TOWARDS UNCERTAINTY QUANTIFICATION OF ADVANCED NEURAL NETWORK MODELS

by

SUDEEP DHANANJAY CHAVARE

A dissertation submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY IN MECHANICAL ENGINEERING

2026

Oakland University
Rochester, MI 48307

Doctoral Advisory Committee:
Zissimos P. Mourelatos, Ph.D. (chair)
Michael Latcha, Ph.D.
Vijitashwa Pandey, Ph.D.
Dorin Drignei, Ph.D.

To the memory of my Father-in-law,

Madhusudan Mohaniraj Joshi (1953-2023)

ACKNOWLEDGMENTS

First and foremost, I would like to express my deepest gratitude to my advisor, Dr. Zissimos Mourelatos, for his unwavering support, insightful guidance, and mentorship throughout my Ph.D. journey. His profound knowledge, thoughtful advice, and continuous encouragement have been a constant source of inspiration during my years at Oakland University.

I would also like to sincerely thank the members of my dissertation committee — Dr. Michael Latcha, Dr. Vijitashwa Pandey, and Dr. Dorin Drignei — for their valuable suggestions, constructive feedback, and generous support whenever needed. Their expertise has greatly enriched my research and academic growth.

I owe my heartfelt thanks to my family, whose love and support have been the bedrock of my journey. To my parents, for instilling in me the values of perseverance, hard work, and education from an early age; to my wife Mrunmayee for being my constant source of understanding, and strength; and to our son Divij, who have been a continuous reminder of the joy and purpose behind this endeavor. Their patience, sacrifices, and unwavering belief in me provided the motivation to pursue and complete this journey.

Sudeep Dhananjay Chavare

ABSTRACT

Towards Uncertainty Quantification of Advanced Neural Networks

By

Sudeep Dhananjay Chavare

Adviser: Dr. Zissimos P. Mourelatos, PhD.

Machine learning has witnessed widespread adoption across various domains, bringing about transformative changes in decision-making, trend prediction, task automation, and personalized experiences. Despite the remarkable predictive capabilities of machine learning models, the associated uncertainty in their predictions remains a critical concern. Uncertainty estimation plays a pivotal role in ensuring robust decision-making, going beyond mere outcome prediction to quantify the model's confidence and potential error. This research first presents a review of existing uncertainty quantification techniques in machine learning, including Monte Carlo dropout and ensemble methods, highlighting their advantages in addressing uncertainty as well as their limitations. The dissertation is centered on advancing methods for model uncertainty quantification (UQ) in machine learning. In particular, two novel approaches are proposed to address critical gaps in the existing body of work.

The first method operates within the framework of Neural Network Ensembles, wherein multiple neural networks are trained using identical hyperparameters, with variations introduced through different subsets of the training data. Ensemble predictions are subsequently generated, and a novel adaptive Gaussian Process Regression (GPR) model is developed to estimate the associated uncertainty bounds. The adaptive GPR model offers accurate and efficient estimation of uncertainty, making it highly suitable for real-time applications involving either scalar outputs or time-series data. Through this development, the dissertation aims to contribute practical and computationally viable solutions to the field of uncertainty quantification, enhancing the reliability of decision-making in complex and dynamic environments.

The second method, while also based on Neural Network Ensembles, is fundamentally different in its objective. After training an ensemble of neural networks, a "Most Probable Model" is constructed to provide a fast and accurate estimation of the mean prediction. Unlike the first method, this approach does not quantify prediction uncertainty or provide confidence bounds. Instead, it offers a significant reduction in computational overhead associated with ensemble averaging, making it particularly advantageous in scenarios where rapid mean prediction is critical. In specific problem domains—further elaborated in subsequent chapters—this method may represent a practical alternative when full uncertainty quantification is either infeasible or unnecessary.

Both methods proposed in this work are highly dependent on the accuracy and coverage of the ensemble models. If the ensemble does not capture the ground truth, these methods become ineffective. To mitigate this limitation, an autoencoder-based

"Trust Index" is introduced to flag out-of-distribution inputs, providing a warning when the input data fall outside the range of the training set.

All the aforementioned methods require training multiple models to form an ensemble of predictions, which can be computationally expensive and time-consuming. To address this limitation, a novel approach based on Conditional Variational Autoencoders (CVAEs) is proposed, wherein synthetic data are generated to emulate the ensemble predictions. This strategy effectively reduces the number of models that need to be trained while maintaining comparable predictive diversity and accuracy.

Collectively, these contributions aim to advance the current capabilities in machine learning by introducing accurate uncertainty estimation methods alongside efficient mean prediction strategies, offering both theoretical insights and practical benefits for real-world applications.

TABLE OF CONTENTS

TABLE OF CONTENTS—Continued

TABLE OF CONTENTS—Continued

TABLE OF CONTENTS—Continued

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES—Continued

LIST OF FIGURES—Continued

LIST OF FIGURES—Continued

LIST OF ABBREVIATIONS

ML	Machine Learning
AI	Artificial Intelligence
ANN	Artificial Neural Networks
DNN	Deep Neural Network
LSTM	Long Short-Term Memory
CAE	Computer Aided Engineering
PiNN	Physics Informed Neural Networks
UQ	Uncertainty Quantification
GPR	Gaussian Process Regression
BNN	Bayesian Neural Networks
NN	Neural Networks
MCMC	Markov Chain Monte Carlo
VI	Variational Inference
MC	Monte Carlo
OoO	Out-of-Distribution
DoF	Degrees of Freedom
CFD	Computational Fluid Dynamics
$\mathbb{I}$	Identity Matrix
$\mathcal{N}$	Gaussian /Normal Distribution
σ^2	Variance of underlying Gaussian /Normal Distribution

μ	Mean of underlying distribution
$p(\theta)$	Prior distribution
SGD	Stochastic Gradient Decent
Tanh	Hyperbolic tangent
Relu	Rectified Linear Unit
CNN	Convolutional Neural Networks
RNN	Recurrent Neural Networks
$k(x_i, x_j)$	Covariance Function
$\mathbb{E}$	Expectation of Distribution
RBF	Radial Basis Function
GA	Genetic Algorithm
PSD	Power Spectral Density
KL	Karhunen-Loeve Expansion
PDF	Probability Density Function
CDF	Cumulative Distribution Function

CHAPTER 1

INTRODUCTION

1.1 Motivation and Background

Machine Learning (ML) is a branch of artificial intelligence as it can be seen from Figure 1.1 that equips computers to learn patterns from data and make decisions without explicit programming. It encompasses three main types: supervised learning, unsupervised learning, and reinforcement learning. Supervised learning involves training models on labeled data, while unsupervised learning discovers patterns in unlabeled data. Reinforcement learning trains models to make decisions through rewards and punishments.

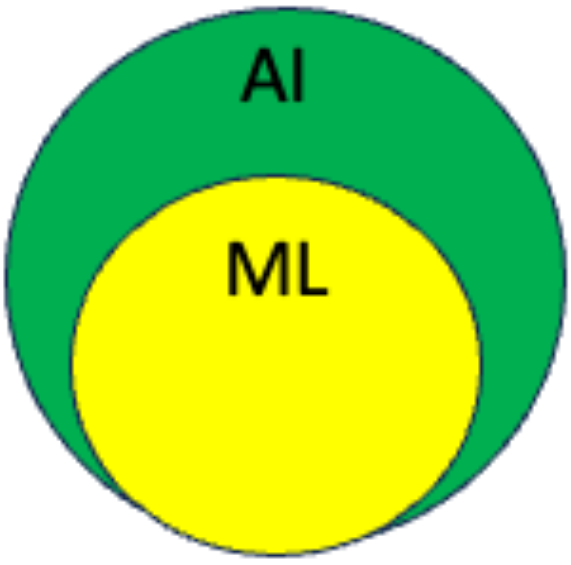

Figure 1.1 ML as a branch of AI

Machine Learning (ML) has been utilized for over three decades, with response surface models for example, employing techniques such as Kriging and radial basis

functions, successfully applied in various optimization problems. The emergence of relatively new methodologies, such as Deep Neural Networks (DNN), Long Short-Term Memory (LSTM), and random forests, among others, has significantly expanded its utilization, finding applications in fields such as image and pattern recognition, which would have seemed unrealistic three decades ago. Particularly in Computer-Aided Engineering (CAE), ML has found novel applications, notably using Physics-Informed Neural Networks (PiNN) [1]. In the automotive industry, CAE tools are extensively used in numerous applications, notably in predicting crash performance by simulating finite element models of vehicles in crash events. Although these CAE simulations are time-consuming, often requiring 24 to 48 hours depending on available computing resources, they remain significantly more efficient in terms of cost and time compared to actual vehicle crash tests. Machine learning has introduced tools that can generate crash simulation animations in minutes without executing actual CAE simulations of real vehicle crash tests [2]. A sample example of crash animation prediction by a commercial ML tool Altair Physics AI along with its comparison with CAE simulation result can be seen from Figure 1.2.

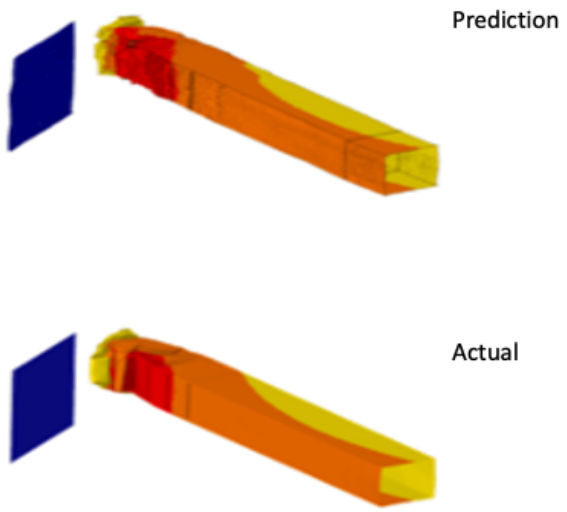

Figure 1.2 Example of machine learning used to predict crash simulation

However, if uncertainty, inherent in ML predictions, is not adequately accounted for, decisions made based on these predictions may lead to catastrophic consequences highlighting the importance of uncertainty quantification (UQ). If decisions based on ML predictions have real-world consequences, such as healthcare, finance, and environmental science, understanding and quantifying uncertainty becomes important. Without it, stakeholders may make decisions based on overly confident but potentially flawed predictions, leading to significant risks and liabilities. Therefore, integrating robust UQ techniques into ML models is crucial for ensuring the reliability, safety, and ethical integrity of decision-making processes in today's data-driven world. By quantifying uncertainty, stakeholders can better assess the reliability of model predictions and make informed decisions that consider potential risks. Moreover, UQ fosters trust and transparency in ML systems, enhancing their acceptance and adoption in critical applications. It also facilitates model improvement by identifying areas of uncertainty

where additional data collection or model refinement may be necessary. Ultimately, incorporating UQ is essential for mitigating risks, ensuring decision-making confidence, and advancing the responsible deployment of artificial intelligence (AI) technologies across diverse sectors.

This proposed research first discusses the aleatory and epistemic uncertainties. Various techniques have been proposed, among which Gaussian Process Regression (GPR) stands out for its capability to quantify both aleatory and epistemic uncertainty [3]. GPR is a generalized Bayesian inference, extending from an inference about a finite set of random variables to the inference of functions, each representing an infinite-dimensional vector of random variables [4]. This generalized Bayesian inference uses a joint probability distribution of a random function, in contrast to a joint distribution of a finite-dimensional random vector. Comprehensive and critical reviews of GPR are available among others, in Rasmussen [4], Brochu et al. [5], and Shahriari et al. [6]. For a thorough understanding of GPR, readers are directed to Rasmussen's seminal textbook [4]. A Gaussian process constitutes a collection of random variables across a defined domain, where any finite subset of these variables adheres to a joint (multivariate) Gaussian distribution. Intuitively, the Gaussian process also defines a probability distribution for an unknown function, composed of infinitely many random variables. However, standard GPR typically struggles to scale efficiently with large training datasets due to its training complexity [7].

Bayesian Neural Networks (BNNs) offer a probabilistic framework for quantifying uncertainty in ML predictions. Unlike traditional neural networks, BNNs treat model parameters (e.g. NN weights) as random variables, allowing for the

estimation of the entire distribution of plausible parameter values given the observed data. This is achieved through Bayesian training, where a prior distribution is updated with training data to obtain a posterior distribution. By capturing uncertainty in parameter estimates, BNNs inherently provide a measure of predictive uncertainty. However, methods like Markov Chain Monte Carlo (MCMC) and Variational Inference (VI), commonly used for approximating the posterior distribution in BNNs, are computationally expensive [8–10]. MCMC, while theoretically appealing due to its asymptotic convergence to the true posterior, can be particularly challenging to implement efficiently, especially for high-dimensional parameter spaces. VI, on the other hand, approximates the posterior within a parameterized family of distributions, which can lead to a loss of fidelity in the representation of uncertainty.

Proposed initially as a regularization technique to mitigate overfitting in DNNs [11], Monte Carlo (MC) dropout has been demonstrated to approximate the posterior predictive distribution within a specific Bayesian framework [12]. The connection between MC dropout and Bayesian inference is elucidated in [12, 13]. An advantage of MC dropout lies in its simplicity of implementation, requiring minimal modifications, typically involving the insertion of dropout layers, to an existing DNN setup. Moreover, it is readily available as a standard component in many programming environments. We provide a brief overview of MC dropout in Section 2.1 and highlight its advantages and disadvantages in Section 2.3 using a simple mathematical example.

Ensemble learning, widely recognized for addressing overfitting and bolstering the generalization of ML models [14], entails training multiple independent models and amalgamating their predictions [15]. Diversity among models is imperative for

performance enhancement, achievable through randomization methods like bagging and boosting techniques [16, 17]. In deep learning, neural network ensembles are obtained by training multiple networks with identical architectures. These ensembles are utilized to characterize prediction uncertainty [18, 19], demonstrating improved uncertainty for Out-Of-Distribution (OOD) data [20]. Exploiting this characteristic, they serve in detecting dataset shifts [20-22], sustaining model robustness in evolving environments.

This research first provides a brief introduction to aleatory and epistemic uncertainty in Section 2. Subsequently, the MC dropout and Network Ensemble techniques for quantifying uncertainty in ML models are discussed in Section 3 and a mathematical example featuring 2 degrees of freedom (DOF), is used to demonstrate their limitations. Section 4 provides the dissertation objectives. In Section 5, we introduce a novel method to quantify uncertainty by extending the Network Ensemble approach coupled with Gaussian Process Regression. Initially, the efficacy of the new approach will be demonstrated on the same 2-DOF mathematical problem. Subsequently, its effectiveness is evaluated on a vehicle dynamics problem in Section 6. Sections 7 and 8 provide a brief description of remaining work and dissertation contributions, respectively.

1.2 Dissertation Objectives

In general, the dissertation objectives are summarized below:

- Review and analyze existing techniques for quantifying uncertainty in machine learning models. Identify the advantages and shortcomings of each method to gain a comprehensive understanding of the state-of-the-art.
- Propose a novel technique that combines the strengths of the network ensemble and Gaussian Process Regression (GPR) methods while mitigating their respective disadvantages. This proposed technique aims to leverage the benefits of both approaches to enhance the accuracy and reliability of uncertainty quantification in machine learning models.
- Develop a UQ method for neural networks using autoencoders. The latter are a promising nonlinear dimension reduction method which can be suitable for large-dimension scalar problems and dynamic problems.
- Demonstrate all developed UQ methods using representative scalar and dynamic examples.

CHAPTER 2

WHAT IS UNCERTAINTY

This section discusses what uncertainty is and its types. Figure 2.1 shows an input-output model. Both the input variables and the model are deterministic. Because the inputs **X** are known precisely and the model f(x) is deterministic, the outputs **Y** are also deterministic. There is no consideration of randomness or variability in the modeling process. This approach assumes perfect knowledge of both the system and the operational environment. An example is the prediction of the deflection of a simply supported beam using classical beam theory, where the material properties (such as Young's modulus) and the applied loads are known exactly without any variability.

Figure 2.1 Deterministic inputs with deterministic model

Figure 2.2 shows a case where the input is uncertain, and the model f(x) remains deterministic. The inputs **X** are characterized by inherent variability for example and are typically modeled using probability distributions. Consequently, the outputs **Y** are uncertain even though the underlying model is deterministic. An example is the

prediction of the fatigue life of a mechanical component where the loading conditions vary randomly, but the fatigue life prediction model is deterministic.

Figure 2.2 Uncertain inputs and deterministic model

Figure 2.3 addresses model uncertainty. The inputs **X** are deterministic while the model f(x) is uncertain due to model form approximations. As a result, even with deterministic inputs, the outputs Y are uncertain. The research in this dissertation is based on this case of deterministic input and an uncertain model due to model uncertainty. An example is the prediction of crash energy absorption of a vehicle structure using a trained neural network model. While the design parameters (e.g., material properties and geometry) are known exactly, the model predictions vary due to epistemic uncertainty stemming from limited training data and modeling assumptions.

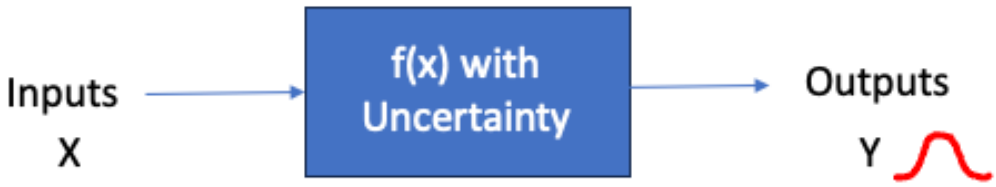

Figure 2.3 Deterministic inputs and uncertain model

Finally, the most comprehensive configuration is shown in Figure 2.4 where both the inputs and the model are uncertain. The outputs Y thus exhibit compounded variability arising from both the inputs and the model. Addressing both types of uncertainty simultaneously is an active area of research representing a natural progression for future work beyond the scope of this dissertation. An example is the prediction of the aerodynamic performance of a new vehicle design where input conditions such as wind speed, ambient temperature, and surface roughness are uncertain, while the Computational Fluid Dynamics (CFD) model used for prediction exhibits model-form uncertainty due to simplifications and numerical approximations.

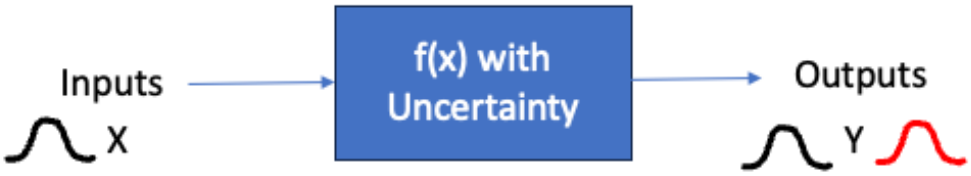

Figure 2.4 Uncertain inputs and model

2.1 Types of Uncertainties

This section summarizes the definitions and origins of aleatory uncertainty and epistemic uncertainty [23, 24]. Aleatory uncertainty represents inherent variability which is irreducible [24]. Examples are the natural system noise in sensor measurements [25] and material variability. In ML, it represents the stochastic nature of inputs, outputs, and

their relationship [23]. Aleatory uncertainty is typically integrated into likelihood functions, rendering ML model predictions probabilistic [25]. This is considered by various UQ methods such as homoscedastic and heteroscedastic Gaussian Process Regressions (GPRs) and neural network ensembles. Figure 2.5 shows a schematic representation of aleatory and epistemic uncertainty.

