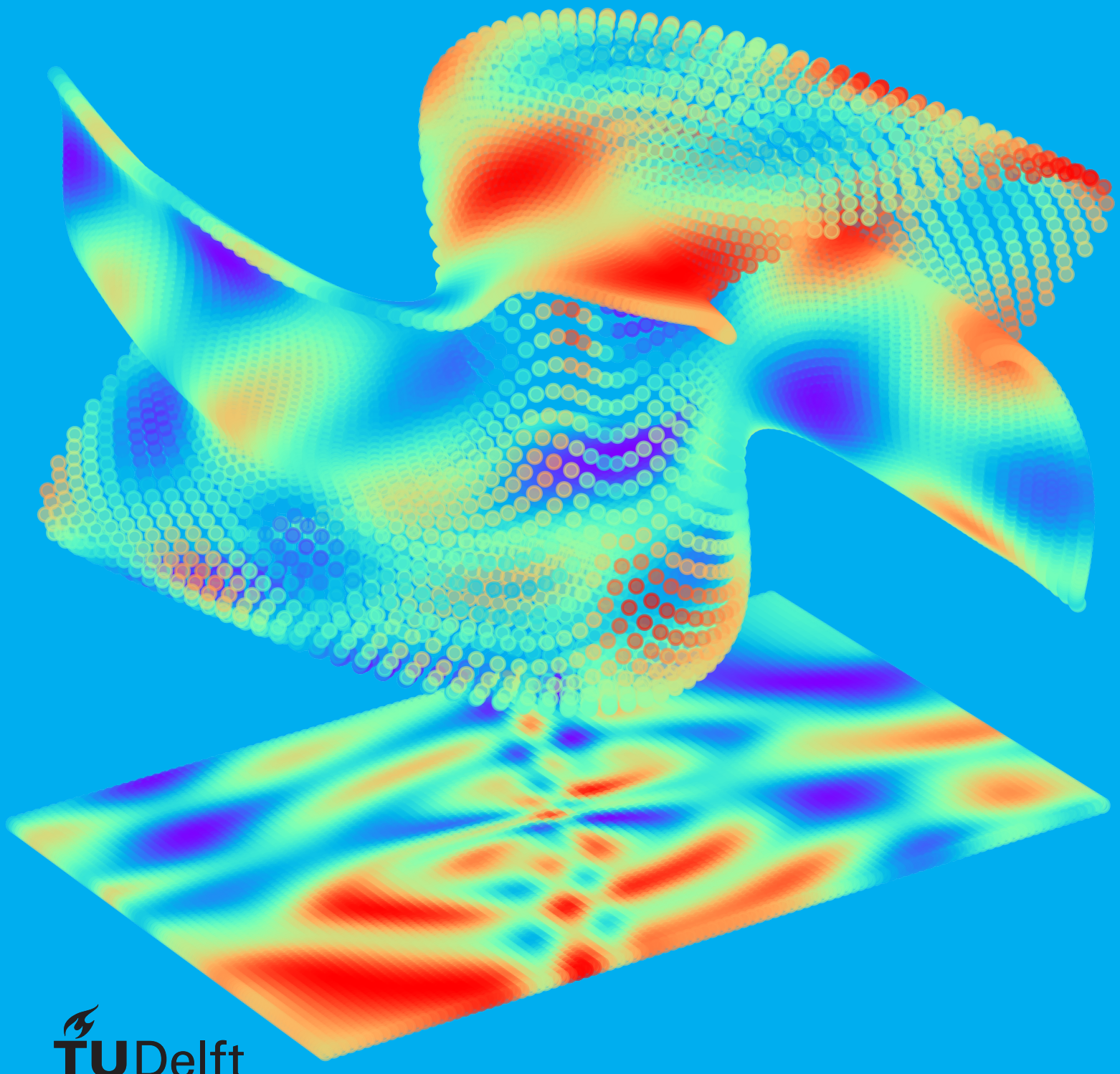
High-dimensional numerical optimization of fiber reinforced polymers with variational autoencoders and Bayesian optimization

Joep Storm



TU Delft

High-dimensional numerical optimization of fiber reinforced polymers with variational autoencoders and Bayesian optimization

by

Joep Storm

in partial fulfillment of the requirements for the degree of

Master of Science
in Civil Engineering

at the Delft University of Technology,
to be defended publicly on Monday Februari 22, 2021 at 15:00 PM.

Student number:	4677765	
Project duration:	March 10, 2020 – Februari 22, 2021	
Thesis committee:	Prof. dr. ir. L. J. Sluys,	TU Delft, Chair
	Dr. ir. I.B.C.M. Rocha,	TU Delft, Daily supervisor
	Dr. ir. T. Li Piani,	TU Delft, Supervisor
	Dr. ir. J. Bierkens,	TU Delft, Supervisor

TU Delft Delft University of Technology

Abstract

Designing engineering structures relies on accurate numerical simulations to predict the behaviour of a structure before its realization. In the design process, many variables influence the final structure. There is an incentive for optimizing the design based on the desire for cheaper structures and less material usage. Although a wide variety of optimization techniques exist, in practice many structures are not fully optimized and could fulfill their purpose with less material usage. This partly results from optimization techniques generally requiring many simulations for problems with a high number of variables. In creating increasingly complex models, simulating its performance can take up to days or weeks to compute, incentivizing the development of optimization methods that use as little simulations as possible.

The goal of this thesis is to provide a method of optimization which significantly reduces the required number of simulations to be performed, aiming to overcome the issue of required computational effort. The optimization problem used in this thesis is the geometric optimization of a fiber reinforced polymer (FRP) microstructure. Each evaluation requires a Finite Element analysis, and many parameters are required to describe the geometry. The goal is to find the maximum stress at perfect plasticity during uniaxial tension, interpreted as a measure of strength. While generally only fibers with a circular cross-section are used, here a morphing parameter is introduced that changes this shape between being circular and square. Furthermore, fibers are allowed to overlap, effectively creating a single fiber with a complex shape. This geometry optimization provides a challenging problem for which different optimization techniques can be compared.

The approach taken in attempting to minimize the number of function evaluations is a combination of several machine learning methods. Based on a limited number of initial samples, Bayesian optimization (BO) is applied. In BO a prediction model in the form of a Bayesian Neural Network (BNN) is created and used to inform further sampling. This prediction model provides a mean and a standard deviation in its prediction, both of which are used to find the best point to sample next. The number of initial samples required to create an accurate prediction model increases exponentially with the number of parameters. It is therefore opted to first use a variational autoencoder (VAE) to reduce the number of parameters in which BO is performed, by encoding the parameters in a lower dimensional representation. The variational autoencoder does this without requiring any function evaluation.

Results show that when encoding the original parameters using the VAE, the encoded representation is unable to recreate all possible configurations of the original parameters. Furthermore, by transforming to a lower dimensional representation, within this representation the complexity of a function increases compared to the original function, generally leading to many local optima. This complexity proves difficult for the BNN to accurately predict based on a limited number of samples. As a result, BO does optimize the result, but does not reliably find the global optimum of the reduced parameter space. No clear conclusion can be made on the overall performance of the method compared to alternative optimization methods. Recommendations are made for what additional studies could be performed. Further recommendations are given for solving issues limiting the current performance as well as possible additions to the framework.

The fiber geometry optimization serves as a good case to compare different optimization methods. It is shown that the geometry has a significant influence on the mechanical performance of a FRP microstructure, and optimization is therefore beneficial. The load case considered is however too specific for the optimized result to have direct practical use. Still, it does demonstrate that the results of an optimization study can be generalized. As the optimization framework is data driven, the optimization objective could easily be extended to study more realistic problems.

Acknowledgements

I would like to express my gratitude to the people who have helped me during my thesis:

First, I would like to thank my graduation committee, Bert sluys, Iuri Rocha, Tiziano Li Piani and Joris Bierkens, for giving me the opportunity to work on this topic. I am grateful for the constructive feedback during our meetings, and for giving me the trust and freedom to explore different ideas. The lessons I have learned will certainly be valuable throughout my career. I would like to emphasize my gratitude towards Iuri, whose invaluable feedback, ideas and support in our many meetings has improved the quality of every part of this work.

I would like to thank my friends that have supported me during this strange year and for being available for much-needed breaks. I am especially thankful towards my housemate Pim van der Honing who has kept me sane and had to put up with me spending many hours behind my desk.

I want to express my gratitude towards my family, for supporting me and showing interest even when I struggled to explain what exactly I was working on.

Thank you everyone who has helped me to complete my MSc degree at the TU Delft.

J. Storm
Delft, February 2021

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

A main responsibility of an engineer is to ensure the safety of a structure to be built. While testing scale models can give insight into the behaviour of a structure, this is seldom done anymore and instead numerical models are created which are based on material tests. In order to create these models, assumptions and simplifications have to be made. With finer and more complex models the number of assumptions and simplifications can be reduced, leading to more accurate representations. However, as models become more complex, the computational effort usually also drastically increases to the point where running a model can take up to days or weeks to compute. In trying to optimize a design, many trial iterations are needed before an optimum is reached. With a significant computation time per iteration this optimization procedure quickly becomes infeasible. This optimization is made even more challenging when problems consist out of a significant number of parameters. With each additional parameter the number of possible configurations increases exponentially. This thesis concerns itself with accelerating these optimization problems.

1.1. Optimization techniques

Structural design optimization consists in first defining a number of structural parameters to tweak (design variables). These n design variables are then iteratively changed until one or more performance indicators (objective function) are either minimized or maximized. In other words, optimization seeks to find an optimum in a n dimensional space, where each axis belongs to one design variable. This is the parameter space, with n being the dimensionality of the optimization problem. The type of optimization problems considered are those where the constraints of each parameter are independent of the values of other parameters. Several optimization techniques exist, the main branches considered in this thesis are presented in Figure 1.1 and explained below. An example of a 2-dimensional design space is presented in Figure 1.2a.

Gradient-based algorithms, such as the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm [1], can be very efficient in finding the optimal solution in convex problems. They however decrease in performance when the problem is high dimensional. At first, this high dimensionality does not appear to be a problem for gradient-based optimization. In practice, if local minima exist, there is a sampling-like procedure involved in gradient methods, since many starting points should be used, significantly increasing their required computational effort. To make matters worse, the number of local minima often increases for an increasing number of variables. In Figure 1.2b an example of a gradient-based method is presented.

In order to circumvent the issue with local minima, metaheuristic based methods can be used to optimize the global design space. Metaheuristic based methods are commonly used in solving high-dimensional non-convex optimization problems [2]. These methods include genetic algorithms [3] and particle swarm optimization [4]. These algorithms explore the complete design space by having iterative batches of samples, and are thus less influenced by local minima. These methods still require a significant number of function evaluations, making them inefficient for computationally expensive problems. An example of one of these batches for a GA is presented in Figure 1.2c.

In order to alleviate computational effort while still performing global optimization, Bayesian optimization can be used. Bayesian optimization is an optimization technique that creates a prediction model based on a limited number of samples. The prediction is used to determine the next point to efficiently optimize the design space. An example of this is presented in Figure 1.2d. The required data to form an accurate prediction scales with the number of dimensions, and therefore this method too requires many function evaluations, although significantly less compared to a GA.

The combination of the high-dimensionality of the optimization problem with the computationally expensive evaluations provides difficulties for all these existing techniques. In this thesis it is attempted

to split these two problems. The high-dimensionality is adressed using computationally cheap evaluations, and the expensive evaluations are only required in a low-dimensional space. This is done by reducing the input space using a variational autoencoder (VAE). In this reduced design space, Bayesian optimization is used to find its optimum. The core idea behind this split is that training the VAE does not require the objective function to be computed, and is therefore essentially free in computational time. Of course, the downside is that the feature space created by the VAE has no information of which regions in the original space are the most interesting for optimization. Therefore, this split naturally involves a tradeoff between efficiency and accuracy of the optimization procedure.

This is visualized in Figure 1.2e.

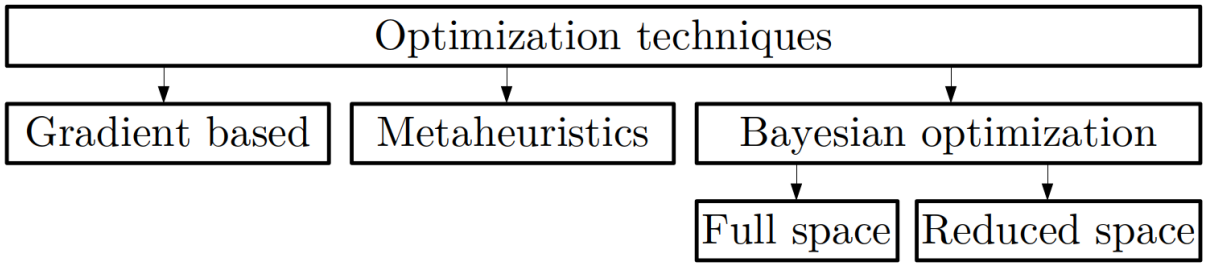


Figure 1.1: Main branches of optimization techniques considered in this thesis. Applying Bayesian optimization in a reduced design space is the novelty explored in this thesis.

All methods are implemented and applied to the practical scenario of finding the optimal geometry of fibers in a fiber reinforced polymer microstructure. The aim of this thesis is to compare these methods in their ability to find the optima with a small number of function evaluations. In complex problems it is unlikely that all methods converge to the global optimum, therefore they are also compared on their reliability of finding the optimum, or in other words the quality of their optimum. Furthermore all optimization techniques rely on some user defined parameters, or hyperparameters. Ideally, settings for these hyperparameters are used which provide similar results for all problems, and the outcome is not sensitive to small changes in them.

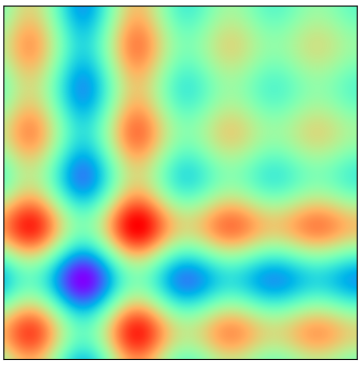
Research question

To guide the research a main research question is formulated, supported by several sub-questions.

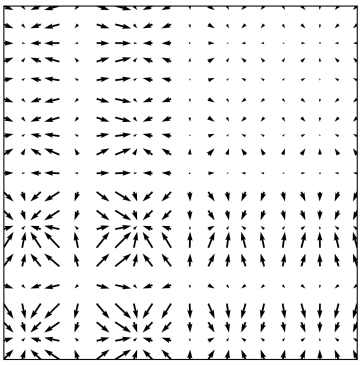
How does Bayesian optimization on a reduced design space compare to conventional optimization methods in the quality of the optima and the required computational effort?

Sub questions:

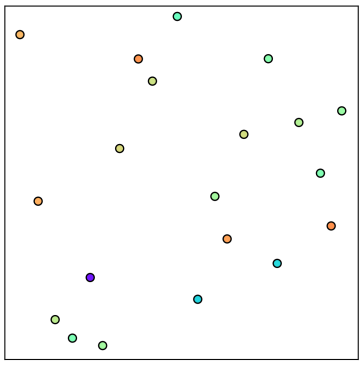
1. Do geometric properties of fibers in a fiber reinforced polymer microstructure influence mechanical properties?
2. Can the optimal solution be used to inform future engineering design decisions?
3. What are the costs and benefits of reducing the dimensionality of a design space for finding an optimal solution?
4. To which extent does Bayesian optimization reduce the computational effort in finding the optima within the reduced space?
5. Can a Bayesian neural network be effectively used for Bayesian optimization?
6. How do the hyperparameters and randomness in the initialization of methods influence the resulting optimum?



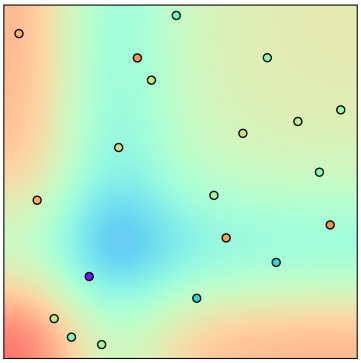
(a) Full 2-dimensional design space, with the two parameters representing the x and y axis, and the color representing the objective function. Brute-force computing all configurations in this way is very computationally expensive.



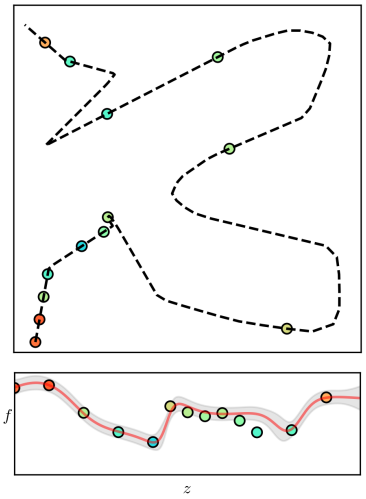
(b) Gradient based method. The gradients, plotted by arrows, display the direction to move in to maximize the objective. This demonstrates that for different initial starting points, different local optima can be obtained.



(c) Metaheuristic method, presented here is a genetic algorithm. Based on some initial sampling for which the objective is computed, a new batch of samples is created focussing on regions with high objective samples.



(d) Bayesian optimization on the full design space, based on a limited number of samples, a surrogate prediction model is trained. This prediction model is used to predict the optimum.



(e) Bayesian optimization in a reduced design space. The top figure represents the full 2D design space and the 'path' of the reduced 1D design space. In this 1D design space BO is applied, shown in the bottom figure.

Figure 1.2: Examples of different optimization techniques. The objective function is the Shubert 3 [5] function for 2 parameters. The objective is plotted as a color, with blue and red representing a low and high objective value respectively.

1.2. Research Methodology

The reduction of the dimensionality is implemented using a variational autoencoder (VAE), creating a non-linear transformation to a lower dimensional space. Bayesian optimization is used in this reduced space, where a different machine learning technique, namely a Bayesian Neural Network (BNN), creates a computationally cheap surrogate model. With the reduction from the VAE, training this prediction model should require significantly less initial samples compared to optimizing in the full space. Furthermore, due to the curse of dimensionality, it is a priori assumed that exploring this lower dimensional space will require significantly less objective function evaluations. This model is used to predict regions with significant potential of containing the optimum, allowing efficient sampling in those regions while ignoring low potential regions. Other studies with similar methods for optimization [6, 7] have studied cases where only a small subset of the original high-dimensional design space gives a valid solution. In contrast, here

the full design space is feasible.

The objective function, to be introduced in Chapter 2, is computed using the Jem/Jive C++ library [8]. An extension is developed in Python to allow optimization and control over the parameters. The variational autoencoder is implemented in Python, making use of the TensorFlow library [9]. The Bayesian neural network is developed using the Jem/Jive C++ library based on an existing neural network implementation [10]. Both for the use of maximizing the acquisition function in Bayesian optimization and as an alternative optimization method, a genetic algorithm is implemented in the Jem/Jive C++ library.

1.3. Thesis Outline

In this thesis various existing methods are combined to create a single optimization framework. Each method is introduced in a separate chapter, and when relevant includes a study of the literature. When applicable a parameter study is performed, the results of which are included at the end of each chapter to assist the reader in understanding the methods. The general outline of the thesis is provided in Figure 1.3. In Chapter 2, the optimization problem to be solved, namely a micromechanical structure, is introduced. Chapter 3 serves as an introduction to neural networks. This technique is the basis used in two methods, first in dimensionality reduction using a variational autoencoder in Chapter 4, followed by a Bayesian neural network in Chapter 5. In Chapter 6 the Bayesian optimization is detailed. A genetic algorithm is presented in Chapter 7, serving as an alternative optimization method. Chapter 8 combines all previous chapters for the full optimization of the problem and presents the results on optimizing the micromechanical structure. Finally, Chapter 9 summarizes the conclusions regarding the research questions and discusses recommendations for further research.

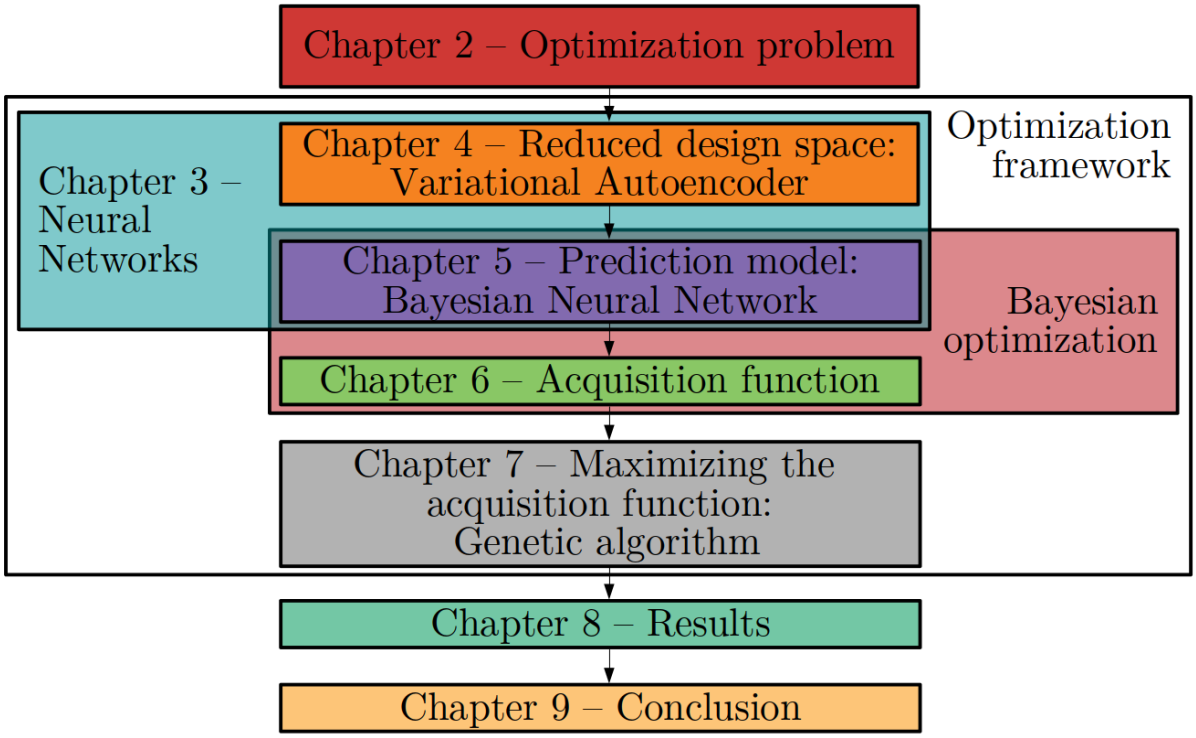


Figure 1.3: Outline of the thesis.

2 Optimization problem

Due to their high stiffness to weight ratio, composites are becoming increasingly popular in Civil Engineering (from strengthening existing structures to new bridges), are widely used in modern aircraft and are the current standard material for wind turbine blades. In these applications having accurate models and being able to optimize designs is essential. These numerical models for composites are chosen as the problem to be optimized in this thesis.

2.1. Fiber reinforced polymer optimization

Fiber reinforced polymers (FRP) are man-made materials that generally consist out of two main constituents. One constituent is the reinforcing phase, namely the fibers, which are embedded in the matrix phase. Both constituents come in different variations with various material properties which influence the complete behaviour. While many natural fibers exist, glass and carbon fibers with diameter of around 4-17 µm are widely used due to their good mechanical properties [11]. A common way of creating FRP structures is using laminates, as depicted in Figure 2.1.

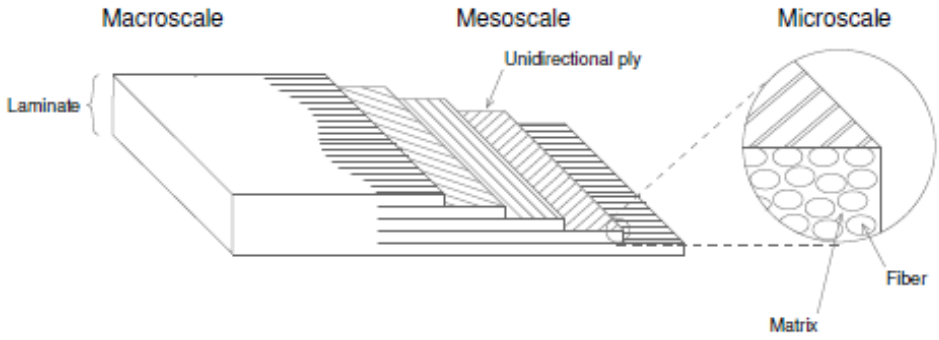


Figure 2.1: Layout of a standard laminated composite over several scales. Source: [12]

There are several properties of composites which depend on the microstructure and are of interest for optimization. These properties range from acoustics [13] and thermal [14] to mechanical properties [4, 14–18]. These properties directly influence the costs of the material, which can be included in the optimization [4, 19]. For these properties the material type [17], volume fraction [19], microscopic geometry [13, 18] and macroscopic shape [4] can be varied. Additive manufacturing [18, 20, 21] and automated fiber placement [20] are relatively new manufacturing methods with an increase of control of the geometry and shape of the material. These methods increasingly allow structures designed using for example topology optimization [14, 15, 22] to be manufactured. It was found that using the mechanical performance as objective provided sufficient complexity for the optimization function.

2.2. Geometry optimization

Requiring more material for a structure, or using materials with higher performance, generally increases the cost of a structure. Changing the microscopic geometry or macroscopic shape from a given set of material does not have this cost, apart from the cost of precision manufacturing. Optimizing the geometry and shape is therefore beneficial. Mechanical properties can be determined using a Finite Element Analysis (FEA) [23]. Where commonly homogenization techniques are used to derive a constitutive relation between stresses and strains in a FRP structure, research has been done, and still is, on creating

more advanced models where this relation is derived from an embedded microscopic model [24]. This embedded model should represent the behavior of a material point of a larger-scale model. This model is therefore known as a representative volume element (RVE). It is often created with a random distribution of fibers and forms a repeating pattern if one stacks multiple models next to each other.

In Figure 2.2 & 2.3 two microscopic fiber configurations are given with the only difference being their geometrical fiber positions. When subjected to a vertical load a FEA results in stress-strain curves as displayed in Figure 2.4. There is a 13.8% difference in the maximum obtained stress in this microstructure based solely on the positioning of the fibers. This suggests that even minor changes in the microscopic geometry can lead to improved macroscopic performance. This observation motivates the use of optimization techniques in order to further exploit this trend.

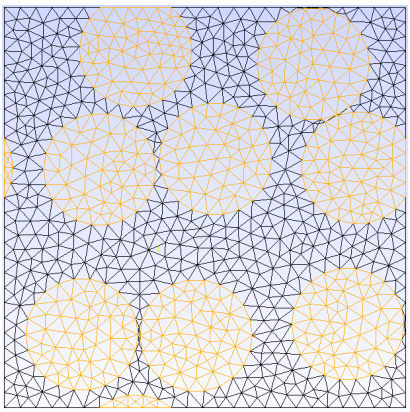


Figure 2.2: Example 1 of a mesh with 8 fibers.

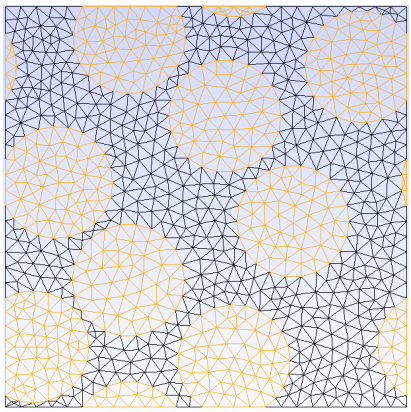


Figure 2.3: Example 2 of a mesh with 8 fibers.

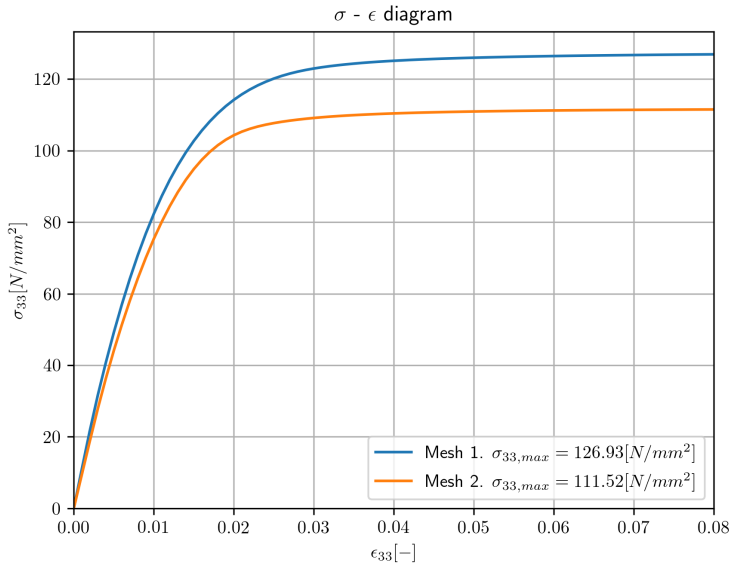


Figure 2.4: Stress strain curves of 2 geometrically different meshes with identical material properties and volume fraction.

A popular method of optimizing geometries is topology optimization (TO). In TO a FEA is used to evaluate a structure and to iteratively remove material in low-stress regions, and generally only optimizes the topology. To serve as RVE, the microstructure is generally periodic and continuous. The FRP microstructure also consists out of both the reinforcing and the matrix phase, providing further difficulties for topology optimization. Optimizing an RVE for any possible objective based on any number of design variables therefore requires significant extensions to TO. In this thesis an alternative approach is taken. The general optimization problem for maximizing mechanical performance, here represented by the

transverse stress σ_{33} at a strain of 0.1, can be defined as:

$$\begin{aligned} \max \quad & \sigma_{33} = FEA(x_1, x_2, ..., x_n) && \text{(2.1a)} \\ \text{subject to} \quad & x_1 \in X_1, && \text{(2.1b)} \\ & x_2 \in X_2, && \text{(2.1c)} \\ & ....., && \text{(2.1d)} \\ & x_n \in X_n, && \text{(2.1e)} \end{aligned}$$

Here $FEA(\cdot)$ is a Finite Element analysis (which does not provide derivative information) and X_i is the feasible space of the variable x_i. Examples of possible variables for x_i are the x & y coordinates of a fiber, the radius of a fiber (or all fibers) or fiber material properties. In this way the meshes assessed in Figures 2.2 & 2.3 would have 16 dimensions. If one discretizes each continuous variable into 100 possible values, this results in 100^{16} possible configurations.

To find the optimal geometry of the microstructure, one would ideally include all possible geometrical configurations of the microstructure. To allow this, the assumption of a circular cross section of the fibers can be relaxed. In this work, aside from the fiber positions, a parameter is introduced as the 'morphing' parameter, which transforms the shape of a fiber between being circular and square. Next to this, fibers are also allowed to 'overlap', creating a single fiber with a complex shape by combining these overlapping fibers. These two adaptations allow a wide variety of geometries when enough fibers are included. As a side effect of allowing the fibers to overlap, the boundaries of the fiber coordinates become independent. Current manufacturing techniques do not allow these complex fiber shapes. If an optimization technique finds a geometry that gives a significant gain in desired properties, it might stimulate the development of methods that do allow it. The performance indicator used is the stress at a fixed strain of 0.1 which is generally a point of perfect plasticity, interpreted as a measure of strength. This proved sufficient in demonstrating the optimization approach taken in this work. Maximizing this stress for a single load case is not particularly realistic from an engineering perspective, as generally multiple load cases apply. An extension could easily be made to optimize it to for example maximize the area of a failure envelope. Nevertheless, it should be investigated to which extent microscopic optimization transfers to macroscopic performance in real structures. For convenience when discussing the goal of maximizing the stress at a strain of 0.1, simply maximizing σ_{33} will be used, the indices of which are explained below.

2.3. Microscopic model

The micromechanical problem is evaluated by embedding the RVE in a single quadrilateral finite element. On this element a strain is incrementally applied using displacement control. In the coordinate system used the '1' direction is defined as being out of plane along the fibers, the '2' and '3' direction as the 'horizontal' and 'vertical' in plane directions respectively. '33' then refers to a plane in the '3' direction, with a load also in the '3' direction. As the load is applied to the top, the bottom is fixed in the '3' direction. The '2' direction is fixed in one side, the other side is left free such as not to induce stresses related to the poisson effect. In order to prevent rigid-body motions, all shear components are fixed.

The plasticity model by Melro *et al.* [25] is used with the plane strain assumption. the SkyLineLU non-linear solver is applied with numerical precision of $1e^{-10}$. The load is by default applied using strain increments of $0.005[-]$, up to a norm of $0.1[-]$. The fiber has a Young's modulus of $74000[N/mm^2]$. The matrix has a stiffness of $3760[N/mm^2]$, and a poisson ratio of $0.39[-]$.

A custom meshing script has been created in Python to translate geometrical properties into a mesh using gmsh [26] and its "OpenCASCADE" kernel. This kernel allowed the implementation of overlapping fibers. In Appendix A a more detailed overview of this implementation is given, followed by a short guide on how the script interacts with the OpenCASCADE functions.

3 Neural networks

Artificial neural networks (NN, or network) form the basis behind many types of machine learning algorithms. Here a general introduction is presented, before going into their specific use cases for the optimization problem. In Chapter 4 the first use case is presented, a variational autoencoder, creating a non-linear transformation from the high-dimensional design space to a lower dimensional subspace. This can be done purely based on the range of input possibilities without any FE simulation, this is known as unsupervised learning. This is followed by supervised learning in Chapter 5 where a Bayesian neural network makes predictions in this subspace based on a limited number of samples from a FE simulation. The applications used in this thesis are thus dimensionality reduction and a prediction model (regression). Other applications not used in this thesis, such as classification, are possible with a similar underlying structure.

3.1. Background

Artificial Intelligence is the concept of machines being able to carry out tasks in a way we consider to be intelligent. One subfield of AI is machine learning, which is based on the concept of giving the machine access to data and to let it learn patterns for itself. Within machine learning, using a NN is a way of applying this. A NN is a computational model which consists out of an interconnected group of nodes, with similarities to neurons in a brain, to create highly nonlinear relations and serve as universal approximators [27]. NNs were first developed in the mid twentieth century and with an increase in computing power have become powerful tools with extensive research being done to improve them. These networks are increasingly becoming a part of our daily lives; they power selfdriving cars, are used in predicting stock markets, steer NASA's rockets, recommend videos, enhance search engine recommendations, perform medical diagnosis and the reader has probably helped train them by filling in Captcha's online.

3.2. Network Architecture

Depending on the application, several types of network exist. Here the most basic type, a feed-forward NN, is considered. Such a network consists out of several layers, the input layer, multiple hidden layers and an output layer. Each node in a layer is connected to all nodes of the previous layer via network weights, as visualized in Figure 3.1.

For each node, a summation of the products between each incoming weight and its corresponding nodal value from the previous layer is computed, and an extra term, known as the bias term is added. This is visualized in Figure 3.2 and can be written as:

$$z_j^{(l)} = \sum_{k=0}^{N} w_{jk}^{(l)} a_k^{(l-1)} + b_j^{(l)} \tag{3.1}$$

This result is transformed through an activation function to form the output of that node.

$$a_j^{(l)} = f(z_j^{(l)}) \tag{3.2}$$

Activation functions are visualized in Figure 3.3. The output layer usually has linear activation functions for regression problems. Examples of activation functions are formulated as:

- Rectified linear unit (ReLU): $f(x) = max(0, x)$

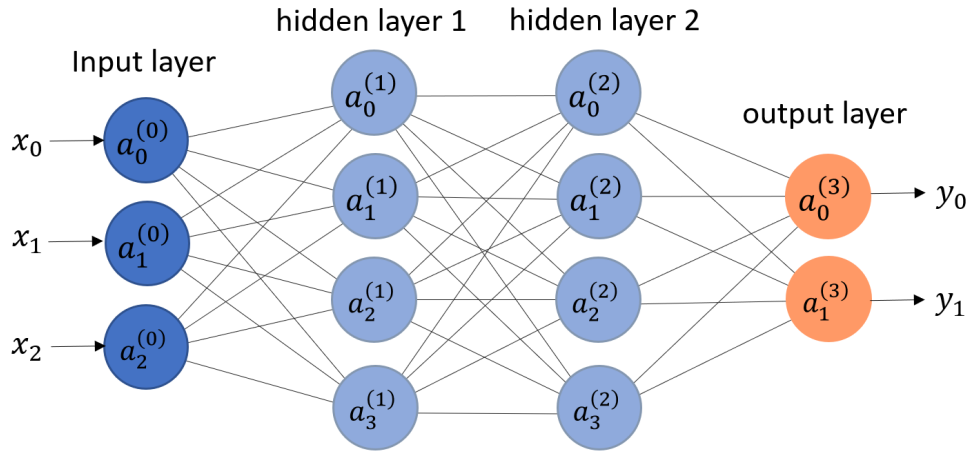


Figure 3.1: Example of a NN with 2 hidden layers.

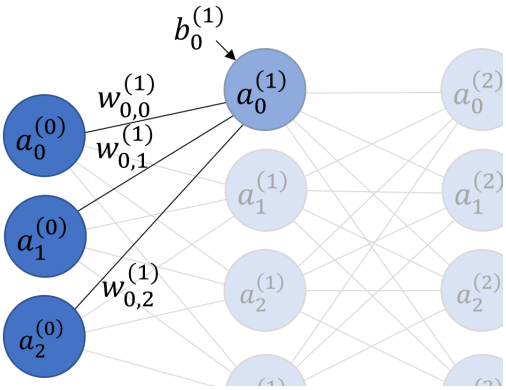


Figure 3.2: Factors influencing the input value of any node.a_i is the value of a node, $w_{i,j}$ and b_j are the weight factor and bias factor respectively, both adjusted during training.

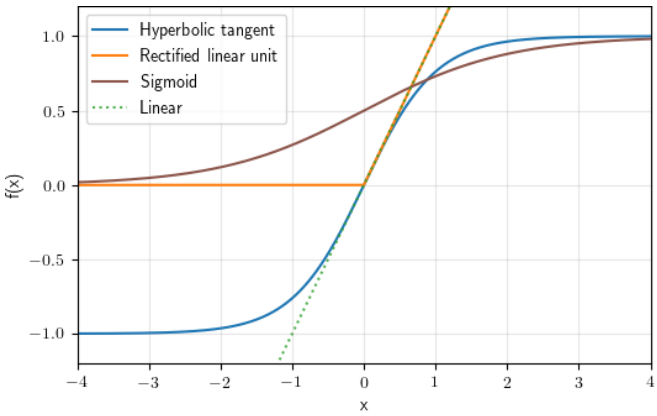


Figure 3.3: Plots of the hyperbolic tangent (tanh), rectified linear unit (ReLU), sigmoid and linear activation functions.

- Sigmoid: $f(x) = \dfrac{e^x}{e^x + 1}$
- Hyperbolic tangent (Tanh): $f(x) = tanh(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$
- Linear: $f(x) = x$

Equations 3.1 & 3.2 are performed for each node of a layer l before continuing this process in the subsequent layer $l + 1$ up to the output layer L. For clarity the bias terms are from now on included in $\mathbf{w}$. The full equation for the output of a neural network with a single hidden layer is then

$$y_k(\mathbf{x}, \mathbf{w}) = f^{(2)}\left(\sum_{j=0}^{H+1} w_{kj}^{(2)} f^{(1)}\left(\sum_{i=0}^{I+1} w_{ji}^{(1)} x_i\right)\right) \tag{3.3}$$

where $f^{(1)}$ is the activation function of the hidden layer and $f^{(2)}$ the activation function of the output layer. The number of iterations in the sums, i.e. $I + 1$ is related to the I nodes in the layer and one additional for including the bias term $x_{I+1} = 1.0$.

3.2.1. Normalization

The input terms $a_k^{(0)}$ and output terms $a_k^{(L)}$ can be normalized, that is scaled between 0 and 1 or between -1 and 1 to allow for better training [28]. Several schemes for normalization exist, such as normalizing to a uniform or a sigmoid distribution. Output terms can also be normalized, depending on the type of problem. In this work the inputs are normalized between 0 and 1, the output is not.

3.2.2. Weight initialization

The weights in a network are generally continuous numbers and are represented by $w_{jk}^{(l)}$. For the network presented in Figure 3.1 there are 36 weights and 10 bias terms. In this work only dense layers are considered, where every node connects via weights to all nodes in the previous layer. Different ways of initializing the weights exist, an incomplete list of possibilities is:

- Starting all weights at 0.
- Start all weights at small values (e.g. -0.01 to +0.01)
- Choosing random values.
- Choosing random values between -1 and 1.
- Initializing weights by drawing samples from a distribution with mean 0 and a variance based on the number of incoming and outgoing connections from that layer. This strategy is found to outperform random initialization for Deep Neural Networks. [29]

Initialization of the weights plays an important role in Deep Neural Networks (containing many hidden layers), and is still an active research area. Special techniques are possible and are dependent on the chosen activation functions. For networks considered in this work the initialization is of less importance.

3.2.3. Hyperparameters

The user defined parameters that decide how the network is created are known as the hyperparameters and include the amount of hidden layers, the amount of nodes per hidden layer, the type of activation function, the initial configuration. Many of these are case dependent and have no general 'best option' but require trial and error.

3.3. Training the network

Starting from a dataset $\mathbf{D}$ of independent and identically distributed observations $x_1, ..., x_N$ with corresponding targets $t_1, ..., t_N$, a neural network with output $y(\mathbf{x}, \mathbf{w})$ is trained in order to approximate these targets. Performing equation 3.2 iteratively for each layer (known as 'forward propagating' the input variables through the network) the outputs are computed. The target t_i and output y are generally vectors, however for clarity the formulation is presented here for a single output. For instance, $\mathbf{x}$ could be a set of fiber coordinates, y the predicted value of σ_{33} and t the value of σ_{33} observed from FE simulations. In the following sections a theoretical description of the training process is provided. In Appendix B a numerical example is presented.

