recent advances in Machine Learning (ML) combined with increasing computing power, storage capacity, and sensor technology have engineering industries scrambling to take advantage of the insights from big data. Digital Twins (DTs) exemplify this shift, increasingly incorporating data-driven ML models to support decision making, yielding cost savings. However, the utility of a DT depends on the trust placed in their predictive output, which can be driven by accuracy and uncertainty quantification. Trust in DTs can be inhibited by imperfections in data arising from poor coverage, the presence of noise and unmeasured system disturbances.

Physics-informed Machine Learning (PIML) is an emerging field that seeks to address data weaknesses through the embedding of physical knowledge about a process into ML model training. Multi-Fidelity (MF) modelling is another field that seeks to optimally combine information sources with different fidelities to improve the accuracy upon estimates about a process, which may be utilised to provide a source of physics via engineering models. There is great potential for the methods contained in these fields to address key trust challenges in DTs; however, research gaps remain. Using MF modelling, within a physics-informed paradigm, overcomes the challenges of complex system representation in many PIML methodologies but has a lack of principled schemes for trading off between data of different fidelity. Furthermore, the current MF methods with uncertainty quantification have high computational cost which scale with data size.

This thesis addresses these challenges. Insights into the limitation of Physics-Informed Neural Networks (PINNs) have motivated the development of two novel Bayesian MF modelling frameworks. The frameworks enable engineers to incorporate prior engineering knowledge about the datasets in a mathematically principled manner, yielding accurate predictive model outputs with uncertainty quantification. The frameworks are implemented on probabilistic Neural Networks (NNs) and demonstrate cost effective inference and improved scalability over existing methods.

Contents

1 Introduction **1**

1.1 Aim and objectives 2

1.1.1 Aim 2

1.1.2 Objectives 2

1.2 Summary of thesis 2

1.3 Main contributions 4

2 Literature review **6**

2.1 Trust in machine learning 7

2.1.1 Transparency, explainability and interpretability 8

2.1.2 Robustness 11

2.1.3 Performance 13

2.1.4 Uncertainty quantification 14

2.2 Trust challenges in digital twins 15

2.3 Approaches for uncertainty quantification in machine learning 18

2.4 Physics-informed machine learning 21

2.4.1 Physics-informed neural networks 24

2.4.2 Considerations to using physics-informed neural networks 26

2.4.3 Applications of physics-informed neural networks 27

2.4.4 Fusion-based multi-fidelity modelling approaches 30

2.4.5 Uncertainty quantification in multi-scale modelling methods 34

2.5 Gaps in research 34

3 Identification of limitations in implementing learning biases in physics-informed neural networks 36

3.1 Background . . . 38

3.2 Problem set up: 1-D heat equation . . . 39

3.2.1 Methodology for training a physics-informed neural network on the 1-D heat equation . . . 41

3.2.2 Model architectures and learning parameters . . . 42

3.3 Evaluating physics-informed neural network performance with coefficient model mismatch and weightings . . . 43

3.4 Evaluating physics-informed neural network performance with noisy experimental data and weighting . . . 45

3.4.1 Conclusions and future work . . . 47

4 Multi-fidelity modelling: a maximum a posteriori approach 49

4.1 Bayesian problem formulation . . . 51

4.1.1 Problem set-up . . . 52

4.1.2 Variational inference theory . . . 52

4.1.3 Fitting the distributional parameters to the approximate posterior . . . 54

4.2 Methodology . . . 59

4.2.1 Hyper-parameter selection . . . 61

4.2.2 Model architecture . . . 65

4.2.3 Addressing convergence issues during training . . . 66

4.3 Results and discussion . . . 67

4.3.1 Simple mismatch example . . . 67

4.3.2 Linear and non-linear mismatch examples . . . 70

4.3.3 Steady-state AGTF30 model example . . . 78

4.4 Conclusions and future work . . . 82

5 Addressing convergence issues in uncertainty estimation using mean variance estimation networks 85

5.1 Background . . . 87

5.1.1 Learning rate annealing algorithm for solving suboptimal convergence when training physics-informed neural networks . . . 88
5.2 Annealing μ algorithm . . . 92
5.3 Simple sinusoidal homoscedastic noise example . . . 94
5.3.1 Adaptations to the annealing μ algorithm . . . 101
5.3.2 Repeat experiments on the homoscedastic noise example . . . 108
5.4 Simple sinusoidal heteroscedastic noise example . . . 110
5.4.1 Repeat experiments on the heteroscedastic noise example . . . 113
5.4.2 Effect of applying weight decay to mean variance estimation networks . . . 115
5.5 Computational cost of algorithms . . . 122
5.6 Conclusions and future work . . . 123

6 Multi-fidelity modelling: a maximum a posteriori and kernel-based approach 125
6.1 Motivation and conceptual overview . . . 127
6.2 Bayesian formulation . . . 130
6.2.1 Assumptions on random variables . . . 131
6.2.2 Deriving the approximate posterior equation . . . 131
6.2.3 Specification of hyper-parameter forms in the approximate posterior . . . 132
6.2.4 Updated analytical expressions . . . 136
6.3 Outline of modelling process . . . 137
6.4 Methodology . . . 138
6.4.1 Choice of models . . . 139
6.5 1-D sinusoidal example: demonstrating the effects of hyper-parameter choices . . . 142
6.6 2-D example: demonstrating effective hyper-parameter tuning . . . 144
6.6.1 Fitting Σ . . . 145
6.6.2 Fitting τ and η . . . 147
6.6.3 Results and discussion . . . 148
6.7 Comparisons to benchmark approaches . . . 151
6.7.1 Deep multi-fidelity Gaussian Process . . . 152
6.7.2 Non-linear information fusion algorithms . . . 152

6.7.3 Fidelity weighted learning . . . 154
6.7.4 Training, test data and model parameters . . . 155
6.7.5 Results . . . 156
6.7.6 Comparing computational cost at training and inference . . . 160
6.8 Conclusions and future work . . . 161

7 Conclusions and future work **164**
7.1 Aim and objectives . . . 164
7.1.1 Aim . . . 164
7.1.2 Objectives . . . 164
7.2 Conclusions . . . 165
7.3 Future work . . . 167
7.4 Final remarks . . . 168

A Expanded derivation of Equation (4.5) **169**

B Training details, hyper-parameter choices and model architectures **171**

C Annealing Y_μ and Y_{σ^2} (I) algorithm for solving convergence issues **174**
REFERENCES . . . 176

Acronyms

AD Automatic Differentiation.

AGTF30 Advanced Geared Turbofan 30000.

AI Artificial Intelligence.

AMISE Asymptotic Mean Integrated Squared Error.

ARD Automatic Relevance Determination.

BC Boundary Condition.

BCV Biased Cross Validation.

BNN Bayesian Neural Network.

CAVI Coordinate Ascent Variational Inference.

CT Clough-Tocher.

DCE Data Centric Engineering.

DL Deep Learning.

DPI Direct Plug In.

DT Digital Twin.

ELBO Evidence Lower Bound.

eLU Exponential Linear Unit.

ERM Empirical Risk Minimisation.

EU European Union.

FCN Fully Connected Network.

GP Gaussian Process.

GPU Graphics Processing Unit.

GTE Gas Turbine Engine.

Hi-Fi High-Fidelity.

HM Health Monitoring.

HMC Hamiltoninian Monte Carlo.

HPC High Pressure Compressor.

IC Initial Condition.

IG Inverse Gamma.

JRC Joint Research Centre.

KDE Kernel Density Estimator.

KL Kullback-Leibler.

LL Log-Likelihood.

Lo-Fi Low-Fidelity.

LSCV Least Squares Cross Validation.

MAE Mean Absolute Error.

MAP Maximum a Posteriori.

MCMC Markov Chain Monte Carlo.

MF Multi-Fidelity.

MFVI Mean-Field Variational Inference.

MISE Mean Integrated Squared Error.

ML Machine Learning.

MRI Magnetic Resonance Imaging.

MSE Mean Squared Error.

MVE Mean Variance Estimation.

NIG Normal-Inverse Gamma.

NLL Negative Log-Likelihood.

NN Neural Network.

NPN Natural Posterior Network.

ODE Ordinary Differential Equation.

OECD Organisation for Economic Co-operation and Development.

PDE Partial Differential Equation.

PDF Probability Density Function.

PGNN Physics-Guided Neural Network.

PIML Physics-Informed Machine Learning.

PINN Physics-Informed Neural Network.

PLA Power Lever Angle.

PR Pressure Ratio.

RBF Radial Basis Function.

ReLU Rectified Linear Unit.

RMSE Root Mean Square Error.

RNN Recurrent Neural Network.

SELU Scaled Exponential Linear Unit.

SWAG Stochastic Weight Averaging Gaussian.

VI Variational Inference.

WD Weight Decay.

Chapter 1

Introduction

Digital Twins (DTs) are designed to mirror a physical asset throughout its life. They enable engineers to make decisions about an asset in real time, thus yielding cost-savings. DTs are frequently updated given sensor data and provide engineers with observed and latent information about an asset. For example, in the aerospace industry, DTs of aircraft can inform maintenance schedules to only performing repairs when required, because latent health and performance become visible. DTs can include Machine Learning (ML) models that have a strong capacity to learn patterns from large datasets. This thesis addresses the challenges in producing trusted predictive ML models of the system when incorporating knowledge sources from different physical domains and of differing fidelities. Thermal, vibration and electrical data are examples of such domains, which are traditionally siloed specialisms within industry with their own methods and language for analysis. The fidelity of a dataset relates to the accuracy with which it represents the current system.

Trust in ML relates broadly to the hesitancy in the adoption of ML in society due to stakeholders not fully understanding how it works or having confidence in its prediction. Algorithms in this PhD have been developed considering a trust framework defined in Chapter 2.1. The approach taken in this thesis was to build predictive models considering data from many data sources (differing in domain and fidelity), enabling a better understanding of engineering system behaviour and improving trust in a model. These models are built to accommodate the realities of imperfect training datasets, such as sparse data or data with outliers or imperfections within. They can utilise both known physics and complex behaviours (e.g., emergent from domain coupling) captured only in real system data. DT applications are used as the case study in this thesis to contextualise and

validate the methods in this thesis. The aims and objectives are given below.

1.1 Aim and objectives

1.1.1 Aim

1. Develop scalable modelling frameworks that produce trusted outputs that accommodate sparse data obtained from real engineering systems. Models produced using this framework will provide trust through uncertainty quantification, interpretability, and high-performance.

1.1.2 Objectives

1. Evaluate and select candidate modelling frameworks for further development, which can accommodate sparse data obtained from real engineering systems.

2. Development of scalable modelling frameworks that leverage multiple information sources using a principled and mathematically grounded approach.

3. Ensure that the models developed are trustworthy through uncertainty quantification of their outputs, having interpretability, and being high-performing.

1.2 Summary of thesis

A summary of the thesis chapters is given below.

Ch.2 Background and literature review: This chapter began by introducing the concept of trust in ML and outlining the trust challenges in DTs that were addressed in this thesis. Three trust challenges were the focus associated with the building of predictive models for DTs applications. First, the mitigation of model mismatch. Second, the incorporation of uncertainty quantification. Third, the building of high-performing models in sparse data regimes. A high level review of ML methods for uncertainty quantification and Physics-Informed Machine Learning (PIML) was then presented. This was followed by a more detailed review of

Physics-informed Neural Networks (PINNs) and MF modelling methods, situating the methods contained in this field within the broader trust paradigm. Finally, the research gaps to be addressed in this thesis were outlined.

Ch.3 Identification of limitations in implementing learning biases in physics informed neural networks: This chapter illustrated the detrimental effect of model mismatch when training PINNs. PINNs were trained to solve the forward 1-D heat equation problem for varying levels of mismatch and noise applied to observation data, providing data analogous to modelling assumption violation and simplification along with real-world sensor imperfections. A mitigation was then explored in the form of applying weightings on the physics-loss term. The results demonstrated some promise in the weighting scheme and was shown to improve the performance for low levels of model mismatch when less weighting on the physics loss was applied. Despite promise, the results, together with a deeper understanding of the requirements of the PINNs methodology, motivated a pivot towards developing MF modelling frameworks as a means of addressing the trust challenges in DTs identified in Chapter 2. This chapter served as a preliminary work that is built upon in Chapters 4 and 5 of this thesis.

Ch.4 Multi-fidelity modelling with mismatch uncertainty: a maximum a posteriori approach: The re-weighting of loss terms was shown to be a promising avenue for addressing the model mismatch problem in Chapter 3. In this chapter the re-weighting idea was extended into the development of novel MF modelling methods which incorporated weighting schemes and had the capacity to implement richer physics-based models into the methodology. A novel Bayesian framework was proposed that enables engineers to incorporate prior beliefs about model mismatch when performing MF modelling. The framework was implemented using surrogate Mean Variance Estimation (MVE) Neural Networks (NNs), enabling scalability to large datasets in terms of dimension and volume. Across synthetic and real-world problems, the method outperformed baseline models trained solely on High-Fidelity (Hi-Fi) data.

Ch.5 Addressing convergence issues in uncertainty estimation using mean variance estimation networks: In Chapter 4, convergence issues were apparent when using the bespoke loss functions for training Neural Networks (NNs). The loss functions were variations of the Gaussian Negative Log-Likelihood (NLL) which is known to have convergence issues

even for simple low dimensional and low data problems. Despite using a state of the art method, convergence issues were still apparent, which inspired the development of original methods to address this problem in this chapter. Further analysis was conducted on this issue corroborating insights found in the literature. The new methods were tested on simple sinusoidal heteroscedastic and homoscedastic noise examples and compared against the state of the art. Recommendations were given to the choice of algorithm to use given the problem and circumstance.

Ch.6 Multi-fidelity modelling: a maximum a posteriori and kernel based approach: The MF modelling approach developed in Chapter 4 had limitations in being able to place modelling assumptions in transitory regions between regions of data and sparsity in the domain. In this chapter a novel MF modelling approach was developed which addressed that limitation and others, building on the work conducted in Chapter 4. The approach reframed the interpretation of the sample size in the posterior equation in Chapter 4, to one that was based upon the density of points across the domain, evaluated through a function of a Kernel Density Estimate (KDE) on the Hi-Fi data. With this re-framing, modelling assumptions could be placed in transitory regions via a domain knowledge informed choice of hyper-parameters. A Bayesian problem formulation was outlined. Then, a process for modelling was outlined. A synthetic 1-D example was used to showcase the effect of varying hyper-parameters of the approach. Then, a method for hyper-parameter tuning was demonstrated on a synthetic 2-D example. Finally, approach was compared to baseline approaches evaluated on the synthetic 2-D example. Comparisons were made in terms of performance on a test set in addition to the computational complexity during training and inference.

1.3 Main contributions

1. A critical analysis of PINNs under model mismatch, showing when PINNs fails to be trustworthy given inaccurate or incomplete governing physics equations used in the physics loss function.

2. A novel Bayesian MF modelling framework implemented using MAP learning. The method

enables prior beliefs about model mismatch to be placed when making predictions. The framework avoids traditional GP-based MF modelling methods, which are computationally infeasible for the volumes of data the target applications in this thesis would consider.

3. Novel gradient annealing algorithms capable of solving issues of suboptimal convergence when using Gaussian NLL loss function. This enables consistent uncertainty quantification when using probabilistic NNs.

4. A novel kernel-based Bayesian MF modelling framework implemented using MAP learning. This provides uncertainty quantification in sparse regions where NNs typically fail.

Chapter 2

Literature review

The engineering industry has adopted the practice of capturing and storing large amounts of data through networks of sensors from their deployed assets. In part, this is a means to monitor large and expensive assets in an industry. In creating Digital Twins (DTs), this motivates the choice of using Deep Learning (DL) over classical Machine Learning (ML) methods to model aspects of a system where there is a high volume of data. It has been shown that in large-scale data regimes, DL methods are the superior choice over classical ML methods. This is in part due to their capacity to capture complex relationships in large datasets [1]. The fundamental theory behind Neural Networks (NNs) was developed in 1943 by Warren McCullough and Walter Pitts [2], however, the world has only recently been able to take advantage of methods using deep NNs within the last decade [3]. This is for a number of reasons including the recent large scale availability of GPUs and their increasing performance enabling accessibility to DL research [4]. Another reason is due to theoretical advances in DL such as improved optimisation methods, regularisation strategies and weight initialisation methods [3]. And finally, the release of open source libraries that take advantage of the aforementioned increase in computing resources further enabling accessibility to DL research and its implementation. Therefore, only in the past decade has there been rapid development and interest in DL for modelling engineering systems. Despite these hardware developments, trust in these models is an obstruction to their deployment, especially for high-impact decision making. One such obstruction is the high dimensionality of NNs, which often have millions of parameters. It is difficult to understand how these individual parameters work together to arrive at a final output or, in some cases, what a particular parameter does in a NN. This is contributed by

the inherent function flexibility making smoothness difficult to guarantee and sensitivity difficult to predict. Another example is the lack of reliable uncertainty quantification, which is essential to inform critical engineering decisions. These examples can give some insight into why engineers may be reluctant to trust NNs.

2.1 Trust in machine learning

ML models have now been shown to equal or exceed human-level performance in an increasing number of human-associated tasks. For example, NNs can now detect abnormalities in clinical imaging more accurately than medical professionals [5]. However, there is a caveat to this success; now there is an expectation of having trust in these models, where trust can be defined by having confidence in something to perform a task or believing something is competent to perform a task [6]. Therefore, trust in ML means having confidence in the outputs of an ML model used for a decision making process. The anthropomorphisation of ML models, which has been stimulated by their capacity to perform human-associated tasks, has contributed to society having the same level of trust in these algorithms as they would do to humans. For example, in trust between humans, it is typical for the person receiving the trust to be motivated by having good-will towards the trustor. There would also be additional constraints on how a person who receives trust behaves according to societal expectations of morality [7]. Clearly, ML models cannot act on these additional human-like principles, which results in accountability on the developers of ML models to ensure stakeholders have trust in their models before their deployment. Otherwise, without trust in ML models, society may be hesitant in their adoption for high-impact decision making. To contextualise this responsibility, it would be up to the developers of ML models used in autonomous vehicles to ensure that the passengers inside the vehicle are safely navigated to their destination, since ML models are incapable of being motivated by compassion or empathy. This results in the necessity of having trust in the usage of ML models for high risk decision making rather than being an afterthought. In practice, creating trusted ML models includes making them transparent, explainable, interpretable, robust, and high-performing [8][9]. Uncertainty quantification can also be considered an aspect of the trust paradigm [10], although sometimes placed as a subdomain of explainability [11]. Definitions have been formulated for each of the aspects of trusted ML models

in the following sections.

2.1.1 Transparency, explainability and interpretability

Transparency, explainability, and interpretability vary in their definitions throughout the literature and in some cases are used interchangeably without nuance [12]. This section attempts to provide definitions for each of these terms with a view to build upon methods to evaluate a model's transparency, explainability and interpretability, or create methods which increase these qualities in models within future work. To formulate these definitions, literature has been sourced from: Artificial Intelligence (AI) policy within the UK and Europe; review papers on explainable AI; and relevant and highly cited papers, which have attempted to define these words. This is not a critical review or survey of how these terms are defined across the literature, but an attempt to provide a basis for definitions from which future work on developing methods can be conducted.

Transparency

Based on publications from several centers of AI policy, The Royal Holloway Society stated in their policy briefing [9] that transparency implies some level of accessibility to the data or algorithm. The EU High Level Expert Group on AI's policy briefing [13] largely agreed with this definition and added two points. Firstly, the requirement of an explanation of the systems that are applicable to the stakeholders involved. Secondly, it is important to bring awareness to humans that they are interacting with an AI system and inform them of the systems' capabilities and limitations. These aspects of transparency (awareness, accessibility, and providing explanations) were shared by OECD in their AI principles [14]. However, they caveated the level of accessibility afforded, stating that it is not required to share out source code and proprietary datasets to all stakeholders of AI systems. The JRC's policy briefing on AI [15] stated that transparency includes aspects of explainability and interpretability, however, they also stated that transparency relates to documenting how an AI is developed, including the technical understanding of the model, and noting the data used to train it.

Outside of AI policy, the definitions of transparency were broadly consistent with each other. Briggs et al. [16] defined transparency as "describing the model structure, equations, parameters

values and assumptions to allow interested parties to understand the model.“ Tomsett et al. [17] stated it as the level to which a system provides information about its internal workings or structure. This alludes to the fact there are differing levels of transparency a system may have which is consistent with how the literature generally describes ML models as black, grey or white boxes depending on how much is understood about a model. Taking into account the literature, the definition of transparency is formalised below.

Definition 1. (Transparency): The accessibility and awareness stakeholders have to an AI system. Where accessibility includes the extent at which components of a system - such as the model structure, equations, parameter values, and assumptions - are explainable and interpretable. In addition, there may be varying levels of accessibility available to different stakeholders.

Explainability

Starting with the publications from AI policy makers, the EU High Level Expert Group on AI's policy briefing [13] describes explainability as the ability to explain the technical process of an AI system in addition to related human decisions. Furthermore, the explanation of the technical details of the system includes the ability to trace the decisions made by the system. In addition, they state that trade-offs may have to be made to make systems more explainable at the expense of performance, or vice versa. They recommend that explanations should be timely and bespoke to the stakeholder concerned. Finally, they suggest that explanations about how AI systems shape and influence organisational decision making should be provided, including the rationale for its deployment. The OECD's AI Principles [14] shared the same sentiment of explanations being curated according to users and those affected by the system. They also suggested that there is a trade-off between explainability and performance. The Royal Holloway Society [9] was more subtle in their definition of explainable, stating that a wide range of users should be able to understand why or how a conclusion was reached when using an AI system.

Outside of AI policy, an overview paper on explainable AI published in Science Robotics by Gunning et al. [18] described an explainable AI system as one that has the ability to describe what is capable of including what is done, what it is currently doing, and what it will do next. In addition, it should be able to explain the information it is acting on. They go on to state that explanations depend on the task, abilities, and expectations of the user of the AI system. A review paper on

explainable AI by Vilone and Longo [19] alluded to cross-literature definitions of explainability. They converged on the idea that explainability is the extent to which users of an AI system can understand a decision or prediction made by it. Taking into account the literature, the definition of explainability is formalised below.

Definition 2. (Explainability): The level to which a model can explain specific outcomes or predictions of a model for a given input to stakeholders of an AI model.

Interpretability

Starting with publications from AI policy makers, the Royal Holloway Society [9] stated that an interpretable model implies understanding how the model works. The JRC policy briefing [15] stated that interpretability is related to understanding the internals of a model. This includes the principles of how data are processed and the reasons for an output. They went further to state that methods to make a model interpretable have two main aspects. First, there needs to be logic to the model; for example, what types of data go into the model and how are the data processed to produce an output. Second, there needs to be a description of the type of input that will be used in the model. In contrast, the EU High Level Expert Group on AI [13] and OECD [14] did not define interpretability in their policy briefings.

The review paper by Vilone and Longo [19] stated that interpretability is the ability to present the meaning of an abstract concept. They also stated that an interpretation is a mapping of an abstract concept, such as a predicted class, into a domain that users can understand. Gilpin et al. [20] stated that the goal of interpretability is to describe the internals of a system in a way that is understandable to humans. They organise the interpretability of a model as an evaluation of an explanation, where explanations include an explanation of the processing of data and an explanation of the representation of data inside a network. Rudin et al. [21], in their survey paper 'Interpretable machine learning: Fundamental principles and 10 grand challenges', stated that interpretable models obey a domain specific set of constraints to allow them to be more easily understood by humans. These constraints will differ depending on the domain. Taking the literature into consideration, a definition for interpretability is formalised below.

Definition 3. (Interpretability): The level of understanding of the underlying mechanisms of a

model that the stakeholders of an ML model have. In particular, the goal of interpretabilty is to bring meaning from the abstract concepts of the model. This includes understanding the cause and effect relationships between arbitrary inputs and outputs; how data are processed when making predictions and during training; knowing the bounds of the model, and a specification of the type of data that is going to be used including the limits of the domain. A model which obeys by a domain-specific set of constraints is therefore interpretable.

2.1.2 Robustness

Robustness refers to the ability of an ML model to resist change in the output when transformations are made to an input. These transformations can be in the form of additive random noise, structured transformations, or adversarial perturbations. Additive noise are random perturbations that are added to an input via the environment or the measurement device itself. Structured transformations are data augmentations of inputs such as re-scaling, rotations, or stretching. Finally, adversarial perturbations are malicious attempts at perturbing an input by an adversary such that the output of a model changes drastically. If a model is not robust, then it means it is vulnerable to outputs which are distant from the ground truth. This section will provide general definitions for properties of robustness in classification models. For completeness, robustness is included in the literature review although not addressed in this thesis. The most common definition of robustness is local robustness defined below [22]:

Definition 4. (Local Robustness): Given $f : \mathbb{R}^n \mapsto \mathbb{R}^m$ with m classes and $F(x) = \arg\max_i f_i(x)$ is a function representing the predictions of f. A model, F is ε-locally-robust at point, x, with respect to norm, $\|.\|$, if $\forall x'$,

$$\|x - x'\| \leq \varepsilon \Rightarrow F(x) = F(x') \tag{2.1}$$

where epsilon $\varepsilon > 0$ is specified by the user and x' is a perturbed point.

Satisfying local robustness guarantees robustness for specified inputs. A common approach to creating robust models is based on satisfying local robustness to a large number of input points in the domain. A flaw in this approach is that it is impossible for two inputs that have overlapping ε regions with differing classifications to be ε-locally robust. To overcome this, Leino et al. [22] have formulated a globally robust perspective. Their formulation constrains a model such that any

two arbitrary points in the input domain within a specified distance $\varepsilon > 0$ of each other have the same classification as one another or are labeled as non-globally robust, and prediction is not given for this input. Their definition of global robustness is shown below [22]:

Definition 5. (Global Robustness): A model, F , is ε-globally-robust, with respect to norm, $\|.\|$, if $\forall x_1, x_2$,

$$\|x_1 - x_2\| \leq \varepsilon \Rightarrow F(x_1) \stackrel{\perp}{=} F(x_2) \tag{2.2}$$

where the relation $(\stackrel{\perp}{=})$ means that $c_1 \stackrel{\perp}{=} c_2$ if $c_1 = \perp$ or $c_2 = \perp$ or $c_1 = c_2$. The $\perp$ symbol is an additional class that is introduced to indicate points that are not locally robust. In other words, not having a locally robust point means that an $\varepsilon > 0$ ball cannot be placed around the point where all the inputs in that ball have the same classification when input into the model F.

When this global robustness property is satisfied, a margin is created between inputs within an input space of width of at least $\varepsilon > 0$ where a model will abstain from making a prediction. This is equivalent to only returning predictions when $\varepsilon/2$-local robustness can be certified for all inputs in the input space. Leino et al. [22] have used this definition to create globally robust neural networks. In the creation of a globally robust NN, there is a trade off between robustness of a model and its performance - if you consider that there are regions that the model cannot formulate a prediction.

Madry et al. [23] formulated an optimisation view of robustness through an objective for training models such that when global optimum is found, the model is as locally-robust as can be for a defined set of inputs. This is in the form of a saddle point equation dependent on two things. Firstly, knowledge of the method by which perturbations are generated, for example via random noise from a known distribution or an adversarial example inputted into the model. Secondly, knowledge of the manipulative power of the method of perturbation. In order to formulate and understand this saddle point equation, consider a standard classification task with an underlying data distribution $\mathcal{D}$ over pairs of examples $x \in \mathbb{R}^d$ and the corresponding labels $y \in [k]$. Assume a suitable loss function $L(\theta, x, y)$, for instance, cross entropy loss for a neural network, where $\theta \in \mathbb{R}^p$ is the set of model parameters. Empirical Risk Minimisation (ERM), is the approach taken to train a model, that is, $\mathbb{E}_{(x,y)\sim\mathcal{D}}[L(\theta, x+\delta, y)]$. Madry et al. [23] defines the saddle point problem by Equation (2.3) below.

$$\min_{\theta} \rho(\theta), \qquad \text{where} \qquad \rho(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}}[\max_{\delta\in\mathcal{S}} L(\theta, x+\delta, y)] \tag{2.3}$$

where for each data point x, there is a set of allow perturbations $\mathcal{S} \subseteq \mathbb{R}^d$ that formalises the manipulative power of method of perturbation. The inner maximisation problem of Equation (2.3) attempts to find the worst set of perturbations on inputs that give a high loss. The outer minimisation problem finds the parameters of the models that obtain vanishing risk. If global maximums and minimums are found for Equation (2.3), this guarantees that a classifier is as robust as possible to a method of perturbations and its specified manipulative strength. This guarantee is dependent on whether the global maximums and minimums can be found. If Equation (2.3) can be solved, it does not mean that the model will be of high performance.

This section has provided definitions and properties of robustness that can be applied to any type of perturbation.

2.1.3 Performance

Performance, in the context of this thesis, refers to the metrics used to compare the outputs of classification and regression models to the ground truth at test time. High-performing models increase trust in the outputs of a model as they directly demonstrate that a model is capable of performing its intended task. Classification models are evaluated on widely used metrics including accuracy, precision, recall, specificity, and the F1 Score. Regression models typically measure performance through error metrics; these include the Mean Square Error (MSE), Mean Absolute Error (MAE), and Root Mean Square Error (RMSE). Whilst trying to make models transparent, explainable, interpretable, and robust, it is vital that performance is not largely compromised as a result. Trade-offs are typically made between the interpretability of a model and its performance when selecting a type of ML model used to fit a dataset. For example, decision trees are interpretable as there is a clear visual representation to show at how decisions are arrived, whereas DL models are known to be less interpretable due to their highly non-linear nature and large number of parameters. Performance also includes the capacity of a model to provide uncertainty to its outputs. For example, Bayesian Neural Networks (BNNs), which is a Bayesian treatment of NNs, are able to provide uncertainty in their outputs. The Negative Log-Likelihood (NLL) or Log-Likelihood (LL) is a common metric for evaluating models with uncertainty.

2.1.4 Uncertainty quantification

Accurate uncertainty quantification is imperative in predictive models, particularly when used to support safety-critical decision making, such as in DTs. In ML literature, two main types of uncertainty are considered: aleatoric and epistemic uncertainty [24].

Aleatoric uncertainty arises from random, unpredictable effects that occur in a physical system. It cannot be reduced from the acquisition of more data, it can only be modelled or quantified. Epistemic uncertainty, on the contrary, is due to a lack of knowledge about the system. This uncertainty can be reduced by acquiring more information.

In the context of engineering, aleatoric uncertainty is the amount of sensor noise or inherent external disturbances that is observed on a predictive output of a model. Epistemic uncertainty reflects the model's limited knowledge of the underlying physical process. Malinin et al. [25] further breaks down epistemic uncertainty into model uncertainty and distributional uncertainty. Model uncertainty measures the uncertainty in estimating model parameters within given data, reflecting how well the model matches the data. Distributional uncertainty arises when there is a mismatch between the training and test distributions. However, Malinin et al. [25] attempt to separately quantify these types of epistemic uncertainty via their unique prior network approach, whereas often these quantities are lumped into a single estimate of epistemic uncertainty.

Epistemic uncertainty is therefore reducible with more data, whereas aleatoric uncertainty is irreducible. Concise definitions for both types of uncertainty are given below.

Definition 6. (Aleatoric Uncertainty): Uncertainty that captures the intrinsic randomness or variability of a process.

Definition 7. (Epistemic Uncertainty) Uncertainty that arises because of the lack of knowledge about a process.

Having an accurate measure of aleatoric uncertainty gives insight into the level of disturbance or variability present in a system for a given input. This information is valuable for determining the expected system behaviour, particularly in condition monitoring. If a model deviates from its mean estimate, the aleatoric uncertainty offers engineers a basis for judging whether such a deviation is anomalous. An accurate estimate of epistemic uncertainty can give an indication if distributional shift has occurred in a system, for example due to degradation or changes in operational patterns.

It also reflects a model's capacity to capture underlying data patterns, giving insight when a model is insufficient and a better modelling choice is required.

Having outlined the trust paradigm, an overview of the trust challenges in DTs are outlined before a deeper literature review is conducted on the existing ML methods that work towards addressing these gaps.

2.2 Trust challenges in digital twins

With the increase in data availability and advances in ML methods, industries are increasingly interested in building DTs of their assets or physical systems. As described by Jones et al. [26], a DT consists of three components: a virtual representation, a physical system, and the bi-directional data flow between the virtual representation and the physical system. A virtual representation can consist of any number of sub-processes that enable specific virtual operations including modelling, testing, and optimisation. DTs therefore enable real-time data-driven decision making of their physical systems. Models in a virtual representation can include physics-based models, whose model parameters may be adjusted via model calibration after obtaining data from the physical system; or surrogate models which are data-driven. The latter includes ML models, which can be trained from measurement data and may be augmented with data generated from physics-based models.

One utility of a DT is for the Health Monitoring (HM) of an asset [27, 28]. Effective HM enables efficient maintenance schedules of an asset to be designed, decreasing the amount of unplanned downtime and optimising an asset's utilisation, ultimately resulting in cost savings. Recently, there has been an increase in ML models used for HM applications such as online novelty detection [29] or fault detection [30]. Highly accurate predictive models are a necessity for these applications of HM where subtle changes from normal operating behaviour need to be detected. A major drawback to ML data-driven methods is the requirement of swathes of historical measurement data throughout the entire domain of operation. In Gas Turbine Engines (GTEs), for example, this poses a challenge where there are space, weight, and cost constraints [31]. GTEs have limited space to house sensors beyond the critical sensor components required for smooth engine operation, and it is practically very difficult to place sensors in certain parts of the engine itself. Furthermore, there is hesitancy in

placing additional components onto an engine which increases the weight and, therefore, increases the fuel consumption, which results in greater expense. ML models are also limited in domains where there is data sparsity, and in HM this is prevalent since assets are rarely run to failure. Data sparsity can exist in both the input and output spaces; for instance, an asset can have varying outputs for the same input over its lifespan due to diverse ageing and degradation effects. For example, an aircraft will typically have quite varied operating regimes. The weather, climate and flight path can vary for a flight depending on where it is travelling and time of the year. This would drastically change the operating parameters of an engine to account for these changes and, therefore, result in sparsity of the data.

Physics-based models offer a solution to data sparsity by enabling predictions across the entire operating domain. However, physics-based models will suffer from a degree of model mismatch, which is the difference between the physical system and a model's output, in conjunction with how inaccurate or incomplete the idealised assumptions about the mathematical equations, or its parameters, used to describe the physical system are [32]. The capacity to make decisions based on a DT, is limited by the confidence we have in the accuracy of its inference. However, model inference performed in areas of the domain where there is data sparsity leaves engineers with no choice but to rely on physics-based models. In such cases, mitigating the effects of model mismatch is imperative, and one solution is Multi-Fidelity (MF) modelling. MF modelling has the capacity to handle data sparsity and sources of error that exist within different fidelity datasets, including model mismatch error or measurement errors caused by external disturbances, distributional drift, or sensor noise. In Chapters 4 and 6, unique MF modelling methods are developed that are capable of mitigating these challenges via an additive learned correction to a Lo-Fi prediction.

Developing trust in DTs is a challenge, especially when incorporating data-driven models, which are often black-boxes which would make stakeholders reluctant to make decisions about expensive assets. Considering the focus of this thesis (i.e. transparency, interpretability, performance and uncertainty), the incorporation of Physics Informed Machine Learning (PIML) models in DTs can naturally lend itself to increased transparency, interpretability and performance as outlined in Chapter 2.4 assuming accurate physics assumptions about the system. An implementation of accurate epistemic uncertainty quantification in data-driven models in a DT increases trust in two ways. Firstly, by informing engineers of distributional drift and when predictions are being made

away from the training data. Secondly, by giving an indication of a models' capacity and whether a better choice of model is required to capture the data. Similarly, accurate aleatoric uncertainty quantification can increase trust. For example, it can aid in tasks in HM by understanding the normal operating regime of an asset, enabling for a better informed decision whether an asset needs to be taken out of operation for service, which can be costly. The MF modelling methods developed in Chapters 4 and 6 have the capacity to capture the aleatoric uncertainty inherent in the Hi-Fi data used to train the model, but also the mismatch uncertainty. The mismatch uncertainty quantifies the confidence in a model's predictive accuracy relative to the true system, based on the contributions of different fidelity datasets. The level of uncertainty depends on the degree of model mismatch present in each dataset. This uncertainty measure considers how engineers may have intuitions or prior knowledge about the model mismatch between different fidelity datasets due to a degree of understanding about both the physical systems process and the Lo-Fi model. A definition of model mismatch is related to the outputs of the processes used to generate the different fidelity datasets and can be defined as

Definition 8. (Model Mismatch): The systematic discrepancy between the outputs of two datasets of different fidelity that arises from structural differences in the underlying processes used to produce them.

Quantifying model mismatch directly is difficult, since the true generating processes of each fidelity are unknown or partially understood, however, given paired observations between the two fidelities, a simple empirical metric can be constructed from the residuals between datasets.

Definition 9. Given a Lo-Fi dataset $\{x_{L_i}, y_{L_i}\} \in \mathcal{D}_{\mathrm{L}}$ and a Hi-Fi dataset $\{x_{H_i}, y_{H_i}\}\mathcal{D}_{\mathrm{H}}$ where for all i, $x_{L_i} = x_{H_i}$ and $|\mathcal{D}_L| = |\mathcal{D}_H| = n$, the model mismatch can be defined as $\sigma^2_{\text{mismatch}} = \frac{1}{n}\sum_i^n (y_{H_i} - y_{L_i})^2$.

A limitation of Definition 9 is that it does not isolate model mismatch from aleatoric noise if it's present within the datasets. Furthermore, it requires a Hi-Fi and Lo-Fi output pair sampled at the same input x_i for every i. Additionally, its accuracy is limited to where there is overlap in the Hi-Fi and Lo-Fi domains, hence it may not be an accurate reflection in sparse regimes. In this thesis, this metric is not used; however, it provides an example of how an engineer might quantify the model mismatch in a multi-fidelity modelling approach.

Having outlined the trust challenges in DTs that will be addressed in this thesis, in addition to signposting towards later chapters that will work to address these challenges, a review of the existing methods that work to address these challenges is outlined in the next two sections. In the following section, a high-level review of methods for uncertainty quantification in ML is conducted. Then, in the subsequent section, a high-level review of PIML is conducted before conducting deeper reviews of Physics-Informed Neural Networks (PINNs) and their variants and 'fusion-based' MF modelling methods.

2.3 Approaches for uncertainty quantification in machine learning

Approaches for aleatoric uncertainty quantification

MVE networks [33] are a seminal NN based method to quantify aleatoric uncertainty in data. This approach learns a mean value, μ and variance, σ^2, which are fitted using a Gaussian NLL loss function. Adaptations of this method exist to fitting the parameters of other types of univariate distribution, such as a student t-distribution [34] or Poisson distributions. Powerful probabilistic time series regression models, including DeepAR [35] and Temporal Convolutional Networks (TCN) [36] rely on this method of fitting probability distributions at each time step of a forecast. Extensions to multivariate distributions are also possible; see, for example, Russell et al. [37] fit parameters of multivariate Gaussian distributions. Salinas et al. [38] further developed a Gaussian copula process method with a low-rank covariance structure to reduce computational complexity and handle non-Gaussian marginal distributions. This approach skirts fitting the high-dimensional covariance matrices typically found in the multi-variate Gaussian distribution. Similar to DeepAR, the Gaussian copula process outputs can be used in time-series forecasting.

Suboptimal convergence issues using the Gaussian NLL

The direct use of the Gaussian NLL loss function to train probabilistic NNs, such as an MVE network, can cause suboptimal convergence of the mean prediction. Seitzer et al. [39] highlight the predictive variance term in the loss function as the primary culprit for this issue. The predictive

variance applies a scaling on the gradient of the mean prediction that is inversely proportional to the magnitude of the predictive variance. This can make the aforementioned approaches for aleatoric uncertainty quantification untrustworthy if not trained effectively. Mitigations for this issue exist. Nix and Wiegend [33] suggested a warm-up in which the predictive mean is trained exclusively on the data before both predictive outputs are trained simultaneously. One drawback is the requirement for an additional training step. Furthermore, the inverse variance weighting issue can still occur when both predictive outputs are trained simultaneously, where the model can still get stuck in local minima. Seitzer et al. [39] developed the $\beta-$NLL approach that mitigated this issue by multiplying the loss function by a power ($\beta \in [0,1]$) of the predictive variance. For β values close to 1, this demonstrated superior performance when evaluated in regression, classification, and generative modelling examples compared to baseline and training using a student-t distribution. Whilst effective, the approach requires a choice of β which can affect how well isolated or outlier data points are captured by the loss. Hence, it may require hyper-parameter tuning which may involve training multiple models and therefore be costly. Sluijterman et al. [40] made a series of recommendations to address this problem. This included the use of the warm-up method, gradient clipping, and the use of the $\beta-$NLL loss [39]. The issue with gradient clipping is that it can also cause suboptimal convergence by throwing away information about the true direction of the gradient, and therefore may not converge to the optimal solution. In addition to these methods, they uniquely recommended the application of weight decay applied to the network. In regression examples, their approach performed better than the $\beta-$NLL method with $\beta = 0.5$. However, without justification, their evaluation does not extend to $\beta = 1$ where it is plausible that the results for these models would have been improved. Furthermore, they recommend separate weight decay parameters for the mean and variance predictions, requiring the tuning of two hyper-parameters, thus requiring potentially expensive validation approaches. Finally, they note that the application of weight decay can also cause convergence problems if the hyper-parameters are not tuned correctly or if weight decay is too large, which is another impediment to this approach. The methods in Chapters 4 and 6 use adapted forms of the MVE network and are trained using variations of the Gaussian NLL loss function. Convergence issues arose when training directly with these loss functions, consistent with the behaviour shown above. Despite the application of the above mentioned mitigations, convergence issues were still present. This inspired the development of an approach

that is hyper-parameter light and can converge reliably in order to optimally train the models in Chapter 6.

Approaches for epistemic uncertainty quantification

Bayesian Neural Networks (BNNs) [41] is the seminal NN based approach for modelling epistemic uncertainty. Bayesian approaches rely on placing priors on the weight space and then calculating the predictive posterior, which is often intractable. Posterior sampling approaches such as Hamiltonian Monte Carlo (HMC) are used, or alternatively, Variational Inference (VI) methods, where an approximation to the posterior is found. Monte Carlo methods are often computationally expensive but lead to more accurate estimates compared to VI methods, which are easier to fit but give a less accurate estimates. 'Bayes by backprop' [42] is an equivalent Bayesian approach that directly learns the distributions on weights via back-propagation. This avoids the need to sample directly from the posterior of the entire weight-space. Ensemble methods offer a non-principled Bayesian approach to estimating epistemic uncertainty. The seminal approach described by Lakshminarayan et al. [43] relies on training ensembles of NNs on the data with differing initialisations. Ensemble methods are known to improve the accuracy of the mean prediction. Monte Carlo dropout [44] and Stochastic Weight Averaging Gaussian (SWAG) offer a cheaper approach than the classical ensemble approaches. However, these alternative approaches tend not to be as accurate in estimating the epistemic uncertainty compared to true Bayesian approaches. In addition, dropout methods are highly sensitive to the type of activation function used in the network. Furthermore, computation at test time is often a problem in ensemble approaches that motivates the use of model distillation methods [45] that are computationally cheaper at inference. The methods developed in this thesis do not seek to model epistemic uncertainty; however, Chapter 4 relies on VI theory to find Maximum A Posteriori (MAP) estimates of posterior distributions. Most NNs have the capacity for epistemic uncertainty quantification by using these methods, including MVE networks.

Deterministic approaches for aleatoric and epistemic uncertainty quantification

Deterministic NN based methods exist which can quantify epistemic and aleatoric uncertainty. These approaches avoid placing priors or assumptions on the weights of the network. Amini et al. [46] developed Deep Evidential Regression which learns the parameters of the Normal-Inverse

Gamma (NIG) distribution that describe a posterior over the Gaussian likelihood parameters. These parameters are learned via the log negative model evidence which fits the parameters given the data. Charpentier et al. [47] similarly learns the parameters of the NIG distribution in the Natural Posterior Networks (NPN). The key difference is that inputs are initially mapped to a low dimensional latent space which determines how close the input is to the training data before mapping out of the latent space to give weight on whether to default to the prior or posterior estimate in addition to the posterior parameters for that input. Both of these methods are highly reliant on calibration and hyper-parameter tuning. NPNs treat the evidence $n \in \mathbb{R}^+$ as some distance in the latent space from the training data. The methods introduced in Chapters 4 and 6 similarly offer a distinctive viewpoint on the interpretation of the evidence when using MAP based approaches.

Gaussian process regression for uncertainty quantification

Gaussian Process (GP) regression is a kernel based approach to measure epistemic uncertainty [48]. Although a reliable and principled approach to uncertainty quantification, it relies on careful choice of priors and hyper-parameter tuning. GP methods can be extended to include aleatoric noise which can be learned when fitting the marginal likelihood - used when tuning the hyper-parameters in the kernel function. They also rely on inverting kernel matrices of the data, which has complexity $O(n^3)$ and is therefore not suitable for big data regimes. Sparse approximations exist that lower the computational complexity, but compared to NN based approaches, it can still be inhibitive [49]. GP methods quantify a mean and variance prediction as functions of kernel matrices between test points and training data. GP regression is the inspiration for the kernel-based MF modelling method developed in Chapter 6.

2.4 Physics-informed machine learning

Despite all of the trust issues in ML, there is still a strong motivation to use DL models, which are extremely high-performing in big data regimes. This has motivated a tremendous amount of research in addressing trust issues in DL in the last decade. However, less explored in the literature is the capacity of PIML to address these trust problems in DL. PIML is a sub-field of Scientific Machine Learning (SciML), which leverages physics priors to bias the learning of ML models.

These methods have been shown to be high-performing when solving physics problems with small amounts of experimental data but a strong understanding of the underlying physics. This directly addresses trust problems in the form of performance, however, does PIML address other aspects of trust? If biases in PIML can be thought of as domain-specific constraints on a DL model, this naturally lends itself to increased interpretability of the model. Increased transparency naturally follows, especially amongst stakeholders who may have domain specific engineering knowledge who can better understand physics constraints in an AI model. This thesis will re-emphasize the increased levels of interpretability, transparency, and performance via the implementation of PIML methods. The main focus of this thesis is on the development of methods for uncertainty quantification in the implementation of PIML methods. For completeness, explainability, and robustness have been outlined in the literature review; however, in this thesis, these paradigms of trust will not be explored further. The rest of this section outlines the capacity of PIML methods in large multi-domain modelling of engineering systems. Initially, a brief outline of the trends in boosting the performance of DL models prior to the emergence of PIML is given.

Around the start of 2010, to increase the performance of DL models the trend was to make NNs larger through the addition of more layers and neurons. However, increasing the size of a NN, increases the non-linearity of the NN function, which results in an increasingly difficult optimisation problem. This prompted a focus on developing more efficient optimisers. An example of this is the invention of the Adam optimiser in 2013 [50], which, even today, is one of the most powerful optimisation algorithms for NNs. Adam was inspired by other powerful optimisation methods, RMSProp [51] and AdaGrad [52], and is capable of handling sparse data on noisy problems. Through these improved optimization methods, one can take advantage of the increased expressivity of larger DL models; however, these models still have the issue of overfitting.

Overfitting reduces a models ability to generalise which means reduced performance when interpolating or extrapolating from training data. This is exacerbated in engineering systems deployed in shifting environments [53]. One simple and effective method to address this is L1 and L2 regularisation [54] which works by placing penalty terms in the cost function to prevent the weights from becoming too large. These methods allow for increased generalisation of NNs on data outside of its training data. This works well when there is plenty of data across the entire domain, however, in sparse data regimes, these purely data-driven methods struggle to generalise. This leads us to the

emergence of PIML. PIML has increased the capacity of DL models to generalise outside of the observed training data, provided that there is sufficient physics or theoretical knowledge about the system to be modelled.

Karniadakis et al. [55] in their review paper on PIML organises the implementations of PIML into three categories: inductive biases, observational biases, and learning biases. In the context of DL models, an inductive bias is achieved through bespoke NN architectures which encodes the underlying physics of a system in its structure. For example, J. Darbon and T. Meng [56] have solved high dimensional Hamilton-Jacobi PDEs using custom activation functions and operations at the layers of a NN. A drawback to inductive biases is that they are limited to what can be encoded into the architecture and depend on how well defined the problem is. An observational bias is achieved by ensuring that physics is completely governed by the data. They can also be thought of as a bespoke initialisation method for training NNs through the use of pre-training strategies that help NNs learn generalisable and physics consistent models. An example of an observational bias implementation in a NN is by Thuerey et al. [57] where velocity and pressure data from the flow around the air foils was generated using numerical simulations that were used to directly train a NN to predict the same outputs given the same input information as the numerical solver. A benefit of this approach is that once trained, the NN can solve problems 1000x faster than the numerical solver. A large drawback to this method in isolation is its poor generalisation outside of the training set. Finally, a learning bias is achieved in a NN through the incorporation of the physics equations of an underlying system into the loss function of a NN. These equations function as a regularization agent which constrains an NN to learn a unique solution to a problem. Raissi et al. [58] developed a seminal approach to incorporating learning biases in NNs called Physics-Informed Neural Networks (PINNs) in 2019.

An alternative paradigm for implementing physics knowledge into ML models is via Multi-Fidelity (MF) modelling. MF modelling consists of a set of methods that efficiently leverage datasets with varying fidelity to make inferences about a physical system's parameters, or outputs, with the necessary accuracy and speed [59]. MF models can improve the accuracy of inference and/or decrease the discrepancy between the underlying mathematical representations in a physics-based model and the physical system. The focus in this thesis is on 'fusion-based' data-driven MF modelling methods - a category of MF modelling which merges data from multiple sources with

varying levels of fidelity to train a model. These methods typically apply a learned correction on Low-Fidelity (Lo-Fi) data to better match the High-Fidelity (Hi-Fi) data when training. The fidelity of a data-driven model is mostly dependent on the fidelity of the training data source. Training data made up of measurement data is likely to be Hi-Fi (i.e. close to the ground truth of the system); however, measurement data can still be imperfect. If training data are obtained from a physics-based model, then the fidelity is dependent on: the choice of parameter selection of physics-based or simulation based methods (e.g., the accuracy of a computational fluid dynamics model can be improved with a higher discretisation or shorter time steps); and the level of simplification from the mathematical representation used in a particular physics-based model (e.g., simplifying the differential equations being solved or reducing the dimensions)[60]. In this thesis, Hi-Fi data is defined as measurement data from the physical system and Lo-Fi data as data generated from mismatched physics-based models. As can be seen, physics-knowledge can be implemented via a choice of data for given fidelity dataset.

Finally, in a paper by Kartapne et al. [61], a paradigm for an emerging branch of Data Science called 'Theory Guided Data Science' is outlined. This includes and adds to the broad categorisation of PIML methods described by Karniadakis et al. [55]. An additional approach to PIML described in the paper is performing refinement on the outputs of an ML model. The ML model prunes candidates on which more expensive calculations can be performed that would not be feasible on the entire range of solutions. Another approach is hybrid solutions where non-ML models, governed by physics which may not be entirely accurate, are used to initially estimate a solution to a problem. Then an ML model is used to estimate the non-ML model's discrepancy. Data assimilation is another approach that can be used for modelling dynamical systems. This approach involves creating a dependence of training outputs to the current state and previous states of a system which would be described by known theory over time. Then there is also the use of ML to calibrate the parameters of other models through the use of observational data of that system.

2.4.1 Physics-informed neural networks

DL models which use learning biases are called Physics-Informed Neural Networks (PINNs) and became famous due to the seminal paper published in 2018 by Raissi et al. [58]. PINNs have been shown to solve a range of physics equations in sparse data regimes including finding forward so-

lutions to the Schrodinger Equation and Wave Equation, inverse problems finding the coefficients of the Navier Stokes Equations [58] or more complicated two-phase Stefan problems which have moving boundaries or moving interfaces [62]. The PINN concept was first described by Lagaris et al. [63] back in 1998. Jia et al. [64] also outlined a similar method using physics based regularization methods to train Recurrent Neural Networks (RNNs) when modelling lake temperatures. However, Raissi et al. [58] went one step further by leveraging the powerful Auto-Differentiation (AD) [65] used for back propagation in their methodology, making these methods more viable and scalable. PIML through learning bias or PINNs is one of the more flexible methods for implementing physics priors into NN and therefore is a good starting point for introducing physics priors into NNs for engineering problems.

An advantage of PINNs is their ability to incorporate some physics knowledge and some empirical data when determining a solution. This is particularly useful in engineering problems where there could be sparsity of data due to corruption, lack of sensors across the entire domain of a system, or difficulties in data collection. Another advantage of PINNs is their mesh free method that enables for quick predictions after the training of a NN. Raissi et al. [66] demonstrates a use case for this benefit by implementing PINNs to model the viscosity and pressure of blood in an aneurysm using flow visualisations obtained from MRI scans. This demonstrates a PINN's utility in non-intrusive diagnostics in the medical field, where the alternatives of measuring viscosity and pressure of blood would be inviable or expensive via direct measurements. This example also highlights the strength of PINNs in solving inverse problems.

One drawback to PINNs is their current inability to outperform spectral or finite element methods for forward problems in terms of performance [58]. Another drawback of PINNs is their scalability. It is found that as domains grow, PINNs take significantly longer to train and decrease performance [67]. This has been mitigated to an extent through domain decomposition methods such as XPINNs [68], CPINNs [69] and PPINNs [70]. However, there are still open questions on the best way to divide a domain for the application of these methods. Another drawback to using PINNs in large domains is the imbalance between the residual and boundary loss terms across layers when training the networks. This imbalance results in PINNs not properly learning the boundaries of a problem, which means the model is incapable of learning a unique solution. This phenomenon is known as gradient stiffness. There are mitigations to this gradient stiffness as shown by Wang et

al. [67] such as using adaptive weights on the loss terms to balance the gradients of the loss terms when backpropagation occurs. A drawback to using PINNs is that, by themselves, they are not particularly applicable to modelling systems with dynamic behaviour with, potentially, changing geometries or boundary conditions. This constrains PINNs to solving problems with well defined parameters. Finally, there are no guarantees for the convergence of PINNs to a unique solution, as elaborated in the next section.

2.4.2 Considerations to using physics-informed neural networks

AD is a point-wise differentiation method used in the backpropagation algorithm [65]. PINNs take advantage of AD by using it to differentiate the output of an ordinary NN and then use this to calculate a residual from a differential equation. This process extends the computational graph AD builds for ordinary NNs. The training of PINNs is computationally more expensive than the training of an ordinary NN in two ways. Firstly, the use of multiple loss functions can increase computation time through the building of multiple computational graphs. Backpropagation then needs to be conducted for each of these additional loss functions in PINNs as opposed to backpropagation being performed on a single loss function in ordinary NNs. Secondly, point-wise differentiation for a single point, $x \in \mathbb{R}^n$ for a NN generally described by a mapping $z : \mathbb{R}^n \mapsto \mathbb{R}^m$, requires an operation of reverse mode or forward mode AD when calculating the residuals. Each of these operations of forward mode or reverse mode has to be repeated for each of the points. This is much more costly than calculating the derivative, for example, of simply calculating the MSE loss, which requires one reverse mode operation to obtain gradients of the model parameters.

Another issue with computation is that PINNs require a sufficient number of physics collocation points spread across the domain in order to converge to a unique solution. NNs trained using a large number of training points have been empirically shown to result in poor performance and generalisation of the model [71]. Batch training is the common mitigation of this problem that has been shown to work for PINNs [72].

There are no guarantees for the convergence of PINNs to a unique solution. In the original PINN methodology, point-wise differentiation is performed on the output of the model. These outputs are approximations to the solution based on the known data, including in the form of Boundary Conditions (BCs) and Initial Conditions (ICs). When using AD to calculate the residual, Equation

(3.2), there is nothing to prevent the PINN from learning trivial solutions across the domain of a problem. For example, all the outputs of the model could equal zero to satisfy the residual equation. In the literature, common mitigations to this problem are random weight initialisations and ensemble methods to decrease the likelihood that a PINN converges to trivial solutions. Another mitigation is to use hard constraints instead of soft constraints to enforce the boundary conditions in a PINN. Soft constraints involve creating additional loss constraints on the collocation points of the boundaries. A hard constraint adjusts a PINN such that, instead of approximating the true solution, it uses an ansatz equation to define an approximate solution to the differential equation in the physics residual of a PINN. This ansatz solution would satisfy the boundary conditions of the problem and, therefore, fewer loss functions are required [73][74]. This eliminates the need for separate boundary loss terms when using PINNs.

As already mentioned, a further challenge to using PINNs is gradient stiffness which occurs during the training of a PINN. Wang et al. [67] observed that PINNs can struggle to fit boundary conditions in even simple physics problems, leading to large prediction errors. When the gradients with respect to the model parameters of each loss function were analysed, it was shown that the gradients for the physics loss function with respect to the model parameters tend to be larger than the gradients of other loss function at each layer. This means that PINNs tend to be better at learning at the physics collocation points than at any data point. This results in the physics loss function not converging to unique values for their collocation points. The consequence of gradient stiffness is that there becomes competition between the physics loss terms and data loss terms. This is one of the reasons why soft constraints fail to make PINNs converge to unique solutions. Wang et al. [67] have created an adaptive learning algorithm called learning rate annealing, which assigns weights to the loss functions during training according to the magnitude of their gradients as a mitigation of this problem. In Chapter 5, Wang et al.'s [67] learning rate annealing algorithm is adapted to solve convergence issues when using Gaussian NLL loss functions to train NNs, where further analysis of this algorithm and its suitability to help address the training of probabilistic models is outlined.

2.4.3 Applications of physics-informed neural networks

PINNs are starting to become more popular in modelling engineering processes. However, there are fewer examples of their applications in which the system is described by lumped parameter

models. Gokhale et al. [75] have developed two variants of PINN architectures for control-oriented modelling of the thermal behavior of a building. These models can predict the building temperature and its thermal mass along with the corresponding control action for temperature regulation. The thermal mass is a latent parameter that is otherwise difficult to measure. These models use a state-spaced model to describe the underlying physics of the system. It is found that these architectures perform better than pure data-driven NNs and are far more data efficient. Another paper by Misyris et al. [76] uses a PINN to model the swing equation for modelling power system dynamics. This method was able to accurately estimate both the rotor angle and frequency of a generator, as well as to discover the generator inertia and damping coefficient from a set of observations. Whilst both papers apply PINNs to model real lumped parameter models, it is unknown how well this method scales when the number of variables in the underlying physics equation used in the loss function increases.