Unlike epistemic uncertainty which stems from limited knowledge, aleatoric uncertainty is irreducible, representing the inherent indeterminacy in certain aspects of a phenomenon. In the context of machine learning, especially in tasks like regression, aleatoric uncertainty acknowledges the inherent variability in predictions that is beyond the model's capacity to precisely account for. This concept is crucial for modeling complex, dynamic systems where certain elements exhibit natural randomness, such as financial markets, weather patterns, or biological processes. Understanding aleatoric uncertainty provides a more realistic perspective, enhancing the adaptability and reliability of models in diverse and unpredictable real-world scenarios. For instance, consider an archer aiming for a target. Despite using the same force and aim, two shots may not be exactly identical due to environmental factors such as wind and humidity. Aleatoric uncertainty can be further classified into homoscedastic and heteroscedastic uncertainty. Homoscedastic uncertainty maintains the same distribution regardless of inputs, while heteroscedastic uncertainty depends on the input and its statistical properties.

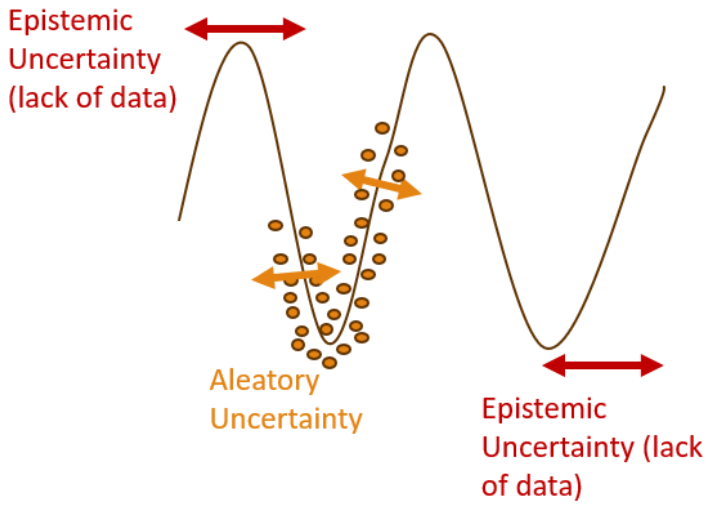

Figure 2.5 Types of uncertainties

Epistemic uncertainty in engineering and ML models is due to lack of information arising from factors such as model simplification, and computational assumptions, among others. ML models share similar sources of uncertainty as engineering models, including model-form uncertainty and parameter uncertainty. The former relates to simplification and approximation procedures in ML model construction, while the latter arises from model calibration and training processes. Epistemic uncertainty can be reduced through increased knowledge and additional data [24].

2.2 Decomposition of Uncertainty

Based on the previous discussion, it is beneficial to decompose the total uncertainty into its aleatory and epistemic components [2]. Let us consider a simple linear regression model as

$$\hat{y}(\boldsymbol{x}) = f(\boldsymbol{x}; \boldsymbol{\theta}) = \boldsymbol{\theta}^T \boldsymbol{x} + \epsilon \tag{2.1}$$

where $\epsilon \sim \mathcal{N}(0, \sigma^2 \mathbf{I})$ is noise represented by a Gaussian distribution with zero mean. $\mathbf{I}$ is the D x D identity matrix. By employing a Bayesian approach, we use a prior distribution $p(\boldsymbol{\theta})$ concerning the model parameters $\boldsymbol{\theta}$. Subsequently, we deduce a posterior distribution from a training dataset $\mathcal{D}$ denoted as $p(\boldsymbol{\theta}|\mathcal{D})$. In essence, this involves constructing a Bayesian linear regression model, allowing us to derive the predictive distribution of y at a specific input x.

For demonstration purposes, let us consider a two-dimensional model (i.e. $D = 2$) with a posterior joint distribution $p(\boldsymbol{\theta}|\mathcal{D}) = \mathcal{N}(\mu_\theta, \boldsymbol{\Sigma}_\theta)$ where μ_θ is the mean and $\boldsymbol{\Sigma}_\theta$ is the covariance matrix

$$\boldsymbol{\Sigma}_\theta = \begin{bmatrix} {\sigma_{\theta_1}}^2 & \rho\sigma_{\theta_1}\sigma_{\theta_2} \\ \rho\sigma_{\theta_1}\sigma_{\theta_2} & {\sigma_{\theta_2}}^2 \end{bmatrix} \tag{2.2}$$

The predicted y then follows a Gaussian distribution as

$$p(y|\boldsymbol{x}, \mathcal{D}) = \mathcal{N}(\mu_\theta, {\sigma_{\theta_1}}^2 {x_1}^2 + {\sigma_{\theta_2}}^2 {x_2}^2 + 2\rho\sigma_{\theta_1}\sigma_{\theta_2} x_1 x_2 + \sigma^2) \tag{2.3}$$

The total uncertainty is provided by the predictive variance as

$$u_{total} = Var(y|\boldsymbol{x}, \mathcal{D}) = \sigma_{\theta_1}^2 x_1^2 + \sigma_{\theta_2}^2 x_2^2 + 2\rho\sigma_{\theta_1}\sigma_{\theta_2} x_1 x_2 + \sigma^2 \tag{2.4}$$

The aleatory uncertainty can be measured by the variance of the noise σ^2; i.e.

$$u_{aleatory} = \sigma^2 \tag{2.5}$$

resulting in the following epistemic uncertainty

$$u_{epistemic} = u_{total} - u_{aleatory} = \sigma_{\theta_1}^2 x_1^2 + \sigma_{\theta_2}^2 x_2^2 + 2\rho\sigma_{\theta_1}\sigma_{\theta_2} x_1 x_2 \tag{2.6}$$

The primary focus of this research is on model uncertainty, which refers to the variability in model predictions arising from imperfections and limitations within the model itself. Specifically, the type of uncertainty addressed in this work is epistemic uncertainty, which stems from incomplete knowledge about the true underlying system or process being modeled. Unlike aleatory uncertainty, which captures inherent randomness in the physical world and is irreducible, epistemic uncertainty can, in principle, be reduced through the acquisition of more information, improved modeling techniques, or more sophisticated training strategies.

In the context of this research, epistemic uncertainty arises due to variability in the weights of neural networks. When training neural networks, particularly with stochastic optimization algorithms such as Adam or stochastic gradient descent (SGD), there is no guarantee of convergence to a unique global minimum. Rather, factors such as random initialization, the stochastic nature of the optimization process, and the availability of limited training data lead to different sets of learned weights across different training runs.

These different weight configurations represent slightly different approximations of the true underlying function, resulting in variability in the model's predictions even when the input remains fixed.

A more detailed discussion of neural networks, their architecture, the role of weights, and the sources of weight variability will be presented in the next chapter to provide a stronger foundation for understanding how model uncertainty is captured and analyzed in this work.

CHAPTER 3

BASICS OF NEURAL NETWORKS

Neural networks have emerged as a powerful class of models capable of approximating highly complex and nonlinear relationships between input and output spaces. They achieve this flexibility through networks of interconnected computational units — neurons — whose behavior is governed by weights and biases which are learned parameters. In the context of this research, understanding neural networks is critical because the weight variability leads directly to model uncertainty, the primary focus of this dissertation. The weight variability is due to different optimal solutions the gradient-based training optimization algorithm produces.

3.1 Structure of a Neural Network

A typical feedforward neural network consists of an input layer, one or more hidden layers, and an output layer as shown in Figure 3.1. Each neuron computes a weighted sum of its inputs, adds a bias, and applies a nonlinear activation function to produce its output [47].

Mathematically, the computation performed by a neuron can be expressed as

$$y = \phi\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{3.1}$$

where y is the output, x_i are the input features, w_i are the corresponding weights, b is the bias, and $\phi(.)$ is the activation function. The summation is over all neurons of each layer.

The weights and biases are learnable parameters that define how the inputs are transformed to determine the output of the network.

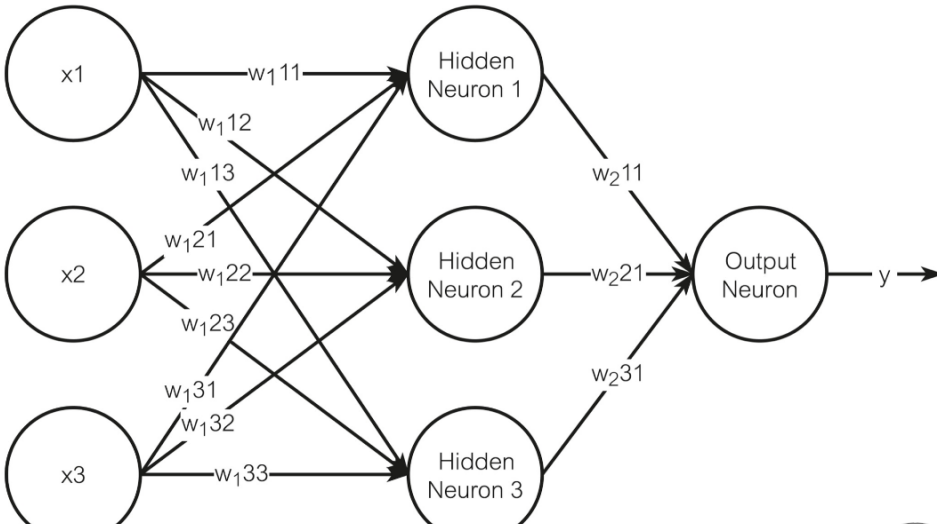

Figure 3.1 Architecture of a typical feedforward neural network

3.2 Activation Functions

The activation functions introduce nonlinearity into neural networks enabling them to model complex patterns that linear models cannot capture [47]. Common activation functions include

Sigmoid function: It is a smooth function (Figure 3.2) bounded between 0 and 1. It is defined as

$$\emptyset(z) = \frac{1}{1+e^{-z}} \tag{3.2}$$

Hyperbolic tangent (tanh): It is also a smooth function (Figure 3.3), bounded between -1 and 1 and defined as

$$\emptyset(z) = \tanh(z) \tag{3.3}$$

Rectified Linear Unit (ReLU): It is a sparse function (Figure 3.4), efficient for deep networks, defined as

$$\emptyset(z) = \max(0, z) \tag{3.4}$$

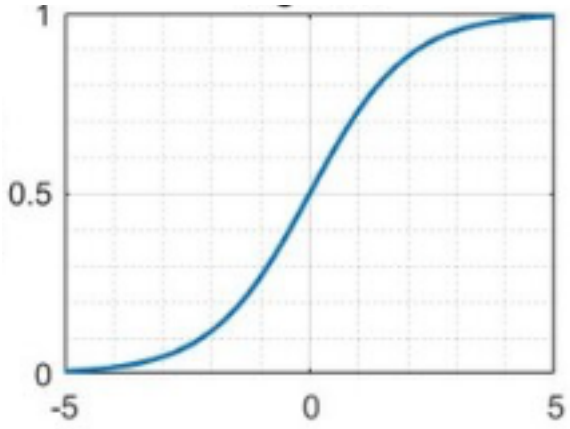

Figure 3.2 Sigmoid activation function

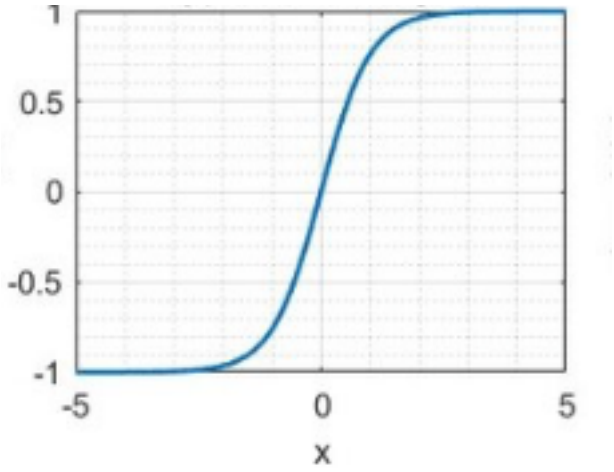

Figure 3.3 Tanh activation function

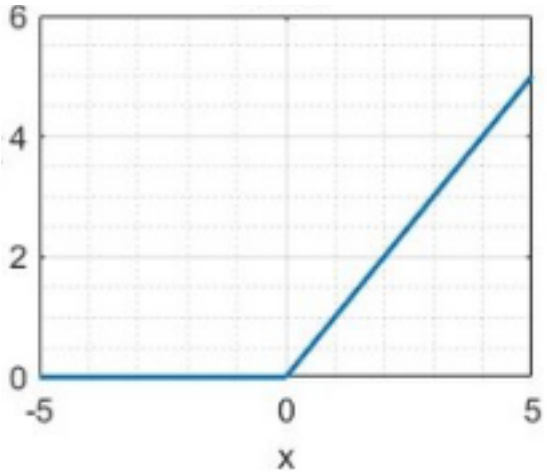

Figure 3.4 Relu activation function

3.3 Training Neural Networks: The Role of Weights and Biases

Training a neural network involves finding optimal values for weights and biases to minimize a loss function, which measures the error between the network's predictions and the true outputs [47]. The training of a neural network involves a series of systematic operations aimed at optimizing the model parameters to minimize the prediction error. As Figure 3.5 shows, the process consists of the following key steps.

The training process begins with a random initialization of the model weights and biases. A non-zero initialization ensures that the network avoids issues such as vanishing or exploding gradients, thus promoting stable convergence. Subsequently a forward propagation is performed. In this phase, the input data is passed through the layers of the network to generate predictions. Each layer applies a linear transformation followed by a nonlinear activation, allowing the network to model complex functions. A loss function measures the deviation between the predicted outputs and the true target values. This scalar loss value reflects the current performance of the model and serves as the objective to be minimized.

Using the backpropagation algorithm, the gradient of the loss function is computed with respect to each trainable parameter in the network. This involves applying the chain rule of calculus across the network's computational graph [47]. The gradients are then used to update the weights via an optimization algorithm such as Stochastic Gradient Descent (SGD) or Adam [48].

The above steps are repeated iteratively over multiple epochs until the model converges to a solution with satisfactory generalization capability (small MSE error for example). Upon completion, the trained weights define the final model, which is used to make predictions on unseen data.

The typical weight update rule (for SGD) is:

$$w_i^{(new)} = w_i^{(old)} - \mathfrak{y}\frac{\partial L}{\partial w_i} \tag{3.5}$$

where $\mathfrak{y}$ is the learning rate and L is the loss function.

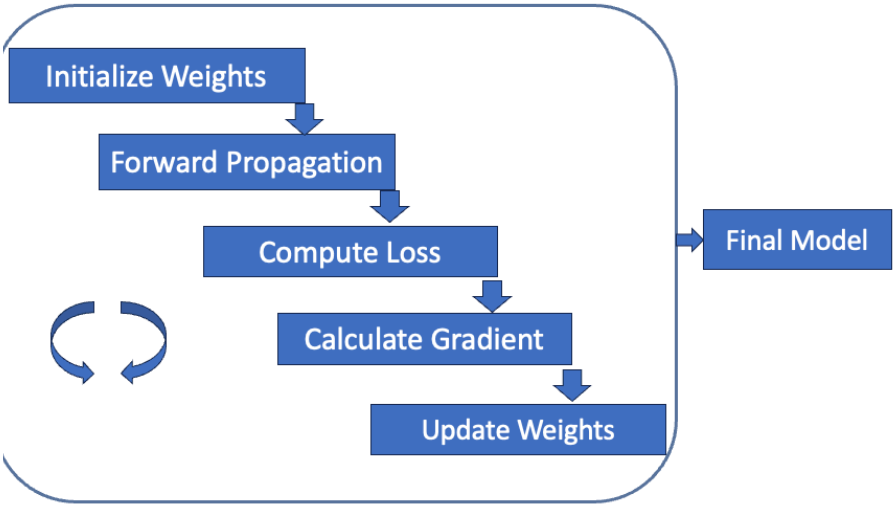

Figure 3.5 Flowchart of neural network training workflow

3.4 Neural Networks as Function Approximators

Neural networks possess the remarkable ability to approximate complex and highly nonlinear functions. According to the Universal Approximation Theorem [47], a feedforward neural network with a single hidden layer containing a finite number of neurons can approximate any continuous function on a compact subset of $\mathbb{R}^n$ to arbitrary accuracy, provided the network has enough neurons and appropriate parameters (weights and biases). Each neuron in the network performs a basic nonlinear transformation of its inputs, and by linearly combining the outputs of many neurons, the network can represent intricate mappings between input and output spaces.

However, the practical ability of a neural network to approximate a given function depends critically on several factors which are the number of neurons, the choice of activation functions and the distribution and optimization of weights and biases. An illustrative example is presented in Figures 3.6-3.9 [51], where a network attempts to approximate the simple quadratic function $y = x^2$ using a varying number of sigmoid neurons.

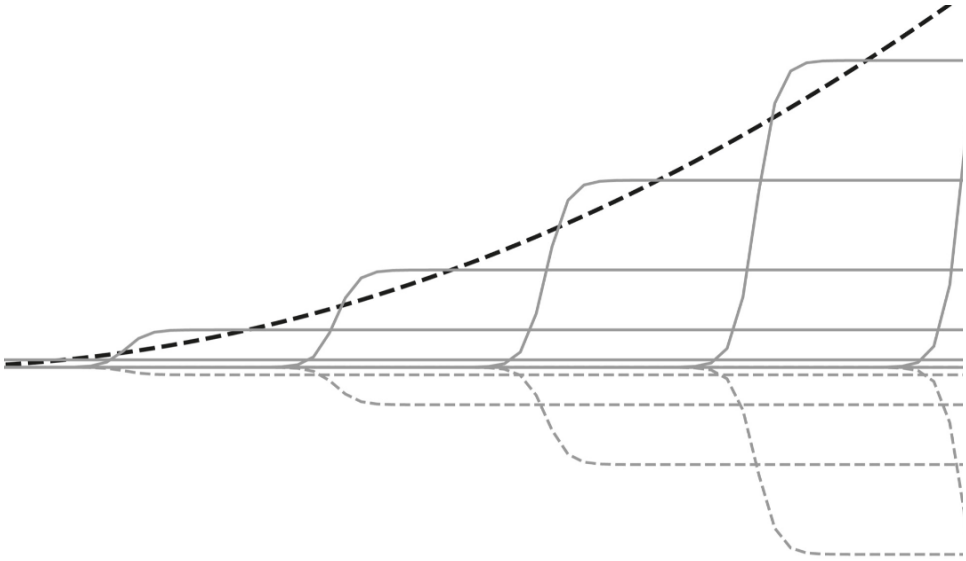

Figure 3.6 Approximation of y=x^2 using 5 sigmoid neurons

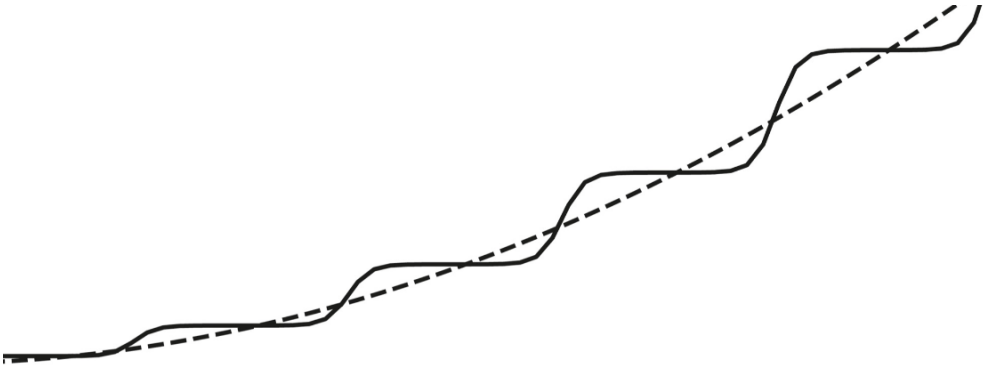

Figure3.7 Approximation of y=x^2using linear combination of 10 sigmoid neurons

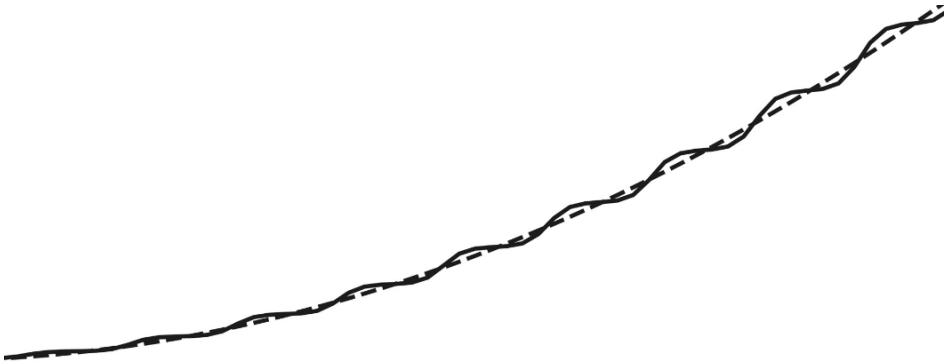

Figure 3.8 Approximation of y=x^2using linear combination of 100 sigmoid neurons

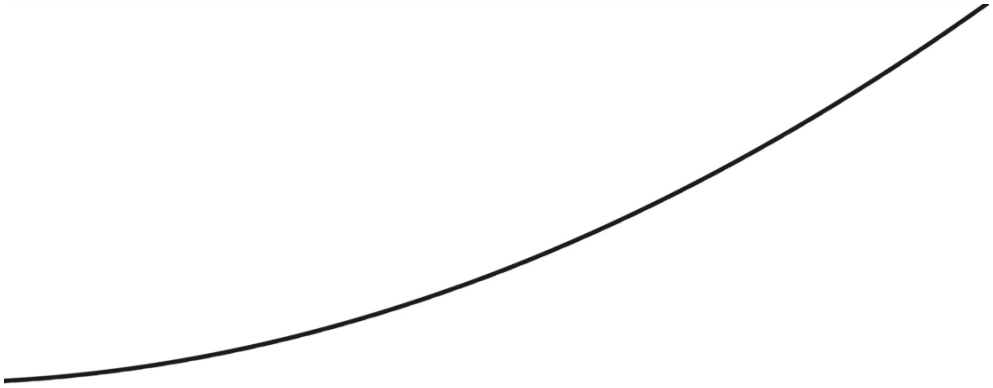

Figure 3.9 Approximation of y=x^2using linear combination of 300 sigmoid neurons

The quality of function approximation thus depends not only on the network architecture but critically on the specific values of the weights and biases learned during training. Different training runs, even on the same dataset, may lead to different learned parameters, resulting in different approximations of the same function. This sensitivity of the final model to weight variability forms a crucial source of model uncertainty, which is explored in depth in the subsequent chapters.