3.3.1. Deriving the loss function

In many introductions to NNs the loss function, that is the function to be minimized, is simply defined as the mean squared error between the network output y and the desired targets t. Here this same loss function is properly derived using a probabilistic view of the network which will help better introduce the applications used in this thesis.

It is assumed that the target variable t is described by the neural network $y(\mathbf{x}, \mathbf{w})$ with added Gaussian noise giving

$$t = y(\mathbf{x}, \mathbf{w}) + \epsilon \tag{3.4}$$

where ϵ is a zero mean Gaussian random variable and $\mathbf{w}$ the weights of a neural network that outputs y. The precision (inverse variance) of this Gaussian is defined as β such that:

$$p(t|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}) \tag{3.5}$$

What this means is that the network output gives the mean of the function and β captures the variance of y around t. For all N observations the likelihood function is:

$$p(t|\mathbf{x}, \mathbf{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n|y(\mathbf{x}_n, \mathbf{w}), \beta^{-1}) \tag{3.6}$$

Training the network comes down to maximizing this likelihood function. It is convenient to do so by maximizing the log likelihood instead:

$$\ln p(\mathbf{t}|\mathbf{w},\beta) = \sum_{n=1}^{N} \mathcal{N}(t_n|y(\mathbf{x}_n,\mathbf{w}),\beta^{-1}) \tag{3.7}$$

$$\ln p(\mathbf{t}|\mathbf{w},\beta) = \frac{N}{2}\ln(\beta) - \frac{N}{2}\ln(2\pi) - \beta E(\mathbf{x},\mathbf{w}) \tag{3.8}$$

Here the first two terms are constant and $E(\mathbf{x},\mathbf{w})$ is a sum-of-squares error function:

$$E(\mathbf{x},\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(t_n - y(\mathbf{x}_n,\mathbf{w}))^2 \tag{3.9}$$

Minimizing this sum-of-squares error is the training of the network. The error is minimized by updating the weights using the negative gradient of the error function. The gradients are computed via a back-propagation procedure. Finding the weights is thus itself a high-dimensional optimization problem. Due to the nonlinearity of a NN the error function is (generally) non-convex, by using a gradient based method a local minima based on the initial configuration is found. It is possible to train several networks with different initial configurations to find separate local solutions.

3.3.2. Backpropagation

The backpropagation algorithm calculates how much the error function is affected by each weight in $\mathbf{w}$. This is done by applying the chain rule of derivatives to go from the error function to the specific weights. The intermediate quantity $\delta_j^{(l)}$ is introduced as the error in the j^{th} neuron in the l^{th} layer.

$$\delta_j^{(l)} = \frac{\partial E}{\partial z_j^{(l)}} = \frac{\partial E_i}{\partial a_j^{(l)}}\frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \tag{3.10}$$

The derivative of E with respect to a weight is then:

$$\frac{\partial E_i}{\partial w_{jk}^{(l)}} = \frac{\partial E_i}{\partial a_j^{(l)}}\frac{\partial a_j^{(l)}}{\partial z_j^{(l)}}\frac{\partial z_j^{(l)}}{\partial w_{jk}^{(l)}} = \delta_j^{(l)}\frac{\partial z_j^{(l)}}{\partial w_{jk}^{(l)}} \tag{3.11}$$

These values are easily computed. For output layer L and the error function as in equation 3.9 it results in:

$$\frac{\partial E_i}{\partial a_j^{(L)}} = (y_n - t_n) \tag{3.12}$$

The error in the nodes $\delta^{(l)}$ is computed from the error $\delta^{(l+1)}$ (hence the name backpropagation), the weights $\mathbf{w}^{(l+1)}$ between them, and the influence of the activation function:

$$\delta^{(l)} = ((w^{(l+1)})^T\delta^{(l+1)})\frac{\partial a^{(l)}}{\partial z^{(l)}} \tag{3.13}$$

Knowing the gradient of each weight to the error, we can make a step in the direction of $-\nabla E(\mathbf{w})$. Doing this iteratively using numerical methods allows us to reach a local minimum where the gradient is close to zero. This is known as gradient descent and is presented for two parameters in Figure 3.4.

3.3.3. Gradient descent methods

Minimizing the error function as presented in equation 3.9 needs to be done for the complete dataset. Updating the weights on all samples of a dataset is considered one epoch and often many epochs are required to find a local minimum. Different methods exist for how to order these samples within one epoch.

In Batch gradient descent (also known as vanilla gradient descent [30]) the complete training batch is used, where the error is averaged over all samples to determine the gradient. This is shown to produce a

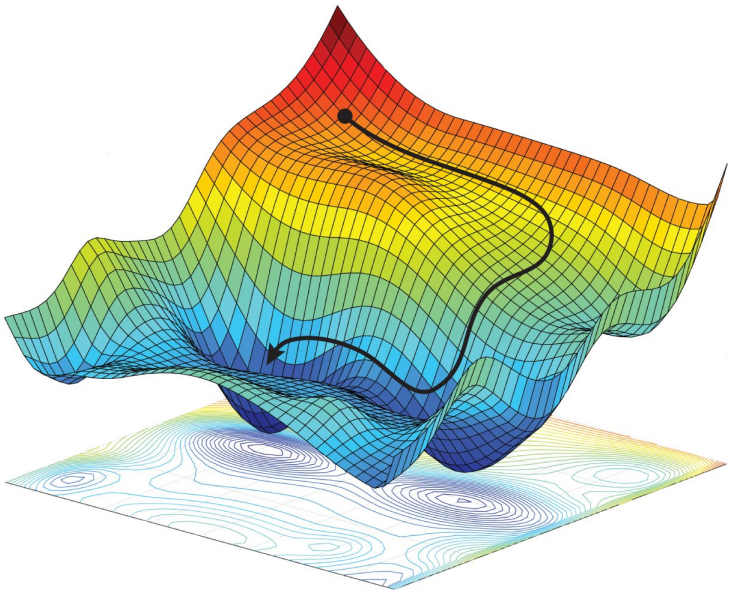


Figure 3.4: Visualization of stochastic gradient descent for 2 parameters [31].

stable learning path. In stochastic gradient descent at every step only a single random sample is used to update the gradient. This deals with a redundancy in Batch gradient descent when the dataset contains similar samples, often making the procedure much faster. Stochastic gradient descent has been shown to have the same convergence behaviour as batch gradient descent, and has the potential to jump to new and potentially better local optima [30].

Combining the previous methods, mini-batch stochastic gradient descent uses a random subset of all the samples to compute the gradient. This generally results in a more stable result than for individual samples, and more computationally efficient than for the full batch.

3.3.4. Overfitting

Minimizing the loss function as defined in Eq. 3.9 can lead to a phenomena knows as overfitting. As the required number of weights in the network is not known beforehand, a network often contains more than the minimum number of weights, which are all used in minimizing the loss function. When a network is underfit, it does not perform well due to not having trained enough or the network being too small. When overfit, it performs well on training data, but fails to generalize to new data, and can be caused by having too many parameters. In Figure 3.5 the network for M=1 is underfit and for M=10 is overfit.

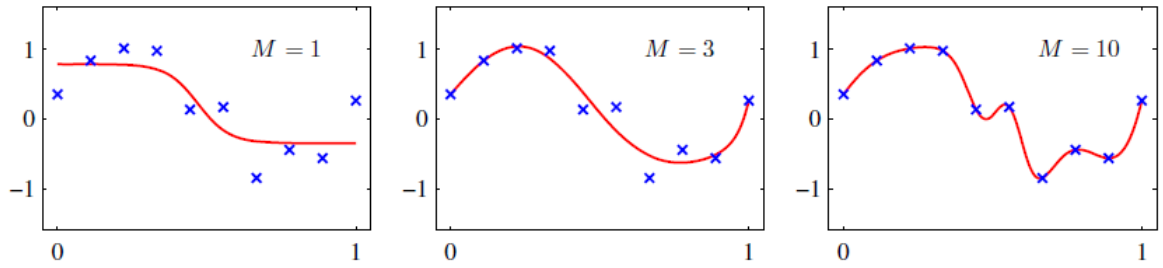


Figure 3.5: Examples of a neural network with M hidden units trained on a sinusoidal dataset. (Source: Bishop [32])

3.3.5. Penalized least squares

A common way to avoid overfitting is by applying a penalized least squares error function to control the complexity of the network. An additional weight decay regularization term is added to the error function.

This error function punishes overly complex networks. Equation 3.9 is than transformed into:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N} ||y(x_n, \mathbf{w}) - t_n||^2 + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} \tag{3.14}$$

where λ is a hyperparameter. This allows networks with many weights to be trained on limited datasets without severely over-fitting. In Section 5.4 this loss function is derived in a probabilistic framework by assuming a Gaussian prior over the weights. The network complexity is now determined by the regularization coefficient λ which transforms the problem of finding the optimal network size into finding a suitable value of λ. The general flow of a Neural network with the penalized least squares error function is shown in Figure 3.6.

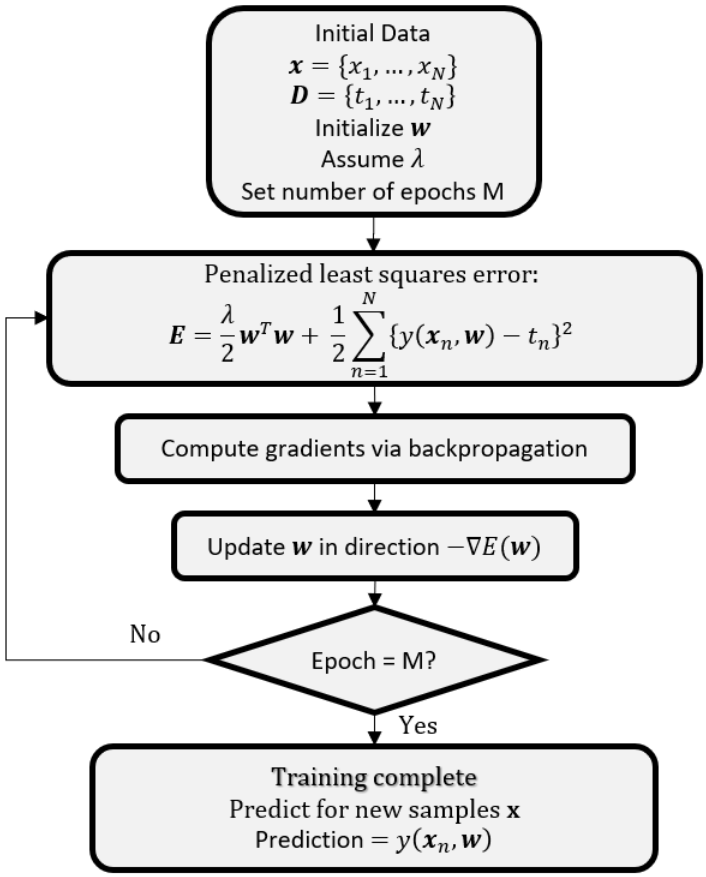


Figure 3.6: General scheme of a neural network training with a penalized least square error function.

3.4. Model selection

Models have a number of hyperparameters. In the case of a neural network these are the network size, activation functions, regularization λ, weight initialization, gradient descent method, number of training samples and convergence criteria. These hyperparameters are not trained for, as for example training λ based on a loss function would always lead to $\lambda = 0$ and an overfit network. As the hyperparameters can have a significant influence on the result a different way of obtaining them is required. Model selection is the process of defining these hyperparameters.

3.4.1. Validation set

To measure how well the network performs, part of the dataset is not used in training and instead in validating the network. Each training cycle, or epoch, the error for both the training and validation data

is computed. Where the training data loss generally constantly decreases, the validation data loss will initially decrease, reducing underfitting. After a certain point however, the validation loss will start to increase and overfitting is occurring. Next to the regularization technique using λ in preventing overfitting, a dropout layer [33] could be used. Neural networks with a high number of neurons are prone to overfitting, whereas using too little neurons can lead to the network always being underfit.

3.4.2. K-fold cross validation

K-fold cross validation (CV) is a way of applying cross validation based on a number of validation sets. From a fixed dataset, a subset is taken as the training data and a subset as the validation data in such a way to allow comparisons of networks with different hyperparameters. K-fold CV divides the data into k folds, or subsets, each fold serving once as the validation data, and k-1 times as part of the training data. This means per configuration of parameters, a network is trained k times. If k is chosen to be the same number as there are datapoints it is referred to as the 'leave-one-out' technique, and can be opted for when data is particularly scarce. Figure 3.7 visualizes training and validation data in a 5-fold CV model selection. The data should be shuffled before being split. By running each configuration of hyperparameters k times and taking the average, a more objective comparison is gained between the different configurations compared to running each configuration a single time. The required number of folds per configuration depends on the dataset and the variance between runs, common values are 5 and 10. [34]

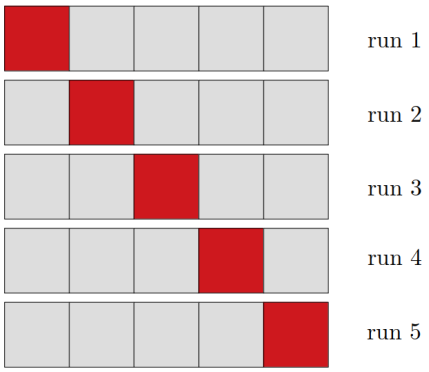


Figure 3.7: Example of the training and validation data in 5-fold CV.

4 Variational autoencoder

As presented in the introduction, the focus of this thesis is on high-dimensional optimization problems with expensive function evaluations. A neural network can be trained to predict the function as presented in Figure 4.1 but requires many computationally expensive samples to do so. The aim of this thesis, using a variational autoencoder, is to separate the dimensionality from the expensive evaluations, as visualized in Figure 4.2. Only considering possible input data, that is without any evaluation of the problem to be optimized, the design space is transformed to a lower dimensionality. This process is detailed in this Chapter, followed by the method of finding the optimum in this reduced design space (with actual function evaluations) in the Chapters that follow. The shape on the cover of this thesis shows a result of an autoencoder, where a 3-dimensional design space, originally a cube, is encoded in a 2-dimensional latent space. The colors that represent the objective function in that image are thus unknown to the variational autoencoder.

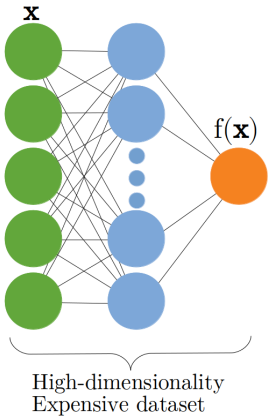


Figure 4.1: Example of a neural network architecture for predicting a function.

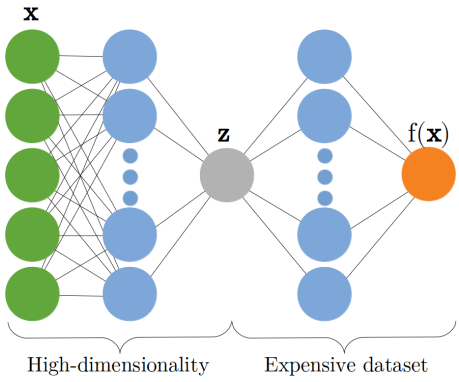


Figure 4.2: Example of a combination of a variational autoencoder and a neural network. The networks from $\mathbf{x} \rightarrow \mathbf{z}$ and $\mathbf{z} \rightarrow f(\mathbf{x})$ are trained separately.

4.1. Introduction

The goal of reducing the dimensionality is to create a lower dimensional design space $\mathbf{z}$ called the latent space that is representative of the full design space $\mathbf{x}$. Dimensionality reduction occurs when the dimensionality M of $\mathbf{z}$ is less than the dimensionality N of $\mathbf{x}$. Several methods exist for achieving this goal, one commonly used is principal component analysis (PCA), also termed as proper orthogonal decomposition (POD). The autoencoder (AE) used here is in essence a non-linear form of PCA.

Ideally, any point in $\mathbf{x}$ corresponds to a specific point in $\mathbf{z}$. This requires some transformation $\mathbf{x} \rightarrow \mathbf{z}$, here called the encoder $e(\cdot)$. From the latent space $\mathbf{z}$, a transformation back to $\mathbf{x}$ is required for the reduction to have any practical use. This transformation $\mathbf{z} \rightarrow \mathbf{x}$ occurs using a decoder $d(\cdot)$. In the process of storing information in less dimensions, a loss of information can occur. This loss of information should be minimized. The general overview of the AE transformations are presented in Figure 4.3.

4.2. Encoding and decoding

To minimize the loss of information, non-linear transformations are required. For general problems no analytical encoders and decoders can be derived, and instead approximation methods are used in the form of a neural network (NN). In general, neural networks cannot be inverted, and separate encoder

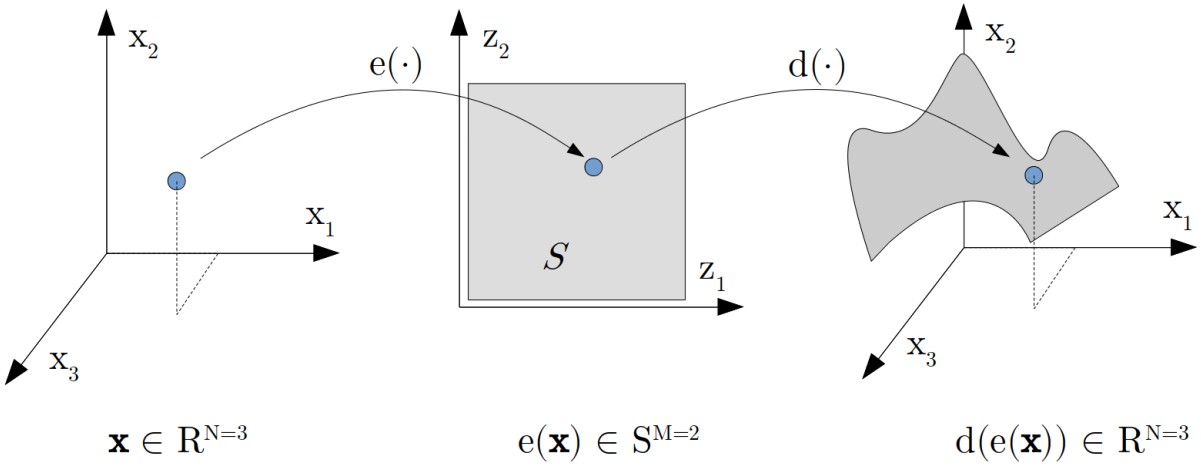


Figure 4.3: Schematic of an AE mapping, this example being a 3-dimensional space *R* being encoded into a 2-dimensional subspace *S*. *N* and *M* can be of any positive integer, when $M < N$ the dimensionality is reduced.

and decoder network parameters are required:

$$\boldsymbol{z} = e\,(\mathbf{x}, \boldsymbol{\theta}) \tag{4.1}$$

$$\tilde{\mathbf{x}} = d\,(\boldsymbol{z}, \boldsymbol{\phi}) \tag{4.2}$$

Where $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ are the weights of the encoder and decoder networks respectively. An example of a NN based on a $6 \rightarrow 2$ dimensionality reduction is presented in Figure 4.4. The encoder and decoder network are required to be trained based on a dataset, where the aim of training is to find the network parameters to minimize the information loss:

$$Loss = \frac{1}{2}\sum_{n=1}^{N} ||\mathbf{x}_n - \tilde{\mathbf{x}}_n||^2 \tag{4.3}$$

Where N is the number of samples in the dataset $\mathbf{D}$. For $\mathbf{D}$ to represent the complete original design space, the required number of samples scales exponentially with its dimensionality. While creating these samples is extremely computationally cheap when only considering possible input data, storing them and considering all of them during training also scales exponentially and becomes infeasible for very high dimensions. As a result, the dataset is generally a subset of the full space. To still consider the broader design space, the AE is required to be generative. This means that when considering a new point in the latent space which is not represented by the original dataset, the AE should interpolate from data nearby in the latent space. With the current loss function in eq. 4.3 the latent space is ill-suited for this, and decoding any point could lead to results that don't make sense based on the training data, similar to overfitting of a standard neural network. An extension is made to solve this issue in the form of a variational autoencoder.

The introduction provided here is specific for the case of dimensionality reduction for optimization. A more common application of a VAE is in generating images, such as images of faces. There the possible configurations of pixels leading to an image of a face is a very small fraction of all possible pixel configurations. In such applications the training data is significantly more difficult to obtain, leading to a similar limited dataset based on a different reason.

4.3. Variational autoencoder architecture

In a variational autoencoder, the latent space $\boldsymbol{z}$ is modified such that it can generate new data similar to the training data. This is done by regularizing $\boldsymbol{z}$ such that it follows a specific distribution, here a standard Gaussian. In this way, all training data should have a distinct point in $\boldsymbol{z}$ but still be close together such that an interpolation in $\boldsymbol{z}$ resembles an interpolation in $\mathbf{x}$.

$$p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}|\mathbf{0}, \mathbf{I}) \tag{4.4}$$

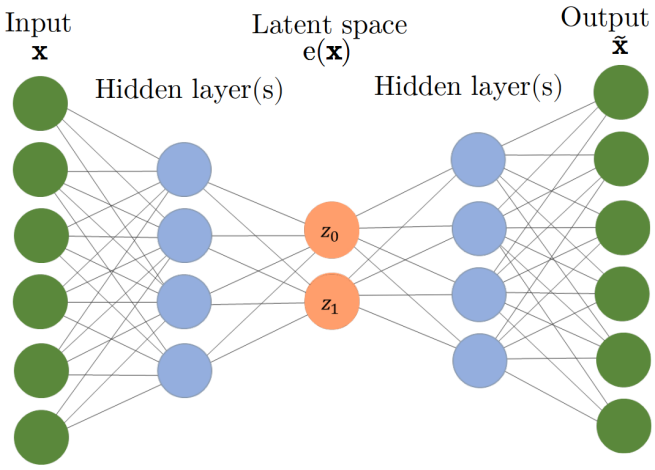


Figure 4.4: Example architecture of the neural network serving as an AE mapping from six to two dimensions. For simplicity a single hidden layer of 4 nodes is presented, in practice significantly larger hidden layers are required.

The encoder notation is changed to $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ which approximates the posterior distribution of the latent space $p(\mathbf{z}|\mathbf{x})$. The goal is to optimize its parameters $\boldsymbol{\phi}$, the networks weights, such that:

$$q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \approx p(\mathbf{z}|\mathbf{x}) \tag{4.5}$$

Simultaneously a generative model learns a joint distribution as

$$p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) = p(\mathbf{z})p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) \tag{4.6}$$

with $p(\mathbf{x}|\mathbf{z})$ the stochastic decoder approximating the transformation back to the original space. The parameter $\boldsymbol{\theta}$ represents the decoder network weights. This is visualized in Figure 4.5. The need for approximations in this derivation (Eq. 4.5) is due to the intractability of computing the posterior distribution. The complete posterior distribution $p(\mathbf{z}|\mathbf{x})$ follows from Bayes theorem [35]:

$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{z})p_{\theta}(\mathbf{x}|\mathbf{z})}{p(\mathbf{x})} \tag{4.7}$$

which requires the marginal likelihood (or model evidence) by marginalizing over $\mathbf{z}$:

$$p(\mathbf{x}) = \int p_{\theta}(\mathbf{x}, \mathbf{z}) d\mathbf{z} \tag{4.8}$$

This integral is in general intractable for neural networks. For a more complete understanding of this intractibility the reader is referred to [36]. The result of this intractibility is that optimization techniques are used to find the best approximations for the encoder and decoder network weights $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ during training.

4.3.1. Optimization objective

As in many other variational methods, finding $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ is done by optimizing the evidence lower bound (ELBO), which is a lower bound on the log-likelihood of the data. It can be derived that this ELBO is [36]:

$$ELBO_{(\boldsymbol{\phi},\boldsymbol{\theta})} = log\left(p(\mathbf{x})\right) - D_{KL}\left(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})\right) \tag{4.9}$$

Where $p(\mathbf{x})$ is the marginal likelihood, and $D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x}))$ the KL (Kullback-Leibler) divergence. By maximizing the ELBO the generative model will lead to a closer reconstruction of the original data. The KL divergence measures the similarity between two probability distributions, in this case between the approximation $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ (assumed a standard Gaussian) and the true posterior $p(\mathbf{z}|\mathbf{x})$. Minimizing the KL divergence therefore means that the approximating distribution becomes more accurate, making the latent space better represent the original space.

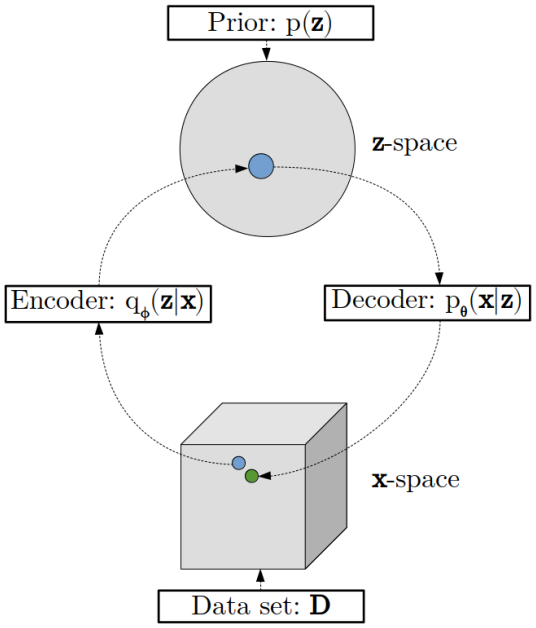


Figure 4.5: Schematic of the VAE mapping, based on: [36]. The reconstruction loss represents the distance between the original (blue) and the transformed (green) datapoint. The KL-loss represents how the distribution of $\mathbf{z}$ (based on the training data) differs from the prior distribution.

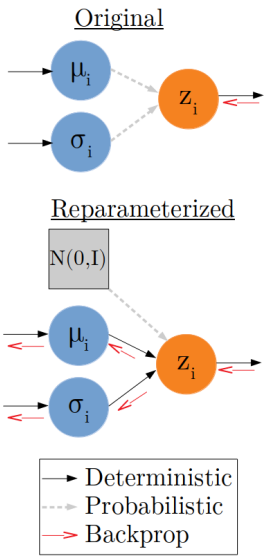


Figure 4.6: Reparameterization trick to allow backpropagating with a random variable when using a standard Gaussian as prior distribution.

4.3.2. Backpropagation

Optimizing the ELBO can be done using stochastic gradient descent as for a standard neural network with a custom loss function. The resulting layer of the encoder, also called the code layer of the autoencoder, consists out of nodes corresponding to the mean and variance of the latent space random variable. A difficulty arises in backpropagation, since drawing random samples from this to form the inputs of the decoder does not allow backpropagation. For a one dimensional latent space, this original form when using a standard Gaussian is:

$$z \sim \mathcal{N}(\mu, \sigma^2) \tag{4.10}$$

Instead what is used is the so-called reparametrization trick, where instead of sampling the Gaussian distribution directly using the mean μ and variance σ result from the encoder, a separate random variable is computed in the form:

$$\xi \sim \mathcal{N}(0, I) \tag{4.11}$$

$$z = \sigma_{\mathbf{x}} \xi + \mu_{\mathbf{x}} \tag{4.12}$$

The comparison between the original form and this reparameterized form is visualized in Figure 4.6.

4.3.3. Differences in type of dimensionality reduction

There is an important difference to be recognized in the common application of a VAE and the application used in this thesis. In most cases of a VAE, the original design space has many regions which are uninteresting. Uninteresting being that the outcome does not make sense or will never be desired. An example is when generating human faces where the inputs and outputs of the VAE represent a grid of $m \times m$ pixels. There is only a very small subset of possible configurations of pixels that will represent a human face. The dataset consists out of these specific configurations where it does represent a face, and this small subset can therefore be mapped to a lower dimensional parameter space.

In the case explored in this thesis for optimization, the complete original design space is of interest, and any possible configuration could be the desired outcome. Instead of mapping a subset of a higher dimensional space into a lower dimensional space, it is attempted to map the full set of possibilities in a higher dimensional space into a lower dimensional space. The difference here is that the dataset does not represent a small subset of the original space but the complete original space. Before applying it in the full optimization scheme, first a study is performed solely using the VAE.

4.3.4. Implementation

A VAE is implemented using the Python libraries Tensorflow and Keras. The loss function from equation 4.9 is transformed into a loss function for regression, assuming a standard Gaussian distribution and adding a regularization parameter λ. For a full derivation the reader is referred to [37].

$$Loss = \frac{1}{2}\sum_{n=1}^{N}\left(||\mathbf{x}_n - d\left(e\left(\mathbf{x}_n, \boldsymbol{\phi}\right), \boldsymbol{\theta}\right)||^2 - \frac{\lambda}{2}\left(1 + log\left(\sigma^2\right) - \mu^2 - \sigma^2\right)\right) \tag{4.13}$$

The first term in the sum of this equation is the reconstruction loss and the second term the KL loss multiplied with regularization parameter λ. This is known as β-VAE [38], in this thesis λ is used instead of β. This function is an extension on Eq. 4.3 and acts similar to the penalized least squares error function in Eq. 3.14. In the second term dependencies on the input and model parameters are omitted. The training and generation processes are presented in Algorithms 1 & 2.

Algorithm 1 Variational autoencoder training

Input: Dataset $\mathbf{D}$ of N samples, each with input and target $\mathbf{x}$
Regularization parameter λ
Encoder & decoder size and activation functions
Number of epochs E
Output: Decoder parameters $\boldsymbol{\theta}$

Initialization
Create the sampling layer
Create the encoder and decoder with random initialization of $\boldsymbol{\phi}$ & $\boldsymbol{\theta}$
Optionally: split $\mathbf{D}$ into a training $\mathbf{D}_t$ and validation $\mathbf{D}_v$ set
Training
for E epochs **do**
    Compute $Loss = \frac{1}{2}\sum_{n=1}^{N}\left(||\mathbf{x}_n - d\left(e\left(\mathbf{x}_n, \boldsymbol{\phi}\right), \boldsymbol{\theta}\right)||^2 - \frac{\lambda}{2}\left(1 + log\left(\sigma^2\right) - \mu^2 - \sigma^2\right)\right)$
    Compute the gradients of $\mathcal{L}$
    Update network parameters $\boldsymbol{\phi}$ & $\boldsymbol{\theta}$ using stochastic gradient descent
end for
Save the decoder parameters $\boldsymbol{\theta}$

Algorithm 2 Variational autoencoder generating new data

Input: Decoder size and activation functions
Decoder parameters $\boldsymbol{\theta}$
Input $\boldsymbol{z}$
Output: new data point $\tilde{\mathbf{x}}$

Load the decoder network parameters $\boldsymbol{\theta}$
Propagate $\boldsymbol{z}$ through the decoder to get $\tilde{\mathbf{x}}$
Output $\tilde{\mathbf{x}}$

4.4. VAE results

Due to the novelty of the application of the VAE for dimensionality reduction in optimization, an important part of this thesis is dedicated to understanding the consequences of different parameters in the VAE. The main features explored are:

- The capability to reconstruct the original space from the lower dimensional latent space.
- The regularity of the latent space.
- The disentanglement of the latent space.

The disentanglement is related to the independence of the latent variables, such that each independent latent variable controls different parameters in the full space. The features are studied by altering a number of relevant parameters in the model. These parameters are the regularization λ, the number of latent dimensions and the activation function in hidden nodes of the encoder and decoder network.

Certain parameters have had some initial exploration but are not extensively studied, such as the number of hidden layers and the amount of nodes in this hidden layer. Unless mentioned otherwise both the encoder and the decoder have a single hidden layer of 300 nodes which allowed for sufficient parametric freedom. Furthermore, the number of epochs in training has been set at 60000.

4.4.1. Test scenario

To study different features, a scenario is created that represents the final optimization goal as presented in Chapter 2, while allowing comparisons for different configurations. A square representative volume element (RVE) is populated in the transversal direction with a random distribution of seven square fibers, as presented in Figure 4.7. The design variables are the center x and y coordinates of an additional fiber moving throughout this space containing fixed fibers, here the morphing parameter is fixed to keep all fibers squares.

$$\begin{aligned} \max \quad & \sigma_{33} = FEA(x, y) && \text{(4.14a)} \\ \text{subject to} \quad & x \in [0.0, 1.0], && \text{(4.14b)} \\ & y \in [0.0, 1.0], && \text{(4.14c)} \end{aligned}$$

In a sense this can be seen as a 2-dimensional slice from a 24-dimensional optimization problem. An example of the mesh for (x,y) = (0.0, 0.0) is plotted in Figure 4.8. Due to the assumption of periodicity, a quarter of this new fiber is in each corner. It is possible to transform any number of input parameters to any number of latent variables. The higher the reduction in parameters, the greater the loss in coverage of the original design space and/or the smoothness of the objective function can be expected. In this preliminary test case the original space is kept at 2 dimensions allowing the full design space to be computed using brute force in reasonable computation time.

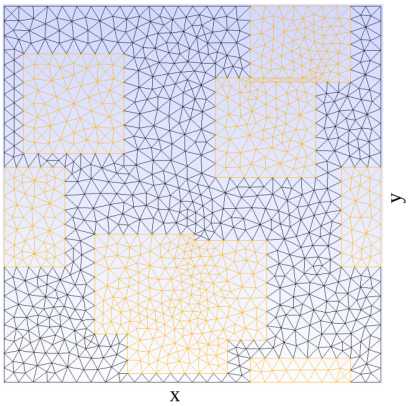


Figure 4.7: Mesh of the 7 static 'background' fibers. The matrix is plotted in black, the fibers in orange.

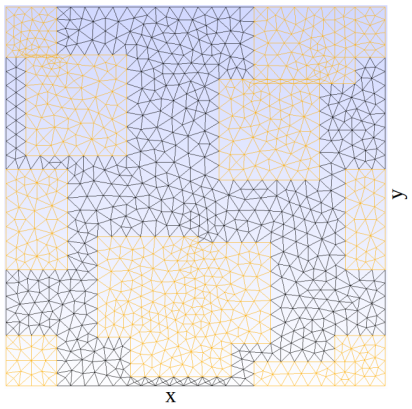


Figure 4.8: Mesh of an example of the 8 fiber scenario, with 7 static fibers. Current fiber center = (0.0, 0.0). The matrix is plotted in black, the fibers in orange.

4.4.2. Reconstruction capability

To study the reconstruction capability, the 2 parameters of the moving fiber are being transformed into a 1-dimensional latent space. The reconstruction capability of the VAE can be visualized but also measured using the reconstruction loss of Eq. 4.13. The dataset is a 256 $\times$ 2 matrix consisting out of values between [0, 1] to form a grid from 256 x & y coordinates.

5-fold cross validation

A 5-fold cross validation study is performed using the k-fold technique which is introduced in Section 3.4.2. An example of the training and validation set of a single run are given in Figure 4.9. Running this 5 times for one particular configuration gives results as in Figure 4.10.

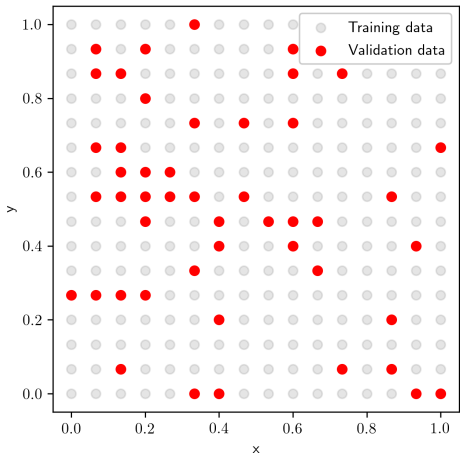


Figure 4.9: Example of the training and validation data in one run of k-fold CV. The dataset contains 256 points, k=5.

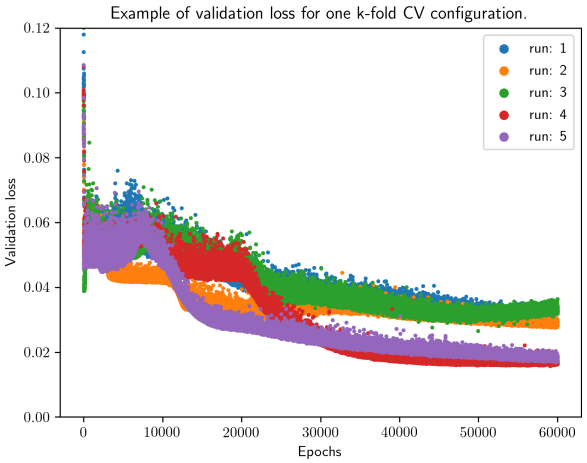


Figure 4.10: Example of the loss functions of 5-fold CV for a single configuration.

Based on the validation loss no longer decreasing (no longer underfit), it can empirically be argued that around 60000 epochs seems to be sufficient for minimizing the validation loss. The validation loss not increasing demonstrates that it is not overfitting either. Taking the mean of the last 100 datapoints of each run, and running this for configurations with varying regularization and activation function results in Figure 4.11.

Here the loss function from Eq. 4.13 (and Eq. 4.9) is distinguished into

$$Loss = Reconstruction\ loss + \lambda \times KL\ loss \tag{4.15}$$

where both the reconstruction and the KL loss sum over the samples. The KL loss includes the $-\frac{1}{2}$ term. In the top plots for the reconstruction loss the first observation is that for a high values of λ, it is significantly higher than for lower values of λ. Plotting the decoded latent space results in a constant fiber coordinate of $(0.5,\ 0.5)$ for $\lambda \geq 1e^{-1}$. From around $\lambda \leq 7e^{-3}$ there is no constant decrease of reconstruction loss observed. This is a result of the second term in Eq. 4.15 being negligible for low values of λ. Furthermore, what can be observed is that for different activation functions there is no significant difference in reconstruction loss.

The first obvious observation about the KL loss is that it increases as the regularization λ decreases. This is expected, since for a high λ minimizing the KL loss has a significant influence on the total loss. For decreasing λ, the gain of having the latent space deviate (from the standard Gaussian the KL divergence is enforcing it to be) for a lower reconstruction loss outweighs the increase of KL loss. In the region $\lambda \geq 1e{-}4$ the KL loss shows to still be effective. Apart from the two outliers of the Tanh activation function, there is no significant difference between the activation functions.

Visualizing results

Next to the numerical values presented previously, results are visualized for different configurations. These visualizations can aid in understanding the results, and give insight into the influence of the parameters on the results. First the full case scenario is presented, followed by results of the VAE in reducing the dimensionality of this space. For the full case scenario the objective function is computed in the 2-dimensional original design space, plotted in Figure 4.12. The function shows to have some clear peaks and valleys, and the possible gain is significant. To be clear, this brute force approach is generally too computationally expensive. It is made tractable here because there are only two design variables, the coordinates of a single fiber.

Transforming this 2-dimensional original space into the 1-dimensional latent space is visualized for nine different VAE configurations in Figures 4.13 & 4.14. The middle plot in each Figure shows the grid of training data being transformed through the autoencoder in red. This is the path the fiber follows through the design space, and the top plot shows the objective function values along this path. Included is the regularization of the VAE by plotting the training data as it is transformed to the latent space, which will be discussed in the next section. For the sigmoid activation function the training loss is included, while

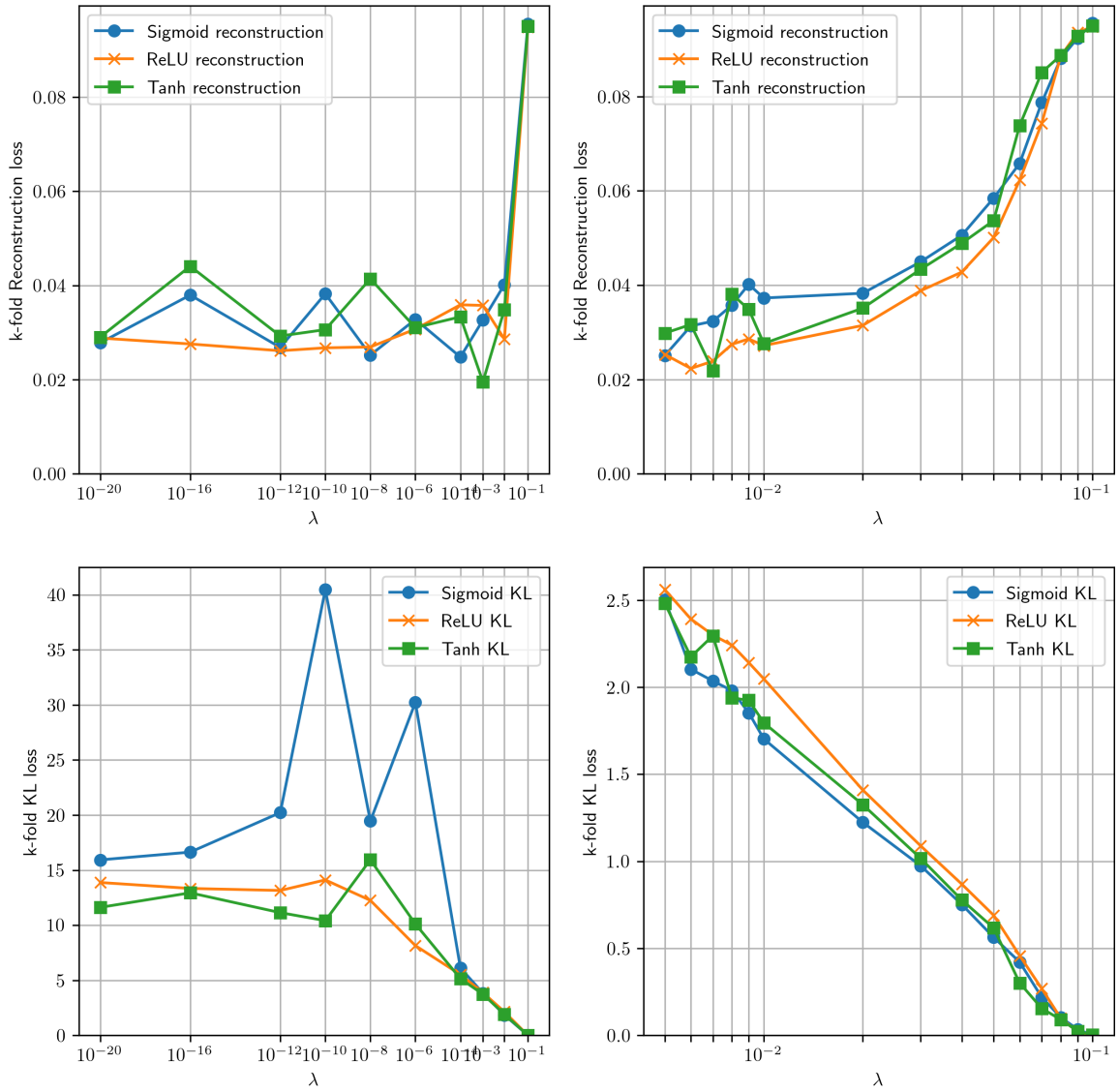


Figure 4.11: Results using 5-fold CV of the reconstruction and KL loss plotted for sigmoid, relu and tanh activation functions with various values of λ. The figures on the right are close-ups of the plots on the left. Each point value is the average of the final 100 epochs for the 5 different runs.