Fully Connected Networks (FCNs) are the most common type of NN used to apply PIML using the PINN methodology. However, there have been a few instances where Recurrent Neural Networks (RNNs) have been used. An example already mentioned is by Jia et al. [64] who created a physics guided RNN for lake temperature modelling. The method initially 'pre-trains' a RNN using data generated from an un-calibrated generic mathematical model of lake temperatures with the motivation that the ML model will be closer to the target solution; thereby only a small amount of observed data is needed to refine the model, which is an example of observational bias. In a second stage of training, a neural network is regularised using physics priors. The output of the pre-trained neural network is fed into two loss functions, one which uses the output to calculate the conservation of energy in the lake through the change in thermal energy and another which uses observational data with pairs of input and output data. In the former loss function, the change in thermal energy using the output of the neural network is compared to another estimation of the change in thermal energy which does not require the temperature at certain depths. This constraint is very similar to how PINNs work, however, no AD is used in calculating this energy balance which is the key difference. However, this method can be easily adjusted for the use of AD for calculating the residuals of differential equations. Another example is by Viana and Nacimento [77], who have created a bespoke RNN architecture (inductive bias) implementing Paris' law, to predict the crack length of the synthetic fleet of airplanes subject to different missions. They used

an RNN to implement Euler's forward method to obtain the crack length.

Building multi-scale DL models, in the physics sense of a particular system increasing or decreasing its scales of time and space, is a challenge that is faced especially when modelling fluids and solids as well as various physical and chemical phenomena. The physics that is applied at a micro level may not be applicable at a macro level, or even at smaller changes in scale. This means there is a range of parameterisations of the physics problem including, but not limited to, variations in boundary and initial conditions, coefficients, source terms etc. This problem is addressed through being able to learn operators instead of functions. Lu et al. [78] have developed a DL architecture for learning operators called DeepONets which enables multi-scale modelling. Provided there is a large set consisting of input-output observations for each of these parameterisations of the physics problem at each scale, DeepONets are fast to train. Wang et al. [79] have extended this architecture to create a Physics-Informed DeepONet by using the PINN model's method of regularisation to alleviate the need for generating large amounts of training data. DeepONets have been shown to be quicker at solving parameterised PDEs than using PINNs to solve each parameterisation individually. There is, however, a slight loss in performance when comparing the output of a DeepONet for a specific parameterisation of problem versus a PINN modelling that same parameterisation. One can train a single PINN to learn all the parameterised representations of a PDE/ODE, however, this is shown to have considerably worse performance than using DeepONets. Physics-informed DeepONets also suffer from the same issue of spectral bias that all NNs face. There is no multi-scale modelling in this thesis, however, the DeepONets methodology may be a method that can allow one to model a system with uncertainty. For example, if there are varying priors for the underlying coefficients and parameters of a model, DeepONets allows for all of those representations to be modelled simultaneously.

Tartakovksy et al. [80] have developed a multi-physics-informed neural network to estimate space-dependent hydraulic conductivity, hydraulic head, and concentration fields from sparse measurements in applications for subsurface transport problems. Their method is particularly interesting, as they use multiple NNs to model each of the fields and use a conservation equation, in this case the Darcy flow equations, to regularise all of the NNs in the PINN fashion. This approach is shown to be more accurate than methods using a singular PINN for estimating the same fields using the same training data. This motivates an approach for modelling larger systems, such as GTEs,

by using PINNs for modelling different aspects of the physics, which are all trained by using conservation laws, or physics equations, which relate each of the variables of interest. It is unknown what issues are faced when scaling this problem to use many more PINNs to model many different systems contained within one larger system, such as those found in DTs.

2.4.4 Fusion-based multi-fidelity modelling approaches

Gaussian process based multi-fidelity modelling approaches

MF modelling methods which are capable of providing an uncertainty measure on predictions similar to the concept of mismatch uncertainty exist, however, they bear computational limitations at inference or training which scales as data size and dimensionality increase. Perdikaris et al. [81] developed autoregressive GP based MF methods, capable of learning non-linear correlations between datasets of different fidelity via autoregressive GPs. Given datasets of increasing fidelity $\mathcal{D}_\rangle \in \{\mathcal{D}_1, \cdots, \mathcal{D}_n\}$ where each dataset has pairs of inputs and outputs $\{X_i, Y_i\} \in \mathcal{D}_i$, an initial GP f_1 is trained on $\mathcal{D}_1$. Recursively, GPs f_{i+1}, are trained on concatenated inputs $[X_i, f_{*_{i-1}}]$, where $f_{*_{i-1}}$ is the posterior mean, on the target outputs Y_i. The hyper-parameters of each of the GPs are trained via the marginal log-likelihood and gradient descent. This step scales as $\mathcal{O}(n_i^3)$ where n_i is the size of the dataset $\mathcal{D}_i$. In evaluation, each of the posteriors at each fidelity need to be sampled and propagated to the next layer, an operation which is $\mathcal{O}(n_i)$. Then, upon prediction, the predictive posterior for the final GP is obtained via Monte Carlo integration. The variance of the predictive posterior represents an epistemic uncertainty, but also the uncertainty on the correlation between each fidelity level. The choice of kernel would be a prior choice on how an engineer believes the correlation between fidelities of datasets is related. This approach is expensive at inference and also does not lend itself to easily retraining the model; hence, for DT applications, this would be inhibitive. Furthermore, capturing noise on the data is difficult using this approach, especially if the lowest fidelity dataset has a lot of noise.

Raissi et al. [82] developed a MF modelling approach using a single, multi-output GP applied to the feature space of all the datasets. The feature space is learned via a NN and the parameters of this model is jointly learned alongside the GP hyper-parameters by optimising the marginal log-likelihood, which can be achieved via gradient descent. Once the parameters are learned, the

standard training and inference costs of a GP then apply.

Cutajar et al. [83] developed a deep GP approach to MF modelling where each layer in the deep GP is conditioned on the fidelity of each dataset $\mathcal{D}_\rangle$. This approach only requires a singular loss function to train, whereby all the kernel parameters at every layer are jointly optimised. The problem with Deep GPs is that they are prohibitively expensive to train, $\mathcal{O}(SNM^2L)$ where S is the number of Monte Carlo samples, N is the number of training points, M is the number of inducing points, and L is the number of layers. As described in Chapter 2.1.4, deep GPs have convergence issues in addition to the need to specify the number of inducing points M at each layer, which is an additional hyper-parameter that needs tuning. Despite these issues, this approach currently outperforms all other GP based approaches for MF modelling with uncertainty.

Neural network based multi-fidelity modelling approaches

NN based MF modelling methods are less sensitive to the curse of dimensionality found using autoregressive MF GP methods. Meng et al. [84] and Chakraborty [85] have developed deterministic MF NN approaches that learn the correlation between different fidelity datasets using NNs and then apply corrections upon prediction. The former approach relies on learning both the linear and non-linear correlation between Lo-Fi data and the Hi-Fi data using 2 separate NNs. The latter approach trains a single NN initially trained on Lo-Fi data, and a second step is applied where initial layers of the NN are frozen, and the final 1 or 2 layers are trained on Hi-Fi data. This transfer learning approach similarly learns a correlation between the Lo-Fi and Hi-Fi data in order to make an improved prediction. They utilise the strong capacity of NNs to learn latent features and learn from big data - demonstrating scalability to their approaches. However, these models are not capable of providing any form of uncertainty in the output. Furthermore, whilst the use of NNs to learn a correlation can yield great accuracy when interpolating within the domain of Hi-Fi data, they may not be as accurate when extrapolating. Meng et al. [86] developed a Bayesian Neural Network (BNN) MF modelling approach with similar capabilities to Perdikaris et al. [81] and Cutajar et al. [83]. Their approach learns a correlation between the Lo-Fi and Hi-Fi datasets using a BNN, where the network is able to capture epistemic uncertainty. Aleatoric uncertainty can also be captured assuming that the noise model on the data is known apriori. As mentioned in Chapter 2.1.4 these methods require MCS methods which can be very expensive. Additionally, at inference time, samples need

to be drawn from the distribution on the weights, which again can be expensive depending on the number of parameters in the model.

Dehghani et al. [87] propose a semi supervised student-teacher approach that, while not explicitly framed as multi-fidelity modelling, can be considered one. Their approach improves predictive performance of a model by leveraging relationships between two datasets of different fidelity. A 'student' NN model is trained on weakly labelled data (reframed here as 'Lo-Fi' data), while a separate 'teacher' GP model is trained on strong data as close to the ground truth as possible ('Hi-Fi' data). The teacher model then generates a soft labelled dataset using the inputs of both the Lo-Fi and Hi-Fi data, this includes the mean prediction along with the associated uncertainty. The student model is then retrained using this soft-labelled dataset, with each target's associated uncertainty used to down-weight the influence during gradient descent. A tuning parameter is used to adjust how much trust is placed on the teacher. The resulting output does not have an associated uncertainty quantification on it.

Computational costs at inference and training

In addition to their modelling capabilities, the implementation of these MF approaches in DTs is strongly influenced by their computational cost at inference and training. A brief and high-level overview of the computational costs associated with the aforementioned methods is provided here.

The scalability of the predictive modelling methods in this thesis is important. For example, for effective fleet management large volumes of data need to be processed. To give a perspective on the quantity, consider the approximate number of data points that is obtained from a single sensor type on an aircraft. Assuming that an aircraft makes a 10 hour flight per day for 200 days throughout the year, per day, the data collected from a single 1hz sensor is 36000 points. Therefore, 7M data points would be collected from a single sensor per year for each flight. For gas turbine applications with a DT, data for at least 10 parameters would likely be required. Moreover, consider that for fleet management problems, information from whole engine types is required, which could approximately be up to 1000 engines. This would increase the volume of the data to be processed by at least 10000. This provides some perspective on the volume of data that might be handled in engineering applications.

Hartwell et al. [29] work provides additional perspective of the volumes of data required in

fleet engine asset management applications. They discovered that to achieve sufficient accuracy when building NN-based predictive models, around 50k to 100k training data points were required to capture all the complex dynamics over a variety of operating conditions. This corroborates with the above example calculation for the swathes of data required for predictive modelling applications. Comparatively, around 30k parameters were required in the NN to create a model with good accuracy.

Having contextualised the cost problem and shown indicative volume of the data sizes required for modelling applications, a comparative overview of the costs at inference is given in comparison to a feed forward NN. The computational complexity of a feed-forward NN at inference is dependent on the number of parameters P. Therefore, it approximately has complexity $O(P)$. In an ordinary GP, the time complexity at inference is $O(N^3)$ where N is the number of data points; however, if the inversion of the kernel matrix is stored in memory, this can be reduced to $O(N^2)$ by storing the inverted kernel matrix in memory, which is size $N \times N$ [48]. This can be inhibitive during inference time in large data regimes. However, memory can be reduced by using low-rank approximations of the matrix. Finally, the time complexity of a feed-forward BNN at inference is $O(SP)$ where S is the number of samples and P is the number of parameters in the model. The number of samples is proportional to the accuracy in the predictive outputs. As a lower bound, the GP-based methods developed by Perdikaris et al. [81], Raissi et al. [82] and Cutajar et al. [83] would at least have an inference cost of $O(N^2)$. The MF frameworks developed in Chapters 4 and 6 are implemented on deterministic NNs. While GP based methods are considered the state of the art for fusion-based MF modelling with uncertainty modelling, the use of deterministic NNs over GPs reduce computational complexity at inference assuming that the number of data points N does not exceed P (i.e. $N > P$). Hartwell et al. [29] justifies this assumption in the example described earlier. Additionally, if models are required to be onboard embedded systems, there are likely to be memory restrictions; assuming that an $N \times N$ matrix would need to be stored onboard for GP regression models, a NN is a better candidate model.

Now, a comparative overview of the costs during training is given. As mentioned already, the cost of training a GP is $O(N^3)$. The cost of training a feed-forward NN is more nuanced and is highly dependent on many variables. To simplify this analysis, consider that there are N data points, E number of epochs, P parameters. The assumption is kept that $N > P$. The number of

epochs required is task dependent, but for simplicity $E < N$. The computational cost to train a feed-forward NN is approximately $O(NPE)$ which is less than $O(N^3)$. This shows that for one instance of training these models, a NN is cheaper. When multiple or retraining is required, each time with M number of new data points, a GP requires another inversion of the kernel matrix with the new data points in addition to the previous data. This would be approximately $O((N+M)^3)$. In contrast, a NN can be retrained using the new data points on the previous model, meaning the cost is $O(MPE)$. Even if a fraction of the previous data points were used to prevent the model 'forgetting', the cost would still be less than what is required for a GP model. This could also be mitigated by slowing down the rate of certain parameter's updates when training instead of retraining [88]. Therefore, even at training time, a NN offers a large cost benefit as a model choice over a GP model.

2.4.5 Uncertainty quantification in multi-scale modelling methods

Although multi-scale methods are not used in this thesis, for completeness, multi-scale modelling approaches with uncertainty quantification are outlined here. Moya et al.[89] develop B-DeepONet, a Bayesian DeepONet built to tackle model degradation when trained on noisy data, a known weakness of DeepONets. Their approach to creating a Bayesian model is similar to that Neal et al. [41], who place priors on the weights space of a neural network and approximate the posterior distribution via MCMC methods. As a result, B-DeepONets offers epistemic uncertainty quantification on its estimates. Another extension to DeepONets is Prob-DeepONets [89], which captures the aleatoric uncertainty in data by placing a mean and variance head on the outputs and fitting the output to a Gaussian distribution. Lin et al. [90] propose an improved B-DeepONet which is trained using stochastic gradient Langevin dynamics, which reduces training cost compared to classical MCMC sampling methods.

2.5 Gaps in research

The literature review began with a motivation for the engineering industry to embrace the recent advancement of DL methodologies and enabling technologies for DTs applications. However, this adoption in many applications is restricted by the trust issues inherent in DL. Therefore, a holis-

tic trust paradigm for ML was outlined, in addition to key definitions of trust and what enables trust across each of the aspects. The trust challenges when incorporating DL into DTs were then discussed, where three main focus areas were identified. Firstly, the sparsity in the observational data can reduce their performance. Secondly, physics-based models have the capacity to alleviate sparsity issues, but they may have model mismatch, which again contributes towards a reduction in performance. Thirdly, these limitations motivate the development of MF modelling methods, that can leverage both observational data and physics-based models to improve predictions in sparse data regimes. However, for MF modelling approaches to be trustworthy, it is essential that aleatoric and mismatch uncertainty can be captured in model predictions. A review of uncertainty quantification ML methods was then presented, comparing their scalability, computational cost, and types of uncertainty that they are able to quantify. Subsequently, an overview of the field of PIML was conducted and a detailed review into PINNs and 'fusion-based' MF modelling methods was outlined. One common drawback to existing 'fusion-based' MF modelling capable of quantifying mismatch uncertainty is their large computational expense at training and test time when using even state-of-the-art GP methods.

All of the research gaps that will be addressed in this thesis are outlined below.

1. There is a lack of scalable MF modelling methods that make principled trade-offs between datasets whilst being computationally effective at inference time, therefore suitable for applications within DTs.

2. There is a need for MF modelling methods that quantify mismatch uncertainty in addition to allowing engineers to place prior beliefs about their understanding of the mismatch. Enabling for engineers to fully leverage their engineering intuition and knowledge about a problem when creating models

3. Current state of the art approaches for mitigating suboptimal convergence issues when using Gaussian NLL loss functions to train probabilistic neural networks require hyper-parameter tuning, multi-stage training or do not converge consistently. Therefore, there is a need for hyper-parameter light methods to ensure stable optimisation using the Gaussian NLL or variants of this loss function.

Chapter 3

Identification of limitations in implementing learning biases in physics-informed neural networks

Chapter 2 stressed the need to mitigate model mismatch when implementing Physics-Informed Machine Learning (PIML). The most common strategies identified were corrective approaches in which misspecified models are transformed, either linearly or non-linearly, to better approximate the true underlying system. The majority of such approaches have been developed in the context of Multi-Fidelity (MF) modelling, where model mismatch naturally arises as a discrepancy found between datasets of different fidelities. Within the literature for Physics-Informed Neural Networks (PINNs), Zou et al. [91] notably identify model mismatch as a key failure mode, particularly in cases of complete structural misspecification of the governing equations. They propose an additive corrective network embedded into the learning scheme for PINNs to account for the missing physics. By contrast, a relatively less explored avenue for mitigating model mismatch in PINNs is the use of loss-function weightings based on prior trust in the physical model. Although re-weighting schemes have been widely implemented to solve gradient imbalances during training, they have not been systematically investigated as a mechanism for mitigating model mismatch in PINNs itself.

This chapter addresses this gap by analysing the effect physics-loss re-weightings have on PINN performance, so as to seek a preliminary mitigation strategy for model mismatch. Using a 1-D heat

equation example as the test case, the model mismatch was artificially generated by increasing the coefficients of the governing equation while keeping the boundary and initial condition data fixed. PINN models were trained with different levels of mismatch to quantify the degree to which predictive accuracy degrades. By reducing the weighting of the physics-loss term, the aim is to see if reducing the learning of the misspecified model can mitigate model mismatch. This approach does not fully correct the model mismatch, but provides important insights into the limitations of the PINNs methodology and motivates a pivot towards methods situated in MF modelling in later chapters. This offers more flexibility for how a re-weighting scheme is implemented but can also incorporate data generated from more complex physics-based models, such as simulation models. In the wider context of model mismatch, a further experiment was conducted to evaluate the effect re-weighting the physics-loss term with no model mismatch has in the presence of noisy observational data. This experiment provides insight into a PINN robustness to noise when training. In general, this chapter serves as preliminary work that informs and justifies the MF approaches developed in Chapters 4 and 6.

This chapter begins by introducing the PINNs methodology for the forward problem. The impact of model mismatch in PINNs is then examined using a 1-D heat transfer example, where the amount of mismatch and weightings applied to the physics-loss are varied. Similar experiments are conducted to assess the effect of observational noise on the data terms under varying physics loss weighting without model mismatch. Finally, the experimental findings are discussed and the paradigm shift towards MF modelling is outlined.

The key contributions of this chapter are summarised below:

- A systematic investigation of the effect of model mismatch and data noise on PINN performance using a 1-D heat equation example.
- An empirical evaluation of physics-loss re-weighting as a strategy for mitigating the effects of model mismatch and measurement noise in PINNs.
- Motivation for transitioning towards MF modelling frameworks, informed by the experimental results and observed limitations of standard PINNs.

3.1 Background

The PINN methodology for obtaining the general solution in a forward problem is outlined. The aim is train a Neural Network (NN) which approximates the solution, defined by $u(t,x)$, given a non-linear Partial Differential Equation (PDE) and sufficient Boundary Conditions (BCs) and (or) Initial Conditions (ICs) to solve the PDE. The general form of the PDE is described by Equation (3.1).

$$u_t + \mathcal{N}[u;\lambda] = 0 \qquad \text{s.t.} \qquad x \in \Omega \subseteq \mathbb{R}^d,\ t \in [0,T] \tag{3.1}$$

where $\mathcal{N}[u;\lambda]$ is a non-linear operator parameterised by λ and u_t is the PDE of u w.r.t. t. For this general solution, the BCs and ICs will not be explicitly defined, but it is assumed that data can be generated from them. This data is encapsulated within the dataset defined by $\left\{t_u, x_u^i, u^i\right\}_{i=1}^{N_u}$ where N_u is the number of data points. This information is now sufficient to train a PINN. The next step is to define the loss functions that are used to train the NN.

First, the residual equation $f := f(t,x)$ is defined below, which is the left hand side of the Equation (3.2).

$$f := u_t + \mathcal{N}[u;\lambda] \tag{3.2}$$

$u(t,x)$ is then approximated by a Neural Network (NN), which is trained using the loss function described by Equation (3.3) below.

$$MSE = MSE_u + MSE_f \tag{3.3}$$

$$MSE_u = \frac{1}{N_u}\Sigma_{i=1}^{N_u}|u(t_u^i, x_u^i) - u^i|^2 \tag{3.4}$$

$$MSE_f = \frac{1}{N_f}\Sigma_{i=1}^{N_f}|f(t_f^i, x_f^i)|^2 \tag{3.5}$$

The residual equation f is evaluated using Automatic Differentiation (AD), a method for point-wise differentiation, on the output $u(t,x)$ at physics collocation points defined by $\left\{t_f^i, x_f^i\right\}_{i=1}^{N_f}$. These data points are defined by the user, where a larger number typically leads to better performance at the cost of computational expense. The addition of this physics-loss ensures that solutions of $u(t,x)$ do not violate the physics of the problem. Unique solutions are obtained due to NN learning BC and IC data via MSE_u. This process is more clearly shown by Figure 3.1 below

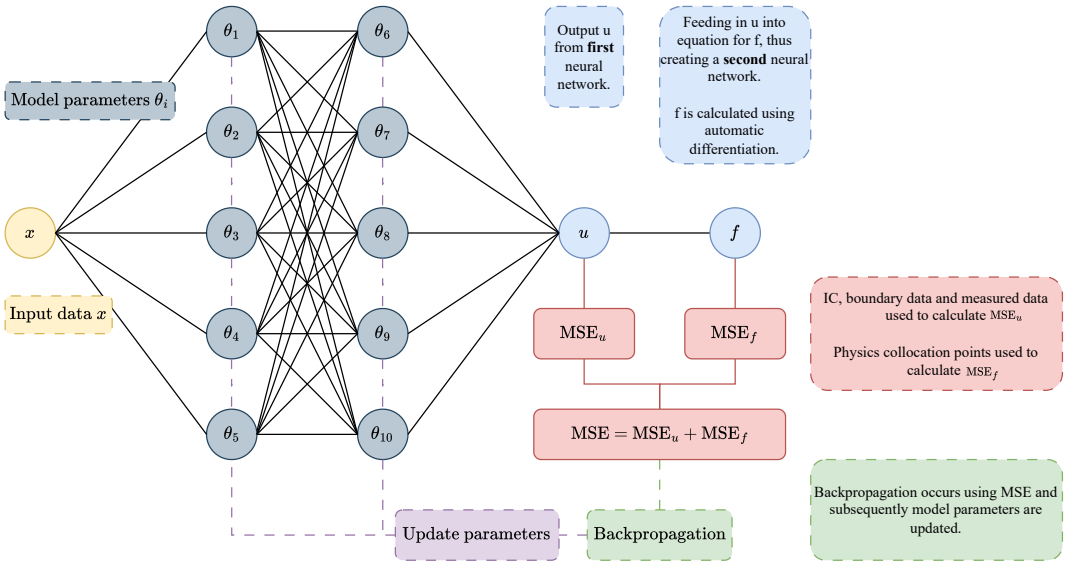


Figure 3.1: Outline of a PINN applied to a forward problem. When performing back-propagation using MSE, there is an extra layer of activation functions in MSE_f. This technically creates a second network despite there seeming to be one network in the diagram. Diagram is an adaptation of Ben Moseley's blog explaining PINNs methodology [92]

.

PINNs are enabled by AD which is fundamentally based on the chain rule. However, unlike the explicit chain rule, an implementation of AD breaks up a function into its primitive operations and connects them via a computational graph. This computational graph allows derivatives to be found for specific inputs by applying chain rule at each point in the computational graph and passing on the outputs to the next part of the graph. AD is the algorithm that is used for back-propagation in popular Deep Learning (DL) libraries such as PyTorch and TensorFlow. A detailed explanation of how AD can be found in Gunes et al. [65]. MSE_f will typically contain an expression that is a differential equation of u. With AD and sufficient boundary and initial conditions, a solution u can be found which satisfies this equation.

3.2 Problem set up: 1-D heat equation

Having provided a primer on the PINNs forward problem methodology, this section introduces the 1-D heat equation example used in this work. The governing equation, boundary conditions, and

initial condition are defined below, along with an outline of the general training strategy, the model architecture, and the learning parameters that are used in every model.

$$u_t = k u_{xx} \qquad \text{where} \qquad k = 0.003 \tag{3.6}$$

$$u(0,t) = u(1,t) = 0 \qquad \text{for} \qquad 0 \leq t \leq 100 \tag{3.7}$$

$$u(x,0) = 50x(1-x) \qquad \text{for} \qquad 0 \leq x \leq 1 \tag{3.8}$$

Here, k is the thermal conductivity constant and $u(x,t)$ denotes the temperature at location x and time t. A small value of k was chosen for this problem. Large coefficients in the physics loss equations have been shown to cause gradient imbalances and consequently leads to suboptimal convergence when training [67][93]. A physical interpretation of the problem is illustrated by Figure (3.2).

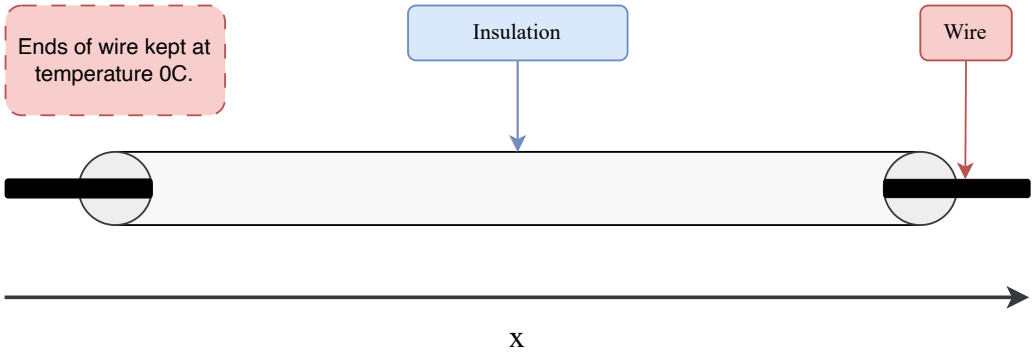
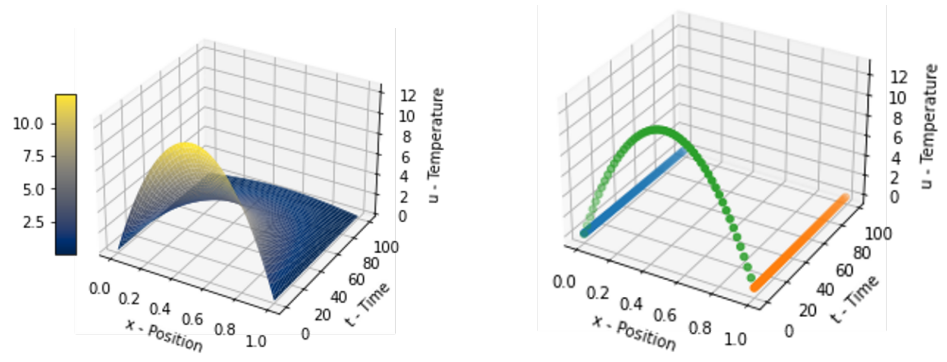


Figure 3.2: A physical representation of the heat equation with boundary conditions. The insulation around the wire is to demonstrate that no external heat sources affect the system.

Equation (3.6) can be solved analytically through the use of the Fourier Series and the separation of variables [94]. This closed form solution was used to generate data for training and evaluating the performance of the models and is described by

$$u(t,x) = \sum_{\substack{n=1 \\ n\ odd}}^{inf} \frac{400}{\pi^3 n^3} \sin(n\pi x)\, e^{-n^2\pi^2 0.003t} \tag{3.9}$$

Figure 3.3 (a) shows a surface map of the analytical solution evaluated at 501 evenly spaced points across the domain $x \in [0,1]$ and $t \in [0,100]$. Figure 3.3 (b) shows a plot of the boundary and initial condition data used for the training.



(a) (b)

Figure 3.3: (a) Surface map of the exact solution $u(t,x)$ generated using a sample of 501 points evenly spaced across the domain of $x \in [0,1]$ and $t \in [0,100]$. (b) Plot of boundary and initial conditions of analytical solution.

3.2.1 Methodology for training a physics-informed neural network on the 1-D heat equation

In modelling the 1-D heat transfer problem, the PINNs model used a loss function described by Equations (3.10) to (3.14) below

$$MSE = MSE_0 + MSE_b + MSE_d + MSE_f \tag{3.10}$$

$$MSE_0 = \frac{1}{N_0}\sum_{i=1}^{N_0}\left|u(0,x_0^i) - u_0^i\right|^2 \tag{3.11}$$

$$MSE_b = \frac{1}{N_b}\sum_{i=1}^{N_b}\left|u(t_b^i,0) - u(t_b^i,1)\right|^2 \tag{3.12}$$

$$MSE_d = \frac{1}{N_d}\sum_{i=1}^{N_d}\left|u(t_d^i,x_d^i) - u_d^i\right|^2 \tag{3.13}$$

$$MSE_f = \frac{1}{N_f}\sum_{i=1}^{N_f}\left|f(t_f^i,x_f^i)\right|^2 \tag{3.14}$$

Here, $\{x_0^i,\ u_0^i\}_{i=1}^{N_0}$ denotes the initial data, of which there are $N_0 = 501$ training points. $\{t_b^i\}_{i=1}^{N_b}$ corresponds to the boundary data of which there are $N_b = 1002$ training points. $\{t_d^i\}_{i=1}^{N_d}$ correspond to experimental observations of which there are $N_d = 1503$ training points. The purpose of these

additional experimental points is to simulate sensor data at sparse locations across the domain. The u_0^i, t_b^i and t_d^i values are equally spaced points sampled from Equation (3.9). $f(t,x) = u_t - ku_{xx}$, where $k = 0.003$, corresponds to the residual equation and $\left\{t_f^i, x_f^i\right\}_{i=1}^{N_f}$, in Equation (3.14), corresponds to collocation points on $f(t,x)$ of which there are $N_f = 2601$. The training data points are shown in Figure 3.4 below.

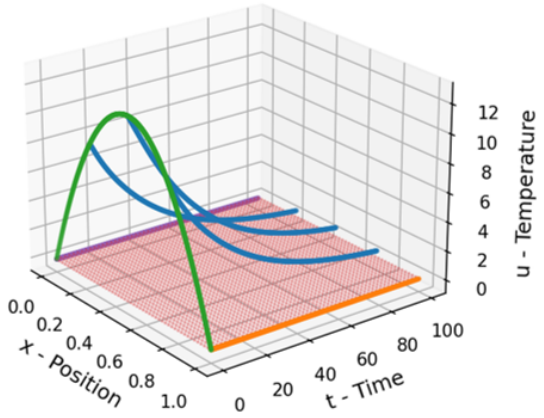


Figure 3.4: Training data with $N_0 = 501$ initial data points shown in green, $N_b = 1002$ boundary training points shown in orange and purple, $N_f = 2601$ physics collocations points shown in red and $N_d = 1503$ experimental points shown in blue.

3.2.2 Model architectures and learning parameters

The model architecture and learning parameters were kept consistent for all experiments. All models used a Fully Connected Network (FCN) having 2 inputs, 3 hidden layers, with 20 neurons per layer. Tanh activation functions were used at the hidden layers as suggested by Mishra et al. [95]. The model was initialised using Glorot normal initialisation [3] which is a standard method for preventing the vanishing gradient problem in NNs. A piece-wise constant decay was used to adjust the learning rate at set epochs. A learning rate of 1E-02 was used for 5000 epochs, then 1E-03 was used for 5000 epochs, then 5E-04 was used for 10000 epochs for a total of 20000 epochs used to train each model. Adam optimiser was used. On input, data was normalised in order to aid in the learning process of the NN.

3.3 Evaluating physics-informed neural network performance with coefficient model mismatch and weightings

Having described the problem set-up and formulation of the standard PINN model, the model performance is investigated when there is coefficient model mismatch in the governing equation. The motivation for this experiment is to see the effect of changing the weighting on the physics loss function, Equation (3.5), in the presence of model mismatch. The identification of k is not hard, but there may be un-modelled behaviours that may not be captured by k. A hypothesis is that a small weighting on the physics loss function in the presence of large model mismatch increases the performance of the model relative to a model with no weighting applied to the physics loss function. The weighting in this experiment is multiplied by the physics loss function as described by Equation (3.15) below.

$$MSE = MSE_0 + MSE_b + MSE_d + (wMSE_f) \tag{3.15}$$

Here, $w \in \mathbb{R}$ is a weighting placed on the physics loss function.

The training data, model architecture, and learning parameters are identical to those described in Chapter 3.2. The only modification in this experiment is the loss formulation, where a weighting factor is applied to the physics loss term as defined in Equation (3.15). In total, 54 PINN models were trained for every different pair-wise combination of k values between 0.003 and 3 and weight values, w between 0.01 and 100. The case where $w = 1$,is where no adjustment in weighting is applied and therefore all loss functions have equal emphasis during training. The case where $k = 0.003$ is when there is no model mismatch. A large range of weights was chosen because it is unknown which range of weights has the greatest effect on the loss functions. The performance of the models on 501 test points evenly spread out across the domain of $x \in [0,1]$ and $t \in [0,100]$ are shown by Figure 3.5.

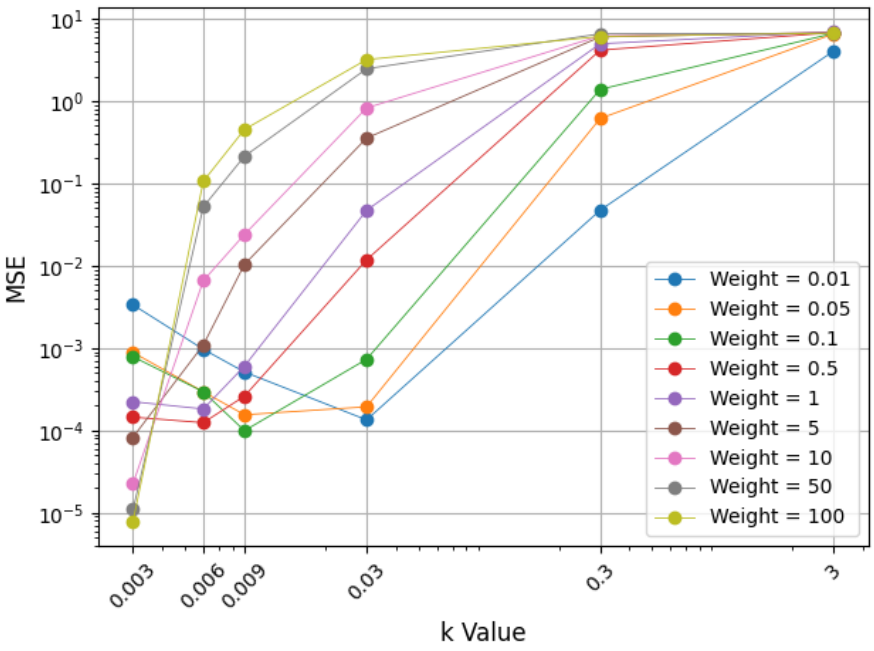


Figure 3.5: Log plot of MSE values for the k parameter PINNs with varying weights on the physics loss function.

Figure 3.5 shows that no weighting is applied to the physics loss term ($w = 1$), the PINN is still able to achieve MSE values below 10^{-3} when the k parameter is between 0.003 and 0.03. Between $k = 0.3$ and $k = 3$, there is a severe deterioration in the model performance. This could be because, from $k = 0.3$ the experimental data, the boundary points and initial points cannot compensate for the large difference in the k-parameter value from its ground truth.

When there is no mismatch ($k = 0.003$) and weightings are applied, the general trend is that increasing the weighting increases the performance, whereas decreasing it reduces it. However, with the introduction of model mismatch from $k = 0.006$ onward, the general trend is that increasing the weighting decreases the performance, whereas decreasing it increases the performance. This trend is less apparent as the mismatch increases, where it is seen in $k = 3$ that the weightings make little difference to the performance. The expectation is that reducing the weights leads to a smaller influence by the physics loss term during training, therefore, there is more influence by the data terms which are more accurate. In regions of the domain where there are only physics collocation points, little learning may be taking place and the model is defaulting to the prior initialisation of the model. If this is true, then some consideration needs to be placed on how the model is

initialised. In effect, the choice of initialisation would represent an implicit prior over the solution, and incorporating engineering knowledge into this prior would be more advantageous than the Glorot normal initialisation. For any future approaches that employ weighting schemes to mitigate model mismatch, should carefully consider the role of priors.

3.4 Evaluating physics-informed neural network performance with noisy experimental data and weighting

In the wider context of model mismatch, it is important to understand the difference between errors arising from noisy experimental data and those from model mismatch from inaccuracies in the governing equations. The following experiment investigates the simpler but related scenario in which the governing physics equation is assumed to be correct, but the experimental observations are corrupted by noise of varying magnitude.

The purpose of this experiment is to see how changing the weightings on the physics loss function, Equation (3.5), affects model performance. The hypothesis is that a small weighting on the physics loss function in the presence of large noise increases the performance of the model relative to a model with no weighting applied to the physics loss function. The weighting in this experiment is multiplied by the physics loss function as described by Equation (3.15).

The training data, model architecture, and learning parameters are identical to those described in Chapter 3.2. In this experiment, two modifications are made. Firstly, a weighting factor is applied to the physics loss term as defined in Equation 3.15. Secondly, noise drawn from a uniform distribution was added to 300 random points of the experimental data. For each model the limits of the uniform distribution were increased from 0 to 20 to replicate increasing levels of noise. An example of training data for experimental data with added noise from the distribution $U(-1,1)$, is shown in Figure 3.6 below.

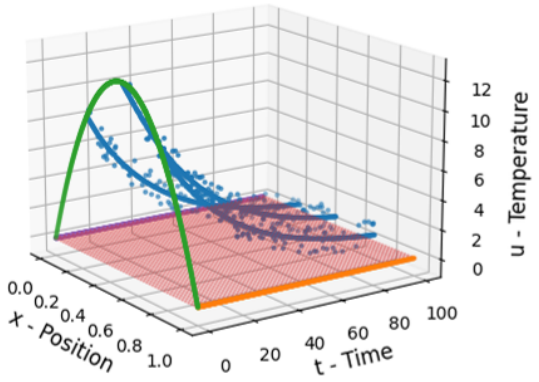


Figure 3.6: Training data with $N_0 = 501$ initial data points shown in green, $N_b = 1002$ boundary training points shown in orange and purple, $N_f = 2601$ physics collocations points shown in red and $N_d = 1503$ experimental points with added noise drawn from $U(-1,1)$ shown in blue.

To see the effect of weighting the physics loss function on PINNs and training them at varying levels of noise, 63 PINN models were trained in total. They were trained for every different pair-wise combination of noise, $U(-x,x)$, added to the experimental data with distribution limits, x between 0 and 20, and weight values, w between 0.01 and 100. The case where $w = 1$, is where no adjustment in weighting is applied and therefore all loss functions have equal emphasis during training. 'None' refers to no noise added to the model. The performance of the models on 501 test points evenly spread out across the domain of $x \in [0,1]$ and $t \in [0,100]$ is shown in Figure 3.7 below.

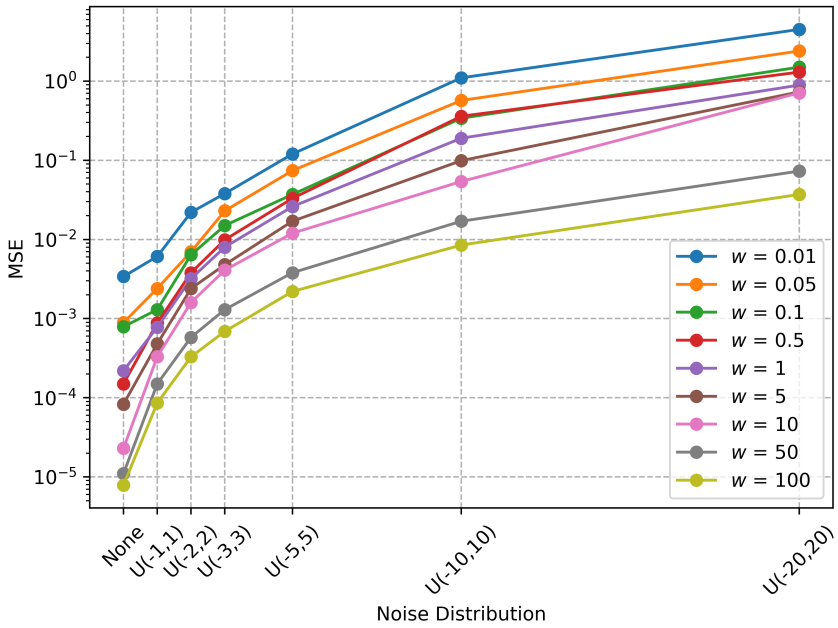


Figure 3.7: Log plot of MSE values for the PINNs trained with varying levels of noise with varying weights on the physics loss function.

Figure 3.7 shows that when no weighting changes are made ($w = 1$), there is reasonable performance between no added noise and noise from the distribution $U(-5,5)$. Thereafter, the model shows a large degradation in performance in the noise distributions $U(-10,10)$ and $U(-20,20)$.

At every level of noise, Figure 3.7 shows that increasing the weights reduces the MSE and decreasing the weights increases the MSE. At $U(-20,20)$ the models with weights of 50 and 100 have a much smaller MSE than the models with smaller weights. This could indicate that at higher levels of noise, large weights on the physics loss function make a greater difference to the performance of the model. The results are indicative that adding higher weights to the physics loss, assuming correct governing equations, can increase the robustness to noisy data.

3.4.1 Conclusions and future work

The detrimental effect of model mismatch on model performance was demonstrated in the 1-D heat equation example. Here, it was shown that an increasing coefficient mismatch can quickly cause a deterioration of the model performance. Reducing the weightings on the physics loss terms can mitigate the effect of model mismatch. However, this potentially causes the model to rely on the

data or the prior initialisation of the model. Where this initialisation is not close to the true solution, it can potentially worsen the performance. Furthermore, despite the effectiveness of the weightings on the model mismatch, applying the weightings on the physics loss function in this fashion is unprincipled and unreasoned.

Weighting schemes on the physics loss functions can be interpreted as trading off the influence of the physics term and other sources of information. In the initial experiment, the trade-off was between the experimental data and model initialisation. To some extent, these can be viewed as priors. If engineering knowledge about the process is available, it may therefore be preferable for the network initialisation to represent this knowledge, rather than relying on the Glorot normal initialisation. When there is uncertainty in the governing equations, an alternative model, for example, a pre-trained model, may be a more sensible initialisation. However, NNs do not necessarily retain pre-trained knowledge during transfer learning.

This motivates a paradigm shift towards MF modelling, where prior information and data of differing reliability can be fused in a principled way. MF modelling approaches enable for PIML methods to still be implemented, but provide an avenue for leveraging multiple information sources with imperfect physics. In Chapters 4 and 6, novel MF modelling methods are developed that involve weighting schemes between different information sources with different fidelities. Both of which have Bayesian problem formulations which allow for priors to be placed on the datasets.

Future work, beyond the scope of this PhD, includes a more detailed investigation into the interplay between PINN robustness to noise and weightings on the physics loss term, assuming perfect physics. Furthermore, it would be valuable to develop principled ways of tuning the weightings according to the quality of the data, potentially allowing different regions of the domain to be weighted according to the level of noise.

Chapter 4

Multi-fidelity modelling: a maximum a posteriori approach

The experimental conclusions of the previous chapter provide a segue to the problem of how model mismatch can be mitigated when conducting Physics Informed Machine Learning (PIML). To review, the forward problem described in Chapter 3.1 assumed some mismatch between the data from the true underlying process and a physics equation. Physics-Informed Neural Networks (PINNs) models were trained and evaluated for different amounts of mismatch in the problem set-up; mismatch was artificially generated by changing the coefficients on the physics equation by increasing or decreasing the coefficients away from their true values. Increased mismatch was shown to have a detrimental effect on the accuracy of the predictive output. However, mismatch could be mitigated via weights placed on the loss functions, rebalancing the amount of trust placed in the physics equation and the true data. Although this approach was effective, certain issues needed to be resolved in order to develop a formal approach to handling mismatch in PIML. Firstly, the weightings were not placed in a principled manner established in mathematical rigour. Furthermore, reducing the weightings on the physics loss amounted to the model defaulting to the initialised model of the Neural Network (NN), of which the behaviour is difficult to control. Additionally, there was no quantification of the uncertainty in the predictive outputs that could incorporate prior engineering knowledge about the mismatch of the model. As highlighted in Chapter 2.2, quantifying the uncertainty in predictive models used in Digital Twins (DTs), for example, can improve stakeholder trust in their predictive outputs for decision making. Finally, the PINNs framework assumes that the

governing physical laws are correct and fully specified, and that they can be enforced as a residual constraint during training. However, in more complex systems, there may be difficulties in embedding the physical properties of a system, for example, component efficiencies or calibration factors, that are not easily captured in a residual equation. In contrast, simulation models may capture richer physical properties, in addition to possibly already being available as design assets.

In order to develop a formal approach of PIML that addresses these issues, a paradigm shift was implemented from PINNs to Multi-Fidelity (MF) modelling. Via these frameworks, models are not enforced as immutable constraints but instead are treated as imperfect surrogates, which can leverage many different information sources of varying fidelity. This can include data generated from simulation-based models which can encompass physical properties not easily captured in residual equations, but also be generated for arbitrary inputs across a broad range. Therefore, although the methodology has changed from a focus on PINNs to MF modelling, the work is situated firmly within the PIML framework. In this chapter, it is assumed that there is mismatch between a Low-Fidelity (Lo-Fi) and a High-Fidelity (Hi-Fi) dataset, both of which describe the true underlying process. The Lo-Fi dataset may be generated from a physics-based model, whereas the Hi-Fi dataset may be assumed to originate from observations of the true system for which a DT is required. As illustrated in Chapter 2, MF modelling approaches also have the capacity to address challenges in building DTs, namely data sparsity, model mismatch, and aspects of the trust paradigm, i.e. transparency, interpretability, and performance.

Chapter 2 highlighted the gaps in the literature for MF modelling methods with the capacity to quantify mismatch uncertainty. Firstly, existing methods rely on Monte Carlo Markov Chain (MCMC) sampling or are based on Gaussian Process (GP) regression, both of which are computationally expensive at training and test time. Furthermore, GP-based approaches have scalability issues and can struggle in big-data regimes, which may be unsuitable for DT applications.

In this chapter, a MF modelling approach is developed using a Bayesian problem formulation that provides a principled approach to place weightings between Lo-Fi and Hi-Fi datasets when learning. The Bayesian problem formulation sets up a framework to capture mismatch and aleatoric uncertainty quantification on the predictive outputs. The MF modelling approach is trained using Maximum a Posteriori (MAP) loss functions obtained from the Bayesian problem formulation and applies a learned correction to the predictive output. The nature of the correction is a choice made

by the engineer, and, with a suitable choice in parametrisation, can lead to improved accuracy in interpolative and extrapolative regions of the domain where there are sparse Hi-Fi data.

This chapter begins with defining the Bayesian problem formulation and problem setup. Then, the methodology is outlined for training general parametrised models of the distributional parameters stated in the problem set-up. Finally, the approach is implemented on an adapted Mean Variance Estimation (MVE) network and evaluated on both synthetic and realistic engineering case studies. Performance is compared with baseline models trained exclusively on Hi-Fi data. The key contributions of this chapter can be summarised below:

- The development of a unique Bayesian problem formulation for datasets with different fidelities, allowing engineers to place prior beliefs on the mismatch between data sources when undertaking MF modelling. The Bayesian approach sets up a reasoned method for making trade-offs between how much a MF model learns from datasets with different fidelity for a particular prediction.
- The design of a MF model using an adapted MVE network[33], trained using MAP loss functions derived from the Bayesian problem formulation. This NN-based approach enhances scalability.
- The inclusion of a learned correction to the predictive output of the MF model, included in the Bayesian problem formulation. The nature of the correction is engineer-driven and with a well-chosen parametrisation can lead to improved accuracy, particularly in sparse Hi-Fi data regions.
- A systematic approach for setting hyper-parameters within the method, outlining key trade-offs to consider in different scenarios.

4.1 Bayesian problem formulation

This section outlines a Bayesian problem formulation from which the minimisation objective used to train the MF model is specified. Firstly, the subspaces on which the Hi-Fi and Lo-Fi data lie and their relationship with one another are defined. Next, the high level problem setup is defined and

the underlying assumptions of the Bayesian framework are stated. Finally, the general theory for variational inference which is used here to approximate the intractable posterior is outlined, before leveraging this theory to train the MF models.

4.1.1 Problem set-up

Let $X \subset \mathbb{R}^{d_x}$ be the input domain where $d_x \in \mathbb{N}$, and let $A \subseteq X$ and $B \subseteq A$ be subspaces. A Lo-Fi dataset is defined as $\mathcal{D}_{Y_L} = \{(x_{L_1}, y_{L_1}), \ldots, (x_{L_n}, y_{L_n}) \,|\, x_{L_i} \in A\}$ and a Hi-Fi dataset is defined as $\mathcal{D}_{Y_H} = \{(x_{H_1}, y_{H_1}), \ldots, (x_{H_m}, y_{H_m}) \,|\, x_{H_i} \in B\}$. For a given input x, Random Variables (RVs) on the Hi-Fi and Lo-Fi outputs are defined as

$$Y_H(x) = \mu_H(x) + \varepsilon_H(x) \tag{4.1}$$

$$Y_L(x) = \mu_L(x) + \varepsilon_L(x) \tag{4.2}$$

where $\varepsilon_H(x) \sim \mathcal{N}(0, \sigma_H^2(x))$ and $\varepsilon_L(x) \sim \mathcal{N}(0, \sigma_L^2(x))$. Therefore $Y_H(x) \sim \mathcal{N}(\mu_H(x), \sigma_H^2(x)) \equiv \mathcal{N}(\mu_{Y_H}(x), \sigma_{Y_H}^2(x))$ and $Y_L(x) \sim \mathcal{N}(\mu_L(x), \sigma_L^2(x)) \equiv \mathcal{N}(\mu_{Y_L}(x), \sigma_{Y_L}^2(x))$.

An estimate for an additive correction to $Y_L(x)$ is defined by

$$D(x) = \mu_H(x) - \mu_L(x) + \varepsilon_D(x) \tag{4.3}$$

where $\varepsilon_D(x) \sim \mathcal{N}(0, \sigma_D^2(x))$. $D(x)$ is therefore stochastic and $D(x) \sim \mathcal{N}(\mu_D(x), \sigma_D^2(x))$, where $\mu_D(x) = \mu_H(x) - \mu_L(x)$. Consequently, $D(x)$ is conditionally Gaussian given $\mu_H(x)$ and $\mu_L(x)$.

$D(x)$ is introduced to allow the model to provide a mechanism for generalising into regions of data sparsity. Restricted parameterisations are placed on $\mu_D(x)$ to reflect modelling assumptions.

4.1.2 Variational inference theory

It is important to distinguish this approach from the use of variational inference in modern deep learning (e.g., Variational Autoencoders), where it is typically used for amortized inference to match latent distributions. Here, classical Mean-Field Variational Inference (MFVI) is applied to approximate the posterior distribution over the global deterministic model parameters.

The general theory to find an approximation, $q(\Theta_{\text{All}})$, to the true posterior, $p(\Theta_{\text{All}}|X)$, is outlined, where Θ_{All} is the set of distributional parameters and X is the data. This approach is obtained from Bishop et al.[96] where a more detailed derivation can be found. This approach is adopted because the posterior distribution is analytically intractable and cannot be expressed in closed form. The model evidence $p(X)$ can be expressed as

$$\ln p(X) = \mathcal{L}(q) + KL(q||p) \tag{4.4}$$

where

$$\mathcal{L}(q) = \int q(\Theta_{\text{All}}) \ln \left\{ \frac{p(X, \Theta_{\text{All}})}{q(\Theta_{\text{All}})} \right\} d\Theta_{\text{All}} \tag{4.5}$$

$$KL(q||p) = -\int q(\Theta_{\text{All}}) \ln \left\{ \frac{p(\Theta_{\text{All}}|X)}{q(\Theta_{\text{All}})} \right\} d\Theta_{\text{All}} \tag{4.6}$$

Here, $KL(q||p)$ is the Kullback-Leibler (KL) divergence between the approximate posterior $q(\Theta_{\text{All}})$ and the true posterior $p(\Theta_{\text{All}}|X)$. Since the KL term is always non-negative, it follows that $\mathcal{L}(q) \leq \ln p(X)$. Therefore, $\mathcal{L}(q)$ acts as a lower bound to the log model evidence, often referred to as the Evidence Lower Bound (ELBO). Because $\ln p(X)$ is constant with respect to q, maximising the ELBO, $\mathcal{L}(q)$, is equivalent to minimising the KL divergence between the approximate and true posterior.

A restriction is placed on the functional form of $q(\Theta_{\text{All}})$ to make the optimisation tractable. Mean-field approximation is employed, assuming that $q(\Theta_{\text{All}})$ factorises as $q(\Theta_{\text{All}}) = \prod^{M} q_i(\theta_i)$, where the set of variables Θ_{All} is split into M subsets denoted as θ_i. Substituting this factorisation into Equation (4.5) gives

$$\mathcal{L}(q) = \int \prod_i q_i \left\{ \ln p\left(X, \Theta_{\text{All}}\right) - \sum_i \ln q_i \right\} d\Theta_{\text{All}} \tag{4.7}$$

$$= \int q_j \left\{ \int \ln p\left(X, \Theta_{\text{All}}\right) \prod_{i \neq j} q_i d\theta_i \right\} d\theta_j - \int q_j \ln q_j d\theta_j + \text{const} \tag{4.8}$$

$$= \int q_j \ln \tilde{p}\left(X, \theta_j\right) d\theta_j - \int q_j \ln q_j d\theta_j + \text{const} \tag{4.9}$$

where a new distribution $\ln \tilde{p}\left(X, \Theta_j\right)$ is defined by the relation

$$\ln \tilde{p}\left(X, \theta_j\right) = \mathbb{E}_{i \neq j}\left[\ln p\left(X, \Theta_{\text{All}}\right)\right] + \text{const} \tag{4.10}$$

Here, the notation $\mathbb{E}_{i\neq j}[\dots]$ denotes an expectation with respect to the q distributions over all variables θ_i for $i \neq j$, so that

$$\mathbb{E}_{i\neq j}\left[\ln p\left(X,\Theta_{\text{All}}\right)\right] = \int \ln p\left(X,\Theta_{\text{All}}\right) \prod_{i\neq j} q_i d\theta_i \tag{4.11}$$

Suppose all $\{q_{i\neq j}\}$ is kept fixed and maximise $\mathcal{L}(q)$ in Equation (4.9) with respect to all possible forms for the distribution $q_j(\theta_j)$. This is done by recognising that Equation (4.9) takes the form of a negative KL divergence between $q_j(\theta_j)$ and $\tilde{p}\left(X,\theta_j\right)$. Thus maximising Equation (4.9) is equivalent to minimising this KL divergence. The minimum occurs when $q_j = \tilde{p}\left(X,\theta_j\right)$. Thus, a general expression for the optimal solution, $q_j^*(\theta_j)$, is given by

$$\ln q_j^*(\theta_j) = \mathbb{E}_{i\neq j}\left[\ln p\left(X,\Theta_{\text{All}}\right)\right] + \text{const} \tag{4.12}$$

Note that 'const' refers to values that do not depend on one of the factors $q_j(\theta_j)$, therefore during optimisation, the other factors can be ignored. See Appendix A for the expanded derivation of Equation (4.5).

The set of equations given by Equation (4.12) for $j = 1,...,M$ represents a set of consistency conditions for the maximum of the lower bound subject to the mean-field factorisation constraint. They do not represent an explicit solution because each expectation depends on the other factors $q_i(\theta_i)$. A consistent solution is therefore achieved via Coordinate Ascent Variational Inference (CAVI).

The CAVI algorithm begins by initialising all factors q_i and iteratively updating each factor $q_j(\theta_j)$ in turn until the ELBO converges [97]. While Equation (4.12) provides the optimal functional form, in practice, these factors are constrained to specific parametric families and only require the optimal parameters to fit the approximate posterior. In the next section, the joint distribution is defined for the problem and the CAVI algorithm is applied to outline a method to fit the distributional parameters to the approximate posterior.

4.1.3 Fitting the distributional parameters to the approximate posterior

The joint distribution for this problem is described by Equation (4.13) which is derived from the graphical model in Figure 4.1. A MAP estimate of the distributional parameters could be taken by directly optimising the joint distribution. However, the high dimensionality of the parameter space

makes direct joint optimisation highly susceptible to local minima. To avoid this, the mean-field variational approach is applied as outlined in Section 4.1.2. By factorising the posterior into disjoint parameter subsets, simpler objectives can be isolated and iteratively fit. For clarity, the dependence on x in the notation for all distributional parameters throughout this section is omitted.

$$
\begin{aligned}
&p(\mu_{Y_H}, \sigma^2_{Y_H}, \mu_{Y_L}, \sigma^2_{Y_L}, \mu_D, \sigma^2_D | \mathcal{D}_{Y_H}, \mathcal{D}_{Y_L}) \propto \\
&p(\mathcal{D}_{Y_H}, \mathcal{D}_{Y_L} | \mu_{Y_H}, \sigma^2_{Y_H}, \mu_{Y_L}, \sigma^2_{Y_L}, \mu_D, \sigma^2_D) \cdot p(\mu_{Y_H}, \sigma^2_{Y_H}, \mu_{Y_L}, \sigma^2_{Y_L}, \mu_D, \sigma^2_D) = \\
&p(\sigma^2_{Y_L}) \cdot p(\mu_{Y_L} | \sigma^2_{Y_L}) \cdot p(\mathcal{D}_{Y_L} | \mu_{Y_L}, \sigma^2_{Y_L}) \cdot \\
&p(\sigma^2_D) \cdot p(\mu_D | \mu_{Y_L}, \mu_{Y_H}, \sigma^2_D) \cdot \\
&p(\sigma^2_{Y_H}) \cdot p(\mu_{Y_H} | \sigma^2_{Y_H}) \cdot p(\mathcal{D}_{Y_H} | \mu_{Y_H}, \sigma^2_{Y_H})
\end{aligned}
\tag{4.13}
$$

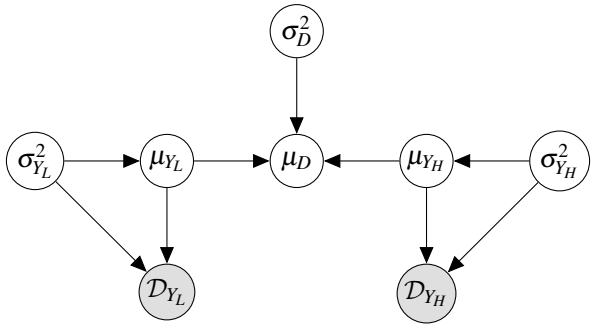


Figure 4.1: Graphical model of posterior distribution.