3.5 Variability in Trained Weights and Model Uncertainty

While neural networks theoretically possess the capacity to approximate complex functions, their real-world behavior is significantly influenced by the stochastic nature of the training process. In practice, training a neural network results in variability in the learned weights, even when the same architecture and dataset are used across different training runs. This variability arises from several key factors as explained below.

Random Initialization: At the beginning of training, the weights are initialized randomly [49]. Different initializations place the optimizer at different starting points on the loss surface.

Stochastic Optimization: Training often relies on mini-batch gradient-based methods such as stochastic gradient descent (SGD) or Adam [48], which introduce randomness at every iteration by sampling subsets of the data.

Non-Convex Loss Landscape: The loss surface associated with neural networks is highly non-convex [49]. It contains multiple local minima, saddle points, and flat regions, creating a complex topology that guides the optimization path.

As a result of these factors, different training runs may converge to different local minima, producing distinct final weight configurations. Thus, even with identical inputs, the resulting neural network models may output slightly or even significantly different predictions. This phenomenon is a direct source of model uncertainty — specifically, epistemic uncertainty, which arises from incomplete knowledge of the true best model configuration [50].

As shown in Figure 3.10, two different training trajectories, beginning from different initialization points, will follow different optimization paths and ultimately settle into distinct local minima on the loss surface. Although both models minimize the loss function locally, the resulting learned functions — and hence the predictions — are different.

Unlike aleatory uncertainty, which is irreducible, epistemic uncertainty can potentially be reduced by improving training methods, increasing the amount of data, or employing ensemble modeling techniques.

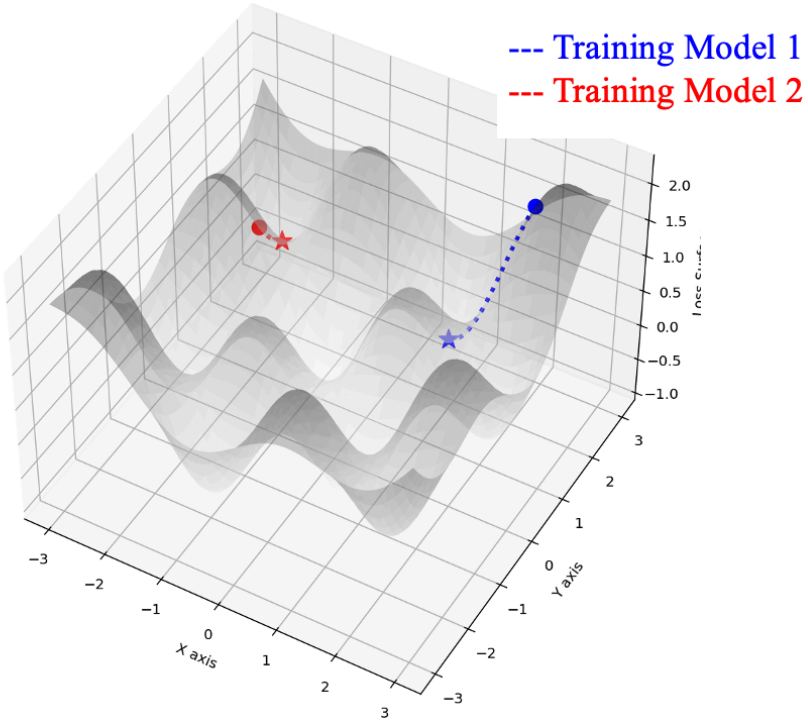

Figure 3.10 A representative 3D visualization of two distant training paths

The inherent variability in the training outcomes emphasizes the importance of developing systematic methods for quantifying and mitigating model uncertainty, which forms the central theme of the subsequent chapters in this dissertation. This phenomenon

also demonstrates that even when identical hyperparameters — such as learning rates, batch sizes, and optimization algorithms — are used, the trained models can still differ because of the different sets of weights learned during training. While it is theoretically possible to avoid random initialization and set the weights deterministically, doing so would make it even more challenging to determine whether the resulting model represents a global or merely a local optimum.

Conversely, employing random initialization across multiple training runs improves the chances of finding a better (or even globally optimal) model. However, it also results in multiple different models. Consequently, there arises a critical need to systematically quantify the uncertainty stemming from variability across these multiple trained models — a need that directly motivates the research developed in this dissertation.

Having established the fundamental structure, behavior, and training processes of neural networks, along with the sources and manifestations of model uncertainty, this chapter has described the key research challenge addressed in this dissertation. The stochastic nature of training, combined with the non-convexity of the loss landscape, results in variability across different trained models, even under identical conditions. This variability leads to epistemic uncertainty in the model outputs — a critical consideration when deploying neural networks in practical applications. The next chapter focuses on reviewing the existing techniques developed for estimating model uncertainty in neural networks, evaluating their capabilities and limitations.

CHAPTER 4

REVIEW OF EXISTING TECHQNIUES FOR UNCERTAINTY QUANTIFICATION

4.1 Monte Carlo Dropout

The mathematical foundation of MC dropout is covered in [12, 13]. Its fundamental concept involves randomly setting the weights of a layer, also known as the "dropout layer," to zero. The frequency of weights being assigned to zero is controlled by the "dropout rate" within the dropout layer. Dropout is activated during both training and prediction. During prediction, multiple forward passes are performed, and uncertainty is estimated by calculating the variance of the predictions. Figure 4.1 shows the MC dropout architecture highlighting the dropout of random weights during each forward pass and therefore, its probabilistic nature of predictions.

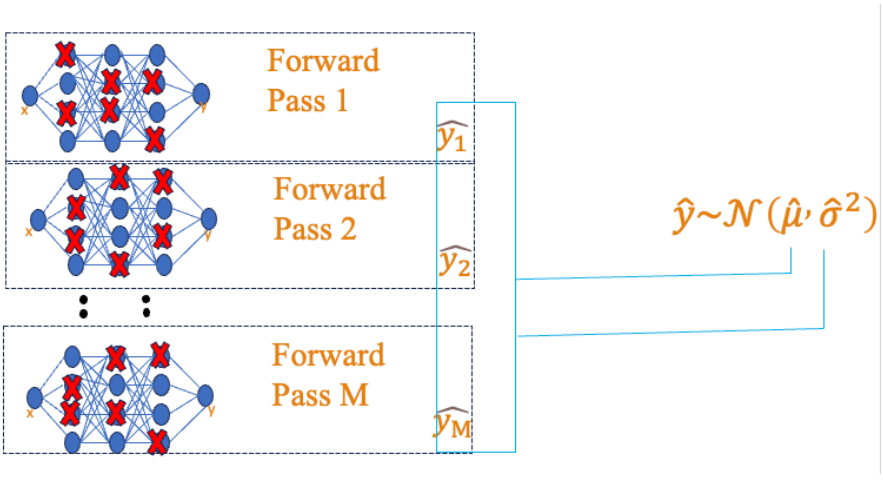

Figure 4.1 MC dropout architecture

The Dropout technique has long been employed in Artificial Neural Networks (ANNs) to mitigate overfitting. During training, network nodes, or neurons, are randomly deactivated, leading to the evaluation and adjustment of different subsets of the network

architecture with each training step. This process serves as a form of regularization, consistently resulting in enhanced generalization across various scenarios.

Monte Carlo Dropout takes this randomization approach a step further. Beyond randomly deactivating network nodes during training, it introduces a similar level of randomness to the prediction process. Consequently, each time an input value is passed to the ANN, a random prediction emerges, characterized by a complex probability distribution. By repeatedly applying the same input to the network, an empirical distribution over the outputs can be deduced, or parameters for a distribution family such as the Gaussian (normal) distribution can be estimated. These empirical distributions or parameter estimates enable the derivation of metrics such as mean values and confidence measures, expressed in terms of the distributional variance. The expectation is that the empirical variance will be low in regions abundant with training data, reflecting the opportunity for all network subsets to learn in these areas. Conversely, in regions devoid of training data, where network behavior is unpredictable, a high variance among different network subsets is anticipated. This aligns with the intended behavior for uncertainty estimation. An algorithmic description of Monte Carlo (MC) Dropout for uncertainty quantification in neural networks is outlined below.

In the training phase, a neural network is trained using standard procedures while retaining Dropout layers during training. The network is optimized using an appropriate probabilistic loss function, with the choice of loss depending on whether the task is a regression or classification problem.

During the testing phase, instead of disabling Dropout—as is typically done in inference—Dropout remains active. Multiple stochastic forward passes are performed for each test input, enabling the generation of a distribution of predictions rather than a single deterministic output.

To obtain the expectation-based prediction and uncertainty estimate, different strategies are applied based on the nature of the task. For regression tasks, the final prediction is computed as the mean of all the stochastic outputs, and the variance across these predictions serves as a measure of uncertainty. In the case of classification, the predicted probabilities for each class are collected across all forward passes. These are then averaged to obtain a mean class probability distribution, which represents the model's expected classification. The dispersion of the class probabilities—often measured using entropy or mutual information—can be used to estimate classification uncertainty. In both settings, the uncertainty estimate is derived from the variability among the predictions. A higher variance indicates lower confidence, thereby quantifying epistemic uncertainty due to model parameter variability captured through stochastic Dropout sampling.

This algorithm leverages the dropout layers during training to introduce stochasticity, and during testing, it utilizes multiple stochastic forward passes to derive both expectation-based predictions and uncertainty estimates tailored to the specific nature of the regression or classification task. One of the primary benefits of MC dropout is its simplicity in implementation. It only necessitates a few alterations into an existing DNN structure. This is often readily available as a dropout layer in numerous coding environments. Moreover, its implementation simplicity is independent of the neural

network architecture, making it easily adaptable for various popular neural network types such as Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) [32, 33].

Another significant benefit of MC dropout is its low computational expense and high scalability. This is because its training process is essentially the same as the standard, non-Bayesian training of DNNs, but with randomized sparse networks. This makes it a cost-effective and scalable for various applications.

4.2 Network Ensemble

In the Network Ensemble approach, multiple neural networks are trained to predict the same response. These networks are trained with the same or different hyperparameters resulting in multiple ML models [14, 17]. The uncertainty is estimated, by calculating the variance of these predictions. Figure 4.2 describes the network ensemble schematically.

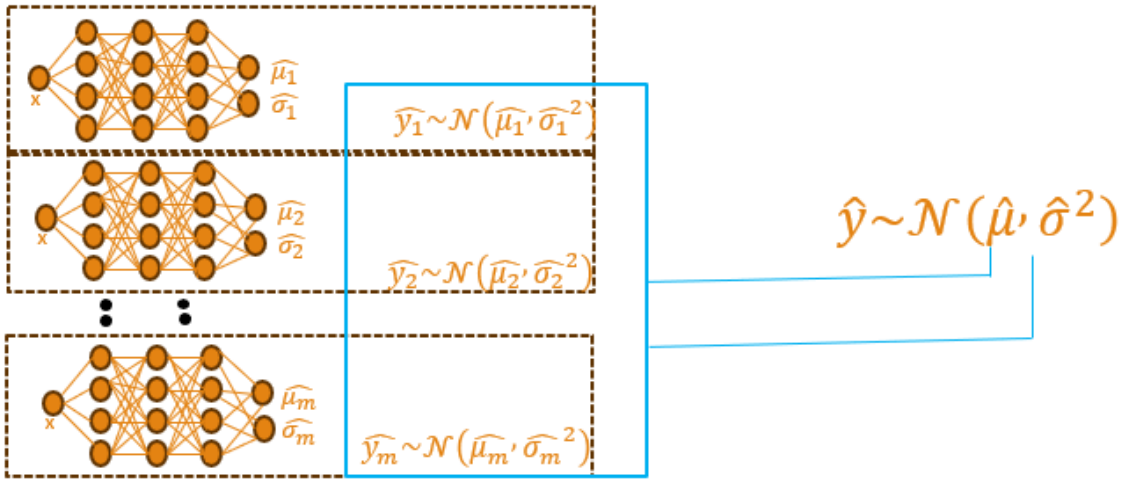

Figure 4.2 Network ensemble architecture

The network ensemble is a well-established technique to address overfitting. Rather than training a single network, the approach involves training an ensemble of M networks [14]. While we anticipate consistent behavior among all networks in regions abundant with training data, outcomes will diverge significantly in data-scarce areas. To derive the ultimate prediction, we aggregate the results of all networks into a Gaussian mixture distribution. Subsequently, we can extract singular mean and variance estimations from this distribution.

In deep learning, ensembles of neural networks are created by independently training multiple networks with the same architecture. This approach is easy to implement and widely used to quantify prediction uncertainty [18, 34]. These ensembles tend to produce higher uncertainty estimates for Out-Of-Distribution (OOD) data compared to similar training data, especially when using well-calibrated uncertainty estimates. This property is exploited to detect dataset shifts, as data from shifted environments often differ significantly from training data [20,22,35]. Network ensemble technique can be implemented different ways to capture aleatory and epistemic uncertainty.

Let's examine the process of quantifying uncertainty. This is done by individually training each network within an ensemble (Figure 4.2). The ensemble produces two outputs: the mean and variance of the output, which are generated in the final output layer. In this setup, it is common to assume that each network's predictive distribution follows a Gaussian distribution. Hence, the final output layer is occasionally referred to

as the Gaussian layer. This allows us to characterize the aleatory observational noise associated with the target values.

The main concept is to customize aleatory uncertainty for each input, making it dependent on the input itself. The noise in the model can be described as

$$y_i = f(\boldsymbol{x}_i) + \sigma_i^2(\boldsymbol{x}_i) \tag{4.1}$$

It has been found that a neural network can be trained to learn the mapping from $\boldsymbol{x}_i$ to $\sigma_i^2(\boldsymbol{x}_i)$. Consequently, a neural network with parameters $\boldsymbol{\theta}$ can be trained to predict both the mean $\mu_i = \hat{\mu}(\boldsymbol{x}_i; \boldsymbol{\theta})$, and variance $\sigma_i^2 = \hat{\sigma}^2(\boldsymbol{x}_i; \boldsymbol{\theta})$ of the target for each input $\boldsymbol{x}_i$. This neural network has two outputs, the predicted mean μ_i and variance σ_i^2, which fully characterize a Gaussian predictive distribution. Optimization of parameters $\boldsymbol{\theta}$ can be achieved by minimizing the following likelihood function

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} \left[\frac{log\, \hat{\sigma}^2(\boldsymbol{x}_i;\boldsymbol{\theta})}{2} + \frac{(y_i - \hat{\mu}(\boldsymbol{x}_i;\boldsymbol{\theta}))^2}{2\hat{\sigma}^2(\boldsymbol{x}_i;\boldsymbol{\theta})} \right] \tag{4.2}$$

Details can be found in [3].

Now let us review the estimation of epistemic uncertainty using an ensemble of independently trained networks. The method described to quantify aleatory uncertainty can pose a problem, as Equation (8) yields deterministic values for hyperparameters $\boldsymbol{\theta}$ and cannot capture the uncertainty associated with the hyperparameters themselves. The issue becomes apparent when dealing with real-world data, as the available data is often limited, leading to significant epistemic uncertainty. To capture epistemic uncertainty, multiple models (say M in number) are trained using the same network architecture, resulting in a different set of parameters $\boldsymbol{\theta}$ for each iteration. If each individual neural

network in the ensemble carries an equal weight, the mean and variance of the ensemble predicted Gaussian distribution are

$$\hat{\mu}_c(\boldsymbol{x}) = \frac{1}{M}\sum_{i=1}^{M}\hat{\mu}_i(\boldsymbol{x}) \tag{4.3}$$

$$\hat{\sigma}^2{}_c(\boldsymbol{x}) = \frac{1}{M}\sum_{i=1}^{M}\hat{\sigma}_i{}^2(\boldsymbol{x}) + \left[\frac{1}{M}\sum_{i=1}^{M}\hat{\mu}^2{}_i(\boldsymbol{x}) - \hat{\mu}^2{}_c(\boldsymbol{x})\right] \tag{4.4}$$

In an ensemble of neural networks, both aleatory and epistemic uncertainty can be measured easily. The aleatory uncertainty, which arises from observation noise, is reflected in the variance predicted by each individual network. On the other hand, the epistemic uncertainty, associated with network structure and parameters, is mainly manifested as the difference in predictions among the individual networks. This difference characterizes the structural and parametric uncertainty of the neural network. Training multiple neural networks with the same architecture independently captures epistemic uncertainty due to the complexity of optimizing a large number of parameters in a high-dimensional space [36]. Random initialization of parameters and perturbations in the training data lead to different locally optimal parameter values for each network.