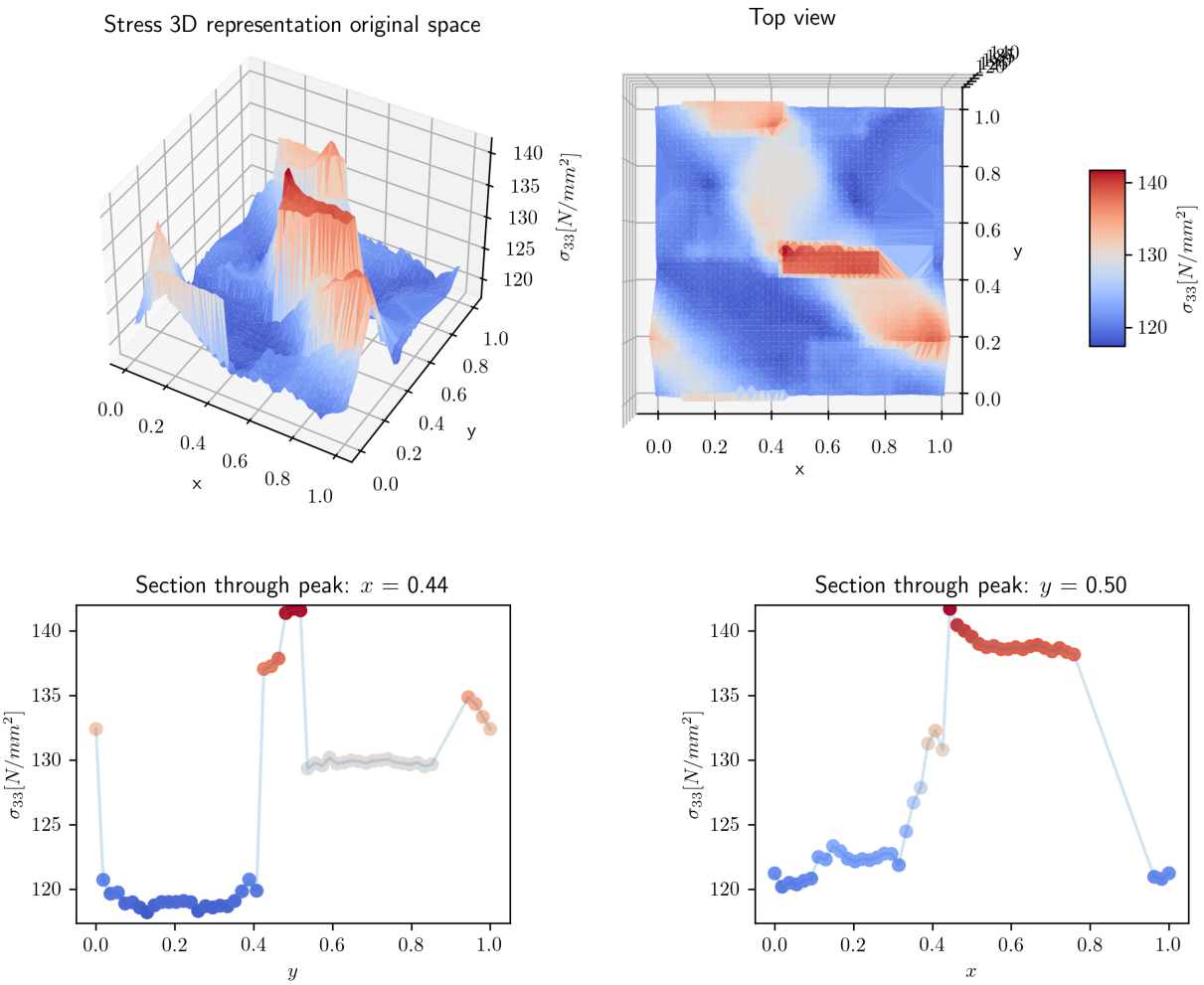
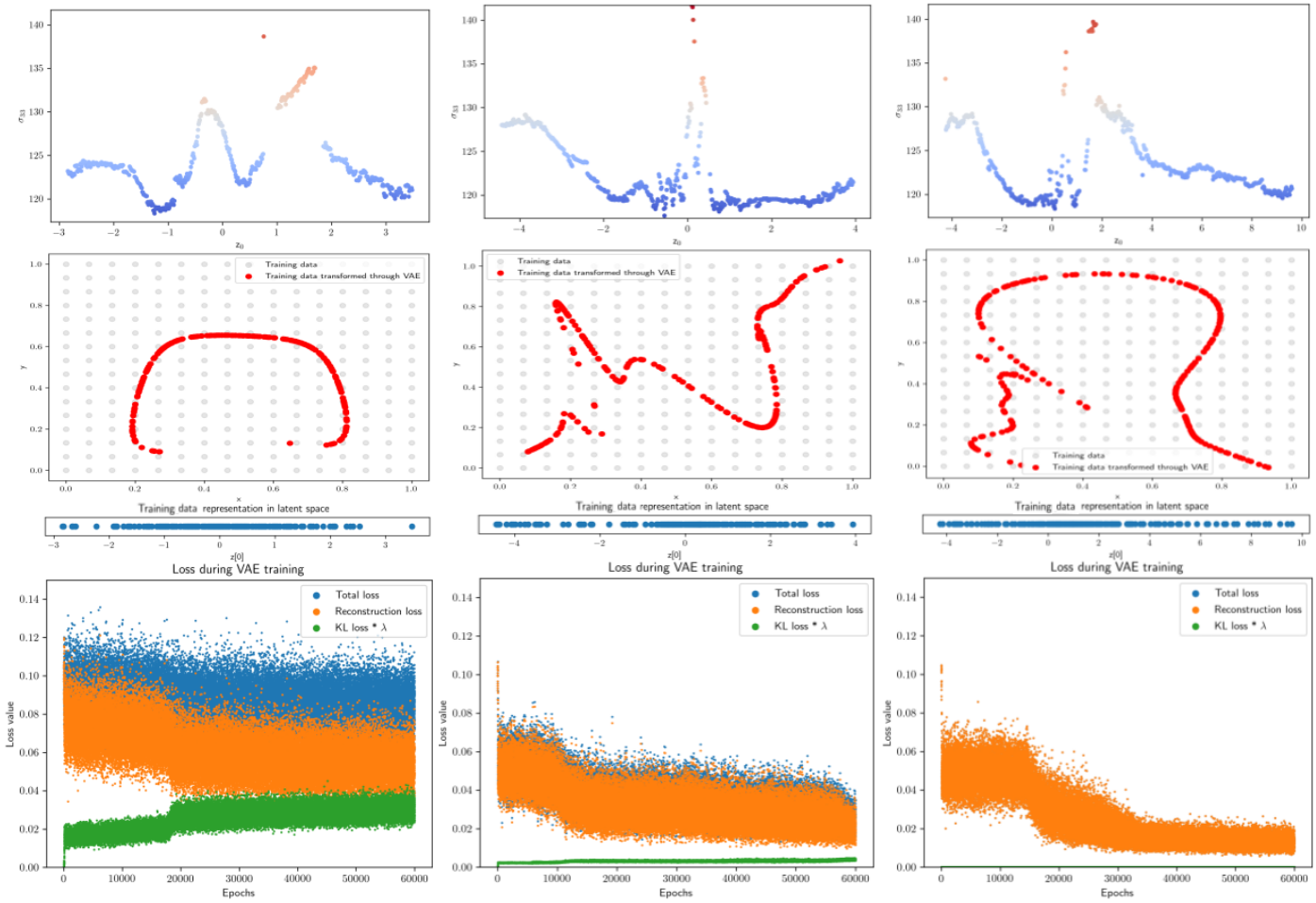


Figure 4.12: σ_{33} (at perfect plasticity) in the original 2-dimensional design space.

for the other activation functions this looks very similar and is therefore not included. These training losses are from a single run, not a k-fold CV study.

For all configurations in Figures 4.13 & 4.14 the plots in the center visualize the position of the fiber for the original and the reconstructed training data, here called the path of the fiber. A clear observation is made that for decreasing λ the path becomes more complex. The top plot can be seen as the path from the middle plot through the top view of Figure 4.12. For this case, the absolute maximum is only included in the reduced space if the path of the fiber goes through this point in the complete space. The VAE, which has no information about the objective function, can only increase this probability by creating a more complex path (decreasing the reconstruction loss). The 3rd plot from the top shows the training data representation in the latent space. The boundaries are included in the description of each figure. For a perfectly regularized VAE this follows a $\mathcal{N}(0, 1)$ distribution and it is observed that as the regularity decreases the distributions deviate increasingly from this (increasing the KL loss).



(a) $\lambda = 5e^{-2}$ | Sigmoid activation

z_0 [-2.84, 3.48]

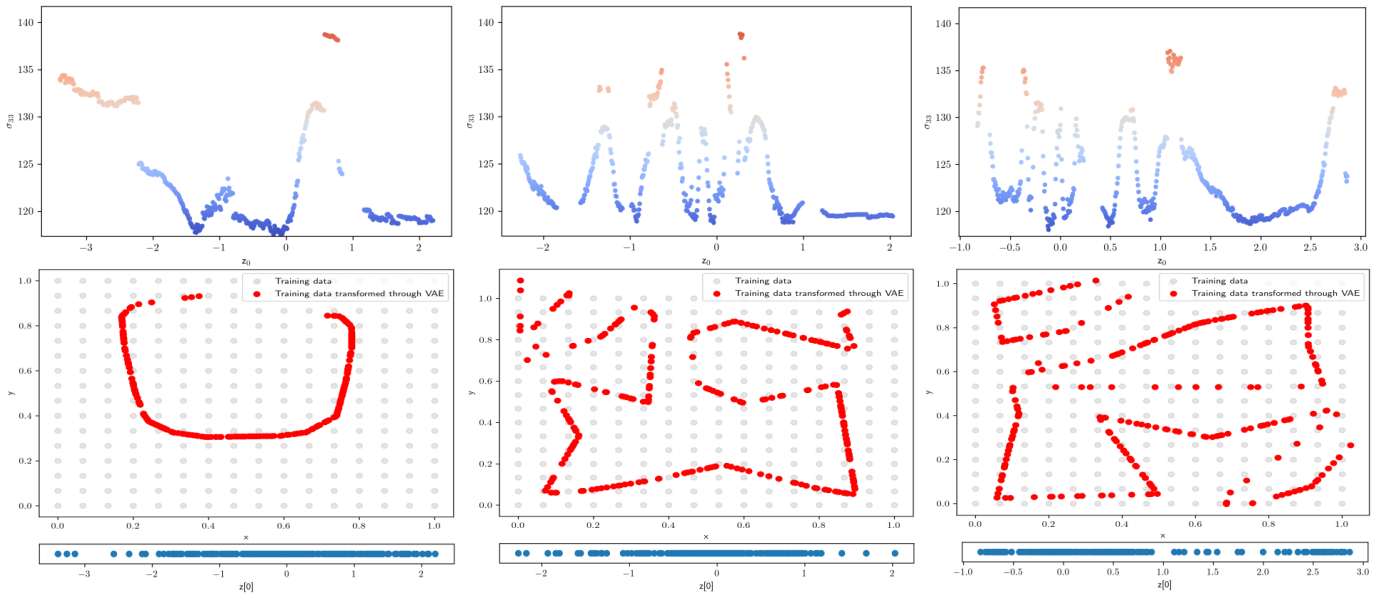
(b) $\lambda = 1e^{-3}$ | Sigmoid activation

z_0 [-4.43, 3.94]

(c) $\lambda = 1e^{-20}$ | Sigmoid activation

z_0 [-4.26, 9.60]

Figure 4.13: From top to bottom: the objective function for the latent space sampled between the boundaries of the reconstructed training data, the corresponding path of the fiber in the design space, the spread of the reconstructed training data in latent space and the loss values of the VAE during training, for different regularization (λ) and with sigmoid activation. The network contains one hidden layer of 300 nodes.



(a) $\lambda = 5e^{-2}$ | ReLU activation

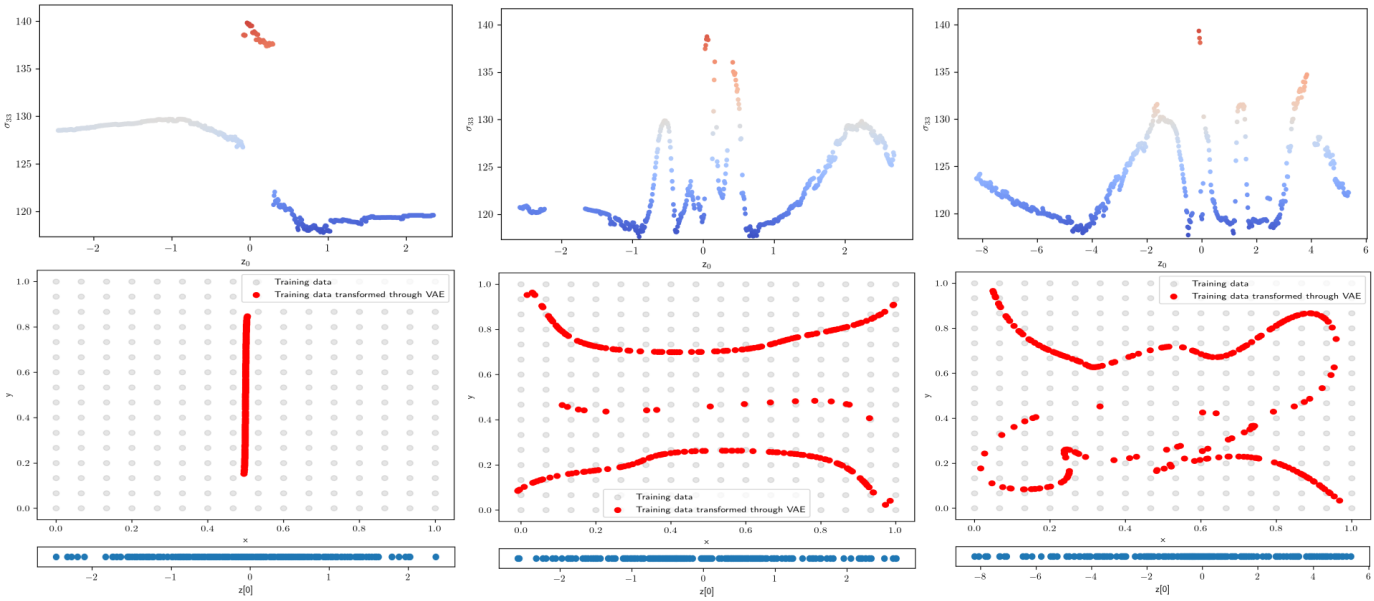
z_0 [-3.40, 2.20]

(b) $\lambda = 1e^{-3}$ | ReLU activation

z_0 [-2.27, 2.03]

(c) $\lambda = 1e^{-20}$ | ReLU activation

z_0 [-0.83, 2.86]



(d) $\lambda = 5e^{-2}$ | Tanh activation

z_0 [-2.45, 2.35]

(e) $\lambda = 1e^{-3}$ | Tanh activation

z_0 [-2.59, 2.69]

(f) $\lambda = 1e^{-20}$ | Tanh activation

z_0 [-8.22, 5.36]

Figure 4.14: From top to bottom: the objective function for the latent space sampled between the boundaries of the reconstructed training data, the corresponding path of the fiber in the design space and the spread of the reconstructed training data in latent space for different regularization (λ) and activation functions. The network contains one hidden layer of 300 nodes.

4.4.3. Regularity of the latent space

The range of the latent space is related to its regularity. The latent space is enforced to follow a Gaussian prior distribution by the KL divergence in the loss function. For a well regularized latent space 95% of the data is expected to be between -2 and 2. As expected, and is demonstrated in the plots of the separate losses for the sigmoid activation function in Figure 4.13, the influence of the regularized KL divergence vastly diminishes as we decrease λ. The sigmoid and hyperbolic tangent both show an increase in the bounds of the latent space for decreasing λ, whereas for the ReLU activation this is less obvious in Figure 4.14. As a result of a decrease in λ, the objective functions also seem to become less regular, most noticeably for the ReLU and Tanh activation functions.

4.4.4. Disentanglement between latent features

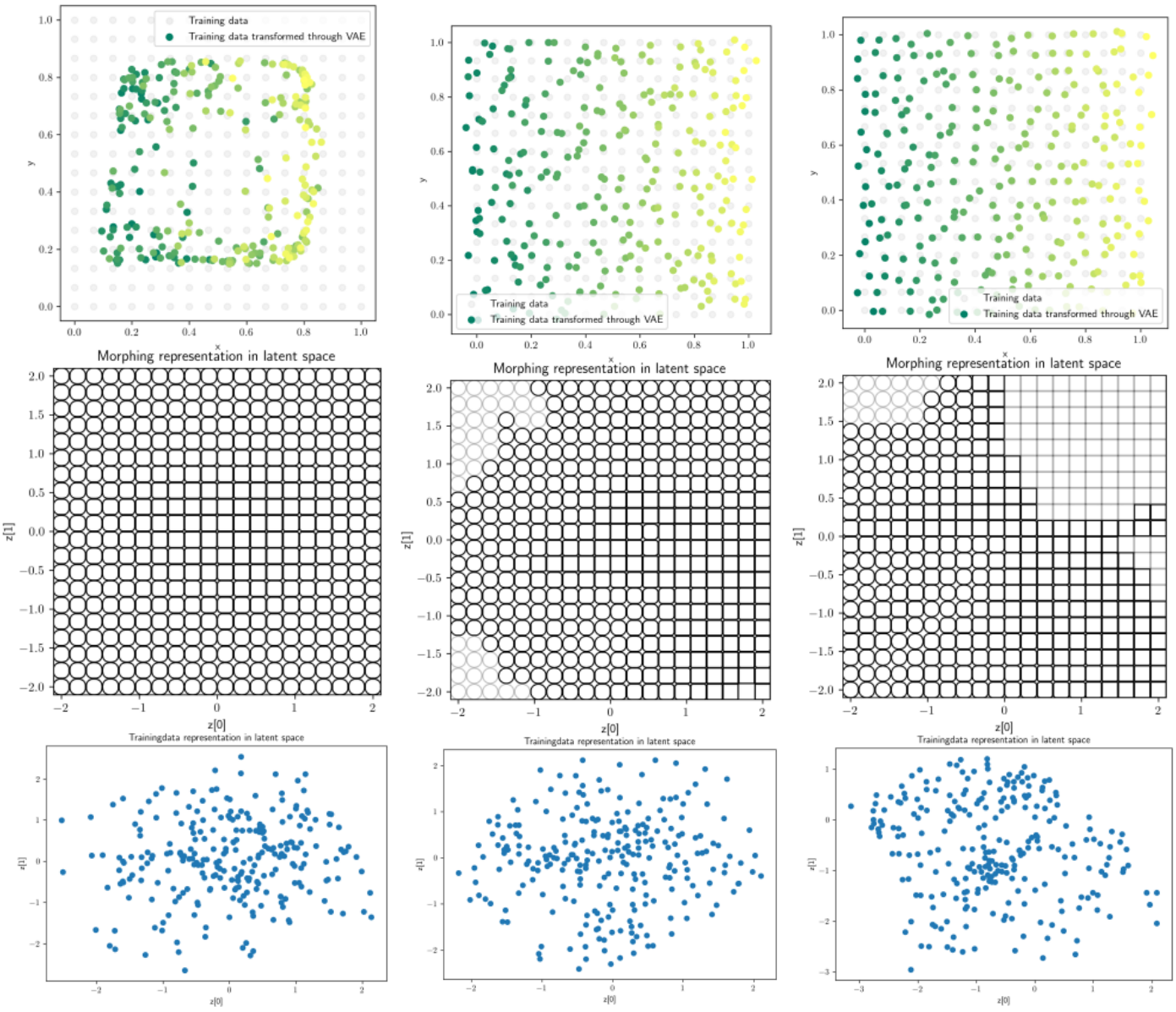
Independence of the latent variables is studied using a case with 2 latent dimensions mapping from an original 3-dimensional space. The 3 original parameters are all the parameters for a single fiber, that is the x & y coordinates and the morphing parameter m. The results are compared for three different settings of λ, with the ReLU activation function being used for all tests. The results are split into 2 different outcomes. Namely the capability of reconstructing the coordinates x & y, and the variation in m throughout the latent space. Both are visualized in Figure 4.15. The reconstructed training data in the latent space is included. While the regularization and the reconstruction of the x & y coordinates are relatively similar for various runs, the result of the morphing parameter can vary wildly for each configuration.

The model with $\lambda = 5e^{-2}$ does not seem to be capable of reconstructing the original space, and shows a very regular, although not disentangled, pattern. Models with $\lambda = 1e^{-3}$ & $\lambda = 1e^{-20}$ do reconstruct the original space well but also show no clear disentangled pattern.

4.4.5. Latent dimensions

The purpose of the VAE is to create a mapping to a lower dimensional space. So far the studies have had a fixed level of original and latent dimensions. Here this is varied to study the influence of differences in dimensionality. An original 8-parameter design space is reduced to a variable number of latent dimensions. The dataset consists of 256 samples of 4 square fibers (m is fixed to 1.0) with random x & y coordinates. 5-fold CV is used to eliminate dependency on initial conditions and it also averages over the uncertainty coming from the limited size of the dataset. The results are presented in Figure 4.16 & 4.17 for the reconstruction loss and the KL loss, respectively.

When the number of latent dimensions is equal to the original design space, the VAE is capable of almost perfectly reconstructing this space. A general trend is observed that as the difference in dimensionality increases, the reconstruction loss increases exponentially. A clear exception here is for a single latent variable, which outperforms the 2-dimensional latent space. The reason this occurs is not clear and requires further studying. For the KL divergence the training and validation losses are very similar. The significant difference in the reconstruction loss between the validation and training data suggests that the number of samples does not represent the complete space and the network might be overfitting. It is suspected that this difference decreases for a larger dataset, but this is not further studied in this thesis.



(a) $\lambda = 5e^{-2}$ (b) $\lambda = 1e^{-3}$ (c) $\lambda = 1e^{-20}$

Figure 4.15: Reconstruction of the coordinates and morphing parameter throughout latent space for different values of λ. Colors in the top plot depend on the original location of the training data. In the middle plot, fibers are grayed when the morphing parameter is either < 0.0 or > 1.0, and have been rounded to 0.0 and 1.0 respectively. This is generally due to being outside of the regularized space, as can be verified using the bottom plot.

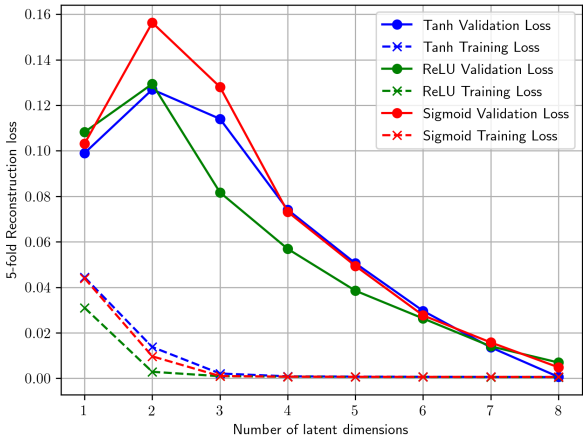


Figure 4.16: The reconstruction loss in 5-fold CV of different levels of dimensionality reduction. Both the encoder and decoder consist of a single hidden layer of 300 nodes. The regularization $\lambda = 1e^{-3}$. An increase in reconstruction loss corresponds to an increase of 'information' loss, that is a loss in the design space.

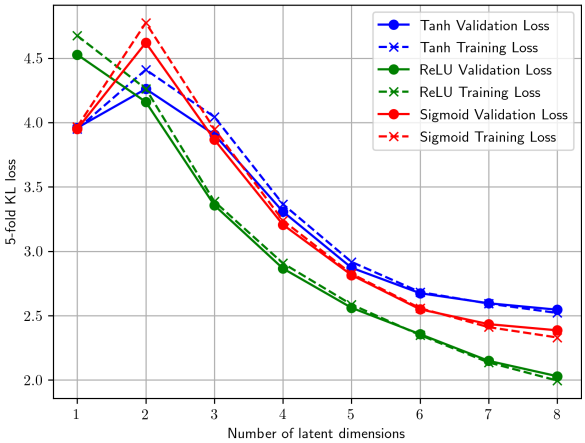


Figure 4.17: The KL divergence loss in 5-fold CV of different levels of dimensionality reduction. Both the encoder and decoder consist of a single hidden layer of 300 nodes. The regularization $\lambda = 1e^{-3}$. An increase in KL-loss corresponds to a loss of regularity in the latent space, leading to predictions that lie further away from the training data.

4.4.6. Preliminary conclusions

Based on the tests performed on the VAE, some preliminary conclusions can be made. There are many different factors that play a role in the capability of the VAE to reduce the dimensionality of a problem. By encoding the result in a lower dimensional design space, part of the original design space can no longer be generated back. This means that the optimum value of the reduced space is not guaranteed to correspond to the optimum of the original design space. The probability that these optima do correspond increases as the reconstruction loss decreases. There is an important trade off between the reconstruction capability and the regularity of the objective function. While several parameters have been presented here, none have a clear optimal value in all cases. Furthermore, it cannot be assumed that the tests performed here scale to problems with many more original design parameters.

Latent dimensions: The number of latent dimensions shows the trend that the larger the dimensionality reduction, the larger the incurred loss. The more computationally expensive the problem, the higher the dimensionality reduction one tends to favour. When requiring a high probability of finding the global optimum, one should however limit this reduction. This results in a careful trade-off to be made between the loss in design space and the number of dimensions, and will depend on the nature of the problem.

Regularization λ: The regularization λ has a significant influence on the result. A large regularization leads to an underfit model, and a low regularization leads to a very irregular objective function which provides difficulties for finding the optimum of the reduced space. λ values of $1e^{-6} \leq \lambda \leq 3e^{-2}$ show good performance, and a value of $\lambda = 1e^{-3}$ seems sensible.

Activation function: The activation functions allow the VAE to create non-linear transformations. The shapes of the functions clearly show to influence the reconstructed design space visually. In numerical comparisons of their reconstruction capabilities, none of the activation functions is distinguishably the best. When comparing different levels of dimensionality reduction, the ReLU activation function performed better overall.

5 Bayesian Neural Network

The latent space of the VAE serves as a lower dimensional space representative of the full design space. When functions are expensive to evaluate, brute force computing in this latent space can still be computationally expensive. A Bayesian optimization (BO) scheme is introduced to maximize this reduced space with very few function evaluations, by predicting the best places to sample. BO requires a surrogate model that can make predictions based on a limited number of samples, and provide information regarding its uncertainty in that prediction. In this thesis a Bayesian neural network is chosen as surrogate model. Apart from providing information on the uncertainty of its predictions, the BNN used also allows automatic model selection to prevent overfitting.

5.1. Background

The main idea behind using a Bayesian Neural Network (BNN) is to replace the deterministic weights used in conventional neural networks with a probability density function (PDF), commonly a Gaussian distribution. By doing this the outcome also becomes a PDF, instead of a single point value, allowing the mean μ and variance σ^2 to be computed. The mean serves as the point value prediction that a standard NN would provide, and the variance can be used as a measure of how confident the network is in that value (the parameter uncertainty [39]). This is visualized in Figure 5.1. Since BNNs are based on a consistent Bayesian formalism, having a separate validation set becomes unnecessary and the marginal likelihood of the dataset can be used for model selection. Using this Bayesian method does increase the computational effort relative to a standard NN, which will become evident in the next sections.

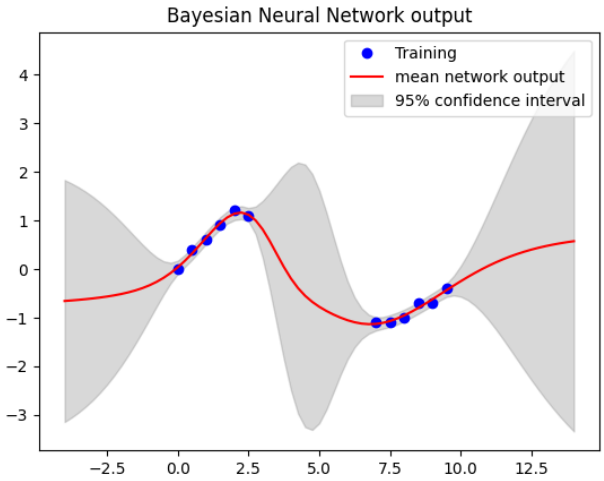


Figure 5.1: Example of the result of a BNN trained on noisy data with a single parameter from a sinusoidal function. The network is confident around the training data as shown by the narrow confidence interval estimating the noise in the data, and less confident away from the training data.

The interest of applying a Bayesian framework to neural networks has grown significantly since works that introduce a practical Bayesian framework for networks based on backpropagation [40] [41]. The Bayesian framework has also been used in convolutional NNs [42] and in recurrent NNs [43].

5.2. Approximating the posterior

In training the BNN, the posterior distribution needs to be evaluated. There is some distribution of weights that has the highest probability of matching the network outputs to the target values for the

dataset. Evaluating the posterior therefore requires using Bayes' Theorem to compute the product of a prior and likelihood function (which generally is a nonlinear function itself), which is intractable for neural networks. Instead, approximation methods need to be used.

5.3. Approximation methods

Markov Chain Monte Carlo MCMC is a sampling technique and used to be the standard for finding an unknown distribution. By sampling using a markov chain the resulting distribution will converge to an accurate result. The problem is that for a large dataset many samples are required, making MCMC computationally expensive. MCMC is suited for small datasets where precise samples are required [44].

Variational Inference (VI) is an alternative approach to MCMC for doing Bayesian inference that uses optimization instead of sampling. A specific shape of the posterior is assumed and approximated using distributions which we can solve for. The accuracy and complexity depends to a large extent on the assumed distribution. VI is known to generally underestimate the variance of the posterior distribution [44]. VI is applied in the variational autoencoder for finding the posterior by optimizing the evidence lower bound.

A third method is the Laplace approximation. It can be termed as a type of variational inference [44], which approximates the posterior with a Gaussian distribution around a local maximum of the posterior. As it is based on a distribution at a specific value it assumes unimodality. Based on the central limit theorem, when observing numerous data points the posterior approximation is expected to be increasingly well approximated by a Gaussian [32], making the Laplace approximation more accurate. An advantage of using a single Gaussian is that the mean and variance are directly available.

5.4. Bayesian loss function

The derivation of the loss function from Section 3.3.1 is extended in order to accommodate an uncertainty over the network weights $\mathbf{w}$. The observation model is still the same as in eq. 3.6 and is

$$p(t|x, \mathbf{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n|y(x_n, \mathbf{w}), \beta^{-1}) \tag{5.1}$$

where the target value is assumed to be Gaussian distributed with the network output $y(x, \mathbf{w})$ as mean and β as precision. Introduced here is a second assumption, namely that the weights $\mathbf{w}$ are themselves also Gaussian distributed:

$$p(\mathbf{w}|\alpha) = \mathcal{N}(0, \alpha^{-1}\mathbf{I}) \tag{5.2}$$

Here α is the precision (inverse variance) parameter of the weights.

In a fully Bayesian approach prior distributions over the hyperparameters α and β would be introduced, and predictions would marginalize over both the weights and these hyperparameters. Here, α and β are set to specific values that are learned directly from the data (Empirical Bayes). For a general Neural Network this still leads to an intractable integral due to the non linearity in the likelihood. Therefore the posterior distribution is approximated by a single Gaussian around a (local) maximum of the true posterior, known as the Laplace method. A second assumption is made that the posterior distribution has small variance compared to the characteristic scales of $\mathbf{w}$, which allows retaining only the linear terms of a Taylor expension of the network around the (local) maximum.

Rewriting Bayes Theorem with the definitions from equations 5.2 and 5.1:

$$p(\mathbf{w}|t, \alpha, \beta) = \frac{p(t|\mathbf{w}, \beta)p(\mathbf{w}|\alpha)}{p(t|\alpha, \beta)} \tag{5.3}$$

The resulting posterior following from this is:

$$p(\mathbf{w}|t, \alpha, \beta) \propto p(\mathbf{w}|\alpha)p(t|\mathbf{w}, \beta) \tag{5.4}$$

$$p(\mathbf{w}|t, \alpha, \beta) \propto \mathcal{N}(0, \alpha^{-1}\mathbf{I}) \prod_{n=1}^{N} \mathcal{N}(t|y(x, \mathbf{w}), \beta^{-1}) \tag{5.5}$$

Substituting the general form of a Gaussian distribution gives:

$$p(\mathbf{w}|t,\alpha,\beta) \propto \frac{1}{(2\pi\alpha^{-1})^{1/2}} e^{-\frac{\mathbf{w}^T\mathbf{w}}{2\alpha^{-1}}} \prod_{n=1}^{N} \frac{1}{(2\pi\beta^{-1})^{1/2}} e^{-\frac{(t_n - y(x_n;\mathbf{w}))^2}{2\beta^{-1}}} \tag{5.6}$$

To find a maximum of this posterior it is convenient to instead maximize the logarithm of the posterior:

$$ln(p(\mathbf{w}|t,\alpha,\beta)) \propto ln(\sqrt{\frac{\alpha}{2\pi}}) - \frac{\alpha}{2}\mathbf{w}^T\mathbf{w} + \sum_{n=1}^{N} ln(\sqrt{\frac{\beta}{2\pi}}) - \sum_{n=1}^{N} \frac{\beta}{2}(t_n - y(x_n;\mathbf{w}))^2 \tag{5.7}$$

Rewriting and dropping constant terms leads to:

$$ln(p(\mathbf{w}|t,\alpha,\beta)) \propto -\frac{\alpha}{2}\mathbf{w}^T\mathbf{w} - \frac{\beta}{2}\sum_{n=1}^{N}(t_n - y(x_n;\mathbf{w}))^2 \tag{5.8}$$

This is equivalent to the penalized least squares error function 3.14 where $\frac{\alpha}{\beta} = \lambda$. Therefore it can be optimized similarly using for example stochastic gradient descent to find the weights corresponding to a (local) maximum of the posterior, namely $\mathbf{w}_{MAP}$. These resulting weights still depend on the chosen α & β and the initial configuration of $\mathbf{w}$.

5.5. Evidence framework

In the previous section α and β are assumed to be fixed values. The evidence framework is implemented to iteratively approximate better values for these hyperparameters. This is a type of model selection, where a model with a higher evidence is considered to more naturally explain the data, as the evidence represents the probability of producing the dataset.

5.5.1. Computing the evidence

As discussed previously, marginalizing over α, β and $\mathbf{w}$ is intractable, and the Laplace approximation is applied. This results in:

$$\ln(p(\mathbf{t}|\alpha,\beta)) \approx -\frac{\alpha}{2}\mathbf{w}^T\mathbf{w} - \frac{\beta}{2}\sum_{n=1}^{N}(t_n - y(x_n;\mathbf{w}))^2 - \frac{1}{2}\ln|\mathbf{A}| + \frac{W}{2}\ln\alpha + \frac{N}{2}\ln\beta + \frac{N}{2}\ln(2*\pi) \tag{5.9}$$

where $\mathbf{t}$ is the complete set of target values, and W the total number of parameters in $\mathbf{w}$. As a result of the approximation using a single Gaussian around a local mode, this equation is only valid at $\mathbf{w}_{MAP}$. The evidence is based on the dataset, and the approximation of the BNN will be increasingly accurate and can therefore be used as a convergence criteria for the complete BNN.

Computing 5.9 is not straightforward. The computation of the determinant $|\mathbf{A}|$ requires computing its eigenvalues. When $\mathbf{A}$ is not positive-definite, negative eigenvalues can occur. Therefore accurately finding $\mathbf{w}_{MAP}$ is crucial. However even when at $\mathbf{w}_{MAP}$ the Hessian matrix is very sensitive to small eigenvalues which can be difficult to determine accurately. An eigenvalue smaller than zero corresponds to a posterior weight uncertainty exceeding the prior uncertainty, which does not make sense. Eigenvalues below a certain threshold are therefore omitted from the computation of $|\mathbf{A}|$. Other studies found a similar behaviour and have similarly put a threshold or neglected any negative eigenvalues [45, 46]. In Appendix C.3 a more detailed study is performed on the eigenvalues. A threshold on the eigenvalues is set to omit any eigenvalues below $1e^{-10}$.

5.5.2. Updating the hyperparameters

Based on the evidence framework the hyperparameters can be updated, for the full derivation the reader is referred to [32]. The error function is defined as:

$$E(\mathbf{x},\mathbf{w},\alpha,\beta) = -\frac{\alpha}{2}\mathbf{w}^T\mathbf{w} - \frac{\beta}{2}\sum_{n=1}^{N}(t_n - y(x_n;\mathbf{w}))^2 - \frac{1}{2}\ln|\mathbf{A}| + \frac{W}{2}\ln\alpha + \frac{N}{2}\ln\beta + \frac{N}{2}\ln(2*\pi) \tag{5.10}$$

The aim is maximizing Eq. 5.9 with respect to α, and to do so the second order derivatives of the error function are required, this is the Hessian $\mathbf{H}$. The eigenvalue equation is defined:

$$\beta \mathbf{H} \mathbf{u}_i = \lambda_i \mathbf{u}_i \tag{5.11}$$

Based on the eigenvalues, the effective number of parameters γ is computed:

$$\gamma = \sum_{i=1}^{W} \frac{\lambda_i}{\alpha + \lambda_i} \tag{5.12}$$

where W is the number of network parameters, equal to the rows or columns in $\mathbf{H}$. Again, eigenvalues below $1e^{-10}$ are omitted from γ (equivalent to setting $\lambda_i = 0$). This is then used to update α & β.

$$\alpha = \frac{\gamma}{\mathbf{w}_{MAP}^T \mathbf{w}_{MAP}} \tag{5.13}$$

$$\beta = \frac{N - \gamma}{\sum_{n=1}^{N} (t_n - y(x_n; \mathbf{w}_{MAP}))^2} \tag{5.14}$$

After updating α & β the posterior has changed, and a new $\mathbf{w}_{MAP}$ has to be found.

In a standard neural network it is common to run a predefined number of epochs to find $\mathbf{w}_{MAP}$. Here after updating α & β the weights are likely to already be closer to the next $\mathbf{w}_{MAP}$ and therefore a convergence criterium is used instead based on the gradients of $\mathbf{w}$. For these gradients to be comparable, batch gradient descent is used as explained in Section 3.3.3. As is evident from eq. 5.12 & 5.14 the number of samples N needs to be greater than the network size W to ensure $N - \gamma > 0$. This is analogous to having more equations than unknowns when solving a system of linear equations. This effectively couples the sizes of the network and dataset, with practical consequences for when sampling is performed sequentially as in BO. After updating the hyperparameters, the gradient descent algorithm for updating the weights should be reset.

5.6. Predicting new outputs

Once several iterations have been ran and α & β have converged the network is considered to be trained and can be used to obtain a normal distribution for any output. The mean is the network output, $y(x_{new}; \mathbf{w}_{MAP})$. The variance is computed via the following steps:

First the matrix $\mathbf{A}$ consisting of second order derivatives of the posterior (eq. 5.8) is computed. $\mathbf{A}$ does not depend on x_{new} and can therefore be computed once at the end of training.

$$\mathbf{A} = -\nabla\nabla \ln p(\mathbf{w}|t, \alpha, \beta) = \alpha \mathbf{I} + \beta \mathbf{H} \tag{5.15}$$

Where $\mathbf{I}$ is the Identity matrix of equal size as $\mathbf{H}$. For each x_{new} the gradient to the output with respect to the weights is computed:

$$\mathbf{g} = -\nabla_{\mathbf{w}} y(x_{new}, \mathbf{w})|_{\mathbf{w}=\mathbf{w}_{MAP}} \tag{5.16}$$

This gradient is computed for each output y separately, and used to compute the variance.

$$\sigma^2(x) = \beta^{-1} + \mathbf{g}^T \mathbf{A}^{-1} \mathbf{g} \tag{5.17}$$

Using this variance a 95% confidence interval can be determined as $\mu \pm 2\sigma$. The general scheme of the Bayesian Neural Network using the Laplace approximation is presented in Figure 5.2. An example of a trained network is given in Figure 5.1. The network can have any number of input and output parameters. The convergence criteria for finding $\mathbf{w}$ is $\mathbf{w}_{MAP,tol}$, and the convergence criteria for the hyperparameters Ev_{tol} is based on the evidence, both are defined in Appendix C.1.