The form of the approximate joint distribution is restricted by assuming it factorises with respect to subsets of the parameter set

$$
q(\tilde{H}, \tilde{L}, \tilde{D}) \approx q_{\tilde{H}}(\tilde{H}) \cdot q_{\tilde{L}}(\tilde{L}) \cdot q_{\tilde{D}}(\tilde{D}) \tag{4.14}
$$

where the parameter groups are defined by $\tilde{H} = \{\mu_{Y_H}, \sigma^2_{Y_H}\}$, $\tilde{L} = \{\mu_{Y_L}, \sigma^2_{Y_L}\}$, $\tilde{D} = \{\mu_D, \sigma^2_D\}$. Using

Equations (4.12) and (4.13) the optimal forms for the three factors are found

$$\ln q^*_{\tilde{H}}(\tilde{H}) = \mathbb{E}_{\tilde{D},\tilde{L}}[\ln p(\mu_D|\mu_{Y_L},\mu_{Y_H},\sigma^2_D) + \ln p(\sigma^2_{Y_H}) + \ln p(\mu_{Y_H}|\sigma^2_{Y_H}) + \ln p(\mathcal{D}_{Y_H}|\mu_{Y_H},\sigma^2_{Y_H})] + \text{const} \tag{4.15}$$

$$\ln q^*_{\tilde{L}}(\tilde{L}) = \mathbb{E}_{\tilde{D},\tilde{H}}[\ln p(\mu_D|\mu_{Y_L},\mu_{Y_H},\sigma^2_D) + \ln p(\sigma^2_{Y_L}) + \ln p(\mu_{Y_L}|\sigma^2_{Y_L}) + \ln p(\mathcal{D}_{Y_L}|\mu_{Y_L},\sigma^2_{Y_L})] + \text{const} \tag{4.16}$$

$$\ln q^*_{\tilde{D}}(\tilde{D}) = \mathbb{E}_{\tilde{L},\tilde{H}}[\ln p(\mu_D|\mu_{Y_L},\mu_{Y_H},\sigma^2_D) + \ln p(\sigma^2_D)] + \text{const} \tag{4.17}$$

In this context, the 'const' in each equation refers to parts of the joint distribution which do not depend on any of the other parameter groups, i.e. in Equation (4.15), 'const' is equal to the distributions that do not depend on the parameter within the parameter groups $\tilde{L}$ or $\tilde{D}$.

Flat priors are placed on $p(\sigma^2_{Y_L})$, $p(\mu_{Y_L}|\sigma^2_{Y_L})$, $p(\sigma^2_D)$, and $p(\mu_{Y_H}|\sigma^2_{Y_H})$. If non-flat priors are placed on these distributional parameters, the optimisation process increases in difficulty. The problem formulation is therefore simplified and priors are placed on distributional parameters that there is information about instead. Since they do not have an effect in the optimisation scheme, the distributions with flat priors are ignored. Informative priors are placed on $p(\mu_D|\mu_{Y_L},\mu_{Y_H},\sigma^2_D) \sim \mathcal{N}(\mu_{Y_H} - \mu_{Y_L},\sigma^2_D)$ and $p(\sigma^2_{Y_H}) \sim \text{IG}(\alpha_0,\beta_0)$ where IG corresponds to an Inverse Gamma distribution. An IG distribution is chosen since it is a conjugate to the Gaussian distribution. Due to the low variance of $\sigma^2_{Y_L}$, the influence of the prior term $\ln p(\mu_D|\mu_{Y_L},\mu_{Y_H},\sigma^2_D)$ in Equation (4.16) is minimal and can effectively be ignored during optimisation. Under these assumptions, Equations (4.15)–(4.17) are approximated as

$$\ln q^*_{\tilde{H}}(\tilde{H}) \approx \mathbb{E}_{\tilde{D},\tilde{L}}[\ln p(\mu_D|\mu_{Y_L},\mu_{Y_H},\sigma^2_D) + \ln p(\sigma^2_{Y_H}) + \ln p(\mathcal{D}_{Y_H}|\mu_{Y_H},\sigma^2_{Y_H})] + \text{const} \tag{4.18}$$

$$\ln q^*_{\tilde{L}}(\tilde{L}) \approx \mathbb{E}_{\tilde{D},\tilde{H}}[\ln p(\mathcal{D}_{Y_L}|\mu_{Y_L},\sigma^2_{Y_L})] + \text{const} \tag{4.19}$$

$$\ln q^*_{\tilde{D}}(\tilde{D}) \approx \mathbb{E}_{\tilde{L},\tilde{H}}[\ln p(\mu_D|\mu_{Y_L},\mu_{Y_H},\sigma^2_D)] + \text{const} \tag{4.20}$$

Writing out the Probability Density Functions (PDF) in full, Equation (4.23) is obtained

$$\ln q^*_{\tilde{H}}(\tilde{H}) \approx \mathbb{E}_{\tilde{D},\tilde{L}}\Bigg[-\frac{1}{2}\frac{(\mu_D - (\mu_{Y_H} - \mu_{Y_L}))^2}{\sigma_D^2} - (\alpha_0 + 1)\ln(\sigma^2_{Y_H}) - \frac{\beta_0}{\sigma^2_{Y_H}} - \frac{n_{Y_H}}{2}\ln(\sigma^2_{Y_H}) - \frac{1}{2\sigma^2_{Y_H}}\sum_{i=1}^{n_{Y_H}}(y_{H,i} - \mu_{Y_H})^2\Bigg] + \text{const} \tag{4.21}$$

$$\ln q^*_{\tilde{L}}(\tilde{L}) \approx \mathbb{E}_{\tilde{D},\tilde{H}}\left[-\frac{n_{Y_L}}{2}\ln(\sigma^2_{Y_L}) - \frac{1}{2\sigma^2_{Y_L}}\sum_{i=1}^{n_{Y_L}}(y_{L,i} - \mu_{Y_L})^2\right] + \text{const} \tag{4.22}$$

$$\ln q^*_{\tilde{D}}(\tilde{D}) \approx \mathbb{E}_{\tilde{L},\tilde{H}}\left[-\frac{1}{2}\frac{(\mu_D - (\mu_{Y_H} - \mu_{Y_L}))^2}{\sigma_D^2}\right] + \text{const} \tag{4.23}$$

Here, n_{Y_H} and n_{Y_L} correspond to the number of Hi-Fi and Lo-Fi data points at a point x.

Having obtained the optimal functional forms for each factor, the approach diverges from a fully Bayesian treatment by making point estimates of the distributional parameters. A fully Bayesian approach would involve inferring the posterior distributions over all parameters. The choice is made to maintain computational tractability while mitigating the risk of local minima in the complex parameter space. Note that this choice is mathematically equivalent to restricting the variational family to Dirac delta distributions. The choice reduces the mean-field variational inference scheme to Block Coordinate Ascent for Maximum A Posteriori (MAP) estimation. Consequently, the expectations in Equations (4.21)–(4.23) simply evaluate to the functions at their current point estimates.

To fit the distributional parameters μ_{Y_H}, μ_{Y_L}, $\sigma^2_{Y_H}$, $\sigma^2_{Y_L}$, and μ_D, the optimal factors are maximised in turn with respect to their corresponding parameter block ($\tilde{H}$, $\tilde{L}$, $\tilde{D}$) while holding the other blocks constant.

In practice, this is implemented by taking the negative of the log-posterior approximations to create three distinct minimisation objectives. Taking the negative of the log PDFs results in a minimisation objective.

$$\mathcal{L}_{\tilde{H}} = \frac{1}{2}\frac{(\mu_D - (\mu_{Y_H} - \mu_{Y_L}))^2}{\sigma_D^2} + (\alpha_0 + 1)\ln(\sigma_{Y_H}^2) + \frac{\beta_0}{\sigma_{Y_H}^2} + \frac{n_{Y_H}}{2}\ln(\sigma_{Y_H}^2) + \frac{1}{2\sigma_{Y_H}^2}\sum_{i=1}^{n_{Y_H}}(y_{H,i} - \mu_{Y_H})^2 \tag{4.24}$$

$$\mathcal{L}_{\tilde{L}} = \frac{n_{Y_L}}{2}\ln(\sigma_{Y_L}^2) + \frac{1}{2\sigma_{Y_L}^2}\sum_{i=1}^{n_{Y_L}}(y_{L,i} - \mu_{Y_L})^2 \tag{4.25}$$

$$\mathcal{L}_{\tilde{D}} = + \frac{1}{2}\frac{(\mu_D - (\mu_{Y_H} - \mu_{Y_L}))^2}{\sigma_D^2} \tag{4.26}$$

Having derived the above loss functions, an alternative set of loss functions is formulated by replacing the likelihood $p(\mathcal{D}_{Y_H} \mid \mu_{Y_H}, \sigma_{Y_H}^2)$ in Equation (4.13) with a weighted likelihood $p(\mathcal{D}_{Y_H} \mid \mu_{Y_H}, \sigma_{Y_H}^2, \lambda_{Y_H})$, defined as

$$p(\mathcal{D}_{Y_H} \mid \mu_{Y_H}, \sigma_{Y_H}^2, \lambda_{Y_H}) = \prod_{i=1}^{n_{Y_H}} \left[\frac{1}{\sqrt{2\pi\sigma_{Y_H}^2}} \exp\left(-\frac{(x - \mu_{Y_H})^2}{2\sigma_{Y_H}^2} \right) \right]^{\lambda_{Y_H}} \tag{4.27}$$

Here, $\lambda_{Y_H} \in \mathbb{R}^+$ is a weighting parameter that controls the relative importance assigned to the Hi-Fi dataset. Since λ_{Y_H} is constant across all observations, it effectively adjusts the influence of the Hi-Fi data relative to the Lo-Fi data. The only loss function that changes is Equation (4.24), which becomes

$$\mathcal{L}_{\tilde{H}_\lambda} = \frac{1}{2}\frac{(\mu_D - (\mu_{Y_H} - \mu_{Y_L}))^2}{\sigma_D^2} + (\alpha_0 + 1)\ln(\sigma_{Y_H}^2) + \frac{\beta_0}{\sigma_{Y_H}^2} + \frac{\lambda_{Y_H} n_{Y_H}}{2}\ln(\sigma_{Y_H}^2) + \frac{\lambda_{Y_H}}{2\sigma_{Y_H}^2}\sum_{i=1}^{n_{Y_H}}(y_{H,i} - \mu_{Y_H})^2. \tag{4.28}$$

The approach retains the core benefits of the Bayesian formulation through the careful choice of priors on the distributional parameters. The priors bias the learning such that the distributional parameters are learned from a weighting between the data and prior. Specifically, placing a prior on μ_D biases the learning of μ_{Y_H} such that when there is Hi-Fi data available, μ_{Y_H} is fit on a weighting of the Hi-Fi data and a prior value $\mu_{Y_L} + \mu_D$. Placing a prior on $\sigma_{Y_H}^2$ biases learning of $\sigma_{Y_H}^2$ such that when there is Hi-Fi data available, $\sigma_{Y_H}^2$ fits between a weighting of the Hi-Fi data uncertainty and a prior mismatch uncertainty. In both cases, the weighting is informed by the engineer's choice of

hyper-parameters. The mismatch uncertainty is expressed through $\sigma^2_{Y_H}$. This concludes the unique Bayesian problem formulation, which successfully enables prior beliefs on the mismatch between data sources while remaining computationally efficient to optimise.

4.2 Methodology

In this section, an optimisation algorithm is outlined for when the distributional parameters are represented by surrogate models with their own parametrisations. Finally, a general strategy is outlined for choosing hyper-parameters before specifying the MF model architecture used in the experiments.

In this section, the dependence on x is reintroduced in the notation for the distributional parameters. The trainable parametrised models are denoted by $\mu_{Y_H}(x;\theta_{\mu_{Y_H}})$, $\sigma^2_{Y_H}(x;\theta_{\sigma^2_{Y_H}})$, $\mu_{Y_L}(x;\theta_{\mu_{Y_L}})$, and $\sigma^2_{Y_L}(x;\theta_{\sigma^2_{Y_L}})$. A parametrisation is also introduced for $\mu_D(x)$, making it a hyper-parameter learned from the data. This parametrised model depends on both x and $\mu_{Y_L}(x)$, and is defined as $\mu_D(x,\mu_{Y_L}(x);\theta_{\mu_D})$. The full set of trainable parameters is therefore given by $\Theta = \{\theta_{\mu_{Y_H}},\theta_{\sigma^2_{Y_H}},\theta_{\mu_{Y_L}},\theta_{\sigma^2_{Y_L}},\theta_{\mu_D}\}$. A dataset $\mathcal{D}_{Y_{\text{Pseudo}}} = \{(x_i,y_i)\}_{i=1}^N$ is introduced, where each $x_i \in \mathcal{D}_{Y_L}$ and each corresponding y_i is a noisy observation with high variance, i.e.

$$y_i = f(x_i) + \varepsilon_i, \quad \varepsilon_i \sim \mathcal{N}(0,\sigma^2), \quad \text{with } \sigma^2 \to \infty. \tag{4.29}$$

Finally, an extended dataset is introduced, $\mathcal{D}_{\hat{Y}_H} = \mathcal{D}_{Y_H} \cup \mathcal{D}_{Y_{\text{Pseudo}}}$, where $\mathcal{D}_{Y_{\text{Pseudo}}}$ consists of pseudo-observations that enable the evaluation of $\mathcal{L}_{\tilde{H}_\lambda}$ across the entire domain. Owing to the high variance associated with each y_i, the likelihood term $p(\mathcal{D}_{Y_H} \mid \mu_{Y_H},\sigma^2_{Y_H},\lambda_{Y_H})$ becomes negligible. Consequently, $\mu_{Y_H}(x;\theta_{\mu_{Y_H}})$ and $\sigma^2_{Y_H}(x;\theta_{\sigma^2_{Y_H}})$ are effectively learned from the prior alone. The optimisation algorithm is defined by Algorithm 1.

Algorithm 1 MF-MVE-NN learning algorithm (vectorised, batch-based)

Require: Learning rate η

Require: High-fidelity dataset $\mathcal{D}_{Y_H}$ of mini-batches $\{\mathbf{x}_{\text{batch}}, \mathbf{y}_{\text{batch}}\}$

Require: Low-fidelity dataset $\mathcal{D}_{Y_L}$ of mini-batches $\{\mathbf{x}_{\text{batch}}, \mathbf{y}_{\text{batch}}\}$

Require: Dataset $\mathcal{D}_{\hat{Y}_H}$ of mini-batches $\{\mathbf{x}_{\text{batch}}, \mathbf{y}_{\text{batch}}\}$

Require: Parametrised models:

$$\mu_{Y_H}(\mathbf{x};\theta_{\mu_{Y_H}}),\ \sigma^2_{Y_H}(\mathbf{x};\theta_{\sigma^2_{Y_H}}),\ \mu_{Y_L}(\mathbf{x};\theta_{\mu_{Y_L}}),\ \sigma^2_{Y_L}(\mathbf{x};\theta_{\sigma^2_{Y_L}}),\ \mu_D(\mathbf{x},\mu_{Y_L}(\mathbf{x});\theta_{\mu_D})$$

Require: Hyper-parameters: $\lambda_{Y_H}, \alpha_0, \beta_0, \sigma^2_D$

Initialise parameters $\theta_{\mu_{Y_H}}, \theta_{\sigma^2_{Y_H}}, \theta_{\mu_{Y_L}}, \theta_{\sigma^2_{Y_L}}, \theta_{\mu_D}$

Initialise $n_{Y_H} = 1$ and $n_{Y_L} = 1$. ▷ Using the point-wise versions of the losses.

Let B correspond to the mini-batch size

while not converged **do**

  for all $\{\mathbf{x}_{\text{batch}}, \mathbf{y}_{\text{batch}}\} \in \mathcal{D}_{Y_L}$ **do**

    Compute vector outputs:

$$\mu_{Y_L}(\mathbf{x}_{\text{batch}};\theta_{\mu_{Y_L}}),\ \sigma^2_{Y_L}(\mathbf{x}_{\text{batch}};\theta_{\sigma^2_{Y_L}})$$

    Compute point-wise loss vector $\mathcal{L}_{\tilde{L}}$

    Compute mean loss:

$$\bar{\mathcal{L}}_{\tilde{L}} = \frac{1}{B}\sum_{i=1}^{B}\mathcal{L}_{\tilde{L},i}$$

    Update parameters $\theta_{\mu_{Y_L}}, \theta_{\sigma^2_{Y_L}}$ by gradient descent on $\bar{\mathcal{L}}_{\tilde{L}}$

  end for

  for all $\{\mathbf{x}_{\text{batch}}, \mathbf{y}_{\text{batch}}\} \in \mathcal{D}_{\hat{Y}_H}$ **do**

    Compute vector outputs:

$$\mu_{Y_H}(\mathbf{x}_{\text{batch}};\theta_{\mu_{Y_H}}),\ \sigma^2_{Y_H}(\mathbf{x}_{\text{batch}};\theta_{\sigma^2_{Y_H}}),\ \mu_{Y_L}(\mathbf{x}_{\text{batch}};\theta_{\mu_{Y_L}}),\ \sigma^2_{Y_L}(\mathbf{x}_{\text{batch}};\theta_{\sigma^2_{Y_L}}),\ \mu_D(\mathbf{x}_{\text{batch}},\mu_{Y_L}(\mathbf{x}_{\text{batch}});\theta_{\mu_D})$$

    Compute point-wise loss vector $\mathcal{L}_{\tilde{H}_\lambda}$

    Compute mean loss:

$$\bar{\mathcal{L}}_{\tilde{H}_\lambda} = \frac{1}{B}\sum_{i=1}^{B}\mathcal{L}_{\tilde{H}_\lambda,i}$$

    Update parameters $\theta_{\mu_{Y_H}}, \theta_{\sigma^2_{Y_H}}$ by gradient descent on $\bar{\mathcal{L}}_{\tilde{H}_\lambda}$

  end for

  for all $\{\mathbf{x}_{\text{batch}}, \mathbf{y}_{\text{batch}}\} \in \mathcal{D}_{Y_H}$ **do**

    Compute vector outputs:

$$\mu_{Y_H}(\mathbf{x}_{\text{batch}};\theta_{\mu_{Y_H}}),\ \mu_{Y_L}(\mathbf{x}_{\text{batch}};\theta_{\mu_{Y_L}}),\ \mu_D(\mathbf{x}_{\text{batch}},\mu_{Y_L}(\mathbf{x}_{\text{batch}});\theta_{\mu_D})$$

    Compute point-wise loss vector $\mathcal{L}_{\tilde{D}}$

    Compute mean loss:

$$\bar{\mathcal{L}}_{\tilde{D}} = \frac{1}{B}\sum_{i=1}^{B}\mathcal{L}_{\tilde{D},i}$$

    Update parameters θ_{μ_D} by gradient descent on $\bar{\mathcal{L}}_{\tilde{D}}$

  end for

end while

It can be seen that Algorithm 1 directly follows the variational approach outlined in Section

4.1.2. Lines 5 to 10 minimise Equation (4.25) for a choice of $\theta_{\mu_{Y_L}}$ and $\theta_{\sigma^2_{Y_L}}$. Lines 11 to 16 minimise Equation (4.28) for a choice of $\theta_{\mu_{Y_H}}$ and $\theta_{\sigma^2_{Y_H}}$. Lines 17 to 22 minimise Equation (4.26) for a choice of $\theta_{\mu_{Y_D}}$. Each of these optimisations represents a MAP approach. This repeats cyclically until convergence is reached.

4.2.1 Hyper-parameter selection

A careful choice of hyper-parameters $(\lambda_{Y_H}, \alpha_0, \beta_0, \sigma_D^2)$ is crucial, as different choices lead to different solutions. It is informative to calculate the analytical form of the gradients that are used to update the surrogate models. Finding the minima of these gradients helps to clarify the contribution of the hyper-parameters and the data. This leads to a method for hyper-parameter selection. The analytical solutions are derived for the minima of each loss function (Equations (4.25), (4.26) and (4.28)) with respect to each of the distributional parameters by taking the partial derivative. The minima of these partial derivatives represent the optimal choice of the distributional parameter given the choice of hyper-parameters. There are different loss functions depending on whether there is overlap in the Hi-Fi and Lo-Fi data. There are therefore two cases to consider when training the model.

1. **Overlap Case:** A single pair of Lo-Fi data points, $\{x_L, y_L\} \in \mathcal{D}_{Y_L}$ and a single pair of Hi-Fi data points $\{x_H, y_H\} \in \mathcal{D}_{Y_H}$ overlaps (i.e. given x_L there exists an x_H such that $|x_L - x_H| < \varepsilon$ for $\varepsilon > 0$). Additionally, $n_{Y_H} = 0$ and $n_{Y_L} = 1$.

2. **Non-Overlap Case:** Only a single pair of Lo-Fi data points $\{x_L, y_L\} \in \mathcal{D}_{Y_L}$, with no corresponding Hi-Fi data at the same location (i.e. given an x_L there does not exist an x_H such that $|x_L - x_H| < \varepsilon$ for $\varepsilon > 0$). Additionally, $n_{Y_H} = 0$ and $n_{Y_L} = 1$. There always exists points of $\mathcal{D}_{Y_{\text{Pseudo}}}$ that share the same input points x_L as Lo-Fi data.

In the overlap case, the point-wise derivatives are given as

$$\frac{\partial \mathcal{L}_{\tilde{H}_\lambda}}{\partial \mu_{Y_H}(x;\theta_{\mu_{Y_H}})} = 0 \implies \mu_{Y_H}(x;\theta_{\mu_{Y_H}}) = \frac{\lambda_{Y_H} y_H + \frac{\sigma^2_{Y_H}(x;\theta_{\sigma^2_{Y_H}})}{\sigma^2_D}\Big(\mu_D(x,\mu_{Y_L}(x);\theta_{\mu_D}) + \mu_{Y_L}(x;\theta_{\mu_{Y_L}})\Big)}{\lambda_{Y_H} + \frac{\sigma^2_{Y_H}(x;\theta_{\sigma^2_{Y_H}})}{\sigma^2_D}} \tag{4.30a}$$

$$\frac{\partial \mathcal{L}_{\tilde{H}}}{\partial \sigma^2_{Y_H}(x;\theta_{\sigma^2_{Y_H}})} = 0 \implies \sigma^2_{Y_H}(x;\theta_{\sigma^2_{Y_H}}) = \frac{\frac{\lambda_{Y_H}}{\alpha_0+1}(\mu_{Y_H}(x;\theta_{\mu_{Y_H}}) - y_H)^2 + 2\frac{\beta_0}{\alpha_0+1}}{\frac{\lambda_{Y_H}}{\alpha_0+1} + 2} \tag{4.30b}$$

$$\frac{\partial \mathcal{L}_{\tilde{L}}}{\partial \mu_{Y_L}(x;\theta_{\mu_{Y_L}})} = 0 \implies \mu_{Y_L}(x;\theta_{\mu_{Y_L}}) = y_L \tag{4.30c}$$

$$\frac{\partial \mathcal{L}_{\tilde{L}}}{\partial \sigma^2_{Y_L}(x;\theta_{\sigma^2_{Y_L}})} = 0 \implies \sigma^2_{Y_L}(x;\theta_{\sigma^2_{Y_L}}) = (\mu_{Y_L}(x;\theta_{\mu_{Y_L}}) - y_L)^2 \tag{4.30d}$$

$$\frac{\partial \mathcal{L}_{\tilde{D}}}{\partial \mu_D(x,\mu_{Y_L}(x);\theta_{\mu_D})} = 0 \implies \mu_D(x,\mu_{Y_L}(x);\theta_{\mu_D}) = \mu_{Y_H}(x;\theta_{\mu_{Y_H}}) - \mu_{Y_L}(x;\theta_{\mu_{Y_L}}) \tag{4.30e}$$

In the non-overlap case, the point-wise derivatives are given as

$$\frac{\partial \mathcal{L}_{\tilde{H}_\lambda}}{\partial \mu_{Y_H}(x;\theta_{\mu_{Y_H}})} = 0 \implies \mu_{Y_H}(x;\theta_{\mu_{Y_H}}) = \mu_D(x,\mu_{Y_L}(x);\theta_{\mu_D}) + \mu_{Y_L}(x;\theta_{\mu_{Y_L}}) \tag{4.31a}$$

$$\frac{\partial \mathcal{L}_{\tilde{H}_\lambda}}{\partial \sigma^2_{Y_H}(x;\theta_{\sigma^2_{Y_H}})} = 0 \implies \sigma^2_{Y_H}(x;\theta_{\sigma^2_{Y_H}}) = \frac{\beta_0}{\alpha_0+1} \tag{4.31b}$$

$$\frac{\partial \mathcal{L}_{\tilde{L}}}{\partial \mu_{Y_L}(x;\theta_{\mu_{Y_L}})} = 0 \implies \mu_{Y_L}(x;\theta_{\mu_{Y_L}}) = y_L \tag{4.31c}$$

$$\frac{\partial \mathcal{L}_{\tilde{L}}}{\partial \sigma^2_{Y_L}(x;\theta_{\sigma^2_{Y_L}})} = 0 \implies \sigma^2_{Y_L}(x;\theta_{\sigma^2_{Y_L}}) = \Big(\mu_{Y_L}(x;\theta_{\mu_{Y_L}}) - y_L\Big)^2 \tag{4.31d}$$

In the overlap case and the non-overlap case, $\mu_{Y_L}(x;\theta_{\mu_{Y_L}})$ learns directly from the Lo-Fi data and $\sigma^2_{Y_L}(x;\theta_{\sigma^2_{Y_L}})$ learns directly from squared error between the model prediction ($\mu_{Y_L}(x;\theta_{\mu_{Y_L}})$) and Lo-Fi data. This shows that the priors have no effect on learning the Lo-Fi model distribution parameters in any instance. In the overlap case, $\mu_{Y_H}(x;\theta_{\mu_{Y_H}})$ trades off between learning directly from the Hi-Fi data or $\Big(\mu_D(x,\mu_{Y_L}(x);\theta_{\mu_D}) + \mu_{Y_L}(x;\theta_{\mu_{Y_L}})\Big)$ and $\sigma^2_{Y_H}(x;\theta_{\sigma^2_{Y_H}})$ trades off between learning the squared error uncertainty $(\mu_{Y_H}(x) - y_H)^2$ and the prior uncertainty β_0. In the non-overlap case, $\mu_{Y_H}(x;\theta_{\mu_{Y_H}})$ is learnt directly from the Lo-Fi prediction ($\mu_{Y_L}(x;\theta_{\mu_{Y_L}})$) in addition to a learned offset, $(\mu_D(x,\mu_{Y_L}(x);\theta_{\mu_D}) + \mu_{Y_L}(x;\theta_{\mu_{Y_L}}))$ and $\sigma^2_{Y_H}(x;;\theta_{\sigma^2_{Y_H}})$ defaults to the prior $\Big(\frac{\beta_0}{\alpha_0+1}\Big)$.

Analysing coefficients on gradients

By examining Equations (4.30b) and (4.30a), coefficients on the data and prior terms that determine the balance or weighting between how much the model learns from the data versus how much it relies on the prior can be defined. These coefficients are given as

$$\text{Variance Data Coefficient} = \frac{\frac{\lambda_{Y_H}}{\alpha_0+1}}{\frac{\lambda_{Y_H}}{\alpha_0+1}+2} \tag{4.32a}$$

$$\text{Variance Prior Coefficient} = \frac{2}{\frac{\lambda_{Y_H}}{\alpha_0+1}+2} \tag{4.32b}$$

$$\text{Mu Data Coefficient} = \frac{\lambda_{Y_H}}{\lambda_{Y_H}+\frac{\sigma^2_{Y_H}(x;\theta_{\sigma^2_{Y_H}})}{\sigma^2_D}} \tag{4.32c}$$

$$\text{Mu Prior Coefficient} = \frac{\frac{\sigma^2_{Y_H}(x;\theta_{\sigma^2_{Y_H}})}{\sigma^2_D}}{\lambda_{Y_H}+\frac{\sigma^2_{Y_H}(x;\theta_{\sigma^2_{Y_H}})}{\sigma^2_D}} \tag{4.32d}$$

The relative weighting of data versus prior during learning is determined by taking the ratio of the data coefficient to the prior coefficient as shown by

$$\text{Variance Data-Prior Ratio} := r_{\sigma^2} = \frac{\lambda_{Y_H}}{2(\alpha_0+1)} \tag{4.33a}$$

$$\text{Mu Data-Prior Ratio} := r_{\mu} = \frac{\lambda_{Y_H}\sigma^2_D}{\sigma^2_{Y_H}(x;\theta_{\sigma^2_{Y_H}})} \tag{4.33b}$$

Setting the hyper-parameters

Hyper-parameter tuning amounts to the setting of the ratios (Equations (4.33a) and (4.33b)). By adjusting these ratios, the trade-off between learning from data and relying on the prior in overlap regions during training can be controlled. In principle, these hyper-parameters include α_0, β_0, n_{Y_H} and σ^2_D. However, two further hyper-parameters are defined for practical implementation.

First, the hyper-parameters α_0 and β_0 are chosen such that the resulting prior variance corresponds to the set mismatch uncertainty, defined by $\sigma^2_{Y_{H\text{prior}}}$. This value reflects the value of $\sigma^2_{Y_H}(x;\theta_{\sigma^2_{Y_H}})$ when Hi-Fi data is unavailable.

Second, although $\sigma^2_{Y_H}(x)$ is technically a distributional parameter, it appears in the mean ratio r_μ. If used directly during training, the effective weighting would change dynamically and uncontrollably throughout the optimisation. Therefore, to have a stable weighting on r_μ, the predictive variance is replaced by an upper bound. This is the maximum plausible predictive variance, considering the mismatch uncertainty and aleatoric uncertainty in the Hi-Fi data. This value is defined by $\sigma^2_{Y_{H\,\text{upper}}}$ and is treated as an additional hyper-parameter. This changes Equation (4.33b) to

$$\text{Mu data-prior ratio} := r_\mu = \frac{n_{Y_H}\sigma_D^2}{2\sigma^2_{Y_{H\,\text{upper}}}} \tag{4.34}$$

The full list of hyper-parameters to be set are α_0, β_0, λ_{Y_H}, σ_D^2, $\sigma^2_{Y_{H\,\text{upper}}}$, $\sigma^2_{Y_{H\,\text{prior}}}$. Below are the steps, in chronological order, for setting the hyper-parameters.

1. **Setting** $\sigma^2_{Y_{H\,\text{prior}}}$: This corresponds to the uncertainty in the sparse regions (no Hi-Fi data) of the domain. It reflects how well the prior model $\mu_{Y_L}(x;\theta_{\mu_{Y_L}}) + \mu_D(x,\mu_{Y_L}(x);\theta_{\mu_D})$ can approximate the ground truth in the sparse regions. It is therefore set based on prior engineering judgment regarding the expected magnitude of model mismatch.

2. **Setting** α_0: This is a free parameter where $\alpha_0 > 0$. For all experiments, this is set to $\alpha_0 = 1$.

3. **Setting** β_0: This is calculated using Equation (4.31b) and $\sigma^2_{Y_{H\,\text{prior}}}$. In the non-overlap case, where there is no Hi-Fi data, the predictive variance is equal to $\frac{\beta_0}{\alpha_0+1}$. Equating $\sigma^2_{Y_{H\,\text{prior}}} = \frac{\beta_0}{\alpha_0+1}$ and rearranging for β_0 gives us the value.

4. **Setting** r_{σ^2} **and** r_μ: Both ratios are equated and constrained such that $r_{\sigma^2} = r_\mu = r$ for a chosen value of r. The value of r corresponds to the trade-off between the Hi-Fi data and the prior model $\mu_{Y_L}(x;\theta_{\mu_{Y_L}}) + \mu_D(x,\mu_{Y_L}(x);\theta_{\mu_D})$. Both $\mu_{Y_H}(x;\theta_{\mu_{Y_H}})$ and $\sigma^2_{Y_H}(x;\theta_{\sigma^2_{Y_H}})$ are parameters of the same distribution; if there are different values of the ratio for each parameter, the model assigns a different level of trust to the same data source depending on which parameter it is updating.

5. **Setting** λ_{Y_H}: This is calculated using Equation (4.33a) given α_0.

6. **Setting** $\sigma^2_{Y_{H\,\text{upper}}}$: This is set as the maximum plausible predictive variance, considering the mismatch uncertainty and aleatoric uncertainty in the Hi-Fi data. It may be set, for example,

as the larger of the two components. In the worst case, this guarantees that $\mu_{Y_H}(x;\theta_{\mu_{Y_H}})$ will be weighted more heavily towards the Hi-Fi data over the prior during learning.

7. **Setting σ_D^2**: This is calculated by rearranging Equation (4.33b) given the rest of the hyper-parameters.

Selecting the functional form of $\mu_D(x,\mu_{Y_L}(x);\theta_{\mu_D})$

$\mu_D(x,\mu_{Y_L}(x);\theta_{\mu_D})$ captures the mismatch between the Hi-Fi and Lo-Fi data and is added to the Lo-Fi output as a form of correction. Parameters θ_{μ_D} are learned during training. A functional form, such as $Ax+B\mu_{Y_L}(x;\theta_{\mu_{Y_L}})+C$, where $A,B,C\in\theta_{\mu_D}$ are parameters to be learned during training, reflects a prior belief that the mismatch is a combination of a linear polynomial in x and a linear polynomial in $\mu_{Y_L}(x;\theta_{\mu_{Y_L}})$. For example, an engineer may have the expectation that Lo-Fi uncertainty grows linearly with a measured input such as engine power. Similarly, they may expect Lo-Fi uncertainties to increase with system temperature. Using an over-parametrised functional form can allow for the model to interpolate well, but struggle to extrapolate. The pitfalls of over-parametrisation are shown in Section 4.3.2. The functional forms used in the experiments are described in Table 4.1.

Table 4.1: Functional forms of $\mu_D(x,\mu_{Y_L}(x);\theta_{\mu_D})$ used in experiments. n is the choice of degree.

Function Type	General Equations
Polynomial on x	$\sum_{i=0}^{n} A_i x^i$
Polynomial on $\mu_{Y_L}(x;\theta_{\mu_{Y_L}})$	$\sum_{i=0}^{n} A_i \mu_{Y_L}^i(x)$
Periodic Fourier on x	$\sum_{i=1}^{5}(A_i\sin(ix)+B_i\cos(ix))+C_0$
Periodic on x	$A_0x^0+B_0\sin(B_1x+B_2)+C_0$
Polynomial on x and Polynomial on $\mu_{Y_L}(x;\theta_{\mu_{Y_L}})$	$A_0+\sum_{i=1}^{n}B_ix^i+\sum_{j=1}^{m}C_j\mu_{Y_L}^j(x;\theta_{\mu_{Y_L}})$
Polynomial on x and Periodic	$\sum_{i=1}^{n}A_ix^i+B_0\sin(B_1x+B_2)+C_0$

4.2.2 Model architecture

NNs are chosen for the parametrisations of the distribution parameters. The general MF model architecture is shown by Figure 4.2 which has five separate networks each of which has its own weights. The architecture is an adaptation of MVE networks, originally developed by Nix and

Weigend [33], which model the outputs of a Gaussian distribution at each point in a domain. Variations of these networks have been used for deterministic uncertainty quantification in recent literature [35, 46]. The use of NNs enables scalability in large data regimes, compared to GP regression, which may struggle in this instance. Furthermore, there is potential for extending the capacity of the model for quantifying model uncertainty via ensembling methods or MCMC sampling methods for example.

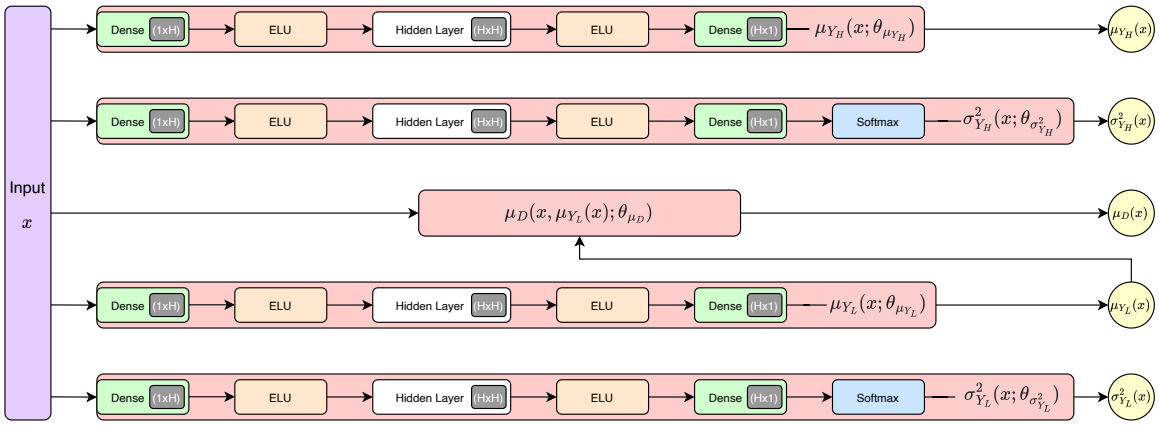


Figure 4.2: General MF model architecture consisting of sub-networks employed for each parameter. ELU activation functions are used at the hidden layers of the NNs. Softmax is applied to $\sigma^2_{Y_H}(x;\theta_{\sigma^2_{Y_H}})$. H is the number of hidden units.

The method differs from the original in that the loss functions are derived from posterior equations in order to fit the distributional parameters. The architecture relies on having a sufficient number of dense layers with ELU activation functions in order to capture the potentially non-linear nature of the functions. At the outputs of the variance networks, a final Softmax activation is applied in order to constrain the outputs to positive values.

4.2.3 Addressing convergence issues during training

A pre-training step using MSE loss functions on the predictive means was implemented before applying the training scheme described by Algorithm 1 to mitigate a negative effect of learning biases from Gaussian log-likelihood loss functions as noted by Seitzer et al. [39]. In regions where the model has small uncertainty i.e. $\sigma^2_{Y_H}(x;\theta_{\sigma^2_{Y_H}})$ is small, the loss function is weighted higher and

hence the learning of the predictive mean is biased compared to regions where the model has large uncertainty which is weighted lower. This biasing causes an imbalance in learning and the model struggles to learn the predictive means equally well across the domain. Hence, the predictive means are learned properly first on the data, before applying the training scheme described by Algorithm 1.

During MAP training in all experiments, a $\beta - NLL$ training scheme was applied as described by Seitzer et al. [39], with $\beta = 0.5$ for the Linear and Non-Linear Mismatch and AGTF30 Model Mismatch experiments and $\beta = 0.7$ for the Simple experiment. This scaled down the magnitude of the variance during training, hence reducing the effect of biasing. In all experiments, gradient clipping with max norm = 1 and the variance was lower bounded at 1E-06 ensuring the NLL never became too large and cause the optimisation to diverge. The convergence problem using Gaussian NLL functions is explored in more detail in Chapter 5. Chapter 5 also describes the $\beta -$ NLL approach in addition to outlining a unique alternative approach to mitigating this issue.

4.3 Results and discussion

This section first demonstrates the effect of hyper-parameter choice on predictive outputs using a simple mismatch example. The method is applied to synthetic examples, highlighting its ability to improve predictions in data-sparse regions. Finally, the approach is validated on an application of modelling the pressure ratio across a compressor in a Gas Turbine Engine (GTE). In the examples, the subspaces are explicitly defined. The subset A represents the space of all possible inputs for a physical system. The subset $B \subset A$ is a subspace that spans the pairs of inputs obtained directly from measurements of the ground truth of the physical system.

4.3.1 Simple mismatch example

An initial experiment using synthetic data was conducted to demonstrate the effect of increasing the relative weightings, $r = r_\mu = r_{\sigma^2}$. The Hi-Fi data was generated from $f(x) = \sin(3\pi x) + \frac{1}{2}x + \frac{1}{4}$, with training data sampled from the domain $x \in [0, 0.5]$. The Lo-Fi data was generated from $f(x) = \sin(3\pi x)$ with training data sampled across the entire domain $x \in [0, 1]$. A small amount of Gaussian noise $\mathcal{N}(0, 0.1)$ was added to the Hi-Fi training outputs. Hyper-parameters were set as $\sigma^2_{Y_{H\,\text{prior}}} =$

2, $\sigma^2_{Y_H\text{upper}} = 0.05, \alpha_0 = 1$ and $\mu_D(x, \mu_{Y_L}(x); \theta_{\mu_D})$ was set as a linear function of the form $Ax + B$. The MF model defined in Section 4.2.2 was used to parameterise the distribution parameters. Evaluations were conducted using test data across the domain by comparing the average uncertainty of $\sigma^2_{Y_H}(x; \theta_{\sigma^2_{Y_H}})$ in both the Hi-Fi and non-Hi-Fi data regions, with the expected value $\sigma^2_{Y_H}(x; \theta_{\sigma^2_{Y_H}})$ for each region. The expected values were calculated using Equation (4.30b) for the Hi-Fi data region and Equation (4.31b) for the no Hi-Fi data region. The experiment was repeated for $0.1 \leq r \leq 100$. Appendix B contains details on the data generation process, training parameters, training scheme, and model architecture.

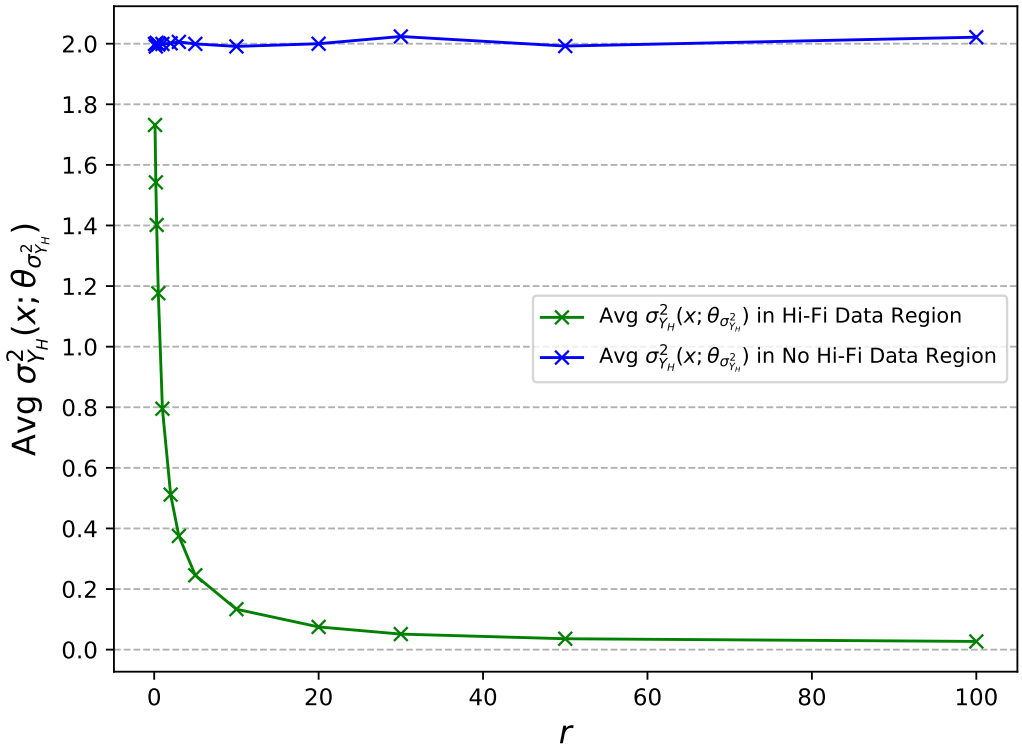
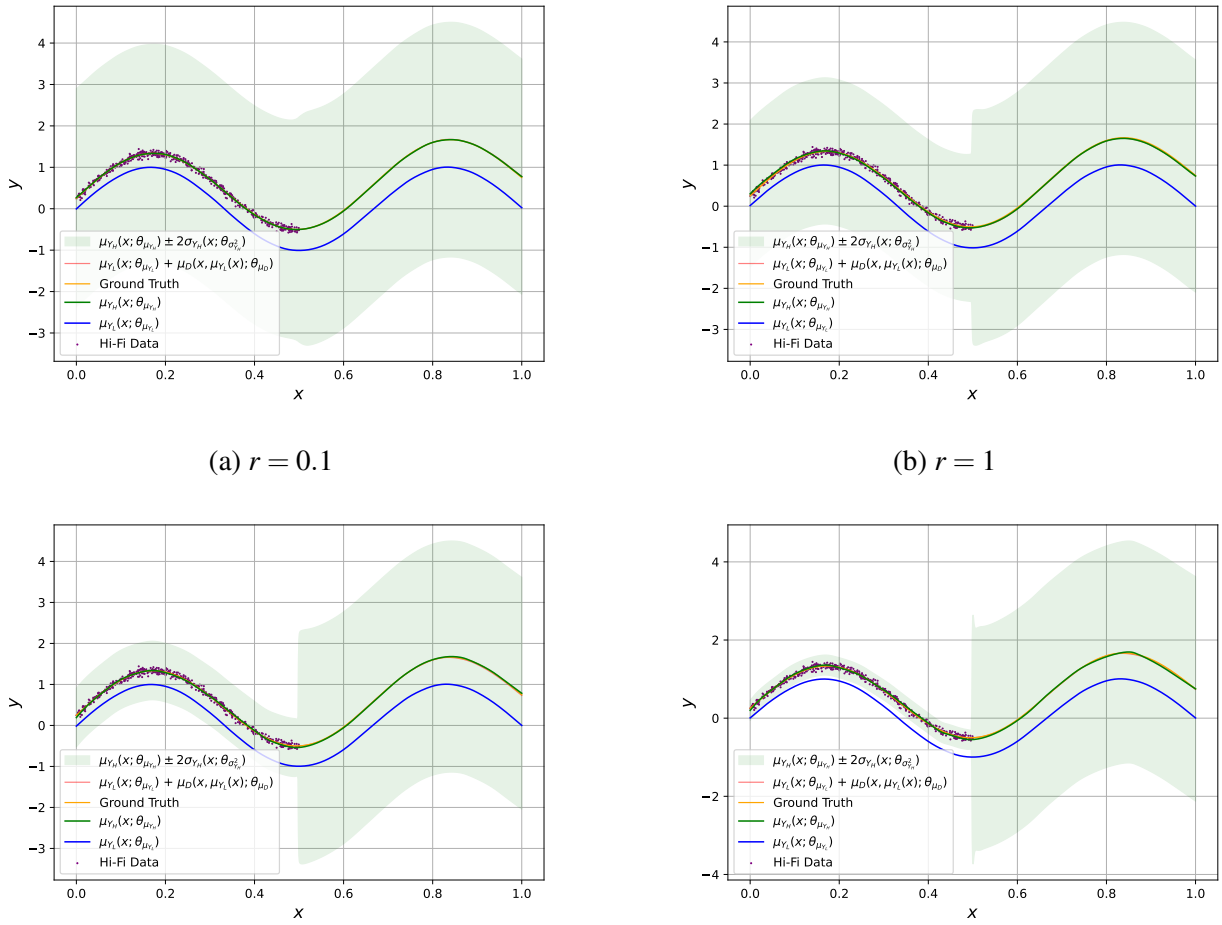


Figure 4.3: Average $\sigma^2_{Y_H}(x; \theta_{\sigma^2_{Y_H}})$ evaluated on test data in each region against choice of r.

Figure 4.3 illustrates that increasing r causes $\sigma^2_{Y_H}(x; \theta_{\sigma^2_{Y_H}})$ to deviate from the prior value of 2, and collapse towards the aleatoric uncertainty inherent within the data. This initially happens rapidly for r between 0 and 20. The prior region values for $\sigma^2_{Y_H}(x; \theta_{\sigma^2_{Y_H}})$ stay roughly around the prior value of 2. An increase in r also biases $\mu_{Y_H}(x; \theta_{\mu_{Y_H}})$ towards learning from data rather than the prior.

Figure 4.4 visually demonstrates how the uncertainty collapses towards the aleatoric uncertainty



(a) $r = 0.1$

(b) $r = 1$

(c) $r = 10$

(d) $r = 100$

Figure 4.4: Effect of increasing r on distributional $\mu_{Y_H}(x;\theta_{\mu_{Y_H}})$ and $\sigma^2_{Y_H}(x;\theta_{\sigma^2_{Y_H}})$.

as r increases in the data region. Furthermore, it demonstrates how $\mu_{Y_H}(x;\theta_{\mu_{Y_H}})$ shifts from learning the prior, $\mu_{Y_L}(x;\theta_{\mu_{Y_L}}) + \mu_D(x,\mu_{Y_L}(x);\theta_{\mu_D})$, to learning the data as r increases. For large values of r, $\mu_{Y_H}(x;\theta_{\mu_{Y_H}})$ over-fits to the data. In this example, a simple functional form of $\mu_D(x,\mu_{Y_L}(x);\theta_{\mu_D})$ was sufficient enough to yield good extrapolative ability of the model in the no Hi-Fi region. This example also demonstrates how the model is capable of defaulting to the desired prior values for the distributional parameters in the no Hi-Fi region, and with a choice of weighting r, controlling how the distributional parameters are learned based on the confidence in the Hi-Fi data relative to the Lo-Fi data.

The results of the experiment, outlined in Table 4.2, show small errors observed for all r demonstrating the model's ability to learn the expected values of $\sigma^2_{Y_H}(x)$. However, in almost all cases

Table 4.2: Comparison of average $\sigma^2_{Y_H}(x;\theta_{\sigma^2_{Y_H}})$ with the expected $\sigma^2_{Y_H}(x;\theta_{\sigma^2_{Y_H}})$ values in Hi-Fi and non-Hi-Fi data regions for a range of r.

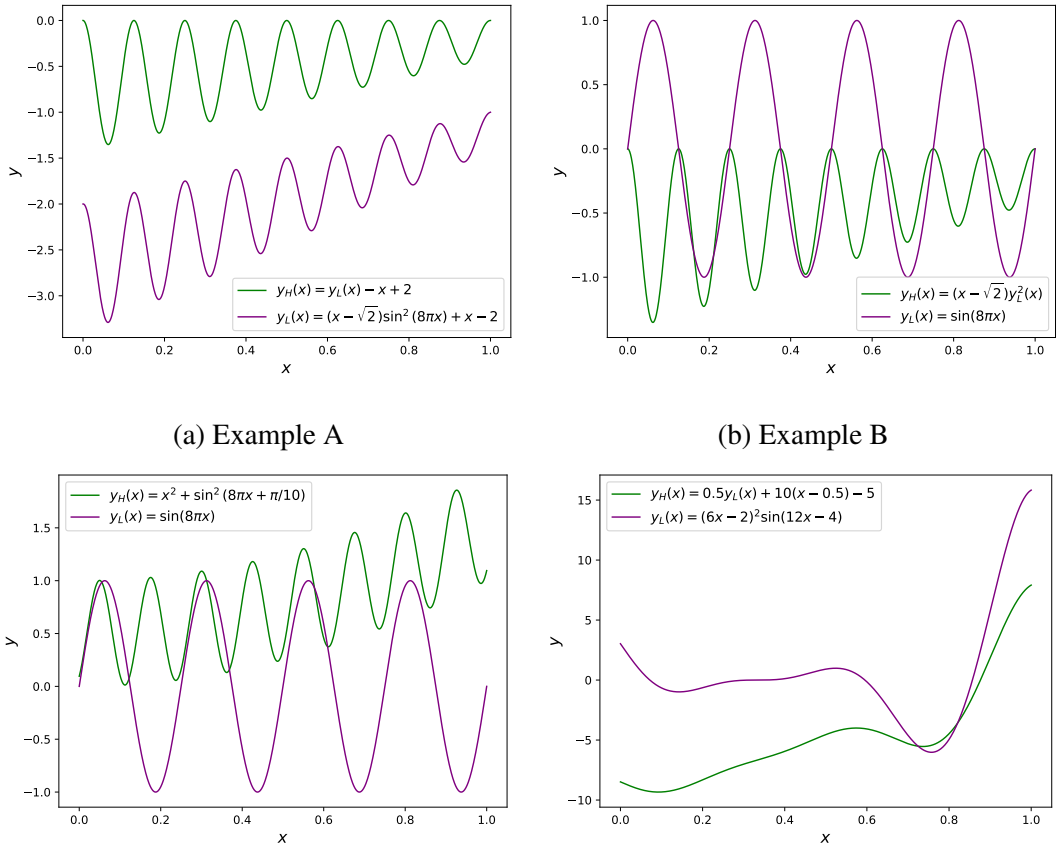
r	**Hi-Fi Data Region** $[0, 0.5]$			**No Hi-Fi Data Region** $[0.5, 1]$		
	Avg $\sigma^2_{Y_H}(x;\theta_{\sigma^2_{Y_H}})$	Exp $\sigma^2_{Y_H}(x;\theta_{\sigma^2_{Y_H}})$	Error	Avg $\sigma^2_{Y_H}(x;\theta_{\sigma^2_{Y_H}})$	Exp $\sigma^2_{Y_H}(x;\theta_{\sigma^2_{Y_H}})$	Error
0.1	1.7317	1.8184	-0.0867	1.9992	2.0000	-0.0008
0.2	1.5424	1.6671	-0.1247	1.9908	2.0000	-0.0092
0.3	1.4017	1.5390	-0.1373	2.0029	2.0000	0.0029
0.5	1.1764	1.3342	-0.1578	1.9972	2.0000	-0.0028
1	0.7952	1.0013	-0.2061	1.9992	2.0000	-0.0008
2	0.5117	0.6683	-0.1566	2.0013	2.0000	0.0013
3	0.3751	0.5019	-0.1268	2.0059	2.0000	0.0059
5	0.2454	0.3354	-0.0900	1.9994	2.0000	-0.0006
10	0.1334	0.1841	-0.0507	1.9911	2.0000	-0.0089
20	0.0750	0.0976	-0.0226	1.9999	2.0000	-0.0001
30	0.0513	0.0669	-0.0156	2.0239	2.0000	0.0239
50	0.0359	0.0417	-0.0057	1.9922	2.0000	-0.0078
100	0.0271	0.0223	0.0048	2.0215	2.0000	0.0215

$\sigma^2_{Y_H}(x)$ in the data regions is underestimated. This aligns with the observations made by Detlefsen et al. [34] and Sluijterman et al. [40] who have noted that Gaussian likelihood loss functions tend to underestimate variance when there is sparsity in the data, which may have occurred in this experiment. Although Sluijterman et al. [40] have suggested L2 regularisation as a means to ensure that the variance is learned correctly. This problem is not addressed here.

4.3.2 Linear and non-linear mismatch examples

The method was applied to four synthetic MF modelling examples found in the literature [85, 84, 81, 86], each with a different type of mismatch between the Lo-Fi and Hi-Fi data. These experiments exemplify how the choice of functional form of $\mu_D(x, \mu_{Y_L}(x); \theta_{\mu_D})$ affects the accuracy in predictive outputs. Figure 4.5 describes the functions from which data was generated for the Lo-Fi and Hi-Fi datasets in each example, all of which share the same domain $x \in [0, 1]$.

In Example A, the true mismatch is a polynomial of x of degree 1. The true mismatch in



(a) Example A

(b) Example B

(c) Example C

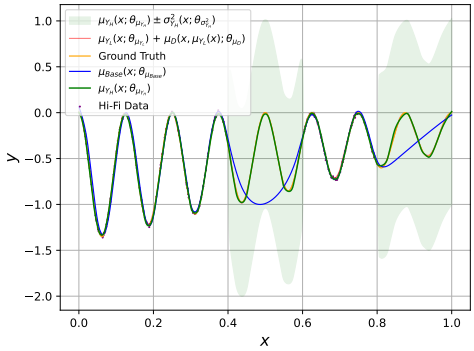
(d) Example D

Figure 4.5: Functional representations from which the Hi-Fi and Lo-Fi data are generated. The relationship between the functional representations differs: (a) Difference is linear in parameters of x (b) Difference is non-linear in parameters of x and $y_L(x)$ (c) Difference is non-linear in parameters of $y_L(x)$ in addition to a phase shift. (d) Difference is non-linear in parameters of x and $y_L(x)$.

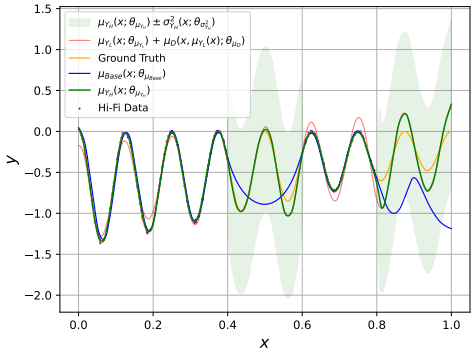
Example D is of a polynomial of $y_L(x)$ of degree 1 in combination with a polynomial in x of degree 1. The mismatch in Examples B and C are more complicated and cannot be captured by the restricted set of functional forms defined by Table 4.1. In practice, there isn't access to the true underlying data-generating process of each data set, in which case the choice of an appropriate function for $\mu_D(x, \mu_{Y_L}(x); \theta_{\mu_D})$ is based on an engineer's prior knowledge and intuition about the mismatch.