4.3 A Mathematical Example for MC Dropout and Ensemble

To illustrate the application of MC dropout and Network Ensemble techniques, we use a specific example represented by Equation (4.5). The function's 3-D representation is depicted in Figure 4.3 A design of experiment consisting of 100 points is created using the maxi-mean approach, which aims to maximize the minimum distance between two designs. This design of experiments can be visualized in Figure 4.4.

$$g = 3 \times {x_1}^2 + sin\, 2\pi \times x_2 \; and \; x_1,\, x_2 \in [0,1] \tag{4.5}$$

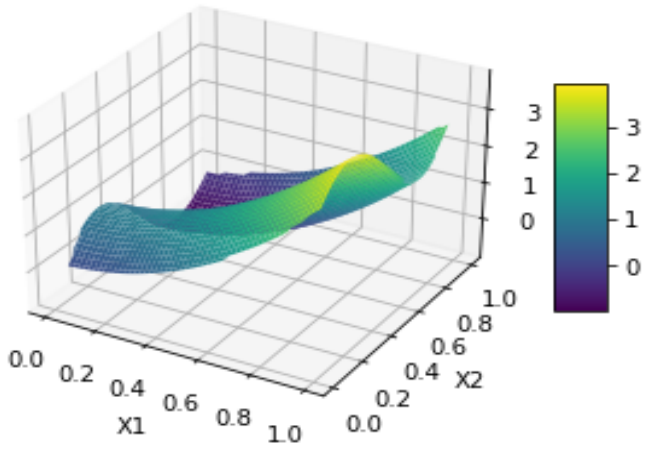

Figure 4.3 Graphical representation of mathematical example

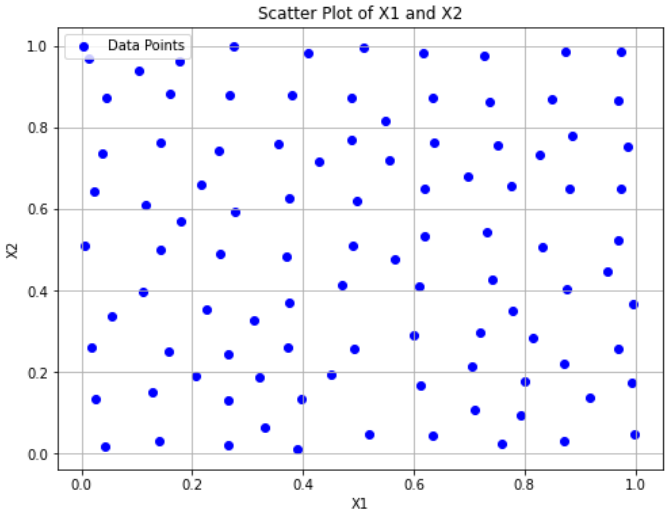

Figure 4.4 DoE of 100 points using maxi-min approach

In this instance, a neural network is constructed with a total of 7 layers and 50 neurons, utilizing the Rectified Linear Unit (ReLU) as the chosen activation function. To incorporate Monte Carlo (MC) dropout during the training of a neural model, a distinct custom layer named “dropout” is formulated (Figure 4.5). This custom layer is essential to implement the dropout technique not only during the training phase but also throughout the testing phase. A drop out probability of 0.2 is used for this example.

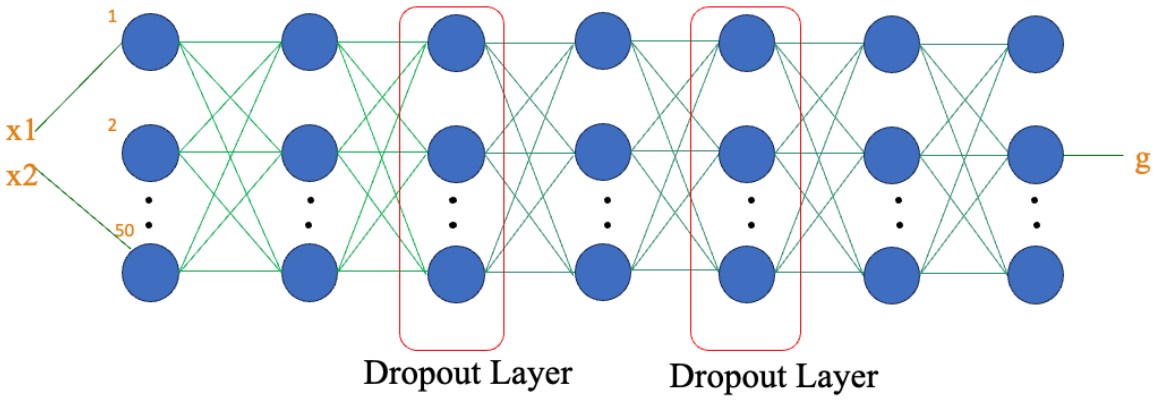

Figure 4.5 MC Dropout neural network representation

For the purpose of uncertainty quantification, an unobserved data point, which is not part of the original DoE, is utilized with a value of $(x_1, x_2) = (0.5, 0.5)$. We assume that the uncertainty is characterized by the mean value plus or minus one standard deviation. Figure 4.6 shows the uncertainty band evaluated using the MC dropout method highlighting the presence of model uncertainty.

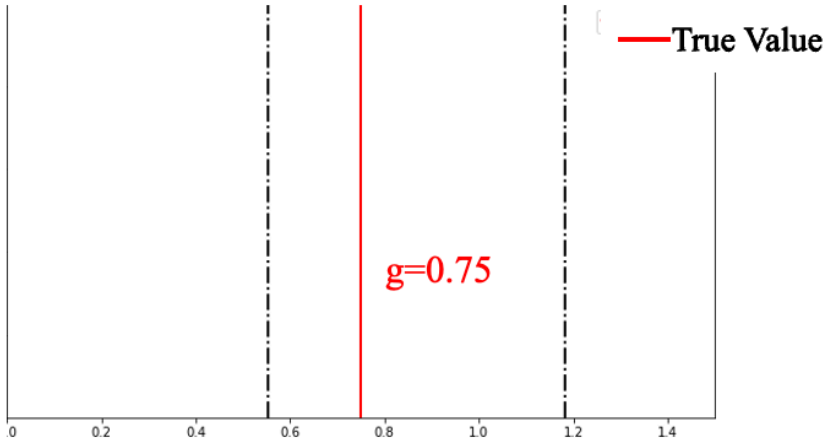

Figure 4.6 UQ with MC dropout using a dropout rate of 0.2

To showcase the UQ using the Network Ensemble approach, the architecture of Figure 4.7 and Table 4.1 is adopted. To predict uncertainty, an ensemble of 500 models were trained using the same architecture. The UQ is captured by calculating the 5th and 95th percentiles of the prediction point (x1, x2) = (0.5, 0.5). Figure 4.8 shows the histogram of all predictions and the two percentiles. The ensemble method is easy to implement, but it can be computationally expensive. Training multiple models, and obtaining a prediction from each model, takes time which depends on the number of models and the complexity of the responses. Also, the number of models in the ensemble can have a significant impact on the accuracy of uncertainty estimation. The uncertainty estimation improves if we increase the number of models in the ensemble.

Table 4.1 Network ensemble architecture

Number of layers	6
Number of neurons per layer	50
Activation function	Rectified Linear Unit (ReLu)

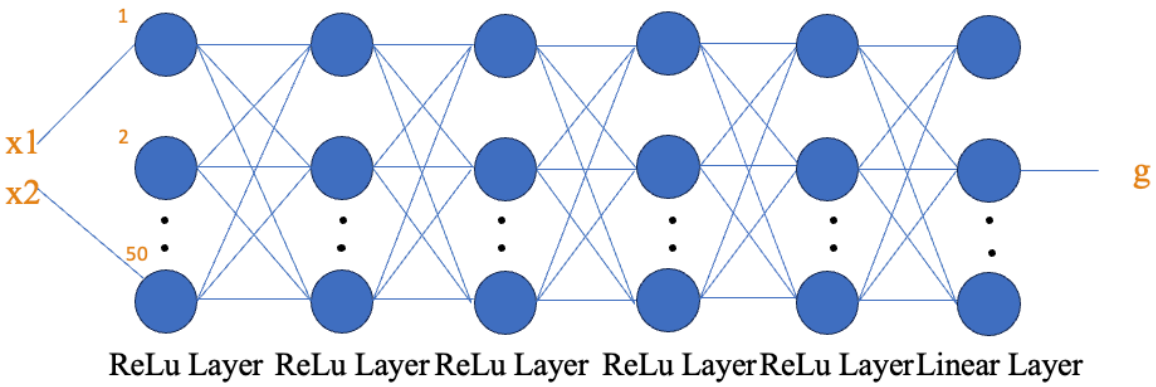

Figure 4.7 Network ensemble representation

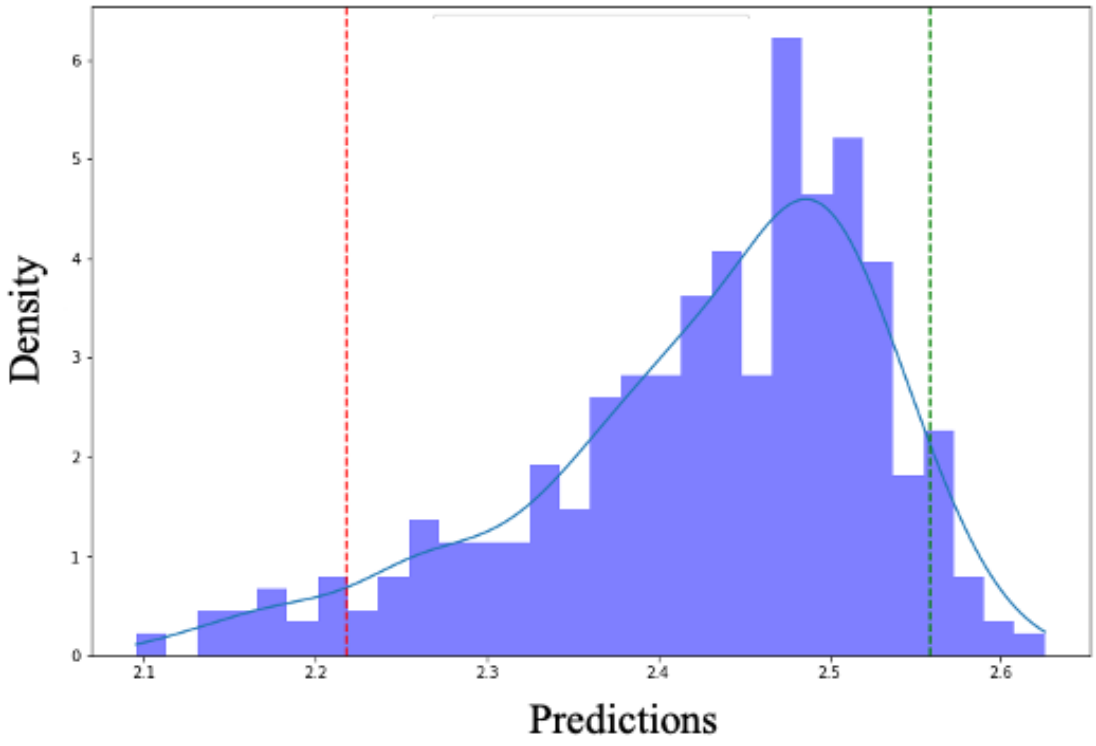

Figure 4.8 UQ using a network ensemble

4.4 Limitations of MC Dropout Approach

Although Monte Carlo Dropout is valued for its simplicity and scalability, it exhibits several limitations when utilized for model uncertainty quantification.

a) Sensitivity to Dropout Probability: A significant limitation of the MC dropout method is its strong reliance on architecture parameters, such as dropout layers and dropout probability [30-32]. This can be observed in Figures 4.9 and 4.10, where uncertainty quantification with dropout probabilities of 0.4 and 0.6 respectively, differs significantly from the uncertainty quantification achieved with a dropout probability of 0.2 (Figure 4.6).

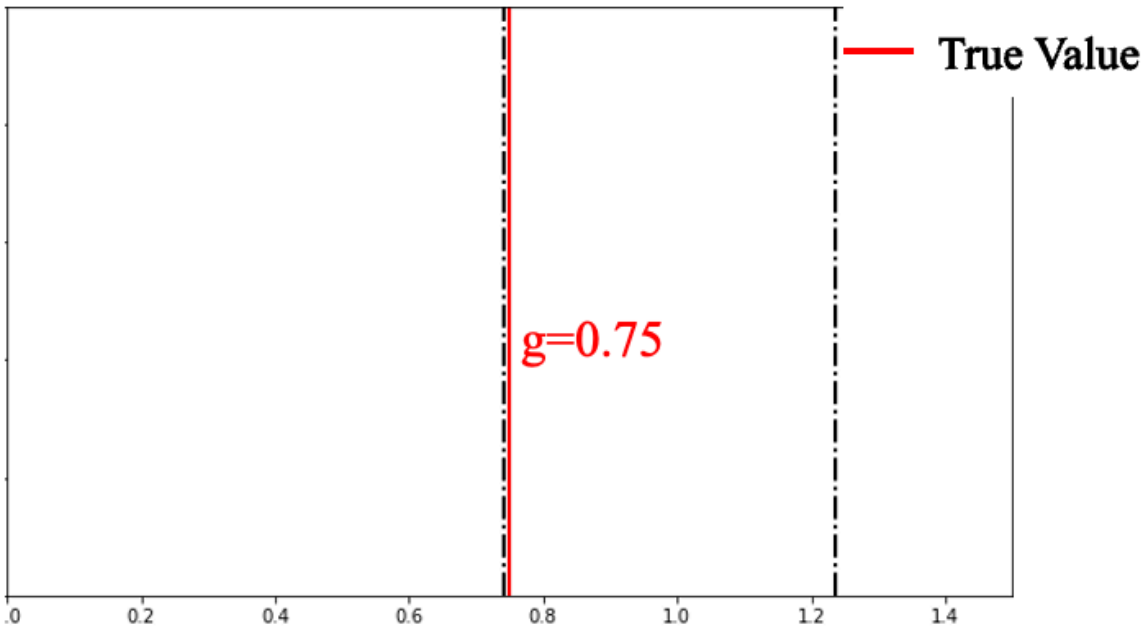

Figure 4.9 UQ with MC dropout using a dropout rate of 0.4

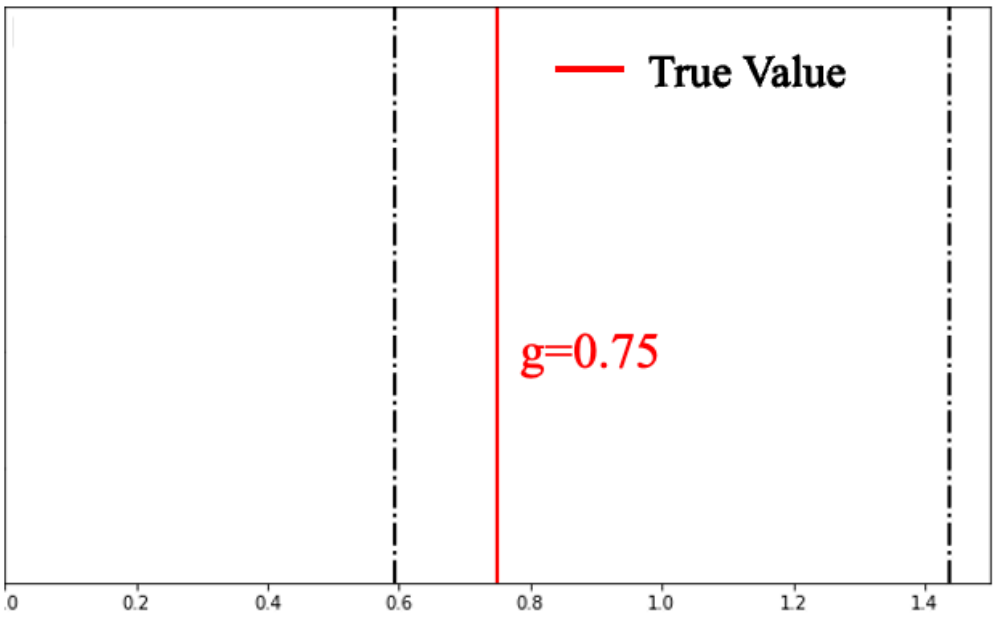

Figure 4.10 UQ with MC dropout using a dropout rate of 0.6

b) Approximate Bayesian Inference**:** MC Dropout only approximates Bayesian inference rather than providing a true posterior distribution. As a result, uncertainty estimates may be biased, especially for complex models or highly non-Gaussian data.

c) Multiple Forward Passes Required: Accurate uncertainty estimation necessitates multiple stochastic forward passes during inference, increasing computational time and energy usage compared to standard deterministic predictions.

d) Inference-Time Dropout Requirement: Dropout must be explicitly enabled during inference, deviating from standard machine learning practice and requiring careful control of implementation.

e) Underestimation of Uncertainty: MC Dropout often underestimates model uncertainty, particularly for out-of-distribution (OOD) inputs, leading to overconfident predictions in regions lacking training data support.

f) Hyperparameter Sensitivity: The overall uncertainty quality depends on multiple hyperparameters, including dropout probability, network architecture, and number of forward passes, often requiring costly empirical tuning.

g) Challenges in Very Deep Models: In very deep architectures, the application of dropout across many layers may lead to degraded model performance and unstable uncertainty estimates.

4.5 Limitations of Network Ensemble Approach

Despite the effectiveness of neural network ensembles in capturing model uncertainty, they also present several practical limitations.

a) Slow Inference Speed: The results are not instantaneous, as predictions must be obtained from multiple models and aggregated. This can negatively impact the user experience, particularly in professional software tools where real-time or near real-time predictions are expected.

b) High Computational Resource Requirements: Inference with an ensemble requires substantial computational resources, including CPU and GPU utilization. Such demands may render ensemble approaches infeasible for deployment on resource-constrained platforms such as edge devices or mobile applications.

c) Large Memory Footprint: All models within the ensemble must be loaded into memory simultaneously during inference. This significantly increases the memory requirements, potentially leading to memory overflow issues, swapping delays, or reduced overall system performance.

4.6 Gaussian Process Regression (GPR)

This section reviews the Gaussian Process Regression (GPR) that has been well documented over the years [3] in effectively quantifying uncertainty. A Gaussian process is a set of random variables defined over a domain, where any finite subset of these variables follows a multivariate Gaussian distribution. In simpler terms, it represents a probability distribution for an unknown function, which consists of infinitely many random variables. To illustrate this concept, let's consider an unknown function ***f(x)***, where $\boldsymbol{x} \in \mathbb{R}^D$. For any finite set of input points $\boldsymbol{x}$, such as $\boldsymbol{X} = [x_1, .., x_N]^T \in \mathbb{R}^{N \times D}$ the corresponding outputs $f(X) = [f(x_1), \dots, f(x_N)]^T \in \mathbb{R}^N$ follow a joint Gaussian distribution.

GPR starts from a prior Gaussian process for the unknown function $f(\boldsymbol{x}) \sim \mathcal{GP}\ (m(\boldsymbol{x}), k(\boldsymbol{x}_i, \boldsymbol{x}_j))$ where ***m(x)*** is the prior mean. The prior mean is often considered zero. If it is a non-zero, it is subtracted from the observations to maintain a zero-mean status. $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is the prior covariance function or kernel which tells us how $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are correlated. It typically takes the following form

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \mathbb{E}[(f(\boldsymbol{x}_i) - m(\boldsymbol{x}_i))(f(\boldsymbol{x}_j) - m(\boldsymbol{x}_j))] \tag{4.6}$$

Since ***f(x)*** is unknown we will approximate it with the following widely used function known as Radial Basis Function (RBF)

$$k(\boldsymbol{x}_i\ , \boldsymbol{x}_j) = \sigma^2 exp\ (-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}{2l^2}) \tag{4.7}$$

The GPR model has two kernel parameters: the signal amplitude σ and the length scale l. The signal variance sets the upper limit of the prior variance and covariance. A large

value of signal amplitude occurs when the function spans a wide vertical range. The covariance between function values decreases as the input points get farther apart. Observations far away in the input space have minimal influence, while out-of-domain inputs have low covariance and high uncertainty. When input points are close, there is a strong correlation and smoothness in the model.