5.7. Implementation

The BNN is implemented using the Jem/Jive C++ library [8] and an existing NN framework [10]. Three different methods of computing the Hessian have been implemented, the outer product approximation,

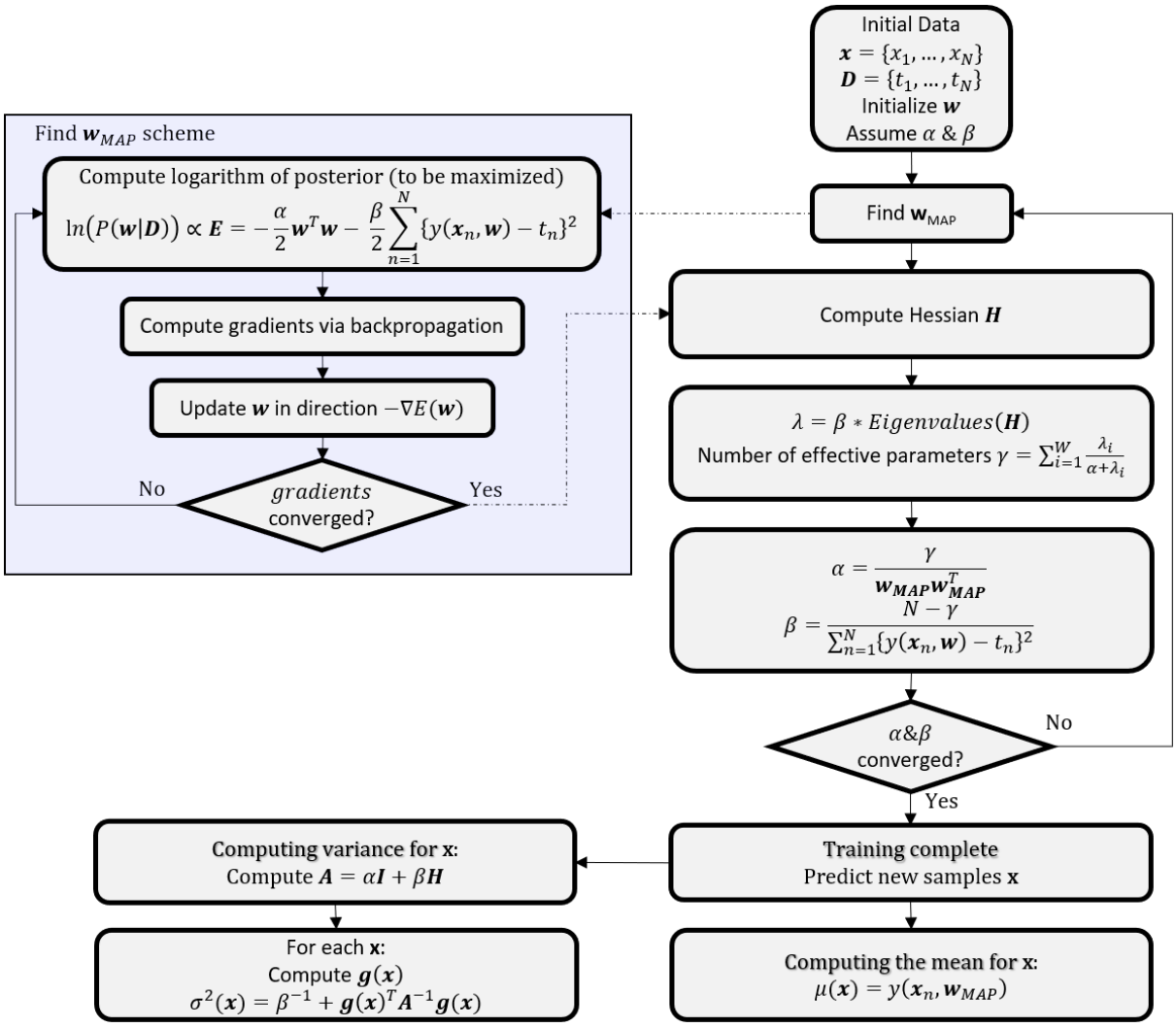


Figure 5.2: General scheme of a Bayesian neural network using the Laplace approximation. The 'find $\mathbf{w}_{MAP}$ scheme' is equivalent to training a standard neural network as presented in Figure 3.6.

the finite differences approximation and an exact computation. The exact computation of the Hessian is used. A comparison between the methods is provided in Appendix C.2. The influence of different values of α & β is studied in the following section.

Different weight initialization schemes are presented in Paragraph 3.2.2, here they are initialized with mean μ and a variance based on the number of incoming and outgoing connections from that layer [29]. Iteratively adjusting the weights to find w_{MAP} happens using Batch gradient descent. The framework used is the ADAM scheme, an algorithm for first-order gradient-based optimization of stochastic objective functions. The initial values for this algorithm are as recommended by Kingma and Ba [47].

5.8. Output uncertainties

Hyperparameter α, as defined in eq. 5.2 represents the precision of the weights. It functions similar to λ for a penalized least squares error function, to prevent the network from overfitting. Hyperparameter β, as defined in eq. 5.1 represents the precision (or certainty) of the outcome. The uncertainty (β^{-1}) in the output represents two possible causes. The first is the network not being fully trained, and therefore giving uncertainty in prediction values. The second possibility arises from the intrinsic noise in the dataset. For a fully trained network of sufficient size the β term will more closely resemble the noise in the original dataset. The second term in eq. 5.17 represents the uncertainty of the network away from training data.

The different terms are visualized in a test case with a dataset containing a number of peaks. The

training data contains 83 points, and is predicted using a BNN with a single hidden layer of 27 nodes, resulting in 81 network parameters. The initial α value is selected at $1e^{-8}$, $\mathbf{w}_{MAP,tol} = 2e^{-6}$, and $Ev_{tol} = 1e^{-2}$. The training is presented in Figure 5.3. At several points before updating the hyperparameters the prediction of the network at its current state is presented in Figure 5.4, where the plots correspond to the points from Figure5.3.

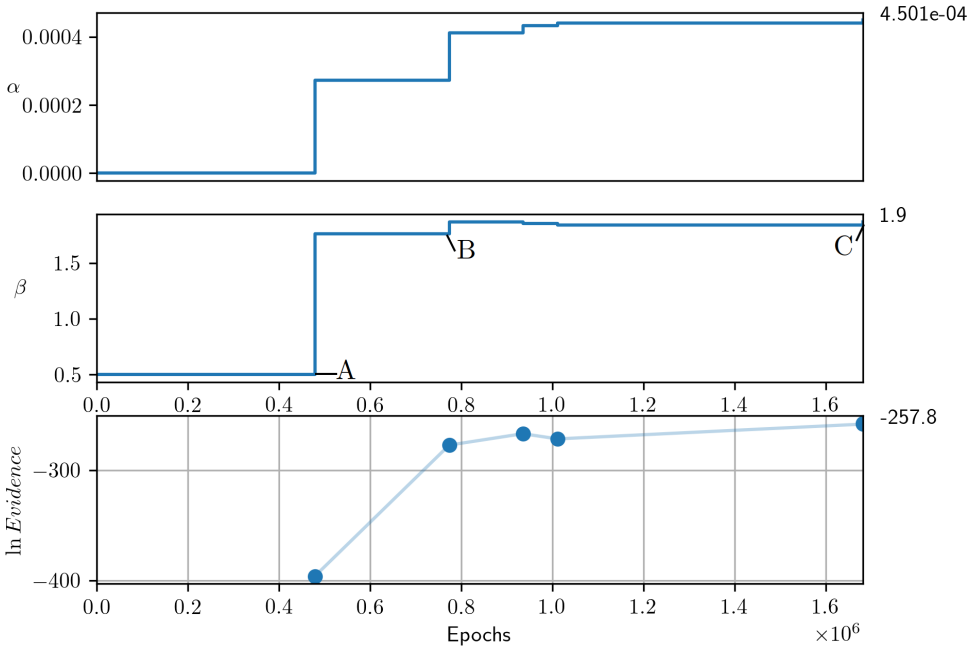


Figure 5.3: The hyperparameters and evidence during training of a BNN. A, B and C represent different converged points, and correspond to the plots in Figure 5.4.

This demonstrates that due to the low initial value of α, the network is allowed high complexity. During further training it reduces this complexity as it increases the evidence. Furthermore, the second term, $\mathbf{g}^T\mathbf{A}^{-1}\mathbf{g}$, clearly shows to increase away from training data. Around $z[0] = 1.2$ there are no training points, but due to the significant difference in training data around it this second term still gives low values.

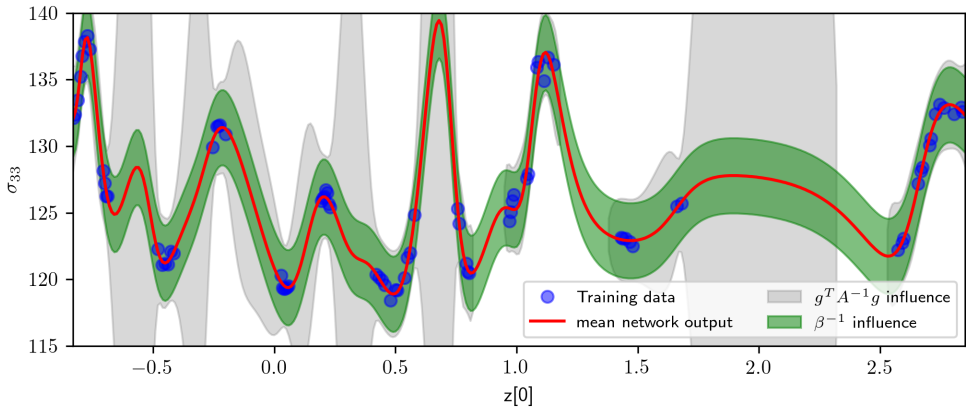
5.9. BNN Results

There are several factors the training of a BNN can be impacted by. The ADAM parameters used for updating the weights are kept at their recommended values. Other parameters are the initial hyperparameters α & β, the convergence criteria $\mathbf{w}_{MAP,tol}$ & Ev_{tol} and the network size and activation function.

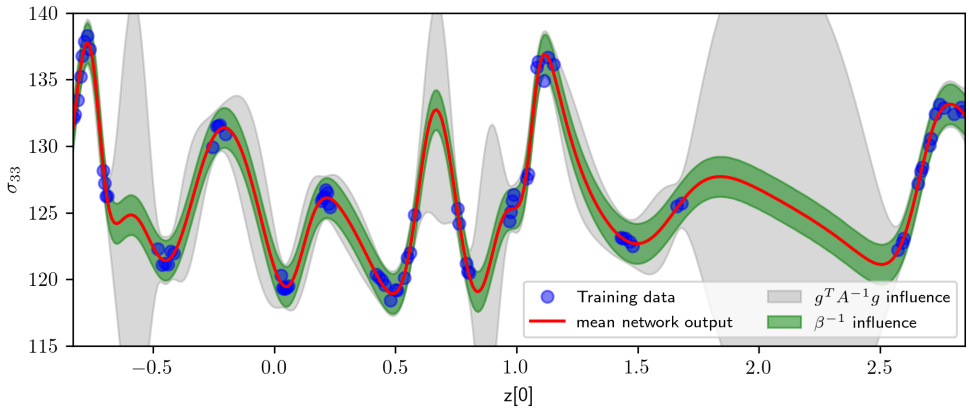
The initialization of the hyperparameters is mostly about finding the ratio between them. It is chosen to keep the initial β constant at $\beta = 0.5$, and study the effect of different initial α values. A low initial α allows the model to find a 'good' fit before being 'punished' on complexity. The convergence criteria both influence the results significantly and are studied further. The network size is constrained by $W \leq N$ (discussed in 5.5.2), where the number of network weights for a single output is $W = (Nodes_{input} + 2) * Nodes_{hidden} + 1$. For a given problem $Nodes_{input}$ is fixed and when the number of samples N is limited, $Nodes_{hidden}$ is maximized while satisfying the constraint. No significant difference between sigmoid and the hyperbolic tangent activation function was found. Due to different initialization of weights, each configuration can still vary in outcome.

The performance of the configurations is measured with three quantities, namely the resulting evidence, the final β value and the computation time. From a data oriented point of view, only the evidence should be considered. For the case of optimization, this cannot be done as is illustrated by a model with a high evidence in Figure 5.5.

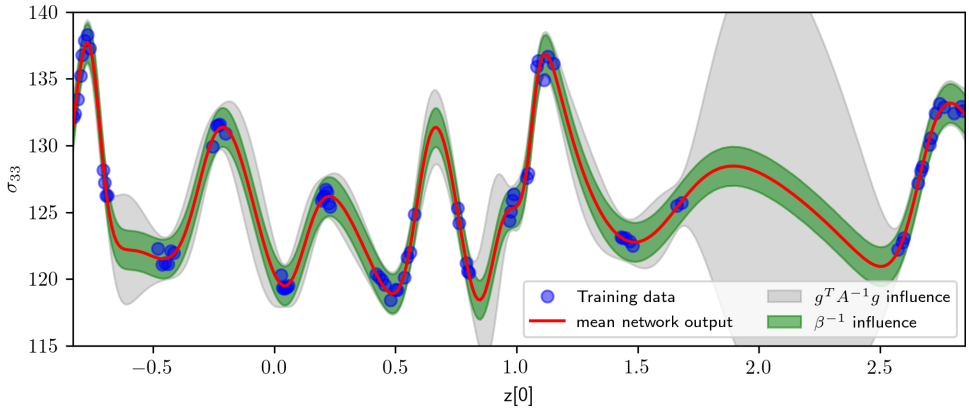
For this reason, the final β value is included in the results, and a low value suggests an underfit network. Finally, the computation time is measured. The values are compared on four different datasets. All datasets are subsets of a single dataset obtained using brute force computation on a micromechanical



(a) Result at training point A



(b) Result at training point B



(c) Result at training point C

Figure 5.4: BNN output at different points during training corresponding to Figure 5.3. The terms in equation 5.17 are separately plotted. As α increases, the complexity of the network decreases to find the simplest model. As β increases, the overall uncertainty decreases.

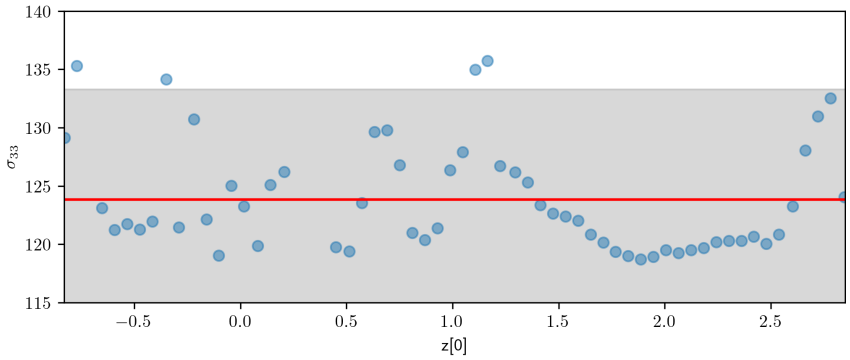


Figure 5.5: BNN surrogate model with a high evidence, but of no interest for optimization due to a final $\beta = 0.045$. While a low β itself is not necessarily a problem, it is linked to an underfit model and an almost constant variance. The used parameters are: $\alpha_{init} = 1e^{-4}$, $\mathbf{w}_{MAP,tol} = 5e^{-6}$, $Ev_{tol} = 1e^{-3}$.

model reduced to 1 dimension using a VAE, from Section 4.4. These subsets are:

- *Full*: The full dataset, $N = 573$.
- *1_10*: every 10th sample of the full dataset, $N = 57$.
- *1_5*: every 5th sample of the full dataset, $N = 114$.
- *Expl*: The full dataset manually modified to cluster data, $N = 83$.

First α_{init} & $\mathbf{w}_{MAP,tol}$ are altered with a fixed Ev_{tol} on several datasets and the results are presented in Figures 5.6 - 5.12. The datasets are presented with a fitted BNN next to them in Figures 5.7 - 5.13.

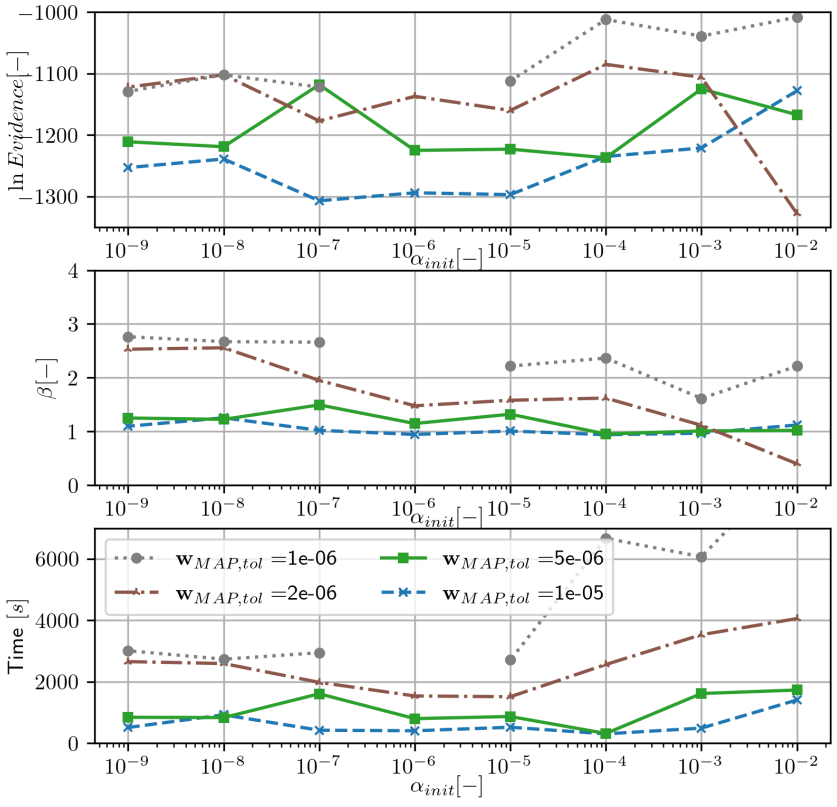


Figure 5.6: Parameter study for the *Full* dataset with varying configurations. The network contains 60 hidden nodes ($W = 181$). $Ev_{tol} = 1e^{-3}$. For models whose points are missing from the plots, $\mathbf{w}_{MAP}$ was not found in $3e^6$ epochs and the training was stopped. Result $\mathbf{w}_{MAP,tol} = 1e^{-6}$ has a computation time 9292 seconds and is cut-off for visability.

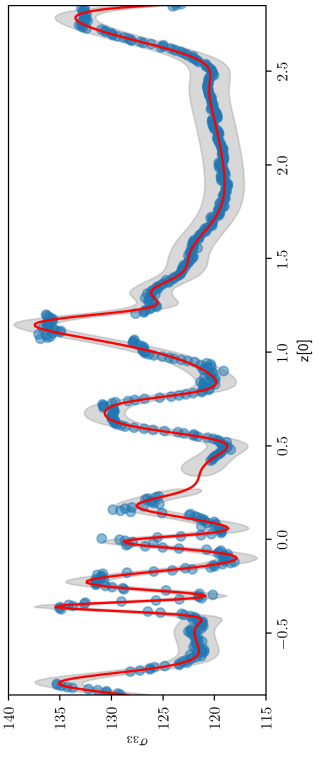


Figure 5.7: Resulting network prediction for the *Full* dataset: $\alpha_{init} = 1e^{-3}$, $\mathbf{w}_{MAP,tol} = 2e^{-6}$ & $Ev_{tol} = 1e^{-3}$. Final $\beta = 1.115$.

Before studying the results it should be noted that the numerical values cannot be compared between the different datasets. The computation time per epoch scales with both the number of samples and the

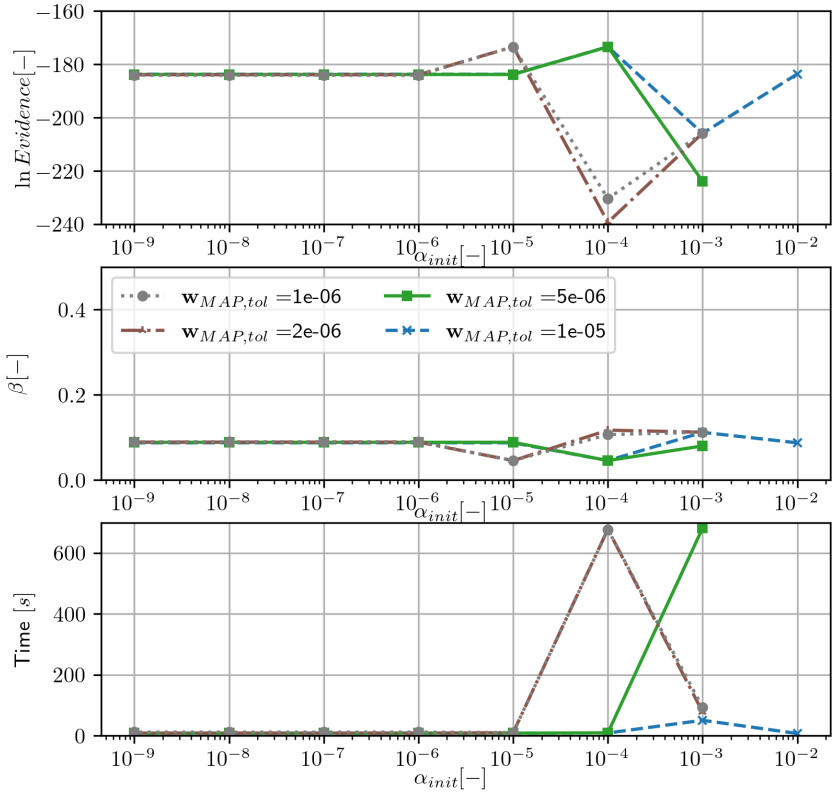


Figure 5.8: Parameter study for dataset *1_10* with varying configurations. The network contains 18 hidden nodes ($W = 55$). $Ev_{tol} = 1e^{-3}$. For models whose points are missing from the plots, $\mathbf{w}_{MAP}$ was not found in $3e^6$ epochs and the training was stopped.

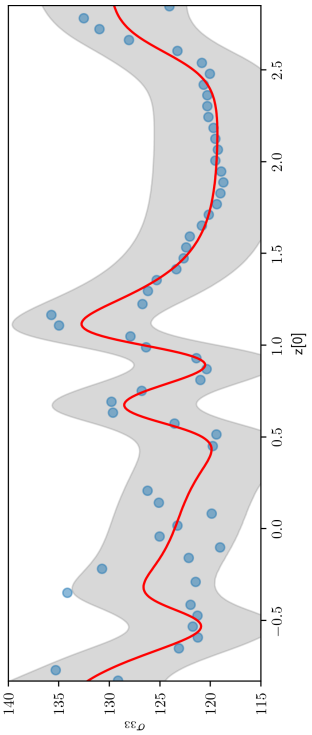


Figure 5.9: Resulting network prediction for dataset *1_10*: $\alpha_{init} = 1e^{-3}$, $\mathbf{w}_{MAP,tol} = 2e^{-6}$ & $Ev_{tol} = 1e^{-3}$. Final $\beta = 0.1121$.

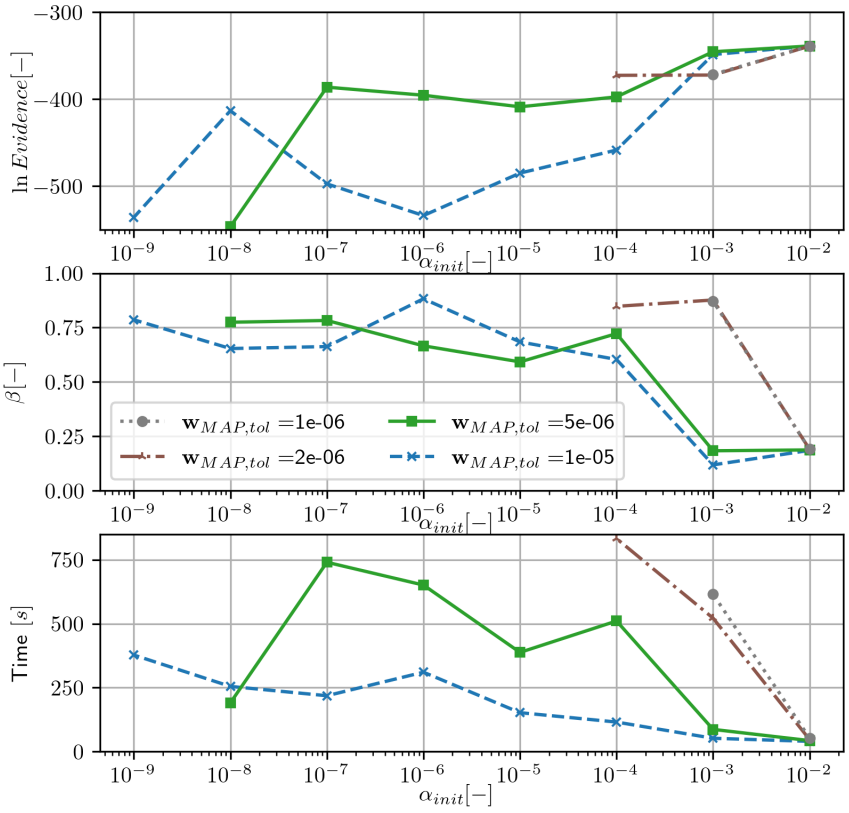


Figure 5.10: Parameter study for dataset *1_5* with varying configurations. The network contains 37 hidden nodes ($W = 112$). $Ev_{tol} = 1e^{-3}$. For models whose points are missing from the plots, $\mathbf{w}_{MAP}$ was not found in $3e^6$ epochs and the training was stopped.

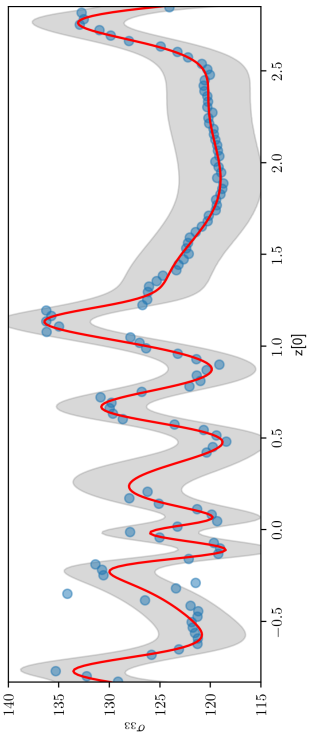


Figure 5.11: Resulting network prediction for dataset *1_5*: $\alpha_{init} = 1e^{-3}$, $\mathbf{w}_{MAP,tol} = 2e^{-6}$ & $Ev_{tol} = 1e^{-3}$. Final $\beta = 0.261$.

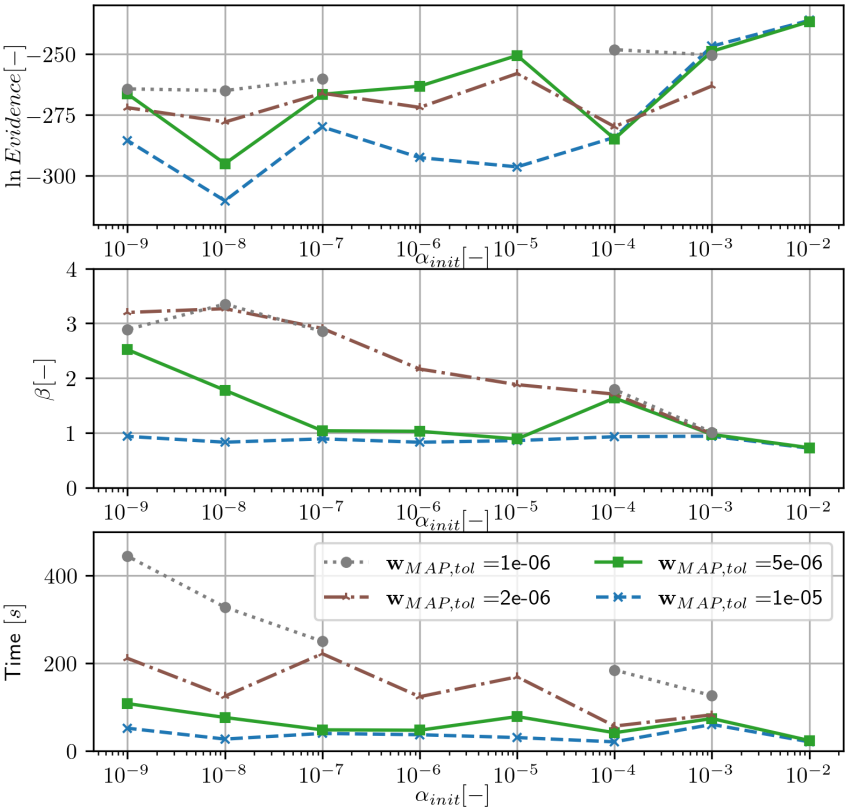


Figure 5.12: Parameter study for the *Expl* dataset with varying configurations. The network contains 27 hidden nodes ($W = 82$). $Ev_{tol} = 1e^{-3}$. For models whose points are missing from the plots, $\mathbf{w}_{MAP}$ was not found in $3e^{6}$ epochs and the training was stopped.

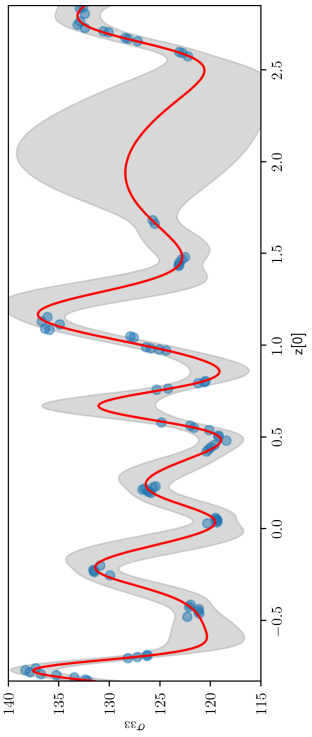


Figure 5.13: Resulting network prediction for the *Expl* dataset: $\alpha_{init} = 1e^{-3}$, $\mathbf{w}_{MAP,tol} = 2e^{-6}$ & $Ev_{tol} = 1e^{-3}$. Final $\beta = 0.985$.

size of the network, leading to significant differences. For dataset *1_10*, N is relatively small leading to W being severely constrained and many results are not able to capture the non-linearity in the data, leading to a low final β. If samples are added to obtain the *1_5* dataset, the network is less constrained without the dataset showing more non-linearity and a higher β, but to capture this significant computational effort is required. For the *Full* dataset, even when $W < N$ the BNN fully captures the behaviour, showing redundancy in the dataset and the network size is no longer an active constraint.

After around a value of $1e^{-5}$, decreasing α_{init} has no significant influence on model selection apart from leading to a higher β in the *Expl* dataset. For all datasets, $\alpha_{init} = 1e^{-2}$ gives the highest evidence, however this comes at the cost of either a decrease of β or a significant increase in computation time, analogous to Figure 5.5. The evidence tolerance Ev_{tol} is altered for fixed α_{init} & $\mathbf{w}_{MAP,tol}$ on the *1_5* dataset, and the result shown in Figure 5.14. The difference in evidence is not significant compared to changes in α_{init}, as long as it is not chosen too low to prevent long computation times. Not included in the graph is that for a low Ev_{tol} the hyperparameters can start oscillating at every update.

The formulas for re-estimating the hyperparameters are equivalent to equations derived by Expectation Maximization (EM) [32]. For EM it is proven that after every update the evidence must increase, which is not always observed during training of the BNN in this thesis. This discrepancy might be caused by the eigenvalue computation, or not being close enough to $\mathbf{w}_{MAP}$, and puts doubt on the exact value of the evidence.

The results on the datasets are shown above in the Figures 5.7,5.9,5.11 & 5.13, and show that the network is able to accurately represent different datasets. When a network cannot attain sufficient complexity (i.e. when a model mostly explained by noise is obtained), its number of nodes should be increased up to the limit $W = N$. Further sampling therefore allows the network to increase in complexity. In this thesis the number of samples is kept at a minimum, therefore the result of *1_5* is the most important. Based on the parameter studies, for the rest of this thesis the parameters are selected as $\alpha_{init} = 1e^{-4}$, $\mathbf{w}_{MAP,tol} = 5e^{-6}$ and $Ev_{tol} = 1e^{-3}$.

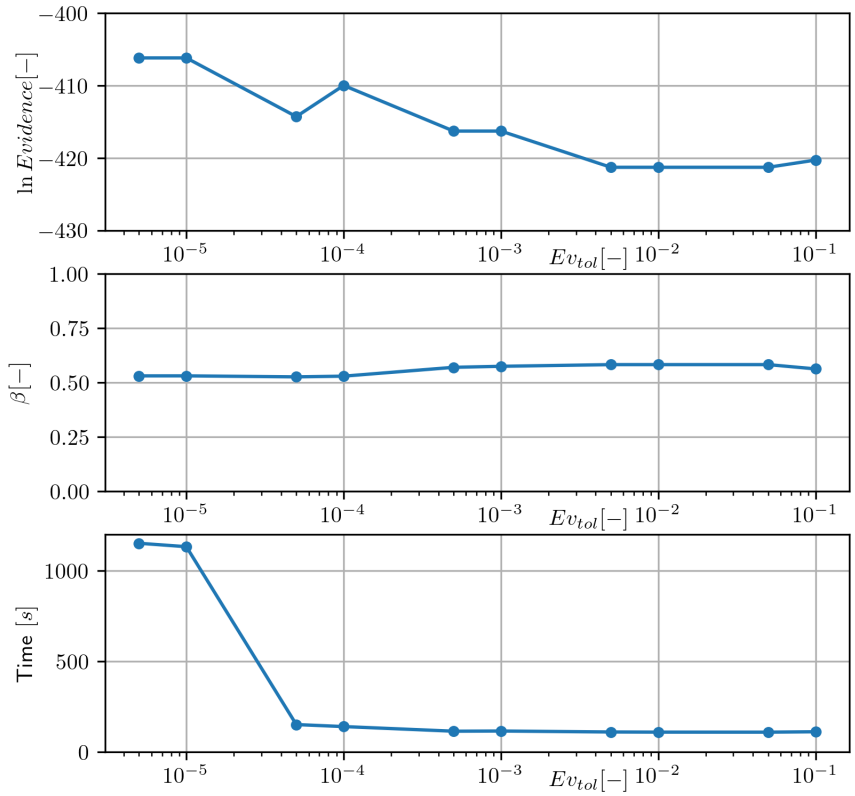


Figure 5.14: Parameter study for the *1_5* dataset for varying Ev_{tol}. The used parameters are: $\alpha_{init} = 1e^{-4}$, $\mathbf{w}_{MAP,tol} = 5e^{-6}$.

6 Bayesian optimization

The variational autoencoder reduces the dimensionality to a lower dimensional space. Finding the optimum within this reduced space requires an optimization method that ideally finds its global optimum with as little function evaluations as possible. A gradient-based method is likely to find a local optima in the reduced space, and a heuristics based method would require many function evaluations. Instead Bayesian optimization (BO) is used, where based on a limited number of initial samples a Bayesian neural network is trained. The prediction mean and variance can be combined to ignore regions where the network is sure the optimum isn't. Instead it can iteratively add informed samples in regions with high potential of containing the optimum.

6.1. Overview

Bayesian optimization (BO) is an approach to optimize problems with objective functions that are deemed too expensive to compute extensively. BO is designed for black-box derivative-free global optimization [48]. BO algorithms typically involve two primary components. The first is a method for statistical inference, generally a Gaussian process. In this case the Bayesian neural network presented in Chapter 5 is used. In theory Bayesian neural networks scale linearly with the number of data points instead of cubic for Gaussian processes. The result of this statistical inference is a prediction model with a mean and a variance. The second component, the focus of this chapter, is the acquisition function.

Based on the posterior distribution of a statistical inference model an acquisition function is created. When maximized, this acquisition function gives the next point in the design space to evaluate and add to the samples. It expresses a trade-off between exploring regions with high uncertainty, and exploiting regions with high mean values (even in high-certainty regions). Typical Bayesian optimization thus is summarized by iteratively adding points by computing the posterior distribution and then maximizing the acquisition function to add a new data point. This general overview is presented in Algorithm 3.

Algorithm 3 Bayesian optimization overview

Input: n initial samples of function f to be optimized
Exploration parameter
Output: Global optimum of f

1: **while** i < budget **do**
2: Determine the posterior distribution based on current samples of f
3: Maximize the acquisition function to find new point x_i
4: Evaluate $f(x_i)$ and add to current samples
5: **end while**
6: Return x_o for the maximum $f(x_o)$

6.2. Expected improvement

There are many types of acquisition functions, some created for specific applications and some more generally applicable. A non-exhaustive list includes expected improvement, probability of improvement, entropy search, predictive entropy search, knowledge-gradient and upper confidence bound [48]. Here one of the more common types, namely the expected improvement (EI) acquisition function, is considered.

$$EI(x) = \mathbb{E}max\left(f(x^*) - f(x^+), 0\right) \tag{6.1}$$

where x^* are the proposal parameters, each one corresponding to a different EI value. $f(x^+)$ is the current known best function evaluation, and therefore stays constant during a BO iteration. The expected improvement can be analytically derived using integration by parts resulting in [49]:

$$EI(x) = \delta\Phi(Z) + \sigma(x^*)\phi(Z) \tag{6.2}$$

where

$$\delta = \mu(x^*) - f(x^+) \tag{6.3}$$

$$Z = \begin{cases} \dfrac{\delta}{\sigma(x^*)} & \text{if } \sigma(x^*) > 0 \\ 0 & \text{if } \sigma(x^*) = 0 \end{cases} \tag{6.4}$$

and Φ and ϕ are the cumulative distribution function and probability distribution function of a unit Gaussian distribution, respectively.

Figure 6.1 shows a contour plot of EI based on different values of σ and δ. The value of δ, as in Equation 6.3, is the expected difference in quality between the proposed point and the best previously evaluated point. The EI increases both for a better optima and for a higher uncertainty.

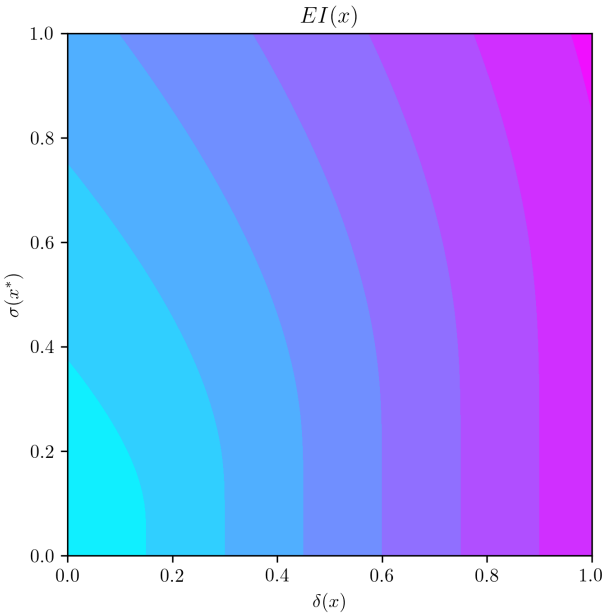


Figure 6.1: Exploration characteristics of $EI(x)$. Light blue corresponds to low EI values, purple to high EI. For increasing values of σ (uncertainty) and δ (quality of improvement) the EI increases.

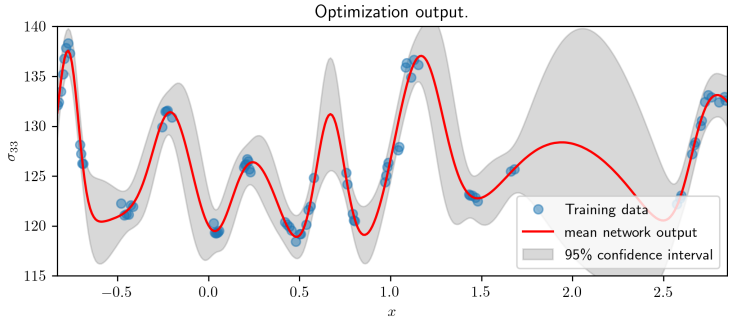
6.3. Exploration versus Exploitation

A typical resulting acquisition function based on a trained surrogate model is presented in Figure 6.2. In regions with few samples, a high uncertainty is observed in the prediction model. Regions of high-uncertainty and regions of high-mean both lead to peaks in the EI acquisition function. To get more control over the focus of either exploring regions of high-uncertainty, or focussing on regions with a lower uncertainty but a higher mean value, an additional parameter can be added to Eq. 6.3:

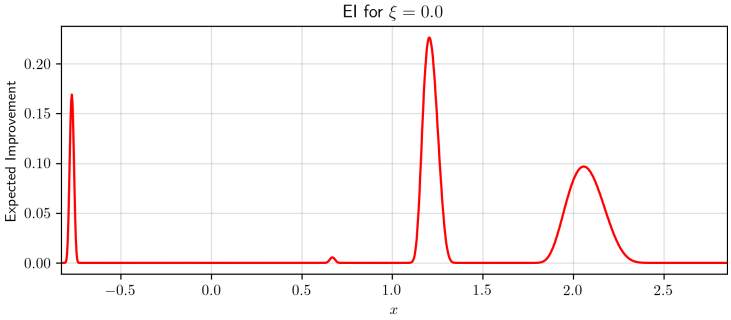
$$\delta = \mu(x^*) - f(x^+) - \xi \tag{6.5}$$

The parameter ξ determines the exploration versus exploitation trade-off. For higher values of ξ, regions of high-uncertainty get prioritized in the EI acquisition function.

In Figure 6.3a the EI is presented for $\xi = 0.5$, which is still relatively similar to the original result of $\xi = 0.0$. When ξ is further increased to 10.0 however, as visualized in Figure 6.3b, only the high-uncertainty region gives a peak. Jasrasaria and Pyzer-Knapp [50] provide an insightful discussion on including ξ as a hyperparameter, and introduce a model dependent formulation based on the mean variance. Between



(a) The BNN network prediction. The highest obtained sample is at $x = -0.77$, the predicted peak at this optimum is close to the predicted peak at $x = 1.16$. Around $x = 2.0$ there is no available data resulting in a large uncertainty of the prediction.



(b) The Expected Improvement acquisition function. Three main peaks are observed, two near values of high-mean prediction, and one in a high-uncertainty region.