To generate the data in each example, Hi-Fi data was sampled in the domains $x \in [0, 0.4]$ and $x \in [0.6, 0.8]$. Lo-Fi data was sampled across the entire domain $x \in [0, 1]$. The Hi-Fi data outputs

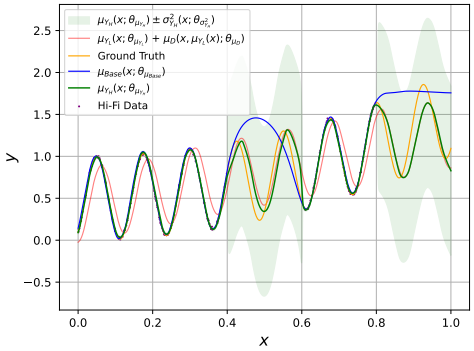
had a small amount of Gaussian noise $\mathcal{N}(0, 0.01)$ added to it in order to facilitate learning using MVE networks. All datasets were also normalised in order to facilitate learning. The focus of these synthetic experiments is to demonstrate how the choice in functional form of $\mu_D(x, \mu_{Y_L}(x); \theta_{\mu_D})$ affects the predictive outputs, hence all the hyper-parameters were set to constant values. Hyper-parameters were set as: $r = 50$ which strongly weights the data in favour of the prior; $\sigma^2_{Y_{H\,\text{prior}}} = 1$ which was kept small to make visualisations clearer; $\sigma^2_{Y_{H\,\text{upper}}} = 0.05$; and $\alpha_0 = 1$. Models were trained for a range of functional choices of $\mu_D(x, \mu_{Y_L}(x); \theta_{\mu_D})$ listed in Table 4.1 in order to demonstrate the effect of a good prior choice of functional choice on model prediction and the effect of over-parametrisation of the functional choice in interpolation and extrapolation. Evaluations were conducted by observing the MSE of $\mu_{Y_H}(x; \theta_{\mu_{Y_H}})$ with the ground truth for the Hi-Fi data, interpolative and extrapolative regions of the data. Appendix B contains details on the data generation process, training parameters, training scheme, and model architecture for each example.



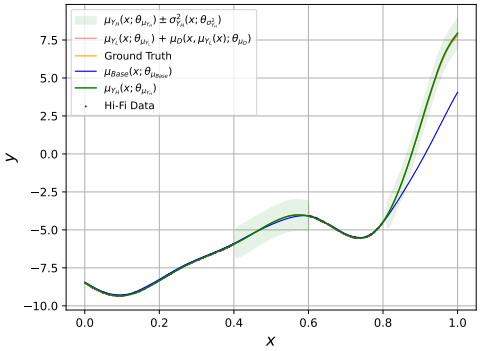
(a) MSE $\mu_{Y_H}(x;\theta_{\mu_{Y_H}})$: 6.3E-05 and MSE $\mu_{\text{Base}}(x;\theta_{\mu_{\text{Base}}})$: 0.058



(b) MSE $\mu_{Y_H}(x;\theta_{\mu_{Y_H}})$: 0.0028 and MSE$\mu_{\text{Base}}(x;\theta_{\mu_{\text{Base}}})$: 0.13



(c) MSE $\mu_{Y_H}(x;\theta_{\mu_{Y_H}})$: 0.0059 and MSE$\mu_{\text{Base}}(x;\theta_{\mu_{\text{Base}}})$: 0.15



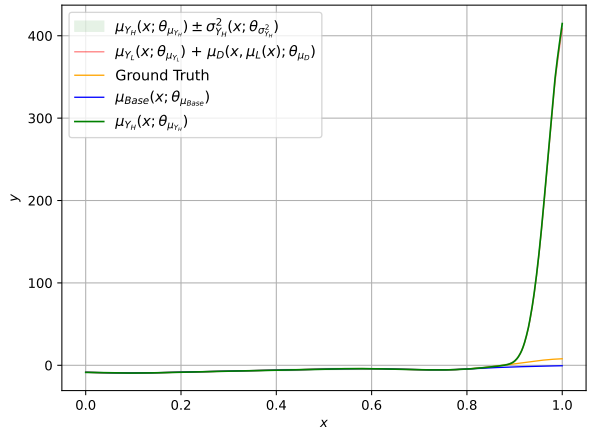
(d) MSE $\mu_{Y_H}(x;\theta_{\mu_{Y_H}})$: 6.5E-05 and MSE$\mu_{\text{Base}}(x;\theta_{\mu_{\text{Base}}})$: 1.5

Figure 4.6: Best performing models and their functional choice of $\mu_D(x,\mu_{Y_L}(x);\theta_{\mu_D})$: (a) Polynomial on x and $\mu_{Y_L}(x;\theta_{\mu_{Y_L}})$ of degree [1,3] (b) Polynomial on x and $\mu_{Y_L}(x;\theta_{\mu_{Y_L}})$ of degree [1,3] (c) Polynomial on x and $\mu_{Y_L}(x;\theta_{\mu_{Y_L}})$ of degree [1,3] (d) Polynomial on x and $\mu_{Y_L}(x;\theta_{\mu_{Y_L}})$ of degree [5,1].

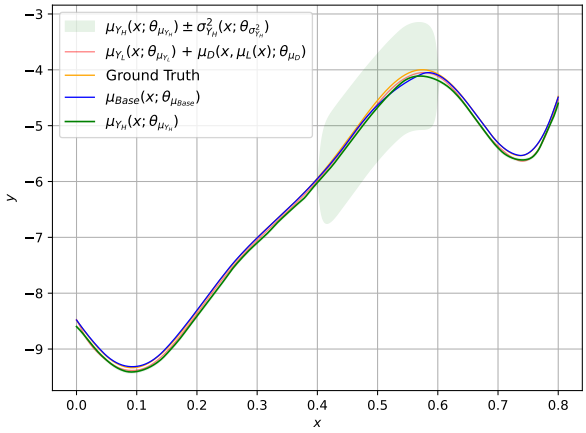
The best performing model results are shown in Figure 4.6 along with results from a baseline MVE network model trained only on the Hi-Fi data in order to demonstrate the improvements the approach makes in sparse regions of the domain. It can be seen that for Example A and D, there are functional choices which allow the model to closely fit the mismatch between the Hi-Fi and Lo-Fi models. In Examples B and C, where no function in Table 4.1 describes the exact mismatch, the best representation is by some linear combination of x and $\mu_{Y_L}(x;\theta_{\mu_{Y_L}})$. In these instances, there is a slight mismatch that occurs in the prior regions ($[0.4,0.8]\cup[0.8,1]$). In all instances, the approach

improves upon the baseline model predictions which doesn't learn anything in the non-Hi-Fi data regions. It can also be observed in every instance that the model is capable of learning the prior uncertainty in each region.

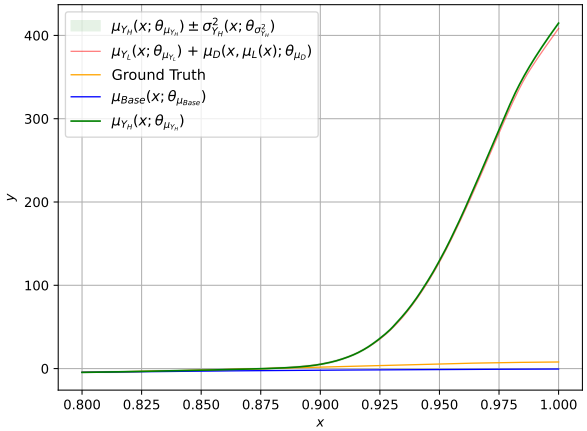
Figure 4.7 visually shows the effect of over parametrising $\mu_D(x, \mu_{Y_L}(x); \theta_{\mu_D})$ can have on interpolative and extrapolative regions. Between $x \in [0, 0.8]$ the model is able to fit reasonably well and increasing the number of parameters in $\mu_D(x, \mu_{Y_L}(x); \theta_{\mu_D})$ enables a reasonable fit in the prior region $x \in [0.4, 0.6]$. However, the model deviates further away from the ground truth in the extrapolation region $x \in [0.8, 1]$. This demonstrates that a simpler choice for $\mu_D(x, \mu_{Y_L}(x); \theta_{\mu_D})$ is better for when trying to use the model for extrapolation.



(a) Domain $x \in [0,1]$. MSE $\mu_{Y_H}(x;\theta_{\mu_{Y_H}})$: 1.5E+03 and MSE $\mu_{\text{Base}}(x;\theta_{\mu_{\text{Base}}})$: 4.9



(b) Domain $x \in [0,0.8]$. MSE $\mu_{Y_H}(x;\theta_{\mu_{Y_H}})$: 0.003 and MSE $\mu_{\text{Base}}(x;\theta_{\mu_{\text{Base}}})$: 0.0018



(c) Domain $x \in [0.8,1]$. MSE $\mu_{Y_H}(x;\theta_{\mu_{Y_H}})$: 7.2E+03 and MSE $\mu_{\text{Base}}(x;\theta_{\mu_{\text{Base}}})$: 24

Figure 4.7: Model evaluated across different parts of the domain for Example B using a polynomial on x and $\mu_{Y_L}(x;\theta_{\mu_{Y_L}})$ with degree $[1,5]$ for the functional choice of $\mu_D(x,\mu_{Y_L}(x);\theta_{\mu_D})$.

Table 4.3: MSE of $\mu_{Y_H}(x;\theta_{\mu_{Y_H}})$ on test data. Degree $[n,m]$ corresponds to a Polynomial on x with degree n and Polynomial on $\mu_{Y_L}(x;\theta_{\mu_{Y_L}})$ with degree m. Data refers to the region $x \in [0,0.2] \cup [0.6,0.8]$. Inter. refers to the interpolative region $x \in [0.4,0.6]$. Extra. refers to the extrapolative region $x \in [0.8,1]$. Bold and italics represents the lowest and highest MSEs for each domain and function type. (Part 1)

		MSE $\mu_{Y_H}(x;\theta_{\mu_{Y_H}})$					
		Example A			**Example B**		
Function Type	**Degree**	**Data**	**Inter.**	**Extra.**	**Data**	**Inter.**	**Extra.**
Polynomial on x	1	3.76E-05	3.35E-04	**1.39E-04**	*2.33E-03*	**2.61E-01**	**2.08E-01**
	3	5.20E-05	**1.42E-04**	9.83E-04	1.29E-03	*3.62E-01*	1.08E+00
	5	**6.42E-05**	3.26E-04	7.67E-04	**9.62E-04**	2.99E-01	6.17E+01
	10	*1.59E-04*	*4.73E-04*	*2.01E+02*	1.48E-03	3.36E-01	*1.13E+03*
Polynomial on $\mu_{Y_L}(x;\theta_{\mu_{Y_L}})$	1	*2.08E-04*	**6.37E-03**	**2.16E-02**	1.32E-04	*3.64E-02*	**2.37E-02**
	3	**7.77E-05**	1.29E-02	1.85E-02	*2.21E-04*	8.70E-03	4.04E-02
	5	1.18E-04	*1.16E-02*	1.82E-02	1.89E-04	1.00E-02	*4.11E-02*
	10	1.23E-04	1.54E-02	*4.21E-01*	**7.65E-05**	**7.74E-03**	4.05E-02
Periodic Fourier	5	8.71E-05	2.70E-04	2.67E-04	1.13E-03	1.51E+00	5.38E+01
Periodic	-	8.96E-05	1.81E-04	1.29E-04	1.33E-03	3.02E-01	3.23E-01
Polynomial on x and Polynomial on $\mu_{Y_L}(x;\theta_{\mu_{Y_L}})$	[1,1]	7.81E-05	2.68E-04	3.52E-04	9.21E-05	3.67E-02	1.57E-02
	[1,3]	4.24E-05	**7.24E-05**	**9.35E-05**	8.17E-05	2.23E-03	**1.16E-02**
	[1,5]	6.70E-05	4.01E-04	3.47E-04	**6.64E-05**	**1.99E-03**	1.25E-02
	[3,1]	6.16E-05	*4.35E-04*	6.10E-04	8.74E-05	5.63E-02	1.83E-01
	[3,3]	**3.95E-05**	2.79E-04	1.23E-04	8.84E-05	3.16E-03	1.29E-02
	[3,5]	5.54E-05	3.20E-04	4.86E-04	1.29E-04	2.32E-03	1.23E-02
	[5,1]	6.67E-05	1.75E-04	*1.28E-03*	1.32E-04	*8.86E-02*	*2.31E+00*
	[5,3]	*1.39E-04*	2.04E-04	1.18E-03	1.20E-04	3.00E-03	3.15E-02
	[5,5]	8.34E-05	1.17E-04	1.08E-03	*1.09E-04*	4.07E-03	1.64E-02
Polynomial on x and Periodic	1	**3.24E-05**	1.31E-04	**1.62E-04**	*1.48E-03*	**3.34E-01**	**5.93E-01**
	3	*4.63E-05*	*1.25E-04*	9.73E-04	1.44E-03	*3.53E-01*	1.09E+00
	5	4.93E-05	**6.68E-05**	*1.24E-03*	**1.29E-03**	3.41E-01	*9.43E+01*

Table 4.4: MSE of $\mu_{Y_H}(x;\theta_{\mu_{Y_H}})$ on test data. Degree $[n,m]$ corresponds to a Polynomial on x with degree n and Polynomial on $\mu_{Y_L}(x;\theta_{\mu_{Y_L}})$ with degree m. Data refers to the region $x \in [0,0.2] \cup [0.6,0.8]$. Inter. refers to the interpolative region $x \in [0.4,0.6]$. Extra. refers to the extrapolative region $x \in [0.8,1]$. Bold and italics represents the lowest and highest MSEs for each domain and function type. (Part 2)

		MSE $\mu_{Y_H}(x;\theta_{\mu_{Y_H}})$					
		Example C			**Example D**		
Function Type	**Degree**	**Data**	**Inter.**	**Extra.**	**Data**	**Inter.**	**Extra.**
Polynomial on x	1	*3.04E-03*	**2.39E-01**	**2.39E-01**	*1.32E-02*	*7.30E-01*	1.41E+01
	3	1.75E-03	2.44E-01	2.58E-01	4.35E-03	7.43E-02	4.86E+01
	5	**1.24E-03**	5.18E-01	9.76E+01	**7.95E-04**	**3.09E-02**	**4.24E+00**
	10	2.23E-03	*5.58E-01*	*1.67E+03*	2.41E-02	2.14E-01	*2.41E+04*
Polynomial on $\mu_{Y_L}(x;\theta_{\mu_{Y_L}})$	1	*4.76E-04*	*5.11E-02*	*1.45E-01*	2.79E-01	**3.64E+00**	6.69E+01
	3	1.62E-04	2.30E-02	**1.00E-01**	**2.68E-01**	4.45E+00	5.69E+02
	5	1.91E-04	2.13E-02	1.08E-01	2.87E-01	4.42E+00	**2.75E+03**
	10	**9.53E-05**	**2.12E-02**	1.06E-01	*1.89E+00*	*7.94E+00*	*6.70E+06*
Periodic Fourier	5	5.16E-04	2.52E-01	2.24E+01	2.57E-05	5.74E-03	1.19E+00
Periodic	-	2.32E-03	2.47E-01	2.60E-01	9.27E-03	4.82E-01	1.78E+01
Polynomial on x and Polynomial on $\mu_{Y_L}(x;\theta_{\mu_{Y_L}})$	[1,1]	*2.16E-04*	3.65E-02	5.53E-02	5.98E-05	6.79E-05	1.11E-03
	[1,3]	1.08E-04	**1.45E-02**	**1.45E-02**	8.63E-05	1.06E-04	1.15E-02
	[1,5]	1.67E-04	1.76E-02	2.02E-02	2.63E-03	3.92E-03	7.18E+03
	[3,1]	1.54E-04	6.08E-02	2.57E-01	**3.08E-05**	5.70E-05	2.32E-03
	[3,3]	1.66E-04	1.93E-02	3.21E-02	1.58E-04	1.27E-04	1.72E-03
	[3,5]	1.60E-04	1.47E-02	2.22E-02	4.39E-04	1.36E-04	3.49E+01
	[5,1]	1.76E-04	*6.75E-02*	5.61E-02	3.67E-05	**4.23E-05**	**1.43E-04**
	[5,3]	1.18E-04	2.07E-02	*2.03E+00*	1.08E-04	2.35E-04	1.20E-02
	[5,5]	**7.71E-05**	2.40E-02	1.04E+00	*2.09E-02*	*1.69E-01*	*7.77E+04*
Polynomial on x and Periodic	1	**2.81E-04**	**2.37E-01**	**2.39E-01**	4.61E-03	4.67E-02	4.17E+01
	3	1.57E-03	2.52E-01	2.42E-01	*7.06E-03*	*8.70E-02*	*5.06E+01*
	5	*1.37E-03*	*6.72E-01*	*8.57E+01*	**1.12E-03**	**3.27E-02**	**4.63E+00**

Finally, the results shown in Tables 4.3 and 4.4 shows that in all experiments, the data regions have a low MSE, which suggests that the value of $r = 50$ was sufficiently high enough for the

model learn heavily from the Hi-Fi data in those regions. In interpolative regions, for each function type, the general trend shows that increasing the degree of the function slightly decreases the MSE. In extrapolative regions, an increase in degree tends to lead to larger MSEs. In all examples, a functional choice of polynomial on x and polynomial on $\mu_{Y_L}(x;\theta_{\mu_{Y_L}})$ is the best choice.

4.3.3 Steady-state AGTF30 model example

GTEs are examples of complex machines that have a great economic incentive for Health Monitoring (HM) through DTs. A common approach is to track the deviations of an engine from normal operation. Although often an engineer may have rich data in certain regions of the operating domain, particularly along typical flight paths, there are instances where deviations in air traffic control may cause a GTE to operate in less familiar regions where there is sparse data [98], but a HM decision is still needed. HM providers commonly have models of GTEs that can make inferences in such cases; however, they are imperfect. They may not be tuned to the asset specific health conditions (e.g. compressor erosion vs turbine blade oxidation) or they are simplified and do not necessarily capture all behaviours (e.g. long thermal time constant effects resulting in tip clearance leakage). This section illustrates the application of the MF modelling approach to building models of a GTE across its entire operating range.

Specifically, over its lifetime, compressors in a GTE experience blade tip wear, which can be caused by various factors, including erosion. This wear increases the clearance between the blade tips and the compressor casing, resulting in air leakage. The consequences of air leakage include reduced compressor efficiency and therefore increased power demand to achieve the same thrust. The effect of loss of air is exacerbated at higher pressures, which in turn causes an increased demand in power by the system. A lower efficiency in the compressor means that for the same amount of thrust demand delivered, a higher amount of work needs to be done on the air in the compressor [99]. In this section, it is demonstrated how the method can improve the modelling of the pressure ratio *PR* across the High Pressure Compressor (HPC) of a GTE given fuel flow w_f. A scenario, as described above, is replicated where Hi-Fi data, representing real measurements obtained from multiple flights, is used, and Lo-Fi data are flexibly generated from a simulation model defined below. There is a level of mismatch between the Hi-Fi data and Lo-Fi data due to the degradation of the compressor and lack of calibration in the Lo-Fi model. Perfect knowledge of the mismatch

of these models cannot be obtained a priori, but physics knowledge and intuition can shape a prior form of the mismatch.

To illustrate the framework in the presence of sparsity and mismatch due to degrading assets, the AGTF30 Simulink model [100] of a GTE was used to generate the Hi-Fi and Lo-Fi datasets. The model takes a pilot's Power Lever Angle (PLA) as one of its inputs where $x_{\text{PLA}} \in [40, 80]$. The Hi-Fi data represent multiple runs of a flight profile with take-off, climb, cruise, and descent operating regimes. Data from multiple runs were generated by applying slight variations on the PLA at each operating regime. Data were then extracted at steady-state values. To simulate wear on the Hi-Fi datasets, fractional bleed valves were opened in the HPC when generating the Lo-Fi data. To represent elements of real-world complexity, the assumption of linear efficiency of the publicly available gas turbine model was replaced with an arbitrary power dependent function, assumed to be captured by an enhanced nominal model of a system. Although data is available from a typical flight profile, circumstances and conditions may push the engine's operation into unfamiliar parts of the domain where there may not be historical data available. An engineer may want to be able to predict in such regions given the availability of Lo-Fi models which can predict in those areas. Using the method, the aim was to give an improved prediction of *PR* across such regions. All datasets were normalised to facilitate learning.

More specifically, to generate both datasets, a random sequence of PLA values, $x_{\text{PLA}} \in [40, 80]$ values was input into the model, and steady-state values were extracted. In order to simulate more complex fluidic air flow regime changes at different power demands, the efficiency value was adjusted for different regions of the input domain. When $x_{\text{PLA}} \in [.40, .55]$ then, the efficiency changed to $g(x_{\text{PLA}}) = 1 + 0.1 \exp(25(x_{\text{PLA}} - .55))$. When $x_{\text{PLA}} \in [.55, .70]$, the efficiency was changed to $h(x_{\text{PLA}}) = 1 + -\frac{20}{90} x_{\text{PLA}} + \frac{20}{90}$. Outside of these values, the efficiency remained at the default value of 1. In order to create an artificial mismatch between the two datasets, fractional bleed valves were opened when generating the Hi-Fi data. Fractional bleed valves subtract a fraction of the input mass flow from the output mass flow during operation. The fraction in each bleed valve was a function of x_{PLA}. The default fractional bleed values at each bleed valve, $[0.0200, 0.0693, 0.0625]$, were multiplied by $f(x_{\text{PLA}}) = 1.2 \exp(\frac{1}{20}(x_{\text{PLA}} - 80))$. This function is increasing monotonically to mimic the effect of how increasing power demands cause greater leakage in the system in the presence of wear. Appendix B contains details on the training parameters, training scheme, and

model architecture.

Hyper-parameters were set as: $\sigma^2_{Y_{H\text{upper}}} = 0.05$ since aleatoric noise within the datasets was expected to be small; $\sigma^2_{Y_{H\text{prior}}} = 0.1$ which is the mismatch uncertainty expected when defaulting to the prior for a prediction and reflects a high confidence in the ability to model the mismatch; $r = 20$, expressing a high degree of confidence in the Hi-Fi data over the Lo-Fi data in the overlap regions. The functional choice of $\mu_D(w_f, \mu_{Y_L}(w_f); \theta_{\mu_D})$ was a combination of polynomial on x with degree 3 and a polynomial on $\mu_{Y_L}(w_f; \theta_{\mu_{Y_L}})$ with degree 3. This correction function provides sufficient flexibility to capture the smooth non-linear mismatch between the Lo-Fi and Hi-Fi data, while being simple enough to prevent over-fitting. Higher orders would lead to poor extrapolation, which is required in this example. Section B provides further details on the data generation process including the artificial mismatch between the datasets, training parameters, training scheme and model architecture.

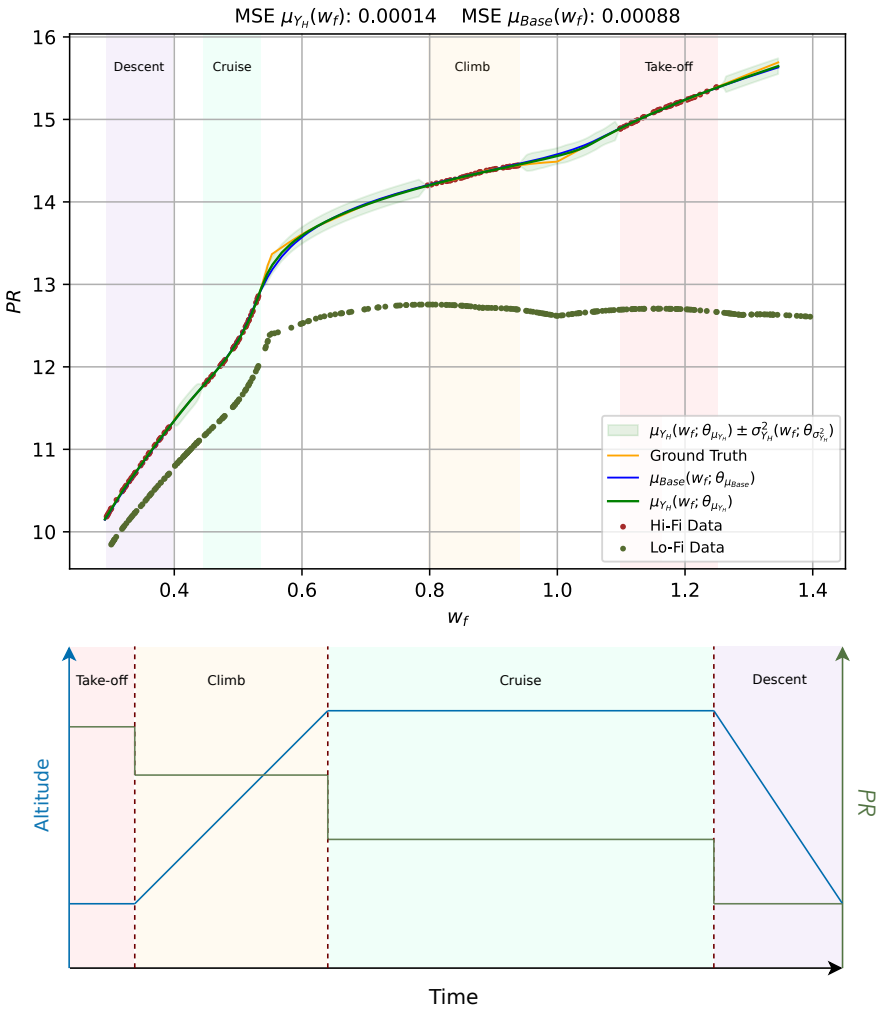


Figure 4.8: Plot of w_f against *PR* for model predictions on test data

The test results are shown by Figure 4.8 that includes the evaluations on a baseline MVE network ($\mu_{Base}(x)$) trained only on the Hi-Fi data. The Hi-Fi data generated at each of the operating regimes are mapped to operating regimes shown in the altitude and *PR* against time graph. Figure 4.8 demonstrates that, having generated Lo-Fi data which reflect a strong understanding of the regime changes the system goes through across the entire domain, the model is capable of accurately modelling the system despite not having Hi-Fi data in those regions. Although the baseline model is able to reasonably interpolate between the regions, it is not able to pick up the subtle operating behaviour in that region. In particular, in $w_f \in [0.54, 0.79]$ the model was able to capture the more complex transitory stage of the engine.

4.4 Conclusions and future work

The need to mitigate model mismatch was clear through the experimental conclusions of Chapter 3, where a large model mismatch was shown to be detrimental when training a PINN on a 1-D heat equation example. Re-weighting the physics loss function seemed like a promising avenue for mitigating model mismatch; however, the original approach was not principled, lacked mismatch uncertainty quantification, and could not default to a meaningful prior when there was a lack of knowledge. Furthermore, PINNs require well-defined governing equations to be applicable, and it would be more advantageous if a modelling framework could incorporate richer physics-based models such as simulation models. As described in Chapter 2, uncertainty quantification is important in establishing trust in the predictive outputs of a model. A paradigm shift towards MF modelling was conducted in order to address these issues whilst still being able to implement PIML methods. Resultantly, this chapter presented a unique Bayesian problem formulation allowing engineers to place prior beliefs on mismatch between data sources when undertaking MF modelling. Model mismatch was mitigated between a Lo-Fi dataset and Hi-Fi dataset via a learned correction, the structure of which is an engineer's choice based on prior knowledge about the problem. A general training approach was outlined for parametrised distributional parameters, along with a systematic approach for setting hyper-parameters considering key trade-offs in MF modelling. The approach was implemented using an adapted MVE NN and was evaluated on a series of synthetic problems before applying the method to an application of predicting the *PR* across a compressor in a GTE.

The initial constant mismatch example validated the ability of the methods to control the influence of the prior through the choice of $r_\mu = r_{\sigma^2} = r$, which biased the distributional parameters according to the relative confidence between the Hi-Fi data and the prior. Furthermore, the expected mismatch uncertainty was accurately expressed across the entire domain, defaulting to the prior value in regions of sparse data. The synthetic linear and non-linear examples showcased how a sensible choice of correction function $\mu_D(x, \mu_{Y_L}(x); \theta_D)$ improves both interpolation and extrapolation compared to a baseline MVE model trained only on Hi-Fi data. Restricting the choice of $\mu_D(x, \mu_{Y_L}(x); \theta_D)$ led to better interpretability of the mismatch of the datasets over highly parametrised models. Additionally, applying the method to an application of modelling the *PR* in

a GTE, showcased applicability of this framework in a real-life setting. The results demonstrated the ability of the MF model to capture subtle changes in the GTE's operating regime that the baseline model was unable to capture. Finally, the implementation of the framework on NNs offers a cheaper computational cost at inference and training compared to the existing state of the art approaches.

Despite its advantages, the approach has limitations. Firstly, the method assumes the data is Normally distributed at each point in the domain, which may be too restrictive of an assumption in some cases. Furthermore, modelling assumptions could not be placed in the transitory regions of the model using the Bayesian problem formulation. This was shown by the sharp transitions between Hi-Fi data and sparse regions of the model. Engineers may want to assume a smoother or slower rate of transition between these regions if they are confident in the capacity of $\mu_{Y_H}(x)$ to extrapolate away from the data. A further related limitation is that the sample size, n_{Y_H}, in the approximate posterior was a fixed hyper-parameter. Consequently, assumptions about the level of trust between datasets of differing fidelity were applied globally and were constant. In practice, an engineer might want vary the amount of trust based upon the density of data points in a region of the domain. Another limitation, is the computational complexity of VI inference approach. In particular, this includes the requirement of optimising groups of distributional parameters separately and in stages. Finally, when directly using the loss functions in training, suboptimal convergence was apparent. Although mitigations were applied, the number of epochs required was too large for the simplicity of the task. The next chapter develops a novel method for solving the convergence issue when training using MVE networks and the Gaussian NLL.

In Chapter 6, an adapted MAP-based approach is developed which allows for modelling assumptions to be placed in transitory regions of the model. The approach also varies the sample size in the approximate posterior based on the density of points in a local region of the domain, allowing information about Hi-Fi data density to be included into the modelling assumptions. Furthermore, a lower-complexity MAP fitting approach is developed in comparison to the VI-based approach outlined in this chapter.

Future work includes providing a detailed explanation about how engineers might set the prior mismatch uncertainty for different use cases. This would make this framework more accessible. Furthermore, there is scope to incorporate the learned predictive variance $\sigma^2_{Y_L}(x)$ into the setting of

the uncertainty value. Finally, extending this work to higher-dimensions in both input and output space, in addition to making it applicable for more than two fidelities of datasets, would increase the versatility of the approach.

Chapter 5

Addressing convergence issues in uncertainty estimation using mean variance estimation networks

Suboptimal convergence is a known problem when training Neural Networks (NNs) with the Gaussian Negative Log-Likelihood (NLL) loss function [39]. This is primarily caused by the variance term in the loss function, which contributes to large biases in the gradients of the loss, which in turn causes learning imbalances across the domain. This issue is apparent even in low-dimensional and small-data examples, where NNs are expected to be able to fit the mean and variance of the data. The fitting of the Gaussian NLL is a common approach for quantifying aleatoric uncertainty in NN regression. For example, this is essential for effective data driven health and condition monitoring for expensive assets. In Chapter 4, convergence issues were present when using loss functions that incorporated Gaussian NLL losses within them. And, whilst these issues were largely mitigated through the use of the $\beta-$NLL method [39], prior knowledge about the data is required to tune the β hyper-parameter, which can be prohibitive in large data regimes. Furthermore, retraining multiple models for different β values is not cost effective. In this chapter, novel algorithms for resolving suboptimal convergence when using Gaussian NLL loss functions are developed, all of which do not require the tuning of hyper-parameters and can be generalised for similar types of loss functions.

This chapter begins by explaining the suboptimal convergence problem along with a critical

review of the $\beta-$NLL method [39], which is the state of the art for mitigating this issue. The algorithms developed in this chapter draw inspiration from the learning rate annealing algorithm developed by Wang et al. [67] to mitigate gradient imbalances when training PINNs. Their analysis and algorithm are then outlined, from which this thesis developed their work in order to build a novel algorithm for solving the related, suboptimal convergence problem when using the Gaussian NLL. Using data generated from a 1-D sinusoidal function with homoscedastic noise, the convergence issue is exposed when training a baseline model with no mitigations applied. The baseline result is then compared to the models trained using the $\beta-$NLL method and the newly developed algorithm. Analyses of all methods were conducted to provide further insights into the suboptimal convergence problem and their proposed solutions. Noise issues were present in the initial annealing μ algorithm. Further adaptations of the novel algorithm were developed in order to address these noise issues and improve convergence. All methods were evaluated on a simple 1-D heteroscedastic noise example with and without varied amounts of Weight Decay (WD) applied to the networks. Based on these results, the chapter concludes by recommending the preferred variant of the annealing algorithm for practitioners.

The key contributions of this chapter can be summarised below:

- The development of a novel 'annealing μ' algorithm capable of addressing the sub-optimal convergence problem when training NNs using the Gaussian NLL loss function.
- The development of variations of the novel 'annealing μ' algorithm which attempt to address gradient imbalances found in either, or both, the parametrised mean and the variance networks.
- Analysis and comparisons of models trained using the 'annealing μ' algorithm and its variations, with a baseline model and models trained using the $\beta-$NLL method when trained on simple 1-D homoscedastic and heteroscedastic noise examples. Further evaluation is conducted on the heteroscedastic noise example with the application of WD on the NN.

5.1 Background

In order to understand the nature of the problem, the original Gaussian NLL loss is described. First, consider a dataset consisting of input output pairs $\mathcal{D} = \{(x_1, y_1), (x_2, y_2) \ldots, (x_n, y_n)\}$. For this dataset, a model that predicts the mean $\mu(x)$ and variance $\sigma^2(x)$ can be fit using the Gaussian NLL loss as defined by Equation (5.1).

$$\mathcal{L}_{\text{NLL}} = \frac{1}{2}\log(\sigma^2(x)) + \frac{1}{2}\frac{(\mu(x)-y)^2}{\sigma^2(x)}. \tag{5.1}$$

The corresponding gradients with respect to the predicted mean and variance, which are required for gradient-based optimisation, are given by:

$$\nabla_{\mu}\mathcal{L}_{\text{NLL}}(\theta_{\mu}) = \frac{\mu(x)-y}{\sigma^2(x)}, \tag{5.2}$$

$$\nabla_{\sigma^2}\mathcal{L}_{\text{NLL}}(\theta_{\sigma^2}) = \frac{\sigma^2(x)-(y-\mu(x))^2}{2(\sigma^2(x))^2}. \tag{5.3}$$

Seitzer et al. [39] showed empirically in their work that the $\frac{1}{\sigma^2(x)}$ term in Equation (5.2) can cause large biases in the gradients during training. Consequently, regions of the domain that learn a low predictive variance have their batch gradient scaled up compared to regions where there is high predictive variance. As such, the predictive mean is not learned correctly and suboptimal convergence occurs. Seitzer et al. [39] demonstrate that even in low dimensional small data regimes, despite being trained for millions of epochs, the models fail to converge. This issue is mitigated by multiplying Equation (5.1) by $\lfloor\sigma^{2\beta}(x)\rfloor$ as shown by Equation (5.4), where $\lfloor . \rfloor$ denotes the stop gradient operation which prevents gradient flow during back-propagation and $0 \leq \beta \leq 1$.

$$\mathcal{L}_{\beta-\text{NLL}} = \lfloor\sigma^{2\beta}(x)\rfloor \left[\frac{1}{2}\log(\sigma^2(x)) + \frac{1}{2}\frac{(\mu(x)-y)^2}{\sigma^2(x)}\right] \tag{5.4}$$

$$\nabla_{\mu}\mathcal{L}_{\beta-\text{NLL}}(\theta_{\mu}) = \frac{(\mu(x)-y)}{\sigma^{2-2\beta}(x)} \tag{5.5}$$

$$\nabla_{\sigma^2}\mathcal{L}_{\beta-\text{NLL}}(\theta_{\sigma^2}) = \frac{(\sigma^2(x)-(y-\mu(x))^2)}{2\sigma^{4-2\beta}(x)} \tag{5.6}$$

Their method reduces the effect of uneven sampling in the domain by scaling down the effect of the biasing term $\frac{1}{\sigma^2(x)}$ on the gradient of the predictive mean. When $\beta = 0$, $\mathcal{L}_{\beta-\text{NLL}}$ is equivalent to $\mathcal{L}_{\text{NLL}}$. When $\beta = 1$, Equation (5.5) is equivalent to the MSE and Equation (5.6) has the denominator $\frac{2}{\sigma^2(x)}$. The choice of β affects the self regularisation strength of the NLL and therefore

the sensitivity of the model in learning high noise regions. Therefore, despite $\mathcal{L}_{\text{NLL}}$ and $\mathcal{L}_{\beta-\text{NLL}}$ having the same optima, in practice it can lead to very different solutions. If the data are a good representation of the true aleatoric uncertainty, then we may want to include the extreme points in the fitting process, and therefore choose a low value of β. Conversely, if these extreme points represent outliers in the data, then a higher value of β is appropriate. Whilst this hyper-parameter provides flexibility of modelling choice, the efficacy of the method is proportional to the strength of β, hence if a lower value of β is suitable for the problem, then suboptimal convergence may occur and this represents a trade-off. Furthermore, there may be difficulty in tuning this parameter for different types of problems. The $\beta-$NLL method may struggle to handle the case where there are extreme outliers, for example, when a faulty sensor might produce a zero periodically. However, in cases where there are outliers that are closer to the true data distribution, the $\beta-$NLL method would be appropriate. Seitzer et al. [39] demonstrate that their adapted loss function can improve the performance in terms of Root Mean Squared Error (RMSE) and Log Likelihood (LL) for a range of problems including regression, generative, and classification tasks.

Other mitigations to the suboptimal convergence problem have been suggested in the literature, such as bounding the predictive variance or applying gradient clipping during training [40]. Bounding the predictive variance requires a good understanding of the noise in the data, which an engineer may not have; furthermore, if the data have inherently low noise, large biases can still occur during training. Gradient clipping prevents large gradients from dominating, which can help mitigate learning biases from occurring; however, they alter both the true gradient direction and magnitude, which therefore can slow down convergence. Finally, both of these methods are highly sensitive to the tuning of their associated hyper-parameters.

5.1.1 Learning rate annealing algorithm for solving suboptimal convergence when training physics-informed neural networks

Suboptimal convergence observed when shown that the suboptimal convergence using the Gaussian NLL loss is primarily caused by biases originating from the inverse variance weighting $\frac{1}{\sigma^2(x)}$ in Equation (5.5). PINNs also face similar convergence issues during training. These are caused by biases inherent within the physics loss term relative to the data loss terms. Wang et al. [67]

identify two fundamental points of failure when training PINNs. First, stiff gradient flow dynamics when back-propagation occurs using the physics loss. Second, upper bounds on the gradients of the physics loss can be larger than the data loss terms. Both points of analysis are restated below, omitting some intermediary steps that can be found in their paper.

In their first point of analysis, Wang et al. [67] show that stiff gradient flow dynamics potentially occurs in the physics loss function during gradient descent when the eigenvalues of the Hessian of the loss terms are sufficiently large. In this section, the physics loss is defined by $\mathcal{L}_r$, the single data loss term is defined by $\mathcal{L}_d$, and the total loss is defined by $\mathcal{L} = \mathcal{L}_r + \mathcal{L}_d$. Suppose that at the nth step of gradient descent during training, the following holds:

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta \mathcal{L}(\theta_n) = \theta_n - \eta |\nabla_\theta \mathcal{L}_r(\theta_n) + \nabla_\theta \mathcal{L}_d(\theta_n)| \tag{5.7}$$

Here, θ_{n+1} and θ_n are the parameters of a PINN in the $n+1$ and nth step of training; $\mathcal{L}(\theta_n)$, $\mathcal{L}_r(\theta_n)$ and $\mathcal{L}_d(\theta_n)$ is the total loss, data loss, and physics loss evaluated with θ_n; η is the learning rate. Then applying a second order Taylor expansion to the total loss $\mathcal{L}(\theta)$ at θ_n gives

$$\mathcal{L}(\theta_{n+1}) = \mathcal{L}(\theta_n) + (\theta_{n+1} - \theta_n) \cdot \nabla_\theta \mathcal{L}(\theta_n) + \frac{1}{2}(\theta_{n+1} - \theta_n)^T \nabla_\theta^2 \mathcal{L}(\xi)(\theta_{n+1} - \theta_n) \tag{5.8}$$

where $\xi = t\theta_n + (1t)\theta_n + 1$ for some $t \in [0,1]$ and $\nabla_\theta^2 \mathcal{L}(\xi)$ is the Hessian matrix of total loss. Substituting Equation (5.7) into Equation (5.8), the following is obtained

$$\mathcal{L}(\theta_{n+1}) - \mathcal{L}(\theta_n) = \eta ||\nabla_\theta \mathcal{L}(\theta_n)||_2^2 (-1 + \frac{1}{2}\eta \sum_{i=1}^{N} \lambda_i y_i^2) \tag{5.9}$$

$$\mathcal{L}_r(\theta_{n+1}) - \mathcal{L}_r(\theta_n) = \eta ||\nabla_\theta \mathcal{L}(\theta_n)||_2^2 (-1 + \frac{1}{2}\eta \sum_{i=1}^{N} \lambda_i^r y_i^2) \tag{5.10}$$

$$\mathcal{L}_d(\theta_{n+1}) - \mathcal{L}_d(\theta_n) = \eta ||\nabla_\theta \mathcal{L}(\theta_n)||_2^2 (-1 + \frac{1}{2}\eta \sum_{i=1}^{N} \lambda_i^d y_i^2) \tag{5.11}$$

where λ_i, λ_i^r, λ_i^d, given $i = \{1, \cdots, N\}$, are the eigenvalues of the Hessian matrices $\nabla_\theta^2 \mathcal{L}(\xi)$, $\nabla_\theta^2 \mathcal{L}_r(\xi)$ and $\nabla_\theta^2 \mathcal{L}_d(\xi)$ respectively, for some $\boldsymbol{y} = (y_1, \ldots y_N)$ satisfying $||\boldsymbol{y}|| = 1$. If gradient flow is stiff in any loss terms then many of the eigenvalues of the Hessian may be large, and consequently, this causes gradient decent to increase the loss per training iteration. Wang et al. [67] showed for the 2-D Helmholtz equation, described by $u(x,y) = \sin(a_1 \pi x)\sin(a_2 \pi y)$, large eigenvalues are

present in the Hessian of the total loss when the coefficients a_1 and a_2 increased in size. This resulted in suboptimal convergence of the network and pointed towards stiff gradient flow.

In their second point of analysis, Wang et al. [67] showed that the eigenvalues of the Hessians of the individual loss terms can yield an upper bound on their gradients. These upper bounds are formed by applying the classical Sobolev inequality [101] to $\nabla_\theta \mathcal{L}_r$ and $\nabla_\theta \mathcal{L}_d$ as shown below.

$$||\nabla_\theta \mathcal{L}_r(\theta)||_{L^\infty} \leq D||\nabla^2_\theta \mathcal{L}_r(\theta)||_{L^\infty} \tag{5.12}$$

$$||\nabla_\theta \mathcal{L}_d(\theta)||_{L^\infty} \leq D||\nabla^2_\theta \mathcal{L}_d(\theta)||_{L^\infty} \tag{5.13}$$

where D is a constant that does not depend on $\nabla_\theta \mathcal{L}_r(\theta)$ and $\nabla_\theta \mathcal{L}_{u_b}(\theta)$. By diagonalising $\nabla^2_\theta \mathcal{L}_r(\theta)$ and $\nabla^2_\theta \mathcal{L}_b(\theta)$ with orthonormal matrices, $||\nabla^2_\theta \mathcal{L}_r(\theta)||_{L^\infty}$ and $||\nabla^2_\theta \mathcal{L}_{u_b}(\theta)||_{L^\infty}$ are bounded by the largest eigenvalues of $\mathcal{L}_r(\theta)$ and $\mathcal{L}_b(\theta)$ respectively. Based on the Helmholtz example described earlier, loose upper bounds were formed in $\nabla_\theta \mathcal{L}_r(\theta)$ and $\nabla_\theta \mathcal{L}_d(\theta)$ indicating that the physics loss gradients have the capacity to be much larger than the data loss terms depending on the nature of the physics loss equations, which can lead to imbalances when training.

In order to mitigate these issues, Wang et al. [67] developed a learning rate annealing algorithm. A simplified version of this algorithm is described by Algorithm 2, which rebalances the gradients during learning.

Algorithm 2 Learning-rate annealing for physics-informed neural networks

Require: Learning rate η

Require: Dataset $\mathcal{D}$ of mini-batches

Require: Physics-informed neural network $f(x;\theta)$

1: Define loss components:

$$\mathcal{L}_r(\theta) \in \mathbb{R}, \quad \mathcal{L}_i(\theta) \in \mathbb{R}, \quad i = 1,\ldots,M$$

▷ $\mathcal{L}_r$ is the PDE residual loss; $\mathcal{L}_i$ are data-fit losses

2: Define total loss:

$$\mathcal{L}(\theta) = \mathcal{L}_r(\theta) + \sum_{i=1}^{M} \lambda_i \mathcal{L}_i(\theta)$$

3: Initialise parameters θ, $\lambda_i = 1 \; \forall i \in \{1,\ldots,M\}$

4: Set momentum hyper-parameter α

5: **while** not converged **do**

6: **for all** $\mathbf{x}_{\text{batch}} \in \mathcal{D}$ **do**

7: Compute gradient norm of the residual loss:

$$g^r_{\max} = \max_k \left\| \nabla_{\theta_k} \mathcal{L}_r(\theta) \right\|, \quad g^i_{\text{mean}} = \left\| \overline{\nabla_{\theta_k} \mathcal{L}_i(\theta)} \right\|$$

8: Compute adaptive weights:

$$\hat{\lambda}_i = \frac{g^r_{\max}}{g^i_{\text{mean}}}, \quad i = 1,\ldots,M$$

9: Update loss weights using moving average:

$$\lambda_i \leftarrow (1-\alpha)\lambda_i + \alpha\hat{\lambda}_i, \quad i = 1,\ldots,M$$

10: Update θ by gradient descent on $\mathcal{L}$

11: **end for**

12: **end while**

The aim of this algorithm is to re-weight the loss terms such that their gradients during back-propagation are similar in magnitude. Their analysis shows that the physics loss term generally dominates over the data loss terms when training PINNs. Hence, a sensible re-weighting scheme would involve approximately normalising the gradients on the data loss terms by multiplying them with the mean of their gradients calculated in Line 8 of Algorithm 2, then multiplying the data loss by the maximum gradient of the physics loss calculated in Line 7 of Algorithm (2). Then a moving average is applied when updating λ_i to adaptively scale the weightings.

Bischof et al. [102] succinctly highlights the drawbacks of the approach. In brief, unbounded re-weightings can cause overshooting of the optima during gradient descent. Additionally, having a uniform scaling on all the loss terms can encourage the network to continue giving a large emphasis on regions of the domain which have already been learned.

The convergence difficulties encountered in PINNs are closely related to those present when optimising the Gaussian NLL, as both stem from gradient imbalances induced by dominating coefficients in the loss terms. Although the precise origin of this imbalance in PINNs depends on the underlying physical system, in the Gaussian NLL it can be directly attributed to the weighting of the inverse-variance $\frac{1}{\sigma^2(x)}$. This observation motivates the adaptation of Algorithm 2 to mitigate the convergence issues associated with the Gaussian NLL. The adapted algorithm is introduced in the following section.

5.2 Annealing μ algorithm

An MVE network is defined by $\mu(x) = \mu(x;\theta_\mu)$ and $\sigma^2 = \sigma^2(x;\theta_{\sigma^2})$ where $x \in \mathbb{R}^n$ and θ_μ and θ_{σ^2} are the sets of parameters within each network such that $\theta_\mu \cap \theta_\sigma^2 = \emptyset$. As recommended by Sluijterman et al. [40], the architecture has two separate subnetworks for $\mu(x;\theta_\mu)$ and $\sigma^2(x;\theta_{\sigma^2})$. This is such that WD can be applied independently to each subnetwork, which can mitigate underestimation of the variance, another known problem with the use of the Gaussian NLL function when fitting NNs. Tanh activation functions are used at the hidden layers of the network in order to improve the fitting of non-linear functions. A softmax activation function is applied to the output of the $\sigma^2(x;\theta_{\sigma^2})$ subnetwork to constrain the output to be positive. The network used in all experiments in this chapter is generally described by Figure 5.1.

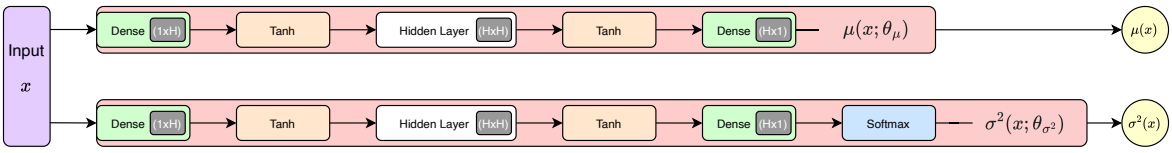


Figure 5.1: General architecture of MVE network consisting of two subnetworks, $\mu(x;\theta_{\mu^2})$ and $\sigma^2(x;\theta_{\sigma^2})$ with tanh activation functions and softmax applied to $\sigma^2(x;\theta_{\sigma^2})$. The size of H and the number of hidden layers are specified for each experiment separately

The novel learning rate annealing algorithm for solving the suboptimal convergence problem when using the Gaussian NLL is described by Algorithm 3.

Algorithm 3 Annealing μ algorithm for vector-valued Gaussian NLL

Require: Learning rate η

Require: Dataset $\mathcal{D}$ of mini-batches

Require: Pointwise Gaussian NLL loss $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}^d$

1: Initialise parameters $\theta = \theta_\mu \cup \theta_{\sigma^2}$ and $\lambda_\mu = 1$

2: Set momentum hyper-parameter $\alpha = 0.9$

3: **while** not converged **do**

4: **for all** $\mathbf{x}_{\text{batch}} \in \mathcal{D}$ **do**

5: Compute vector variances $\boldsymbol{\sigma}^2 = \sigma^2(\mathbf{x}_{\text{batch}}) \in \mathbb{R}^d$

6: Compute $\sigma^2_{\text{min}} = \min_i \sigma_i^2$, $\sigma^2_{\text{max}} = \max_i \sigma_i^2$

7: Scale variance elementwise:

$$\boldsymbol{\sigma}^2_{*_\mu} = \frac{\boldsymbol{\sigma}^2 - \sigma^2_{\text{min}}}{\sigma^2_{\text{max}} - \sigma^2_{\text{min}}} \in \mathbb{R}^d$$

8: Compute pointwise loss vector $\mathcal{L} = \mathcal{L}(\mathbf{x}_{\text{batch}}) \in \mathbb{R}^d$

9: Define weighted vector losses:

$$\hat{\mathcal{L}}_{S_\mu} = \boldsymbol{\sigma}^2_{*_\mu} \odot \mathcal{L}, \quad \hat{\mathcal{L}}_{L_\mu} = (1 - \boldsymbol{\sigma}^2_{*_\mu}) \odot \mathcal{L}$$

$$\hat{\mathcal{L}}_{\lambda_\mu} = \lambda_\mu \hat{\mathcal{L}}_{S_\mu} + \hat{\mathcal{L}}_{L_\mu}$$

10: Compute reduced losses:

$$\mathcal{L}_{L_\mu} = \frac{1}{d}\sum_{i=1}^{d} \hat{\mathcal{L}}_{L_\mu,i}, \quad \mathcal{L}_{S_\mu} = \frac{1}{d}\sum_{i=1}^{d} \hat{\mathcal{L}}_{S_\mu,i} \quad \mathcal{L}_{\lambda_\mu} = \frac{1}{d}\sum_{i} \hat{\mathcal{L}}_{\lambda_\mu,i}$$

11: Compute gradient statistics:

$$g^\mu_{\text{max}} = \max \left\| \nabla_{\theta_\mu} \mathcal{L}_{L_\mu} \right\|, \quad g^\mu_{\text{mean}} = \left\| \overline{\nabla_{\theta_\mu} \mathcal{L}_{S_\mu}} \right\|$$

12: Update annealing coefficient:

$$\hat{\lambda}_\mu = \frac{g^\mu_{\text{max}}}{g^\mu_{\text{mean}}}, \quad \lambda_\mu \leftarrow (1-\alpha)\lambda_\mu + \alpha\hat{\lambda}_\mu$$

13: Update θ by gradient descent on $\mathcal{L}_{\lambda_\mu}$

14: **end for**

15: **end while**

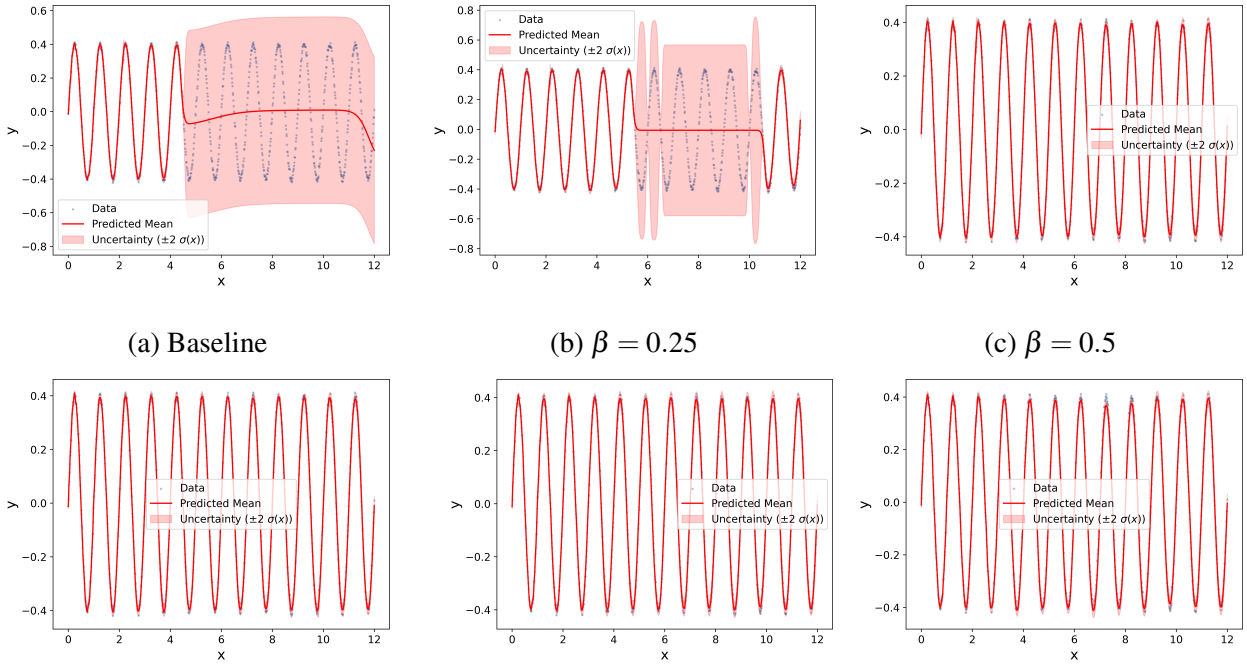
The algorithm initially recasts the single objective Gaussian NLL problem into a multi-objective one. This is achieved by initially computing a vector, $\boldsymbol{\sigma}^2_{*_\mu}$, consisting of values of $\boldsymbol{\sigma}^2$ rescaled between 0 and 1 as shown by Line 7. Then three vector losses are calculated on Line 9. Firstly $\mathcal{L}_{S_\mu}$ is calculated, which corresponds to a vector on the pointwise losses $\mathcal{L}$ weighted by the scaled variance at each point. Secondly, $\mathcal{L}_{L_\mu}$ is calculated, which corresponds to a vector on pointwise losses $\mathcal{L}$, weighted by 1 minus the scaled variance at each point. $\mathcal{L}_{S_\mu}$ and $\mathcal{L}_{L_\mu}$ can be thought of as losses in areas where there may be small and large gradients, informed by the magnitude of the variance. This allows for the calculation of the gradient statistics shown by Line 11 and ultimately calculates $\mathcal{L}_{\lambda_\mu}$ on Line 9. Momentum is also applied to the updates of λ_μ at each iteration of gradient descent to speed up learning and increase convergence [50]. In contrast to the $\beta-$NLL method where loss

re-weighting is prescribed heuristically, the Algorithm 3 dynamically determines re-weighting on losses from gradient statistics, directly targeting the gradient imbalances.

Models trained using Algorithm 3 will be referred to as the annealing μ model and models trained using the $\beta-$NLL model will be referred to as the $\beta = k$ model for a specified value of $0 \leq k \leq 1$, or $\beta \in \{l, m, n\}$ models to specify multiple $\beta-$NLL models.

5.3 Simple sinusoidal homoscedastic noise example

In this section, MVE networks are trained on data generated from a sine wave function with homoscedastic noise to showcase the convergence issue on a baseline model, which was compared to $\beta \in \{0.25, 0.5, 0.75, 1\}$ and annealing μ models. The training dataset consisted of 1000 linearly spaced points across the domain $x \in [0, 12]$ with the corresponding outputs $y = 0.4\sin(2\pi x) + \varepsilon$ where $\varepsilon \sim \mathcal{N}(0, 0.01)$. The MVE network had 2 hidden layers with 128 units within each subnetwork of the model. The models were evaluated at 500 linearly spaced test points throughout the domain. Each network was trained for 100000 epochs with learning rate = 5E-04. This should be a very simple learning problem and any standard choice of NN and learning parameters should lead to convergence, yet, as shown by Figure 5.2 (a), the baseline model struggles to converge.



(a) Baseline (b) $\beta = 0.25$ (c) $\beta = 0.5$

(d) $\beta = 0.75$ (e) $\beta = 1$ (f) Annealing μ

Figure 5.2: Plots of baseline, $\beta \in [0.25, 0.5, 0.75, 1]$ and annealing μ models on the homoscedastic noise example after 100000 epochs.

Figure 5.2 shows that the model $\beta = 0.25$ improves on the baseline model but still does not converge between $x \in [6, 10]$. As expected, the $\beta \in [0.5, 0.75, 1]$ models converge across the entire domain. Finally, the annealing μ model is also capable of converging to the solution.

The approach for analysis conducted by Wang et al. [67] for the Physics-Informed Neural Network (PINN) convergence problem, as described in Chapter 5.1.1, is reapplied to the baseline model after 100000 epochs. For this analysis, two losses defined in Algorithm 3, $\mathcal{L}_{S_\mu}$ and $\mathcal{L}_{L_\mu}$, are calculated. The losses represent areas where the predictive variance is large and small, respectively.

The motivation for this analysis is to determine whether the suboptimal convergence in the baseline model is due to the same imbalances commonly found in between the PINNs loss functions. By splitting the losses into high and low variance contributions, the analysis will show if the key contributor to the suboptimal convergence issues is due to inverse scaling by the predictive variances. Furthermore, if the insights show congruence to the issues found in PINNs, then it further validates the approach to adapting their algorithm to address the convergence issues when using the Gaussian NLL.