A sample covariance matrix assembled for a given set of inputs using the chosen kernel is

$$\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) = \begin{bmatrix} k(\boldsymbol{x}_1,\boldsymbol{x}_1) & \cdots & k(\boldsymbol{x}_1,\boldsymbol{x}_n) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{x}_n,\boldsymbol{x}_1) & \cdots & k(\boldsymbol{x}_n,\boldsymbol{x}_n) \end{bmatrix} \tag{4.8}$$

For practical purposes, white noise is added to the model such that

$$y = f(\boldsymbol{x}_i) + \sigma_n^2 \tag{4.9}$$

where y is the true prediction and σ_n^2 is the variance of the added white noise. For N_* new test points $\boldsymbol{X}_*$, the joint Gaussian distribution of the observed outputs $\boldsymbol{y}$ and predicted outputs $\mathbf{f}_*$ is

$$\begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{f}_* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} \boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) + \boldsymbol{\sigma_n^2 I} & \boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}_*) \\ \boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X}) & \boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X}_*) \end{bmatrix}\right) \tag{4.10}$$

To make a prediction of $\boldsymbol{f}_*$ given the observed data $\boldsymbol{y}$ the posterior distribution of the test outputs $\boldsymbol{f}_*$ is also Gaussian, with mean $\boldsymbol{\mu}_*$ and covariance *cov($\boldsymbol{f}_*$)* such that

$$\boldsymbol{\mu}_* = \boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X})[\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) + \boldsymbol{\sigma_n^2 I}]^{-1}\,\boldsymbol{y} \tag{4.11}$$

$$cov(\boldsymbol{f}_*) = \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X}_*) - \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X})[\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \boldsymbol{\sigma_n^2 I}]^{-1}\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}_*) \quad (4.12)$$

The Gaussian Process Regression (GPR) is a modeling technique that can capture both aleatory (inherent) and epistemic (knowledge-based) uncertainties. In the context of regression problems, the posterior variance for a test point is a concise representation of the overall predictive uncertainty. The variance of the additive white noise represents the aleatory uncertainty, which is assumed to be constant and learned from the observations. This is referred to as a "homoscedastic" GPR model.

On the other hand, a "heteroscedastic" GPR model considers the noise variance as a function of the input variables, allowing for varying levels of uncertainty. The epistemic uncertainty, mainly determined by the covariance between the training points and the query points, is influenced by the distance between them. The farther a query point is from the training points, the smaller the covariance elements and the larger the epistemic uncertainty.

This distance-awareness property of GPR makes it suitable for reliable uncertainty quantification (UQ) and out-of-distribution (OOD) detection, particularly for problems with low dimensions and training sizes. The aleatory and epistemic uncertainty components of the posterior variance together determine the width of the confidence interval for model predictions at the query point, reflecting the overall predictive uncertainty.

4.7 Limitations of GPR

a) Scalability Issues: The standard Gaussian Process Regression (GPR) method tends to have scalability issues for large training datasets (large N). This is because the training complexity of GPR is primarily due to the computation of the inverse and the determinant of the N x N covariance matrix $K(X, X)$ during model training and hyperparameter optimization.

b) High Memory Requirements: Storing and manipulating the covariance matrix scales quadratically with the dataset size, leading to significant memory usage, especially for high-dimensional problems.

c) Choice of Kernel: The performance and reliability of uncertainty estimates are highly sensitive to the choice of the kernel function and its hyperparameters. Poor kernel selection can result in overconfident or underconfident uncertainty estimates.

d) Hyperparameter Tuning Challenges: Training a GPR model involves optimizing kernel hyperparameters, which often requires non-convex optimization techniques. This can lead to convergence to suboptimal solutions and poor model calibration if not handled carefully.

e) Overconfidence in Extrapolation: Outside the range of the training data, GPR models tend to produce uncertainty estimates that can be either too small (overconfident) or unrealistically large, depending on the kernel structure. In particular, extrapolation outside observed domains remains unreliable.

f) Difficulty with High-Dimensional Inputs: GPR models generally perform poorly as the dimensionality of the input space increases ("curse of dimensionality"). The kernel similarity between points becomes less meaningful in high-dimensional spaces, degrading both predictions and uncertainty estimates.

g) Approximate Methods Compromise Accuracy: To scale GPR to larger datasets, sparse approximations or inducing point methods are commonly used. However, these approximations can compromise the quality of both mean predictions and uncertainty quantification if not carefully managed.

4.8 Other UQ Approaches

In addition to MC dropout, network ensemble and GPR there are other techniques as well to estimate model uncertainty in neural networks. They include the following methods.

Dropout Ensembles

The fusion of MC dropout and deep ensembles is termed dropout ensembles. Analogous to the approach in deep ensembles, the focus is on estimating distributional parameters rather than actual output values [38]. However, diverging from the practice of training multiple networks, a singular network with dropout is trained. During predictions, dropout remains active, generating random parameter outputs. These outputs provide ensemble outputs, facilitating the derivation of a singular estimation encompassing both mean and variance.

CHAPTER 5

PROPOSED ADAPTIVE GAUSSIAN PROCESS REGRESSION METHOD USING A NN ENSEMBLE

As discussed earlier, there are various methods available for estimating model uncertainty in neural networks. MC dropout is relatively easy to implement but has limitations in terms of its reliance on hyperparameters, particularly the dropout rate. The network ensemble can provide robust predictions but can become inefficient when dealing with a large number of networks in the ensemble, especially if they exhibit complex responses. Gaussian Process Regression (GPR) is an efficient technique, but as mentioned earlier, it faces scalability concerns. Therefore, there is a need to develop a method that is both accurate and efficient, capable of scaling up without encountering the above issues.

We propose a novel approach that extends the NN ensemble method by incorporating Gaussian Process Regression (GPR), aiming to enhance both accuracy and efficiency. Because of the GPR in the proposed approach, its NN ensemble does not require a very large number of models reducing therefore, the computational demand of the proposed method. Figure 5.1 shows schematically the four steps of the proposed approach, named Adaptive GPR – NN Ensemble, which combines the strengths of both GPR and NN Ensemble methods to achieve improved performance. The four steps are: Creation of multiple ML models (Section 5.1), creation of percentile data (Section 5.2), adaptive GPR (Section 5.3), and prediction phase (Section 5.4).

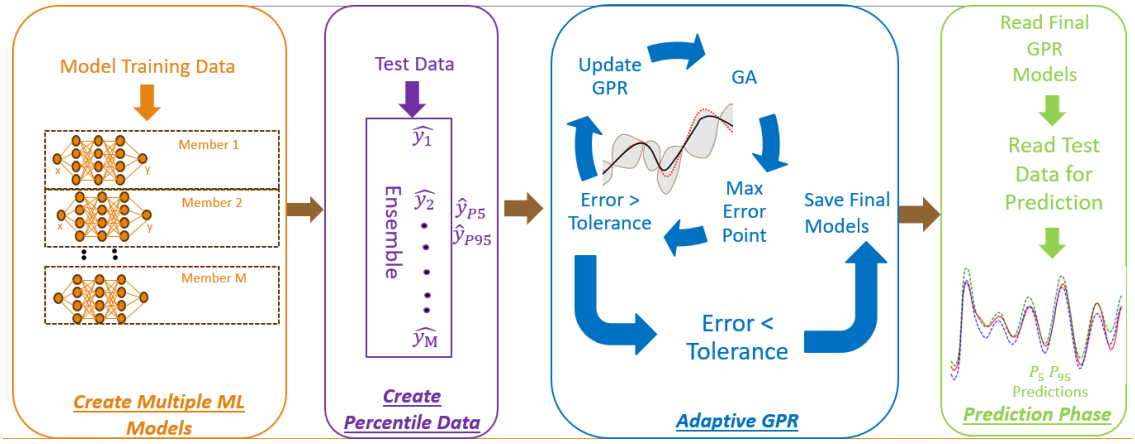

Figure 5.1 Schematic of proposed Adaptive GPR – NN Ensemble approach

5.1 Creation of Multiple ML Models

The initial step of the proposed process involves training multiple ML models to predict the output as a function of the input. The models can be trained with the same or different hyperparameters to generate a Neural Network ensemble. Suppose we have m available data such as the maxi-min DoE of Figure 4.4. The m data is partitioned into model training data m_{train} (e.g. 80% of m) and testing data m_{test} (e.g. 20% of m). The latter are used in Section 5.2. Although not necessary, the points in m_{test} are preferably spread in the entire design space.

Using the same training data m_{train}, a large number k_{ML} of ML models is trained. During training, m_{train} is partitioned into training and validation data. Note that the output of each ML model for the same input, will be different due to the inherent uncertainty of the training process (e.g. optimized values of weights and biases). To reduce the effect of prediction uncertainty of each of the k_{ML} models, the number m of available data is selected so that the R^2 statistic between prediction and exact value for the validation set

of each of the k_{ML} models, is at least 0.95. Once trained, the k_{ML} ML models are saved for future use during the process.

5.2 Creation of Percentile Data

In this step, the test data m_{test} is utilized to generate an ensemble of output predictions using the previously saved k_{ML} models. Each point in m_{test} is used as input in each of the k_{ML} models to predict the output. All k_{ML} output values are then used to calculate the 5th and 95th percentiles for this point in m_{test}. This allows to assess the range of predictions and provides a measure of model uncertainty.

5.3 Adaptive Gaussian Process Regression (GPR)

In Section 5.2, the 5th and 95th percentiles are calculated for each test data point using the generated ensemble of k_{ML} ML models (Section 5.1). We then create a GPR model for the 5th percentile and a separate GPR model for the 95th percentile. Each GPR model provides a mean prediction and the variance of prediction for the percentiles which can be used to calculate an upper bound and a lower bound based on an assumed confidence level as in Equations (5.1) and (5.2) for a 95% confidence level.

$$u_5 = |(\mu_{5*} + 1.96 \times \sigma^2{}_{5*}) - (\mu_{5*} - 1.96 \times \sigma^2{}_{5*})| \tag{5.1}$$

$$u_{95} = |(\mu_{95*} + 1.96 \times \sigma^2{}_{95*}) - (\mu_{95*} - 1.96 \times \sigma^2{}_{95*})| \tag{5.2}$$

The GPR models are developed in Python [29].

Subsequently, a Genetic Algorithm (GA) is used to identify the point in the design space with the maximum value of $u_5^2 + u_{95}^2$. If the maximum value exceeds a predefined tolerance level, the identified by the GA design point is added to the previous design points (m_{test} for the 1st GA run) and the GPR models for the 5th and 95th percentiles are

updated before the GA is run again to identify the design point with the maximum value of $u_5^2 + u_{95}^2$. This process is repeated until the predefined tolerance level is achieved.

5.4 Prediction Phase

The converged GPR models from Section 5.3 are used to predict the 5th and 95th percentiles of a new design point providing therefore, a measure of uncertainty. It is noted that the proposed method considers model-form uncertainty and parameter (e.g. NN weights and biases) uncertainty.

The proposed Adaptive GPR – NN Ensemble approach (Sections 5.1 through 5.4) is summarized below.

a. Develop initial GPR models to predict 5th and 95th percentiles using m_{test} (Section 5.1)
b. Set tolerance for ($u_5^2 + u_{95}^2$)
c. Use a Genetic Algorithm to identify the point with the maximum ($u_5^2 + u_{95}^2$)
d. If max ($u_5^2 + u_{95}^2$) > tolerance, add design point to the training data and retrain GPR
e. Adaptive loop: Repeat step c and d until max ($u_5^2 + u_{95}^2$) < tolerance
f. Save the converged GPR models for prediction (Section 5.4)

The converged GPR models from Step f are used to predict the 5th and 95th percentiles of a new design point providing therefore, a measure of uncertainty. It is noted that the proposed method considers model-form uncertainty and parameter (e.g. NN weights and biases) uncertainty.

5.5 A Mathematical Example for Validation

The proposed method of Sections 5.1 through 5.4 was applied to the mathematical example of Equation (11). For that, 100 design points (m) were generated as seen from Figure 5. Out of 100 design points 80 points are used for model training (m_{train}) and 20 points are used as testing data (m_{test}) and 100 neural network models (k_{ML}) are generated with model architecture shown in Table 4.1. Figure 5.2 illustrates the convergence of the GA algorithm. It took 21 GA iterations to achieve the desired accuracy of the two GPR models. Figure 5.3 shows the histograms of the x_1 and x_2 coordinates of the points added to m_{test} for the GPR models to converge. It is apparent that quite a few points were added at the boundary of the design space suggesting that the uncertainty is more pronounced at the boundary.

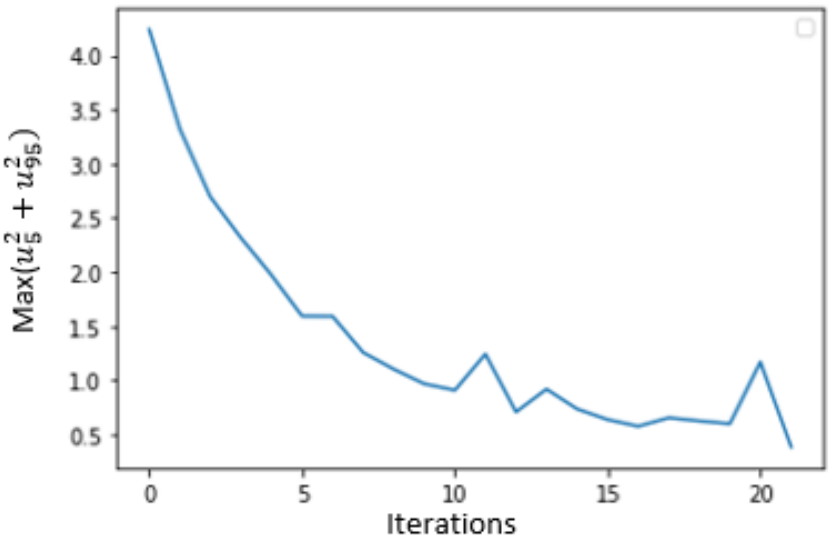

Figure 5.2 Maximum value of $u_5^2 + u_{95}^2$ from GA for mathematical example

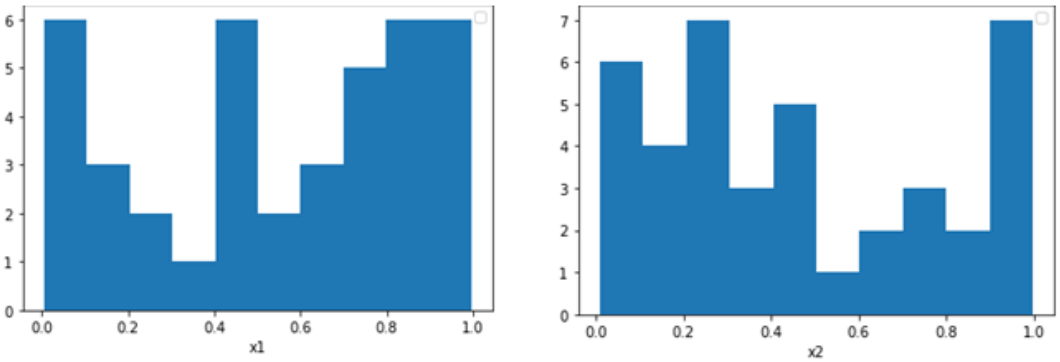

Figure 5.3 Histograms of x_1 and x_2 coordinates of points added to m_{test}

Figure 15 shows the uncertainty estimation using the proposed Adaptive GPR – NN Ensemble approach. The predicted 5th and 95th percentiles are within a 3% error margin compared to the actual 5th and 95th percentiles, indicating that the predictions are accurate. Also, as expected, the true value falls within the range defined by the 5th and 95th percentiles.

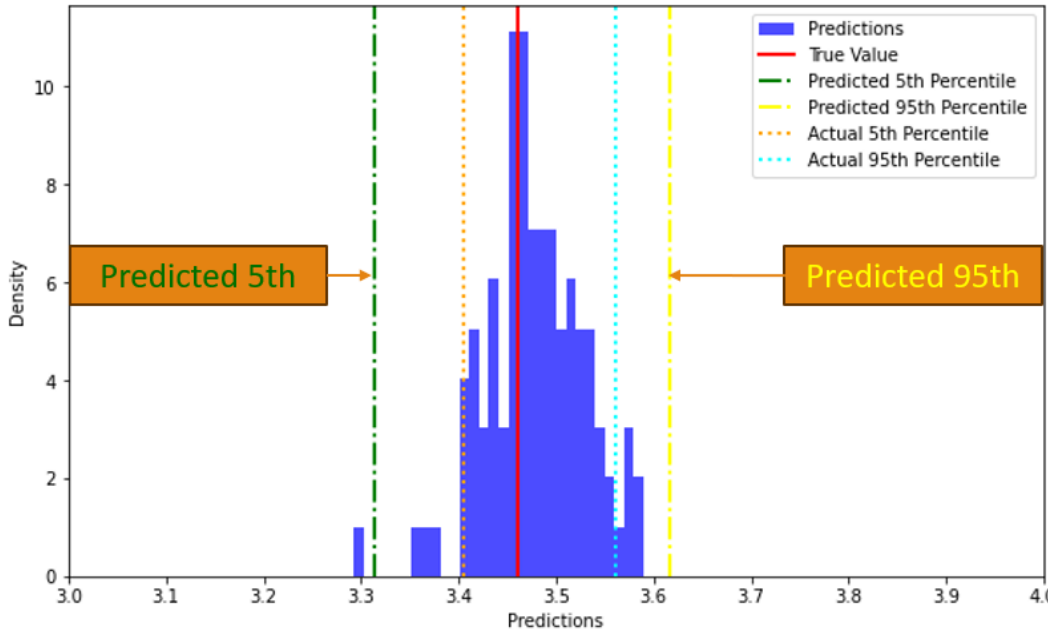

Figure 5.4 Uncertainty quantification from the proposed method

CHAPTER 6

PROPOSED ADAPTIVE GPR – NN ENSEMBLE APPROACH: A VEHICLE DYNAMICS EXAMPLE

This section demonstrates the effectiveness of the proposed method to quantify uncertainty using the vehicle dynamics example of Figure 6.1.

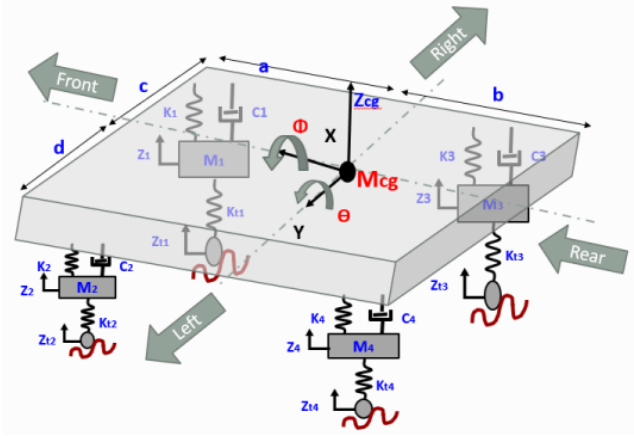

Figure 6.1 Full-vehicle analytical model

Figure 6.2 describes all modeling quantities. It is a full-vehicle model with 11 Degrees-Of-Freedom (DOFs) – 2 DOFs (vertical displacement of unsprung mass Z_i, i=1,…, 4 and road elevation Z_{ti}, i=1,…, 4) to represent the suspension of each wheel, the vertical vehicle displacement Z_{cg} at the CG location, the vehicle roll angle Φ at CG, and the vehicle pitch angle Θ at CG. The equations of motion are presented in state-space form in Appendix A.