Figure 6.2: The result of a statistical inference model, and the EI acquisition function derived from it.

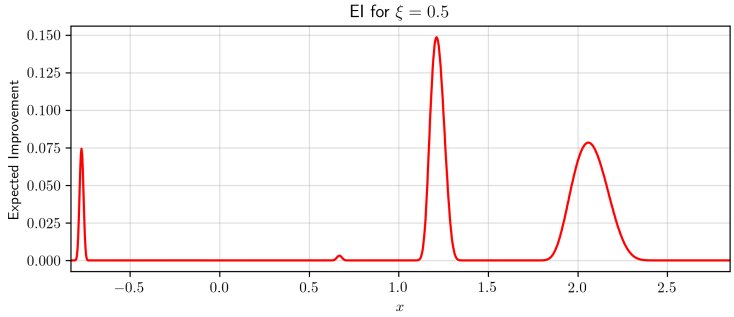
these plots the numerical values of the EI change significantly, however the magnitudes in these plots should not be compared with each other and the only goal is to find the optimal x value for its own specific ξ.

For the same prediction model, many acquisition functions have been computed with varying levels of ξ, and are plotted together in Figure 6.4 with the function color representing the EI value. This demonstrates how the EI function changes from exploitation to exploration with an increase of ξ.

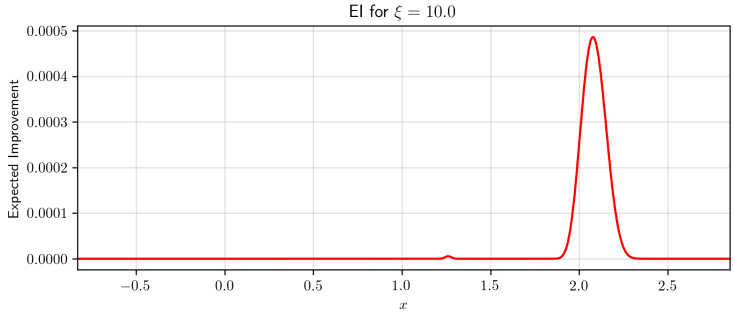
In Bayesian optimization, generally several iterations are performed of creating a prediction model, maximizing the acquisition function, evaluating the real function, and retraining the prediction model as presented in Algorithm 3. The exploration parameter ξ can be varied during these iterations. For example it can be opted to initially start with high ξ values to explore the uncertain regions, before moving to lower ξ values that focus on exploiting the function to obtain its maximum.

Depending on the function and the initial sampling, the number of iterations one should focus on exploring before moving to exploiting can be unclear. Therefore a cyclic scheme can be adopted, where each cycle consists of several exploration iterations with high ξ followed by several exploitation iterations with low ξ. Many of these cycles can be performed, and the BO scheme can be considered converged when the optimum has not improved after a number of cycles.

In order to plot the functions as is done in this section, the EI improvement function is computed for a large number of points along the x-axis. As each computation is relatively cheap, this is still feasible. However when the optimization function consists of multiple dimensions, brute-force computing the EI for every point becomes infeasible, and maximizing the EI function requires an optimization strategy. It can for example be opted to use a genetic algorithm for this purpose, which will be introduced in Chapter 7. Furthermore the EI acquisition function only provides useful predictions when the prediction model performs well.



(a) The Expected Improvement acquisition function for $\xi = 0.5$. The optimum is the same as in plot 6.2b, but the peaks in areas of high uncertainty have increased relatively.



(b) The Expected Improvement acquisition function for $\xi = 10.0$. With a high exploration parameter the focus of the BO is fully on exploring regions of uncertainty.

Figure 6.3: The EI results for different values of ξ. With an increasing ξ value, the importance of improvements to the mean μ decreases relative to the importance of potential improvements in regions of high-uncertainty, represented by large σ values.

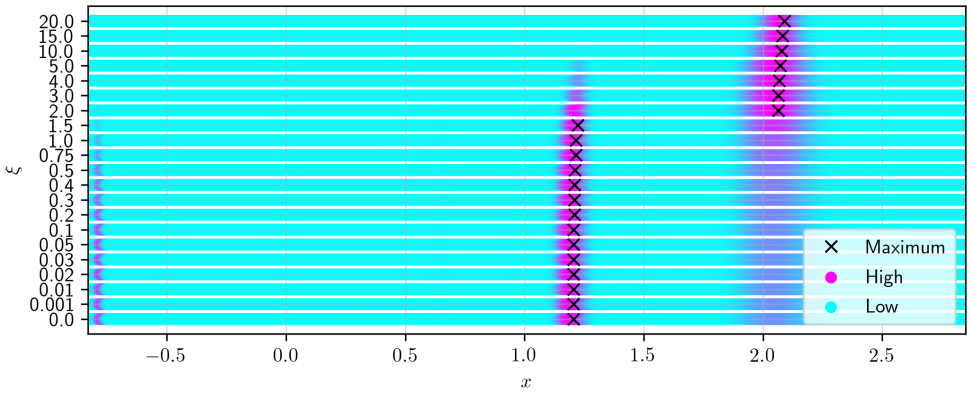
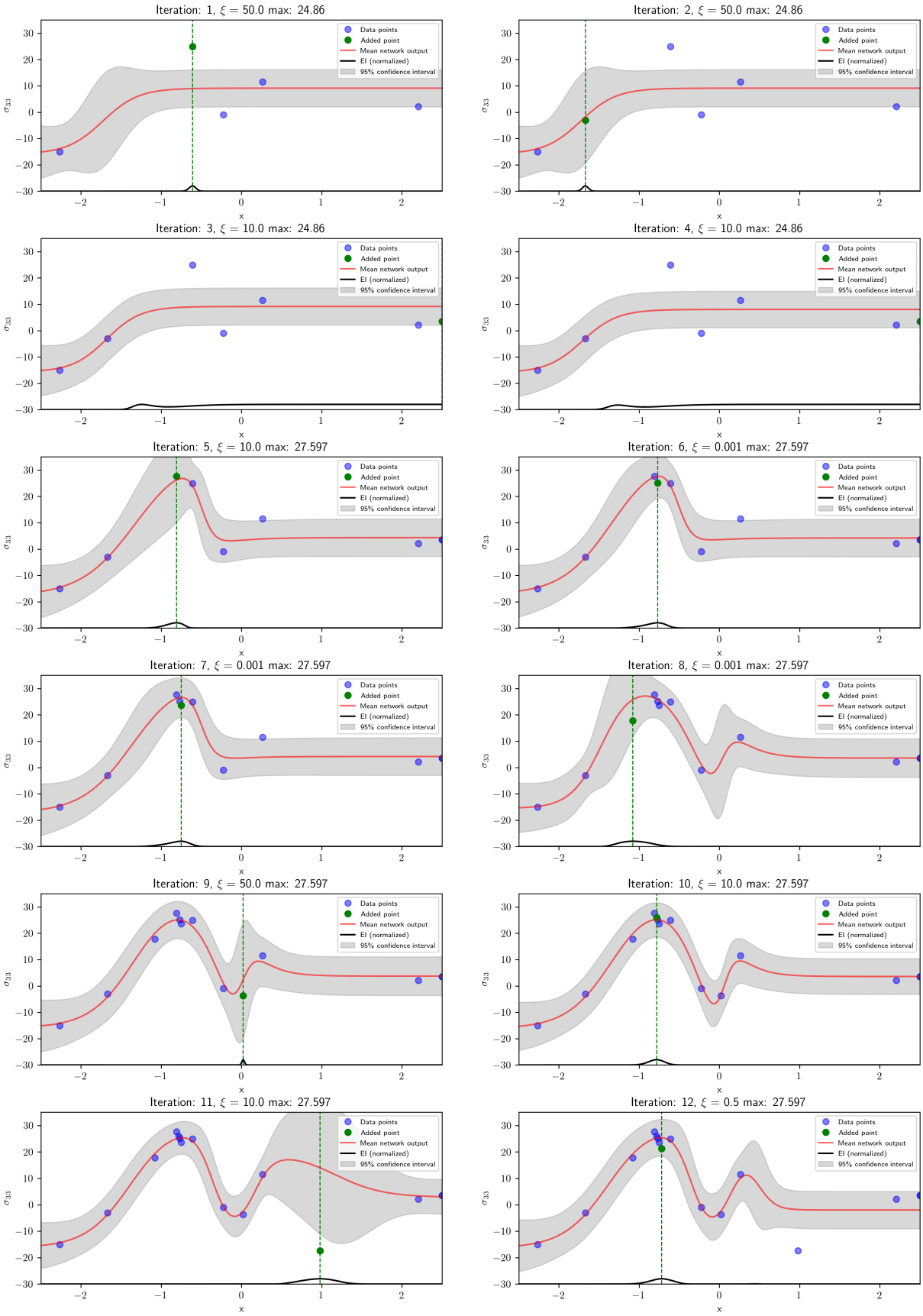


Figure 6.4: The resulting EI curves for varying values of ξ for the same statistical inference model presented in Figure 6.2. The EI is represented by the color, and the maximum for each parameter is presented. The values are normalized for each individual value of ξ. Note that the y-axis is not to scale.

6.4. Example result

An example result with several iterations of BO is presented in Figure 6.5. A cyclic ξ parameter is used, varying between 50.0, 10.0, and 0.001. The values for each iteration are included in the figure title. 4 initial points are used, each figure shows the data points obtained up to the current iteration, the prediction, the EI curve, and the actual computed function point. Between iterations the prediction model, the Bayesian neural network, is retrained.


Iteration: 1, ξ = 50.0 max: 24.86
Iteration: 2, ξ = 50.0 max: 24.86
Iteration: 3, ξ = 10.0 max: 24.86
Iteration: 4, ξ = 10.0 max: 24.86
Iteration: 5, ξ = 10.0 max: 27.597
Iteration: 6, ξ = 0.001 max: 27.597
Iteration: 7, ξ = 0.001 max: 27.597
Iteration: 8, ξ = 0.001 max: 27.597
Iteration: 9, ξ = 50.0 max: 27.597
Iteration: 10, ξ = 10.0 max: 27.597
Iteration: 11, ξ = 10.0 max: 27.597
Iteration: 12, ξ = 0.5 max: 27.597
Data points
Added point
Mean network output
EI (normalized)
95% confidence interval
x
σ_{33}

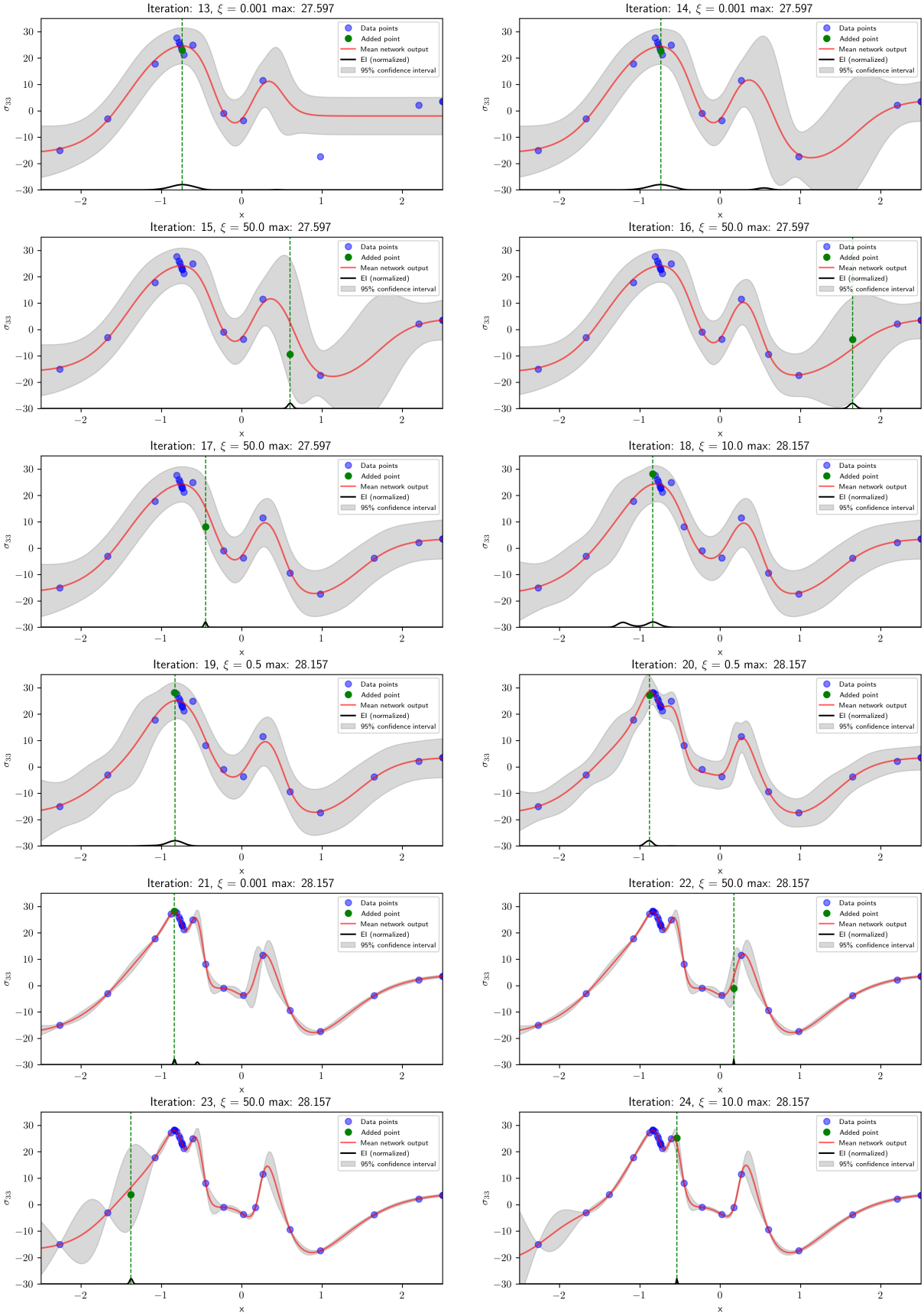


Figure 6.5: Several iterations of Bayesian optimization. The current available data points are included, as well as the network prediction and the (normalized) Expected Improvement at the bottom of each plot. The green vertical line represents the found optimum where the sample is added.

7 Genetic algorithms

A genetic algorithm (GA) is a metaheuristic optimization method. Where the neural networks presented earlier are inspired by the brain, genetic algorithms are also inspired by nature with the process of natural selection. It simulates a population over several generations with the aim of maximizing the fitness by selecting the fittest individuals.

Genetic algorithms do not require gradient information, optimize the global design space, and perform well in areas where a single function evaluation is relatively cheap as generally many evaluations are required. This is not the case for the optimization problem considered in this thesis. However, this makes them well suited for maximizing the acquisition function in Bayesian optimization, which is cheap to evaluate. The original design space is thus first encoded in a lower dimensional space. A prediction model is trained to create a computationally cheap surrogate model in this reduced space, and the GA presented here is used to optimize this surrogate model.

7.1. Algorithm overview

A genetic algorithm (GA) is a type of evolutionary algorithm introduced in the 1970s [3]. It is a heuristic search method, usually applied to optimize nonlinear functions, inspired by the process of natural selection. GA's are generally relatively easy to implement and have been used in a wide range of optimization problems. Since its first use, many variations and different types of GA's have been developed. Here a basic GA is introduced and implemented.

The goal of the numerical optimization is to find the parameters $\mathbf{x}$ inside the design space that maximizes a function $f(\cdot)$.

$$y = f(\mathbf{x}) \tag{7.1}$$

Initially a number of points in the design space are generated within defined boundaries, either randomly or in some other way spread throughout the design space. Each point $\mathbf{x}_i$ is considered an individual, and is evaluated to compute the actual function value y. Based on the function value, the fitness of each individual is computed. What follows is a process of selection where individuals with a high fitness are selected to create offspring. The offspring is created via a process of crossover, where the chromosomes of individuals are combined to form the individuals of the next generation. The idea behind this process is to search the design space in promising (high-fitness) areas. This process is repeated throughout multiple generations to iteratively find better solutions. This process is described in algorithm 4. A GA does not use derivative information about the function and is generally applied for nonlinear functions that are relatively cheap to evaluate. Several random processes occur within a GA, and it is not guaranteed to converge to a global optimum. On the other hand, it is less sensitive to local optima and provides a set of candidate solutions with a single run.

Algorithm 4 Overview of a simple genetic algorithm

```
1: Create an initial population of individuals
2: Evaluate the fitness of each individual
3: while not converged do
4:     Select the fittest individuals
5:     Create offspring using crossover to form a new population
6:     Evaluate the fitness of each individual in the new population
```

Some common terminology of GA's is introduced to assist the upcoming section on the implementation of a GA:

- **Individuals:** An individual represents one 'entity', it corresponds to a point x_i in the design space associated with a value for the objective function $f(x_i)$ used to compute its fitness relative to the rest of the population. It is generally given an encoded representation referred to as chromosome.
- **Fitness:** The fitness of an individual is computed based on its associated value of the objective function $f(\cdot)$, such that higher function values correspond to a higher fitness. The fitness determines the probability of being selected for crossover, the equivalent of creating offspring.
- **Chromosome:** A chromosome is an encoding of an individuals input value that allows for multiple individuals to be 'combined' to form offspring. A range of different types of chromosomes is possible, examples being binary values, continuous values or letters. A common simple representation is as a binary value.
- **Population:** The population is the set of individuals that make up the current generation.
- **Generation:** Generations are analogous to iterations of the whole population. To iteratively find better individuals, multiple generations are performed where each population is created as offspring from (high-fitness) individuals of the previous generation.
- **Selection:** Selection is the process with which the fittest individuals of a population are selected to create offspring. Different types of selection are possible such as fitness proportionate selection, rank selection, tournament selection and elitism selection.
- **Crossover:** Crossover is the method used to combine chromosomes of selected individuals to form chromosomes for individuals of the next generation. Many types of crossover exist, largely depending on the type of chromosome used. For chromosomes using binary values, commonly used types of crossover are (N-)point crossover and uniform crossover [51].

7.2. GA implementation

A GA is implemented with the aim of finding the point in a multi parameter design space that maximizes a function evaluation. The design space consists out of continuous values, and is encoded in a binary chromosome. In this encoding the space is discretized based on the number of bits in the chromosome.

7.2.1. Encoding

Converting a numerical design space value between x_a and x_b into a chromosome of length l discretizes the design space into 2^l points. Any chromosome of bits can be converted into its binary value b with range 0 to $2^l - 1$. This is then normalized between the boundaries: $x_i = x_a + (x_b - x_a)\frac{b}{2^l - 1}$. From this numerical value in the design space the function can be evaluated using equation 7.1. Depending on the range of this function the outputs should again be normalized to form the fitness of the individual. For multi-dimension design spaces this is done separately for each dimension, followed by a concatenation of the binary representations. This process is visualized in figure 7.1

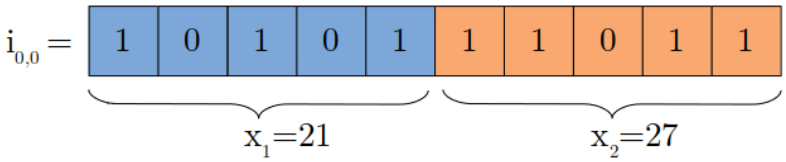


Figure 7.1: Example of 2 parameters being concatenated into a single chromosome for an individual.

7.2.2. Fitness proportionate selection

Fitness proportionate selection, also known as roulette wheel selection, is implemented to select individuals. All individuals get a probability of being selected (p_i) based on the proportion of their fitness (f_i).

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j} \tag{7.2}$$

The selected individuals (parents) are randomly combined to form offspring. The number of selected parents depends on the type of crossover used. For certain schemes two parents are 'interchanged' to create two new individuals, whereas in others two parents create only one new individual. In certain situations combining chromosomes of more than two parents can be advantageous [52].

7.2.3. Uniform crossover

A uniform crossover scheme is implemented where two parents are selected to create offspring for the next generation. It follows from this that for P new individuals, $2 * P$ individuals of the current population should be selected. In uniform crossover, every $k-th$ bit is randomly selected from the $k-th$ bit of one of the parents. Uniform crossover does not guarantee the binary value of offspring to be in-between that of its parents, as can easily be imagined with two parents 100 and 010 producing offspring 110 or 000.

7.2.4. Mutation

After crossover is performed, a small probability of a random flip of bit is introduced, termed a mutation. In the case that the $k-th$ bit in both parents is the same, a mutation at this point could mean that the offspring still has the opposite bit here. This is analogous to a mutation in biological DNA.
Where crossover promotes new individuals to explore high-fitness areas, mutation can be used for exploring unexplored regions. There is no consensus on the ideal mutation rate. Where some suggest a rate of mutation of $\frac{1}{L}$, with L being the length of the chromosome [53], often times this rate should be tuned together with other hyperparameters.

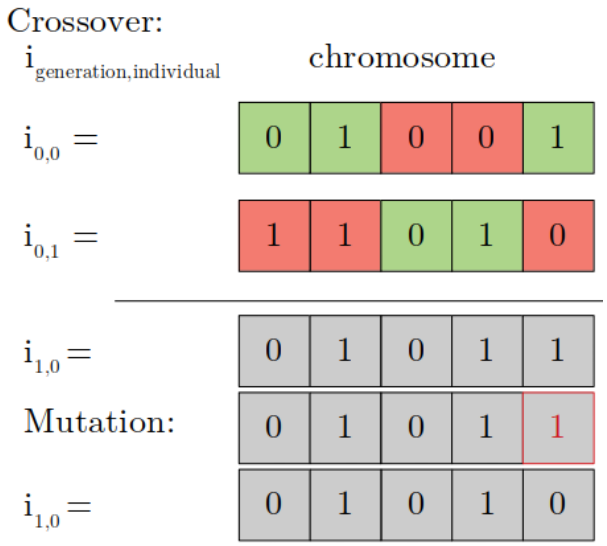


Figure 7.2: Example of two selected individuals undergoing uniform crossover to generate an individual in the next generation. Followed by a mutation in the new individual.

In this thesis, optimization problems with different levels of dimensions and different lengths of chromosomes will be used, leading to very different mutation rates. In an attempt to make the mutation rate between problems more comparable, an effective mutation rate is computed based on the length of the chromosome:

$$m_{eff} = m \times L = m \times D \times L_x \tag{7.3}$$

where m is the mutation rate and L is the total length of the chromosome computed by multiplying the dimensionality D with the length per dimension L_x. The effective mutation rate now provides the probability of a mutation in an individual, rather than per bit. It is not claimed here that two different lengths of chromosomes with the same effective mutation rate behave the same, this has not been studied. Instead the purpose of defining it this way is to provide a ballpark figure on the number of mutations when comparing two different optimization problems

7.2.5. Elitism

To prevent loss of the fittest individuals (elites), some of the fittest individuals can automatically be selected into the next generation without applying a mutation. This process is known as elitism. Elite individuals should still be included in crossover to allow for exploring the surrounding region it.

7.3. GA Results

To illustrate how a GA works in practice and to be able to compare different parameters, an analytical function is created wich is optimized using a GA.

$$y = \sin(4\pi x+3)+cos(2\pi x)+\frac{1}{3}(2-x^2)+1.5\,(\mathcal{H}(x+0.7)-\mathcal{H}(x+0.5))+2.3\,(\mathcal{H}(x-1.4)-\mathcal{H}(x-1.5)) \tag{7.4}$$

Where $\mathcal{H}$ represents the Heaviside function. This function is highly non-linear and has multiple local maxima. The global optimum is at $x = 1.4$.

The fitness f of an individual i is computed without prior knowledge of this function by normalizing the function values based on all values in the current population:

$$f_i = \frac{y_i - y_L}{y_H - y_L} \tag{7.5}$$

where y_L and y_H are the lowest and highest y value of the current population respectively. This way the least-fit individual has 0 probability of being selected. Optionally an additional term can be introduced for an exploration focussed fitness f_e:

$$f_{i,e} = f_i(1 - f_i)\zeta \tag{7.6}$$

where ζ is an exploration parameter between 0 and 1. The least-fit individual now has a ζ probability of being selected compared to the fittest individual. Unless mentioned, ζ is 0.

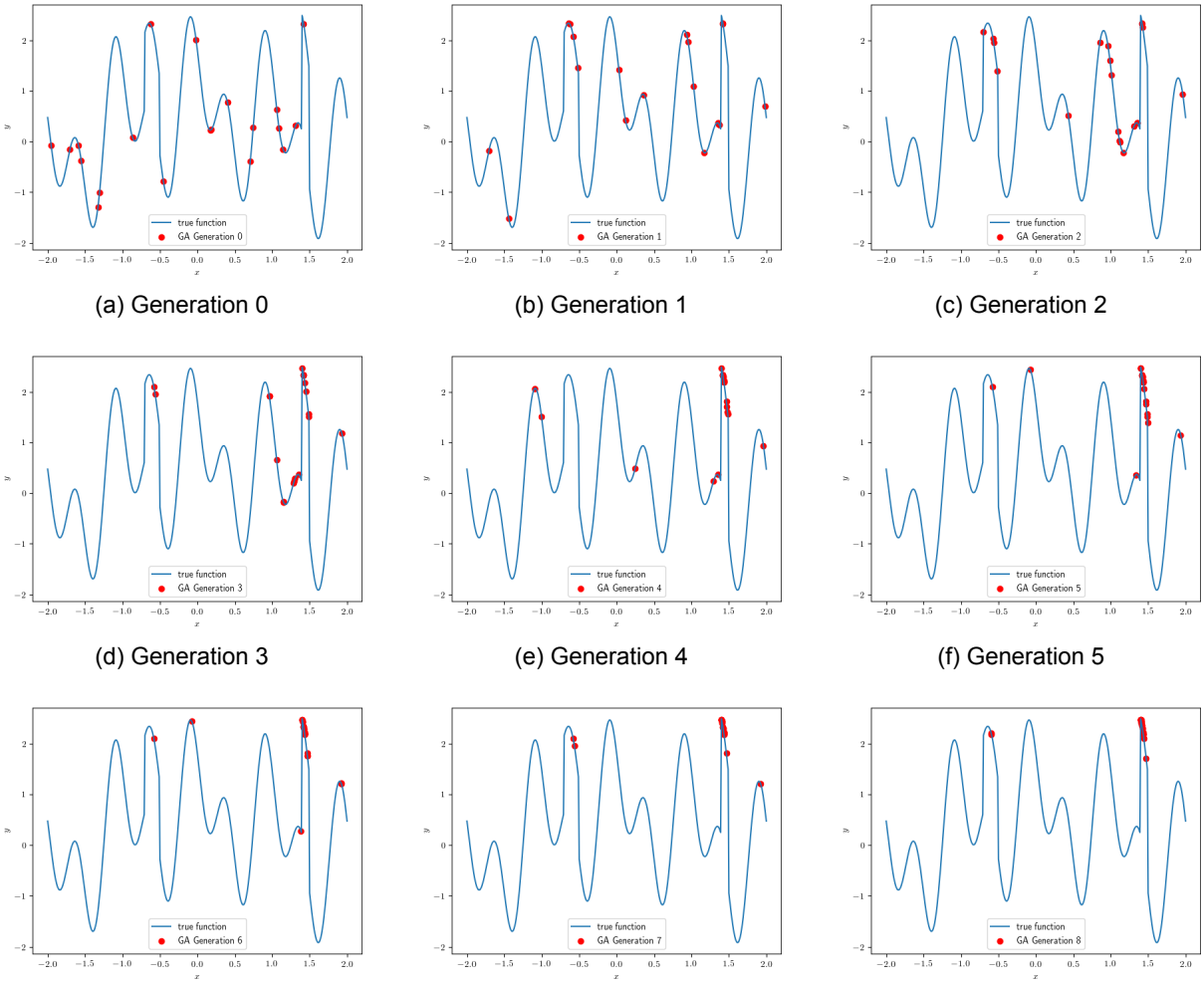
The performance of a GA with a population size of 20 individuals, 2 elites and a mutation rate of 0.01 is given in Figure 7.3. In this case the initial population has an individual two individuals very close to this optimum, however one has a much higher fitness than the other. If the right-most individual would not have been there, there is a significant probability that a different local optimum would have been found.

7.3.1. Model selection

To compare the influence of different hyperparameters, many GA's have been trained to optimize the function in Eq. (7.4). With a chromosome length of 12, a mutation rate of 0.0833 would give an effective mutation rate equal to 1. Each GA is created such that a total of 160 function evaluations would be required (ignoring elitism). In a full optimization a convergence criteria could be applied to stop the GA after the optimum did not increase for a number of generations. As a baseline comparison, the result of creating random samples is presented in Figure 7.4. GA's with different configurations of population size, number of generations, elites, and mutation rate are evaluated 10 times. An example result for a single configuration is plotted in Figure 7.5. From the maxima of these different runs the mean and standard deviation are taken, and the results for each configuration are presented in Figure 7.6.

A much more extensive parameter study could be performed here including many different functions, but that is not the main goal of this study. The main trends observed are expected to carry over to all functions. In all cases, some form of elitism results in increased performance. With elitism, having some mutation rate increases the performance. A high mutation rate with some elitism is almost equal to generating random samples and the result is similar to that of Figure 7.4. The average population fitness decreases significantly as the mutation rate increases, further supporting this. A small population for many generations leads to less optimal results unless a high mutation rate is selected, giving similar results to random. No significant difference is observed between $\zeta = 0.0$ and $\zeta = 0.2$.

From this result it can be concluded that some mutation and elitism is beneficial for optimization. The mutation rate should be kept relatively small to prevent the population from being near random, and the average population fitness can serve as an indication for this.



(a) Generation 0 (b) Generation 1 (c) Generation 2

(d) Generation 3 (e) Generation 4 (f) Generation 5

(g) Generation 6 (h) Generation 7 (i) Generation 8

Figure 7.3: Example of the result of the GA, starting from a random population for 8 generations. The population contains 20 individuals with a chromosome length of 12. A mutation rate of 0.01 is applied and 2 elites are selected each generation.

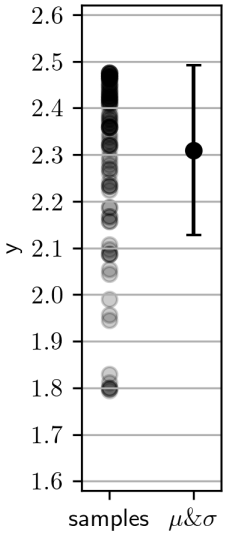


Figure 7.4: Generating 160 random samples and plotting the best gives a single dot. This is done 100 times and the mean μ and standard deviation σ are given.

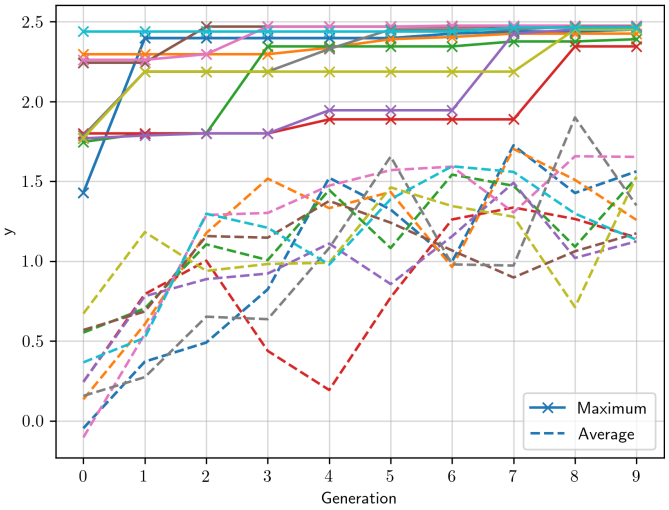


Figure 7.5: The maximum and average of 10 runs of a GA all with the same parameters, for different random initial populations. The GA properties are 10 generations with a population size of 16, 2 elites and a mutation rate of 0.10. The average shows a clear increase and all runs get close to the maxima, leading to a small variance in obtained optimum.

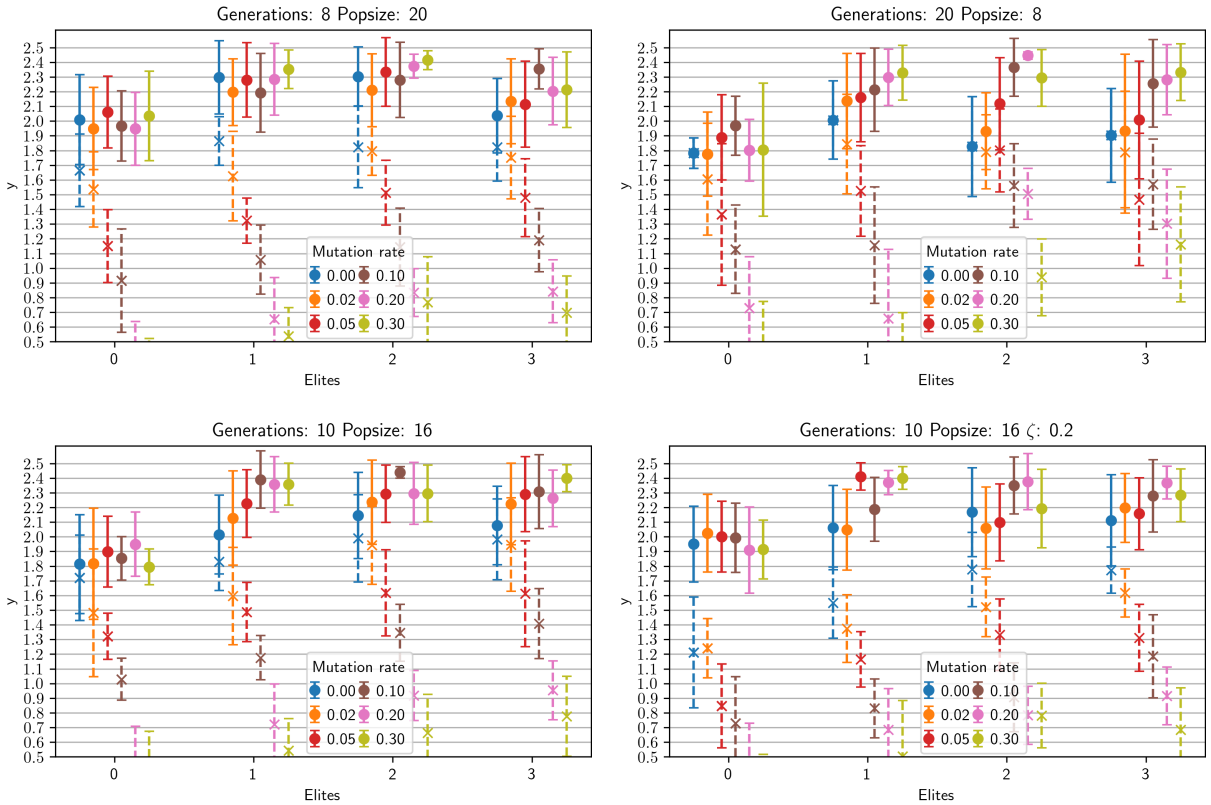


Figure 7.6: Results of GA performance for varying parameters, each configuration performed 10 times to get a mean and standard deviation. Both the maximum (continuous) and average (dashed) are plotted. Parameters altered are the number of generations, population size, mutation rate, number of elites, and ζ. The effective mutation rates are: 0.00, 0.24, 0.60, 1.20, 2.40, 3.60.

8 Framework results and discussion

In this chapter all individual techniques introduced in the previous chapters are combined into a single framework. This framework is used in optimizing the arrangement of fibers in a microstructure to maximize mechanical performance. This is a problem with many variables and where for each evaluation a Finite Element simulation is required.

The framework uses Bayesian optimization (BO) in the latent space of a variational autoencoder (VAE). BO consists of a Bayesian neural network (BNN) and the expected improvement (EI) acquisition function. This acquisition function is maximized using either a grid search for a one-dimensional latent space or a genetic algorithm (GA) for a higher-dimensional latent space. This complete scheme is presented in Figure 8.1.

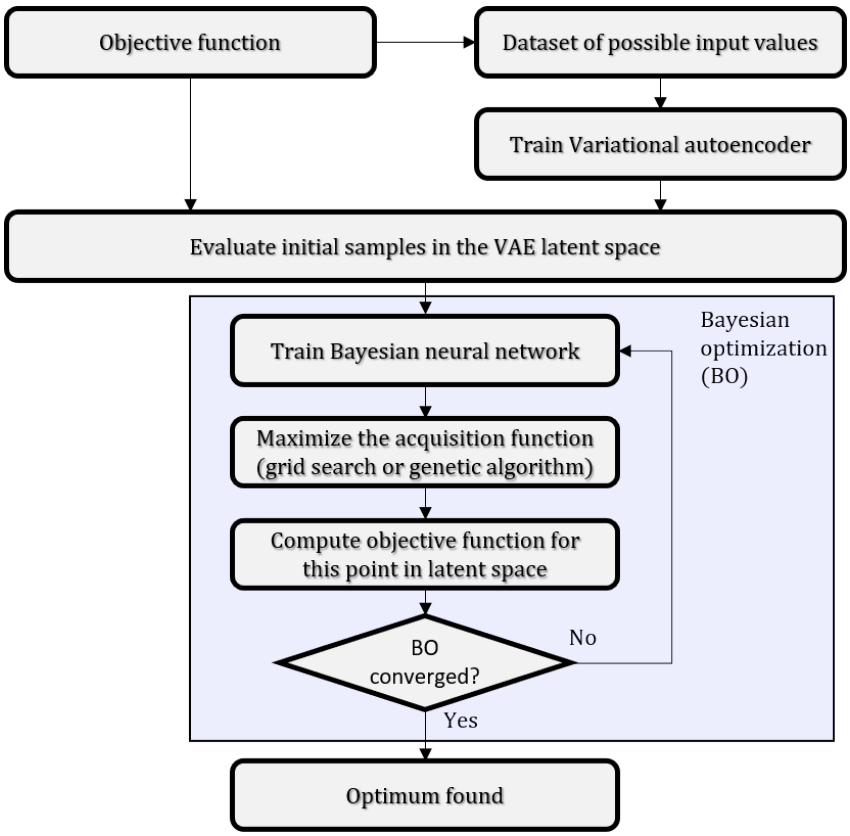


Figure 8.1: Full scheme of the optimization framework. The full design space $\mathbf{x}$ with dimensionality D is encoded in a M-dimensional latent space $\mathbf{z}$ where $M \leq D$ using the variational autoencoder. A prediction model, here a BNN, estimates $\mathbf{z}$, and is transferred to an acquisition function EI. This EI function (dimensionality D) can be optimized using a genetic algorithm, further encoding the $\mathbf{z}$ space into a binary representation. When the objective function is computed, the VAE decoder network is used.

After an explanation of how the experiments are run, the results of the framework are presented. The results are compared to a number of alternative optimization techniques.

The methods are first compared on an analytical function where the overall performance is measured. Following this, they are applied in optimizing the fiber geometries.

8.1. Design of experiments

The aim of the experiments is to demonstrate the performance of the various optimization methods. Both high and low dimensional problems are considered. The optimization techniques used as comparisons are a gradient based method, namely the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm, a GA and applying BO in the full space.

8.1.1. Optimization problems

Aiming to demonstrate the capabilities of the various techniques, multiple optimization problems have been defined. To study the overall performance of the methods, two analytical problems are used for which function evaluations are computationally cheap, the gradients are easily evaluated, and the optimal solution is known. Both of these have a two-dimensional full space, but they have varying levels of complexity. Following the analytical functions, the mechanical problem introduced in Chapter 2 is optimized. First a 2-fiber case without morphing (a 4-dimensional problem) is studied. Here the influence of randomness and hyperparameters in the framework is studied. This is followed by a 5-fiber case with morphing, resulting in 15 dimensions, with an additional focus on the mechanical properties.

8.1.2. L-BFGS-B

The Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm [1] is an efficient gradient based non-linear optimization algorithm. Here a limited memory (L-BFGS) version is used that can also handle box (L-BFGS-B) constraints. An implementation from SciPy, an open source Python library for scientific computation, is used.

The L-BFGS-B implementation is used to optimize in the full design space. When the design space has many local optima, the initial starting point of L-BFGS-B has a significant influence on its result, making a comparison to global optimization techniques difficult. It is therefore opted to consider an ensemble of L-BFGS-B optimizations as a single optimization run, starting from a Latin hypercube (briefly explained in Appendix D). The L-BFGS-B method requires the gradients to be computed at every evaluated point. For the mechanical problem, the gradients can not be derived analytically. Approximating the gradients using a finite difference scheme proved inconsistent even for small step sizes, it is suspected that this is due to small changes in geometry causing discrete changes in the meshing but not further studied. For this reason the L-BFGS-B method is not applied on the mechanical problems. For the analytical functions the gradients are derived up to machine precision using automatic differentiation implemented using the TensorFlow Python library.

8.1.3. Genetic algorithm

For the GA a single run provides multiple candidate optima within the global design space. The hyperparameters are based on findings of the GA parameter study from Chapter 7, and are presented together with the results. The initial population is created randomly. The number of samples should increase exponentially to maintain equivalent density for an increase in dimensions. This however is infeasible, resulting in the populations of higher-dimensional optimization problems having a severly lower density compared to low-dimensional optimization problems. The GA is considered converged when the optimum has not improved in 10 consecutive populations.

8.1.4. Bayesian optimization in full space

Bayesian optimization (BO) in the full space creates a surrogate model using the Bayesian neural network (BNN) based on an initial dataset. The expected improvement acquisition function based on this prediction is maximized using a GA. Both BO and the GA require exponential scaling of their initial samples when the dimensionality increases to maintain equal density, but this is infeasible. For higher dimensional problems this could lead to sub-optimal prediction models, and the GA not finding the global EI optimum. Recall that this manifestation of the curse of dimensionality is what motivated the hybrid VAE-BO framework in the first place. As a compromise, the initial samples of the BNN are instead selected in a Latin hypercube manner. BO is performed in cycles, where each cycle a number of iterations

are performed in the order of the following exploration ζ values: [50.0, 50.0, 10.0, 10.0, 0.5, 0.001, 0.001], the effect of different values is studied in Section 6.3. When the optimal point has not changed in three consecutive cycles the BO scheme is considered converged.