To start the analysis, the gradient flow stiffness is observed for $\mathcal{L}_{S_\mu}$ and $\mathcal{L}_{L_\mu}$. The gradient flow equations are given by Equations (5.15) and (5.16).

$$\mathcal{L}(\theta_{n+1}) - \mathcal{L}(\theta_n) = \eta ||\nabla_\theta \mathcal{L}(\theta_n)||_2^2 (-1 + \frac{1}{2}\eta \sum_{i=1}^{N} \lambda_i y_i^2) \tag{5.14}$$

$$\mathcal{L}_{S_\mu}(\theta_{n+1}) - \mathcal{L}_{S_\mu}(\theta_n) = \eta ||\nabla_\theta \mathcal{L}_{S_\mu}(\theta_n)||_2^2 (-1 + \frac{1}{2}\eta \sum_{i=1}^{N} \lambda_i^{S_\mu} y_i^2) \tag{5.15}$$

$$\mathcal{L}_{L_\mu}(\theta_{n+1}) - \mathcal{L}_{L_\mu}(\theta_n) = \eta ||\nabla_\theta \mathcal{L}_{L_\mu}(\theta_n)||_2^2 (-1 + \frac{1}{2}\eta \sum_{i=1}^{N} \lambda_i^{L_\mu} y_i^2) \tag{5.16}$$

Figure 5.3 shows that the 100 largest eigenvalues of $\nabla_\theta^2 \mathcal{L}_{S_\mu}$ are much larger than the 100 largest eigenvalues of $\nabla_\theta^2 \mathcal{L}_{L_\mu}$ in the baseline model. The stiffness of the gradient flow is therefore dominated by $\mathcal{L}_{L_\mu}$ and it is possible that, in some cases, $\mathcal{L}(\theta_{n+1}) - \mathcal{L}(\theta_n) > 0$ resulting in the failure of the gradient descent method in decreasing the loss.

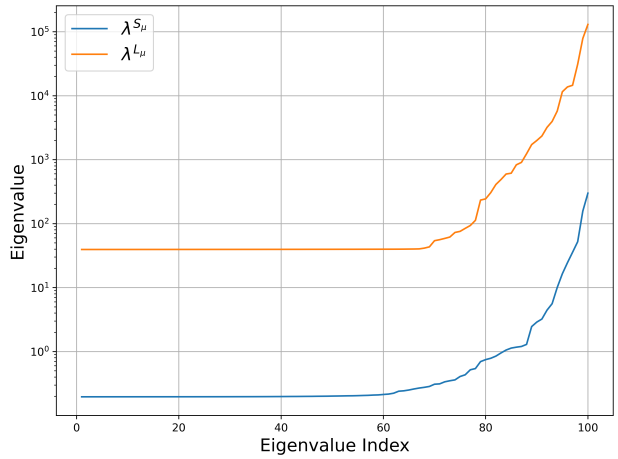


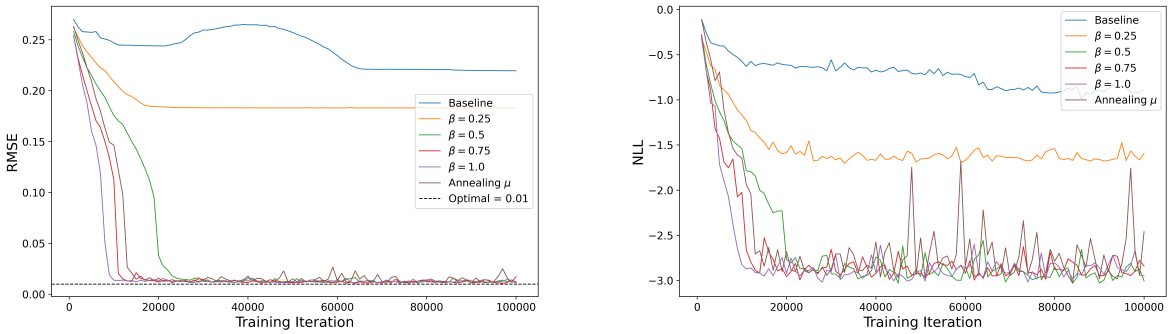
Figure 5.3: 100 largest eigenvalues of $\nabla_{\theta_\mu}^2 \mathcal{L}_{S_\mu}(\theta_\mu)$ and $\nabla_{\theta_\mu}^2 \mathcal{L}_{L_\mu}(\theta_\mu)$ arranged in increasing order of the baseline model trained on the homoscedastic noise example after 100000 epochs. Note that the model fails to converge and learn further after this point.

Secondly, the upper bounds on $\nabla_{\theta_\mu} \mathcal{L}_{S_\mu}$ and $\nabla_{\theta_\mu} \mathcal{L}_{L_\mu}$ are found by applying the classical Sobolev inequality to give the following

$$||\nabla_\theta \mathcal{L}_{S_\mu}(\theta)||_{L^\infty} \leq D ||\nabla_\theta^2 \mathcal{L}_{S_\mu}(\theta)||_{L^\infty} \tag{5.17}$$

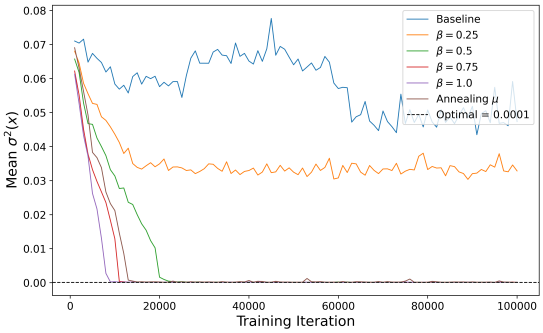
$$||\nabla_\theta \mathcal{L}_{L_\mu}(\theta)||_{L^\infty} \leq D||\nabla^2_\theta \mathcal{L}_{L_\mu}(\theta)||_{L^\infty} \tag{5.18}$$

Taking into account the size of the eigenvalues in Figure 5.3 and Equations (5.17) and (5.18), it can be inferred that the gradients $\nabla^2_{\theta_\mu} \mathcal{L}_{L_\mu}$ have the potential to be much larger than $\nabla^2_{\theta_\mu} \mathcal{L}_{S_\mu}$ and therefore the regions of the model with small variance are likely to dominate during training. Figure 5.4 shows the performance (RMSE, NLL and Mean $\sigma^2(x)$) across the training iteration for each of the models.



(a) RMSE

(b) NLL

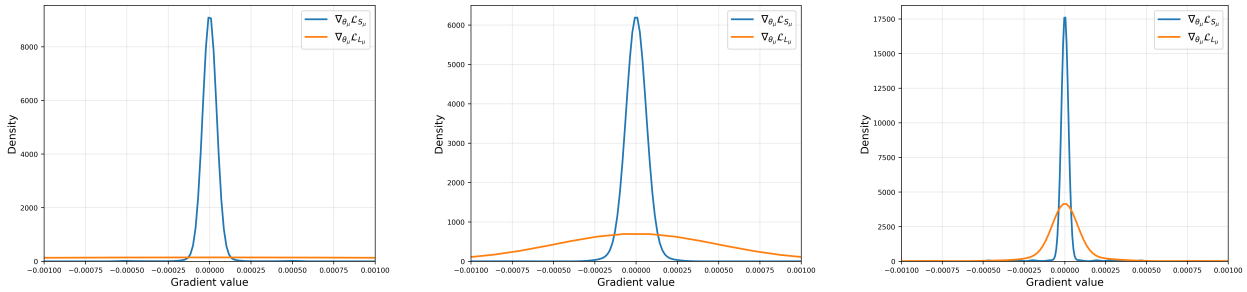


(c) Mean $\sigma^2(x)$

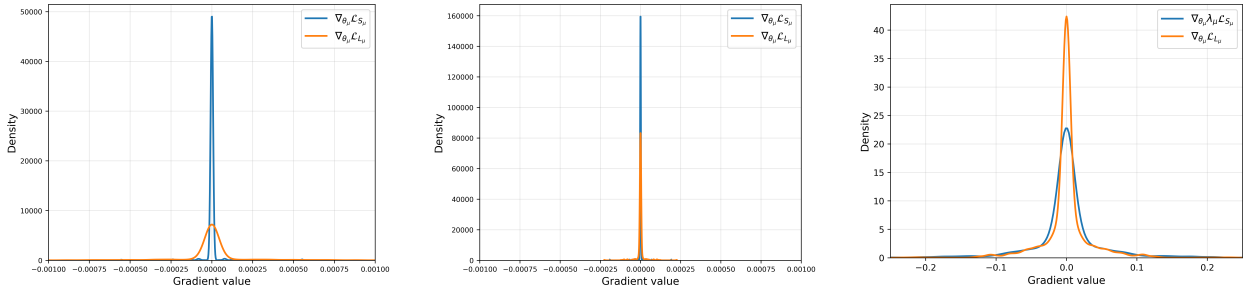
Figure 5.4: RMSE, NLL and Mean $\sigma^2(x)$ across 100000 training iterations (epochs) for the baseline model, $\beta \in [0.25, 0.5, 0.75, 1]$ and annealing μ model for the homoscedastic noise example.

Figure 5.4 shows that the annealing μ model is marginally slower to converge to the optimal RMSE and Mean $\sigma^2(x)$ than the $\beta \in \{0.75, 1\}$ models. Furthermore, the annealing μ model also shows more fluctuation in the RMSE and NLL at convergence than the other models. The large fluctuations could be due to the overshooting or over-fitting of the model.

Analysis was conducted on the gradients of the losses $\mathcal{L}_{L_\mu}$ and $\mathcal{L}_{S_\mu}$, defined in Algorithm 5.2, w.r.t the weights of the last hidden layer of the $\mu(x;\theta_\mu)$ subnetwork in the baseline and $\beta \in \{0.25, 0.5, 0.75, 1\}$ models. A slightly different analysis was conducted for the annealing μ model, where gradients of the losses $\lambda_\mu \mathcal{L}_{S_\mu}$ and $\mathcal{L}_{L_\mu}$, where both terms summed in the total loss for back-propagation. The gradients were observed for both models at 5000 epochs and 100000 epochs as shown by Figures 5.5 and 5.6. In a balanced learning scheme, the expectation is that the distributions of the gradients are reasonably similar.



(a) Baseline (b) $\beta = 0.25$ (c) $\beta = 0.5$

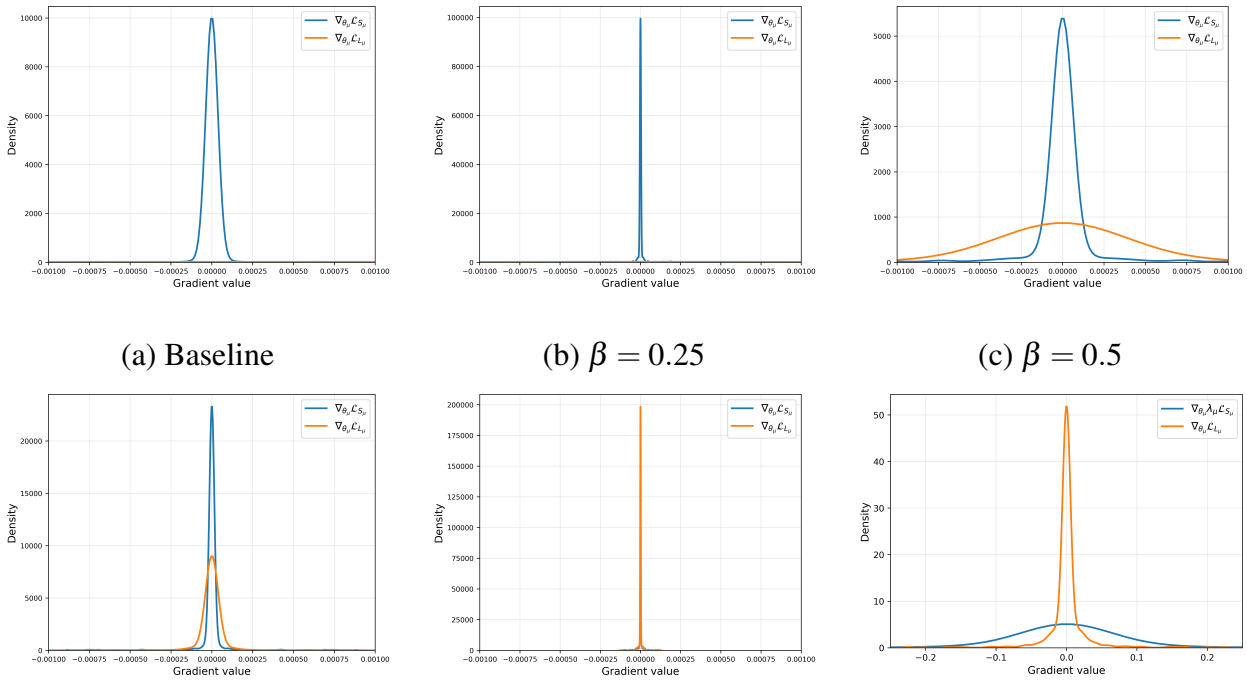


(d) $\beta = 0.75$ (e) $\beta = 1$ (f) Annealing μ

Figure 5.5: Density plots of $\nabla_{\theta_\mu}\mathcal{L}_{S_\mu}$ and $\nabla_{\theta_\mu}\mathcal{L}_{L_\mu}$ from the last hidden layer of $\mu(x;\theta_\mu)$ for the baseline, $\beta \in [0.25, 0.5, 0.75, 1]$ and annealing μ models trained on the homoscedastic noise example at 5000 epochs.

Figure 5.5 shows that in the baseline model, the gradients of $\mathcal{L}_{S_\mu}$ are sharply concentrated near 0 and therefore the learning is being dominated by $\mathcal{L}_{L_\mu}$, or dominated at the training points which have a low predictive variance. The general trend in the $\beta-$NLL models shows a decrease in the discrepancies between the densities of $\nabla_{\theta_\mu}\mathcal{L}_{S_\mu}$ and $\nabla_{\theta_\mu}\mathcal{L}_{L_\mu}$ as β increases. This suggests that in the early phase of training the $\beta-$NLL method rebalances the gradients of the losses at points of

high and low variances to increasing effectiveness as β increases. Additionally, in the annealing μ model, $\nabla_{\theta_\mu}\lambda_\mu\mathcal{L}_{S_\mu}$ has gradients larger than $\nabla_{\theta_\mu}\mathcal{L}_{L_\mu}$, suggesting that more emphasis is being placed on learning regions of the model where there is greater variance in the early phase of training.



(a) Baseline (b) $\beta = 0.25$ (c) $\beta = 0.5$

(d) $\beta = 0.75$ (e) $\beta = 1$ (f) Annealing μ

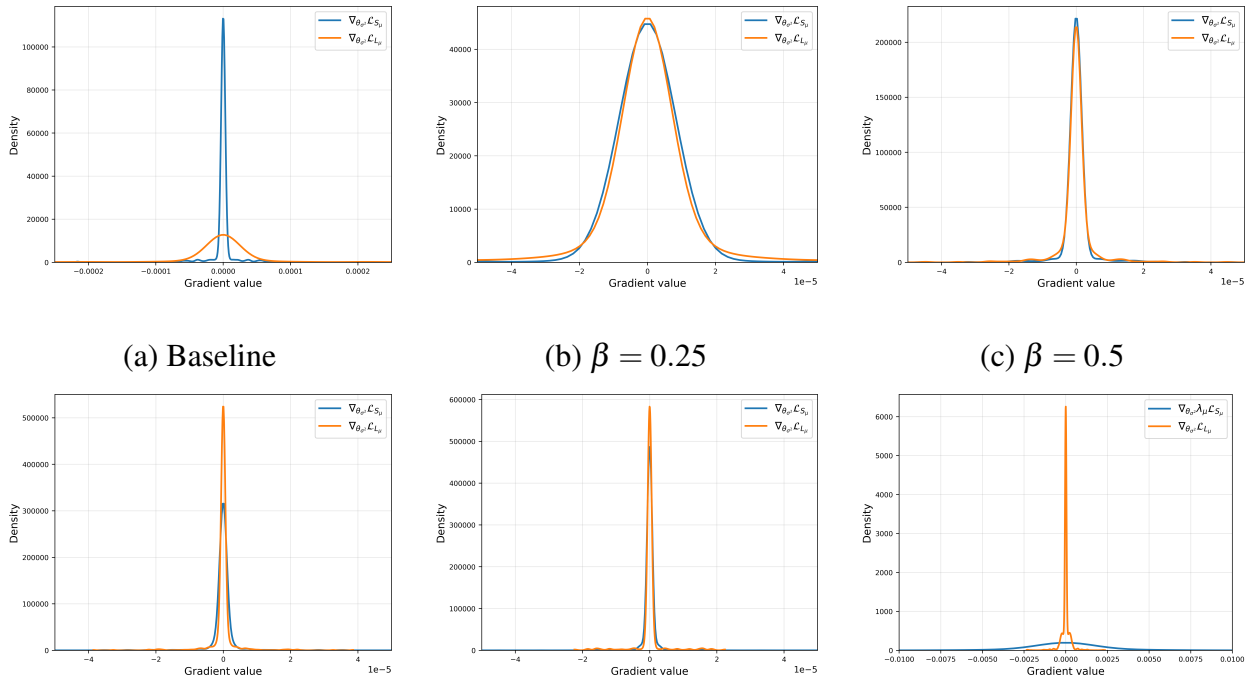
Figure 5.6: Density plots of the losses w.r.t. to the weights from the last hidden layer of $\mu(x;\theta_\mu)$ for the baseline, $\beta \in [0.25, 0.5, 0.75, 1]$ and annealing μ models trained on the homoscedastic noise example at 100000 epochs.

At 100000 epochs, Figure 5.6 shows virtually no change in the baseline model between the densities of the gradients compared to 5000 epochs. This is in agreement with the lack of convergence shown by Figure 5.2 (a). The $\beta-$NLL models show the same general trends as in Figure 5.5, however, the $\beta = 0.25$ model has a larger discrepancy between $\nabla_{\theta_\mu}\mathcal{L}_{S_\mu}$ and $\nabla_{\theta_\mu}\mathcal{L}_{L_\mu}$. This indicates that the $\mathcal{L}_{L_\mu}$ loss dominates the learning compared to the model at 5000 epochs. This supports the findings of Figure 5.2 (b) where the model fails to converge even after 100000 epochs. This could also suggest that the extent to which $\beta-$NLL can rebalance the gradients of the losses diminishes over time for low values of β. Finally, in the annealing μ model, the discrepancy between $\nabla_{\theta_\mu}\lambda_\mu\mathcal{L}_{S_\mu}$ and $\nabla_{\theta_\mu}\mathcal{L}_{L_\mu}$ is further exacerbated despite having achieved convergence. Finally,

the gradients in the annealing μ model are much larger than the other models which may explain the large fluctuations found in the NLL (Figure 5.4 (b)) for the annealing μ model and supports evidence of overshooting occurring.

Clearly, in this simple homoscedastic noise example, the $\beta \in \{0.75, 1\}$ models are still the optimal choice to address the convergence problem, as there is not much difference between the performance and the convergence rate against the annealing μ model. However, a choice of β is still required for each problem type which represents an advantage for the annealing μ model.

Similar analysis is conducted on the gradients of the losses w.r.t the weights of the $\sigma^2(x;\theta_{\sigma^2})$ subnetwork for all models at 5000 epochs, as shown by Figure 5.7.



(a) Baseline (b) $\beta = 0.25$ (c) $\beta = 0.5$

(d) $\beta = 0.75$ (e) $\beta = 1$ (f) Annealing μ

Figure 5.7: Density plots of the losses w.r.t. to the weights from the last hidden layer of $\sigma^2(x;\theta_{\sigma^2})$ for the baseline, $\beta \in [0.25, 0.5, 0.75, 1]$ and annealing μ models trained on the homoscedastic noise example at 5000 epochs.

Figure 5.7 large discrepancies in the gradients of the baseline model. However, in the $\beta-$NLL models, the densities are more balanced. In the $\beta \in \{0.75, 1\}$ models there seems to be slightly more emphasis in learning the $\mathcal{L}_{S_\mu}$ loss than the $\mathcal{L}_{L_\mu}$ loss. The annealing μ model shows large

domination by the $\mathcal{L}_{S_\mu}$ loss over the $\mathcal{L}_{L_\mu}$ loss. The purpose of the annealing μ algorithm is to mitigate the biases shown in $\nabla_\mu \mathcal{L}_{\text{NLL}}$ (Equation (5.2)), caused by $\frac{1}{\sigma^2(x)}$, however, in $\nabla_{\sigma^2} \mathcal{L}_{\text{NLL}}$ (Equation (5.3)), the biases would be caused by the denominator of the form $\frac{1}{2(\sigma^2(x))^2}$. The next section investigates whether variations of the annealing μ algorithm can balance the gradients in the $\sigma^2(x;\theta_{\sigma^2})$ subnetwork in addition to the $\mu(x;\theta_\mu)$ subnetwork during learning and whether this can improve the performance and convergence of the model.

5.3.1 Adaptations to the annealing μ algorithm

In the previous section, the annealing μ model was capable of solving the suboptimal convergence issue for the homoscedastic example. However, $\beta \in \{0.75, 1\}$ models demonstrated overall better performance, despite the need to tune the parameter β. In this section, adaptations to the algorithm are outlined where the gradients in both subnetworks ($\mu(x;\theta_\mu)$ and $\sigma^2(x;\theta_{\sigma^2})$) are balanced, where some improvement in performance over the annealing μ algorithm is expected. The same analysis on the gradients of the losses will be conducted in order to confirm the algorithms are doing what is intended.

The first adaptation of the annealing μ algorithm attempts to directly balance the gradients of the $\mu(x;\theta_\mu)$ subnetwork. The bias in Equation (5.3) was shown to occur from the denominator $\frac{1}{2(\sigma^2(x))^2}$. This motivates a change to the function $\boldsymbol{\sigma}^2_{*_\mu} = \frac{\boldsymbol{\sigma}^2 - \sigma^2_{\min}}{\sigma^2_{\max} - \sigma^2_{\min}} \in \mathbb{R}^d$ in Algorithm 3 to $\boldsymbol{\sigma}^2_{*_{\sigma^2}} = \frac{2(\boldsymbol{\sigma}^2)^2 - 2(\sigma^2_{\min})^2}{2(\sigma^2_{\max})^2 - 2(\sigma^2_{\min})^2} \in \mathbb{R}^d$ as described in Algorithm 4.

Algorithm 4 Annealing σ^2 algorithm for vector-valued Gaussian NLL

Require: Learning rate η

Require: Dataset $\mathcal{D}$ of mini-batches

Require: Pointwise Gaussian NLL loss $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}^d$

Initialise parameters $\theta = \theta_\mu \cup \theta_{\sigma^2}$, $\lambda_{\sigma^2} = 1$

Set momentum hyper-parameter $\alpha = 0.9$

while not converged **do**

  for all $\mathbf{x}_{\text{batch}} \in \mathcal{D}$ **do**

    Compute vector variances $\boldsymbol{\sigma}^2 = \sigma^2(\mathbf{x}_{\text{batch}}) \in \mathbb{R}^d$

    Compute $\sigma^2_{\text{min}} = \min_i \sigma_i^2$, $\sigma^2_{\text{max}} = \max_i \sigma_i^2$

    Scale variance elementwise:

$$\boldsymbol{\sigma}^2_{*_{\sigma^2}} = \frac{2(\boldsymbol{\sigma}^2)^2 - 2(\sigma^2_{\text{min}})^2}{2(\sigma^2_{\text{max}})^2 - 2(\sigma^2_{\text{min}})^2} \in \mathbb{R}^d$$

    Compute pointwise loss vector $\mathcal{L} = \mathcal{L}(\mathbf{x}_{\text{batch}}) \in \mathbb{R}^d$

    Define weighted vector losses:

$$\hat{\mathcal{L}}_{S_{\sigma^2}} = \boldsymbol{\sigma}^2_{*_{\sigma^2}} \odot \mathcal{L}, \quad \hat{\mathcal{L}}_{L_{\sigma^2}} = (1 - \boldsymbol{\sigma}^2_{*_{\sigma^2}}) \odot \mathcal{L}$$

$$\hat{\mathcal{L}}_{\lambda_{\sigma^2}} = \lambda_{\sigma^2} \hat{\mathcal{L}}_{S_{\sigma^2}} + \hat{\mathcal{L}}_{L_{\sigma^2}}$$

    Compute reduced losses:

$$\mathcal{L}_{L_{\sigma^2}} = \frac{1}{d} \sum_{i=1}^{d} \hat{\mathcal{L}}_{L_{\sigma^2},i}, \quad \mathcal{L}_{S_{\sigma^2}} = \frac{1}{d} \sum_{i=1}^{d} \hat{\mathcal{L}}_{S_{\sigma^2},i}, \quad \mathcal{L}_{\lambda_{\sigma^2}} = \frac{1}{d} \sum_{i} \hat{\mathcal{L}}_{\lambda_{\sigma^2},i}$$

    Compute gradient statistics:

$$g^{\sigma^2}_{\text{max}} = \max \left\| \nabla_{\theta_{\sigma^2}} \mathcal{L}_{L_{\sigma^2}} \right\|, \quad g^{\sigma^2}_{\text{mean}} = \left\| \overline{\nabla_{\theta_{\sigma^2}} \mathcal{L}_{S_{\sigma^2}}} \right\|$$

    Update annealing coefficient:

$$\hat{\lambda}_{\sigma^2} = \frac{g^{\sigma^2}_{\text{max}}}{g^{\sigma^2}_{\text{mean}}}, \quad \lambda_{\sigma^2} \leftarrow (1 - \alpha)\lambda_{\sigma^2} + \alpha \hat{\lambda}_{\sigma^2}$$

    Update θ by gradient descent on $\mathcal{L}_{\lambda_{\sigma^2}}$

  end for

end while

Note that in Algorithm 4, the maximum gradient values and mean gradient values are calculated using weights from the $\sigma^2(x;\theta_{\sigma^2})$ subnetwork, as opposed to the $\mu(x;\theta_\mu)$ subnetwork as shown in Algorithm 3. There is no expectation that the annealing σ^2 algorithm will improve over the performance of the annealing μ algorithm, since the suboptimal convergence issue is mainly related to the inability to learn μ. However, an analysis of the gradients of $\mathcal{L}_{L_{\sigma^2}}$ and $\lambda_{\sigma^2}\mathcal{L}_{S_{\sigma^2}}$ should show little discrepancy between the distribution of the gradients. This would validate the approach for balancing the gradients in the $\sigma^2(x;\theta_{\sigma^2})$ subnetwork and is the springboard for two further adaptations to Algorithm 3 which attempt to balance the gradients of both subnetworks as described by Algorithms 5 and 6.

Algorithm 5 Annealing μ and σ^2 (Single Loss) (I) for vector-valued Gaussian NLL

Require: Learning rate η

Require: Dataset $\mathcal{D}$ of mini-batches

Require: Pointwise Gaussian NLL loss $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}^d$

Initialise parameters $\theta = \theta_\mu \cup \theta_{\sigma^2}$, $\lambda_\mu = 1$, $\lambda_{\sigma^2} = 1$

Set momentum hyper-parameter $\alpha = 0.9$

while not converged **do**

  for all $\mathbf{x}_{\text{batch}} \in \mathcal{D}$ **do**

    Compute vector variances $\boldsymbol{\sigma}^2 = \sigma^2(\mathbf{x}_{\text{batch}}) \in \mathbb{R}^d$

    Compute $\sigma^2_{\min} = \min_i \sigma_i^2$, $\sigma^2_{\max} = \max_i \sigma_i^2$

    Scale variances elementwise:

$$\boldsymbol{\sigma}^2_{*_\mu} = \frac{\boldsymbol{\sigma}^2 - \sigma^2_{\min}}{\sigma^2_{\max} - \sigma^2_{\min}}, \quad \boldsymbol{\sigma}^2_{*_{\sigma^2}} = \frac{2(\boldsymbol{\sigma}^2)^2 - 2(\sigma^2_{\min})^2}{2(\sigma^2_{\max})^2 - 2(\sigma^2_{\min})^2}$$

    Compute pointwise loss vector $\mathcal{L} = \mathcal{L}(\mathbf{x}_{\text{batch}})$

    Define weighted vector losses:

$$\hat{\mathcal{L}}_{S_\mu} = \boldsymbol{\sigma}^2_{*_\mu} \odot \mathcal{L}, \quad \hat{\mathcal{L}}_{L_\mu} = (1 - \boldsymbol{\sigma}^2_{*_\mu}) \odot \mathcal{L}$$

$$\hat{\mathcal{L}}_{S_{\sigma^2}} = \boldsymbol{\sigma}^2_{*_{\sigma^2}} \odot \mathcal{L}, \quad \hat{\mathcal{L}}_{L_{\sigma^2}} = (1 - \boldsymbol{\sigma}^2_{*_{\sigma^2}}) \odot \mathcal{L}$$

$$\hat{\mathcal{L}}_{\lambda_\mu} = \lambda_\mu \hat{\mathcal{L}}_{S_\mu} + \hat{\mathcal{L}}_{L_\mu}$$

$$\hat{\mathcal{L}}_{\lambda_{\sigma^2}} = \lambda_{\sigma^2} \hat{\mathcal{L}}_{S_{\sigma^2}} + \hat{\mathcal{L}}_{L_{\sigma^2}}$$

    Compute reduced losses:

$$\mathcal{L}_{L_\mu} = \frac{1}{d}\sum_{i=1}^{d} \hat{\mathcal{L}}_{L_\mu,i}, \quad \mathcal{L}_{S_\mu} = \frac{1}{d}\sum_{i=1}^{d} \hat{\mathcal{L}}_{S_\mu,i}, \quad \mathcal{L}_{\lambda_\mu} = \frac{1}{d}\sum_i \hat{\mathcal{L}}_{\lambda_\mu,i}$$

$$\mathcal{L}_{L_{\sigma^2}} = \frac{1}{d}\sum_{i=1}^{d} \hat{\mathcal{L}}_{L_{\sigma^2},i}, \quad \mathcal{L}_{S_{\sigma^2}} = \frac{1}{d}\sum_{i=1}^{d} \hat{\mathcal{L}}_{S_{\sigma^2},i}, \quad \mathcal{L}_{\lambda_{\sigma^2}} = \frac{1}{d}\sum_i \hat{\mathcal{L}}_{\lambda_{\sigma^2},i}$$

    Compute gradient statistics:

$$g^\mu_{\max} = \max \left\| \nabla_{\theta_\mu} \mathcal{L}_{L_\mu} \right\|, \quad g^\mu_{\text{mean}} = \left\| \overline{\nabla_{\theta_\mu} \mathcal{L}_{S_\mu}} \right\|$$

$$g^{\sigma^2}_{\max} = \max \left\| \nabla_{\theta_{\sigma^2}} \mathcal{L}_{L_{\sigma^2}} \right\|, \quad g^{\sigma^2}_{\text{mean}} = \left\| \overline{\nabla_{\theta_{\sigma^2}} \mathcal{L}_{S_{\sigma^2}}} \right\|$$

    Update annealing coefficients:

$$\hat{\lambda}_\mu = \frac{g^\mu_{\max}}{g^\mu_{\text{mean}}}, \quad \hat{\lambda}_{\sigma^2} = \frac{g^{\sigma^2}_{\max}}{g^{\sigma^2}_{\text{mean}}}$$

$$\lambda_\mu \leftarrow (1-\alpha)\lambda_\mu + \alpha\hat{\lambda}_\mu, \quad \lambda_{\sigma^2} \leftarrow (1-\alpha)\lambda_{\sigma^2} + \alpha\hat{\lambda}_{\sigma^2}$$

    Update θ by gradient descent on $\mathcal{L}_{\lambda_\mu} + \mathcal{L}_{\lambda_{\sigma^2}}$

  end for

end while

Algorithm 6 Annealing μ and σ^2 (Separate Updates) (II) for vector-valued Gaussian NLL

Require: Learning rate η

Require: Dataset $\mathcal{D}$ of mini-batches

Require: Pointwise Gaussian NLL loss $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}^d$

Initialise parameters θ_μ, θ_{σ^2}, $\lambda_\mu = 1$, $\lambda_{\sigma^2} = 1$

Set momentum hyper-parameter $\alpha = 0.9$

while not converged **do**

  for all $\mathbf{x}_{\text{batch}} \in \mathcal{D}$ **do**

    Compute $\boldsymbol{\sigma}^2 = \sigma^2(\mathbf{x}_{\text{batch}})$, σ^2_{min}, σ^2_{max}

    Scale variance for μ:

$$\boldsymbol{\sigma}^2_{*_\mu} = \frac{\boldsymbol{\sigma}^2 - \sigma^2_{\text{min}}}{\sigma^2_{\text{max}} - \sigma^2_{\text{min}}}$$

    Compute loss vector $\mathcal{L} = \mathcal{L}(\mathbf{x}_{\text{batch}})$

    Define weighted losses:

$$\hat{\mathcal{L}}_{S_\mu} = \boldsymbol{\sigma}^2_{*_\mu} \odot \mathcal{L}, \quad \hat{\mathcal{L}}_{L_\mu} = (1 - \boldsymbol{\sigma}^2_{*_\mu}) \odot \mathcal{L}$$

$$\hat{\mathcal{L}}_{\lambda_\mu} = \lambda_\mu \hat{\mathcal{L}}_{S_\mu} + \hat{\mathcal{L}}_{L_\mu}$$

    Compute reduced losses:

$$\mathcal{L}_{L_\mu} = \frac{1}{d}\sum_{i=1}^{d} \hat{\mathcal{L}}_{L_\mu,i}, \quad \mathcal{L}_{S_\mu} = \frac{1}{d}\sum_{i=1}^{d} \hat{\mathcal{L}}_{S_\mu,i} \quad \mathcal{L}_{\lambda_\mu} = \frac{1}{d}\sum_i \hat{\mathcal{L}}_{\lambda_\mu,i}$$

    Compute gradient statistics:

$$g^\mu_{\text{max}} = \max \left\| \nabla_{\theta_\mu} \mathcal{L}_{L_\mu} \right\|, \quad g^\mu_{\text{mean}} = \left\| \overline{\nabla_{\theta_\mu} \mathcal{L}_{S_\mu}} \right\|$$

    Update $\hat{\lambda}_\mu = g^\mu_{\text{max}} / g^\mu_{\text{mean}}$, $\lambda_\mu \leftarrow (1-\alpha)\lambda_\mu + \alpha\hat{\lambda}_\mu$

    Update θ_μ by gradient descent on the mean of $\mathcal{L}_{\lambda_\mu}$

    Scale variance for σ^2:

$$\boldsymbol{\sigma}^2_{*_{\sigma^2}} = \frac{2(\boldsymbol{\sigma}^2)^2 - 2(\sigma^2_{\text{min}})^2}{2(\sigma^2_{\text{max}})^2 - 2(\sigma^2_{\text{min}})^2}$$

    Define weighted losses:

$$\hat{\mathcal{L}}_{S_{\sigma^2}} = \boldsymbol{\sigma}^2_{*_{\sigma^2}} \odot \mathcal{L}, \quad \hat{\mathcal{L}}_{L_{\sigma^2}} = (1 - \boldsymbol{\sigma}^2_{*_{\sigma^2}}) \odot \mathcal{L}$$

$$\hat{\mathcal{L}}_{\lambda_{\sigma^2}} = \lambda_{\sigma^2} \hat{\mathcal{L}}_{S_{\sigma^2}} + \hat{\mathcal{L}}_{L_{\sigma^2}}$$

    Compute reduced losses:

$$\mathcal{L}_{L_{\sigma^2}} = \frac{1}{d}\sum_{i=1}^{d} \hat{\mathcal{L}}_{L_{\sigma^2},i}, \quad \mathcal{L}_{S_{\sigma^2}} = \frac{1}{d}\sum_{i=1}^{d} \hat{\mathcal{L}}_{S_{\sigma^2},i}, \quad \mathcal{L}_{\lambda_{\sigma^2}} = \frac{1}{d}\sum_i \hat{\mathcal{L}}_{\lambda_{\sigma^2},i}$$

    Compute gradient statistics:

$$g^{\sigma^2}_{\text{max}} = \max \left\| \nabla_{\theta_{\sigma^2}} \mathcal{L}_{L_{\sigma^2}} \right\|, \quad g^{\sigma^2}_{\text{mean}} = \left\| \overline{\nabla_{\theta_{\sigma^2}} \mathcal{L}_{S_{\sigma^2}}} \right\|$$

    Update $\hat{\lambda}_{\sigma^2} = g^{\sigma^2}_{\text{max}} / g^{\sigma^2}_{\text{mean}}$, $\lambda_{\sigma^2} \leftarrow (1-\alpha)\lambda_{\sigma^2} + \alpha\hat{\lambda}_{\sigma^2}$

    Update θ_{σ^2} by gradient descent on $\mathcal{L}_{\lambda_{\sigma^2}}$

  end for

end while

Note that Algorithm 5 simultaneously applies Algorithm 3 and Algorithm 4 for a batch of data using a single loss function. In contrast, Algorithm 6 applies Algorithm 3 and Algorithm 4

sequentially for a batch of data. Since both approaches attempt to balance the gradients in the $\mu(x;\theta_\mu)$ subnetwork, they are expected to perform at least as well as the annealing μ algorithm in solving the suboptimal convergence issue. Furthermore, the gradients between the loss pairs

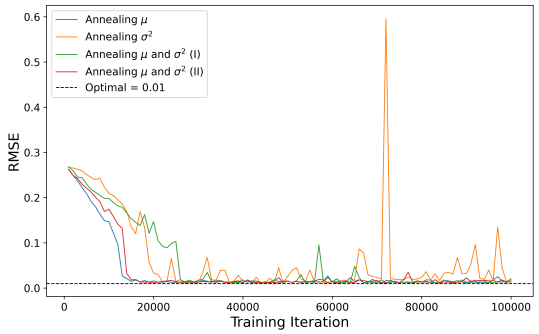
$$\mathcal{L}_{L_\mu} + \mathcal{L}_{L_{\sigma^2}} \text{ and } \lambda_\mu \mathcal{L}_{S_\mu} + \lambda_{\sigma^2} \mathcal{L}_{S_{\sigma^2}}$$

in the annealing μ and σ^2 (I) algorithm and

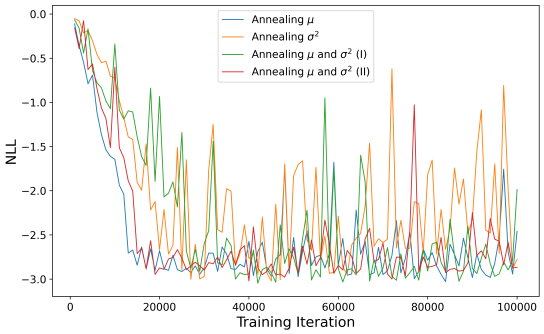
$$\mathcal{L}_{L_{\sigma^2}} \text{ and } \lambda_{\sigma^2} \mathcal{L}_{S_{\sigma^2}}$$

in the annealing μ and σ^2 (II) algorithm, should result in models with more balanced gradients compared to the baseline, $\beta-$NLL and annealing μ models during training.

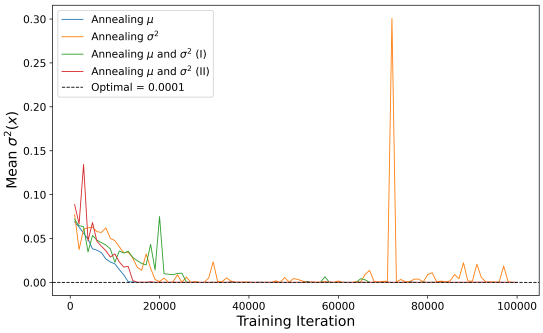
MVE models are trained using Algorithms 4, 5 and 6 using the same data, model architecture, and learning parameters described in Chapter 5.3. Henceforth, models trained using Algorithms 4, 5 and 6 will be referred to as the annealing σ^2, annealing μ and σ^2 (I) and annealing μ and σ^2 (II) models, respectively. Figure 5.8 shows the performance of the adapted models and the annealing μ model across 100000 epochs.



(a) RMSE



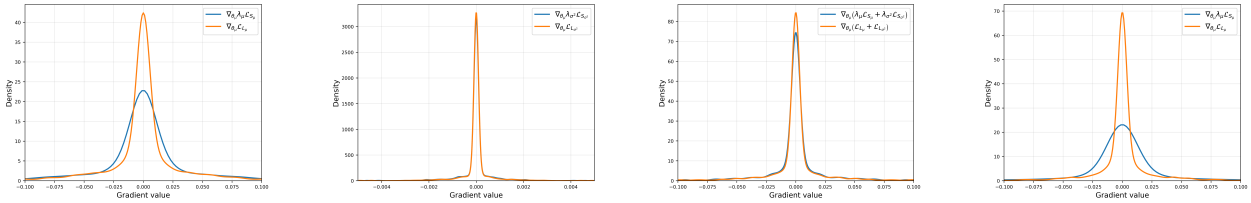
(b) NLL



(c) Mean $\sigma^2(x)$

Figure 5.8: RMSE, NLL and Mean $\sigma^2(x)$ across 100000 training iterations (epochs) for all annealing models for the homoscedastic noise example.

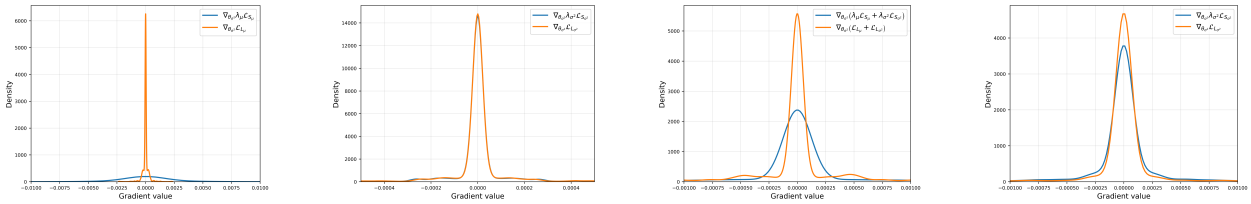
Figure 5.8 shows that all methods improve on the baseline model and the $\beta = 0.25$ model in achieving convergence. The annealing μ and annealing μ and σ^2 (II) models seem to converge at similar rates towards the optimal RMSE and Mean $\sigma^2(x)$ values. The other models converge, but show a lot more instability. This is most notable in the annealing σ^2 model, which has a large spike at around 70000 epochs. The annealing σ^2 model is able to converge close to the optimal values possibly due to the similarity in $\sigma^2_{*_\mu}$ and $\sigma^2_{*_{\sigma^2}}$ for this problem, i.e. the maximum and minimum variances on the data are very close together and resultantly $\boldsymbol{\sigma}^2_{*_{\sigma^2}} \approx \boldsymbol{\sigma}^2_{*_\mu}$. However, the annealing σ^2 model struggles to fully converge, this is possibly due to fact the maximum and mean gradient values, as described by Lines 11 in Algorithm 4, is informed by gradients w.r.t θ_{σ^2} compared to θ_μ in the Algorithm 3. Figure 5.9 shows the gradients of the losses used to train each annealing model's $\mu(x;\theta_\mu)$ subnetwork w.r.t to the weights of the last hidden layer of the $\mu(x;\theta_\mu)$ subnetwork.



(a) μ (b) σ^2 (c) μ and σ^2 (I) (d) μ and σ^2 (II)

Figure 5.9: Densities of gradients of the losses gradients used to train each annealing model's $\mu(x;\theta_\mu)$ subnetwork w.r.t to the weights of the last hidden layer of the $\mu(x;\theta_\mu)$ subnetwork at 5000 epochs.

Figure 5.9 shows that in all annealing models, the gradients between the two losses are healthily balanced, where neither of the loss terms seems to be dominating training. This reflects the results shown by Figure 5.8 where all models initially begin to converge at the initial stage of training. Figure 5.10 shows the gradients of the losses used to train each annealing model's $\sigma^2(x;\theta_{\sigma^2})$ subnetwork w.r.t to the weights of the last hidden layer of the $\sigma^2(x;\theta_{\sigma^2})$ subnetwork.



(a) μ (b) σ^2 (c) μ and σ^2 (I) (d) μ and σ^2 (II)

Figure 5.10: Densities of gradients of the losses gradients used to train each annealing model's $\sigma^2(x;\theta_\sigma^2)$ subnetwork w.r.t to the weights of the last hidden layer of the $\sigma^2(x;\theta_{\sigma^2})$ subnetwork at 5000 epochs.

Figure 5.10 shows evidence that the annealing σ^2 and both annealing μ and σ^2 models have the capacity to better balance the gradients between the two losses compared to the annealing μ model. The annealing μ and σ^2 (I) model seems to have slightly less balanced gradients compared to the annealing μ and σ^2 (II) model. From a single initialisation of the model and training, it is unclear whether these alternative methods improve the performance. However, there is an indication of greater balancing between the gradients of the loss terms when using the adapted algorithms.

Looking at the results in this section wholistically, the $\beta-$NLL method still seems to address the suboptimal convergence issue better than the newly developed algorithms. In particular, they demonstrate faster rates of convergence and less fluctuations in the NLL for the homoscedastic noise example. However, the annealing μ and σ^2 (I) and (II) methods remain promising candidates for further testing, as there may be scenarios where these methods are more advantageous. Firstly, repeat experiments are conducted in the next section on the homoscedastic noise example in order to better assess the effectiveness of these algorithms and determine if there is consistency in the results shown in this section.

5.3.2 Repeat experiments on the homoscedastic noise example

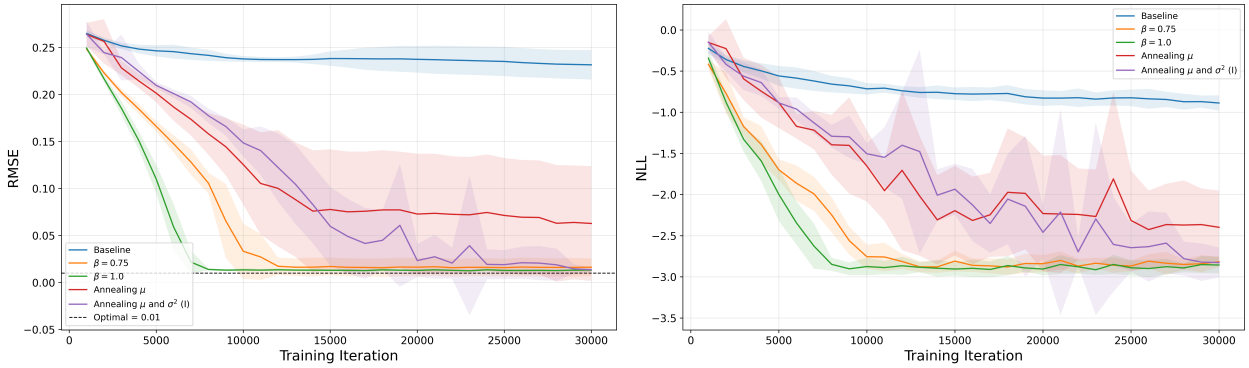
For the homoscedastic noise example, all methods, including the annealing μ variants, are trained for 10 different initialisations of the model for 30000 epochs. The training and test data, model architecture, and learning parameters are the same as described in Chapter 5.3. Table 5.1 shows the mean and standard deviation of the performance statistics (RMSE and NLL) in all repeat experiments.

Table 5.1: Mean and standard deviation of NLL and RMSE of the 10 repeated trainings of all model variants on the homoscedastic noise example experiments after 30000 epochs.

Model	**NLL**	**RMSE**
Baseline	-0.6936 $\pm$ 0.080	0.2402 $\pm$ 0.013
$\beta = 0.25$	-1.4553 $\pm$ 0.296	0.1905 $\pm$ 0.021
$\beta = 0.5$	-2.4930 $\pm$ 0.290	0.0784 $\pm$ 0.048
$\beta = 0.75$	-2.8104 $\pm$ 0.114	0.0220 $\pm$ 0.026
$\beta = 1$	-2.8672 $\pm$ 0.098	0.0135 $\pm$ 0.002
Annealing μ	-2.3392 $\pm$ 0.470	0.0600 $\pm$ 0.057
Annealing σ^2	-2.3385 $\pm$ 0.446	0.0477 $\pm$ 0.049
Annealing μ and σ^2 (I)	-2.7893 $\pm$ 0.210	0.0142 $\pm$ 0.003
Annealing μ and σ^2 (II)	-2.0162 $\pm$ 0.696	0.0815 $\pm$ 0.068

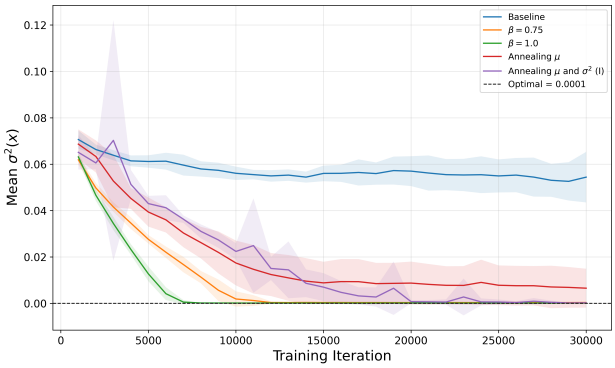
Table 5.1 shows that the $\beta = 1$, $\beta = 0.75$ and the annealing μ and σ^2 (I) models are the top performing models, both in terms of NLL and RMSE. Although the difference is marginal, it seems that for the homoscedastic noise example, the annealing μ and σ^2 (I) model offers an improvement

over the annealing μ model. All of the annealing models show large variances in the NLL values compared to the other models. This is in agreement with the large fluctuations in the NLL shown in Figure 5.8 (b). Figure 5.11 shows the performance over training iterations for the homoscedastic repeat experiments for the top performing models.



(a) RMSE

(b) NLL



(c) Mean $\sigma^2(x)$

Figure 5.11: Mean and standard deviation of the RMSE, NLL and Mean $\sigma^2(x)$ of the 10 repeated homoscedastic noise experiments across 30000 training iterations (epochs) for the baseline, $\beta \in \{0.75, 1\}$, annealing μ and annealing μ and $\sigma^2(x)$ (I).

Figure 5.11 shows that the $\beta \in \{0.75, 1\}$ models converge faster than the annealing models and have a lower variance in comparison. The annealing μ and σ^2 (I) model seems to converge faster in all instances compared to the annealing μ model, suggesting that an attempt to balance the loss gradients in $\sigma^2(x; \theta_{\sigma^2})$ can yield an improvement in performance.

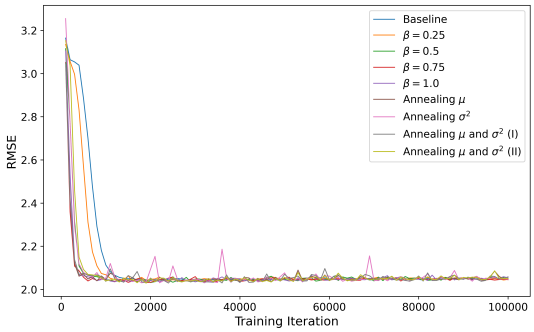
The repeat experiments have shown consistent behaviour with the single initialisation exper-

iments in the previous section. For the homoscedastic example, it is apparent that the $\beta-$NLL method is still a better candidate to address the problem of suboptimal convergence. However, the annealing μ and annealing μ and σ^2 (I) algorithms are promising candidates for further testing. In the next section, all the methods are tested on a simple sinusoidal heteroscedastic noise example to see if the resultant models still produce outputs comparable to the baseline model.

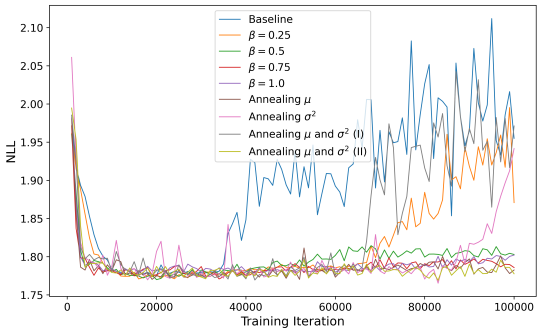
5.4 Simple sinusoidal heteroscedastic noise example

In this section all algorithms are tested using a simple sinusoidal heteroscedastic noise example. Seitzer et al. [39] have already shown using this example that the baseline model is capable of converging and with reasonable speed. Therefore, the purpose of these experiments is not to demonstrate a solution to suboptimal convergence, but rather the ability of the methods to fit the distributional parameters comparable to the baseline model.

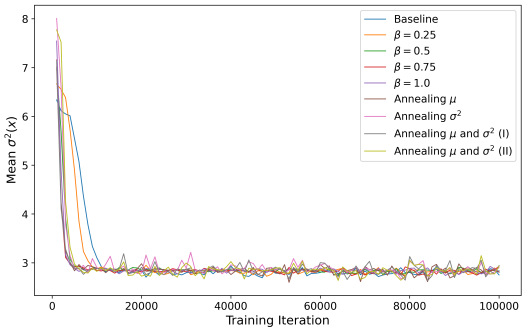
The training dataset consisted of 1000 linearly spaced training data points in the domain $x \in [0,10]$ with corresponding outputs $y = \sin(x) + x\varepsilon + \varepsilon$ where $\varepsilon \sim \mathcal{N}(0,0.3)$. The MVE network had 1 hidden layer with 50 units in each subnetwork of the model. The models were evaluated at 200 linearly spaced test points throughout the domain. A baseline model, $\beta \in \{0.25, 0.5, 0.75, 1\}$ and all annealing model variants were trained for 100000 epochs with a learning rate = 1E-03. Figures 5.12 shows the performance of the experiments over 100000 epochs.



(a) RMSE



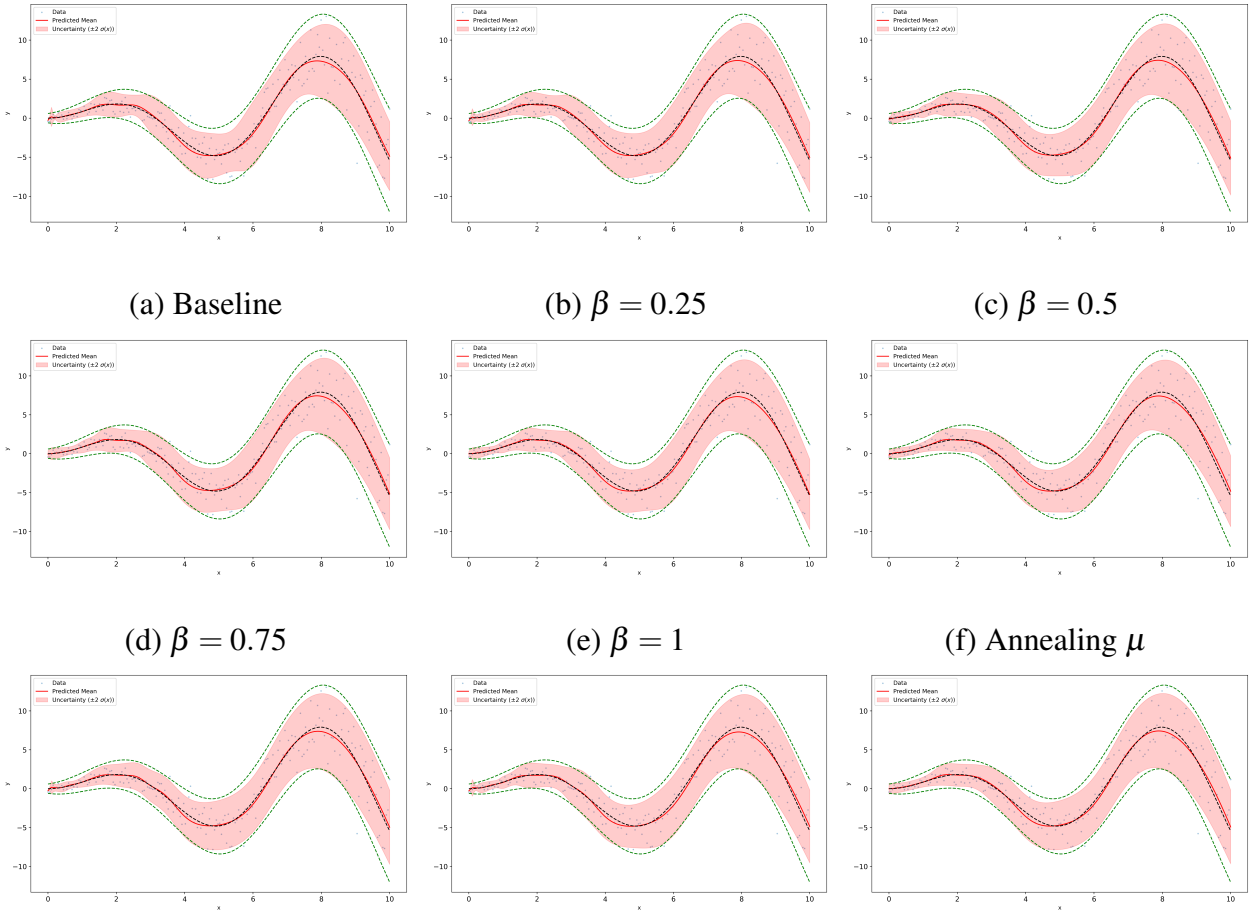
(b) NLL



(c) Mean $\sigma^2(x)$

Figure 5.12: RMSE, NLL and Mean $\sigma^2(x)$ across 100000 training iterations (epochs) for the baseline model, $\beta = 0.25$, $\beta = 0.5$, $\beta = 0.75$, $\beta = 1$ annealing μ model for the heteroscedastic noise example.

Figures 5.12 (a) and (c) show that all models converge down to the same RMSE and Mean $\sigma^2(x)$ as training progresses. Although the β models converge faster than the baseline, there does not seem to be a significant difference in their convergence time. Figure 5.12 (b) shows that all models quickly converge, however, the baseline, $\beta \in \{0.25, 0.5\}$, annealing σ^2 and annealing μ and σ^2 models all begin to diverge. The annealing μ and annealing μ and σ^2 (II) models seem to be more robust to over-fitting. over-fitting may have occurred due to regions of the domain where there are both mis-calibrated predictive mean values and incorrect variances. Figure 5.13 shows the plots of all models after 100000 epochs of training.