M_{cg}	: Mass of vehicle body (Kg)
M_1 , M_2 , M_3, M_4	: Unsprung mass of front RH, front LH, rear RH and rear LH wheels (Kg)
Z_{cg}	: Body displacement at CG location (m)
Φ	: Body roll angle at CG (degrees)
θ	: Body pitch angle at CG (degrees)
I_{xx}, I_{yy}	: Moment of inertia around X-X and Y-Y axes (kg-m^2)
a, b	: Distance from CG of front and rear wheels (m)
c,d	: Distance from CG of feft and right wheels (m)
K_1, K_2, K_3, K_4	: Spring stiffness of front RH, front LH, rear RH and rear LH suspension (N/m)
C_1, C_2, C_3, C_4	: Damping coefficient of front RH, front LH, rear RH and rear LH suspension (N-s/m)
Z_1, Z_2, Z_3, Z_4	: Unsprung mass displacement of front RH, front LH, rear RH and rear LH suspension (m)
K_{t1}, K_{t2}, K_{t3}, K_{t4}	: Tire spring stiffness of front RH, front LH, rear RH and rear LH suspension (N/m)
Z_{t1}, Z_{t2}, Z_{t3}, Z_{t4}	: Road inputs of front RH, front LH, rear RH and rear LH wheel (m)

Figure 6.2 Parameters of Full-vehicle analytical model

The vehicle is driven over rough terrain which is modeled as a stationary Gaussian field. The road elevation has the same mean, standard deviation and Power Spectral Density (PSD), or equivalently autocorrelation function, with a Gaussian Road elevation according to ISO 8608 standard [30]. A Gaussian longitudinal road profile following the ISO 8608 standard is described by a zero mean, Gaussian random process with the following PSD

$$S_{UU}(\omega) = \frac{2\sigma^2 av}{(\omega^2 + a^2 v^2)} \; m^2/(rad/s) \tag{6.1}$$

where σ^2 and a are the road profile variance and coefficient respectively, ω is the circular road frequency in rad/s, and v is the vehicle speed. Table 6.1 shows the utilized values. Figure 6.3 shows $S_{UU}(\omega)$.

Table 6.1 Road profile parameters

Quantity	Description	Value
σ^2	Variance of Road Profile	0.001 m^2
A	Road Profile Coefficient	0.3375 rad/m
V	Vehicle Speed	60 km/h

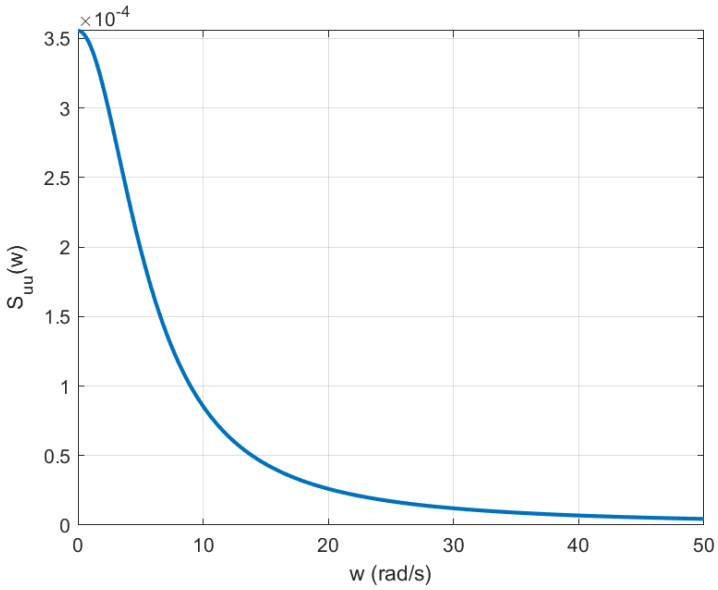

Figure 6.3 PSD of sample road profile

The PSD is transformed into the corresponding autocorrelation function using an inverse Fourier transform and then used to form the covariance matrix which expresses the correlation of road elevation between two road locations. The Karhunen-Loeve Expansion (KLE) is finally used, based on an eigenvalue analysis of the covariance matrix, to provide a space representation of the random field (Figure 6.4) expressing the road elevation in two dimensions. Figure 6.4 shows the generated road random field covering a 200 *m* by 200 *m* terrain and some arbitrarily chosen paths (red lines) on the field. The vehicle is run on each of 500 generated paths assuming that the road elevation

at each wheel is the same. Figure 6.5 shows the road elevation in meters for the 500 paths.

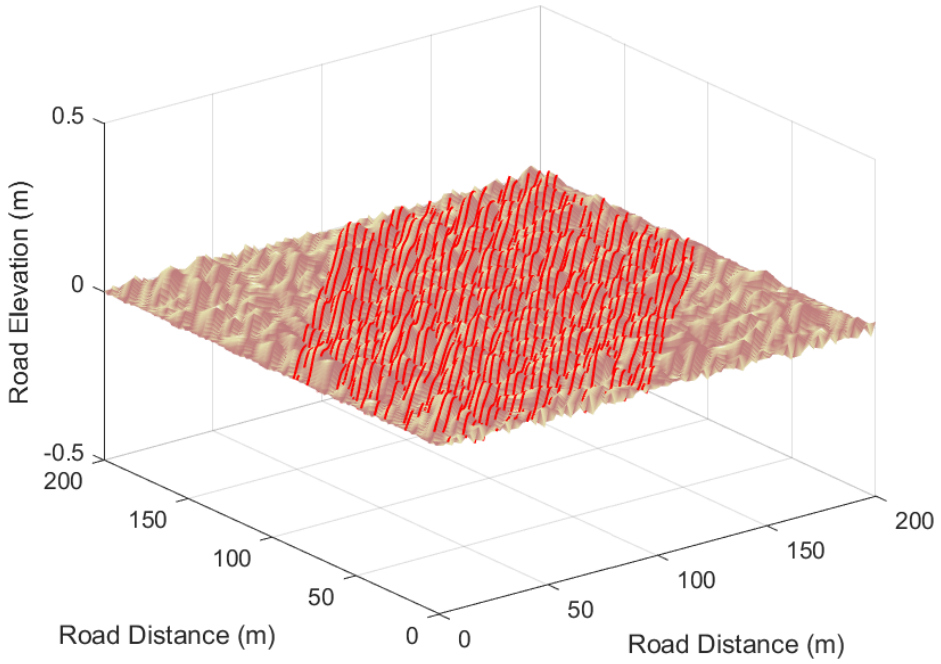

Figure 6.4 Random field and 500 realizations of road profiles

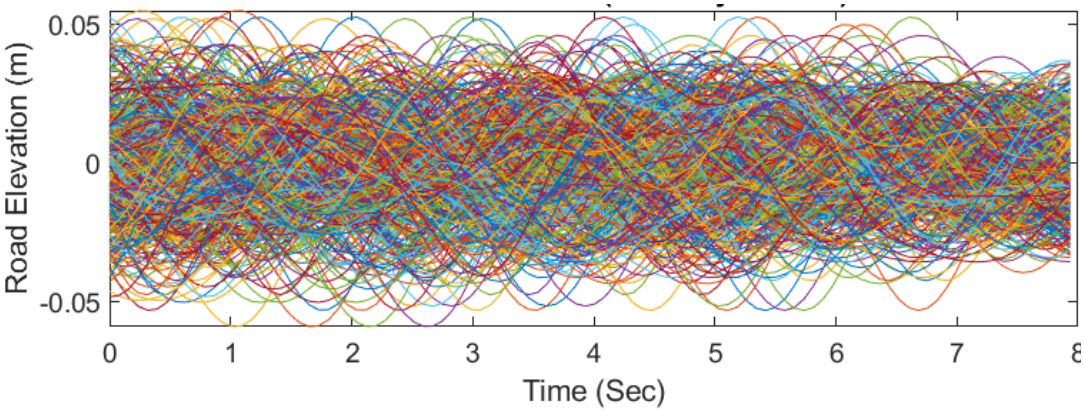

Figure 6.5 Road input elevation for 500 realizations of road profiles

Each road elevation of Figure 6.5 is applied to all four wheels and the resulting acceleration response is obtained at the vehicle center of gravity (CG) assuming a vehicle speed of 64.8 kph (18 m/s). Each acceleration response consists of 102-time steps covering the path length in 8 sec. Out of the 500 input-output profiles (*m)*, 475 are used for training a machine learning model (m_{train}) , while the remaining 25 (m_{test}) are used for validation. To predict the acceleration response for an unseen (not used for training) road profile, a Long Short-Term Memory (LSTM) network is constructed as shown in Figure 6.6.

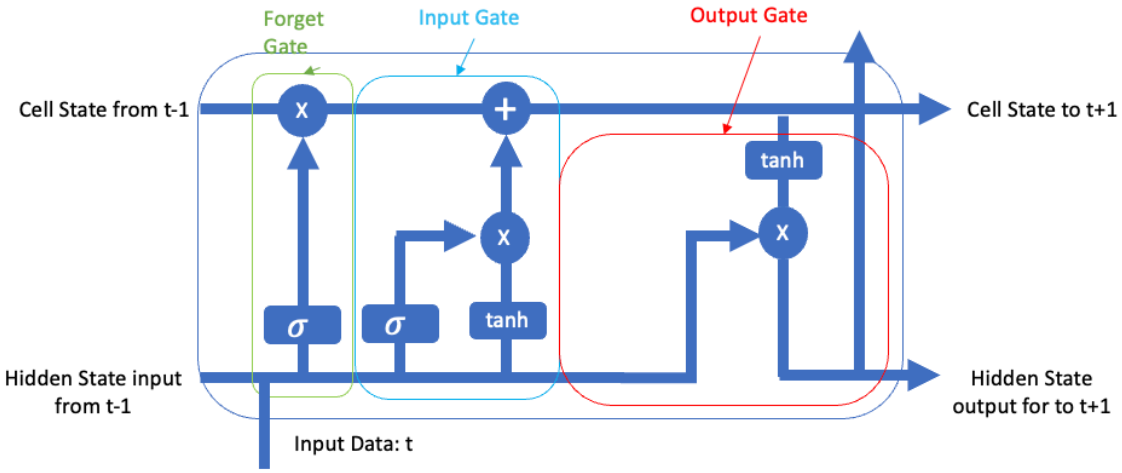

Figure 6.6 Sample LSTM neural network architecture

6.1 Long Short-Term Memory (LSTM) Cell Architecture

Traditional Recurrent Neural Networks (RNNs) often encounter difficulty in capturing long-term dependencies due to the vanishing and exploding gradient problem. To mitigate this, the Long Short-Term Memory (LSTM) architecture was proposed by Hochreiter and Schmidhuber [40]. The LSTM cell introduces a memory mechanism that

regulates information flow via gating units, enabling the network to preserve relevant information over extended sequences. As illustrated in Figure 6.6, an LSTM cell at time step t receives three inputs: the current input vector x_t the previous hidden state h_{t-1} and the previous cell state C_{t-1}.The internal operations of the cell are governed by the following components:

Forget Gate: The forget gate decides which parts of the previous cell state should be discarded. It is calculated as

$$f_t = \sigma(W_{f[h_t-1,x_t]} + b_f) \tag{6.2}$$

where σ denotes the sigmoid function, and $W_{f[h_t-1,x_t]}$, b_f are the associated weight matrix and bias, respectively [40].

Input Gate and Candidate Values: The input gate controls the extent to which new information is added to the cell state, while a candidate vector $\tilde{C}_t$is created as a potential addition:

$$i_t = \sigma(W_{i[h_t-1,x_t]} + b_i) \tag{6.3}$$

$$C_t = \tanh(W_{C[h_t-1,x_t]} + b_C) \tag{6.4}$$

Cell State Update: The new cell state is computed by combining retained memory from the past and the candidate values:

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \tag{6.5}$$

Where $\odot$ denotes the elementwise (Hadamard) product.

Output Gate and Hidden State: The output gate decides how much of the cell state should influence the hidden state output:

$$o_t = \sigma(W_{o[h_t-1,x_t]} + b_o) \tag{6.6}$$

$$h_t = o_t \odot \tanh(C_t) \tag{6.7}$$

This design enables the LSTM cell to selectively remember or forget information over time, making it well-suited for sequential modeling tasks such as language modeling, time-series forecasting, and speech recognition [40][41].

The LSTM neural network used in this study is designed to model time-series data with multiple inputs and outputs. The architecture includes 102 input features (road elevation in 102-time steps) and produces 102 corresponding outputs (acceleration in 102-time steps). The network operates over a sequence length of 10-time steps, capturing temporal dependencies across this range. Each LSTM cell contains 50 hidden units, allowing the model to maintain a robust memory over time, which is crucial for tasks requiring sequential information processing. The internal structure of the LSTM unit (Figure 6.7) includes the typical components of an LSTM cell: input gate, forget gate, output gate, and a cell state. These gates regulate the flow of information, enabling the model to retain or discard information as necessary for effective sequence learning. The accuracy of the developed LSTM network is evaluated by using the 14^{th} of the 25 test data points. As shown in Figure 6.8 the predicted response from the LSTM model

matches closely the actual response.

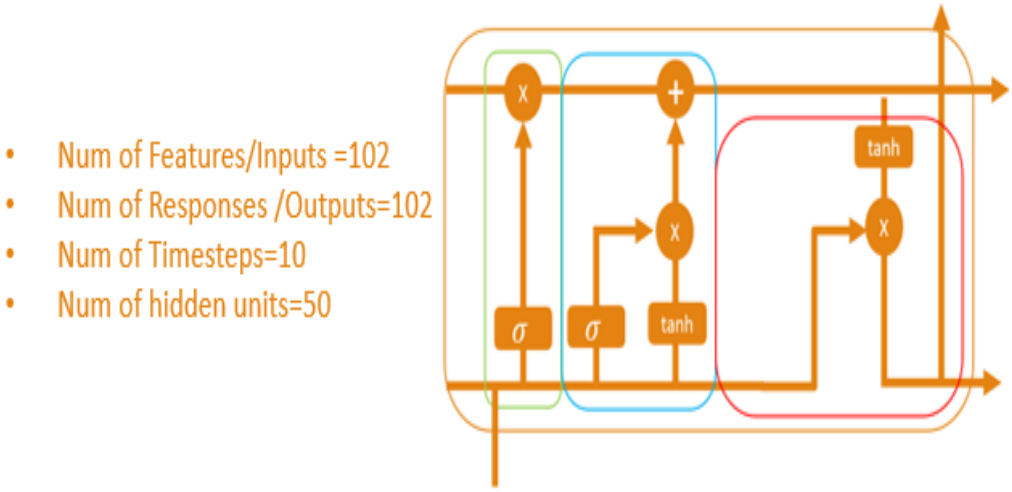

Figure 6.7 LSTM architecture used for vehicle dynamics problem

Using the same architecture as in Figure 6.7, k_{ML} =100 separate LSTM models were trained. The predicted responses from each of the 100 models (Figure 6.9) were used in a network ensemble to demonstrate the capability of the proposed method to assess UQ in dynamic systems.

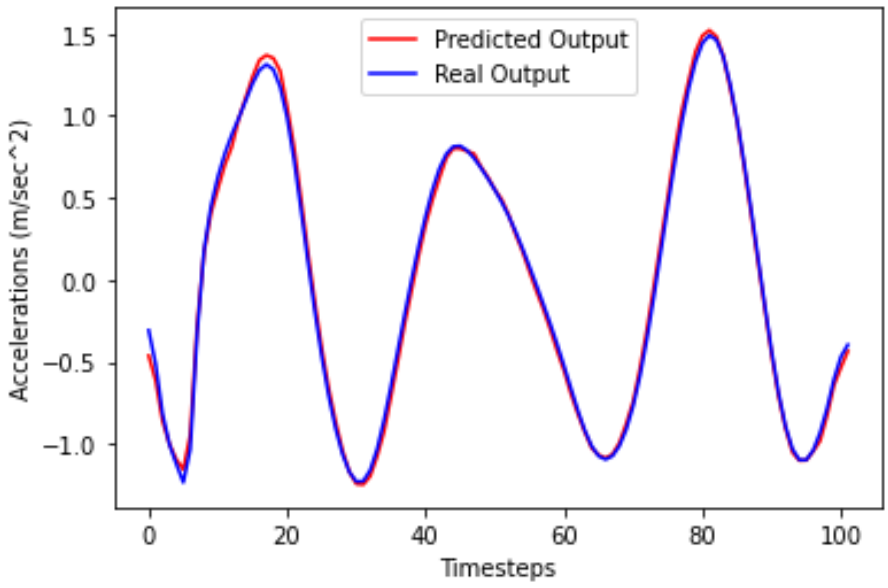

Figure 6.8 Comparison of actual and predicted acceleration response

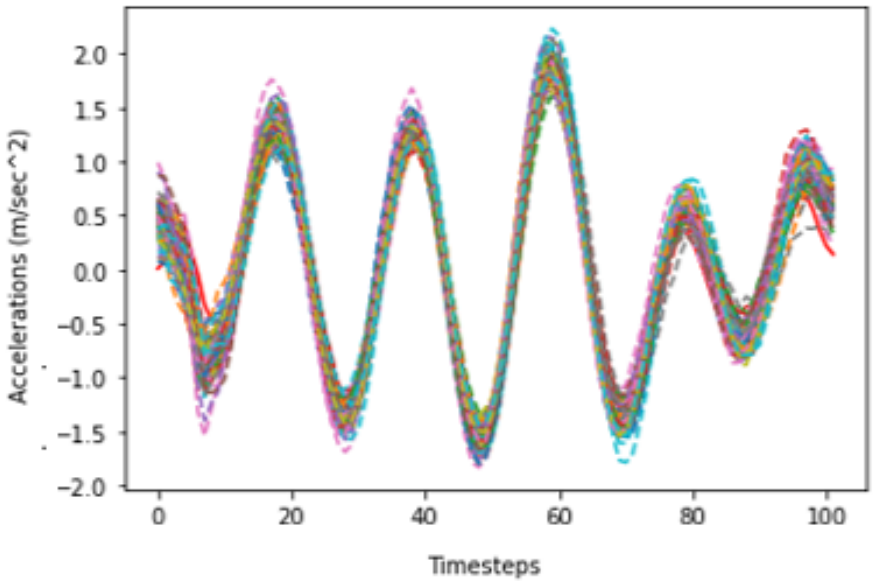

Figure 6.9 Network ensemble of LSTM models

Using the proposed method, uncertainty was estimated using the unseen 14th data point (Figure 6.10). The actual response is represented by the red line, while the predicted 5th and 95th percentiles are represented by the blue and green dotted lines, respectively. Although not seen in a figure, the predicted percentiles are almost the same with their exact values. Table 6.2 demonstrates the efficiency of the proposed method indicating that it provides uncertainty estimation in a fraction of a second.