During training it was observed that often the prediction model with the highest evidence had a very low complexity, associated to a low β value. As it is known that the considered functions do not have any noise, this β is bounded to a minimum of 0.1. This results in allowing a higher network complexity, even if it leads to models with lower evidence. This is a pragmatic choice made in order to enable the optimization procedure. A different problem observed during the results was the BNN update getting stuck and failing to converge to $\mathbf{w}_{MAP}$. Although a more thorough investigation is required, it is suspected that this is a consequence of some weights being very small. Dividing the gradients by these low weights therefore results in too high values to reach convergence. This problem is remedied by considering $\mathbf{w}_{MAP}$ to be reached if the number of epochs reached $3e^6$ more than that of the previous $\mathbf{w}_{MAP}$ convergence.

When a datapoint is added using BO, the BNN can update its prediction by retraining, that is finding $\mathbf{w}_{MAP}$ again starting from $\mathbf{w}_{MAP}$ of the previous iteration. Retraining generally requires less computational effort compared to starting with a random weight initialization. However, as discussed in the results of the BNN, the number of parameters in the BNN should be as close as possible to its bound, which is the number of samples. As samples get added, the BNN network size is therefore increased by adding nodes to its hidden layer. This requires the BNN training to be reset. For networks with small initial sizes (i.e. 1-4 initial hidden nodes based on 4-16 initial samples) the network size is increased as soon as possible. For larger initial networks (i.e. ≥ 20 initial hidden nodes based on ≥ 100 initial samples), this restarting is delayed to when 3 hidden nodes can be added simultaneously, reducing computational effort.

8.1.5. Bayesian optimization in reduced space

BO in a design space reduced by a variational autoencoder (VAE) aims to reduce the issues of the scaling of the initial BO samples and the difficulty in finding the optimum of the EI. In a single dimension, the EI is simply maximized using exhaustive grid search as it is computationally cheap to do for a single dimension. For more than one dimension, it is maximized using a GA similar to BO in full space. The rest of the arguments and settings presented in the previous section on BO in the full space also hold for the reduced space.

8.1.6. Initial sampling

If a 1-dimensional design space is sampled at 4 points, a 15-dimensional design space requires $4^{15} = 1073741824$ samples for equal density. This is the problem that has motivated the work in this thesis, but still provides difficulties in using consistent comparisons to the alternative optimization methods. The variational autoencoder is also still susceptible to this exponential growth in number of samples. There are several possible ways of addressing this issue. One option is to use a budget, that is a fixed number of iterations, where each optimization technique is stopped when this budget is reached. In high-dimensional optimization problems for the optimization techniques that scale with dimensionality, this could result in all samples being required in the initial sampling, and no real optimization taking place. Alternatively, one can opt to allow the techniques that scale with dimensionality to use more samples, ideally in a consistent way relative to the lowest sampled method. Given the exponential nature this can in practice result in the sampling still being significantly less then required for equal density compared to the lowest dimension. Still, using more samples likely increases the performance of those techniques, and the results can than be used to assess the differences in obtained optimum as well as the differences in required evaluations. It is opted to use this second method. In high-dimensional problems where the ideal case cannot be realised, the number of samples is taken as large as possible while being feasible for computation in this thesis.

8.2. Analytical function

The optimization methods are compared on an analytical benchmark function. It is opted to use Shubert 3 [5], a continuous non-linear non-convex function:

$$f(\mathbf{x}) = f(x_1, ..., x_n) = \sum_{i=1}^{n} \sum_{j=1}^{5} jsin((j+1)x_i + j) \tag{8.1}$$

Where n can be any number of dimensions. This function is chosen due to its many local minima. The function is also continuous, allowing the gradients to be evaluated straightforwardly. By using this benchmark function many evaluations can be performed with cheap computational effort. The variables in this function are independent. A common boundary for which to optimize this function is $[-10.0, 10.0]$, as is visualized for two dimensions in Figure 8.2. Within this boundary it has 3^n global optima, the number of local optima also scales exponentially with a higher base value. To compare the various optimization techniques on different complexities of optimization functions, the function is optimized in two domains. Here considered to be the complex case is between $[-10.0, 10.0]$, and a simpler case by limiting the boundary to $[-1.5, 0.0]$ for every variable. This simpler case has a single global optimum and is presented in Figure 8.3. Both functions are optimized for two dimensions. For 2 variables, the minimum of this function is: -29.6733337. For consistency with the results of maximizing the mechanical performance later in this chapter, the results for this analytical case are presented as maximizing the negative objective function.

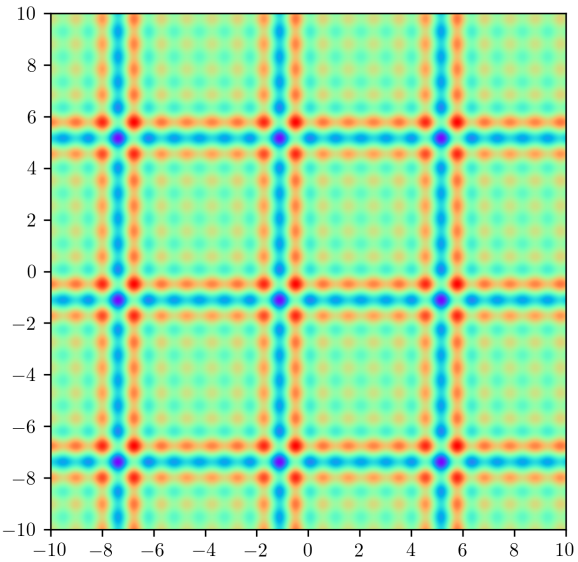


Figure 8.2: Two dimensional design space for the Shubert 3 function to be minimized, bounded between $[-10.0, 10.0]$, blue and red correspond to a low and high function value respectively. The function has 9 equal global optima.

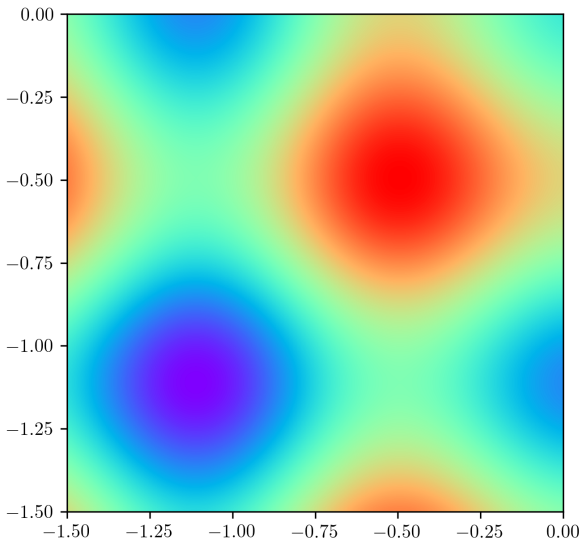


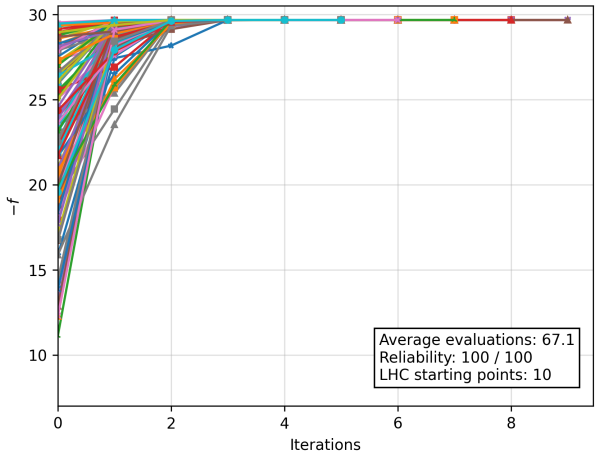
Figure 8.3: Two dimensional design space for the Shubert 3 function to be minimized, limited to $[-1.5, 0.0]$, blue and red correspond to a low and high function value respectively. The optima is at $x_i = -1.1141$ for both variables.

8.2.1. Simple case

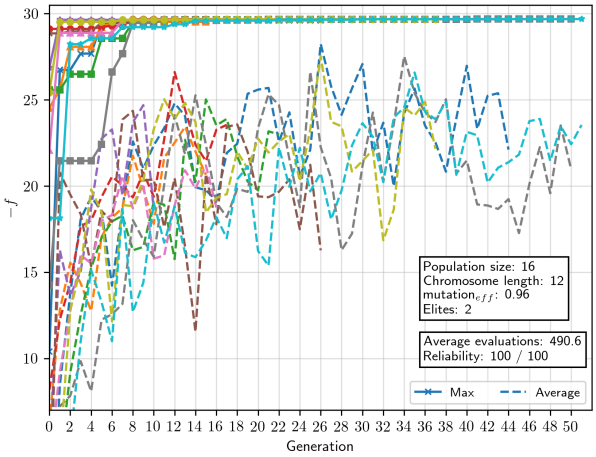
The simple case has four local optima, and a single global optimum. The results using all discussed optimization methods are presented in Figure 8.4. Starting at Figure 8.4a, the result using the gradient based method L-BFGS-B is given, where each curve represents the best result of the maximum of an ensemble of 10 runs starting at different initial points. With x curves on the plot, the problem was therefore optimized $10x$ times, from x Latin hypercube configurations. Based on the objective function in Figure 8.3, it can understood that any parameter starting in the area between around $[-1.5, 0.5]$ finds the minimum. The probability of both variables starting in this space is significant, and with 10 initial starting points always reached. The average required function evaluations for the combined 10 runs is 67.1. The GA, presented in Figure 8.4b, also reliably finds the optimum in the 100 runs performed. As

the GA starts with more initial samples, the optima in the first generation are already relatively high. It requires a significant amount of evaluations, on average 491, to converge to its optimum. BO in the full space also reliably finds the optimum, with on average 106.1 function evaluations. The additional training time of the BNN is on average 307.4 seconds. It finds the optimum with relatively few iterations, but requires several more iterations to converge.

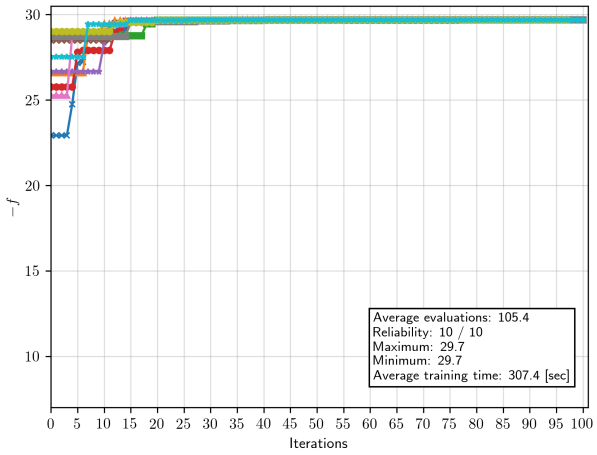
When applying BO in the reduced space of a VAE less initial samples are used, as the dimensionality is also lower. All runs use a separate VAE trained on 512 samples, the average training time of which is 335 seconds. The BNN training time is on average 2702 seconds. The latent space for every VAE is optimized within $z = [-2.5, 2.5]$. It is observed that the optimum is not reliably found, which can result from the VAE not capturing the optimum of the full space, the BO not capturing the optimum of the reduced space, or a combination of both.



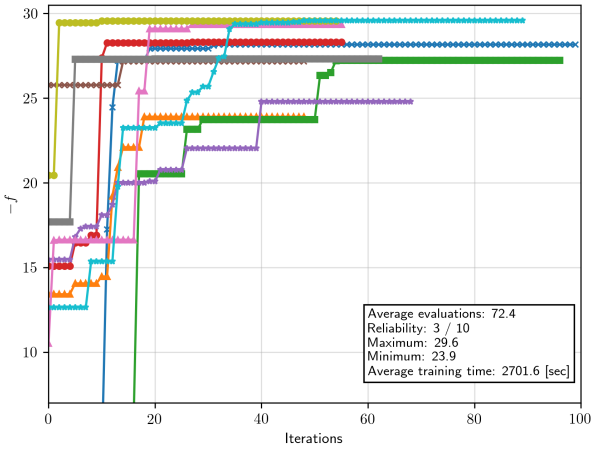
(a) Result of 100 BFGS ensembles, where each consists of 10 individual BFGS runs.



(b) Result of 100 GA runs, for clarity only 10 are plotted. The mean improving indicates that it is not simply sampling many random points. The number of evaluations is initially 16, with 14 added every generation.



(c) Results of BO applied in the full space. 16 initial starting points are used, every iteration adds a sample.



(d) Result of BO applied in a reduced space from 2 to 1 dimensions using a VAE. 4 initial starting points are used, every iteration adds a sample.

Figure 8.4: Results of the different optimization methods on optimizing the Shubert 3 function for two variables within the domain of $[-1.5, 0.0]$. The optimum is considered found, and added to the reliability, when a run finds a position within 0.5 of the known optimum.

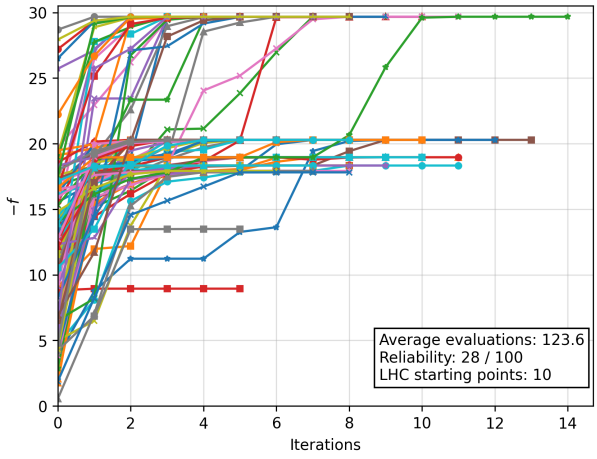
In this case the BFGS reliably finds the optimum with the minimum function evaluations. The GA takes a considerable amount of generations to converge, leading to a high number of evaluations. The BO predicts the optimum relatively quick as well. The combination of BO and a VAE does find a much better solution compared to its starting point, but does not reliably capture the optimum.

8.2.2. Complex case

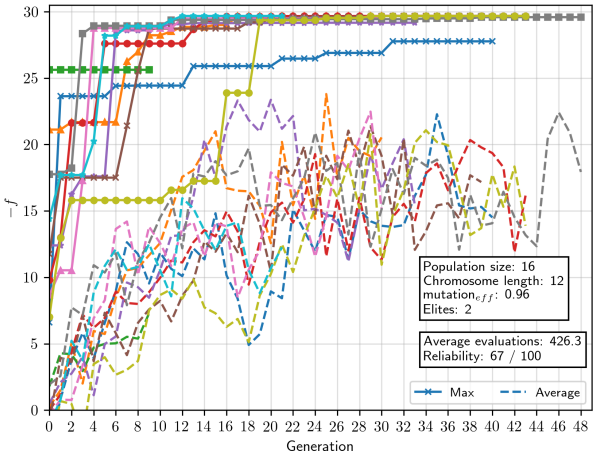
While the dimensionality of any optimization problem plays a significant role in the ability of optimization schemes to find the optimum, a different important factor is the complexity of the design space. The Shubert 3 function between bounds $[-10.0, 10.0]$ has the same optimal value as the simple case, but has many more local optima. Similar settings for the optimization techniques are used as for the simple case. In addition, the techniques using BO have an extra case where their number of initial samples is matched. The level of complexity of a function to be optimized is generally unknown beforehand.

The results of the optimization methods are presented in Figure 8.5. Compared to the simple analytical function the performance of all methods decreases significantly. The result of BFGS, in Figure 8.5a, indicates that the ensemble of 10 runs is no longer sufficient to reliably find the global optimum. Furthermore, there are more evaluations required before convergence. The GA performance is the most reliable of all the optimization techniques as presented in Figure 8.5b. In 69 out of 100 evaluations it finds the optimum. The number of evaluations has decreased compared to the simple function. BO in the full space performs significantly worse compared to the simple case. This likely is due to the BNN not able to make accurate predictions based on the sampling. Any prediction can therefore lead to a near random sample being added. With less samples the results even seem to be better, but this is likely do to randomness and would vanish when more runs are performed. Drastically increasing the number of initial samples is expected to allow the BNN to create more accurate predictions. However this would require a prior knowledge of the complexity of the function, and this is not studied here.

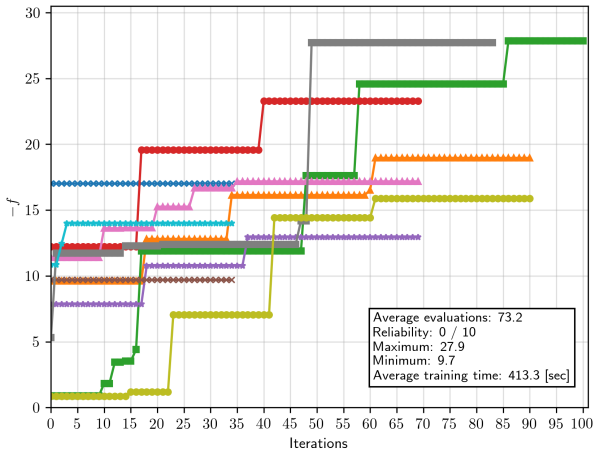
Similar to the other methods, the result when adding a VAE is also significantly worse compared to the simple case. Increasing the number of initial samples to match that of the GA and BO does not provide a significant difference. Similar to the BO, it is expected that if the number of samples were to be increased drastically there would be a noticeable difference.



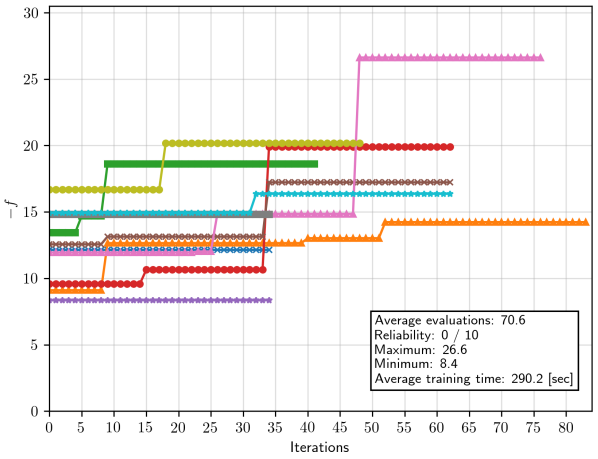
(a) Result of 100 BFGS ensembles, where each consists of 10 individual BFGS runs with initial points determined using a Latin hypercube.



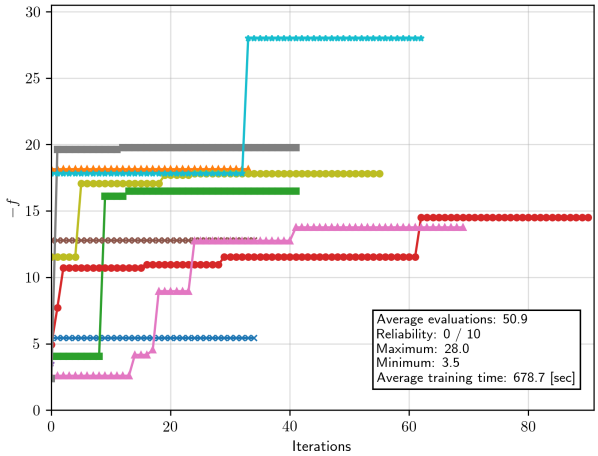
(b) Result of 100 GA runs, for clarity only 10 are plotted.



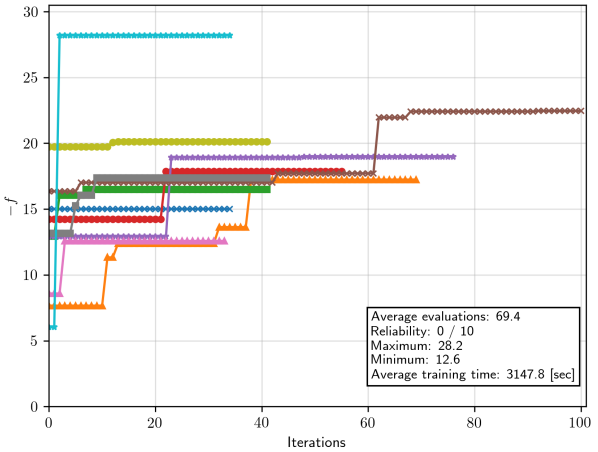
(c) Results of BO applied in the full space based on 5 initial starting points, to be comparable to the sampling of the 1D VAE case.



(d) Results of BO applied in the full space based on 16 initial starting points.



(e) Result of BO applied in a reduced space from 2 to 1 dimensions using a VAE based on 4 initial points.



(f) Result of BO applied in a reduced space from 2 to 1 dimensions using a VAE based on 16 initial points to compare to the 2D BO case. The improvement during training is minimal.

Figure 8.5: Results of the different optimization methods on optimizing the Shubert 3 function for two variables within the domain of $[-10.0, 10.0]$. The optimum is considered found, and added to the reliability, when a run finds a position within 0.5 of the known optimum.

8.3. Mechanical 2 fiber optimization

An arrangement of two circular fibers is described using their coordinates, leading to a 4-dimensional design space. This is still a relatively low number of dimensions, and serves as an introduction to a more complex fiber geometry. This section aims to study the influences of randomness in different parts of the optimization technique. As discussed earlier, the BFGS method is not applied due to the inability to accurately compute gradients.

8.3.1. Genetic algorithm

A genetic algorithm is used to optimize the 2-fiber mechanical problem. This is done six times for a single set of GA settings. The parameters used are presented together with the results in Figure 8.6. The GA converges to nearly the same optimum for all iterations, and shows a significant improvement compared to the average in the initial population. This further demonstrates that there is something to be gained from optimizing the mechanical performance.

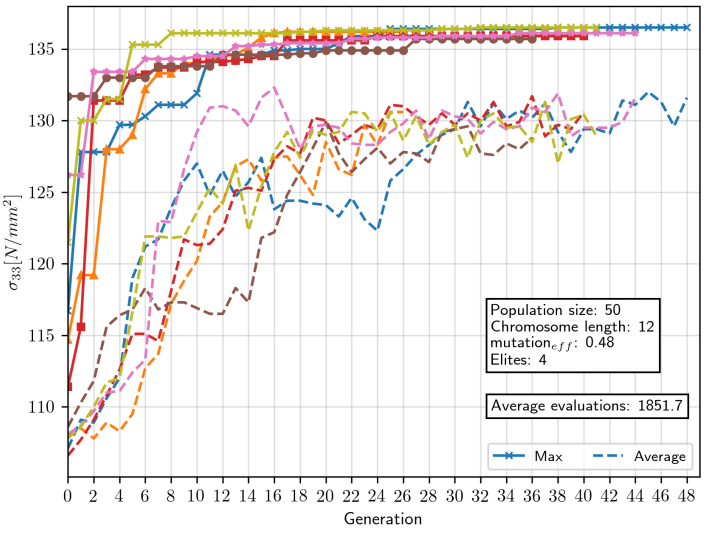


Figure 8.6: The result of six different GA optimization studies. While there is a significant difference between optima in the initial population, all GA's converge to nearly the same optimum. The increase of the mean of the population indicates that it is not simply trying random configurations.

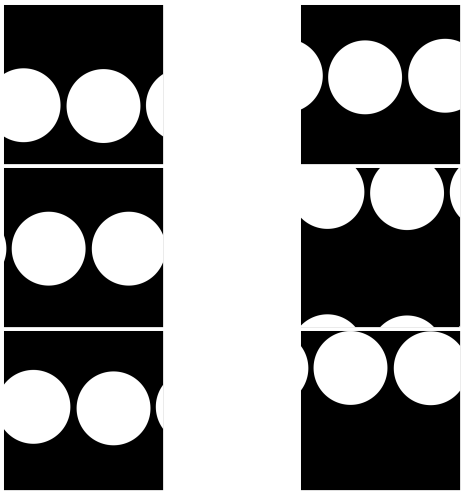


Figure 8.7: Resulting fiber arrangement for each GA run before processing. Having 2 fibers side by side with equal spacing is apparently the optimal geometry. As a consequence of this, there are many local optima .

Based on all GA's having converged to nearly the same optimum, and the similarity of optimal fiber geometries in Figure 8.7, it is likely that the global optimal solution is having two fibers horizontally aligned with maximum space in between. This solution can be achieved with many possible combinations in the full space, and the reliability of the GA in finding it is $6/6$. The reliability of finding the optimum, and the increase of the mean of the population of each run indicate relatively well defined GA settings.

8.3.2. Full Bayesian optimization

In determining the number of initial samples when applying full BO, the same logic is applied as for the analytical case resulting in $4^4 = 256$ samples. Six runs are performed, with the only difference between the runs being the initialization of the samples. The performance of all runs is presented in Figure 8.8. It is not as reliable as the GA in finding the optimum. Furthermore the training time of the BNN is significant, making BO only valid for problems where the functions themselves are expensive as well.

8.3.3. Bayesian optimization with a variational autoencoder

To explore the randomness in the various parts of the proposed framework, several combinations of runs are performed as presented in Table 8.1. Configuration A serves as the base case, and in configurations B to E individual parts of the framework are altered to show the difference. Based on the observed results of these configurations, two additional properties have been found of interest which are studied in configurations F and G.

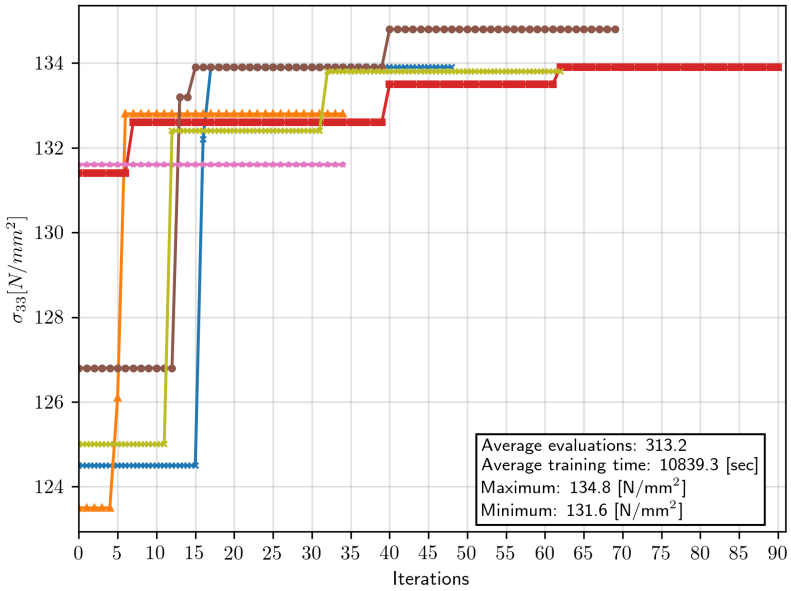


Figure 8.8: The result of six different Bayesian optimization runs in a 4-dimensional design space. With the number of initial samples being large, the maximum of the initial samples is relatively high. It is not reliable in finding the optimum.

Configuration	Latent dimensions	VAE training data	BO initial sampling	BNN initialization
A	1	VAE 1	$\mathcal{D}_1^{init}$	$\mathbf{w}_1^{init}$
B	1	VAE 1	$\mathcal{D}_2^{init}$	$\mathbf{w}_1^{init}$
C	1	VAE 1	$\mathcal{D}_1^{init}$	$\mathbf{w}_2^{init}$
D	1	VAE 2	$\mathcal{D}_1^{init}$	$\mathbf{w}_1^{init}$
E	2	VAE 3	$\mathcal{D}_3^{init}$	$\mathbf{w}_3^{init}$
F	1	VAE 1	$\mathcal{D}_4^{init}$	$\mathbf{w}_4^{init}$
G	1	VAE 1	$\mathcal{D}_5^{init}$	$\mathbf{w}_5^{init}$

Table 8.1: The variations of VAE+BO configurations used for optimizing the 2-fiber optimization problem. The initial samples $\mathcal{D}_i^{init}$ are created using a Latin hypercube. The BNN initialization $\mathbf{w}_i^{init}$ represents all network parameters.

VAE-1

Several combinations use the same variational autoencoder, VAE-1. It is trained using using the parameters given in Table 8.2. The result of the possible fiber positions in the full space is visualized in Figure 8.9 which demonstrates a loss in the design space. The performance during training is presented in Figure 8.10. The latent space is bound to $[-2.4, 1.95]$ based on its distribution. The training is stopped after a fixed number of epochs, there is no validation set used. The samples are created in a Latin hypercube manner.

Parameter	Value
Samples	512
Latent dimensions	1
Hidden layer nodes	300
Hidden layer activation	ReLU
Training epochs	60000
Regularization λ	$1e^{-3}$
Computation time	290.7 [sec]

Table 8.2: VAE-1 properties. The samples are created using a LHC.

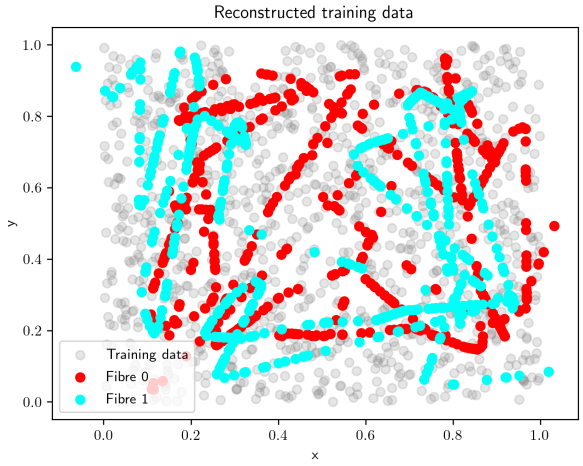


Figure 8.9: The result when the training data is transformed through VAE-1. Each point represents the center of a fiber.

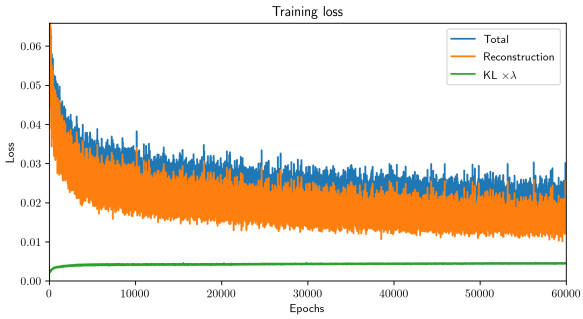


Figure 8.10: The reconstruction, KL and total loss during training of VAE-1. The curve no longer improving towards the end is an indication that the VAE is sufficiently trained.

Configuration A

The first configuration of the 2-fiber mechanical problems optimizes in a design space reduced from 4 to 1 dimensions using VAE-1. 4 initial samples are used, resulting in a single hidden node. The initial sampling and the result of the final BO iteration are presented in Figure 8.11. The hyperparameter β and the evidence during training are provided in Figure 8.12. The obtained optimum is significantly lower compared to that of the GA. It is clear that the final BNN prediction does not capture the full complexity of the dataset, and much is considered noise. With a near constant variance, the expected improvement acquisition function is very similar to the mean prediction, favouring samples around the current best peak. Because of this, it is likely that the VAE-1 does include a better result, but this is not captured during BO.

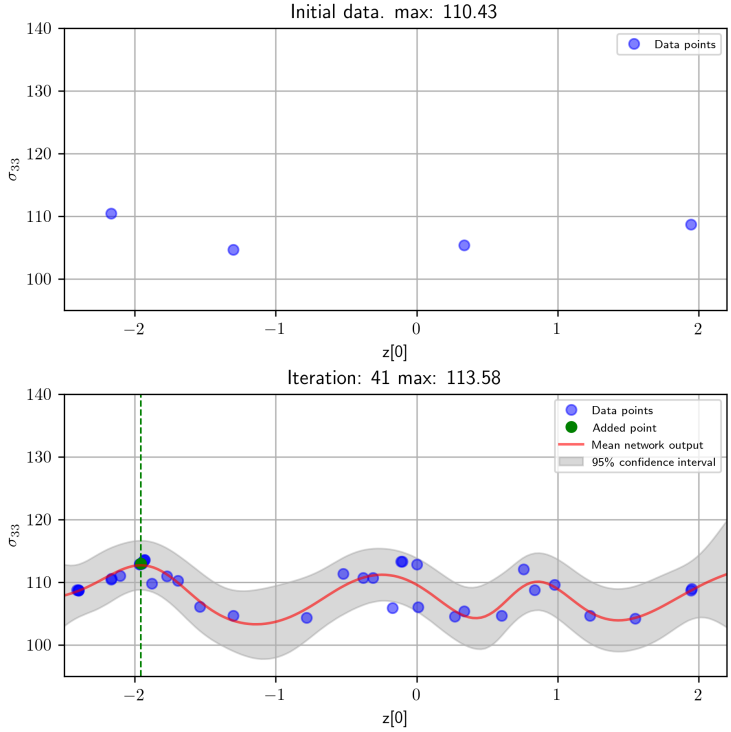


Figure 8.11: The result of configuration A. In the top figure the initial data is shown, followed by the final result of BO in the bottom.

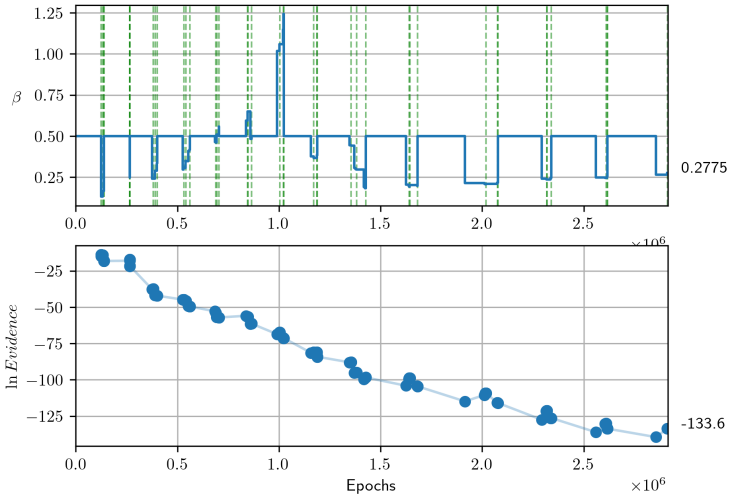


Figure 8.12: β and the evidence during training. The green lines correspond to samples being added. While the BNN aims to maximize the evidence for each model, the model changes as the dataset changes with every point added. The evidence decreasing is therefore not necessarily a sign of bad training. This figure also demonstrates the difference in computational effort between training and retraining the BNN.

Configuration B

Configuration B has the same settings as A with the only difference being the initial samples used in BO. As a result a different prediction model is trained. The initial sampling and the final results are presented in Figure 8.13. While in total more iterations are required, the resulting optimum is nearly the same as in configuration A. Around $z_0 = 0.0$ there is a single sample with a relatively high function value, but this is not further sampled by the BO due to the BNN considering it to be noise.

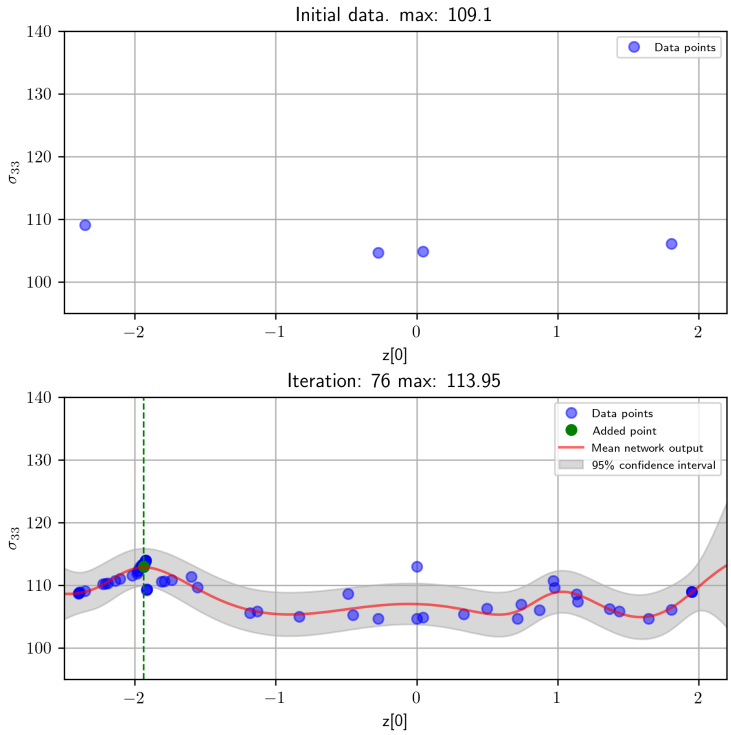


Figure 8.13: The result of configuration B. In the top figure the initial data is shown, followed by the final result of BO in the bottom.

Configuration C

Configuration C has the same settings as A with the only difference being the weight initialization in the BNN. As a result a different prediction model is trained. The result of BO is presented in Figure 8.14. Even though the prediction model is very similar, a different optimum is found with a higher objective value. The result is still significantly lower than the result obtained using a GA.

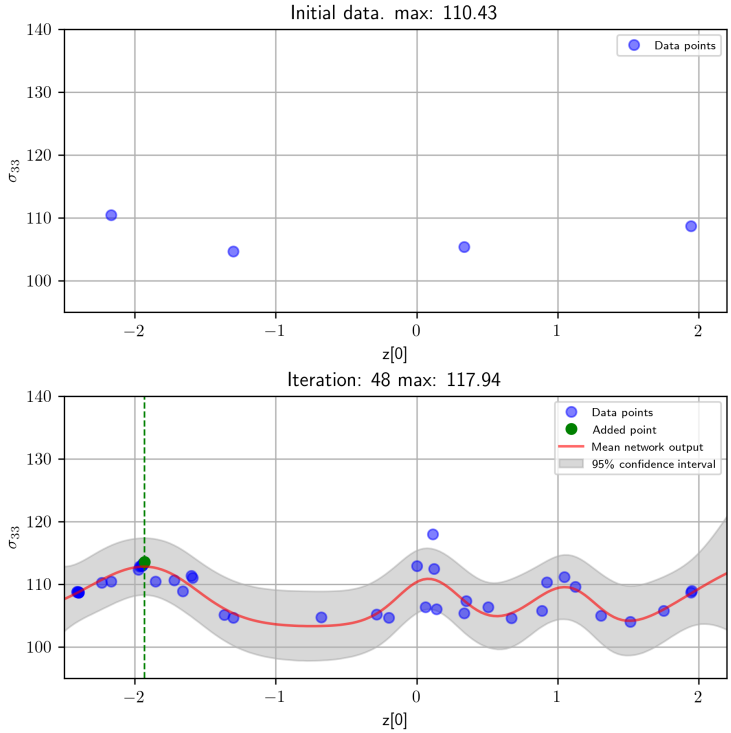


Figure 8.14: The result of configuration C. In the top figure the initial data is shown, followed by the final result of BO in the bottom.

Configuration D

The same parameters are used for the BO scheme in configuration D as for A, but now the latent space of VAE-2 is used. For VAE-2 all parameters from table 8.2 apply with the exception of the computation time, which is 281.88 seconds. The difference is the 512 LHC samples are created with different randomness. The result of the training is presented in Figures 8.15 & 8.16.

With a different latent space, the latent space objective function changes completely. The same values are selected for the initial sampling and the prediction model, but with a different function the result in Figure 8.17 changes significantly. The initial samples contain one sample significantly higher than the others. Still, the rest of the space is still explored during BO. The optimum here is comparable to that using a GA. This demonstrates that randomness in the VAE has a significant influence on the final result.

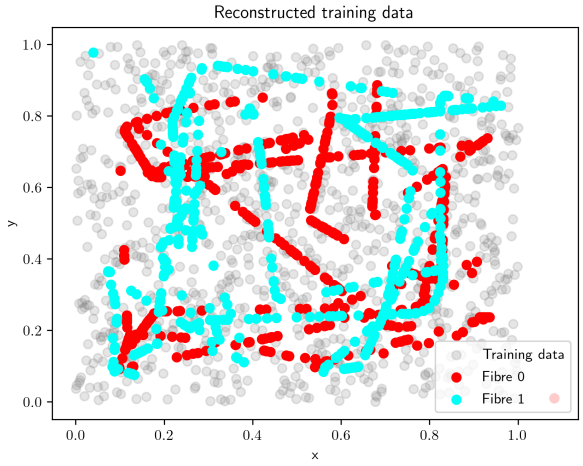


Figure 8.15: The result when the training data is transformed through VAE-2. Each point represents the center of a fiber.

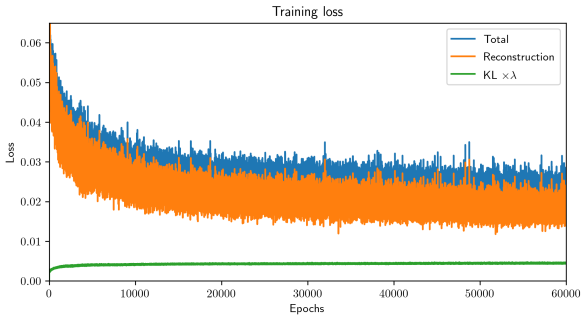


Figure 8.16: The reconstruction, KL and total loss during training of VAE-2. The curve no longer improving towards the end is an indication that VAE-2 is sufficiently trained.