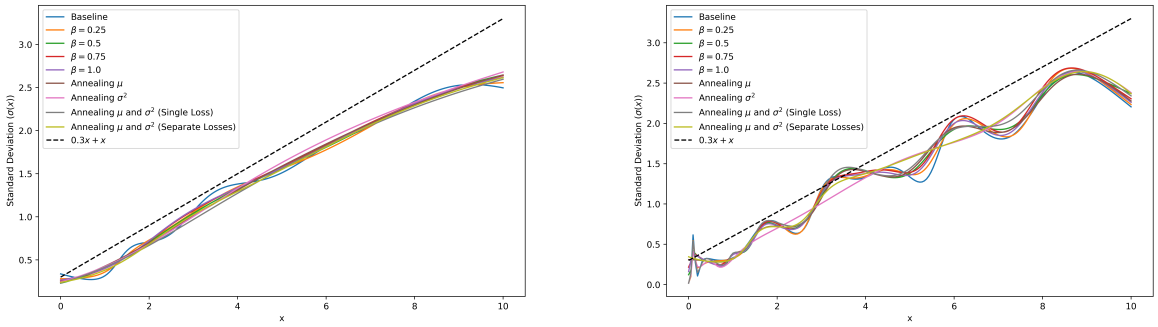
(a) Baseline (b) $\beta = 0.25$ (c) $\beta = 0.5$

(d) $\beta = 0.75$ (e) $\beta = 1$ (f) Annealing μ

(f) Annealing σ^2 (f) Annealing μ and σ^2 (I) (f) Annealing μ and σ^2 (II)

Figure 5.13: Plots of the heteroscedastic noise example for the baseline, $\beta \in \{0.25, 0.5, 0.75, 1\}$ and all annealing models trained for 100000 epochs.

Figure 5.13 shows that all models have over-fit the mean estimate and have underestimated the variance. The baseline and $\beta = 0.25$ models show signs of more over-fitting compared to the other models particularly in the domain $x \in [0, 2]$. This is in agreement with the divergence in the NLL shown in Figure 5.13 (b). The underestimation of the variance is a known problem when capturing the variance using MVE networks and the Gaussian NLL loss function [40]. Whilst the over-fitting and underestimation of the variance may seem small in this simple example, when scaled to large problems of higher dimensionalities, this can be problematic. Figure 5.14 shows plots of standard deviation across the domain for all models at 20000 (optimal) and at 100000 epochs.



(a) Optimal

(b) 100000 epochs

Figure 5.14: $\sigma(x)$ across the domain of x for every model trained on the heteroscedastic noise example after a) an optimal number of epochs (20000 epochs) and b) after 100000 epochs.

Figure 5.14 (a) shows that the standard deviation follows a linear trend, but all models have underestimated the true value. Figure 5.14 (b) shows an oscillating upward trend which is probably a symptom of over-fitting and also the sinusoidal nature of the data. These results show that none of the models are robust to over-fitting over a long period of training. However, all models perform at least as well as the baseline model. To some extent, over-fitting the predictive mean and underestimating the predictive variance can be addressed with WD, which will be shown in Chapter 5.4.2. Before that, in the next section repeat experiments are run on 30000 epochs, before the over-fitting occurs in all the models, to investigate consistency in the results before the over-fitting problem occurs.

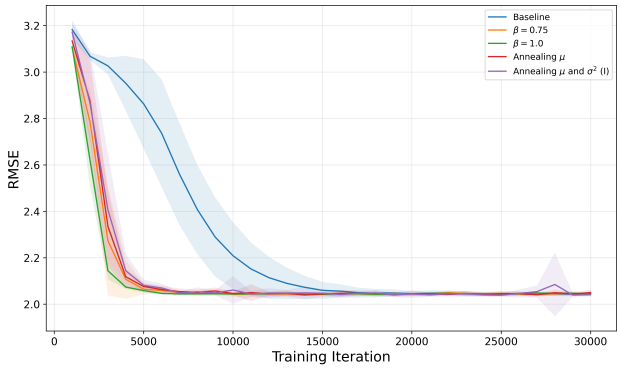
5.4.1 Repeat experiments on the heteroscedastic noise example

For the heteroscedastic noise example, all of the models are trained for 10 different initialisations of the model for 30000 epochs. The training and test data and learning parameters are the same as described in Chapter 5.4. Table 5.2 shows the mean and standard deviation of the performance statistics (RMSE and NLL) for all repeat experiments.

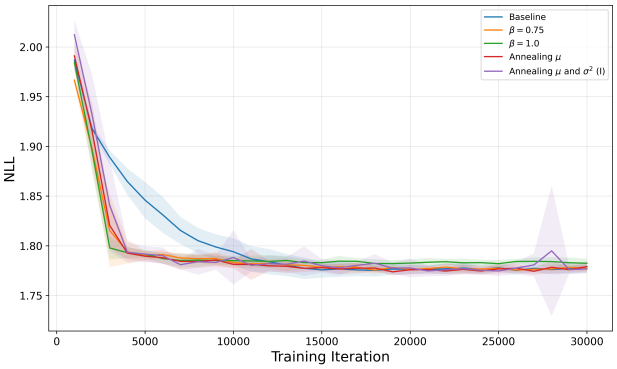
Table 5.2: Mean and standard deviation of NLL and RMSE of the 10 repeated heteroscedastic noise experiments for all model variants after 30000 epochs.

Model	**NLL**	**RMSE**
Baseline	1.7773 ± 0.004	2.0430 ± 0.006
$\beta = 0.25$	1.7800 ± 0.004	2.0510 ± 0.006
$\beta = 0.5$	1.7798 ± 0.004	2.0496 ± 0.007
$\beta = 0.75$	1.7774 ± 0.002	2.0442 ± 0.004
$\beta = 1$	1.7825 ± 0.005	2.0452 ± 0.005
Annealing μ	1.7792 ± 0.005	2.0502 ± 0.011
Annealing σ^2	1.7814 ± 0.009	2.0513 ± 0.017
Annealing μ and σ^2 (I)	1.7768 ± 0.004	2.0409 ± 0.008
Annealing μ and σ^2 (II)	1.7792 ± 0.003	2.0495 ± 0.006

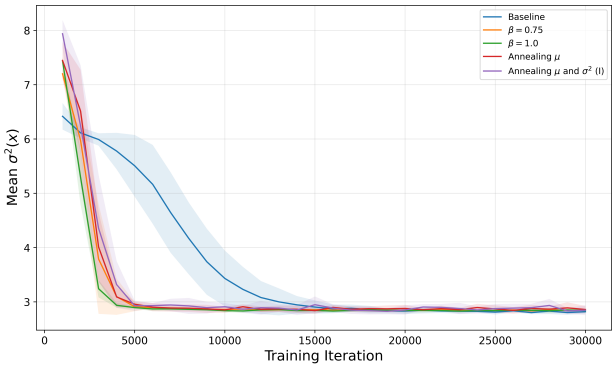
Table 5.2 shows that all models converge, with little variation between the final values for all models. These results simply demonstrate consistency in all of the methods.



(a) RMSE



(b) NLL



(c) Mean $\sigma^2(x)$

Figure 5.15: Mean and standard deviation of the RMSE, NLL and Mean $\sigma^2(x)$ of the 10 repeated heteroscedastic noise experiments across 30000 training iterations (epochs) for the baseline, $\beta \in \{0.75, 1\}$, annealing μ and annealing μ and $\sigma^2(x)$ (I).

Figure 5.15 confirms that the annealing μ and annealing μ and σ^2 (I) models converge at a similar rate to the $\beta \in \{0.75, 1\}$ models for all metrics.

5.4.2 Effect of applying weight decay to mean variance estimation networks

One approach to mitigating over-fitting of the predictive mean and underestimation of the predictive variance using MVE networks is via the application of WD or L2 regularisation as recommended by Sluijterman et al. [40]. Their approach is shown to address the suboptimal convergence issue in the predictive mean and perform better than the $\beta-$NLL method when $\beta = 0.5$ on simple sinusoidal examples and UCI regression datasets. However, their method is highly sensitive to the choice

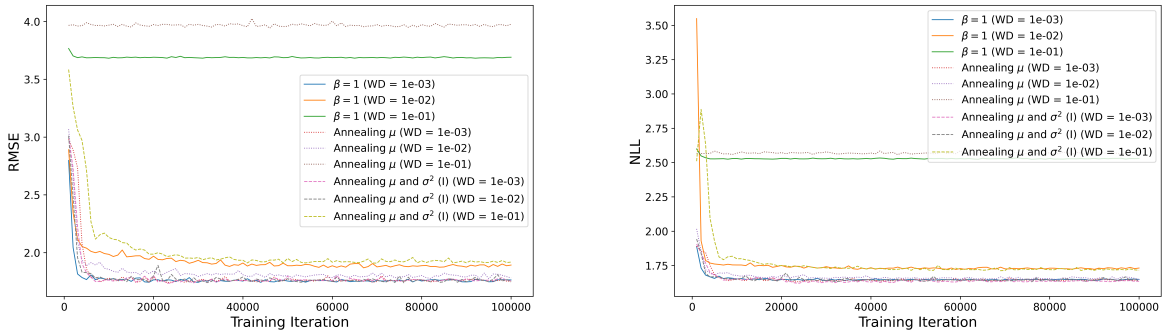
of WD parameters where different values are required on each of the subnetworks and, therefore, requires extensive hyper-parameter tuning. Furthermore, their method is only compared to the $\beta = 0.5$ models and, without justification, the authors have not conducted a comparative analysis with models trained with higher levels of β. Hence, in this chapter, the annealing methods have not been compared with the methods outlined by Sluijterman et al. [40], because there is a belief that $\beta-$NLL would be better at solving the convergence issue for the predictive mean for higher values of β. Additionally, there were shown to be convergence issues when applying their method using higher values of WD. Despite these disadvantages, their approach of applying WD is promising for addressing the underestimation of the predictive variance, but also preventing over-fitting of the predictive means.

In this section, the algorithms are evaluated on the heteroscedastic sinusoidal noise example described in the previous section given varying levels of WD on the MVE networks. This is to evaluate whether convergence issues under WD can be addressed, and therefore see if the application of WD is a promising avenue to mitigate the underestimation issue. Using the same training data points, model architecture and learning parameters as described in Chapter 5.4, a baseline model, $\beta \in \{0.25, 0.5, 0.75, 1\}$, annealing μ and annealing μ and σ^2 (I) models are trained for different coefficient values of weight decay, WD $\in \{$1E-05, 1E-04, 1E-03, 1E-02, 1E-01$\}$. The NLL and RMSE results at 100000 epochs are shown in Table 5.3.

Table 5.3: NLL and RMSE of the baseline, $\beta \in \{0.25, 0.5, 0.75, 1\}$, annealing μ and annealing μ and σ^2 (I) models trained with WD $\in \{$1E-05, 1E-04, 1E-03, 1E-02, 1E-01$\}$ trained on the heteroscedastic noise example at 100000 epochs.

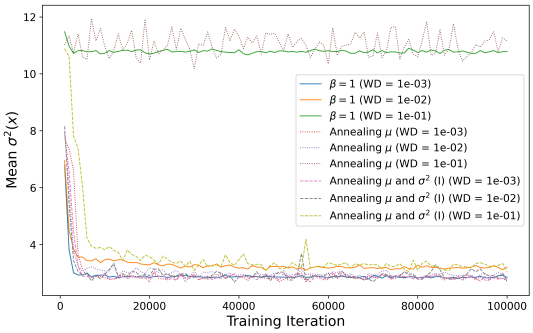
	Weight Decay (WD)									
	1E-05		**1E-04**		**1E-03**		**1E-02**		**1E-01**	
Model	**NLL**	**RMSE**	**NLL**	**RMSE**	**NLL**	**RMSE**	**NLL**	**RMSE**	**NLL**	**RMSE**
Baseline	1.7697	2.3174	1.7644	1.6410	1.8138	1.6620	3.5477	2.2748	4.1116	2.7027
$\beta = 0.25$	1.7719	1.6497	1.7653	1.6362	1.7890	1.6581	2.9651	2.0486	4.0885	2.6379
$\beta = 0.5$	**1.7660**	1.6402	1.7580	1.6349	1.7843	1.6569	2.4897	1.8904	4.0259	2.5924
$\beta = 0.75$	1.7674	1.6374	1.7650	1.6393	1.7633	1.6471	1.9896	1.7331	3.9187	2.5732
$\beta = 1$	1.7698	**1.6373**	1.7679	1.6441	1.7640	1.6489	1.8931	1.7318	3.6913	2.5302
Annealing μ	1.7664	1.6396	**1.7525**	**1.6334**	**1.7514**	1.6398	1.7819	1.6488	3.9771	2.5748
Annealing μ and σ^2 (I)	1.7668	1.6385	1.7638	1.6376	1.7581	**1.6351**	**1.7466**	**1.6348**	**1.9189**	**1.7153**

Table 5.3 shows that in the presence of WD, the annealing models tend to perform better than all the other models. The $\beta -$NLL models become more robust to WD as β increases. However, the annealing μ and σ^2 (I) model shows significantly better performance when WD = 1E-01, while all other models perform badly and are unable to fit the data. Figure 5.16 shows the performance of the top performing models ($\beta = 1$, annealing μ and annealing μ and σ^2 (I) models) for WD $\in$ $\{$1E-03, 1E-02, 1E-01$\}$ across 100000 epochs.



(a) RMSE

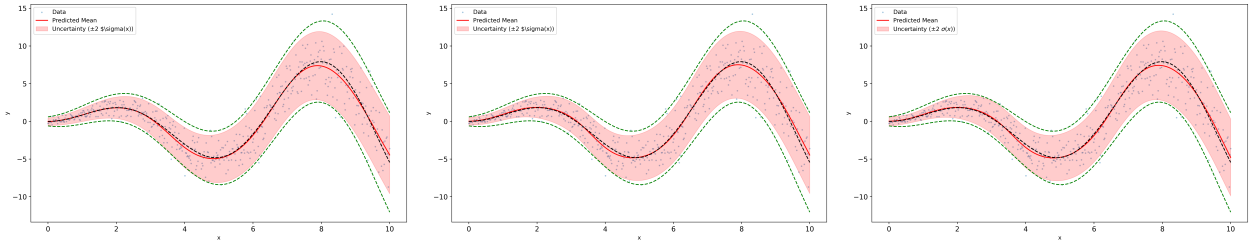
(b) NLL



(c) Mean $\sigma^2(x)$

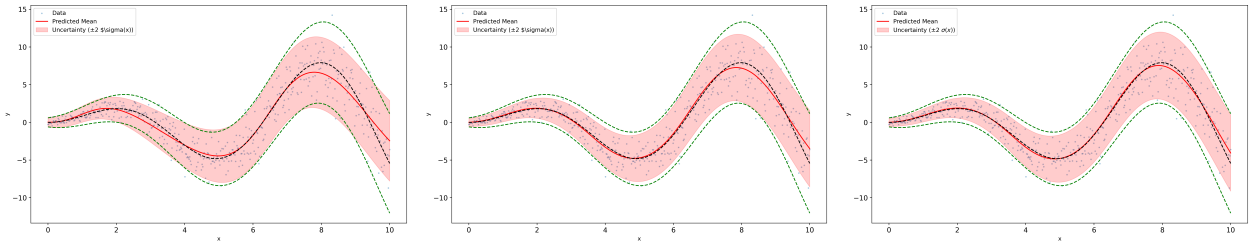
Figure 5.16: RMSE, NLL and Mean $\sigma^2(x)$ across 100000 training iterations (epochs) for the $\beta = 1$ annealing μ and annealing μ and σ^2 (I) models with WD $\in \{$1E-03, 1E-02, 1E-01$\}$ for the heteroscedastic example.

Figure 5.16 (a) clearly shows that with WD $\in \{$1E-02, 1E-01$\}$, the annealing μ and $\beta = 1$ models completely fail to learn the data, while the annealing μ and σ^2 (I) model is capable of reducing the RMSE. Figure 5.16 (b) shows that in the presence of WD the NLL is more stable in the annealing models and there is no sign of divergence occurring in any of the models. Figure 5.16 (c) shows that after 20000 epochs the value of Mean $\sigma^2(x)$ in all models remains stable. Figures 5.17, 5.18 and 5.19 show the final plots for the models that perform the best under WD $\in$ $\{$1E-03, 1E-02, 1E-01$\}$.



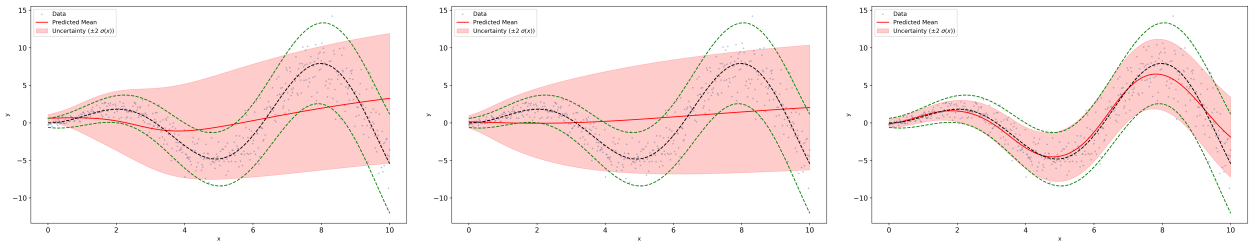
(a) $\beta = 1$ (b) Annealing μ (c) Annealing μ and σ^2 (I)

Figure 5.17: Plots of the $\beta = 1$, annealing μ and annealing μ and σ^2 (I) models with WD = 1E-03 trained on the heteroscedastic noise example at 100000 epochs.



(a) $\beta = 1$ (b) Annealing μ (c) Annealing μ and σ^2 (I)

Figure 5.18: Plots of the $\beta = 1$, annealing μ and annealing μ and σ^2 (I) models with WD = 1E-02 trained on the heteroscedastic noise example at 100000 epochs.



(a) $\beta = 1$ (b) Annealing μ (c) Annealing μ and σ^2 (I)

Figure 5.19: Plots of the $\beta = 1$, annealing μ and annealing μ and σ^2 (I) models with WD = 1E-01 trained on the heteroscedastic noise example at 100000 epochs.

Figure 5.17 shows that even with weak WD, over-fitting can be alleviated, particularly at $x \in [0,2]$. Figure 5.18 shows that with slightly increased WD, the $\beta = 1$ model is beginning to reach

its limit of robustness as can be seen in $x \in [8, 10]$ where the model struggles to fit the data. Finally, Figure 5.19 clearly shows that the annealing μ and σ^2 (I) model has the greatest robustness to WD, while the $\beta = 1$ and annealing μ models completely do not fit the data. The standard deviations of the best performing models in the domain after 100000 epochs of training are shown in Figure 5.20.

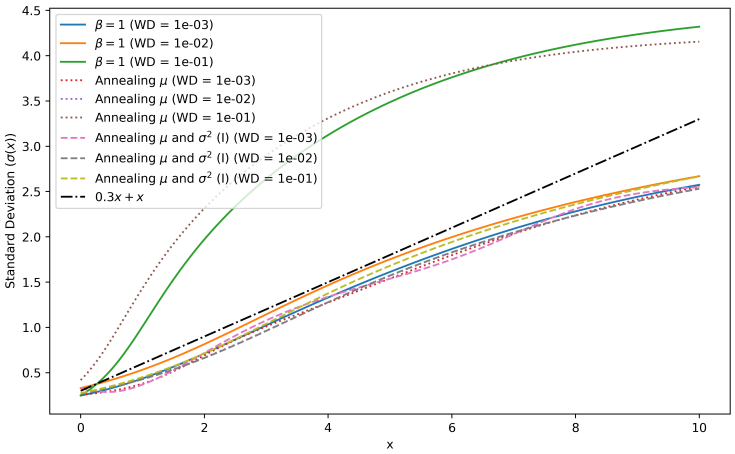


Figure 5.20: $\sigma(x)$ across the domain of x for the $\beta = 1$, annealing μ and annealing μ and σ^2 (I) models trained on the heteroscedastic noise example with WD $\in \{$1E-03, 1E-02, 1E-01$\}$ after 100000 epochs.

The general trend of Figure 5.20 shows that the application of WD can improve the variance fit across the domain. However, it can still be seen that all models which sufficiently converge still underestimate the variance.

In all models, the application of WD seems to have addressed the fluctuations in the NLL that were evident in the previous section. Furthermore, the over-fitting of the predictive mean shown in the domain region $x \in [0, 1]$ seems to be mitigated in addition to preventing, to some extent, the under-fitting of the predictive variances. Overall, the annealing μ and σ^2 (I) algorithm seems to have great robustness to WD in comparison to all the other methods, which is promising, as it can further enable WD as a suitable approach to addressing the underestimation problem. In the next section, repeat experiments are conducted to see if there is consistency in these results.

Repeat experiments on the heteroscedastic noise example with weight decay

For the heteroscedastic noise example with WD, the baseline, $\beta \in \{0.25, 0.5, 0.75, 1\}$, annealing μ and annealing μ and σ^2 (I) models are trained for 10 different initialisations of the model for 30000 epochs. The annealing μ and annealing μ and σ^2 were the top performing annealing models in the previous section and therefore further evaluation was considered. The training and test data, model architecture, and learning rate are the same as described in Chapter 5.4.2. Tables 5.4 and 5.5 show the mean and standard deviation of the performance statistics (RMSE and NLL) of all repeated experiments.

Table 5.4: Mean and standard deviation of NLL and RMSE of the 10 repeated trainings of the baseline, $\beta \in \{0.25, 0.5, 0.75, 1\}$ and annealing μ models trained with WD $\in \{$1E-05, 1E-04, 1E-03, 1E-02, 1E-01$\}$ on the heteroscedastic noise example after 30000 epochs.(Part 1)

	Weight Decay (WD)					
	1E-05		**1E-04**		**1E-03**	
Model	**NLL**	**RMSE**	**NLL**	**RMSE**	**NLL**	**RMSE**
Baseline	1.7782 ± 0.003	2.0438 ± 0.006	1.7816 ± 0.003	2.0482 ± 0.006	1.7867 ± 0.004	2.0815 ± 0.007
$\beta = 0.25$	1.7794 ± 0.003	2.0502 ± 0.006	1.7813 ± 0.003	2.0473 ± 0.007	1.7861 ± 0.003	2.0595 ± 0.007
$\beta = 0.5$	1.7809 ± 0.004	2.0493 ± 0.007	1.7836 ± 0.004	2.0476 ± 0.007	1.7829 ± 0.003	2.0516 ± 0.007
$\beta = 0.75$	1.7773 ± 0.002	2.0434 ± 0.003	1.7834 ± 0.004	2.0479 ± 0.008	1.7812 ± 0.002	2.0417 ± 0.005
$\beta = 1$	1.7824 ± 0.003	2.0441 ± 0.004	1.7843 ± 0.005	2.0425 ± 0.005	1.7797 ± 0.002	2.0392 ± 0.005
Annealing μ	1.7820 ± 0.004	2.0530 ± 0.010	1.7799 ± 0.004	2.0493 ± 0.009	1.7842 ± 0.007	2.0474 ± 0.014
Annealing μ and σ^2 (I)	1.7769 ± 0.005	2.0404 ± 0.007	1.7834 ± 0.006	2.0528 ± 0.014	1.7758 ± 0.007	2.0393 ± 0.006

Table 5.5: Mean and standard deviation of NLL and RMSE of the 10 repeated trainings of the baseline, $\beta \in \{0.25, 0.5, 0.75, 1\}$ and annealing μ models trained with WD $\in$ {1E-05, 1E-04, 1E-03, 1E-02, 1E-01} on the heteroscedastic noise example after 30000 epochs.(Part 2)

	Weight Decay (WD)			
	1E-02		**1E-01**	
Model	**NLL**	**RMSE**	**NLL**	**RMSE**
Baseline	2.2753 ± 0.031	3.6354 ± 0.090	2.7278 ± 0.001	4.2078 ± 0.002
$\beta = 0.25$	2.0858 ± 0.001	3.1294 ± 0.003	2.6635 ± 0.001	4.1906 ± 0.002
$\beta = 0.5$	1.9431 ± 0.009	2.6606 ± 0.032	2.6610 ± 0.001	4.1236 ± 0.003
$\beta = 0.75$	1.8306 ± 0.007	2.2289 ± 0.025	2.5865 ± 0.001	4.0230 ± 0.002
$\beta = 1$	1.8142 ± 0.007	2.1304 ± 0.005	2.5258 ± 0.004	3.8120 ± 0.001
Annealing μ	1.7830 ± 0.002	2.0069 ± 0.005	2.5853 ± 0.004	4.0746 ± 0.007
Annealing μ and σ^2 (I)	1.7816 ± 0.007	2.0513 ± 0.017	2.0442 ± 0.015	1.7783 ± 0.008

Tables 5.4 and 5.5 show that all models perform similarly when WD $\in$ {1E-05, 1E-04, 1E-03}. However, when WD $\in$ {1E-02, 1E-01} the annealing models clearly outperform the other models. The standard deviations of all results are similar, suggesting that WD has the capacity to smooth the learning process. Furthermore, the standard deviations of the results are much lower compared to the results shown in Table 5.2 when WD was not applied.

5.5 Computational cost of algorithms

A brief comment is made on the computational cost of each algorithm. Assuming a model with architecture described by Figure 5.1 with each subnetwork, $\sigma^2(x;\theta_{\sigma^2})$ and $\mu(x;\theta_\mu)$, having the same depth, hidden units, activations and input size. The computational cost of a forward pass for each subnetwork is C_f and the cost of a backward pass for each subnetwork is C_b. Table 5.6 shows an approximation of the computational cost for each algorithm.

Table 5.6: Approximate computational costs for a single training iteration of each algorithm when training an MVE network described by Figure 5.1, where C_f is the computational cost of forward pass of a subnetwork and C_b is the computational cost of a backwards pass of a subnetwork.

Training Algorithm	**Computational Cost**
Baseline, $\beta-$NLL	$2C_f+2C_b$
Annealing μ, Annealing σ^2	$2C_f+4C_b$
Annealing μ and σ^2 (I)	$2C_f+6C_b$
Annealing μ and σ^2 (II)	$4C_f+8C_b$

Table 5.6 shows that there is a significant increase in costs between the baseline and $\beta-$NLL method compared to the annealing algorithms. Between the annealing μ and annealing μ and σ^2 (I) algorithms, which demonstrated better performance compared to the other annealing methods, there is a difference of $2C_b$ when training, which should be an additional consideration when deciding to use the algorithm. However, the tuning of β in the $\beta-$NLL method also needs to be considered, which would require validation approaches that could be computationally expensive. The cost of C_f compared to C_b varies depending on the model architecture, however, assuming that both grow linearly as batch size increases, it can be said that $C_b \approx 2C_f$ [103].

5.6 Conclusions and future work

The Gaussian NLL loss is a common objective function used to provide aleatoric uncertainty estimates when training NNs. Accurate uncertainty quantification is crucial to providing trust in predictive outputs of a model and can better inform engineers in decision making. As shown in the simple sinusoidal homoscedastic noise example, a standard parametrisation of an MVE network, trained using the Gaussian NLL was incapable of converging to optimal solutions. The failure on this trivial example diminishes the trust in this objective function when training NNs on higher dimensional and large-data regimes. The $\beta-$NLL [39] method is the current state of the art for mitigating the suboptimal convergence problem, however, it requires a hyper-parameter choice of β. which in turn requires some knowledge of the data or multiple training runs to tune. In this chapter novel learning rate annealing algorithms were developed, which had the capacity to solve the suboptimal convergence problem to a standard comparable to the $\beta-$NLL method on homoscedas-

tic and heteroscedastic noise examples. The developed algorithms rebalanced the losses in difficult to train regions of the domain via a re-weighting informed by the magnitudes of the gradients. The main advantages of the algorithms are the absence of any hyper-parameters that need to be tuned and the robustness of the algorithms when training models with WD. WD is essential in mitigating over-fitting of the mean estimate in addition to preventing underestimation of the variance in NNs. Of all annealing methods, the annealing algorithm μ and σ^2 (I) had the highest robustness to WD. Therefore, as a means of applying WD to mitigate over-fitting and underestimation issues using MVE networks, annealing μ and σ^2 (I) alongside WD is the recommended approach.

Future work includes varying the α parameter in annealing algorithms to see if the rate of convergence improves. Furthermore, evaluating the algorithm on a larger range of domains and tasks would demonstrate the applicability and versatility of the algorithms beyond the simple regression examples shown in this chapter. Additionally, in order to reduce computational complexity, further experiments can be performed to test Wang et al. [67] suggestion of implementing the algorithm periodically rather than after every iteration of gradient descent in order to cut down on computational costs. Moreover, the annealing algorithms reframe the convergence issue from a single objective problem to a multi-objective one. This would inspire the adaptation of other existing multi-objective loss balancing methods such as GradNorm [104], SoftAdapt [105], or Relative Loss Balancing with Random Lookback [102]. SoftAdapt and ReLoBRaLo in particular avoid the use of gradient statistics in their method, which would skirt the computational issues observed in the developed algorithms.

Chapter 6

Multi-fidelity modelling: a maximum a posteriori and kernel-based approach

Accurate quantification of mismatch uncertainty in Multi-Fidelity (MF) models is crucial for providing trust in their outputs. As described in Chapter 2, the mismatch uncertainty quantifies the confidence in a model's predictive accuracy relative to the true system, based on the contributions of different fidelity datasets. The level of uncertainty depends on the degree of model mismatch present in each dataset. Existing approaches for quantifying mismatch uncertainty are typically incorporated within the estimate of epistemic uncertainty. These methods can be inhibitive due to their large computational cost at training and test time, extensive hyper-parameter tuning, unsuitability for large data regimes, or difficulty in embedding prior assumptions about the model mismatch. In order to address these problems, a novel MF modelling approach was developed in Chapter 4 that mitigated these issues. This was applied to a Neural Network (NN), enabling its applicability on large data regimes and reduced computational cost.

In this chapter, an alternative Maximum a Posteriori (MAP) based MF modelling approach is developed with a revised Bayesian problem formulation that addresses the limitations of the approach developed in Chapter 4. Similarly to Chapter 4, the approach in this chapter still provides a principled method of trading off between datasets of different fidelity when performing MF modelling. Instead of treating the sample size n_{Y_H} in the approximate posterior as a fixed hyper-parameter, through which assumptions about the level of trust between datasets of differing fidelity were applied globally and were constant, the approach taken in this chapter treats this quantity as

a spatially-varying sample size which allows the level of trust to be based on the density and sparsity of High-Fidelity (Hi-Fi) data. Rather than being a fixed quantity, the sample size is defined as a function of a Kernel Density Estimate (KDE) fit on the available Hi-Fi data. This is advantageous compared with the approach of Chapter 4 as it allows contributions from the Hi-Fi data to vary spatially based on the density of the data. Furthermore, this function enables for modelling assumptions to be placed in transitory regions between the datasets of different fidelities through tuneable hyper-parameters. This Kernel-based approach mimics some of the desirable qualities found in GP regression, as will be described in the motivation and conceptual overview section.

The chapter begins by providing a motivation and conceptual overview of the framework. The revised Bayesian problem formulation is then introduced in addition to defining the careful choice of hyper-parameters. Then, a generalised modelling process for implementing the framework is outlined. On a 1-D example, the influence of hyper-parameter choices on the predictive performance and uncertainty is illustrated. Then, the modelling process is demonstrated on a 2-D example. Finally, the approach is compared to baseline MF modelling methods based on GPs. The key contributions of this chapter can be summarised below:

- A novel adaptation of the Bayesian problem formulation developed in Chapter 4 for datasets with different fidelities, allowing engineers to place prior beliefs on the mismatch between data sources when undertaking MF modelling.
- A lower complexity MAP fitting approach, avoiding the need for the Variational Inference (VI)-based training approach required in Chapter 4.
- A novel KDE-based interpretation of sample size in the approximate posterior, which reflects the density of data points and enables modelling assumptions to be placed in transitory and sparse regions of the domain, improving upon the limitations of the MAP-based approach in Chapter 4.
- A high-level process for implementing the modelling framework illustrating the method for general hyper-parameter tuning and the applicability of the method on 1-D and 2-D examples.
- A comparison of the proposed approach with baseline MF modelling methods in terms of predictive performance and computational complexity.

6.1 Motivation and conceptual overview

As outlined in Chapter 2, GP based MF modelling methods have the capacity to capture mismatch uncertainty within their estimate of epistemic uncertainty. However, it was also shown that these methods are unfeasible for big data regimes, especially at test time in Digital Twin (DT) applications. Although state of the art methods such as sparse GPs [49] or matrix decomposition techniques can reduce the complexity, the computational burden is still significant.

Nevertheless, these methods provide inspiration for the development of alternative approaches to MF modelling which can handle model mismatch and are computationally cheaper at inference. In particular, GP regression exhibits two desirable properties that are incorporated into the framework developed in this chapter. These properties are outlined below.

- **Property 1:** The predictive outputs revert to a prior when test inputs lie in regions of the domain with little or no training data.
- **Property 2:** The rate at which predictions revert to the prior depends on the proximity of the test inputs to the training data and can be controlled.

An intuition for how these properties will be implemented into the Bayesian framework will be explained below starting with the second property.

Property 2 implementation

A brief reflection is made about the interpretation of the sample size, n_{Y_H}, in the Bayesian framework described in Chapter 4. The sample size represented the level of trust placed in the Hi-Fi data over the Lo-Fi data (and possibly a correction). In practical terms, the magnitude of n_{Y_H} determined the contribution a Hi-Fi data point had relative to a Lo-Fi data point when making a prediction. The level of trust was set a priori and was assigned uniformly to all Hi-Fi data points. The drawback of this was the inability to account for the density of Hi-Fi data points at different parts of the domain when assigning trust values. Also, when there is high data density, there may be a reasonable amount of trust that a surrogate model on the Hi-Fi data can accurately interpolate or extrapolate. The Bayesian formulation in this chapter reinterprets the notion of a uniformly applied sample size

to one that is spatially varying and density dependent; in addition, one where modelling assumptions about the transitory regions between data and sparsity can be placed. This was achieved by setting n_{Y_H} as a function of $f(x)$ where x is a value in the input space, $n_{Y_H} \propto f(x)$, and $f(x)$ is a function of a KDE. $f(x)$ is parametrised further, but this is omitted from the notation in this section. This function allows for a sample size to be calculated for input points that are not directly at the Hi-Fi data points. It provides a sample size at any point based on the density and distance away from other Hi-Fi data points. This notion reflects how well a surrogate model that is trained on the Hi-Fi data might perform in accurately predicting in those transitory regions. Therefore, in this chapter, the trust between datasets means the trust between two surrogate models that have implicitly carried all of this information within them.

KDEs are a non-parametric method to estimate probability density functions, therefore they operate on the input space of a problem. They are kernel smoothers, meaning they use a weighted average of neighbouring observed data. The weightings are strongly influenced by the choice of kernel function and the tunable bandwidth hyper-parameter. For this Bayesian framework, $f(x)$, and therefore n_{Y_H}, provides a trust measure on the Hi-Fi data. This is informed by the location and density of the Hi-Fi Data points in the inputs domain, in addition to tunable hyper-parameters where additional modelling assumptions can be placed, including the transitory regions of the domain. $f(x)$ is a mapping of the input domain to $[0,1]$.

When $f(x) = 1$ total trust is placed on the Hi-Fi data when a prediction is made. Conversely, when $f(x) = 0$, no trust is placed on the Hi-Fi data and predictions default to the Lo-Fi data (or prior). The choice of $f(x)$ as a function of a KDE means that increased trust can be placed in regions of dense sampling and decreased where there is sparsity. Figure 6.1 below gives an intuition via a cartoon example of how $f(x)$ may affect modelling assumptions in this framework.

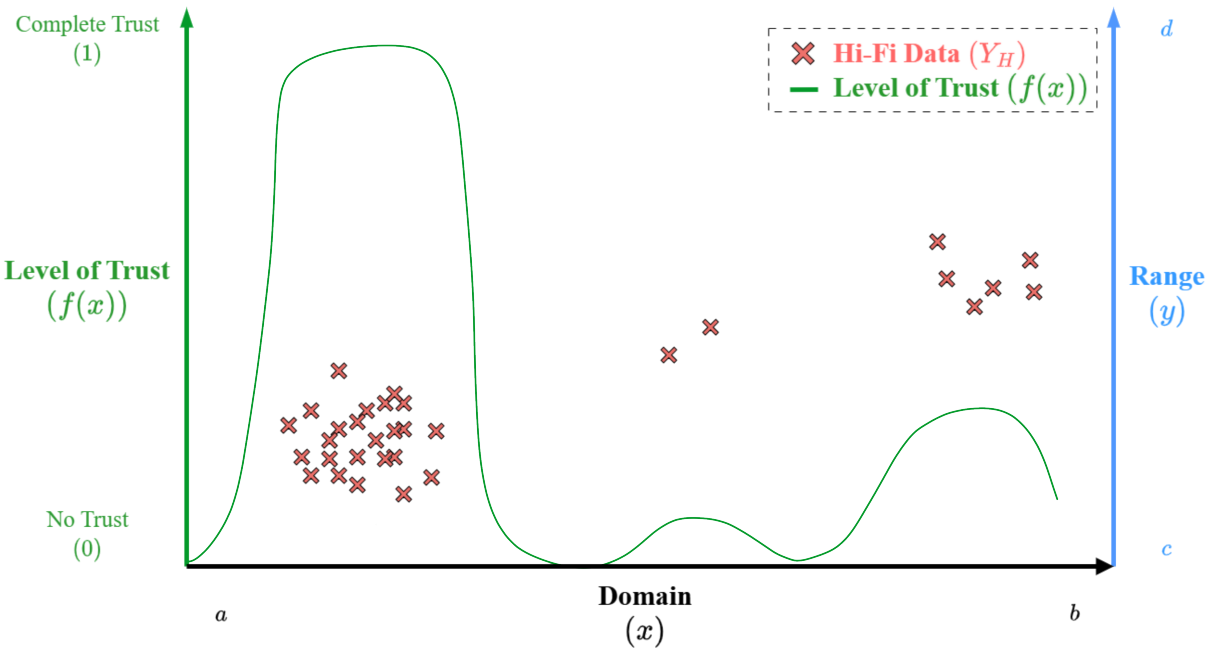


Figure 6.1: Cartoon plot of n_{Y_H} on a made-up 1-D example on an arbitrary dataset Y_H with pairs of data points in (x_H, y_H) in an arbitrary domain $[a,b]$ and arbitrary range $[c,d]$.

In Figure 6.1, $f(x)$ sets the amount of trust placed between the Hi-Fi data and the Lo-Fi data (or prior). In regions where there is high density, the level of trust is close to 1. In contrast, in areas where there is sparsity or little data, the trust decreases to 0 or is small. In the Bayesian framework, this will inform the sample size of any input points across the domain, i.e. how well a surrogate model trained on the Hi-Fi data does over a surrogate model trained on the Lo-Fi data. As will be shown later, hyper-parameter choices can control the modelling assumptions about trust in the transitional regions of the domain.

Property 1 implementation

Incorporating the first property requires a careful selection of hyper-parameters in the Bayesian problem formulation, which has yet to be outlined. However, the end result is described here conceptually.

Given a function $f(x)$ that encodes all the modelling assumptions about how trust in the Hi-Fi data varies across the domain, the objective is to construct predictive outputs that linearly interpolate between the Hi-Fi data and the Lo-Fi data (or prior) in a manner governed by $f(x)$. In regions

where $f(x) \approx 1$, the predictions should be informed mostly by Hi-Fi data. Conversely, where there is sparsity or $f(x) \approx 0$, predictions should revert to the Lo-Fi data (or prior). The intuition is described by Equation (6.1) below.

$$\big(\mu(x), \sigma^2(x)\big) \;\approx\; f(x)\,\big(\text{Hi-Fi posterior contribution}\big) + \big(1 - f(x)\big)\,\big(\text{Prior contribution}\big) \tag{6.1}$$

In Equation (6.1), where $f(x)$ is close to 1, the predictions are more informed by the Hi-Fi data, whereas when it is close to 0, predictions are informed by the Lo-Fi data (or prior). Having explained the motivation and intuition, the Bayesian formulation is outlined in the next section.

6.2 Bayesian formulation

The subspace $A = \{x \in X \subset \mathbb{R}^{d_x}\}$ represents the space of all possible inputs, x, for a physical system. $B \subset A$ is the subspace that spans the regions of the inputs obtained directly from measurements of the ground truth of the physical system. A Lo-Fi dataset is defined as $\mathcal{D}_{Y_L} = \{(x_{L_1}, y_{L_1}), \ldots (x_{L_n}, y_{L_n}) \,|\, x_{L_i} \subset A\}$ and a Hi-Fi dataset is defined as $\mathcal{D}_{Y_H} = \{(x_1, y_1), \ldots (x_{H_m}, y_{H_m}) \,|\, x_{H_i} \subset B\}$. For a given input x, Random Variables (RVs) on the Hi-Fi and Lo-Fi measurements are defined as

$$Y_H(x) = H(x) + \varepsilon_H(x) \tag{6.2}$$

$$Y_L(x) = L(x) + \varepsilon_L(x) \tag{6.3}$$

where $Y_H(x)$ is a RV representing output data from a Hi-Fi data source, $H(x)$ is a function representing the system output, $\varepsilon_H(x)$ is a RV representing additive external noise, $Y_L(x)$ is a RV representing data from a Lo-Fi data source, $L(x)$ is the Lo-Fi model, and $\varepsilon_L(x)$ is a RV representing additive external noise. It is assumed that the Hi-Fi system is related to the Lo-Fi model through an additive correction. This correction is learned from the data and yields a RV defined as

$$\begin{aligned} Y_D &\sim \mathcal{N}(\mu_{Y_H}(x) - \mu_{Y_L}(x), \sigma_{\varepsilon_H}(x) + \sigma_{\varepsilon_L}(x)) \\ &\equiv \mathcal{N}(\mu_{Y_D}(x), \sigma_{Y_D}(x)) \end{aligned} \tag{6.4}$$

6.2.1 Assumptions on random variables

Given a stochastic approach to modelling the data, it is assumed that $\varepsilon_H(x)$ and $\varepsilon_L(x)$ are normally distributed at each input x in the domain as described by

$$\varepsilon_H(x) \sim \mathcal{N}(\mu_{\varepsilon_H}(x), \sigma^2_{\varepsilon_H}(x)) \tag{6.5}$$

$$\varepsilon_L(x) \sim \mathcal{N}(\mu_{\varepsilon_L}(x), \sigma^2_{\varepsilon_L}(x)) \tag{6.6}$$

where $\mu_i(x)$ and $\sigma^2_i(x)$ are the mean and variance of the random variable i. Using addition rules on Gaussian RVs, the distributions for $Y_H(x)$ and $Y_L(x)$ are also Gaussian.

6.2.2 Deriving the approximate posterior equation

In this framework, it is assumed that the parameters of the distribution $Y_L(x)$ are known throughout the problem domain and that the parameters for $Y_D(x)$ are known in the Hi-Fi data domain. This follows from Equation (6.7).

An Inverse-Gamma prior $\sigma^2_{Y_H}(x) \sim IG(\alpha_0, \beta_0)$ and a Gaussian prior $\mu_{Y_H}(x) \sim \mathcal{N}(\mu_{Y_L}(x) + \mu_{Y_{Y_D}}(x), \sigma^2_{\text{mis}})$ are placed. An approximation to the posterior of $Y_H(x)$ given the Hi-Fi data is shown by

$$\begin{aligned} p(\mu_{Y_H}(x), \sigma^2_{Y_H}(x) | \mathcal{D}_{Y_H}, \mu_{Y_D}(x)) \propto & \\ & p\left(\mathcal{D}_{Y_H} | \mu_{Y_H}(x), \sigma^2_{Y_H}(x)\right) \cdot \\ & p\left(\mu_{Y_H}(x) | \mu_{Y_L}(x) + \mu_{Y_{Y_D}}(x), \sigma^2_{\text{mis}}(x)\right) \cdot \\ & p\left(\sigma^2_{Y_H}(x) | \alpha_0, \beta_0\right) \end{aligned} \tag{6.7}$$

Figure 6.2 describes the graphical model for the approximate posterior.

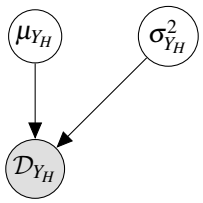


Figure 6.2: Graphical model of posterior distribution.

Figure 6.2 shows fewer distributional parameters to fit compared to the approximate posterior described by Figure 4.1 in Chapter 4. This mitigates the need for the expensive VI-based approach for the MAP estimation. This comes with the caveat that priors cannot be placed on the Lo-Fi dataset, which the framework in Chapter 4 had the potential to do. In the method described in Chapter 4, this would potentially enable modelling of more than two fidelities.

There also exists an approximation to $\mu_{Y_H}(x)$, denoted by $\hat{\mu}_{Y_H}(x)$, trained solely on the available Hi-Fi data. This approximation implicitly carries the information about the mean values in $\mathcal{D}_{Y_H}$. The predictive mean, $\mu_{Y_H}(x)$, learns directly from $\hat{\mu}_{Y_H}(x)$ rather than the Hi-Fi data points. As a result, aleatoric uncertainty is not captured in the predictive variance. Despite this limitation, this is a practical necessity for learning in regions of the input space where Hi-Fi data is absent. The optimisation procedure, described by Algorithm 7, involves batch sampling of inputs across the domain without any corresponding target values. Targets are instead obtained from evaluations of $\hat{\mu}_{Y_H}(x)$. Future work will be to incorporate aleatoric uncertainty into an implementation of this framework. For the current implementation, the Hi-Fi data is effectively replaced with $\hat{\mu}_{Y_H}(x)$.

Finally, the negative log of Equation (6.7) is given by Equation (6.8) below, which is used to fit the distributional parameters of the distribution $Y_H(x)$ using a MAP approach. In this equation, $\hat{\mu}_{Y_H}(x)$ is in place of the target Hi-Fi data.

$$\mathcal{L}_{nl} = \frac{n_{Y_H}}{2}\log(\sigma^2_{Y_H}(x)) + \frac{n_{Y_H}}{2}\frac{(\mu_{Y_H}(x) - \hat{\mu}_{Y_H}(x))^2}{\sigma^2_{Y_H}(x)} \tag{6.8a}$$

$$+\frac{1}{2}\frac{(\mu_{Y_H}(x) - (\mu_{Y_L}(x) + \mu_{Y_D}(x)))^2}{\sigma^2_{\text{mis}}} \tag{6.8b}$$

$$+(\alpha_0 + 1)\log(\sigma^2_{Y_H}(x)) + \frac{\beta_0}{\sigma^2_{Y_H}(x)}. \tag{6.8c}$$

6.2.3 Specification of hyper-parameter forms in the approximate posterior

A careful choice of the functional forms of the hyper-parameters in Equation (6.7) can enable principled trade-offs between learning the Hi-Fi data over the prior in regions of data-sparsity in addition to setting a prior mismatch uncertainty value. The set of hyper-parameters considered is $\Lambda = \{n_{Y_H}, \sigma^2_{\text{mis}}, \alpha_0, \beta_0\}$. In this section, the focus is on defining the general forms and modelling

roles of these quantities, rather than assigning specific numerical values, which would be problem specific. Although not technically a hyper-parameter within the posterior equation, n_{Y_H} is chosen to be a function of a KDE which has hyper-parameters that need tuning.

The general choice of Λ is based on an observation of the optimal analytical solutions of the distributional parameters $\mu_{Y_H}(x)$ and $\sigma^2_{Y_H}(x)$. This is derived from Equation (6.8) and is shown by Equations (6.9b) and (6.10c).

$$\frac{\partial \mathcal{L}_{nl}}{\partial \mu_{Y_H}(x)} = 0 \tag{6.9a}$$

$$\implies \mu_{Y_H}(x) = \frac{n_{Y_H}\sigma^2_{\text{mis}}\hat{\mu}_{Y_H}(x) + \sigma^2_{Y_H}(x)\left(\mu_{Y_L}(x) + \mu_{Y_D}(x)\right)}{n_{Y_H}\sigma^2_{\text{mis}} + \sigma^2_{Y_H}(x)} \tag{6.9b}$$

$$\frac{\partial \mathcal{L}_{nl}}{\partial \sigma^2_{Y_H}(x)} = 0 \tag{6.10a}$$

$$\implies \sigma^2_{Y_H}(x) = \frac{n_{Y_H}(\mu_{Y_H}(x) - \hat{\mu}_{Y_H}(x))^2 + 2\beta_0}{n_{Y_H} + 2(\alpha_0 + 1)} \tag{6.10b}$$

$$= \frac{\frac{n_{Y_H}}{\alpha_0+1}(\mu_{Y_H}(x) - \hat{\mu}_{Y_H}(x))^2 + 2\frac{\beta_0}{\alpha_0+1}}{\frac{n_{Y_H}}{\alpha_0+1} + 2} \tag{6.10c}$$

The functional forms of Λ are chosen in order to simplify the process of problem specific numerical hyper-parameter selection, while concentrating the majority of modelling assumptions within the sample size n_{Y_H}. To achieve this, the functional forms are specified in order to reduce the analytical solutions for $\mu_{Y_H}(x)$ and $\sigma^2_{Y_H}(x)$ to a weighted linear interpolation between learning from Hi-Fi data and learning from Lo-Fi data (or prior). The weights are determined by a function of a KDE, which controls the influence of each dataset based on prior assumptions about the datasets and their relationship with one another.

Functional form of n_{Y_H}

The functional form of n_{Y_H} is defined by:

$$n_{Y_H} = \frac{2(\alpha_0 + 1)f(x; \Sigma, \tau, \eta, \mathcal{D}_{Y_H})}{1 - f(x; \Sigma, \tau, \eta, \mathcal{D}_{Y_H})} \tag{6.11}$$

where the function $f(x;\Sigma,\tau,\eta,\mathcal{D}_{Y_H})$ is defined by:

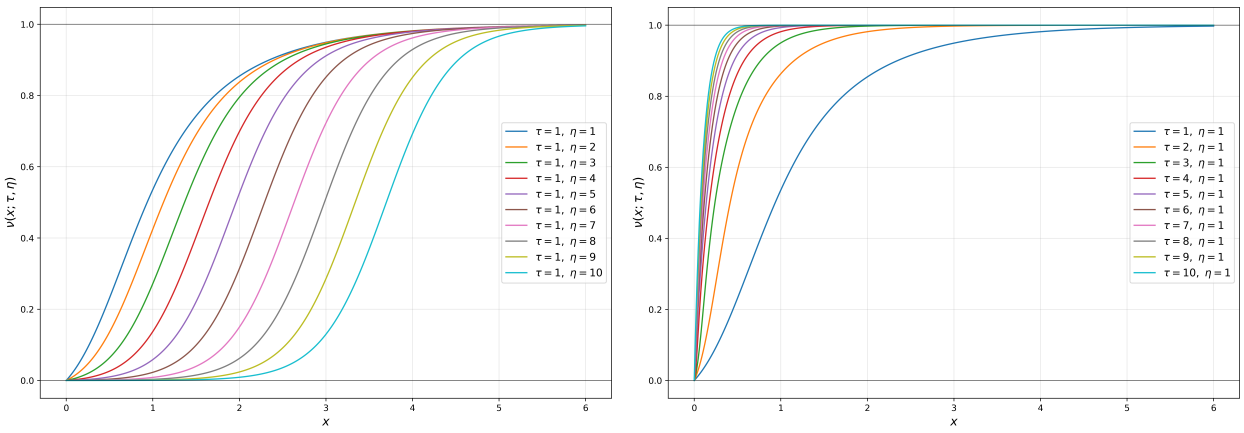
$$f(x;\Sigma,\tau,\eta,\mathcal{D}_{Y_H}) = \nu(K(x;\Sigma);\tau,\eta) \tag{6.12}$$

$$\nu(x;\tau,\eta) = \frac{1-e^{-\tau x}}{1+e^{\eta}e^{-e^{\tau x}}} \tag{6.13}$$

$$K(x;\Sigma) = \frac{1}{n}\sum_{\mathcal{D}_{Y_H}} |\Sigma|^{-\frac{1}{2}}\phi\left(\Sigma^{-\frac{1}{2}}(x-x_i)\right) \tag{6.14}$$

where $K(x;\Sigma):\mathbb{R}\to\mathbb{R}^+$ is a kernel density estimate; $\phi(x)$ is the choice of the kernel function; and $\Sigma\in\mathbb{R}^{n_{d_x}\times n_{d_x}}$ is the bandwidth parameter. For all experiments in this chapter, a Gaussian kernel is chosen, defined by $\phi(x) = (2\pi)^{-\frac{n_{d_x}}{2}}|\Sigma|^{-\frac{1}{2}}e^{-\frac{1}{2}x^\top\Sigma^{-1}x}$. Although different kernel functions can be used, the choice of kernel has a small effect on the estimated density compared to the choice of bandwidth parameter [106]. The Gaussian kernel is used because it is smooth and makes it possible to analyse the gradient properties of f.

$\nu(x;\tau,\eta):\mathbb{R}\to[0,1]$, where $\tau\in\mathbb{R}$ and $\eta\in\mathbb{R}^+$, maps $K(x;\Sigma)$ to a value between $[0,1]$. This 'sigmoid type' function allows for greater flexibility of prior assumptions to be placed about the transitory regions between Hi-Fi data and Lo-Fi data (or prior) primarily through η, and about how much trust can be placed in regions based on the density of Hi-Fi points primarily through τ. This function is essential because there are a limited number of modelling assumptions that can be placed through the bandwidth parameter Σ. Figure 6.3 plots $\nu(x;\tau,\eta)$ for different values of η and τ.



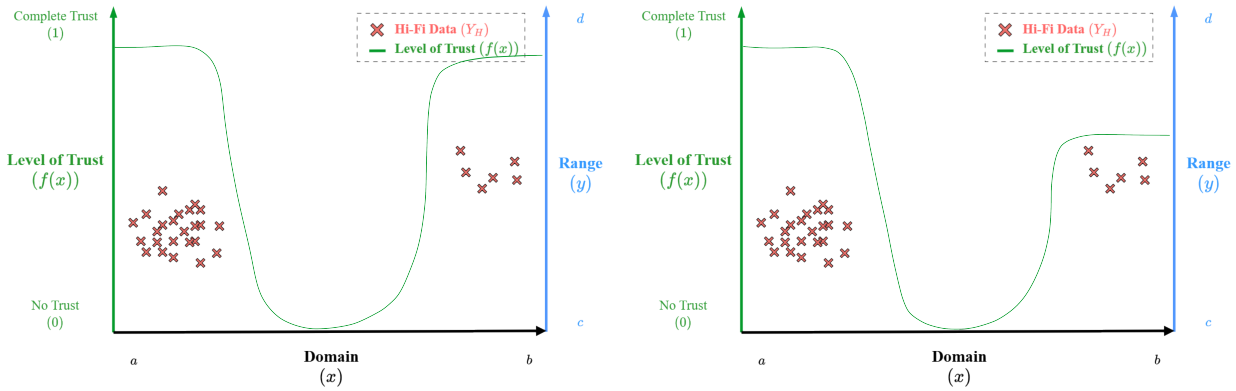
(a) Fixing $\tau = 1$ and varying η

(b) Fixing $\eta = 1$ and varying τ

Figure 6.3: Plots of $\nu(x;\tau,\eta)$ varying η and τ in $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$

Figure 6.3 (a) clearly shows that as η increases, $\nu(x;\tau,\eta)$ shifts to the right. This corresponds to the threshold density value required to have an increase of $f(x)$, or an increase in trust placed in the Hi-Fi data at any one point. As η increases, so does the threshold value. Figure 6.3 (b) shows that as τ increases, the rate of change of $\nu(x;\tau,\eta)$ from 0 to 1 increases. This corresponds to the speed with which transitions occur from regions of Hi-Fi data to sparse regions.

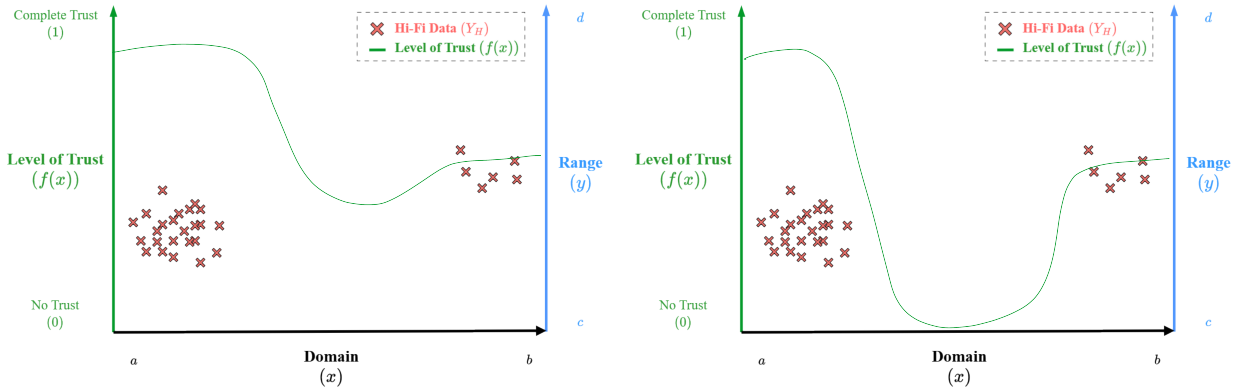
Figures 6.4 and 6.5 show cartoon plots of $f(x)$ when hyper-parameters τ and η are large and small in order to provide an intuitive understanding of how this affects the trust placed in the Hi-Fi dataset.



(a) η small

(b) η large

Figure 6.4: Cartoon plots showing the effect of a small or large η on $f(x)$ for a made-up 1-D example.



(a) τ small

(b) τ large

Figure 6.5: Cartoon plots showing the effect of a small or large τ on $f(x)$ for a made-up 1-D example.

Figures 6.4 (a) and (b) show that as η increases, the level of trust in the cluster of data points on the right decreases. Figures 6.5 (a) and (b) show that as τ increases, the rate of transition from regions of Hi-Fi data to sparse regions decreases, as shown by the sharpness and shallowness of the dip in the middle of the domains.