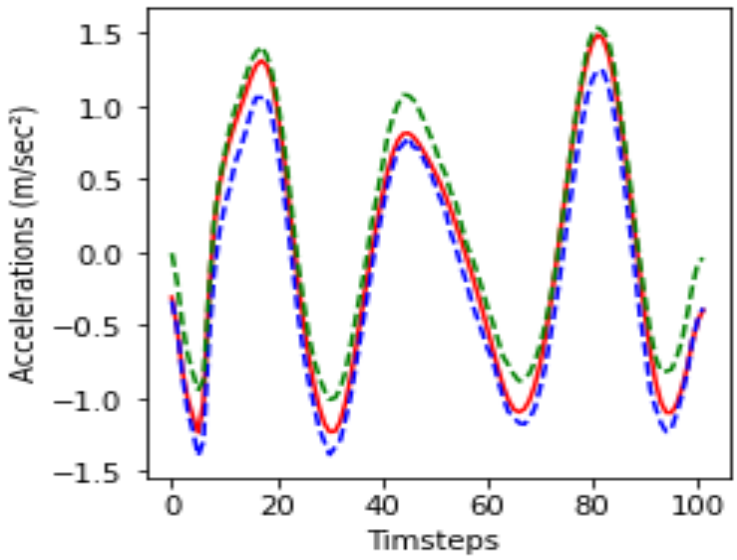

Figure 6.10 Uncertainty estimation using proposed method

Table 6.2 Computational time to estimate uncertainty

	Time
Ensemble of 100 LSTM models	24 sec
5^{th} and 95^{th} percentile predictions from proposed method	0.4 sec

We also estimated the quality of confidence intervals using a calibration plot technique [3]. The results are summarized in the calibration plot of Figure 6.11. The "Expected Confidence" refers to the confidence level based on the prediction intervals. For example, for a 90% confidence interval, we expect 90% of the true values to fall

within this interval. The “Observed Confidence,” on the other hand, is the actual fraction of true values that fall within the predicted intervals.

The diagonal line represents an ideal scenario where the observed confidence perfectly matches the expected confidence. The blue dots show the observed confidence for different expected confidence levels. If the points closely follow the diagonal line, the model's UQ prediction intervals are accurate. If the points are below the diagonal line, the model is overconfident, indicating that the intervals are too narrow and therefore, fewer true values fall within them than expected. Conversely, if the points lie above the diagonal line, the model is underconfident, with wide UQ intervals and more true values falling within them than expected. In applications where safety is crucial, an underconfident model might be preferred ensuring that most of the true values fall within the predicted UQ intervals reducing the risk of making overconfident and potentially dangerous decisions.

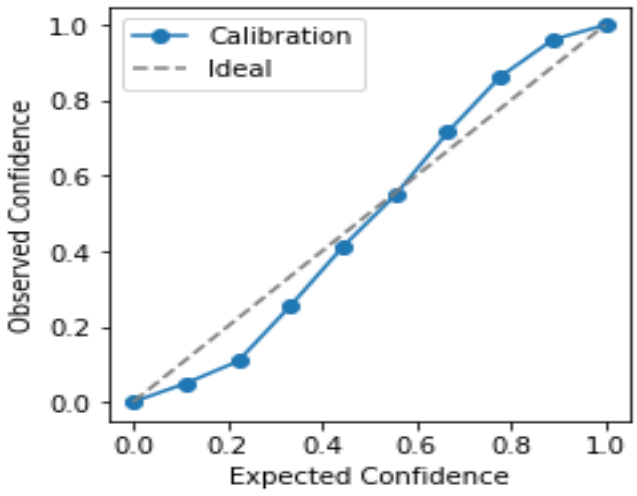

Figure 6.11 Calibration plot for network ensemble

CHAPTER 7

VALIDATION OF THE PROPOSED METHOD ON REAL WORLD CRASH SIMULATION PROBLEM

This chapter discusses the validation of the method developed to quantify uncertainty in machine learning models using a real-world problem. For demonstration, a public domain crash simulation model [42] is utilized to evaluate the behavior of a car colliding with a barrier. The simulation uses Finite Element Analysis (FEA), also known as Computer-Aided Engineering (CAE), to analyze the complex physical phenomena involved in such impact events. The Toyota Yaris (2010) model, which is publicly available [42] is used for demonstration purposes.

Figure 7.1 Public domain crash model

In the CAE process, the initial step involves converting a Computer-Aided Design (CAD) model into a finite element mesh. The discretization process divides the geometry into smaller elements, which are analyzed to approximate the physical behavior of the vehicle during impact. For this study, a mesh size of 10 mm was chosen, ensuring a balance between computational efficiency and the level of detail required to capture critical deformation patterns. The discretized model is then prepared for simulation by assigning appropriate material properties, boundary conditions, and contact interactions, making it ready for dynamic analysis using a robust finite element solver.

The Toyota Yaris (2010) model represents a passenger vehicle, incorporating essential structural features such as crumple zones, energy-absorbing members, and reinforcements. These features are crucial for understanding the mechanisms that mitigate impact forces and protect occupants during a collision. The rigid barrier, which the vehicle impacts, is modeled as an immovable structure to replicate standardized crash test scenarios, such as frontal impact tests defined by regulatory agencies. The simulation uses LS-DYNA, a highly advanced and versatile solver, well-suited for non-linear dynamic problems. LS-DYNA excels in handling the complexities of crash simulations, including large deformations, non-linear material behavior, and intricate contact mechanics. The solver provides detailed insights into the vehicle's performance during the crash, capturing key metrics such as deformation patterns, stress distribution, energy absorption, and potential points of structural failure.

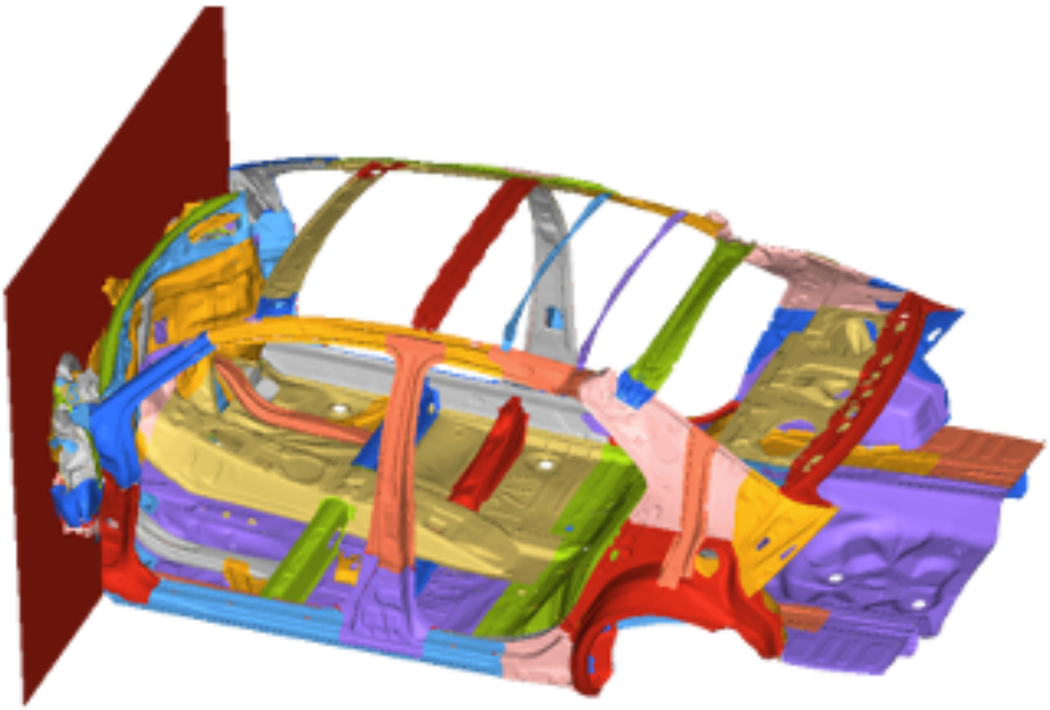

Figure 7.2 Simplified vehicle body used for crash simulation

A full-vehicle finite element model is inherently large and highly complex, as it includes all major subsystems such as the vehicle body, powertrain, interior, exterior components, and closures. While these comprehensive models provide a detailed representation of the vehicle, they are computationally intensive and typically require significant runtime — often exceeding 16 hours per simulation. To optimize computational efficiency for this exercise, only the vehicle body subsystem is used in the simulation as shown in Figure 7.2. This simplification significantly reduces the computational load while still capturing the essential dynamics of the crash event. The current simulation models the crash event up to 120 milliseconds and completes in approximately 10 minutes, striking a balance between accuracy and efficiency.

The response measured during the simulation is acceleration, measured in mm/msec². This measurement is taken at a specific location referred to as the “toepan”, as illustrated in Figure 7.3.

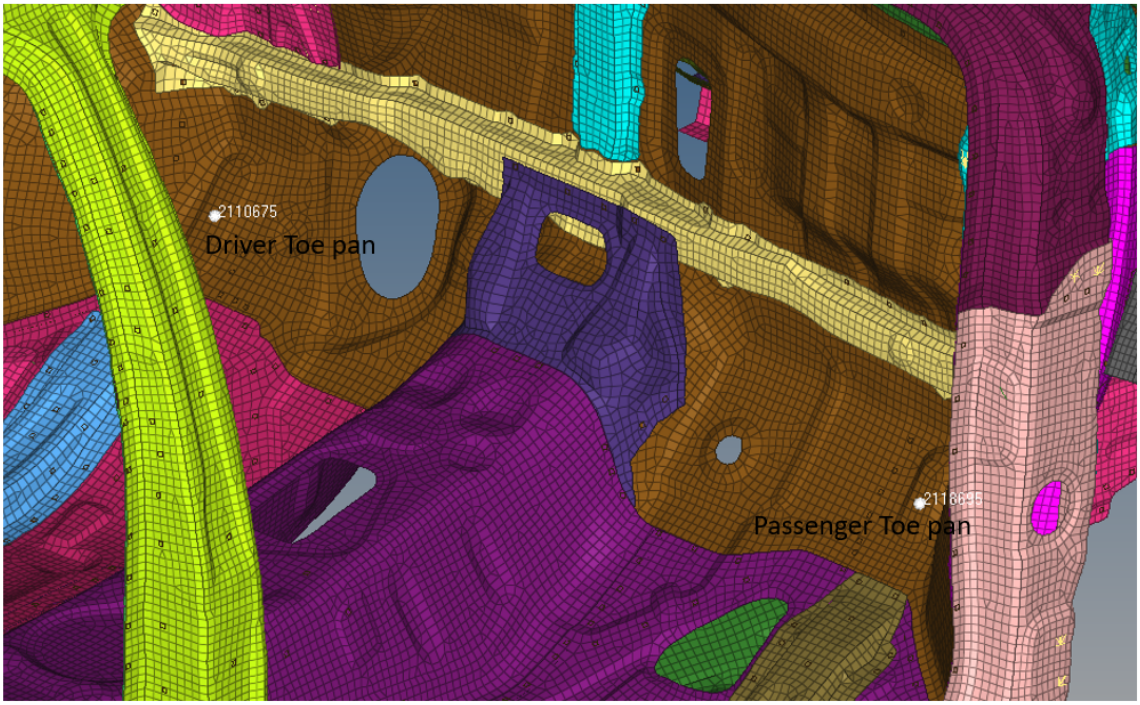

Figure 7.3 Acceleration response at toe pan

To generate the dataset for the aforementioned simulation, 33 parts from the front of the vehicle were selected, as illustrated in Figure 7.4. Each part was assigned a design variable corresponding to its thickness. The range for each design variable was set to ±20% of its nominal thickness. A total of 662 designs were generated, including two extreme cases: one with all design variables set to their minimum values and another with all design variables set to their maximum values. Correspondingly, 662 FEA (Finite Element Analysis) decks were created, and simulations were performed using LS-DYNA.

For each simulation run, the acceleration at the Driver Toe Pan over a duration of 120 milliseconds was recorded as the response. The acceleration response for all 662 runs can be seen in Figure 7.5.

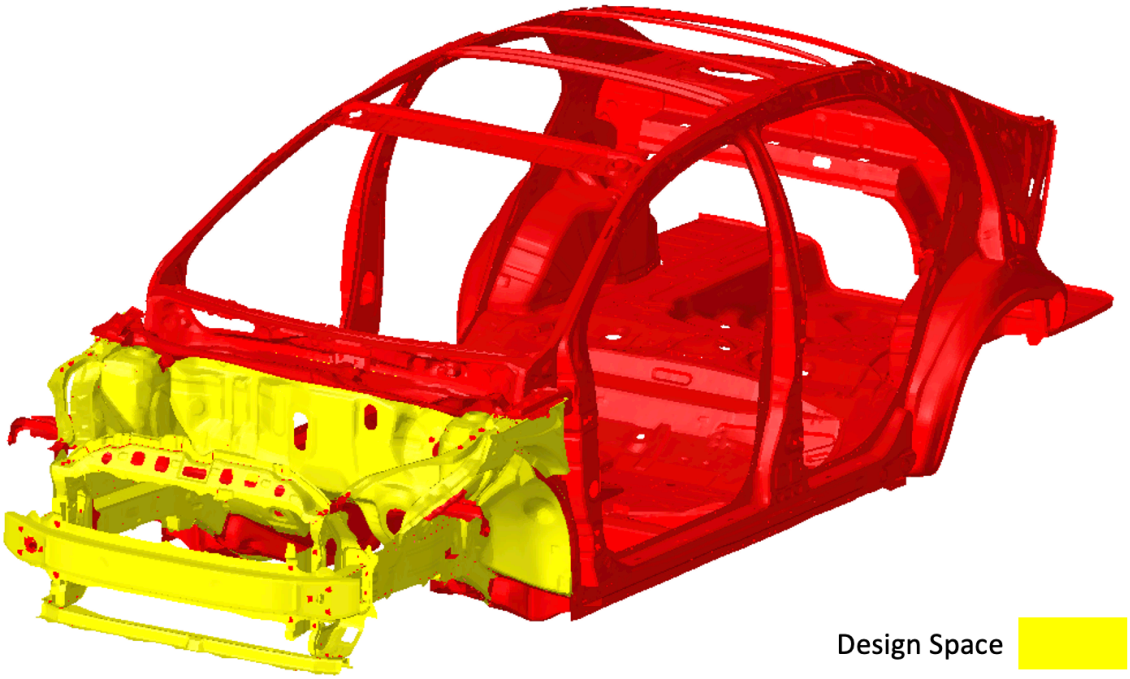

Figure 7.4 Thirty-three parts included in design space

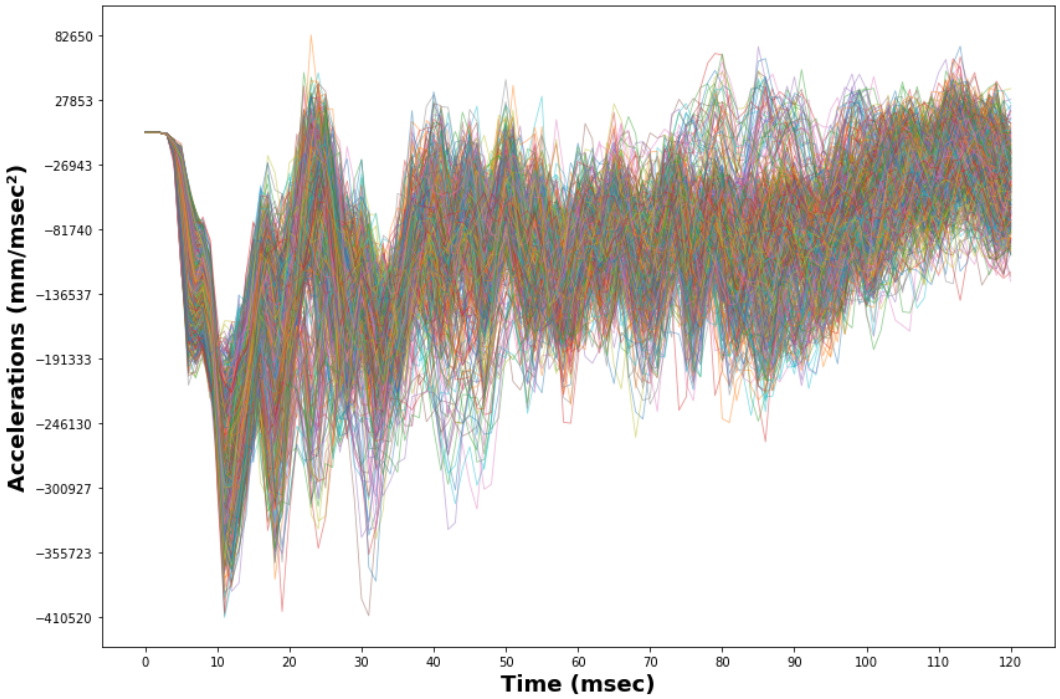

Figure 7.5 Acceleration response of toe pan

7.1 Development of Machine Learning Model

In this study, the problem involves 33 input variables, representing thickness parameters, and a time-series response corresponding to the acceleration measured at the driver toe pan location over a duration of 120 milliseconds. A Fully Connected Neural Network (FCNN) architecture was selected as the machine learning model.

The architecture of the network is illustrated in Figure 7.6. The input layer consists of 33 neurons, representing the 33 design variables, while the output layer comprises 121 neurons to represent the acceleration response across 120-time steps. The network features six layers with the following neuron configurations: 256, 128, 64, 32, and 16 neurons, as detailed in Figure 7.6. Additional parameters used in the neural network are summarized in Table 1. Out of 662 designs, 650 were used to train the ML model, while 12 were set aside as the testing

Table 7.1 Neural network parameters

Parameter	Value
Activation function	relu
Kernel regularizer	L2
Learning rate	0.0001
Loss function	mse
Early stopping patience	10
Early stopping monitor	Validation loss
Epochs	1000
Batch size	16

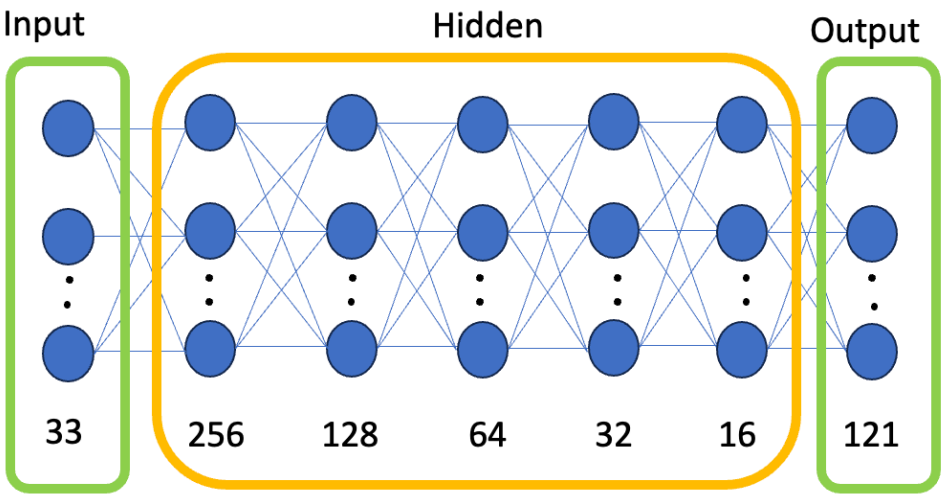

Figure 7.6 Neural network architecture

Figure 7.7 shows that the training loss steadily decreases, while the validation loss plateaus around 80-100 epochs, indicating convergence. The gap between losses suggests slight overfitting, where the model performs better on training data than validation.