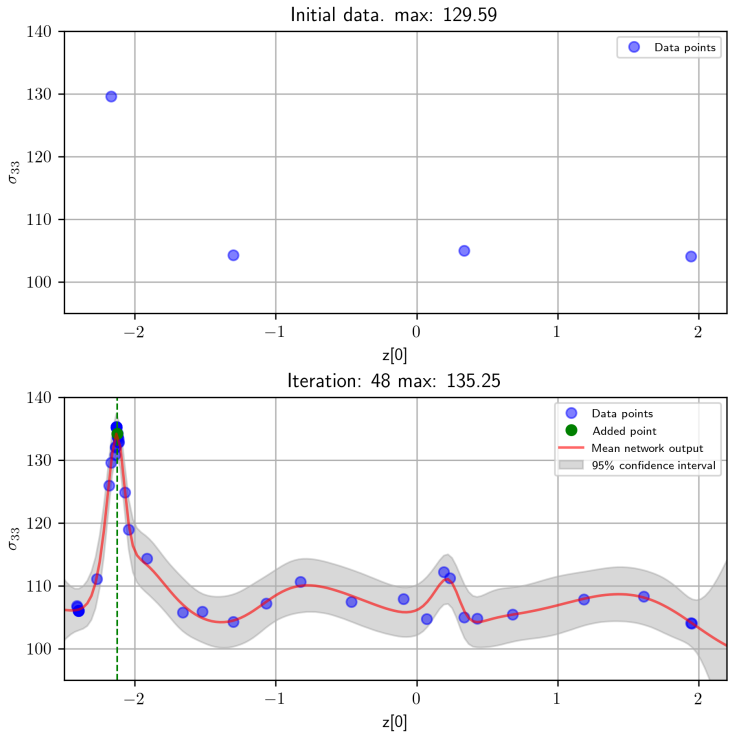


Figure 8.17: The result of configuration D. In the top figure the initial data is shown, followed by the final result of BO in the bottom.

Configuration E

In configuration E, a different VAE latent space with 2-dimensions is used. The training data for this VAE-3 is exactly the same as for VAE-1. The resulting reconstructed space and the training loss are presented in Figures 8.18 & 8.19.

By moving to a 2D latent space, 16 initial samples are used. The result at the start and the end of BO is presented in Figure 8.20. The maximum has improved significantly, but the BNN prediction still does not seem capable of accurately representing the data and it considers much of the data as noise. Furthermore the points around the optimum are not extensively sampled, making it probable that the optimum was found based on randomness more than by an accurate prediction.

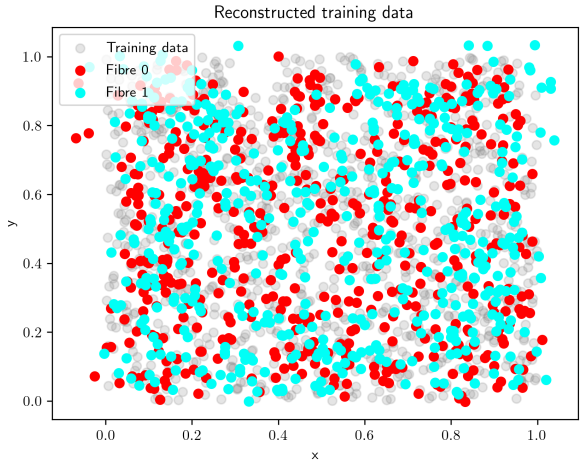


Figure 8.18: The result when the training data is transformed through VAE-3. The fibers are presented with the different colours.

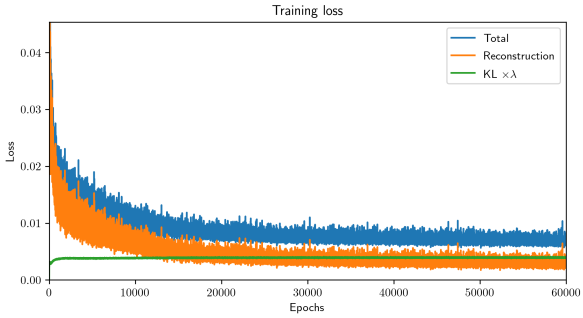


Figure 8.19: The reconstruction, KL and total loss during training of VAE-3. The curve no longer improving towards the end is an indication that the VAE is sufficiently trained.

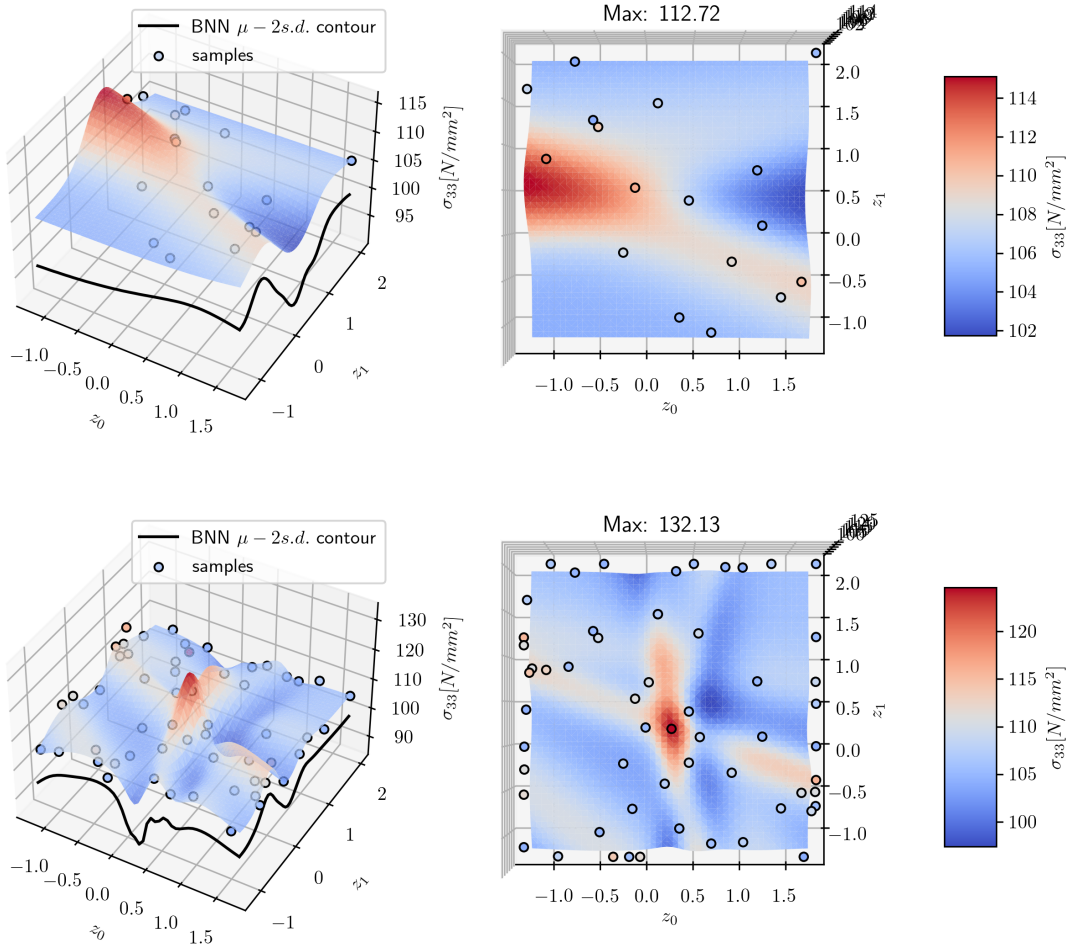


Figure 8.20: The result of configuration E. In the top figure the initial data is shown and the BNN prediction, followed by the final result of BO in the bottom. The color scales change between the plots.

Configuration F

In the results of configurations A to E, it is argued several times that the BNN failed to capture the full complexity of the design space, leading to poor predictions. Here the number of initial samples is increased to 50, allowing a more complex BNN network by reducing the limitation of the data being larger than the number of parameters. The results are presented in Figure 8.21. It is clear that the complexity of the prediction has increased, and several potential peaks are sampled. The probability of having found the optimum of the reduced space has increased significantly. This also demonstrates that configuration A, B and C, which use the same VAE, fail to find the optimum.

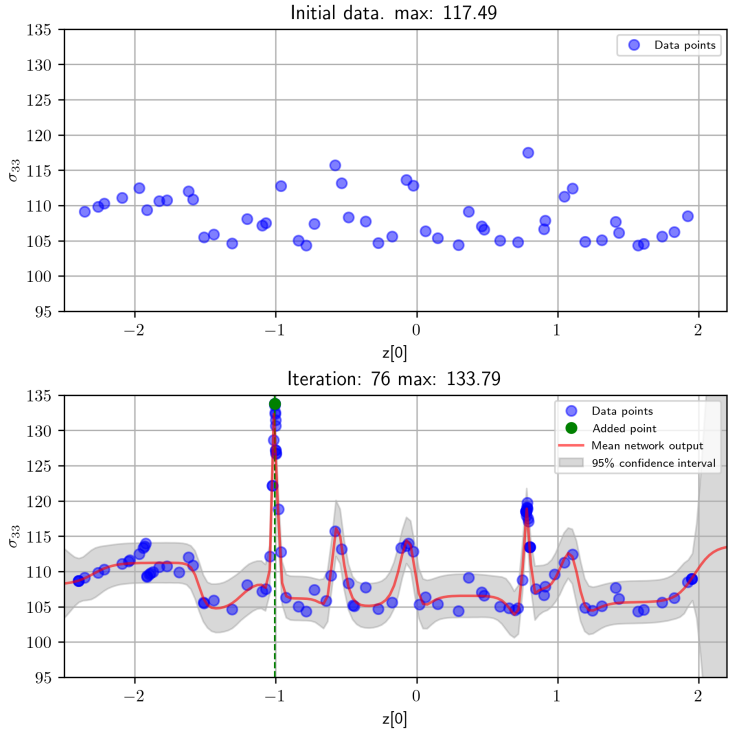


Figure 8.21: The result of configuration F. In the top figure the initial data is shown, followed by the final result of BO in the bottom.

Configuration G

Based on the previous results, the difficulty of the optimization framework seems to come from the BNN prediction model requiring a large number of samples. Apart from there simply being more data, this also allows the BNN to use more parameters, as explained in Section 5.5.2. Here this constraint on the number of parameters in the BNN is violated. The practical limitation of allowing the number of BNN parameters to be larger than the number of samples comes from β possibly becoming negative. As explained at the start of this chapter, for all previous results β is already bounded at a minimum of 0.1, making this no longer a practical issue. Whether doing this is theoretically sound is a legitimate question but not studied in this thesis.

VAE-1 is used again with 4 initial samples. The number of hidden nodes is fixed at 50, giving 151 BNN parameters. The result in Figure 8.22 shows no significant difference compared to the earlier results. Although no definitive conclusions should be made from this, it indicates that the limited BNN size is not the reason it fails to capture the optimum.

Overall

As every considered case has different settings, most cannot be compared directly. To present the results similar to that of the full BO and the GA, all configurations are combined in a single plot in Figure 8.23. The results show that even with few samples the optimum is improved. In cases A to E, the maximum objective after initial sampling is on average 114.4. With 51 additional BO samples

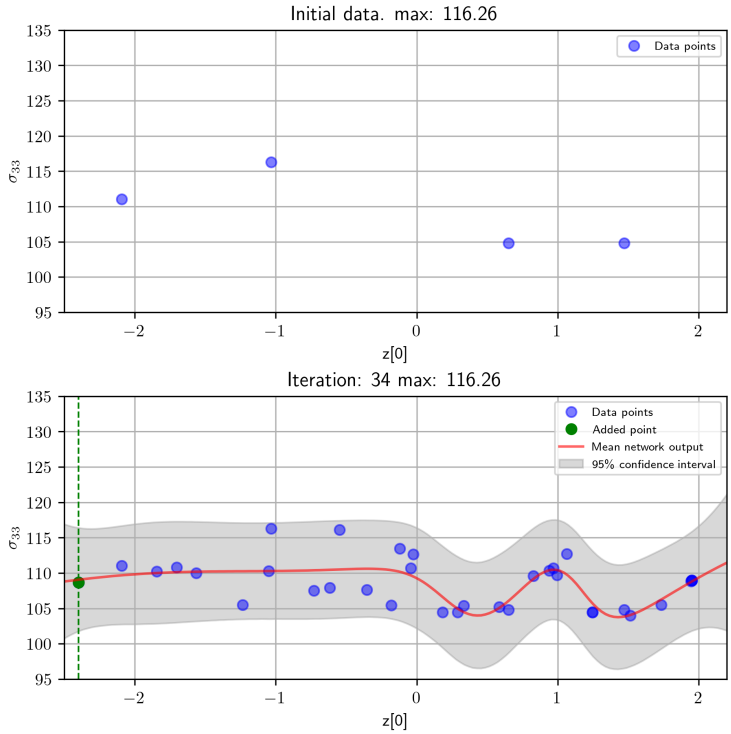


Figure 8.22: The result of configuration D. In the top figure the initial data is shown, followed by the final result of BO in the bottom.

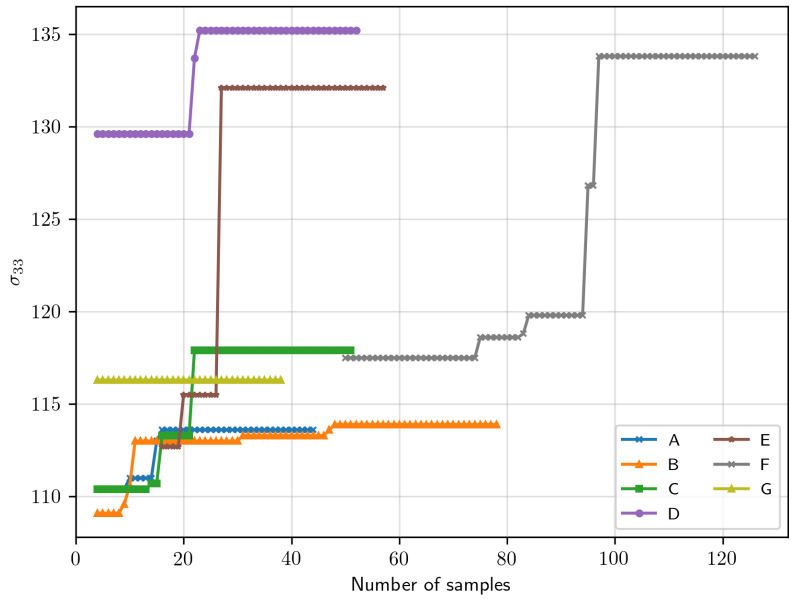


Figure 8.23: The result of all configurations with respect to the number of function evaluations.

this is increased to 122.5. The region with the highest function value in the initial sampling is generally the region BO found the final optimum. This results from the data being complex, leading to many predictions giving a near random result. Cases D and E show that there is potential for the framework to find a good optimum with very few function evaluations, but the reliability is very poor. Where the computation time of a GA itself is redundent, the computation time of the methods using a VAE and BO are not, as presented in Table 8.3. This training time is independent of the computational effort of a function evaluation. It also shows that significantly less samples are used compared to BO in the full space and the GA. The fiber geometries are not provided but for the results with a high objective value are similar to those of the GA.

Configuration	Total number of samples	VAE training time [sec]	BNN training time [sec]
A	45	291	52.39
B	79	291	571
C	52	291	85
D	53	282	147
E	58	284	160
F	127	291	29160
G	39	291	1981

Table 8.3: The computation time for the different configurations. The significant increase in computation time of configuration F results from the poor $\mathbf{w}_M AP$ convergence criteria, and instead relies on the additional $3e^6$ epoch criteria. The training time of configuration G results from having many more network parameters.

8.4. Mechanical 5 fiber optimization

A scenario is created with 5 morphing fibers, leading to a 15-dimensional optimization problem. Similar to the two fiber case, the gradients could not be computed and therefore no gradient based method is used. A genetic algorithm, Bayesian optimization in the full space, and Bayesian optimization in a reduced 1-dimensional space are used. The fiber radius is decreased compared to the two fiber case as more fibers are used.

8.4.1. Genetic algorithm

6 runs using a genetic algorithm are performed. The chromosome length per dimension is reduced from 12 to 8 bits. While this reduces the resolution of the GA, the discretization step size is $2^{-8} = \frac{1}{256} = .003906$, the performance increases by having $2/3$ of the parameters, and therefore requires less function evaluations. The result is provided in Figure 8.24. The GA requires a significant amount of generations before converging. Furthermore, a significant increase in maximum obtained stress is observed compared to the 4-dimensional case. The fiber geometry presented in Figure 8.25 makes the reason for this evident. The fibers are arranged such that a continuous feature is created from the top to the bottom of the RVE. This is similar to shifting from a series of springs to having parallel springs, drastically changing the stiffness of the model. The stress strain curve for this configuration is given in Figure 8.26, showing an almost linear relationship. While this is likely the optimum for the optimization problem as it is defined, it is unlikely that this geometry will perform good in practice, but this is outside the scope of this thesis. It is suspected that the function is not continuous between a geometry with a small gap and one with such a continuous geometrical feature.

8.4.2. Bayesian optimization

Applying Bayesian optimization in this 15-dimensional design space is not straightforward. Where in 4-dimensional case $4^4 = 256$ samples are used, an equivalent sampling here leads to 1073741824 samples. As this is not feasible and probably also not required as BO itself samples regions of uncertainty, the number of samples provided to the full BO scheme is 512 based on the available computation time within this thesis. Apart from the BO sampling, also the GA for maximizing the EI is affected. It is set to a population size of 200, 6 elites, and an effective mutation rate of 0.9, performing significantly more evaluations than the GA used in the example above. The results are presented in Figure 8.27.

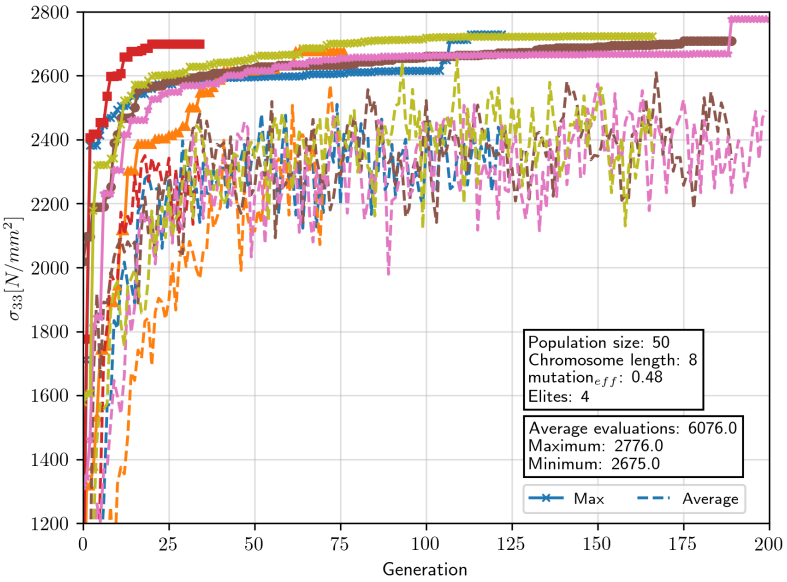


Figure 8.24: Results of a genetic algorithm on the 5-fiber case. A significant number of generations are required before convergence.

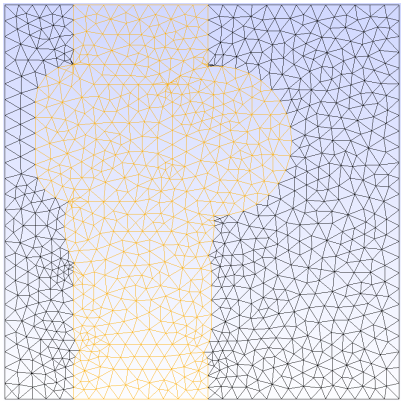


Figure 8.25: The mesh of the fiber geometry with the highest function value found by a GA.

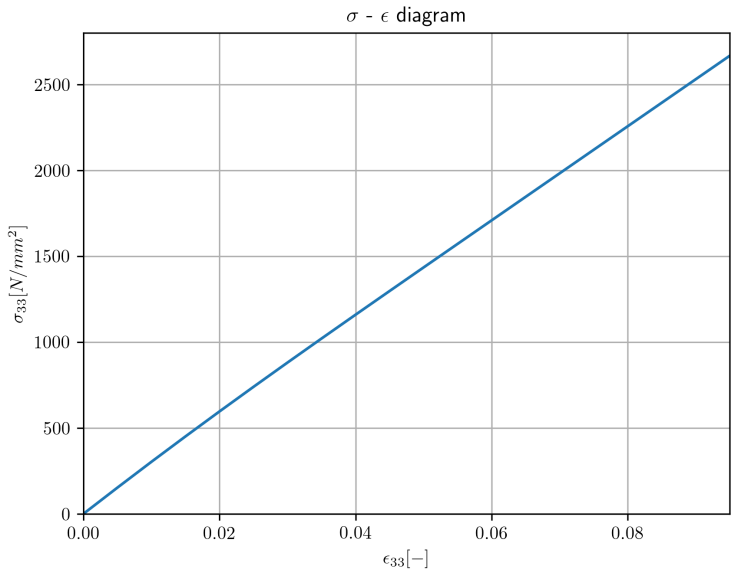


Figure 8.26: The stress strain relationship for the geometry presented in Figure 8.25, which indicates that an almost linear relation is found. This also indicates that the result of this particular optimization study is dependent on the strain level.

Since no optimization is happening, it is likely that the BNN fails to make accurate predictions. The initial sampling includes fiber arrangements in which the top and bottom of the RVE are connected, as without this the objective would definitely not reach above 200 [N/mm^2]. Imagining most of the samples having an objective value of around 130, with one or a couple of outliers around 2000 makes it easy to imagine that the prediction model considers much of the samples as noise.

8.4.3. Bayesian optimization with a variational autoencoder

The number of samples for training the VAE is kept at 512, making it the same number of samples as the BO is trained on. Based on the results of previous sections, it can be expected that when reducing the dimensionality from 15 to a single dimension the complexity of this latent space becomes immense. The results are presented in Figure 8.28. Noticably, 4 out of the 6 runs find a configuration where a continuous fiber is created from the top to the bottom of the RVE. As only a single dimensional space

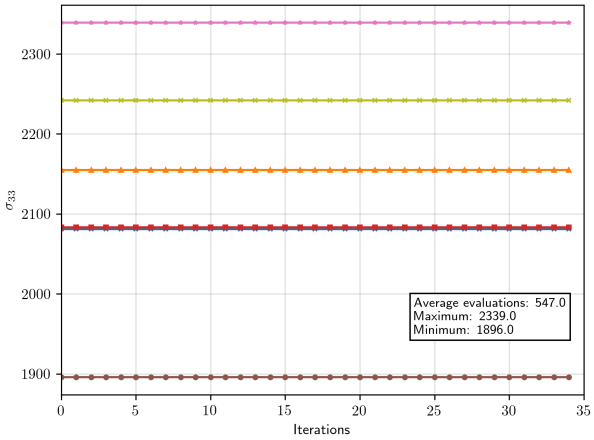


Figure 8.27: Results of Bayesian optimization on the 5-fiber case in the full 15-dimensional design space. It is initialized with 512 samples. No improvement is made during training, indicating that the BO is unable to make sensible predictions.

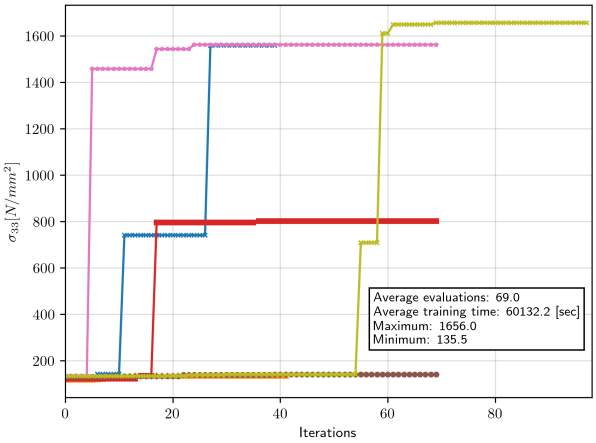


Figure 8.28: Results of Bayesian optimization in a reduced design space from 15 to a single dimension. Each run has a separately trained variational autoencoder, and are initialized with 4 BO samples. Note that a different y-scale is used compared to Figure 8.27.

is optimized, the effect of such a significant difference between different samples on the BNN prediction can be visualized and is shown in Figure 8.29. It shows that in order for the BNN prediction to capture the high function value, the prediction in other regions becomes worse. The space around the optimum also does not seem continuous.

The values of the optima are significantly lower compared to those captured by the GA and BO in the full space, even if that last one is only the result of random sampling. The network prediction plots indicate that the high-function samples are not predicted, but instead randomly found. When such a region is found however, the BNN attempts to sample around it and in several cases finds a similar geometry with an even higher value. The number of samples used is less than 2 generations of a GA.

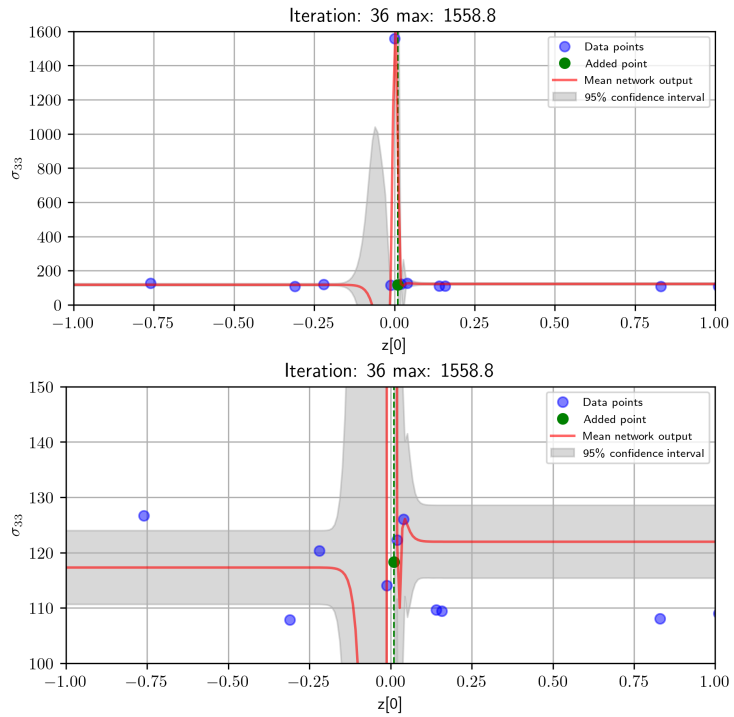


Figure 8.29: The training data and BNN prediction for a zoomed in region of the latent space. Both figures display the same moment during training, the difference being the y-axis scaling.

8.5. Discussion

The result of the the simple analytical function showed that the method presented does not reliably find the global optimum as a result of the VAE not having any function information. Still, based on less initial samples it is able to optimize the results quite a bit. With having the gradients available directly, the BFGS method performs significantly better, having both less function evaluations and reliably finding the optimum. When the dimensionality stays the same, but the complexity of the design space increases, all methods perform worse. BO in the full space is no longer able to reliably create an accurate prediction, neither is BO in a reduced design space. This demonstrates that the complexity of a function matters.

Based on the optimal results for the 2-fiber configuration, the objective function is believed to contain many parameter combinations with an equivalent global optimum. This optimum is believed to be found for all x and y where $y_1 = y_2$ and $|x_1 - x_2| = 0.5$. This means the problem solved is actually a 2-dimensional problem, as an optimum can be found with 1 fiber being fixed.

In any framework, ideally the same global optimum is found regardless of random initialization. It is however shown to be dependent on the training of the VAE, the initialization of the BO and the initialization of the weights. The area around the best initial sample in latent space is exploited further and the BO does explore other regions as well, however this exploring is generally not enough to find all potential regions. This results from the BNN considering much of the difference between samples as model noise, resulting in a near constant variance. When more initial samples are used to the extent where much of the shape is already defined, the BO scheme is able to explore several peaks. The performance based on a single run is better than that of the average when applying BO in the full space. This should be studied further to see if there is an advantage in reducing the dimensionality and using less samples, but without reducing the number of samples with the same order as the design space.

One factor that makes it difficult to draw conclusions from the results is the settings of the various methods. If the hybrid VAE BO is set to require much more cycles before being considered converged, it would automatically lead to a more densely sampled space and a therefore likely a better result. A similar criteria is the number of generations and the population size in a GA, which by default set the GA up to use more samples compared to the other methods considered.

The 5-fiber case exploited the defined optimization problem by finding a solution with little practical value, but that otherwise makes sense mechanically. It provides interesting insight on the micromechanical problem being solved, as the continuous fiber arrangement effectively blocks the formation of a plastic localization band. The GA was still relatively reliable here. BO in the full space however was unable to perform any optimization. The combination of a VAE and BO was able to find better results during training, although this probability is also higher due to having sampled less. Its optima were still lower than those in the random initialization of the full BO, but at the same time much less evaluations are performed. The BO methods required a significant amount of additional training time which scales with the number of samples used, favouring problems with very expensive function evaluations.

Based on all results it is shown that reducing the dimensionality using a VAE increases the complexity of the latent space to the point where reducing the number of samples with the same dimensionality does not provide good results. However it can not be concluded that reducing the dimensionality is worse in all cases. More studies are required to set up experiments specifically aimed at the density of the sampling in the reduced space.

9 Conclusion and recommendations

9.1. Conclusions

This thesis investigated the performance of a combination of a variational autoencoder (VAE) and Bayesian optimization (BO) in optimizing high dimensional problems, with the aim of separating the high-dimensionality from the expensive function evaluations. The framework of methods was demonstrated by optimizing analytical functions and by optimizing the geometry of fibers in a microstructure to maximize mechanical performance. The problem has been generalized by releasing constraints on the geometry of the fibers using a custom meshing procedure. Starting from the basics of a neural network, a variational autoencoder and Bayesian neural network have been introduced. The Bayesian neural network using the evidence approximation was extended to Bayesian optimization, and a genetic algorithm has been implemented. The methods used in the framework have beed studied individually on the influence of user defined parameters (hyperparameters) on their performance. Based on these studies informed choices are made for the settings when all methods are combined. The results have been compared to a gradient based method (BFGS), a genetic algorithm (GA), and BO in the full design space. The performance of the complete framework is discussed in the first research question, followed by the answers to the sub-questions.

How does Bayesian optimization on a reduced design space compare to conventional optimization methods in the quality of the optima and the required computational effort?

To summarize, the drawback of this method in optimizing a high-dimensional design space where every configuration is valid is that it is unlikely to find the global optimum. However, with some additional problem independent computational cost, optimization is possible with significantly less function evaluations than other methods. Results are promising, but whether this gain makes it a favourable optimization method are inconclusive. The result depends on many factors which are discussed further.

In reducing the dimensions of the design space using the variational autoencoder, part of the design space is lost. This loss increases as the factor between the original and reduced dimensions increases. Although the training of the variational autoencoder does not require expensive function evaluations, its training time does still scale with the number of samples and therefore does not fully get rid of the problem of high-dimensionality.

Applying Bayesian optimization in the reduced space provided poor results when the number of initial samples was very low and the full design space already contained many local optima. In such cases, the Bayesian neural network (BNN) predicts most of the local optima as noise in the data. This partly results from the BNN being optimized to maximize the evidence, favouring simple models that explain the data. It is also affected by the level of regularization of the variational autoencoder, more investigation is required in studying this relation.

The region around the point of the initial sample with the highest function value was generally explored more and lead to some optimization with little function evaluations, but other regions were not sufficiently explored given the difference between datapoints. For a 2-fiber example, the maximum objective after on average 6 initial samples was 114.4 [N/mm^2], which improved to 122.5 [N/mm^2] using 51 BO iterations. Using more samples, but still significantly less compared to the amount in the full space showed promise, but is not studied enough to draw conclusions. Initializations in both the VAE and in BO influenced the results significantly. The influence of the VAE is directly related to the loss in design space. The BO influence is expected to decrease with a better prediction model. For larger datasets, the

training time of the BNN increased significantly, adding a computational cost to the optimization scheme independent of the function computation time. The framework used therefore becomes increasingly attractive as the computational effort per function evaluation increases. The obtained results are encouraging, but still inconclusive.

The type of problem used in this thesis is one where every parameter is always relevent for the result. Other types of problems exist with large amounts of data and a vast number of parameters that describe much less underlying features. These features could be captured in the latent space of the VAE, making optimization feasible. For these problems the proposed framework could provide better results.

Do geometric properties of fibers in a fiber reinforced polymer microstructure influence mechanical properties?

In running random configurations of fiber geometries, a significant influence on the maximum stress at a strain of 0.1 in uniaxial tension is already observed. The optimization study further confirmed this, and by comparing the difference in the average and optimized result it is shown that there is a significant possible gain from optimization of microstructure mechanical properties. In a 2 fiber study, this maximum stress is on average $108[N/mm^2]$, while an optimum fibre arrangement gives $136[N/mm^2]$, a 26% increase.

For a 5-fiber arrangement, the optimum was found by creating a continuous fiber connection between the top and bottom of the representative volume elment, exploiting the definition of the optimization problem. In practice, such a microstructure is unlikely to perform well, as more failure mechanisms and load cases considered.

Can the optimal solution be used to inform future engineering design decisions?

Based on the observed result, the 2-fiber arrangement should be placed tangential to the load direction in uniaxial tension to maximize the strength of a FRP microstructure. The extent to which this microscopic optimization transfers to macroscopic structures requires further investigation. Furthermore, as generally multiple load cases apply, this feature is unlikely to be as relevant in actual structures. Nevertheless, there is no intrinsic limitation in the present framework, which would be equally applicable for more complex loading. For the 5-fiber arrangement, a very good result is observed but which does not translate into practice. So while this case is unlikely to be used, it does however demonstrate the possibility for solutions that provide surprisingly good results and are easily generalizable.

What are the costs and benefits of reducing the dimensionality of a design space for finding an optimal solution?

A reduced parameter space has significant advantages for optimization, and the variational autoencoder has been used to create it. As the number of possible configurations increases exponentially with the number of dimensions, dimensionality reduction leads to a significant reduction in possible configurations and therefore computational effort. There are several drawbacks to this. The first disadvantage is that part of the original design space is lost, and with that a probability of not including the global optimum. Secondly, the reduced space has an increased complexity, making optimizing that dimension more difficult and likely requiring more sampling, reducing the initial gain. A further drawback is the training of the variational autoencoder, adding computational effort. The training of the VAE is still susceptible to the problem of high-dimensionality, as the required number of samples still grows exponentially. This results in the training data potentially missing features of the original design space. In problems where only a small subset of the full design space leads to valid outcomes, the usage of the VAE is closer to its more common application of generating images and its performance could increase.

To which extent does Bayesian optimization reduce the computational effort in finding the optima within the reduced space?

With the reduced space containing many local optima, a significant number of evaluations will be required with any method. This depends on the complexity of the original design space and the variational autoencoder. While Bayesian optimization uses very little function evaluations, in the results it does not reliably find the optima of the reduced space, making any comparison difficult. The performance of other methods in finding the optimum of the reduced space requires further investigation.

Can a Bayesian neural network be effectively used for Bayesian optimization?

The results of a simple analytical case show that the BNN is able to create accurate predictions and an uncertainty in its predictions to effectively perform BO. When having a more complex function, the BNN prefers simple models, often leading to predictions with little change in its mean, and having a constant high uncertainty. As a result of this the variance of the network was not sufficient for BO to explore the appropriate regions, resulting in the final optimum found being near the optimum found during the initial sampling.

How do the hyperparameters and randomness in the initialization of methods influence the resulting optimum?

It has been demonstrated using parameter studies for each individual component of the framework that some of the hyperparameters have a significant influence on its performance. For the VAE the regularization λ determines the extent to which the reduced space covers the original design space. In the BNN, the convergence criteria pose a trade-off between an accurate model and computation time. For BO, the exploration parameter is changed iteratively. For the GA, using elitism and a mutation rate increased performance, but optimum values are problem dependent.

Randomness is a part of every method used in the framework. In the VAE it is in the initial data set and network parameters. In the BO scheme it similarly is in the initial samples and the network parameters of the BNN. The randomness in the VAE results in a very different possible optimum for each configuration. The randomness in the BO scheme should in theory not influence the optimum of the reduced space, but does as a consequence of the BO performing sub-optimally.

9.2. Recommendations

Based on the work in this thesis, several recommendations are presented for further work on the topic of high dimensional optimization. The recommendations are split into solving issues with the current methodology and extending the framework to increase reliability or reduce computational effort. Furthermore a recommendation considering the optimization of microstructures is made, and several types of optimization problems are discussed where using a variational autoencoder is promising.

9.2.1. Increasing current performance

Several recommendations are made with regards to increasing the performance of the framework as it is provided in this thesis.

Alternative prediction methods

A serious limitation of the framework proved to be the capability of the network to distinguish complexity in a function from noise. An alternative prediction method could be used. There exist many methods for making predictions based on observations, from different types of Bayesian neural networks to Gaussian processes and sampling techniques such as the ensemble Kalman filter. Investigating these and other models on whether they make better predictions could improve the framework.

Minimizing complexity in the variational autoencoder latent space

Currently the VAE is trained to minimize the loss in its reconstruction capabilities. Minimizing this loss is related to increasing the complexity of the design space. Simultaneously the latent space is regularized to follow a specific distribution. While this regularization also influences the complexity of the design space, it is not its main purpose. It could be studied if an additional term in the loss function could specifically reduce the complexity of the design space.

Bayesian optimization settings

In the cases demonstrated in this thesis with dimensionality reduction, the initial samples in the BO scheme were generally less than required to make accurate predictions. While the ideal number of initial samples depends on the complexity of the function, which is unknown, a study could be performed where the scaling in the number of samples is varied based on the level of dimensionality reduction.

Related to this issue, the influence of different convergence criteria of the BO scheme should be studied. For example a criteria could be included related to the level of predicted noise.

Thorough parametric study
The parametric study of the VAE on the impact of different levels of dimension reduction is not as extensive as it should have been. For example no guideline is presented on the optimal number of latent dimensions, even though this has a significant influence on the result. Furthermore, each parameterstudy is performed separately. Ideally the parameters of each method are studied based on the performance of the complete framework. In such a study several different benchmark functions should be used.

9.2.2. Extending the framework
Here possible additions to the framework are introduced that could increase its reliability and decrease computational effort.

Bayesian optimization in a reduced design space followed by a gradient based step
Assuming that Bayesian optimization finds the optimum of the reduced space, it is unlikely to be at a local optima of the full design space. At the same time, the BFGS result is highly dependent on the initial starting point. This can be combined, by the VAE+BO scheme quickly finding a relatively good solution, followed by a gradient based step to find the exact (local) optima. This is a relatively easy extension that would increase the probability of finding the global optimum.

Batch Bayesian optimization
A significant amount of time is spent training the BNN and updating it for each new sample. This could be adapted such that based on a trained BNN, the acquisition function is maximized for multiple values of ζ to both explore and exploit at the same time. This batch of samples is evaluated and added together, before retraining the BNN. This would significantly reduce the BNN training time.

Informed VAE training
The current VAE is trained based on possible input data with no information of the objective values. Some initial function evaluations could be performed to inform the training of the VAE such as to prioritize high optimization regions to begin with. When doing this the advantage compared to applying BO in the full space becomes unclear, but perhaps there is still a possible gain.

9.2.3. Microstructure customization
This thesis has shown that geometric properties of the micro scale can have a significant influence on its performance. The extent to which the optimum found translates into different macroscopic behaviour should be investigated. If it is the case that a similar gain is achievable, it could inspire further research into manufacturing techniques that allow this level of control. It is recommended to use a much more general mechanical property for optimization, such as the area of a yield surface. Furthermore, many structures are currently homogenized materials. Assuming full control of the micro structure, a beam could for example be split into different parts separately optimized. This could result in the geometry of fibers in the top optimized for compression while the geometry in the bottom is optimized for tension, and near the supports for shear.

9.2.4. Problems with a lot of promise
The underlying methods of the framework makes it suitable for certain optimization problems, while for other problems different optimization techniques should be preferred. Here some situations are presented where using a variational autoencoder can be promising, and it is recommended to study this further.

Non-parametrizable geometries
Performing a full-order optimization is impossible in certain non-parametrizable geometries. An example of this is a complex microstructure based on images. A convolutional variational autoencoder could be used, similar to many existing applications of image generation, to automatically extract features from these images into the lower dimensional design space.

Strong underlying lower dimensionality
For certain problems it is known a priori that there is an underlying lower dimensionality. The example of images in the previous recommendation is such a case, where only the recognized features are

relevant. In such a case neighboring pixels are highly correlated. Different examples include optimizing performance based on a vast amount of measured temperature or wind data, or based on data from digital image correlation. This is linked to problems for which automatic relevance determination is used, a method which quantifies the relevance of input features.

Complex geometric constraints
Certain problems have complex geometric constraints which depend on the specific settings of other variables. An example would be if the fiber arrangements considered in this thesis could not contain overlapping fibers. With a dataset that only contains valid configurations, the latent space can become unconstrained. Due to its generative ability, with the current VAE settings it cannot be guarenteed that the latent space becomes unconstrained, and further studies are required.

Prior knowledge of high-potential regions
In problems where certain prior knowledge of high-potential regions is known, a smarter VAE dataset could be created that increases the likely hood of the optimum being found. Many other optimization techniques also benefit from such prior knowledge.

Bibliography

[1] Ciyou Zhu, Richard H Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, 23(4):550–560, 1997.

[2] Wen Long, Jianjun Jiao, Ximing Liang, and Mingzhu Tang. An exploration-enhanced grey wolf optimizer to solve high-dimensional numerical optimization. *Engineering Applications of Artificial Intelligence*, 68:63–80, 2018.

[3] Manoj Kumar, Mohamed Husain, Naveen Upreti, and Deepti Gupta. Genetic algorithm: Review and application. *Available at SSRN 3529843*, 2010.

[4] Gy Kovács, AA Groenwold, K Jarmai, and J Farkas. Analysis and optimum design of fibre-reinforced composite structures. *Structural and Multidisciplinary Optimization*, 28(2-3):170–179, 2004.

[5] Momin Jamil and Xin-She Yang. A literature survey of benchmark functions for global optimisation problems. *International Journal of Mathematical Modelling and Numerical Optimisation*, 4(2):150–194, 2013.

[6] Ryan-Rhys Griffiths and José Miguel Hernández-Lobato. Constrained bayesian optimization for automatic chemical design. *arXiv preprint arXiv:1709.05501*, 2017.