Functional forms of α_0, β_0 and $\sigma^2_{\text{mis}}(x)$

α_0 is a free parameter; however, by the constraints of the inverse-gamma distribution the value must be greater than 0. The default chosen in this thesis is $\alpha_0 = 1$ for all experiments. $\sigma^2_{\text{mis}}(x) = \frac{\lfloor \sigma^2_{Y_H}(x) \rfloor}{2(\alpha_0+1)}$, where $\lfloor \cdot \rfloor$ denotes an operator that fixes the enclosed quantity with respect to differentiation. In practice, this represents the stop gradient operation which prevents gradient flow during back-propagation. Finally, β_0 does not have a functional form, but is selected as the prior value for mismatch uncertainty in the prior regions of the model. This is equivalent to $\frac{\beta_0}{\alpha_0+1}$, as will be shown in the next section.

6.2.4 Updated analytical expressions

Substituting the functional forms of the hyper parameters into the analytical Equations (6.9b)) and (6.10c), Equations (6.15) and (6.16) are obtained. The dependence on hyper-parameters in these functions are dropped for simplicity.

$$\mu_{Y_H}(x) = \frac{\frac{2(\alpha_0+1)f(x)}{(1-f(x))}\frac{\lfloor\sigma^2_{Y_H}(x)\rfloor}{2(\alpha_0+1)}\hat{\mu}_{Y_H}(x) + \sigma^2_{Y_H}(x)\left(\mu_{Y_L}(x)+\mu_{Y_D}(x)\right)}{\frac{2(\alpha_0+1)f}{(1-f(x))}\frac{\lfloor\sigma^2_{Y_H}(x)\rfloor}{2(\alpha_0+1)} + \sigma^2_{Y_H}(x)}$$

$$= f(x)(\hat{\mu}_{Y_H}(x)) + (1-f(x))\left(\mu_{Y_L}(x)+\mu_{Y_D}(x)\right) \tag{6.15}$$

$$\sigma^2_{Y_H}(x) = \frac{\frac{2(\alpha_0+1)f(x)}{(1-f(x))}\frac{1}{\alpha_0+1}(\mu_{Y_H}(x)-\hat{\mu}_{Y_H}(x))^2 + 2\frac{\beta_0}{\alpha_0+1}}{\frac{2(\alpha_0+1)f(x)}{(1-f(x))}\frac{1}{\alpha_0+1}+2}$$

$$= f(x)(\mu_{Y_H}(x)-\hat{\mu}_{Y_H}(x))^2 + (1-f(x))\frac{\beta_0}{\alpha_0+1} \tag{6.16}$$

Equations (6.15) and (6.16) clearly show that the chosen functional forms reduce the optimal solutions to a linear interpolator between the data and the prior, driven by $f(x)$. Observing the right hand side of Equation (6.16), the prior mismatch uncertainty is equivalent to $\frac{\beta_0}{\alpha_0+1}$ since this is what the predictive variance defaults to in regions of sparsity.

6.3 Outline of modelling process

Having defined the Bayesian framework, the general modelling process for its implementation is illustrated by Figure 6.6 below.

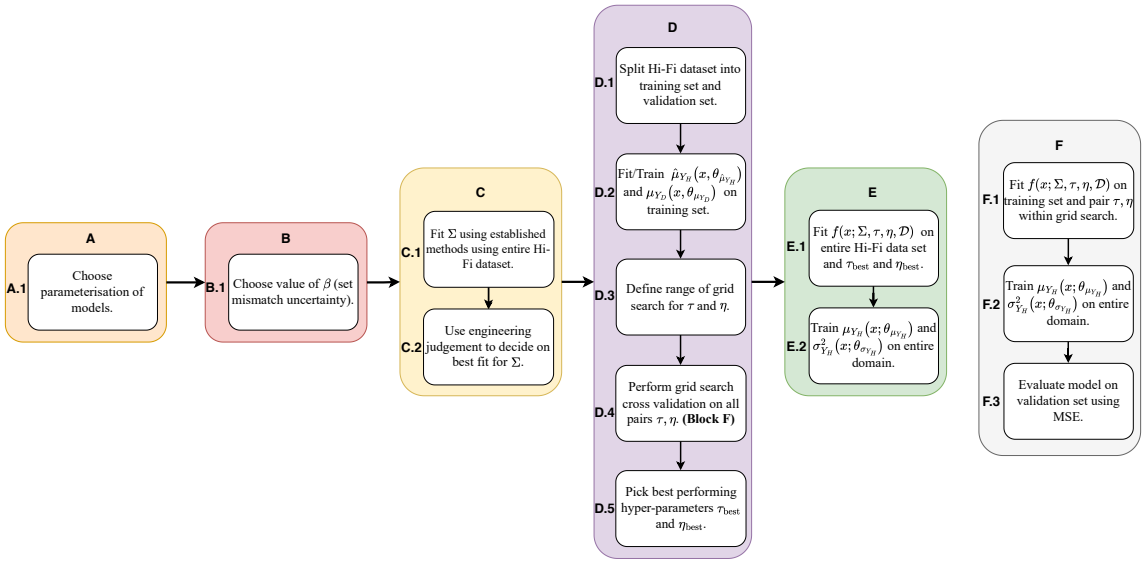


Figure 6.6: Diagram of modelling process. Blocks group related tasks to one another and start from Block A and ends at Block E. Block F represents the subprocess of grid search cross-validation to tune the hyper-parameters τ and η.

The subsequent sections walk through the modelling process step by step, with each task discussed in the order shown in Figure 6.6 and explicitly linked to its corresponding block. In the process, it is assumed that $\mu_{Y_L}(x; \theta_{\mu_{Y_L}})$ and $\mu_{Y_D}(x; \theta_{\mu_{Y_D}})$ are given and therefore not included in the diagram.

6.4 Methodology

Having defined the NL by Equation (6.8), the optimisation problem is defined as

$$\sum_{\substack{\hat{\mu}_{Y_H}(x), \mu_{Y_L}(x), \\ \mu_{Y_D}(x), \Lambda}} \min_{\mu_{Y_H}(x), \sigma^2_{Y_H}(x)} \mathcal{L}_{nl} \tag{6.17}$$

The functions $\hat{\mu}_{Y_H}(x)$, $\mu_{Y_L}(x)$, $\mu_{Y_D}(x)$, $\mu_{Y_H}(x)$ and $\sigma^2_{Y_H}(x)$ are replaced by trainable, parametrised models $\hat{\mu}_{Y_H}(x; \theta_{\hat{\mu}_{Y_H}})$, $\mu_{Y_L}(x; \theta_{\mu_{Y_L}})$, $\mu_{Y_D}(x; \theta_{\mu_{Y_D}})$, $\mu_{Y_H}(x; \theta_{\mu_{Y_H}})$, and $\sigma^2_{Y_H}(x; \theta_{\sigma^2_{Y_H}})$, respectively. Assuming the existence of trained models for $\hat{\mu}_{Y_H}(x; \theta_{\hat{\mu}_{Y_H}})$, $\mu_{Y_L}(x; \theta_{\mu_{Y_L}})$, $\mu_{Y_D}(x; \theta_{\mu_{Y_D}})$ and a choice of

hyper-parameters Λ, the objective becomes the following

$$\sum_{\substack{\hat{\mu}_{Y_H}(x;\theta_{Y_H}),\mu_{Y_L}(x;\theta_{\mu_{Y_L}}),\\ \mu_{Y_D}(x;\theta_{\mu_{Y_D}}),\Lambda}} \min_{\theta_{\mu_{Y_H}},\theta_{\sigma^2_{Y_H}}} \mathcal{L}_{nl} \tag{6.18}$$

The training algorithm is described by Algorithm 7.

Algorithm 7 Kernel-based MF modelling learning algorithm (vectorized, batch-based)

Require: Learning rate η

Require: Set of input points $\mathcal{X}$ of mini-batches $\{\mathbf{x}_{\text{batch}}\}$

Require: Untrained distribution models: $\mu_{Y_H}(\mathbf{x};\theta_{\mu_{Y_H}})$, $\sigma^2_{Y_H}(\mathbf{x};\theta_{\sigma^2_{Y_H}})$

Require: Trained or fitted models: $\hat{\mu}_{Y_H}(\mathbf{x};\theta_{\hat{\mu}_{Y_H}})$, $\mu_{Y_L}(\mathbf{x};\theta_{\mu_{Y_L}})$, $\mu_{Y_D}(\mathbf{x},\mu_{Y_L}(\mathbf{x});\theta_{\mu_{Y_D}})$

Require: Hyper-parameters $\alpha_0,\beta_0,\tau,\zeta,\Sigma$

Initialise parameters $\theta = \theta_{\mu_{Y_H}} \cup \theta_{\sigma^2_{Y_H}}$

while not converged **do**

  for all $\{\mathbf{x}_{\text{batch}}\} \in \mathcal{X}$ **do**

    Evaluate model outputs:

$$\mu_{Y_H}(x;\theta_{\mu_{Y_H}}),\sigma^2_{Y_H}(x;\theta_{\sigma^2_{Y_H}}),\hat{\mu}_{Y_H}(x;\theta_{\hat{\mu}_{Y_H}}),\mu_{Y_L}(x;\theta_{\mu_{Y_L}}),\mu_{Y_D}(x,\mu_{Y_L}(x);\theta_{\mu_{Y_D}})$$

    Compute pointwise non-linear loss vector $\mathcal{L}_{nl}(\mathbf{x}_{\text{batch}},\mathbf{y}_{\text{batch}})$

    Compute mean loss over batch:

$$\mathcal{L}_{nl} = \frac{1}{d}\sum_{i=1}^{d}\hat{\mathcal{L}}_{nl,i}$$

    Update θ by gradient descent on $\mathcal{L}_{nl}$

  end for

end while

In practice, when training models directly using the NL described by Equation (6.8) the model was unable to converge. To overcome these issues, an adaptation to Algorithm 5 was developed and implemented for the gradient descent step in the optimisation algorithm above. This was able to solve the convergence issues. The details of this algorithm are in Appendix C.

6.4.1 Choice of models

The tasks carried out in this section describe the tasks in Block A in the overall modelling process described by Figure 6.6. The choice of models for $\mu_{Y_L}(x;\theta_{\mu_{Y_L}})$, $\mu_{Y_H}(x;\theta_{\mu_{Y_H}})$, and $\sigma^2_{Y_H}(x;\theta_{\sigma^2_{Y_H}})$ is neural networks. For all experiments in this chapter, neural networks with 6 hidden layers, 20 hidden units, and exponential Linear Unit (eLU) activation functions were used. The choice of model for $\mu_{Y_D}(x;\theta_{\mu_{Y_D}})$ is a polynomial function of x and $\mu_{Y_L}(x)$. The choice of $\hat{\mu}_{Y_H}(x;\theta_{\hat{\mu}_{Y_H}})$

was a piece-wise linear interpolator for the 1-D experiment, and the Clough-Tocher (CT) scheme was used for the 2-D experiments fitted to the Hi-Fi data. This was deliberate to ensure sensible interpolations and extrapolations. A feed forward NN would be a naive parametrisation for $\hat{\mu}_{Y_H}(x)$, which is known to be unable to make sensible extrapolations outside of the training domain [34]. The general architecture of the models is described by Figure 6.7.

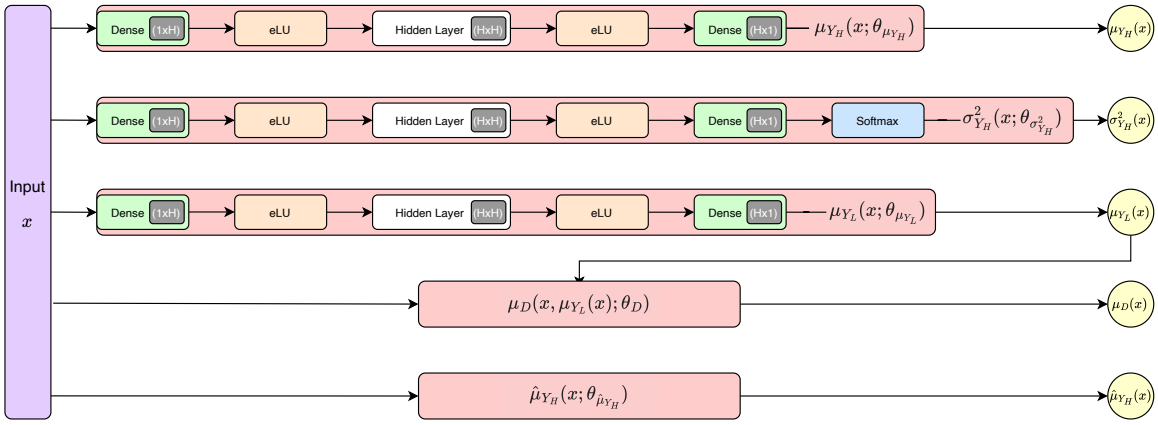


Figure 6.7: General MF model architecture consisting of sub-networks employed for each distributional parameter. eLU activation functions are used in the hidden layers of the NNs. Softmax is applied to $\sigma^2_{Y_H}(x;\theta_{\sigma^2_{Y_H}})$. H is the number of hidden units in the NNs.

For subsequent sections, the dependence on parameters is assumed implicit and dropped from the notation for clarity. The minimisation objective was achieved using back-propagation. Training of $\mu_{Y_H}(x)$ and $\sigma^2_{Y_H}(x)$ occurs throughout the domain. $\hat{\mu}_{Y_H}(x)$, $\mu_{Y_D}(x)$ and $\mu_{Y_L}(x)$ can be evaluated across the entire domain; therefore, implicitly, $\mu_{Y_H}(x)$ and $\sigma^2_{Y_H}(x)$ can be trained anywhere in the domain.

1-D spline interpolator: piecewise linear interpolation

Piecewise linear interpolation was the choice of $\hat{\mu}(x)$ for the 1-D experiments in this chapter. Suppose that the interpolation function is $y = s(x)$ where $x \in \mathbb{R}$ and there are $n+1$ points to interpolate. A function $g(x)$ constructed out of n local linear interpolators is defined by

$$g(x) = g_i(x), \qquad \text{for } x_i \leq x \leq x_{i+1} \tag{6.19}$$

where

$$g_i(x) = a_i(x - x_i) + b_i, \quad a_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}, \quad b_i = y_i \tag{6.20}$$

and $i = 1, ..., n$. Since the purpose of the 1-D experiment is to demonstrate the effect of hyper-parameter choices, this simple interpolator is sufficient, especially as there are no tunable hyper-parameters.

2-D spline interpolator: Clough-Tocher scheme

The Clough-Tocher interpolator was the choice of $\hat{\mu}(x)$ for the 2-D experiments in this chapter. In this approach, the input domain is first triangulated, and a piecewise cubic Bezier polynomial is constructed over each triangle using the Clough-Tocher splitting scheme. The resulting interpolant is guaranteed to be continuously differentiable (C^1) across triangle boundaries. The gradients are selected so that the curvature of the interpolating surface is approximately minimised, yielding a smooth representation of the data. In general, the steps involved in the Clough-Tocher scheme are as follows:

1. The scattered data points are triangulated where each triangle T has vertices A, B and C.
2. Each triangle is divided into 3 smaller subtriangles by connecting the centroid of the triangle to each vertex.
3. A local cubic polynomial is fit per sub-triangle using barycentric coordinates.
4. Coefficients on cubic polynomials are found so that C^1 continuity is enforced.

For more information, see Peter Alfield [107]. The method is extended to handle extrapolations using the value of the nearest data point to the evaluation points.

A key advantage of the Clough-Tocher scheme is the lack of hyper-parameter tuning required. By contrast, RBF kernel interpolation offers more flexibility in the modelling of the data and can naturally handle extrapolation, however, its performance is often highly dependent on the choice of kernel and associated hyper-parameters. Furthermore, extrapolating using the nearest-neighbour value can give control in the extrapolation regions such that predictions do not wildly deviate from the training data.

6.5 1-D sinusoidal example: demonstrating the effects of hyper-parameter choices

This section breaks away from the overall modelling process described by Figure 6.2 in order to demonstrate the effect of hyper-parameter choices Λ ve on the solutions of $\mu_{Y_H}(x)$ and $\sigma^2_{Y_H}$. Models are trained on Hi-Fi and Lo-Fi data generated from distinct 1-D sinusoidal functions with non-linear mismatch between the datasets. 1000 uniformly sampled Lo-Fi training points were sampled from the function $y_L(x) = \sin(8\pi x)$ throughout the domain $x \in [0,1]$, and 500 uniformly sampled Hi-Fi training data points were generated from the function $y_H(x) = (x-\sqrt{2})y_L^2(x)$ in the domain $x \in [0.0, 0.25] \cup [0.75, 1.0]$. The training data and generating functions are shown by Figure 6.8.

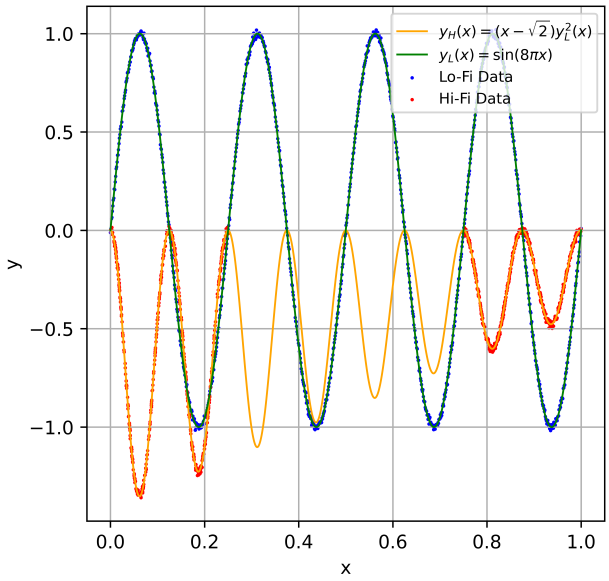
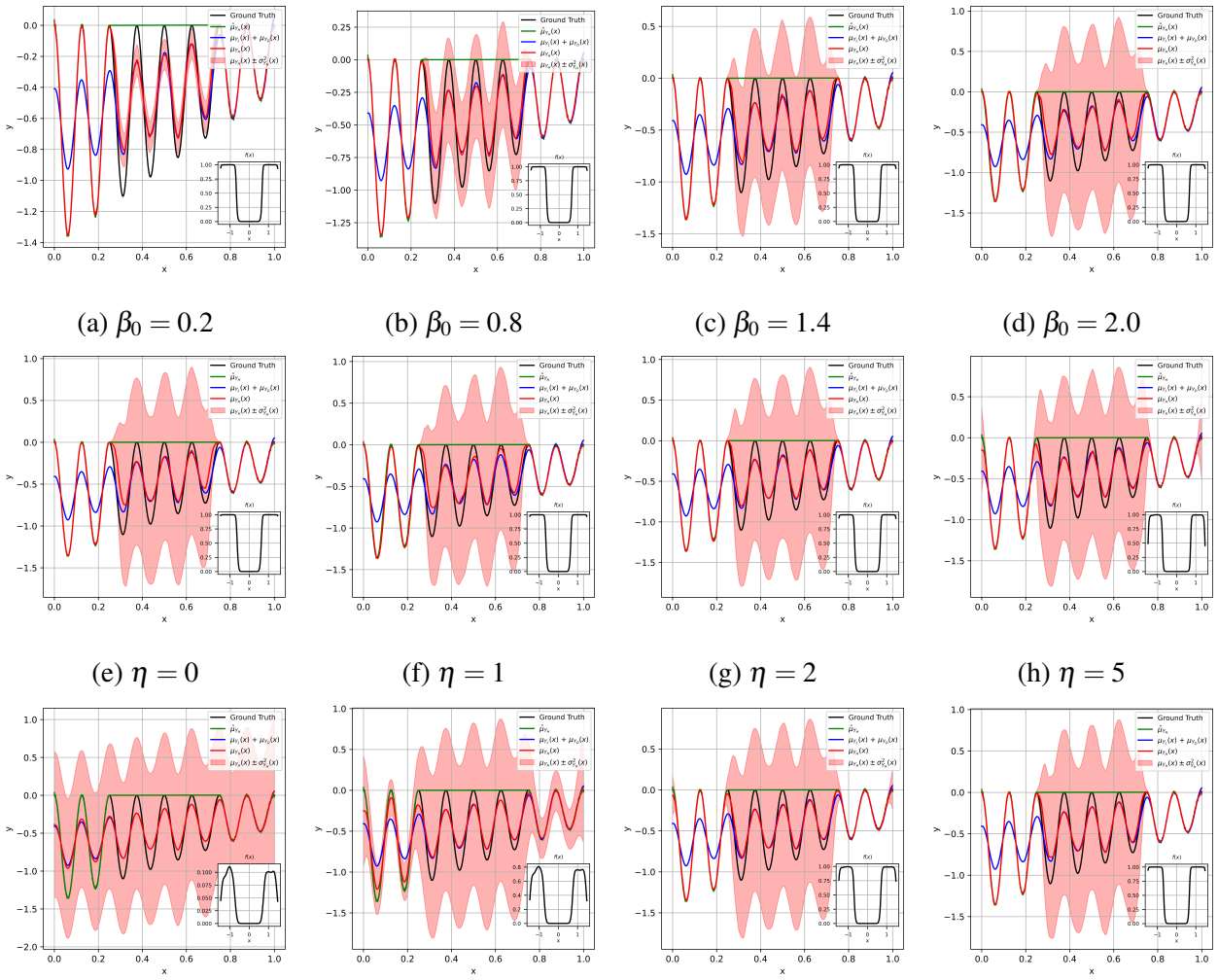


Figure 6.8: Overview of 1-D problem including data regions. 500 Hi-Fi data points in each of $x \in [0.0, 0.25]$ and $x \in [0.75, 1.0]$. 1000 Lo-Fi data points in $x \in [0,1]$.

When training all models, the default hyper-parameters were $\Sigma = 0.08984889$, $\tau = 15$, $\eta = 2$, $\alpha_0 = 1$, $\beta_0 = 2$ and one of the following hyper-parameters was varied over the values $\beta_0 \in \{0.2, 0.8, 1.4, 2.0\}$, $\eta \in \{0,1,2,5\}$ and $\tau \in \{1,5,10,15\}$. The general architecture of the model was the MVE network described by Figure 6.7 which consisted of subnetworks with 6 hidden layers and 20 hidden nodes. The 1-D spline interpolator $\hat{\mu}_{Y_H}(x)$ was initially fit to Hi-Fi data. Then subnetworks $\mu_{Y_L}(x)$ and $\mu_{Y_D}(x)$ were pre-trained for 1000 epochs on the training data using an MSE

loss, before training the subnetworks for 2000 epochs using the Gaussian NLL function. Finally, the μ_{Y_H} and $\sigma^2_{Y_H}$ subnetworks were trained for 3000 epochs using Algorithm 7 in addition to the annealing μ_{Y_H} and $\sigma^2_{Y_H}$ algorithm. The final unscaled results are shown by Figure 6.9.



(a) $\beta_0 = 0.2$ (b) $\beta_0 = 0.8$ (c) $\beta_0 = 1.4$ (d) $\beta_0 = 2.0$

(e) $\eta = 0$ (f) $\eta = 1$ (g) $\eta = 2$ (h) $\eta = 5$

(i) $\tau = 1$ (j) $\tau = 5$ (k) $\tau = 10$ (l) $\tau = 15$

Figure 6.9: Final plots of models with varying β_0, η and τ hyper-parameters trained on the 1-D sinusoidal example. Default hyper-parameters were $\Sigma = 0.08984889$, $\tau = 15$, $\eta = 2$, $\alpha_0 = 1$ and $\beta_0 = 2$. Models were trained keeping all default hyper-parameters fixed whilst varying one parameter in $\beta_0 \in \{0.2, 0.8, 1.4, 2.0\}$, $\eta \in \{0, 1, 2, 5\}$ and $\tau \in \{1, 5, 10, 15\}$.

In all experiments, the weighting of the data term relative to the prior term is determined directly by $f(x)$. When $f(x) \approx 0$, the model relies predominantly on the prior (blue line), whereas when $f(x) \approx 1$, the model learns primarily from the Hi-Fi data (green line).

The top row shows that increasing β_0 leads to a larger mismatch uncertainty within the region $x \in [0.25, 0.75]$. The middle row demonstrates that increasing η reduces the effective threshold value; this is most clearly observed in the tails of $f(x)$, which decreases as η increases. Finally, the bottom row illustrates the influence of τ, where increasing τ sharpens the transition between prior- and data-dominated regions. This effect is particularly evident in Figures 6.9 (k) and (l), where the uncertainty transitions more abruptly near $x = 0.25$ when $\tau = 15$.

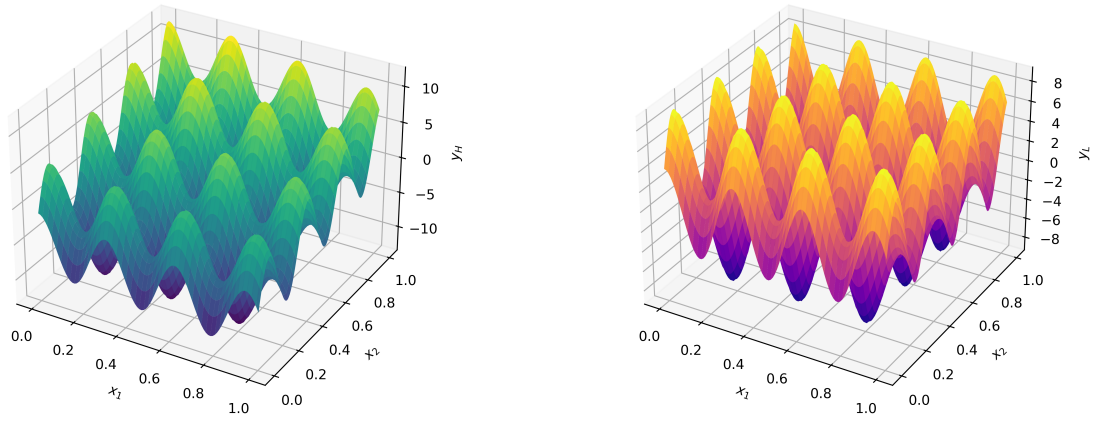
Although the threshold location and the sharpness of the transition depend on both η and τ, in practice each parameter predominantly controls one aspect: η primarily determines the threshold, while τ primarily governs the slope of the transition.

6.6 2-D example: demonstrating effective hyper-parameter tuning

This section continues with the overall modelling process described by Figure 6.6

The approach was implemented on a synthetic 2-D example, including an explanation of how to tune the hyper-parameters of $f(x; \Sigma, \eta, \tau)$ for this problem. Hi-Fi and Lo-Fi generating functions are described by Figure 6.10. The Hi-Fi dataset was generated by partitioning the domain $(x_1, x_2) \in [0,1]^2$ into $n \times n$ non-overlapping, uniformly spaced grids. To introduce sparsity into the data, samples were taken within a randomly chosen radius of a randomly chosen centre. Initially, 10 random points within the domain with centre $c_k = (c_{x_1}, c_{x_2})$ were chosen as the centre of a cluster. For each cluster $k \in \{1, \ldots, 10\}$, a radius $r_k \in [0.05, 0.2]$ was randomly sampled. A random number of samples, $j_k \in [0, 100]$, was then sampled in the circle described by centre point c_k and radius r_k. 542 sparse Hi-Fi data points were generated this way from the generating function $y_H = 5\sin(20x_2) + 4\cos(20x_1) + (x_1^2 - 3x_1 + 8x_2 - 5) + 5(x_1^{0.5}x_2^{0.7})(1 - x_1^{0.6}x_2^{0.5})$. 1000 Lo-fi data points were uniformly sampled from the generating function $y_L = 5\sin(20x_2) + 4\cos(20x_1)$.

Although generated from a known function, the Hi-Fi data in this example are intended to represent measurement data obtained from a physical system. In contrast, the Lo-Fi data represent data from a physics-based model with model mismatch.



(a) (b)

Figure 6.10: Surface plot of Hi-Fi and Lo-Fi generating functions. (a) $y_H = 5\sin(20x_2) + 4\cos(20x_1) + (x_1^2 - 3x_1 + 8x_2 - 5) + 5(x_1^{0.5}x_2^{0.7})(1 - x_1^{0.6}x_2^{0.5})$. (b) $y_L = 5\sin(20x_2) + 4\cos(20x_1)$

The functional form for $\mu_{Y_D}(x)$ is $Ax_1^2 + Bx_1 + Cx_2^2 + Dx_2 + E$, the parameters of which are updated from the available Hi-Fi data. The hyper-parameters Λ are tuned according to the Hi-Fi dataset.

Finally, $\beta_0 = 2$ is fixed, which implies that the mismatch uncertainty in prior regions yields $\sigma^2_{Y_H}(x) = 1$. This would ordinarily be set by an engineer's prior understanding of the problem set-up. This task corresponds to Block B in the overall modelling process described by Figure 6.6.

6.6.1 Fitting Σ

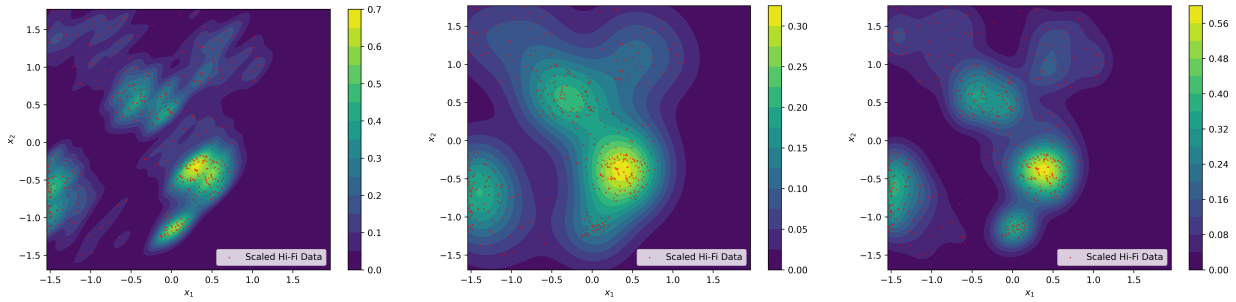
This section implements the tasks in Block C as described in the overall modelling process in Figure 6.6. It is dedicated to going through the process of selecting the hyper-parameters of f in addition to giving justifications for their choices for the 2-D synthetic example.

First, Σ is selected using well-established methods for fitting KDEs (Task B.1 in Figure 6.6). To start with, a good estimate of the probability distribution is required in order to provide engineers with an understanding of where there is information in the domain. Once this is achieved, further modelling assumptions can be placed through η and τ.

The most common well established methods include Direct Plug-In (DPI), Least Squares Cross Validation (LSCV) and Biased Cross Validation (BCV). DPI relies on minimising the Mean Inte-

grated Squared Error (MISE), a metric for evaluating the performance of a bandwidth parameter, by plugging in a parametric assumption directly into an asymptotic estimation of the MISE. The MISE is otherwise intractable without knowing the true density distribution. A plug in without attempting a non-parametric estimation of the data distribution is known as a zero-stage plug in. ℓ-stage plug in methods exist which use non-parametric estimates of the density in the asymptotic estimate of the MISE, initialised through a parametric estimation of the distribution. Increasing the number of stages can reduce the variance of the density at the expense of the bias. LSCV attempts to directly minimise the MISE through cross validation. A subset of the data is used to estimate the KDE for a choice of the bandwidth parameter whilst a hold out set is used to evaluate the choice of the bandwidth parameter. Finally, there is BCV which is a hybrid strategy that combines plug-in and cross validation. A cross-validatory approach is applied to evaluate the bias term in the Asymptotic Mean Integrated Squared Error (AMISE).

The DPI methods have a convergence rate that is much faster than that of cross-validation methods. In theory, they are expected to perform better than LSCV and BCV and is generally the preferred bandwidth selector in the literature. However, cross validation methods may be better suited for highly non-normal and rough densities, where plug in selectors may end up over smoothing. Considering the understanding of the non-linearity of the Lo-Fi data, and observing the Hi-Fi data, a cross-validatory approach is taken for selecting Σ in order to avoid over smoothing of the density. BCV also tends to over smooth compared to the LSCV method, therefore, LSCV is the chosen method for fitting $\Sigma = [0.0173, 0.0161; 0.0161, 0.0214]$. Cross-validatory methods are fit using gradient descent. Figures 6.11 (b) and (c) show the affect of over smoothing in the BCV method and the 2-stage plug in method, compared to the LSCV method, which has more variability.



(a) LSCV method (b) BCV method (c) 2-stage plug-in method

Figure 6.11: Contour plots of the KDE fit on the scaled Hi-Fi data for different choices of Σ.

In this 2-D example, a further evaluation of the fit can be performed by observing the contour plots described by Figure 6.11 (Task C.2 in Figure 6.6). It shows that the BCV and 2-stage plug in method over smooth the density, which corresponds with the theory.

6.6.2 Fitting τ and η

This section implements the tasks in Block D as described in the overall modelling process in Figure 6.6.

Having selected a value for Σ, a grid search was conducted to tune the values of τ and η. The Hi-Fi dataset was first split into training and validation sets using a 70:30 split (Task D.1 in Figure 6.6). $\hat{\mu}_{Y_H}(x)$ was fitted using the training set (Task D.2 in Figure 6.6), $\mu_{Y_D}(x)$ was trained for 2000 epochs on the training set using the Gaussian NLL loss (Task D.2 in Figure 6.6), and $\mu_{Y_L}(x)$ was trained for 2000 epochs on the Lo-Fi data using the Gaussian NLL loss. For every pair $\tau \in [1,2,4,8]$ and $\eta \in [0.5,1,2,4]$ (Task D.3 in Figure 6.6), $f(x;\Sigma,\tau,\eta)$ was fit using all available Hi-Fi data and $\Sigma = [0.0173, 0.0161; 0.0161, 0.0214]$. The models were then trained using Algorithms 7 and 8 for each configuration of (τ,η) for 3000 epochs (Task D.4 (Block F) in Figure 6.6). The validation set was used to evaluate model performance using the MSE, from which the optimal hyper-parameters were deemed to be $\tau = 4$ and $\eta = 1$ (Task D.5 in Figure 6.6). Table 6.1 shows the final results of the validation losses. A contour plot of the resulting function f with the optimal setting of hyper-parameter parameters is shown in Figure 6.12.

Table 6.1: Validation loss for different combinations of τ and η.

τ	$\eta = 0.5$	$\eta = 1$	$\eta = 2$	$\eta = 4$
1	0.1077	0.1165	0.1280	0.1403
2	0.0675	0.0795	0.1048	0.1294
4	0.0056	**0.0041**	0.0053	0.0195
8	0.2370	0.4223	0.9108	0.3142

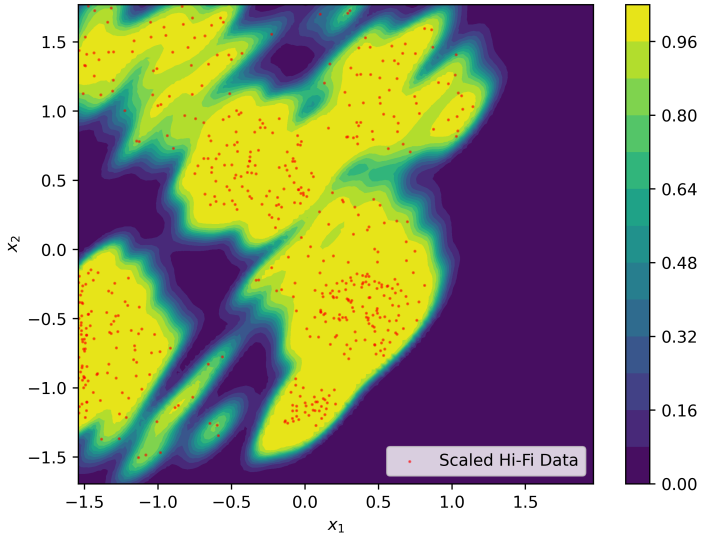
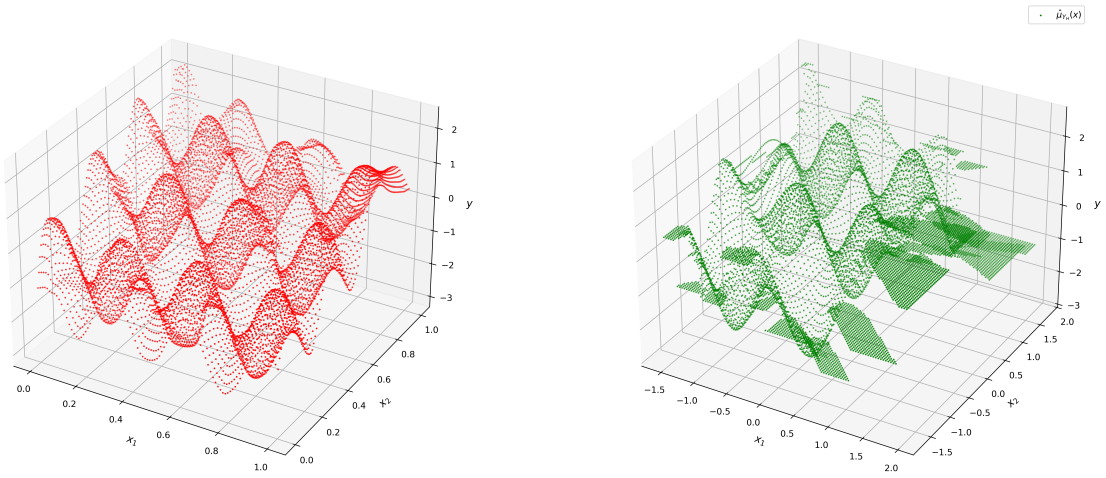


Figure 6.12: Contour plot of f on scaled Hi-Fi data given hyper-parameters $\Sigma = [0.0173, 0.0161; 0.0161, 0.0214]$, $\tau = 4$ and $\eta = 1$.

6.6.3 Results and discussion

This section implements the tasks in Block E as described in the overall modelling process in Figure 6.6.

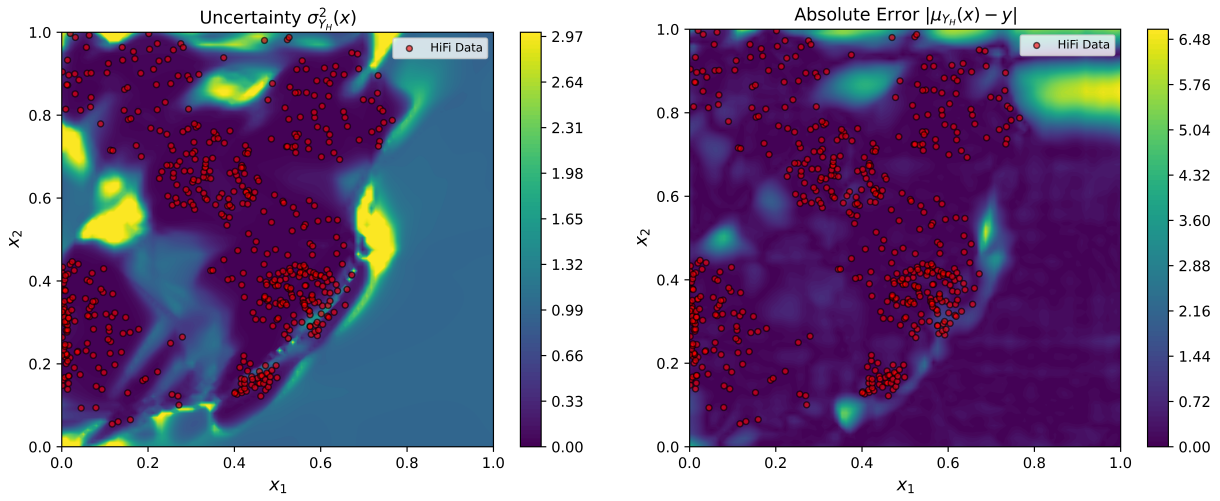
After hyper-parameter selection, f was fit (Task E.1 in Figure (6.6) and a model was trained for 10000 epochs using Algorithm 8 and optimal hyper-parameters. A training scheme was implemented where the learning rate, starting at 1E-03, was multiplied by $\left(\frac{10^{-5}}{10^{-3}}\right)^{\left(\frac{1}{4}\right)}$ every 2000 epochs (Task E.2 in Figure (6.6). The final model was evaluated on 10000 test points uniformly sampled across the entire problem domain. The final plots are shown in Figure 6.13.



(a) $\mu_{Y_H}(x)$ (b) $\hat{\mu}_{Y_H}(x)$

Figure 6.13: 3D plots of trained model evaluated at 10000 test points. (a) The predictive mean output μ_{Y_H} and (b) The adapted Clough-Tocher spline fit on the Hi-Fi data $\hat{\mu}_{Y_H}(x)$

Figure 6.13 (a) shows the final predicted mean, $\mu_{Y_H}(x)$, after training. Qualitatively, the model appears to have converged well throughout the entire domain. Figure 6.13 (b) illustrates the CT interpolator fitted to the model output. The interpolator follows the predictions closely in regions with a high density of data, while in extrapolation regions the predicted mean, $\hat{\mu}_{Y_H}(x)$, tends towards a constant value. Figure 6.14 presents the predictive uncertainty and the absolute error on the test points. For clarity of presentation, the maximum variance shown in the uncertainty plot has been truncated to a value of 3.



(a) (b)

Figure 6.14: (a) Predictive uncertainty $\sigma^2_{Y_H}(x)$ evaluated at 10000 test points. The variance is truncated at a maximum value of 3. (b) Absolute prediction error $|\mu_{Y_H}(x) - y|$ evaluated at 10000 test points.

Despite reasonable convergence for $\mu_{Y_H}(x)$, Figure 6.14 (a) shows many areas where the uncertainty spikes above the prior mismatch uncertainty (equal to 1). These seem to occur in transitory regions between the data and prior regions of the model. Although Algorithm 8 is capable of improving the convergence of $\mu_{Y_H}(x)$ there seems to be difficulty in converging to the correct variance in transitory regions of the domain.

Figure 6.14 (b) further shows that the model accuracy deteriorates for $x_1 > 0.8$ and $x_2 > 0.8$, which can be attributed to the poor predictive performance of $\mu_{Y_L}(x) + \mu_{Y_D}(x)$ in these regions. The reduced accuracy is also evident in the transitory regions. This further corroborates the fact that the model has difficulty learning reliably in these areas. Performance in both the Hi-Fi and test datasets is summarised in Table 6.2.

Table 6.2: Results of $\mu_{Y_H}(x)$ evaluated on test points across the entire domain.

	Hi-Fi Data		**Test Data**	
Model	**MSE**	**MAE**	**MSE**	**MAE**
$\mu_{Y_H}(x)$	9.02E-02	1.83E-01	1.61E-00	5.97E-01
$\hat{\mu}_{Y_H}(x)$	3.10E-14	9.50E-08	1.67E+01	2.32E-00
$\mu_{Y_L}(x)+\mu_{Y_D}(x)$	2.62E-00	7.55E-01	3.61E-00	9.74E-01

The performance of $\mu_{Y_H}(x)$ on the Hi-Fi data is at least an order of magnitude better than that of the prior $(\mu_{Y_L}(x)+\mu_{Y_D}(x))$. This is expected because regions with high data density had contour values close to 1, meaning that in these regions the model is more heavily based on $\hat{\mu}_{Y_H}(x)$, which itself has a very low error.

In the test data, $\mu_{Y_H}(x)$ outperforms both $\hat{\mu}_{Y_H}(x)$ and $\mu_{Y_L}(x)+\mu_{Y_D}(x)$ in both the MSE and MAE metrics. The relative improvement is more apparent in the MAE than in the MSE, which suggests that although the model performs well on average, there remain a small number of test points that contribute disproportionately to the squared error.

Overall, these results indicate that the Kernel-based MF approach is able to successfully exploit both the information present in the prior and the information from the Hi-Fi data, leading to improved predictive accuracy over models trained on either source of information alone.

6.7 Comparisons to benchmark approaches

In this section, the results of the Kernel-based approach applied to the 2-D example are compared against three other multi-fidelity modelling methods. Each of these methods incorporate a GP into its modelling process. The three methods are Dehghani et al.'s [87] fidelity weighted learning approach, Perdikaris et al.'s [81] non-linear information fusion algorithm, and Raissi et al.'s [82] deep multi-fidelity Gaussian Processes approach. A brief description of how each method work's is provided in the next section. For more detailed explanations refer to the original papers.

6.7.1 Deep multi-fidelity Gaussian Process

Raissi et al. [82] developed a hybrid NN and GP architecture that maps Lo-Fi and Hi-Fi input data into a shared latent space using an NN. This latent representation is then passed as input to a GP, whose kernel is carefully chosen to capture correlations between the Lo-Fi and Hi-Fi data. The modelling process is summarised below in the form of a multivariate Gaussian Process.

$$\begin{bmatrix} f_1(h) \\ f_2(h) \end{bmatrix} \sim \mathcal{GP}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} k_{11}(h,h') & k_{12}(h,h') \\ k_{21}(h,h') & k_{22}(h,h') \end{bmatrix} \right) \tag{6.21}$$

Here, h and h' correspond to a neural network mapping of the input space i.e. $h(\boldsymbol{x}) \rightarrow \mathbb{R}^n$; $k_{11} \equiv k_1$, $k_{12} \equiv k_{21} \equiv \rho k_1$, and $k_{22} \equiv \rho^2 k_1 + k_2$ given choices of kernel functions k_1 and k_2. The Hi-Fi process is modelled by $f_2(h)$ and the Lo-Fi process is modelled by $f_1(h)$. The parameters to train the model are the neural network parameters, kernel parameters and ρ. Predictions are made using the predictive distribution defined by

$$p\left(f_2(h(x_*)) \mid x_*, \boldsymbol{x}_1, \boldsymbol{f}_1, \boldsymbol{x}_2, \boldsymbol{f}_2\right) \tag{6.22}$$

Here, x_* corresponds to test input points, $\{\boldsymbol{x}_1, \boldsymbol{f}_1\}$ and $\{\boldsymbol{x}_2, \boldsymbol{f}_2\}$ correspond to input and outputs of the Lo-Fi and Hi-Fi datasets respectively. Figure 6.15 gives an overview of the method.

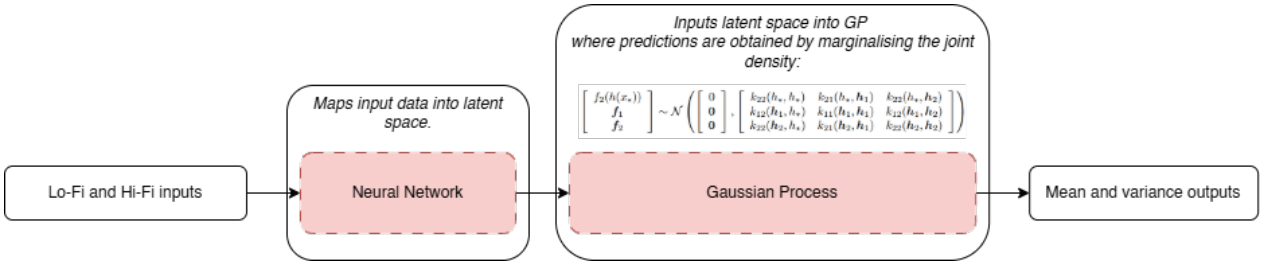


Figure 6.15: A composite NN and GP architecture which learns latent features using a NN before before being input into a GP with a careful choice of covariance kernel which learns the correlations between the datasets.

6.7.2 Non-linear information fusion algorithms

Perdikaris et al. [81] developed an autoregressive GP-based MF method capable of learning non-linear correlations between datasets of different fidelities. Given a Lo-Fi dataset and a Hi-Fi dataset,

a Lo-Fi GP is trained exclusively on the Lo-Fi data. A higher-tier model, or 'Hi-Fi' GP, is then trained on the Hi-Fi data, with its inputs augmented by the mean prediction of the Lo-Fi GP. The index 1 denotes the low-fidelity level and 2 the high-fidelity level. The kernel used to train the high-fidelity GP is

$$k_g(\boldsymbol{x},\boldsymbol{x}') = k_\rho(\boldsymbol{x},\boldsymbol{x}';\theta_\rho)k_f\left(f_1^*(\boldsymbol{x}),f_1^*(\boldsymbol{x}');\theta_f\right) + k_\delta(\boldsymbol{x},\boldsymbol{x}';\theta_\delta), \tag{6.23}$$

Here, $\boldsymbol{x},\boldsymbol{x}' \in \mathbb{R}^n$ denote two input vectors in the n-dimensional input space, and $k_g(\cdot,\cdot)$ is the composite covariance function used for the high-fidelity Gaussian process. The kernel $k_\rho(\cdot,\cdot;\theta_\rho)$ models the input-dependent correlation (or scaling) between the Lo-Fi and Hi-Fi processes, while $k_f(\cdot,\cdot;\theta_f)$ is a covariance function defined over the predicted Lo-Fi outputs, allowing the Hi-Fi model to exploit similarities in the Lo-Fi predictions. Here, $f_1^*(\boldsymbol{x})$ and $f_1^*(\boldsymbol{x}')$ denote the posterior mean predictions of the trained Lo-Fi Gaussian process evaluated at $\boldsymbol{x}$ and $\boldsymbol{x}'$, respectively. The additive kernel $k_\delta(\cdot,\cdot;\theta_\delta)$ models the discrepancy between the transformed Lo-Fi predictions and the high-fidelity observations. The hyper-parameter sets associated with the three kernels are denoted by θ_ρ, θ_f, and θ_δ. Each kernel adopts the squared-exponential automatic relevance determination form

$$k_1(\boldsymbol{x},\boldsymbol{x}';\theta_1) = \sigma_1^2 \exp\left(-\frac{1}{2}\sum_{i=1}^{n} w_{i,1}(x_i - x_i')^2\right), \tag{6.24}$$

where σ_1^2 is the signal variance, controlling the overall magnitude of the covariance, and n is the number of input dimensions. The quantities x_i and x_i' denote the i^{th} components of the input vectors $\boldsymbol{x}$ and $\boldsymbol{x}'$, respectively. The parameter $w_{i,1}$ is the ARD weight associated with the i^{th} input dimension, where larger values correspond to shorter characteristic length scales and therefore a more rapid decay in covariance with distance along that dimension. The kernel hyper-parameter set is given by

$$\theta_1 = \left\{\sigma_1^2, w_{1,1}, \ldots, w_{d,1}\right\}. \tag{6.25}$$

Similarly, the kernels k_ρ, k_f, and k_δ each possess their own independent hyper-parameter sets, denoted by θ_ρ, θ_f, and θ_δ, respectively, while sharing the same ARD squared-exponential functional form. Figure 6.16 gives an overview of the method.

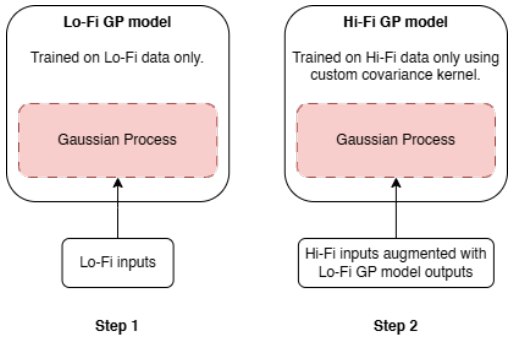


Figure 6.16: Lo-Fi GP model is trained exclusively using Lo-Fi data. Hi-Fi GP model is trained using the Hi-Fi data in Step 2, however the input data is augmented with the corresponding Lo-Fi model mean output such that non-linear relationships between the two fidelities of datasets can be learned.

6.7.3 Fidelity weighted learning

Dehghani et al.'s [87] fidelity weighted learning is a semi-supervised student-teacher multi-fidelity modelling approach. Initially, a student NN model is trained on Lo-Fi data. The first layer from this model is frozen and is used as a feature extractor. A GP is then trained exclusively on Hi-Fi data before first passing the Hi-Fi inputs into the feature extractor. Finally, the student model is trained again on the Hi-Fi and Lo-Fi data. However, the importance placed on each data-point is determined by the uncertainty output when inputs are passed through the teacher GP model. Figure 6.17 gives an overview of the method.

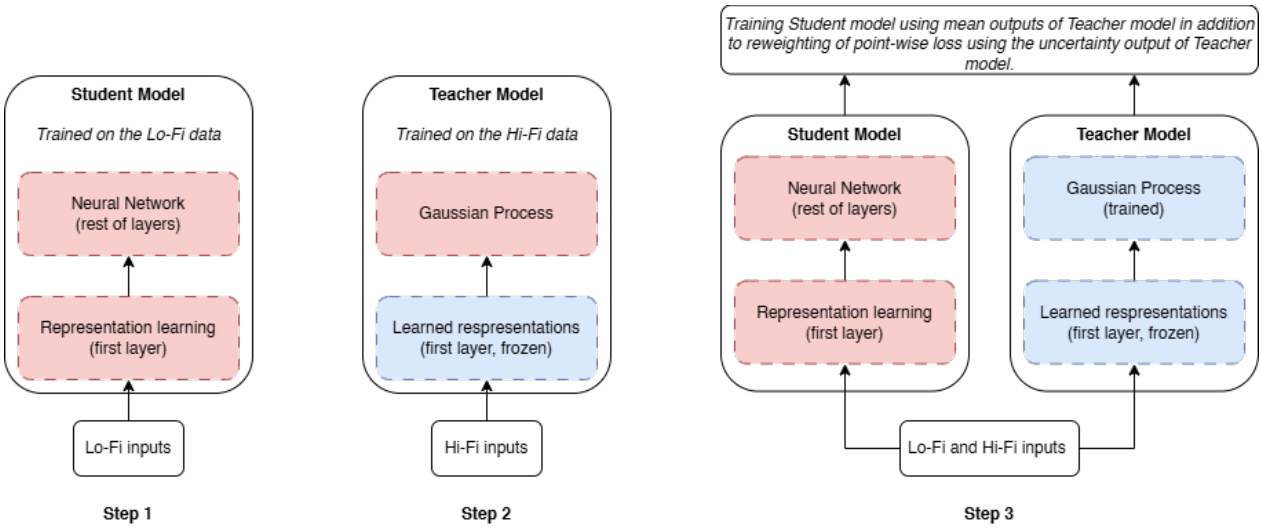


Figure 6.17: A student NN model is trained in Step 1 on the Lo-Fi data. The first layer is extracted and frozen, then compounded as an input to a student GP model. The compounded GP model is trained using the Hi-Fi data in Step 2. Finally, in Step 3, the student model is retrained using guidance from the teacher model.

6.7.4 Training, test data and model parameters

The same training and test data are used as described in Section 6.6.3. The model architectures, training parameters and hyper-parameter choice are described in Table 6.3 below. A Base GP and Base NN model are included for comparison, which are trained exclusively on Hi-Fi data.

Table 6.3: Model architectures, hyper-parameters, and training configurations used for the baseline methods on the 2-D heteroscedastic example.

Method Name	**Model architecture**	**Hyper-parameters**	**Training information**
Dehghani NN (Dehghani et al. [87]	Neural Network [64, 32] Gaussian Process model	Mean Kernel = Constant Covariance Kernel = RBF Kernel (ARD) Loss weighting β = 1.0	Step 1 (NN pretrain): epochs = 50000, lr = 0.001 Step 2 (GP fit): epochs = 50000, lr = 0.001 Step 3 (NN reweighted retrain): epochs = 50000, lr = 0.001
Perdikaris GP (Perdikaris et al. [81])	Model 1: Gaussian Process (Lo-Fi) Model 2: Gaussian Process (Hi-Fi, autoregressive)	Model 1: Mean Kernel = Constant Covariance Kernel = RBF Kernel Model 2: Mean Kernel = Constant Covariance Kernel = RBF($\hat{\mu}_{\text{lofi}}$, ARD dim=1) $\times$ RBF($\mathbf{x}$, ARD dim=d) $+$ RBF($\mathbf{x}$, ARD dim=d)	Step 1 (Model 1): epochs = 3000, lr = 0.001 Step 2 (Model 2): epochs = 3000, lr = 0.001
Raissi GP (Raissi et al. [82])	Neural Network [32, 16, 4] (feature extractor) Gaussian Process model (multi-fidelity kernel)	Mean Kernel = Zero Covariance Kernel = Multi-fidelity kernel: Lo-Lo: RBF(z) Lo-Hi / Hi-Lo: ρ·RBF(z) Hi-Hi: ρ^2·RBF(z) + RBF$_\delta$(z) (z = NN($\mathbf{x}$), NN output dim = 2) Correlation parameter ρ learned jointly	epochs = 20000 lr = 0.001 (joint NN + GP training)
Baseline GP	Gaussian Process model	Mean Kernel = Constant Covariance Kernel = RBF Kernel	epochs = 3000 lr = 0.001
Baseline NN	Neural Network [64, 32]		epochs = 50000 lr = 0.001

The NNs used in Raissi GP, Dehghani NN and Baseline NN methods are sufficiently parameterised to capture the features in the data. The number of epochs in every training instance was chosen to allow for ample time for convergence. The learning rate was kept fixed at a small value 0.001 for training all models in order to prevent overshooting. The choice of kernel functions was the default options described in each paper. In Dehghani NN method, a value of $\beta = 1.0$ is the fixed loss weighting used when training the joint model in step 3. This value was chosen since it was recommended as a good default choice in their paper.

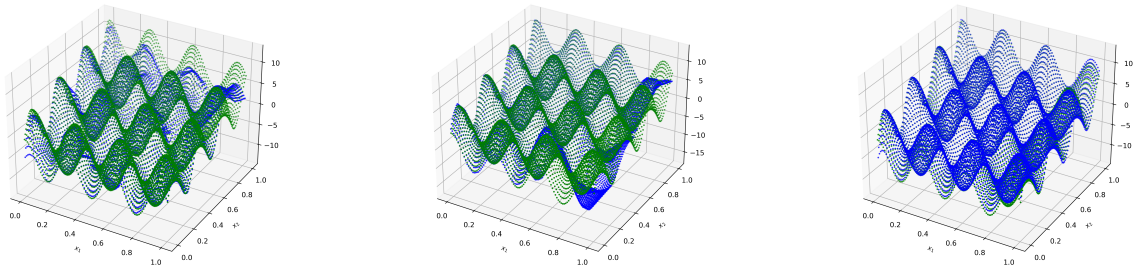
6.7.5 Results

The test set performance of the baseline models and the proposed Kernel-based approach is evaluated using the MSE, with the results summarised in Table 6.4. The plots of the predictions when evaluated on the test data are shown in Figure 6.18.