[7] Rika Antonova, Akshara Rai, Tianyu Li, and Danica Kragic. Bayesian optimization in variational latent spaces with dynamic compression. In *Conference on Robot Learning*, pages 456–465. PMLR, 2020.

[8] Jive - software development kit for advanced numerical simulations. `https://software.dynaflow.com/jive/`. Accessed: 25-03-2020.

[9] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[10] I.B.C.M. Rocha, P. Kerfriden, and F.P. [van der Meer]. Micromechanics-based surrogate models for the response of composites: A critical comparison between a classical mesoscale constitutive model, hyper-reduction and neural networks. *European Journal of Mechanics - A/Solids*, 82:103995, 2020.

[11] MR Sanjay, GR Arpitha, and Basavegowda Yogesha. Study on mechanical properties of natural-glass fibre reinforced polymer hybrid composites: A review. *Materials today: proceedings*, 2(4-5):2959–2967, 2015.

[12] Frans P Van der Meer. Mesolevel modeling of failure in composite laminates: constitutive, kinematic and algorithmic aspects. *Archives of Computational Methods in Engineering*, 19(3):381–425, 2012.

[13] Camille Perrot, Fabien Chevillotte, and Raymond Panneton. Bottom-up approach for microstructure optimization of sound absorbing materials. *The Journal of the Acoustical Society of America*, 124(2):940–948, 2008.

[14] Mikhail Osanov and James K Guest. Topology optimization for architected materials design. *Annual Review of Materials Research*, 46:211–233, 2016.

[15] Xiaoyu Zheng, William Smith, Julie Jackson, Bryan Moran, Huachen Cui, Da Chen, Jianchao Ye, Nicholas Fang, Nicholas Rodriguez, Todd Weisgraber, et al. Multiscale metallic metamaterials. *Nature materials*, 15(10):1100–1106, 2016.

[16] Narasimha Boddeti, Yunlong Tang, Kurt Maute, David W Rosen, and Martin L Dunn. Optimal design and manufacture of variable stiffness laminated continuous fiber reinforced composites. *Scientific reports*, 10(1):1–15, 2020.

[17] Hongwei Yao, Xianhang Sui, Zhongbo Zhao, Zhiwei Xu, Lei Chen, Hui Deng, Ya Liu, and Xiaoming Qian. Optimization of interfacial microstructure and mechanical properties of carbon fiber/epoxy composites via carbon nanotube sizing. *Applied Surface Science*, 347:583–590, 2015.

[18] Christian Schumacher, Bernd Bickel, Jan Rys, Steve Marschner, Chiara Daraio, and Markus Gross. Microstructures to control elasticity in 3d printing. *ACM Transactions on Graphics (TOG)*, 34(4):1–13, 2015.

[19] Mehdi Kalantari, Chensong Dong, and Ian J Davies. Multi-objective robust optimization of multi-directional carbon/glass fibre-reinforced hybrid composites with manufacture related uncertainties under flexural loading. *Composite Structures*, 182:132–142, 2017.

[20] Felix Raspall, Rajkumar Velu, and Nahaad Mohammed Vaheed. Fabrication of complex 3d composites by fusing automated fiber placement (afp) and additive manufacturing (am) technologies. *Advanced Manufacturing: Polymer & Composites Science*, 5(1):6–16, 2019.

[21] Nanya Li, Guido Link, Ting Wang, Vasileios Ramopoulos, Dominik Neumaier, Julia Hofele, Mario Walter, and John Jelonnek. Path-designed 3d printing for topological optimized continuous carbon fibre reinforced composite structures. *Composites Part B: Engineering*, 182:107612, 2020.

[22] Bo Zhu, Mélina Skouras, Desai Chen, and Wojciech Matusik. Two-scale topology optimization with microstructures. *ACM Transactions on Graphics (TOG)*, 36(4):1, 2017.

[23] Klaus-Jürgen Bathe. *Finite element procedures*. Klaus-Jurgen Bathe, 2006.

[24] Frédéric Feyel and Jean-Louis Chaboche. Fe2 multiscale approach for modelling the elastovis-coplastic behaviour of long fibre sic/ti composite materials. *Computer methods in applied mechanics and engineering*, 183(3-4):309–330, 2000.

[25] AR Melro, PP Camanho, FM Andrade Pires, and ST Pinho. Micromechanical analysis of polymer composites reinforced by unidirectional fibres: Part i–constitutive modelling. *International Journal of Solids and Structures*, 50(11-12):1897–1905, 2013.

[26] Gmsh - a three-dimensional finite element mesh generator. `https://gmsh.info/`. Accessed: 19-03-2020.

[27] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[28] Genevieve B Orr and Klaus-Robert Müller. *Neural networks: tricks of the trade, Chapter 1.4.3*. Springer, 2003.

[29] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[30] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[31] N. Azizan R. and B. Hassibi. Image stochastic gradient descent:. Accessed: 09-04-2020.

[32] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[33] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[34] Juan D Rodriguez, Aritz Perez, and Jose A Lozano. Sensitivity analysis of k-fold cross validation in prediction error estimation. *IEEE transactions on pattern analysis and machine intelligence*, 32(3):569–575, 2009.

[35] Thomas Bayes. Lii. an essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, frs communicated by mr. price, in a letter to john canton, amfr s. *Philosophical transactions of the Royal Society of London*, (53):370–418, 1763.

[36] Diederik P Kingma and Max Welling. An introduction to variational autoencoders. *arXiv preprint arXiv:1906.02691*, 2019.

[37] Stephen Odaibo. Tutorial: Deriving the standard variational autoencoder (vae) loss function. *arXiv preprint arXiv:1907.08956*, 2019.

[38] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. 2016.

[39] Matthias Seeger. Bayesian modelling in machine learning: A tutorial review. Technical report, 2006.

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

[41] Wray L Buntine and Andreas S Weigend. Bayesian back-propagation. *Complex systems*, 5(6):603–643, 1991.

[42] Kumar Shridhar, Felix Laumann, and Marcus Liwicki. A comprehensive guide to bayesian convolutional neural network with variational inference. *arXiv preprint arXiv:1901.02731*, 2019.

[43] Meire Fortunato, Charles Blundell, and Oriol Vinyals. Bayesian recurrent neural networks. *arXiv preprint arXiv:1704.02798*, 2017.

[44] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

[45] Hans Henrik Thodberg. A review of bayesian neural networks with an application to near infrared spectroscopy. *IEEE transactions on Neural Networks*, 7(1):56–72, 1996.

[46] William D Penny and Stephen J Roberts. Bayesian neural networks for classification: how useful is the evidence framework? *Neural Networks*, 12(6):877–892, 1999.

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

[48] Peter I Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

[49] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[50] Dipti Jasrasaria and Edward O Pyzer-Knapp. Dynamic control of explore/exploit trade-off in bayesian optimization. In *Science and Information Conference*, pages 1–15. Springer, 2018.

[51] Padmavathi Kora and Priyanka Yadlapalli. Crossover operators in genetic algorithms: A review. *International Journal of Computer Applications*, 162(10), 2017.

[52] Agoston E Eiben, P-E Raue, and Zs Ruttkay. Genetic algorithms with multi-parent recombination. In *International Conference on Parallel Problem Solving from Nature*, pages 78–87. Springer, 1994.

[53] Edwin A Williams and William A Crossley. Empirically-derived population size and mutation rate guidelines for a genetic algorithm with uniform crossover. In *Soft computing in engineering design and manufacturing*, pages 163–172. Springer, 1998.

[54] Chris Bishop. Exact calculation of the hessian matrix for the multilayer perceptron. 1992.

[55] Wei-Liem Loh et al. On latin hypercube sampling. *The annals of statistics*, 24(5):2058–2080, 1996.

Appendices

A Meshing fiber geometries

Generally scrips for creating meshes can be created relatively easily. When it is desired to allow fibers to overlap and become one fiber, as is the case here, this becomes more complex. This appendix provides a script for such a complex case, and some comments on what parts of the script does.

A.1. General

Running a finite element analysis requires dividing a continuous geometric space into a subdivision of discrete cells, each with a simple geometry. Each cell has a stiffness property, used to assemble the stiffness matrix. These stiffness properties vary for different materials. Many algorithms exist to create a mesh from a geometric input. Here the program gmsh [26] is used. Apart from creating a mesh by inputting from a user interface, a mesh can also be created 'automatically' by creating a geometric script; a specific script with a '.geo' extension. In this thesis this results in a Python script that automatically writes the .geo script used to create the mesh. The mesh used in this example is given in Figure A.1.

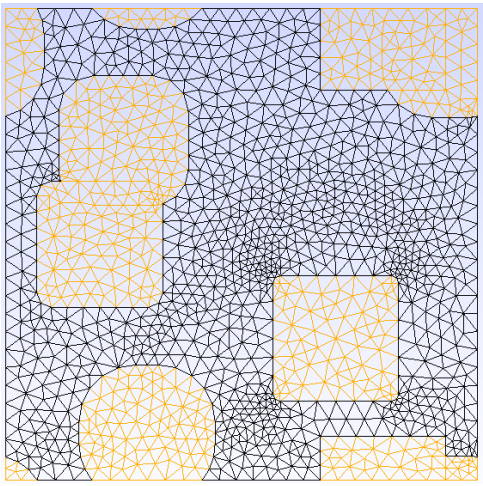


Figure A.1: Visual representation of the mesh created. Individual clusters of fibers are coloured separately.

A.2. Implementation

A requirement for the implementation presented here is using the OpenCASCADE geometry kernel instead of the default gmsh kernel. The advantage is that it allows boolean operations, allowing complex operations to be performed when running the .geo file based on knowledge not available at the moment of writing the .geo file. The resulting geo script is provided in Figure A.2, this is generated using a separate Python script. In the first three lines the kernal and the mesh element size is set. This size depends on the size of the RVE, here a box of 1x1 as seen in line 5. After defining the size, all fibers are added as Rectangles with the following properties:
Rectangle(id) = Bottom left x, Bottom left y, Bottom left z, Width, Height, Corner radius;

As can be observed, fibers that overlap the edge are duplicated with an offset of + or - 1.0 such that they also overlap on the other side. When all these are added, the result is shown in Figure A.3. Next in line 18 the BooleanIntersection operation is used, where all parts of the fibers and the 1x1 boundary that overlap are stored in the ”Fibers[]” array. The results for when these parts are meshed is visualized in Figure A.4. The opposite is done to create the matrix, using the BooleanDifference operation. The ”Delete;” tags here make sure the original fiber and box (id’s 1 to 12) are deleted. At this stage the geometry is completed. Lines 23 to 28 make sure that the correct Ids correspond to the correct Physical surface used to provide the material properties to the elements.

In line 30 the mesh is created, and line 31 is required to remove any duplicate nodes from where the matrix and fibers share nodes. Finally some lines are added in post-processing to visualize the results.

```
 1 SetFactory("OpenCASCADE");
 2 Mesh.CharacteristicLengthMin = 0.04;
 3 Mesh.CharacteristicLengthMax = 0.04;
 4
 5 Rectangle(1) = {0, 0, 0, 1, 1, 0.0};    // Boundary
 6 Rectangle(2) = { 0.155910, -0.044089, 0, 0.28817, 0.28817, 0.12};
 7 Rectangle(3) = { 0.066341, 0.3663414, 0, 0.26731, 0.26731, 0.03};
 8 Rectangle(4) = { 0.112600, 0.5826006, 0, 0.27479, 0.27479, 0.075};
 9 Rectangle(5) = { 0.667065, 0.8270659, 0, 0.26586, 0.26586, 0.0};
10 Rectangle(6) = { 0.798446, 0.7684463, 0, 0.28310, 0.28310, 0.105};
11 Rectangle(7) = { 0.566884, 0.1668844, 0, 0.26623, 0.26623, 0.015};
12 Rectangle(8) = { 0.155910, 0.9559107, 0, 0.28817, 0.28817, 0.12};
13 Rectangle(9) = { 0.667065, -0.172934, 0, 0.26586, 0.26586, 0.0};
14 Rectangle(10) = { -0.2015, 0.7684463, 0, 0.28310, 0.28310, 0.105};
15 Rectangle(11) = { -0.2015, -0.231553, 0, 0.28310, 0.28310, 0.105};
16 Rectangle(12) = { 0.79844, -0.231553, 0, 0.28310, 0.28310, 0.105};
17 // Figure 0
18 Fibers[] = BooleanIntersection{ Surface{ 2:12}; }{ Surface{1};};
19 // Figure 1
20 Matrix[] = BooleanDifference{ Surface{1}; Delete; }{ Surface{2:12}; Delete;};
21 // Complete Figure
22
23 For j In {Fibers[0]:Fibers[#Fibers[]-1]}
24   Plane Surface(1000+j) = {j};
25 EndFor
26
27 Physical Surface(0) = {1000+Fibers[0]:1000+Fibers[#Fibers[]-1]};
28 Physical Surface(1) = {Matrix[0]:Matrix[#Matrix[]-1]};
29
30 Mesh 2 ;
31 Coherence Mesh;
32 Save "mesh.msh";
33 // VISUALIZATION
34 Hide "*";
35 Show { Surface{Fibers[0]:Fibers[#Fibers[]-1]};}
36 Color Orange{Surface{Fibers[0]:Fibers[#Fibers[]-1]}; }
37 Show { Surface{Matrix[0]:Matrix[#Matrix[]-1]};}
38 Color Black{Surface{Matrix[0]:Matrix[#Matrix[]-1]};}
```

Figure A.2: Example script used to mesh overlapping fibers.

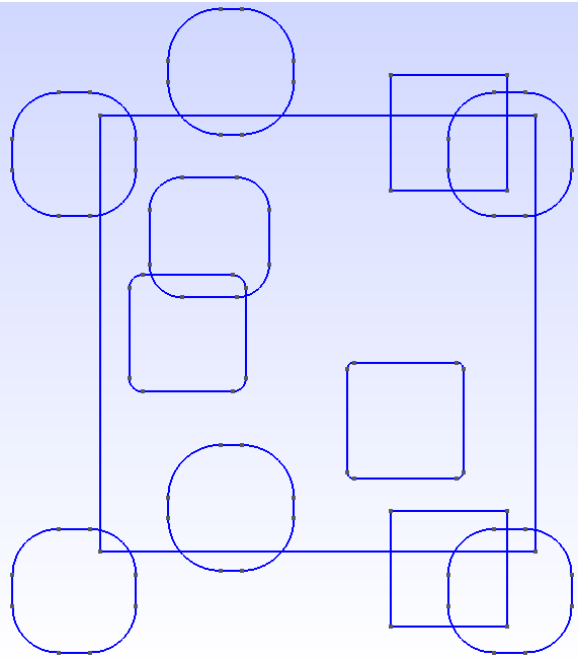


Figure A.3: Geometry after adding all fibers, related to Figure 0 in the code.

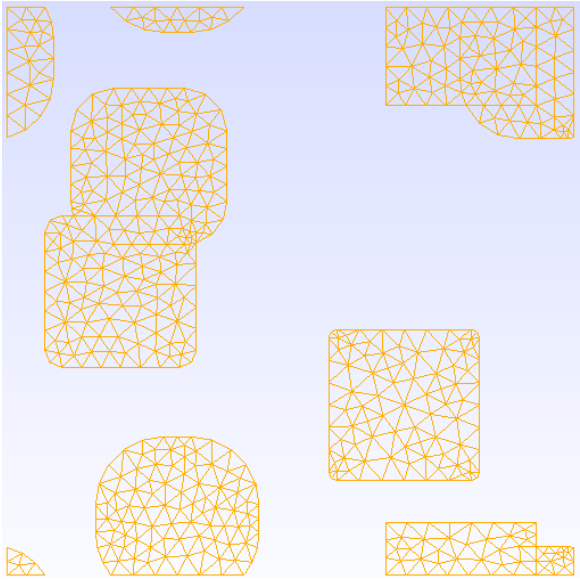


Figure A.4: Geometry after applying the BooleanIntersection operation and meshing, related to Figure 1 in the code.

B Neural network example

Network layout

This appendix serves as a numerical example of a simple neural network (NN). The network used consists out of a single input node, a single hidden node, and a single output node as presented in Figure B.1.

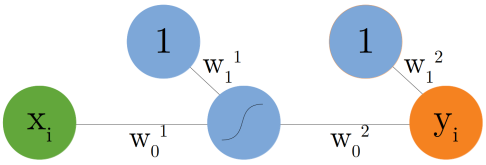


Figure B.1: Simple neural network with a single input node, a single hidden node with a sigmoid activation, and a single output node. The output node has a linear activation function. The neurons with value 1 are the bias terms, depicted as being a part of the weights. The in and outputs have index i, here specified as the index of the training sample.

This network is trained based on a dataset:

$$\mathbf{D}_i = \{\mathbf{x}_i, \mathbf{t}_i\} \tag{B.1}$$

In general these inputs and outputs are vectors. As in the simple example here they are both scalar values, their boldness is dropped. The dataset is:

$$\begin{aligned} \mathbf{D}_0 &= \{x_0 = 0.5, t_0 = 1.2\} \\ \mathbf{D}_1 &= \{x_1 = 2.0, t_1 = 0.3\} \end{aligned} \tag{B.2}$$

The dataset is in this example not normalized. The weights are randomly initialized:

$$\mathbf{w} = [w_0^1 = 0.2, w_1^1 = 0.5, w_0^2 = -0.2, w_1^2 = 0.1] \tag{B.3}$$

The equations for computing the values in hidden nodes and the output nodes are as follows:

$$z_j^l = \sum_{k=0}^{N} w_{jk}^l a_k^{l-1} \tag{B.4}$$

where the bias terms are included. This result is transformed through an activation function to form the output of that node.

$$a_j^l = f(z_j^l) \tag{B.5}$$

Forward propagation

Based on the initial weights, the input can be propagated through the network using Equations B.4 & B.5.

$$z_0^1 = x_0 w_0^1 + w_1^1 = 0.5 \times 0.2 + 0.5 = 0.6 \tag{B.6}$$

The sigmoid activation function leads to:

$$a_0^1 = \frac{e^{0.6}}{e^{0.6} + 1} = 0.6457 \tag{B.7}$$

Propagating to the output node:

$$z_0^2 = a_0^1 w_0^2 + w_1^2 = 0.6457 \times -0.2 + 0.1 = -0.0291 \tag{B.8}$$

And as a result of having a linear activation function the output is the same:

$$y_0 = a_0^2 = z_0^2 = -0.0291 \tag{B.9}$$

For the second sample in the dataset the result can similarly be computed to obtain:

$$y_1 = -0.0422 \tag{B.10}$$

These predictions are far from their known target values, the resulting mean squared error for both is:

$$E(\mathbf{x}, \mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(t_n - y_n)^2 = \frac{1}{2}\left((1.2 - -0.0291)^2 + (0.3 - -0.0422)^2\right) = 1.6279 \tag{B.11}$$

The goal is to minimize this mean squared error by adjusting the weights $\mathbf{w}$. In stochastic gradient descent, the error is computed and the weights adjusted per sample. In batch gradient descent this computation is combined in a batch. How much each weight should be adjusted follows from the chain rule of differentiation, here in the form of backpropagation.

Backward propagation

In backpropagation, the error is propagated from the output node back to the input node to compute the gradients of each weight with respect to this output error. The quantity δ_j^l is introduced as the quantity of the error in node j in layer l.

$$\delta_0^2 = \frac{\partial E}{\partial z_0^2} = \frac{\partial E}{\partial a_0^2}\frac{\partial a_0^2}{\partial z_0^2} \tag{B.12}$$

For the mean squared error function and a linear activation function this is easily computed, for the first sample it is:

$$\delta_0^2 = (y_n - t_n) \times 1 = (-0.0291 - 1.2) = -1.2291 \tag{B.13}$$

The derivative with respect to the weights in the second layer are now easily evaluated. The derivative is:

$$\frac{\partial E_i}{\partial w_{jk}^l} = \frac{\partial E_i}{\partial a_j^l}\frac{\partial a_j^l}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{jk}^l} = \delta_j^l \frac{\partial z_j^l}{\partial w_{jk}^l} \tag{B.14}$$

and recalling how z depends on w from Eq. B.4 gives:

$$\delta_j^{(l)}\frac{\partial z_j^{(l)}}{\partial w_{jk}^{(l)}} = -1.2291 * a_k^{l-1} \tag{B.15}$$

$$\frac{\partial E}{\partial w_0^2} = -1.2291 * a_0^1 = -1.2291 * 0.6457 = -0.7936 \tag{B.16}$$

Similarly for the weight representing the bias term:

$$\frac{\partial E}{\partial w_1^2} = -1.2291 * a_1^1 = -1.2291 * 1 = -1.2291 \tag{B.17}$$

With the gradients with respect to the second layer known, the error is propagated back to the hidden layer node:

$$\delta_0^1 = (w_0^2 \delta_0^2)\frac{\partial a^l}{\partial z^l} \tag{B.18}$$

As the sigmoid activation function $\sigma(x)$ is used, its derivative is:

$$\frac{\partial a^1}{\partial z^1} = \sigma'(x) = \sigma(x)(1 - \sigma(x)) \tag{B.19}$$

Resulting in:

$$\delta_0^1 = (-0.2 \times -1.2291) \times 0.6457(1 - 0.6457) = -0.05624 \tag{B.20}$$

The gradients in the first weight layer are then:

$$\frac{\partial E}{\partial w_0^1} = -0.05624 * 0.5 = -0.02812 \tag{B.21}$$

$$\frac{\partial E}{\partial w_1^1} = -0.05624 * 1.0 = -0.05624 \tag{B.22}$$

Giving the gradients $\mathbf{g}$:

$$\mathbf{g} = [-0.02812, -0.05624, -0.7936, -1.2291] \tag{B.23}$$

Updating the weights happens by adding the negative gradient multiplied with a learning factor. There are many techniques for selecting this learning factor, in this thesis the ADAM scheme is used [47]. For a learning factor of 0.1 this results in:

$$\mathbf{w} = [w_0^1 = 0.2028, w_1^1 = 0.5056, w_0^2 = -0.2794, w_1^2 = 0.2229] \tag{B.24}$$

As a single sample is considered here, this would be repeated for all samples resulting in a single epoch. For the case where only this single sample is considered, the prediction (Eq. B.9) with the new weights would improve from -0.0291 to 0.0421 for the target of 1.2.

C BNN details

C.1. Convergence criteria

C.1.1. $\mathbf{w}_{MAP}$ convergence criteria

Ideally, one uses second order computations to evaluate the convergence of weights. To prevent these computations, the relative size of the gradients with respect to its corresponding weights is used instead. This is computed as:

$$g_{diff} = \frac{g_i}{w_i} \leq \mathbf{w}_{MAP,tol} \quad | \quad \text{for all i in W} \tag{C.1}$$

where W is the number of weights. The derivative of the posterior distribution (eq. 5.8) w.r.t. $\mathbf{w}$ scales with β, and an additional term $\alpha\mathbf{w}^T$. It was found that scaling $\mathbf{w}_{MAP,tol}$ with the change in beta whenever the hyperparameters were updated resulted in more consistent behaviour:

$$\mathbf{w}_{MAP,tol,j+1} = \mathbf{w}_{MAP,tol,j}\frac{\beta_{j+1}}{\beta_j} \tag{C.2}$$

The initial value of $\mathbf{w}_{MAP,tol}$ has a significant influence on the training, and is studied in Paragraph 5.9.

C.1.2. Evidence convergence criteria

The hyperparameters α and β get updated every time $\mathbf{w}_{MAP}$ has been found. In doing so, the Hessian is computed. This is used to compute the evidence Ev, or marginal likelihood, of the network. This evidence can be seen as the fitness of the network, and converges after a number of iterations. The convergence criteria for the evidence is:

$$\frac{Ev_j - Ev_{j-1}}{Ev_j} \leq Ev_{tol} \tag{C.3}$$

The Ev_{tol} criteria influences the training of the BNN, and is studied in Paragraph 5.9.

C.2. Hessian computation

Evaluation of the posterior distribution is intractable, and is instead approximated using a Gaussian centered at a (local) maximum of that distribution. This requires second derivatives of the log posterior, for which we compute the Hessian of the sum-of-squares error function

$$E = \frac{1}{2}\sum_{n=1}^{N}(y_n - t_n)^2 \tag{C.4}$$

for which the Hessian is:

$$\frac{\partial^2 E}{\partial w_{ji}\partial w_{lk}} \tag{C.5}$$

Different techniques of computing this Hessian are explored here. Two of these methods approximate the Hessian, namely the outer product and finite differences method. After, the exact Hessian is presented. These methods are compared on accuracy and computation time.

C.2.1. Finite differences

The finite differences (FD) method can easily be implemented to approximate the Hessian at any stage during training. By making small perturbations to any weight and computing the error, gradients can be found. The general forward differences gradient formula is:

$$f'(x) = lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x)}{\epsilon} \tag{C.6}$$

For a neural network, this can be extended to compute the Hessian by applying central differences to the first derivatives of the error function, which are themselves calculated during backpropagation [32]:

$$\frac{\partial^2 E}{\partial w_{ji} \partial w_{lk}} = \frac{1}{2\epsilon}\left\{\frac{\partial E}{\partial w_{ji}}(w_{lk} + \epsilon) - \frac{\partial E}{\partial w_{ji}}(w_{lk} - \epsilon)\right\} + O(\epsilon^2) \tag{C.7}$$

Using a small ϵ generally results in a very accurate Hessian. Ranging from $\epsilon = 1e^{-9}$ to $\epsilon = 1e^{-12}$ gave little difference in results, and $\epsilon = 1e^{-11}$ is used. Smaller values give problems related to the machine precision. The network scales with $O(W^2)$ operations (W being the number of parameters in $\mathbf{w}$), and requires propagating and backpropagating twice for each element.

C.2.2. Outer product

For a single sample the Hessian written as

$$H = \nabla\nabla E = \sum_{n=1}^{N} \nabla y_n \nabla y_n + \sum_{n=1}^{N} (y_n - t_n)\nabla\nabla y_n \tag{C.8}$$

The outer product (OP) approximation exploits the fact that the Hessian only needs to be computed when a local optima of the posterior distribution is reached. As a consequence of this the second term in this equation is already minimal, and is therefore ignored in this computation leading to:

$$H \approx \sum_{n=1}^{N} y_n y_n^T \tag{C.9}$$

For every training sample the gradients are computed and the resulting outer product is summed over all training samples. Using this approximation does increase the importance of finding a good local mode.

C.2.3. Exact calculation

Using the values computed during backpropagation the Hessian can be evaluated exactly. The formulas for a neural network with a single hidden layer, and therefore two weight layers, can be separated in three blocks. Indices k, l and m represent the input, hidden and output layer respectively. The formulation is then as follows: [54]

Both weights in the first layer:

$$\frac{\partial^2 E_n}{\partial w_{lk} \partial w_{l'k'}} = z_k z_{k'} \left(f''(a_{l'})\delta_{ll'} \sum_m w_{ml'}\sigma_m + f'(a_l)f'(a_{l'}) \sum_m w_{ml'} w_{ml} H_m \right) \tag{C.10}$$

Both weights in the second layer:

$$\frac{\partial^2 E_n}{\partial w_{ml} \partial w_{m'l'}} = z_l z_{l'} \delta_{mm'} H_{m'} \tag{C.11}$$

One weight in each layer:

$$\frac{\partial^2 E_n}{\partial w_{lk} \partial w_{ml'}} = z_k f'(a_l)\left(\sigma_m \delta_{ll'} + z_{l'} w_{ml} H_m\right) \tag{C.12}$$

where:
a_i is the value of node i.

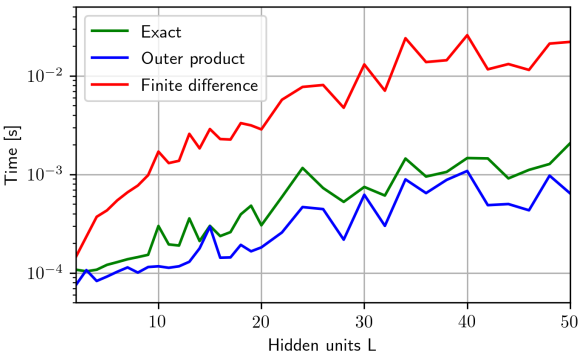


Figure C.1: The computation time for an exact evaluation, the outer product approximation and the finite differences approximation ($\epsilon = 1e-11$).

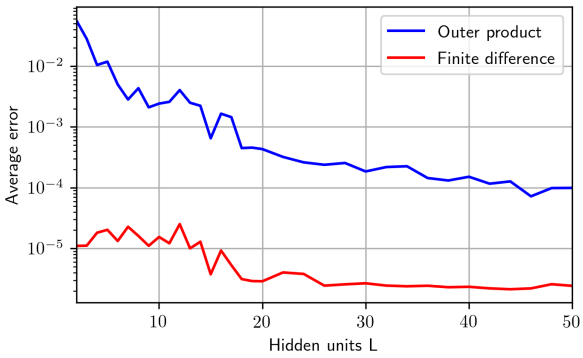


Figure C.2: The average error per Hessian term for the outer product approximation and the finite differences approximation ($\epsilon = 1e-11$) compared to the exact evaluation.

z_i is the activation output of node i (set to 1.0 when the node is a bias)
$f'(a_i)$ is the first derivative of a_i depending on the activation function of that node.
$f''(a_i)$ is the second derivative of a_i depending on the activation function of that node.
$\delta_{ii'}$ is the kronecker-delta symbol, i.e. 1.0 when $i = i'$, and 0.0 otherwise.

$$H_m \equiv \frac{\partial^2 E_n}{\partial a_n^2} = f''(a_m)\frac{\partial E_n}{\partial z_m} + (f'(a_m))^2 \frac{\partial^2 E_n}{\partial z_m^2}$$

C.2.4. Comparison

The difference in accuracy and computation time is evaluated for the aforementioned methods. All methods rely on computing the Hessian for each individual training sample, and then summing to get the complete Hessian, therefore scaling linearly with respect to the samples N. A more significant difference comes from the number of parameters W in the network, directly influencing the $W \times W$ Hessian size. A case is created with a single input node, a variable number of hidden nodes L, and a single output nodes. The number of parameters for this network is $W = 3L + 1$.

In Figure C.1 the computation time for various numbers of hidden units is presented. The OP approximation requires the least computational effort. The FD method requires about an order of magnitude more computation time. Next to the computation time, also the accuracy of the approximation methods is studied. The absolute difference between the approximation methods and the exact calculation is evaluated and averaged over all $W \times W$ elements, plotted in Figure C.2. It is clear that the FD method provides a much more accurate approximation. During training the OP approximation becomes increasingly accurate as the neglected terms decrease.

Both the OP and FD approximations are easy to implement. The OP can serve a purpose when a quick rough approximation is required. For a more precise approximation the FD can be used at the cost of an increase of computation time. Evaluating the exact Hessian requires more work to implement, but once implemented there is no reason to use either of the approximation methods. It is not constrained by requiring a local maximum to be found and does not depend on any external parameters (such as ϵ for FD). Updating the hyperparameters requires the eigenvalues of the Hessian. These eigenvalues have shown to be very sensitive to the Hessian, therefore even a small error from the FD method can lead to significant differences.

C.3. Eigenvalue computation

The Bayesian neural network requires the effective number of parameters γ as presented in 5.12 and repeated here for clarity of the following section.

$$\gamma = \sum_{i=1}^{W} \frac{\lambda_i}{\alpha + \lambda_i} \tag{C.13}$$

Here α is the hyperparameter, a constant value here. W is the number of rows or columns in H, equal to the number of weights. The eigenvalues λ_i are computed from the Hessian:

$$\beta \mathbf{H}\mathbf{u}_i = \lambda_i \mathbf{u}_i \tag{C.14}$$

In the theory for this framework the Hessian is expected to be positive definite [32], meaning by definition that all eigenvalues are positive. In the application for a neural network, this can be problematic, since second order terms are expected to be positive only when a (local) optima has been found. In the Laplace framework the Hessian does only need to be evaluated when a local minimum has been found. Still, it can occur that the Hessian is not positive definite. Computing the eigenvalues exactly based on the Hessian still results in a number of negative eigenvalues. For a typical training of a network, the Hessian and its eigenvalues are computed once every 10 epochs, and the result plotted in Figure C.3. The resulting prediction and the corresponding eigenvalues are presented in Figures C.4 and C.5 respectively. The network consists of 27 hidden nodes, resulting in 81 network parameters. The dataset contains 83 samples. For this training, the hyperparameters are computed with all eigenvalues below a threshold of $1e^{-10}$ not considered. From eq. C.13 one can observe that this is equivalent to setting the eigenvalue to 0.0. The end result seems sensible, and the evidence increases as expected.

The eigenvalues at the end of training are:

$$[-0.0007918, -0.0005479, -0.0003553, -0.0003539, -0.0003385, -0.0003285, -0.0003261, -0.0002838, -2.26e^{-05}, -5 \tag{C.15}$$

Next to the hyperparameters, the eigenvalues are also required for computing the evidence. In particular the determinant, the product of the eigenvalues, is required. In network with redundant weights the Hessian can be near-singular. Eigenvalues of a near-singular matrix can be in the order of machine precision, resulting in an unreliable determinant. One option is to reconstruct the Hessian using only those eigenvalues above a certain threshold [45].

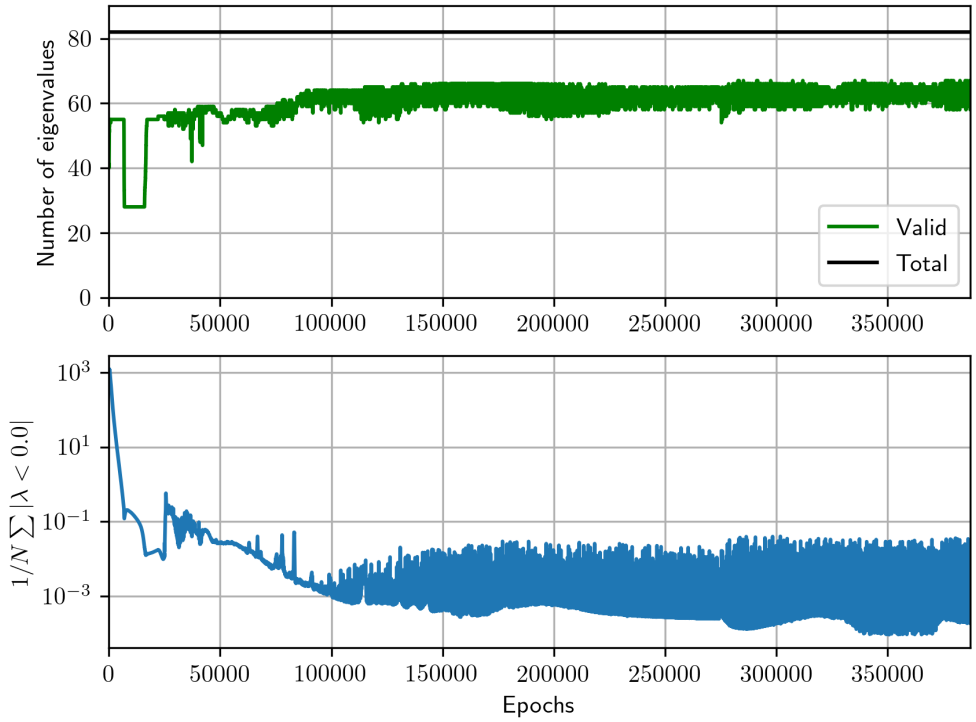


Figure C.3: In the top plot the number of $\lambda > 0.0$. For all $\lambda < 0.0$ the bottom plot gives the absolute average. The number of negative eigenvalues decreases, and the absolute average value of negative eigenvalues also decreases to insignificant values during training.

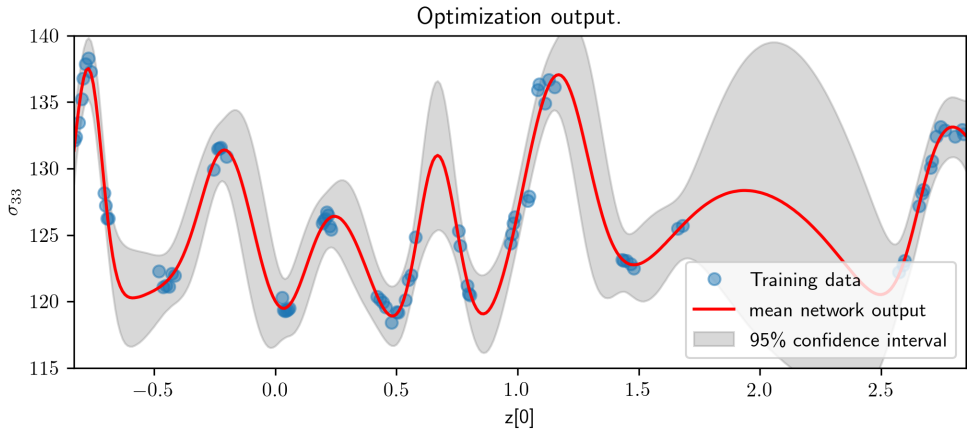


Figure C.4: Network prediction at the end of training. Network with 27 hidden nodes, the dataset contains 83 samples.

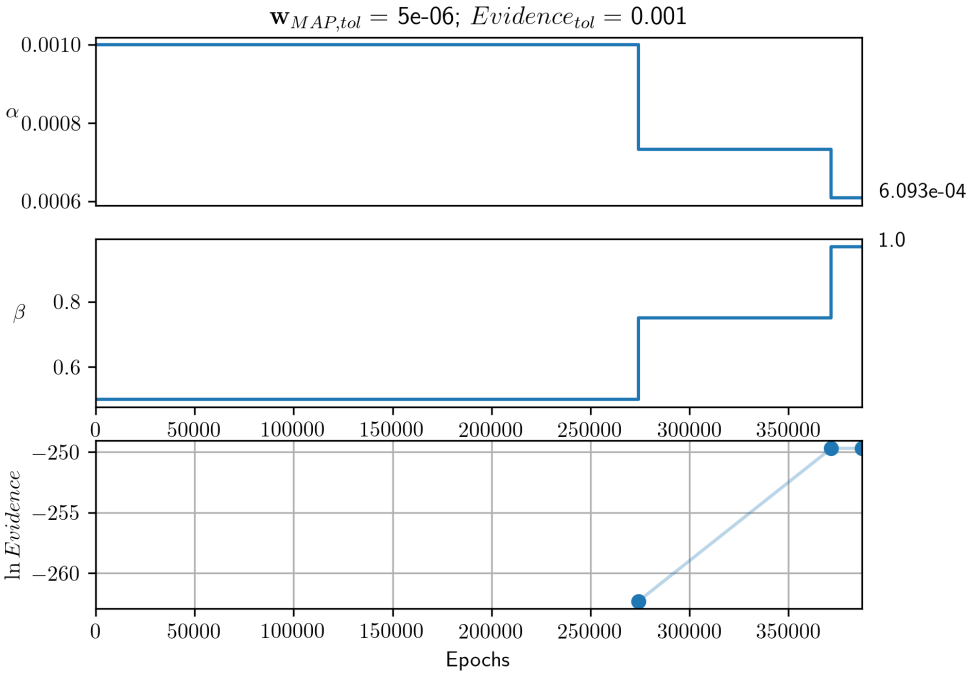


Figure C.5: Hyperparameters α and β and the log evidence during training.

D Latin hypercube sampling

Latin hypercube sampling was proposed in 1979 by McKay et al. as an alternative random sampling in computer experiments [55]. It generates near-random samples from a multidimensional distribution. It can be imagined easily for 2 dimensions, where for N samples a $N \times N$ grid is created. Now the N samples are spread in this grid such that there is only one sample in each row and column, as visualized in Figure D.1. It is implemented here such that within the boundaries of each grid, the sample takes a random point.

An additional step can be taken to get Orthogonal sampling, of which LHC is a subset. In Orthogonal sampling all points are chosen simultaneously resulting in a LHC sample such that each subspace is sampled with the same density. This is not applied in this thesis.

LHC defines a boundary for each sample with its grid, and it is implemented here such that within those boundaries the samples are randomly chosen.

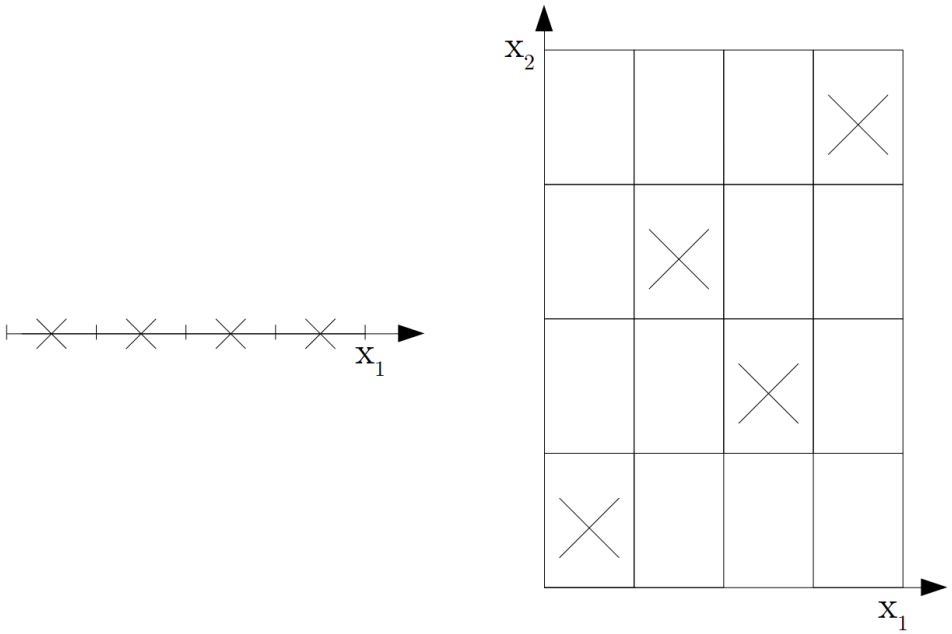


Figure D.1: Example of Latin hypercube sampling for 1 and 2 dimensions. As the dimensionality increases, the samples does not have to increase. The example on the right shows that dimensions can have different scales.