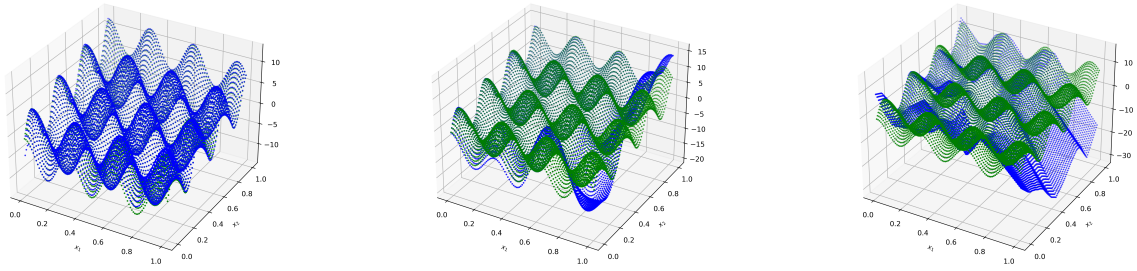
Table 6.4: Mean Squared Error (MSE) comparison between the baseline models and the proposed Kernel-based approach.

Model	**MSE**
Kernel-based approach	1.61E+00
Dehghani NN	5.73E+00
Perdikaris GP	**8.68E−02**
Raissi GP	1.17E−01
Base GP	1.12E+01
Base NN	5.64E+01

Table 6.4 shows the Perdikaris GP and Raissi GP models are the top performing models. In contrast, the Base GP and Base NN are the worst performing models. This is expected since the model is only trained on the Hi-Fi data and therefore extrapolates poorly outside of the Hi-Fi domain. The Base GP has better accuracy than Base NN, which can be explained by the capacity of the kernel function to capture the sinusoidal behaviour of the data. A good prior choice of kernel function can lead to better extrapolation performance. The Kernel-based approach does not perform as well as the Perdikaris GP and Raissi GP models; both of these models can better capture non-linear correlations between the datasets. Comparatively, the Kernel-based approach uses an additive correction on the Lo-Fi data of the form $\mu_D(x) = Ax_1^2 + Bx_1 + Cx_2^2 + Dx_2 + E$, which is not expressive enough to capture the discrepancy between the generating functions described by $(x_1^2 - 3x_1 + 8x_2 - 5) + 5(x_1^{0.5}x_2^{0.7})(1 - x_1^{0.6}x_2^{0.5})$. A non-linear mapping of $\mu_{Y_L}(x)$ would do a better job of capturing the mismatch between the Hi-Fi and Lo-Fi processes. The Kernel-based approach does better than the Dehghani NN model, this is probably due to no correction being applied to the Lo-Fi student model in Dehghani's approach. Furthermore, this could be due to the use of a CT model as a surrogate model to represent $\mu_{Y_H}(x)$ in the Kernel-based approach, which uses the nearest neighbour when extrapolating, ensuring sensible predictions in regions of no data, which are not wildly away from the distribution of the Lo-Fi data. In comparison, the Dehghani NN model, defaults to extrapolations made by a NN trained on Lo-Fi data, which could contribute towards predictions made further away from the true Lo-Fi distribution.



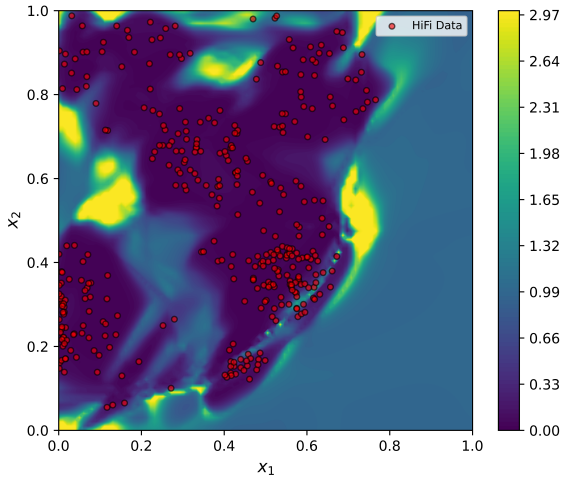
(a) Kernel-based approach (b) Dehghani NN (c) Perdikaris GP



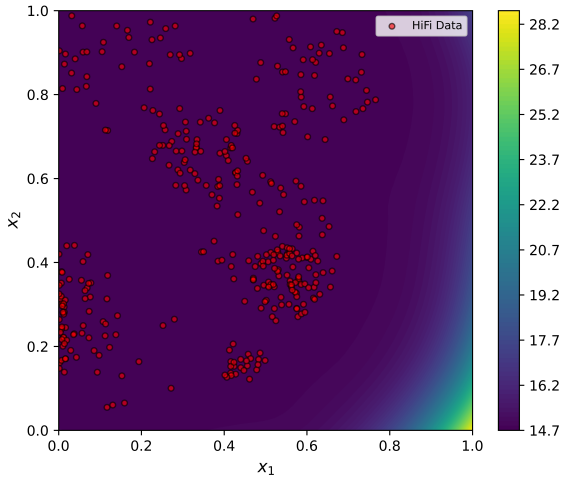
(d) Raissi GP (e) Base GP (f) Base NN

Figure 6.18: Predictions (Blue) of each model evaluated on the 10000 test points (Green).

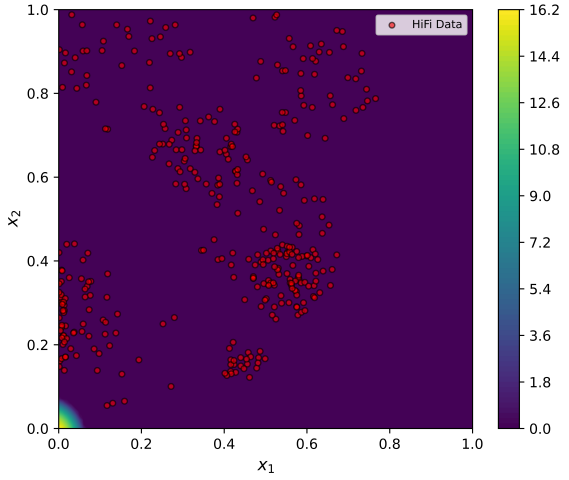
Figure 6.18 corroborates with the results shown in Table 6.4. It can be seen in Figures 6.18f and 6.18e that the Baseline NN struggles in extrapolation in comparison to the Base GP, the latter which is even able to capture the sinusoidal nature of the ground truth. It can be seen in Figure 6.18b that the Dehghani NN model does poorly in region where there is no Hi-Fi data, particularly where $x_1 > 0.8$ and $x_2 \in [0.0, 1.0]$. The Kernel-based approach performs reasonably well in this area, however it struggles where $x_1 > 0.2$ and $x_2 > 0.8$ which reflects the limited expressiveness of the corrective term on the Lo-Fi model. The best performing models are the Perdikaris GP and Raissi GP, and this is shown by Figures 6.18c and 6.18d where the predictions fit closely to the ground truth. The uncertainty plots are shown for the models which are capable of producing uncertainty predictions in Figure 6.19.



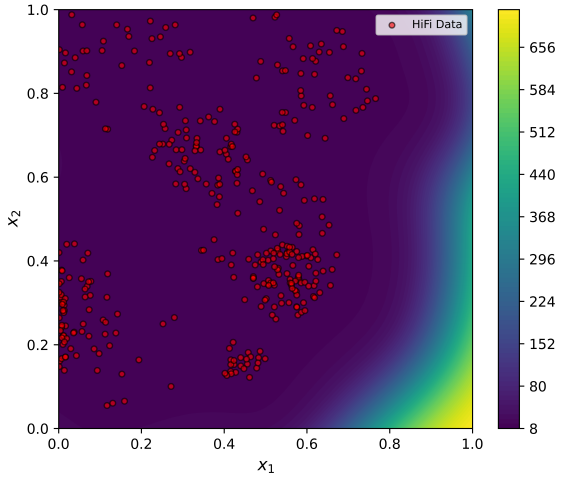
(a) Kernel-based approach



(b) Perdikaris GP



(c) Raissi GP



(d) Base GP

Figure 6.19: Predictive uncertainty $\sigma^2_{Y_H}(x)$ for each GP-based model evaluated on the test grid, with Hi-Fi training points overlaid.

A direct comparison with the Kernel-based approach isn't conducted since the uncertainty quantifies different things in each instance; however, Figure 6.19 can provide an indication if the model is doing the correct thing. Figures 6.19b and 6.19c, 6.19d all show that the model has smaller uncertainties in the regions where there is poor mean prediction. This is corroborated by Figures 6.18c, 6.18d and 6.18e. The Raissi GP has high uncertainty in the region local to $(x_1, x_2) = (1, 1)$, in the same area the mean prediction is inaccurate. The Base GP model has the largest uncertainty

of the three baseline models which reflects how it has the worst prediction accuracy of the models. The Perdikaris GP and Raissi GP uncertainties capture the epistemic uncertainty, which reflects the models ability to capture the information. Larger uncertainties reflect that the model is less confident in being able to capture the data and (or) the relationship between the Hi-Fi and Lo-Fi processes. This value is chosen a priori in the Kernel-based approach via the quantification of the model mismatch. It is worth reiterating that no aleatoric noise is captured in any of these methods.

6.7.6 Comparing computational cost at training and inference

Table 6.5 provides a comparison of the computational cost between the different methods at training and inference.

Table 6.5: Cost at training and inference for the Kernel-based approach and baseline models. N is the number of training data points, P is the number of parameters in a neural network and E is the number of epochs.

Method	**Cost at training**	**Cost at inference**
Kernel-based approach	Training surrogate Lo-Fi only model: $O(NPE)$ Training surrogate Hi-Fi only model: $O(N\log(N))$ Training additive correction (μ_D): $O(NPE)$ Training KDE: $O(N^2E)$ Training final model: $O(NPE)$	$O(P)$
Dehghani NN	Training student model (Step 1): $O(NPE)$ Training teacher model (Step 2): $O(N^3E)$ Training student model (Step 3): $O(NPE+N^2)$	$O(P)$
Perdikaris GP	Training surrogate Lo-Fi only model (Step 1): $O(N^3E)$ Training final model (Step 2): $O(N^3E)$	$O(N^2)$
Raissi GP	Training model: $O(N^3E+NPE)$	$O(N^2)$

The complexities in Table 6.3 are generalised to any dataset, number of epochs and parameter sizes. It can be seen that any step within the Kernel-based approach has a lower cost of training than any of the other methods which require the use of a GP. This is because GPs require the tuning of hyper-parameters and the repeated inversion of an updated kernel matrix which has minimum complexity $O(N^3E)$. At inference, the Kernel-based approach offers an advantage over the GP-based methods since the only evaluation required is a forward pass. The Dehghani NN also holds this advantage; however, does not have uncertainty quantification in its outputs. Table 6.6 shows

the approximated calculated complexities based on the training parameters for each method.

Table 6.6: Cost at training and inference for the Kernel-based approach and baseline models. N is the number of training data points, P is the number of parameters in a neural network and E is the number of epochs.

Method	**Cost at training**	**Cost at inference**
Kernel-based approach	Training surrogate Lo-Fi only model: 2.40×10^8 Training surrogate Hi-Fi only model: 1.48×10^3 Training additive correction (μ_D): 1.30×10^8 Training KDE: 1.47×10^8 Training final model: 1.20×10^{10} **Total:** 1.25×10^{10}	1.20×10^2
Dehghani NN	Training student model (Step 1): 4.80×10^9 Training teacher model (Step 2): 7.96×10^{13} Training student model (Step 3): 7.40×10^9 **Total:** 7.96×10^{13}	9.60×10^1
Perdikaris GP	Training surrogate Lo-Fi only model (Step 1): 3.00×10^{12} Training final model (Step 2): 3.82×10^{12} **Total:** 6.82×10^{12}	2.38×10^6
Raissi GP	Training model: 1.83×10^{14} **Total:** 1.83×10^{14}	2.38×10^6

Table 6.6 shows that the GP-based methods clearly dominate in terms of cost of training. The other methods are at least 2 orders of magnitude greater than the Kernel-based approach. The inversion of the kernel matrix is the most expensive operation when fitting a GP that scales quickly in large data regimes. The cost of inference is also much smaller than the Perdikaris GP and Raissi GP methods, which again scale quickly in large data regimes.

6.8 Conclusions and future work

Chapter 4 introduced a novel MAP-based MF modelling approach with the capacity for mismatch uncertainty quantification. However, this approach could not easily place modelling assumptions in transitory regions of the domain. Furthermore, the VI-based training scheme was computationally expensive due to the large amounts of distributional parameters in the Bayesian problem formulation.

In this chapter, a novel Kernel-based MF modelling method was developed. As in Chapter

4, a Bayesian problem formulation was adopted which provided a principled framework for making trade-offs between relying on data or prior knowledge, particularly in regions of the domain with sparse data. The Bayesian problem formulation required fewer distributional parameters than in Chapter 4, allowing a MAP approximation without using the VI-based approximation. By reframing the sample size n_{Y_H} to be a function of a kernel density estimate, modelling assumptions can be placed more effectively in transitory regions of the model. A modelling process was then outlined. Applying the process to a 2-D synthetic example demonstrated how the MF model can ultimately do better than a Lo-Fi prediction with an additive learned correction or spline fit purely on the Hi-Fi data. Comparisons with GP-based approaches showed that current state of the art still achieves more accurate mean predictions over the Kernel-based approach; however, the Kernel-based approach offers large improvements in terms cost of training and inference over the alternative method which highlights the promise the novel approach has if the flaws are ironed out.

Future work will focus on developing robust hyper-parameter tuning methods with a more rigorous validation scheme. Additionally, further work is required to understand the large spikes in the variance of transitory regions of the domain in the 2-D problem. Furthermore, the scalability of the Kernel-based approach to the higher-dimensional problems will be explored in addition to alternative methods for density estimation suitable for higher dimensions. Also, with the use of the spline fit $\hat{\mu}(x)$ instead of Hi-Fi data points, aleatoric uncertainty on the Hi-Fi dataset was not captured in the implementation of this Bayesian framework. Therefore, future work would be to develop an implementation of the existing Bayesian framework which includes aleatoric uncertainty on its outputs in addition to mismatch uncertainty. This could potentially be achieved by learning the predictive variance on the Hi-Fi data separately and slightly augmenting the Bayesian framework to incorporate it. A more rigorous comparison with baseline models would be beneficial including the effect of changing different kernel functions, using different size datasets, different generating functions with different levels of mismatch. This would better draw out the limitations of the different methods, including how well each method can correct for model mismatch, and give insights on how to improve the novel approach.

Similar to the future work described in Chapter 4, a detailed explanation about how engineers might set the prior mismatch uncertainty for different use cases would make this approach more accessible. Finally, further experiments conducted on realistic engineering examples will help

further demonstrate the practicality and versatility of the approach.

Chapter 7

Conclusions and future work

In this chapter, an overview of the main advancements realised through the course of this thesis is provided in addition to potential future work that could be undertaken in its context as a whole. The conclusions directly address the aim and objectives of this thesis as outlined in Chapter 1 and Chapter 2. These are restated below.

7.1 Aim and objectives

7.1.1 Aim

1. Develop scalable modelling frameworks that produce trusted outputs that accommodate sparse data obtained from real engineering systems. Models produced using this framework will provide trust through uncertainty quantification, interpretability, and high-performance.

7.1.2 Objectives

1. Evaluate and select candidate modelling frameworks for further development, which can accommodate sparse data obtained from real engineering systems.

2. Development of scalable modelling frameworks that leverage multiple information sources using a principled and mathematically grounded approach.

3. Ensure that the models developed are trustworthy through uncertainty quantification of their outputs, having interpretability, and being high-performing.

7.2 Conclusions

In Chapter 3, Physics-informed Neural Networks (PINN) were trained to solve the 1-D heat equation for varying levels of model mismatch. The results of the experiments illustrated the detrimental effect that model mismatch can have on the performance of a PINNs model. A mitigation was explored in the form of applying weightings on the physics-loss term. The intent was to increase or reduce the influence that incorrectly specified governing equations had on the solution. Promisingly, the results showed reasonable effectiveness at low levels of mismatch; however, as the level of mismatch between the observational data and the governing equation grew larger, the weightings became less effective. Despite their promise, the PINNs methodology was not further evaluated for the reasons uncovered on its unsuitability to achieve the aim of this thesis. The application of PINNs typically requires well specified equations that represent the process fully. This includes the accommodation of the real world violation of many of the underpinning assumptions in Partial Differential Equations (PDEs) (e.g. assuming an adiabatic process, continuous fluidic behaviours or the closed-system conservation of energy). This is difficult to achieve given all the potential sources of complex interactions in real systems. Accepting the physics (e.g. PDEs) are structurally imperfect and that they cannot be parametrised, motivates their utility instead as a prior in the absence of noise free data. This inspired a pivot towards Multi-Fidelity (MF) frameworks. These methods have the capacity to leverage a wider range of sources of information, including observational data and clearly defined physics-models (e.g. PDEs), but also more complex physics-based models (e.g. simulation models) that cannot be defined by clearly specified equations. The promising results served as a motivation for the MF modelling approaches which relied on weighting schemes in later chapters. The insights from this experiment helped to fulfil objective 1 of this thesis.

Objectives 2 and 3 of this thesis were overall met by the contributions described in Chapters 4 and 6. Two novel Bayesian frameworks were developed that allowed engineers to trade-off between fidelities of datasets when making predictions. This was achieved in a mathematically principled manner based on prior engineering understanding of the problem. The frameworks were applied

to Mean Variance Estimation (MVE) Neural Networks (NNs), demonstrating their scalability for large volume applications that may be found in DT applications. These aspects of the work directly fulfilled objective 2.

Both frameworks had the capacity to quantify mismatch uncertainty providing more trustworthy outputs. Since the framework handles multiple data sources, it has the capacity to include data from physics-based models whose generating processes are clearly understood, therefore providing an avenue for increasing the interpretability of the model. The performance of the model trained for the experiments in Chapters 4 and 6 was evaluated via the expected behaviour after certain modelling choices had been made. In terms of expected behaviour, the models in Chapter 4 were high-performing. However, in Chapter 6 it was evident in the 2-D example that in transitory regions of the domain, the uncertainty was not accurately quantified. These aspects of the work directly fulfilled objective 3.

There were differences between the frameworks. The framework developed in Chapter 4 uniformly applied the same value of trust (or sample size) to every Hi-Fi data point. This is more consistent with traditional methods of Bayesian modelling where the influence of a data point would be uniform in a sample. In comparison to the Chapter 6 framework, where the trust was informed by a function of a (Kernel Density Estimator) KDE. The values of trust at a Hi-Fi data point could vary between $(0, 1)$, where 1 implies total trust in the Hi-Fi data and 0 implies no trust. Therefore, this notion is less consistent with typical Bayesian modelling. There are, however, benefits to the Chapter 6 approach. The Kernel-based approach allowed for a value of trust to be quantified in sparse regions of the domain where there was no Hi-Fi data, therefore enabling for modelling assumptions to be placed in transitory regions of the model. The current implementation of the framework in Chapter 6 does not include aleatoric uncertainty on its outputs in contrast to Chapter 4, and further work is required to address this. Additionally, the performance of the Kernel-based approach was worse in comparison to the GP based methods suggesting a better form of correction is required on the surrogate Lo-Fi model. Further experiments could have also been done in order to fully understand the capacity of the Kernel-based approach and baseline models in capturing various levels of mismatch.

Conclusions drawn from Chapter 5 warrants a separate discussion since it forms the basis of the other chapters. In Chapter 5, novel algorithms were developed in order to address convergence is-

sues that occur when training probabilistic NNs with the Gaussian Negative Log-Likelihood (NLL) loss function. Without these novel algorithms, the developed Bayesian frameworks would be inviable, especially in large-scale data regimes for which the methods were designed. More generally, the algorithms provide a means for more consistent convergence of probabilistic models, resulting in their higher-performance and improved uncertainty quantification. This was essential to satisfy objective 3 of this thesis. One of the novel algorithms demonstrated significant improvement over the state of the art in the presence of Weight Decay (WD). This is a known strategy for preventing over-fitting when training with the Gaussian NLL, thus further enabling this gradient annealing strategy. The novel algorithms were also hyper-parameter free, offering an advantage over the state of the art algorithms that required hyper-parameter tuning.

7.3 Future work

In both of the developed modelling frameworks developed in Chapters 4 and 6, guidance for setting the prior value on the mismatch uncertainty would aid engineers in their predictive modelling tasks. Currently, this is left to an engineer's intuition as to how that value is set. One approach is applying existing Gaussian Process (GP)-based approaches for MF modelling using the predictive uncertainty away in regions of data sparsity in order to provide an estimate value. Residual analysis could also be used by observing the magnitudes of the discrepancies between Hi-Fi data and Lo-Fi data. The magnitudes could be used to inform the prior mismatch uncertainty. Another aspect of future work pertaining to both frameworks is the need to evaluate the models against state of the art methods. The foreseen challenges include building comparative evaluation metrics to account for the fact that different modelling approaches may quantify a different type of uncertainty. Finally, there may be benefits to merging PINNs with the MF models described in this thesis [108, 84]. This could enforce additional constraints on the surrogate models used in the implementation, which can aid in interpretability and therefore further increase trust in the models.

A limitation of the Chapter 6 implementation of the framework was the inability to quantify aleatoric uncertainty on the Hi-Fi data predictions. Amini et al. [46] developed a method for deterministically quantifying the epistemic and aleatoric uncertainty when performing predictive modelling. One possible avenue is to adapt this approach such that the epistemic uncertainty quan-

tification also encompasses mismatch uncertainty, and the aleatoric uncertainty is then modelled using the original process described in the paper. They similarly use feed-forward NNs in their implementation; therefore, there is scope to adapt their models.

The future work for Chapter 5 includes reducing the computational complexity of the algorithms. This could be achieved via implementing the algorithm periodically rather than on every training iteration; it would be beneficial to users to know how often to implement the algorithm for different problem types. Furthermore, the recasting of the Gaussian NLL loss function into a multi-objective loss brings scope to adapt and implement other multi-objective loss balancing algorithms that exist in the literature. Finally, the algorithms need to be tested on wider range of models and problem types to fully demonstrate the versatility of the algorithms.

7.4 Final remarks

Overall, the work presented in this thesis advances the field of PIML in the application of large-scale engineering problems. Here, we transitioned from rigid physics-constrained methods to more flexible MF methods which enable a more pragmatic pathway for fusing sparse observational data with more complex physics-based models rooted in a mathematical framework. Several novel contributions, including gradient annealing algorithms that ensure consistent convergence for a wide range of probabilistic models, to the Bayesian frameworks that quantify mismatch uncertainty, all collectively build a foundation for building models that are trustworthy.

The methods developed in this thesis make significant progress towards the integration of ML into DTs applications. The demand for DTs and data-driven decision making is ever increasing, and therefore, the need for interpretable models that can communicate their uncertainty about different information sources will become of greater importance. The contributions of this thesis will serve as a platform for researchers and engineers to harness the full potential of data-centric engineering.

Appendix A

Expanded derivation of Equation (4.5)

$$\mathcal{L}(q) = \int \prod_i q_i \left\{ \ln p\left(X, \Theta_{\text{All}}\right) - \sum_i \ln q_i \right\} d\Theta_{\text{All}} \tag{A.1}$$

$$= \underbrace{\left\{ \int \prod_i q_i \ln p\left(X, \Theta_{\text{All}}\right) d\Theta_{\text{All}} \right\}}_{\text{LHS}} - \underbrace{\left\{ \int \prod_i q_i \sum_i \ln q_i d\Theta_{\text{All}} \right\}}_{\text{RHS}} \tag{A.2}$$

In the LHS, we factor out q_j to obtain

$$\text{LHS} = \int q_j \left\{ \int \ln p\left(X, \Theta_{\text{All}}\right) \prod_{i \neq j} q_i d\theta_i \right\} d\theta_j \tag{A.3}$$

$$\tag{A.4}$$

We use the definition for $\ln \tilde{p}(X, Z)$ (Equations (4.10) and (4.11)) to obtain

$$\text{LHS} = \int q_j \ln \tilde{p}\left(X, \theta_j\right) d\theta_j \tag{A.5}$$

Now we turn attention to the RHS. We expand the product and the summation to obtain

$$\text{RHS} = \int \left(q_1 \cdot q_2 \cdots q_M\right) \left\{ \ln q_1 + \cdots \ln q_M \right\} d\Theta_{\text{All}} \tag{A.6}$$

Now expand term by term and consider the i-th term $\int q_1 \cdot q_2 \cdots q_M \ln q_i d\theta_1 d\theta_2 \cdots d\theta_M$. We group all the dependent terms together and obtain

$$\int q_1 q_2 \cdots q_M \ln q_i \, d\theta_1 \cdots d\theta_M = \int \ln q_i \left[\int q_1 d\theta_1 \int q_2 d\theta_2 \cdots \int q_M d\theta_M \right] d\theta_i \tag{A.7}$$

Each marginal integral $\int q_j d\theta_j = 1$, since q_j is a distribution. Therefore we obtain

$$\int q_i \ln q_i \, d\theta_i \tag{A.8}$$

So the RHS becomes

$$\text{RHS} = \sum_i \int q_i \ln q_i \, d\theta_i \tag{A.9}$$

Finally isolating the terms dependent on q_j and the non-dependent terms into 'const', the RHS becomes

$$\text{RHS} = \int q_i \ln q_i \, d\theta_i + \text{const} \tag{A.10}$$

Combining the final results for the LHS and RHS we obtain the final result as shown by Equation (4.9).

Appendix B

Training details, hyper-parameter choices and model architectures

Table B.1: Model parameters and learning parameters for every experiment.

Experiment	**Hidden Layers, Units and Activation functions**	**Training Scheme**	**Epochs**	**Optimiser**	**Lr**
Simple Mismatch	6 hidden layers 20 units each layer ELU activation functions	Pretraining using MSE loss	2000	Adam	1E-03
		MAP training	4000		
Linear and Non-Linear Mismatch		Pretraining using MSE loss	5000		
		MAP training	10000		
AGTF30 Model Mismatch		Pretraining using MSE loss	5000		
		MAP training	10000		

Table B.2: Hyper-parameter choices for MAP training.

Experiment	r	$\mu_D(x, \mu_{Y_L}(x); \theta_{\mu_D})$ **Function Type**	α_0	$\sigma^2_{Y_{H\,\text{prior}}}$	$\sigma^2_{Y_{H\,\text{upper}}}$
Constant Mismatch	0.1, 0.2, 0.3, 0.5, 1, 2, 3, 5, 10, 20, 30, 50, 100]	$Ax+B$	1	2	0.05
Linear and Non-Linear Mismatch	50	All functions are in Table with varying degrees.	1	1	0.05
AGTF30 Model Mismatch	20	$Ax^3+Bx^2+Cx+D\mu^3_{Y_L}(x;\theta_{\mu_{Y_L}})$ $+E\mu^2_{Y_L}(x;\theta_{\mu_{Y_L}})+F\mu_{Y_L}(x;\theta_{\mu_{Y_L}})+G$	1	0.1	0.05

Table B.3: Overview of number of training data points and domain for each experiment.

Experiment	**Hi-Fi Training Data Details**	**Lo-Fi Training Data Details**
Constant Mismatch	500 points uniformly sampled across $x \in [0, 0.5]$with additive Gaussian noise($\mathcal{N}(0, 0.1)$)	500 points at same domain points as Hi-Fi data with additive Gaussian noise ($\mathcal{N}(0, 0.01)$). 500 points linearly spaced across $x \in [0.5, 1]$ with additive Gaussian noise ($\mathcal{N}(0, 0.01)$) .
Linear and Non-Linear Mismatch	200 points uniformly sampled across $x \in [0, 0.4]$ with additive Gaussian noise($\mathcal{N}(0, 0.1)$) 100 points uniformly sampled across $x \in [0.6, 0.8]$with additive Gaussian noise($\mathcal{N}(0, 0.1)$)	300 points at same domain points as Hi-Fi data with additive Gaussian noise ($\mathcal{N}(0, 0.01)$). 100 points linearly spaced across $x \in [0.4, 0.6]$ with additive Gaussian noise ($\mathcal{N}(0, 0.1)$). 100 points linearly spaced across$x \in [0.8, 1]$ with additive Gaussian noise ($\mathcal{N}(0, 0.01)$)
AGTF30 Model Mismatch	40 points uniformly sampled in each of $w_f \in [0.29, 0.40]$, $w_f \in [0.44, 0.54]$, $w_f \in [0.79, 0.94]$ $w_f \in [1.09, 1.25]$.	160 points at the same domain points as Hi-Fi data. 40 points uniformly sampled in each of $w_f \in [0.40, 0.44]$, $w_f \in [0.54, 0.79]$, $w_f \in [0.94, 1.09]$ $w_f \in [1.25, 1.4]$

Table B.4: Overview of number of test data points and domain for each experiment.

Experiment	**Test Data Details**
Constant Mismatch	1000 points linearly spaced across $x \in [0,1]$.
Linear and Non-Linear Mismatch	1000 points linearly spaced across $x \in [0,1]$ for entire domain evaluation. 1000 points linearly spaced across $x \in [0,0.4] \cup [0.6,0.8]$ for data evaluation. 1000 points linearly spaced across $x \in [0.4,0.6] \cup [0.8,1]$ for data evaluation.
AGTF30 Model Mismatch	400 points linearly spaced across $w_f \in [0.29,1.40]$.

Appendix C

Annealing Y_μ and Y_{σ^2} (I) algorithm for solving convergence issues

The motivation for the adapted algorithm used in Chapter 6 comes from an analysis of the gradients of the NL loss, Equation (6.8), w.r.t μ_{Y_H} and $\sigma^2_{Y_H}$. Substituting into the functional forms of the hyper-parameters yields Equations (C.1) and (C.2).

$$\frac{\partial \mathcal{L}}{\partial \mu_{Y_H}} = \frac{4(\alpha_0+1)(\alpha_0+1)f(\mu_{Y_H}-\hat{\mu}_{Y_H})+(1-f)\sigma^2_{Y_H}\lfloor\sigma^2_{Y_H}\rfloor(\mu_{Y_H}-(\mu_{Y_L}-\mu_{Y_D}))}{(1-f)\sigma^2_{Y_H}(4(\alpha_0+1))} \tag{C.1}$$

$$\frac{\partial \mathcal{L}}{\partial \sigma^2_{Y_H}} = \frac{((\alpha_0+1)+(1-f)(\alpha_0+1))\sigma^2_{Y_H}-((\alpha_0+1)f(\mu_{Y_H}-\hat{\mu}_{Y_H})^2+\beta_0(1-f))}{(1-f)(\sigma^2_{Y_H})^2} \tag{C.2}$$

The denominator terms in Equations (C.1) and (C.2) are $\frac{1}{(1-f)\sigma^2_{Y_H}(4(\alpha_0+1))}$ and $\frac{1}{(1-f)(\sigma^2_{Y_H})^2}$. This is the common factor on the prior and data terms in each of the equations and are used to adapt $\boldsymbol{\sigma}^2_{*_\mu}$ and $\boldsymbol{\sigma}^2_{*_{\sigma^2}}$ in Algorithm 5. The losses $\mathcal{L}_{S_{Y_\mu}}$, $\mathcal{L}_{L_{Y_\mu}}$, $\mathcal{L}_{S_{Y_{\sigma^2}}}$ and $\mathcal{L}_{L_{Y_{\sigma^2}}}$ now correspond to training data points with large and small gradients, specific to the negative log described by Equation (6.8). The adaptation is clearly shown in Algorithm 8. Model's trained using the adapted algorithm performed better than without it, or using an adapted version of $\beta-$NLL. The adapted version of $\beta-$NLL was simply a multiplication of Equation (6.8) with $\left[(1-f)\lfloor\sigma^2_{Y_H}\rfloor(4(\alpha_0+1))\right]^\beta$ with $\beta=1$.

Algorithm 8 Annealing Y_μ and Y_σ^2 (Single Loss) (I) for vector-valued approximate posterior NLL

Require: Learning rate ω

Require: Dataset $\mathcal{D}$ of mini-batches

Require: Pointwise approximate posterior NLL loss $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}^d$

Require: Tuned hyper-parameters Σ, η, τ, α_0 and β_0.

Require: Trained or fitted models $f(x;\Sigma,\eta,\tau)$, $\mu_{Y_H}(x)$ and $\hat{\mu}_{Y_H}(x)$.

Initialise parameters $\theta = \theta_{Y_\mu} \cup \theta_{Y_{\sigma^2}}$, $\lambda_{Y_\mu} = 1$, $\lambda_{Y_{\sigma^2}} = 1$

Set momentum hyper-parameter $\gamma = 0.9$

while not converged **do**

  for all $\mathbf{x}_{\text{batch}} \in \mathcal{D}$ **do**

    Compute vector variances $\boldsymbol{\sigma}^2 = \sigma^2_{Y_H}(\mathbf{x}_{\text{batch}}) \in \mathbb{R}^d$

    Compute $\boldsymbol{f} = f(\mathbf{x}_{\text{batch}}) \in \mathbb{R}^d$

    Compute:

$$\sigma^2_{\min_\mu} = \min_i(\boldsymbol{\sigma}^2 \odot 4(\alpha_0+1)(1-\boldsymbol{f}))_i, \quad \sigma^2_{\max_\mu} = \max_i(\boldsymbol{\sigma}^2 \odot 4(\alpha_0+1)(1-\boldsymbol{f}))_i$$

$$\sigma^2_{\min_{\sigma^2}} = \min_i(\boldsymbol{\sigma}^2 \odot \boldsymbol{\sigma}^2 \odot (1-\boldsymbol{f}))_i, \quad \sigma^2_{\max_{\sigma^2}} = \max_i(\boldsymbol{\sigma}^2 \odot \boldsymbol{\sigma}^2 \odot (1-\boldsymbol{f}))_i$$

    ,

    Scale variances elementwise:

$$\boldsymbol{\sigma}^2_{*_{Y_\mu}} = \frac{(\boldsymbol{\sigma}^2 \odot 4(\alpha_0+1)(1-\boldsymbol{f})) - \sigma^2_{\min_\mu}}{\sigma^2_{\max_\mu} - \sigma^2_{\min_\mu}}, \quad \boldsymbol{\sigma}^2_{*_{Y_{\sigma^2}}} = \frac{(\boldsymbol{\sigma}^2 \odot \boldsymbol{\sigma}^2 \odot (1-\boldsymbol{f})) - \sigma^2_{\min_{\sigma^2}}}{\sigma^2_{\max_{\sigma^2}} - \sigma^2_{\min_{\sigma^2}}}$$

    Compute pointwise loss vector $\mathcal{L} = \mathcal{L}(\mathbf{x}_{\text{batch}})$

    Define weighted vector losses:

$$\mathcal{L}_{S_{Y_\mu}} = \boldsymbol{\sigma}^2_{*_{Y_\mu}} \odot \mathcal{L}, \quad \mathcal{L}_{L_{Y_\mu}} = (1-\boldsymbol{\sigma}^2_{*_{Y_\mu}}) \odot \mathcal{L}$$

$$\mathcal{L}_{S_{Y_{\sigma^2}}} = \boldsymbol{\sigma}^2_{*_{Y_{\sigma^2}}} \odot \mathcal{L}, \quad \mathcal{L}_{L_{Y_{\sigma^2}}} = (1-\boldsymbol{\sigma}^2_{*_{Y_{\sigma^2}}}) \odot \mathcal{L}$$

$$\mathcal{L}_{\lambda_{Y_\mu}} = (\lambda_{Y_\mu}(1-\boldsymbol{\sigma}^2_{*_{Y_\mu}}) + \boldsymbol{\sigma}^2_{*_{Y_\mu}}) \odot \mathcal{L}$$

$$\mathcal{L}_{\lambda_{Y_{\sigma^2}}} = (\lambda_{Y_{\sigma^2}}(1-\boldsymbol{\sigma}^2_{*_{Y_{\sigma^2}}}) + \boldsymbol{\sigma}^2_{*_{Y_{\sigma^2}}}) \odot \mathcal{L}$$

    Compute reduced losses:

$$\bar{\mathcal{L}}_{L_{Y_\mu}} = \frac{1}{d}\sum_{i=1}^d \mathcal{L}_{L_{Y_\mu},i}, \quad \bar{\mathcal{L}}_{S_{Y_\mu}} = \frac{1}{d}\sum_{i=1}^d \mathcal{L}_{S_{Y_\mu},i}, \quad \bar{\mathcal{L}}_{\lambda_{Y_\mu}} = \frac{1}{d}\sum_i \mathcal{L}_{\lambda_{Y_\mu},i}$$

$$\bar{\mathcal{L}}_{L_{Y_{\sigma^2}}} = \frac{1}{d}\sum_{i=1}^d \mathcal{L}_{L_{Y_{\sigma^2}},i}, \quad \bar{\mathcal{L}}_{S_{Y_{\sigma^2}}} = \frac{1}{d}\sum_{i=1}^d \mathcal{L}_{S_{Y_{\sigma^2}},i}, \quad \bar{\mathcal{L}}_{\lambda_{Y_{\sigma^2}}} = \frac{1}{d}\sum_i \mathcal{L}_{\lambda_{Y_{\sigma^2}},i}$$

    Compute gradient statistics:

$$g^{Y_\mu}_{\max} = \max\left\|\nabla_{\theta_{Y_\mu}} \bar{\mathcal{L}}_{L_{Y_\mu}}\right\|, \quad g^{Y_\mu}_{\text{mean}} = \left\|\overline{\nabla_{\theta_{Y_\mu}} \bar{\mathcal{L}}_{S_{Y_\mu}}}\right\|$$

$$g^{Y_{\sigma^2}}_{\max} = \max\left\|\nabla_{\theta_{Y_{\sigma^2}}} \bar{\mathcal{L}}_{L_{Y_{\sigma^2}}}\right\|, \quad g^{Y_{\sigma^2}}_{\text{mean}} = \left\|\overline{\nabla_{\theta_{Y_{\sigma^2}}} \bar{\mathcal{L}}_{S_{Y_{\sigma^2}}}}\right\|$$

    Update annealing coefficients:

$$\hat{\lambda}_{Y_\mu} = \frac{g^{Y_\mu}_{\max}}{g^{Y_\mu}_{\text{mean}}}, \quad \hat{\lambda}_{Y_{\sigma^2}} = \frac{g^{Y_{\sigma^2}}_{\max}}{g^{Y_{\sigma^2}}_{\text{mean}}}$$

$$\lambda_{Y_\mu} \leftarrow (1-\gamma)\lambda_{Y_\mu} + \gamma\hat{\lambda}_{Y_\mu}, \quad \lambda_{Y_{\sigma^2}} \leftarrow (1-\gamma)\lambda_{Y_{\sigma^2}} + \gamma\hat{\lambda}_{Y_{\sigma^2}}$$

    Update θ by gradient descent on $\bar{\mathcal{L}}_{\lambda_{Y_\mu}} + \bar{\mathcal{L}}_{\lambda_{Y_{\sigma^2}}}$

  end for

end while

REFERENCES

[1] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[2] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity. 1943," *Bull Math Biol*, vol. 52, no. 1-2, pp. 99–115, 1990.

[3] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks,"

[4] D. R. V. L. B. Thambawita, R. Ragel, and D. Elkaduwe, "To use or not to use: Graphics processing units (GPUs) for pattern matching algorithms," 2014.

[5] R. De Man, G. J. Gang, X. Li, and G. Wang, "Comparison of deep learning and human observer performance for detection and characterization of simulated lesions," *Journal of Medical Imaging*, vol. 6, p. 1, 6 2019.

[6] O. ONeill, "Linking Trust to Trustworthiness," *International Journal of Philosophical Studies*, vol. 26, no. 2, pp. 293–300, 2018.

[7] M. Ryan, "In AI We Trust: Ethics, Artificial Intelligence, and Reliability," *Sci Eng Ethics*, vol. 26, no. 5, pp. 2749–2767, 2020.

[8] S. Lockey, N. Gillespie, D. Holm, and I. A. Someh, "A Review of Trust in Artificial Intelligence: Challenges, Vulnerabilities and Future Directions," Hawaii International Conference on System Sciences.

[9] A. C. Bundy J; Ghahramani Z; Reid N; Weller A; McCarthy N; Montgomery J., "Explainable AI: the basic," tech. rep., 2019.

[10] J. Deuschel, A. Foltyn, K. Roscher, and S. Scheele, "The Role of Uncertainty Quantification for Trustworthy AI," in *Unlocking Artificial Intelligence*, pp. 95–115, Cham: Springer Nature Switzerland, 2024.

[11] A. Thuy and D. F. Benoit, "Explainability through uncertainty: Trustworthy decision-making with neural networks," *European Journal of Operational Research*, vol. 317, pp. 330–340, 9 2024.

[12] H. Felzmann, E. Fosch-Villaronga, C. Lutz, and A. Tamò-Larrieux, "Towards Transparency by Design for Artificial Intelligence," *Science and Engineering Ethics*, vol. 26, pp. 3333–3361, 12 2020.

[13] High-Level Expert Group on Artificial Intelligence, "Ethics Guidelines For Trustworthy AI," tech. rep., European Commmision, Brussels, 4 2019.

[14] Organisation for Economic Co-operation and Development, "Recommendation of the Council on Artificial Intelligence," tech. rep., OECD, 2022.

[15] R. Hamon, H. Junklewitz, I. Sanchez, and European Commission. Joint Research Centre., "Robustness and Explainability of Artificial Intelligence,"

[16] A. H. Briggs, M. C. Weinstein, E. A. Fenwick, J. Karnon, M. J. Sculpher, and A. D. Paltiel, "Model Parameter Estimation and Uncertainty: A Report of the ISPOR-SMDM Modeling Good Research Practices Task Force-6," *Value in Health*, vol. 15, pp. 835–842, 9 2012.

[17] R. Tomsett, D. Braines, D. Harborne, A. Preece, and S. Chakraborty, "Interpretable to Whom? A Role-based Model for Analyzing Interpretable Machine Learning Systems," 6 2018.

[18] D. Gunning, M. Stefik, J. Choi, T. Miller, S. Stumpf, and G. Z. Yang, "XAI-Explainable artificial intelligence," *Science Robotics*, vol. 4, 12 2019.

[19] G. Vilone and L. Longo, "Notions of explainability and evaluation approaches for explainable artificial intelligence," *Information Fusion*, vol. 76, pp. 89–106, 12 2021.

[20] L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal, "Explaining explanations: An overview of interpretability of machine learning," *Proceedings - 2018 IEEE 5th International Conference on Data Science and Advanced Analytics, DSAA 2018*, pp. 80–89, 1 2019.

[21] C. Rudin, C. Chen, Z. Chen, H. Huang, L. Semenova, and C. Zhong, "Interpretable machine learning: Fundamental principles and 10 grand challenges," *https://doi.org/10.1214/21-SS133*, vol. 16, pp. 1–85, 1 2022.

[22] K. Leino, Z. Wang, and M. Fredrikson, "Globally-Robust Neural Networks," in *Proceedings of the 38th International Conference on Machine Learning*, pp. 6212–6222, PMLR, 7 2021.

[23] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards Deep Learning Models Resistant to Adversarial Attacks," 2017.

[24] E. Hüllermeier and W. Waegeman, "Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods," *Machine Learning*, vol. 110, pp. 457–506, 3 2021.

[25] A. Malinin and M. Gales, "Predictive uncertainty estimation via prior networks," in *Advances in Neural Information Processing Systems*, vol. 2018-December, 2018.

[26] D. Jones, C. Snider, A. Nassehi, J. Yon, and B. Hicks, "Characterising the Digital Twin: A systematic literature review," *CIRP Journal of Manufacturing Science and Technology*, vol. 29, pp. 36–52, 5 2020.

[27] M. Xiong and H. Wang, "Digital twin applications in aviation industry: A review," *International Journal of Advanced Manufacturing Technology*, vol. 121, pp. 5677–5692, 8 2022.

[28] M. Attaran and B. G. Celik, "Digital Twin: Benefits, use cases, challenges, and opportunities," *Decision Analytics Journal*, vol. 6, p. 100165, 3 2023.

[29] A. Hartwell, F. Montana, W. Jacobs, V. Kadirkamanathan, N. Ameri, and A. R. Mills, "Distributed digital twins for health monitoring: resource constrained aero-engine fleet management," *The Aeronautical Journal*, vol. 128, pp. 1556–1575, 7 2024.

[30] J. Zhang, Y. Wang, B. Jiang, H. He, S. Huang, C. Wang, Y. Zhang, X. Han, D. Guo, G. He, and M. Ouyang, "Realistic fault detection of li-ion battery via dynamical deep learning," *Nature Communications 2023 14:1*, vol. 14, pp. 1–8, 9 2023.

[31] V. Zaccaria, M. Stenfelt, I. Aslanidou, and K. G. Kyprianidis, "Fleet Monitoring and Diagnostics Framework Based on Digital Twin of Aero-Engines," *Proceedings of the ASME Turbo Expo*, vol. 6, 8 2018.

[32] D. J. Wagg, K. Worden, R. J. Barthorpe, and P. Gardner, "Digital Twins: State-of-the-Art and Future Directions for Modeling and Simulation in Engineering Dynamics Applications," *ASCE-ASME Journal of Risk and Uncertainty in Engineering Systems, Part B: Mechanical Engineering*, vol. 6, 9 2020.

[33] D. Nix and A. Weigend, "Estimating the mean and variance of the target probability distribution," in *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, pp. 55–60, IEEE, 1994.

[34] N. S. Detlefsen, M. Jørgensen, and S. Hauberg, "Reliable training and estimation of variance networks," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[35] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski, "DeepAR: Probabilistic forecasting with autoregressive recurrent networks," *International Journal of Forecasting*, vol. 36, no. 3, 2020.

[36] Y. Chen, Y. Kang, Y. Chen, and Z. Wang, "Probabilistic forecasting with temporal convolutional neural network," *Neurocomputing*, vol. 399, pp. 491–501, 7 2020.

[37] R. L. Russell and C. Reale, "Multivariate Uncertainty in Deep Learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, pp. 7937–7943, 12 2022.

[38] D. Salinas Naverlabs, M. Bohlke-Schneider, L. Callot, R. Medico, and J. Gasthaus, "High-dimensional multivariate forecasting with low-rank Gaussian Copula Processes," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[39] M. Seitzer, A. Tavakoli, D. Antic, and G. Martius, "On the Pitfalls of Heteroscedastic Uncertainty Estimation with Probabilistic Neural Networks," 3 2022.

[40] L. Sluijterman, E. Cator, and T. Heskes, "Optimal training of Mean Variance Estimation neural networks," *Neurocomputing*, vol. 597, p. 127929, 9 2024.

[41] Radford M. Neal, *Bayesian Learning for Neural Networks*, vol. 118. Springer, 1996.

[42] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural networks," in *32nd International Conference on Machine Learning, ICML 2015*, vol. 2, 2015.

[43] B. Lakshminarayanan, A. Pritzel, and C. B. Deepmind, "Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles,"

[44] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning," 6 2015.

[45] A. Malinin, B. Mlodozeniec, and M. Gales, "Ensemble Distribution Distillation," *8th International Conference on Learning Representations, ICLR 2020*, 4 2019.

[46] A. Amini, W. Schwarting, A. Soleimany, and D. Rus, "Deep evidential regression," in *Advances in Neural Information Processing Systems*, vol. 2020-December, 2020.

[47] B. Charpentier, O. Borchert, D. Zügner, S. Geisler, and S. Günnemann, "Natural Posterior Network: Deep Bayesian Uncertainty for Exponential Family Distributions," 3 2022.

[48] C. K. I. Williams and C. E. Rasmussen, *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[49] M. K. Titsias, "Variational Learning of Inducing Variables in Sparse Gaussian Processes," 4 2009.

[50] P. K. Diederik and B. Jimmy, "Adam: A Method for Stochastic Optimization," *arXiv.org*, 2017.

[51] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the Gradient by a Running Average of Its Recent Magnitude," 2012.

[52] J. Duchi, E. Hazan, and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," *Journal of machine learning research*, vol. 12, no. null, pp. 2121–2159, 2011.

[53] J. Lu, A. J. Liu, F. Dong, F. Gu, J. Gama, and G. Q. Zhang, "Learning under Concept Drift: A Review," *IEEE transactions on knowledge and data engineering*, vol. 31, no. 12, pp. 2346–2363, 2019.

[54] A. Y. Ng, "Feature selection, L1 vs. L2 regularization, and rotational invariance," 2004.

[55] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. F. Wang, and L. Yang, "Physics-informed machine learning," *Nature reviews physics*, vol. 3, no. 6, pp. 422–440, 2021.

[56] J. Darbon and T. W. Meng, "On some neural network architectures that can represent viscosity solutions of certain high dimensional Hamilton-Jacobi partial differential equations," *Journal of computational physics*, vol. 425, p. 109907, 2021.

[57] N. Thuerey, K. Weissenow, L. Prantl, and X. Y. Hu, "Deep Learning Methods for Reynolds-Averaged Navier-Stokes Simulations of Airfoil Flows," *AIAA journal*, vol. 58, no. 1, pp. 15–26, 2020.

[58] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of computational physics*, vol. 378, pp. 686–707, 2019.

[59] B. Peherstorfer, K. Willcox, and M. Gunzburger, "Survey of Multifidelity Methods in Uncertainty Propagation, Inference, and Optimization," *SIAM Review*, vol. 60, pp. 550–591, 1 2018.

[60] M. G. Fernández-Godino, "Review of multi-fidelity models," 9 2016.

[61] A. Karpatne, G. Atluri, J. H. Faghmous, M. Steinbach, A. Banerjee, A. Ganguly, S. Shekhar, N. Samatova, and V. Kumar, "Theory-guided data science: A new paradigm for scientific discovery from data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, pp. 2318–2331, 10 2017.

[62] S. Z. Cai, Z. C. Wang, S. F. Wang, P. Perdikaris, and G. E. M. Karniadakis, "Physics-Informed Neural Networks for Heat Transfer Problems," *Journal of Heat Transfer-Transactions of the Asme*, vol. 143, no. 6, 2021.

[63] I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Trans Neural Netw*, vol. 9, no. 5, pp. 987–1000, 1998.

[64] X. Jia, J. Willard, A. Karpatne, V. Tech, J. S. Read, J. A. Zwart, U. S. G. Survey, V. Kumar, . J. Willard, M. Steinbach, V. Kumar, . A. Karpatne, and M. Steinbach, "Physics-Guided Machine Learning for Scientific Discovery: An Application in Simulating Lake Temperature Profiles," *ACM/IMS Transactions on Data Science*, vol. 2, pp. 1–26, 5 2021.

[65] A. Güneş Baydin, B. A. Pearlmutter, A. Andreyevich Radul, and J. Mark Siskind, "Automatic differentiation in machine learning: A survey," *Journal of machine learning research*, vol. 18, pp. 1–43, 2018.

[66] M. Raissi, A. Yazdani, and G. E. Karniadakis, "Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations," *Science*, vol. 367, no. 6481, pp. 1026–1030, 2020.

[67] S. Wang, Y. Teng, and P. Perdikaris, "Understanding and Mitigating Gradient Flow Pathologies in Physics-Informed Neural Networks," *SIAM Journal on Scientific Computing*, vol. 43, pp. A3055–A3081, 1 2021.

[68] A. D. Jagtap and G. E. Karniadakis, "Extended Physics-informed Neural Networks (XPINNs): A Generalized Space-Time Domain Decomposition based Deep Learning Framework for Nonlinear Partial Differential Equations," *Communications in Computational Physics*, 2021.

[69] A. D. Jagtap, E. Kharazmi, and G. E. Karniadakis, "Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems," *Computer methods in applied mechanics and engineering*, vol. 365, no. C, p. 113028, 2020.

[70] X. H. Meng, Z. Li, D. K. Zhang, and G. E. Karniadakis, "PPINN: Parareal physics-informed neural network for time-dependent PDEs," *Computer methods in applied mechanics and engineering*, vol. 370, no. C, p. 113250, 2020.

[71] N. S. Keskar, J. Nocedal, P. T. P. Tang, D. Mudigere, and M. Smelyanskiy, "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima," *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, 9 2016.

[72] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, "DeepXDE: A Deep Learning Library for Solving Differential Equations," *https://doi.org/10.1137/19M1274067*, vol. 63, pp. 208–228, 2 2021.

[73] M. Ben, M. Andrew, and N.-M. Tarje, "Finite Basis Physics-Informed Neural Networks (FBPINNs): a scalable domain decomposition approach for solving differential equations," *arXiv.org*, 2021.

[74] L. Sun, H. Gao, S. Pan, and J. X. Wang, "Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data," *Computer Methods in Applied Mechanics and Engineering*, vol. 361, p. 112732, 4 2020.

[75] G. Gokhale, B. Claessens, and C. Develder, "Physics informed neural networks for control oriented thermal modeling of buildings," *Applied Energy*, vol. 314, p. 118852, 2022.

[76] G. S. Misyris, A. Venzke, and S. Chatzivasileiadis, "Physics-Informed Neural Networks for Power Systems," 2019.

[77] R. G. Nascimento and F. A. C. Viana, "Fleet Prognosis with Physics-informed Recurrent Neural Networks," in *Structural Health Monitoring 2019*, (Lancaster, PA), DEStech Publications, Inc., 11 2019.

[78] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, "Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators," *Nature Machine Intelligence 2021 3:3*, vol. 3, pp. 218–229, 3 2021.

[79] S. Wang, H. Wang, and P. Perdikaris, "Learning the solution operator of parametric partial differential equations with physics-informed DeepONets," *Science Advances*, vol. 7, pp. 8605–8634, 10 2021.

[80] Q. Z. He, D. Barajas-Solano, G. Tartakovsky, and A. M. Tartakovsky, "Physics-informed neural networks for multiphysics data assimilation with application to subsurface transport," *Advances in Water Resources*, vol. 141, p. 103610, 7 2020.

[81] P. Perdikaris, M. Raissi, A. Damianou, N. D. Lawrence, and G. E. Karniadakis, "Nonlinear information fusion algorithms for data-efficient multi-fidelity modelling," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 473, p. 20160751, 2 2017.

[82] M. Raissi and G. Karniadakis, "Deep Multi-fidelity Gaussian Processes,"

[83] K. Cutajar, M. Pullin, A. Damianou, N. Lawrence, and J. González, "Deep Gaussian Processes for Multi-fidelity Modeling," 3 2019.

[84] X. Meng and G. E. Karniadakis, "A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse PDE problems," *Journal of Computational Physics*, vol. 401, p. 109020, 1 2020.

[85] S. Chakraborty, "Transfer learning based multi-fidelity physics informed deep neural network," *Journal of Computational Physics*, vol. 426, p. 109942, 2 2021.

[86] X. Meng, H. Babaee, and G. E. Karniadakis, "Multi-fidelity Bayesian neural networks: Algorithms and applications," *Journal of Computational Physics*, vol. 438, p. 110361, 8 2021.

[87] M. Dehghani, A. Mehrjou, S. Gouws, J. Kamps, and B. Schölkopf, "Fidelity-Weighted Learning," 11 2017.

[88] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 114, pp. 3521–3526, 3 2017.

[89] C. Moya, S. Zhang, M. Yue, and G. Lin, "DeepONet-Grid-UQ: A Trustworthy Deep Operator Framework for Predicting the Power Grid's Post-Fault Trajectories," 2022.

[90] G. Lin, C. Moya, and Z. Zhang, "B-DeepONet: An enhanced Bayesian DeepONet for solving noisy parametric PDEs using accelerated replica exchange SGLD," *Journal of Computational Physics*, vol. 473, p. 111713, 1 2023.

[91] Z. Zou, X. Meng, and G. E. Karniadakis, "Correcting model misspecification in physics-informed neural networks (PINNs)," *Journal of Computational Physics*, vol. 505, p. 112918, 5 2024.

[92] B. Moseley, "So, what is a physics-informed neural network?," 8 2018.

[93] A. S. Krishnapriyan, A. Gholami, S. Zhe, R. M. Kirby, and M. W. Mahoney, "Characterizing possible failure modes in physics-informed neural networks," 2021.

[94] Jiří Lebl, "4.6: PDEs, Separation of Variables, and The Heat Equation," 6 2022.

[95] S. Mishra and R. Molinaro, "Estimates on the generalization error of physics-informed neural networks for approximating PDEs," *IMA journal of numerical analysis*, 2022.

[96] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics*. Berlin, Heidelberg: Springer-Verlag, 2006.

[97] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 3 2004.

[98] Y. Wu, B. Sicard, and S. A. Gadsden, "Physics-informed machine learning: A comprehensive review on applications in anomaly detection and condition monitoring," *Expert Systems with Applications*, vol. 255, p. 124678, 12 2024.

[99] R. Kurz and K. Brun, "Degradation in Gas Turbine Systems," *Journal of Engineering for Gas Turbines and Power*, vol. 123, pp. 70–77, 1 2001.

[100] J. L. Kratz, "The Advanced Geared Turbofan 30,000 lbf electrified (AGTF30-e): A Virtual Testbed for Electrified Aircraft Propulsion Research," in *AIAA AVIATION FORUM AND ASCEND 2024*, (Reston, Virginia), American Institute of Aeronautics and Astronautics, 7 2024.

[101] R. A. Adams and J. J. F. Fournier, *Sobolev spaces*. Academic Press, 2008.

[102] R. Bischof and M. A. Kraus, "Multi-Objective Loss Balancing for Physics-Informed Deep Learning," *Computer Methods in Applied Mechanics and Engineering*, vol. 439, p. 117914, 5 2025.

[103] Jimmy Ba and Bo Wang, "CSC413/2516 Lecture 2: Multilayer Perceptrons & Backpropagation," 2021.

[104] Z. Chen, V. Badrinarayanan, C.-Y. Lee, and A. Rabinovich, "GradNorm: Gradient Normalization for Adaptive Loss Balancing in Deep Multitask Networks," 2018.

[105] A. A. Heydari, C. A. Thompson, and A. Mehmood, "SoftAdapt: Techniques for Adaptive Loss Weighting of Neural Networks with Multi-Part Loss Functions," 12 2019.

[106] E. García-Portugués, *Notes for Nonparametric Statistics*. 6.12.1 ed., 2025.

[107] P. Alfeld, "A trivariate cloughtocher scheme for tetrahedral data," *Computer Aided Geometric Design*, vol. 1, pp. 169–181, 11 1984.

[108] D. Liu and Y. Wang, "Multi-Fidelity Physics-Constrained Neural Network and Its Application in Materials Modeling," *Journal of Mechanical Design*, vol. 141, 12 2019.