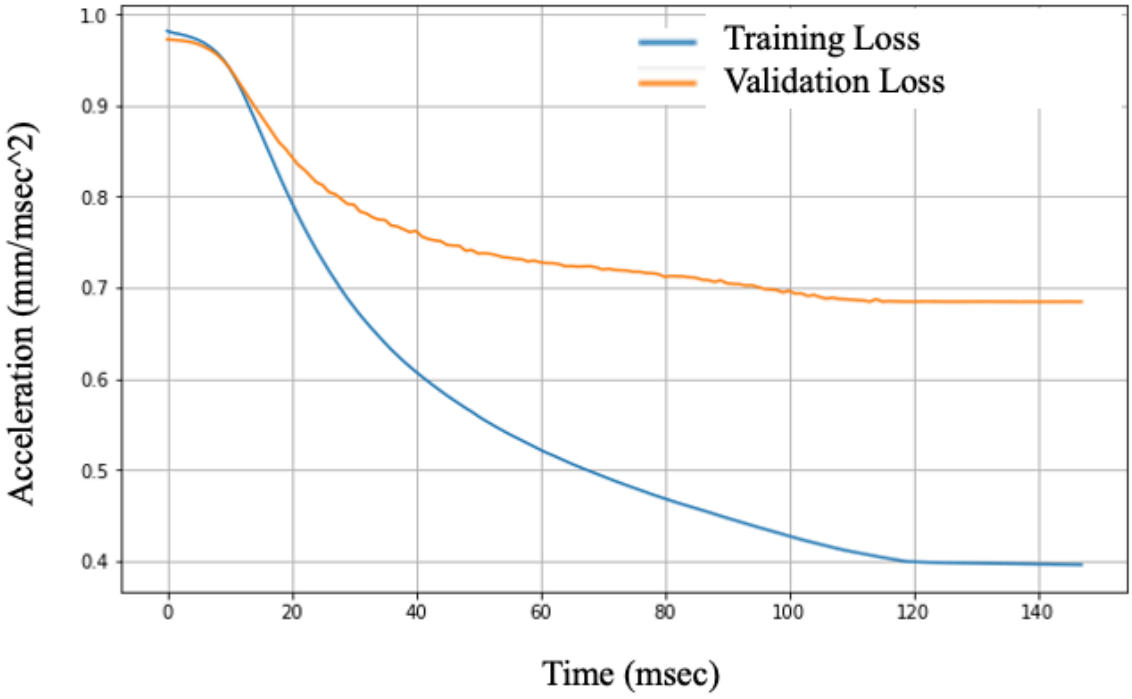

Figure 7.7 Training of neural network model

The third design from the testing dataset was used to evaluate the accuracy of the prediction. As shown in Figure 7.8, the predicted curve aligns well with the trends of the actual curve.

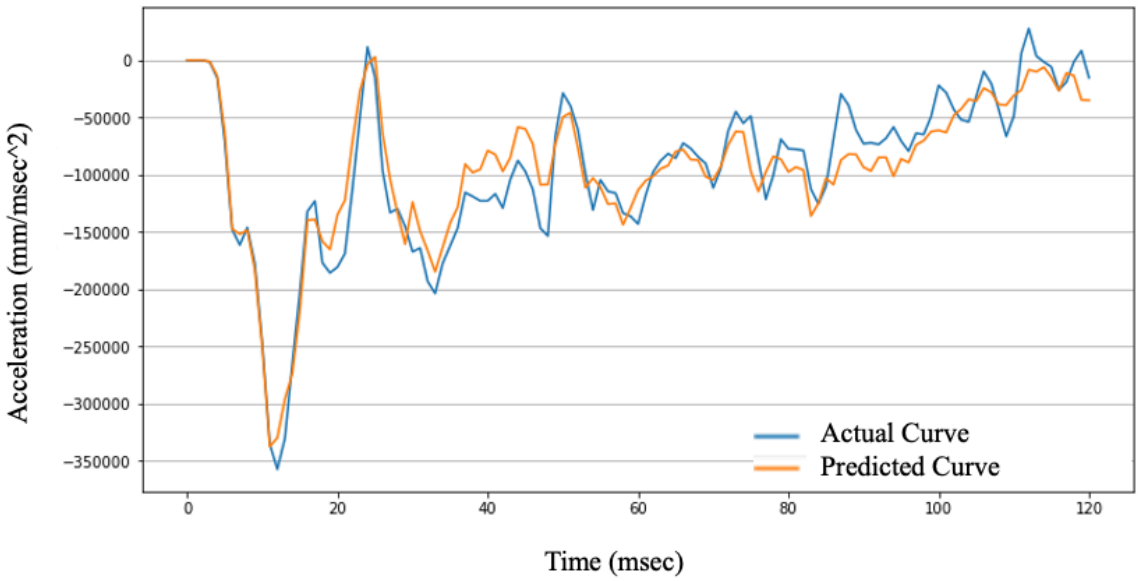

Figure 7.8 Predicted acceleration vs actual acceleration

To demonstrate uncertainty, the same neural network model was trained 500 times using an identical architecture. The predictions were obtained using the same unseen test data (i.e., the third design from the test dataset). The ensemble of predictions is illustrated in Figure 7.9, revealing noticeable uncertainty. This uncertainty must be quantified to enhance confidence in the predictions.

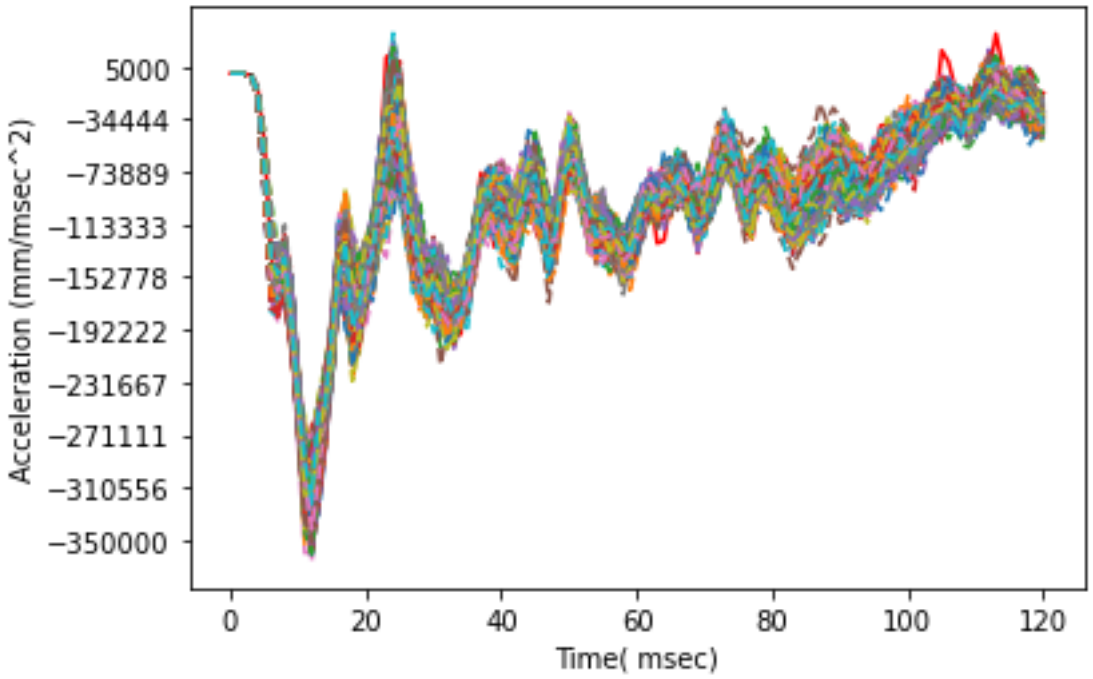

Figure 7.9 Assemble of predictions using 500 ML models

7.2 Uncertainty Quantification using the Proposed Method

As illustrated in Figure 7.9, significant uncertainty is observed in machine learning models trained repeatedly with identical configurations, highlighting the inherent variability in predictions. To address this, an Adaptive Gaussian Process Regression (GPR) framework is developed using a Neural Network Ensemble. The training dataset is divided into two parts: 80% is used to train an ensemble of 500 neural networks, while the remaining 20% is reserved for testing. The trained ensemble generates predictions for the test set, from which the 5th and 95th percentiles are computed at each test point, capturing the lower and upper bounds of the output distribution. These percentile estimates serve as targets for initial GPR models that are trained to predict the respective quantiles. The GPR models are then refined using the Adaptive GPR procedure described in Chapter 5, iteratively updating the models over 25 cycles to improve accuracy. The

resulting GPR models are subsequently employed to estimate uncertainty for unseen inputs. The effectiveness of this approach is demonstrated in Figure 7.10, where the uncertainty bounds for the same test data used in Figure 7.9 are shown using the final GPR models derived from the Adaptive GPR loop. As shown, the actual output lies within the bounds estimated by the proposed method, demonstrating its effectiveness in uncertainty quantification for real-world time series problems. Figures 7.11 to 7.15 show the uncertainty quantification of the acceleration response using the proposed technique for 5 more unseen input data.

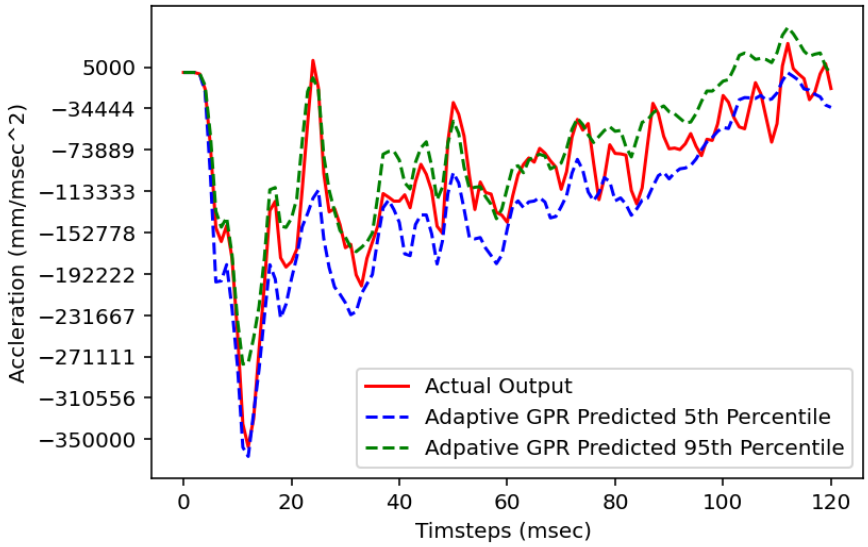

Figure 7.10 Uncertainty estimation using the proposed method for test 3

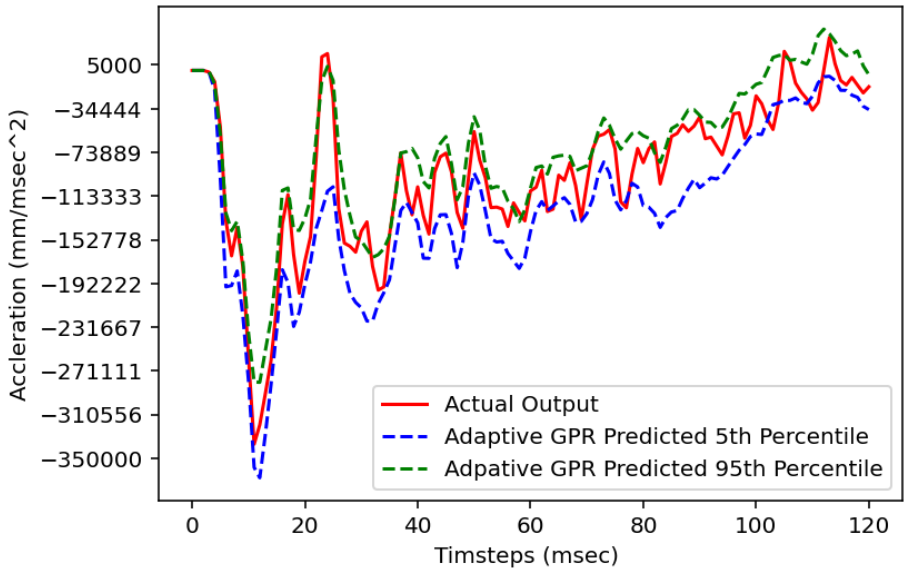

Figure 7.11 Uncertainty estimation using the proposed method for test 1

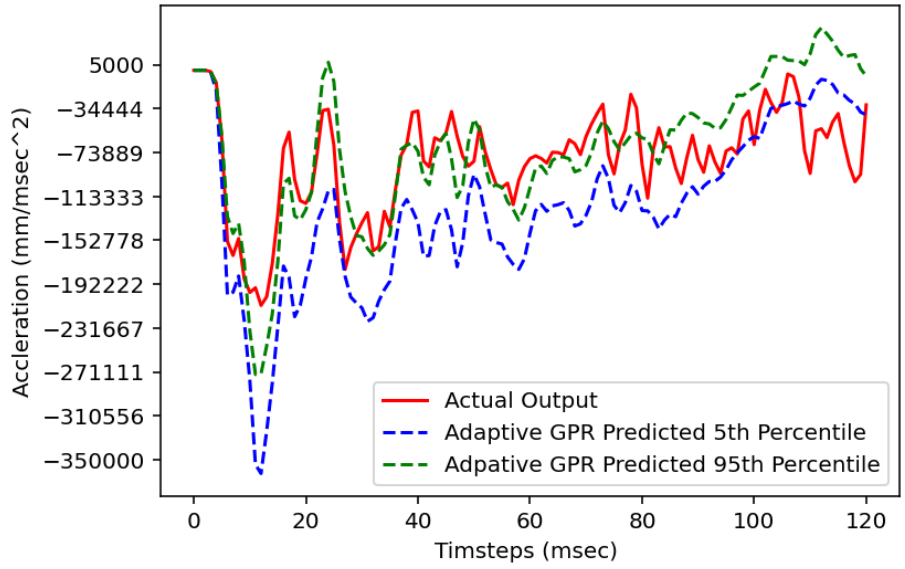

Figure 7.12 Uncertainty estimation using the proposed method for test 2

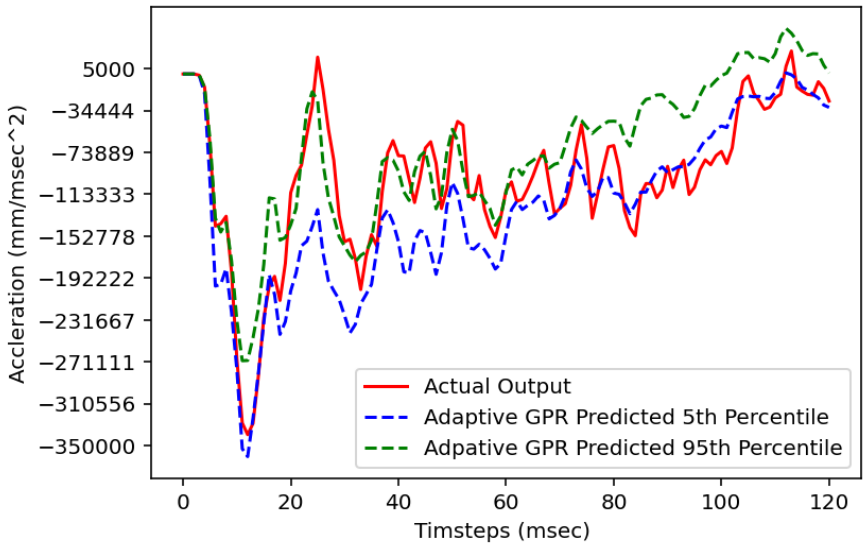

Figure 7.13 Uncertainty estimation using the proposed method for test 6

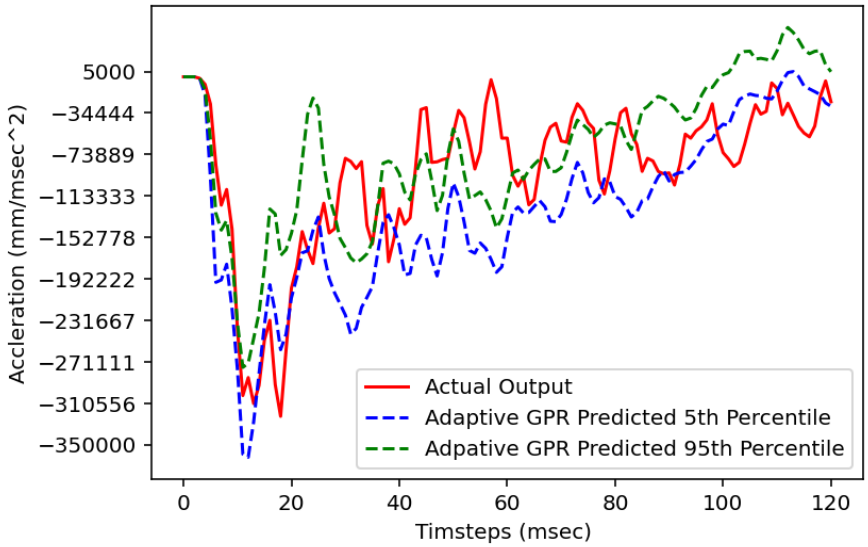

Figure 7.14 Uncertainty estimation using the proposed method for test 7

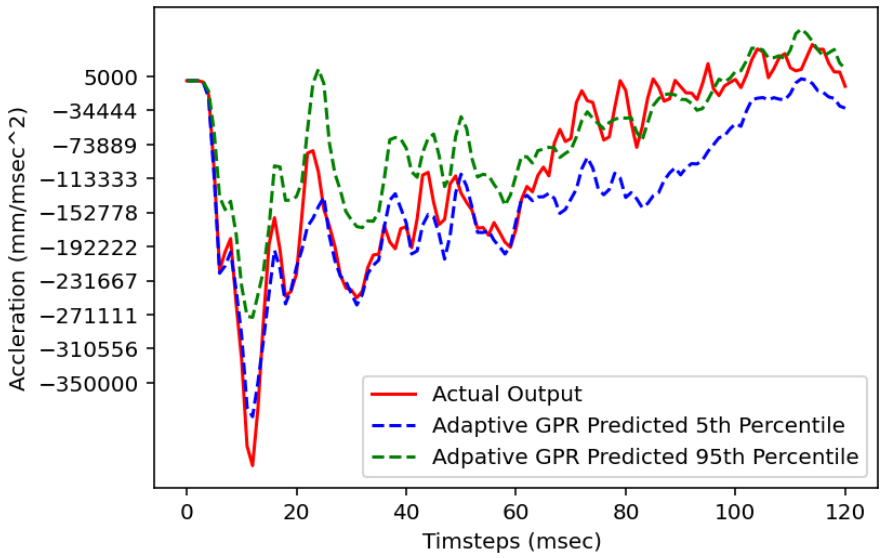

Figure 7.15 Uncertainty estimation using the proposed method for test 10

7.3 Accuracy of Predictions

The proposed method can be effectively used to quantify uncertainty in machine learning models, even when the underlying data distribution is non-Gaussian. The approach hinges on two key strategies: (1) utilizing percentile-based targets rather than full probabilistic modeling of the raw data, and (2) interpreting GPR not as a generative model of data, but as a functional interpolator. The standard GPR assumes a Gaussian prior over functions, not over the observed data itself. That is, it posits that any finite collection of function evaluations is jointly Gaussian distributed. However, this assumption pertains to the latent function space, not to the distribution of the observed outputs [43]. This key distinction allows GPR to model mean behavior and function uncertainty even when the real-world data is heavy-tailed, skewed, or heteroscedastic.

The GPR's strength lies in smooth kernel-based interpolation, making it suitable for modeling relationships rather than raw empirical distributions.

In the proposed method, multiple neural networks are trained with varying initializations and hyperparameters to form an ensemble that captures parameter uncertainty. For each test input, the ensemble outputs are aggregated to compute the 5th and 95th percentiles, effectively summarizing predictive uncertainty without assuming a specific parametric form. By focusing on quantile surfaces (5th and 95th percentile functions) instead of full distributions, the method avoids making any assumptions regarding the Gaussian nature of the target data. This idea echoes principles from quantile regression, which estimates conditional quantiles without making assumptions about distributional form [45]. The separate GPR models trained on the 5th and 95th percentiles serve as interpolators of these empirical summaries. Because percentiles are deterministic functions of input data, GPR is used not to model uncertainty in the percentile values themselves, but to interpolate how they vary across the input space [44].

To validate the claim, the data shown in Figure 7.5 is used to identify timesteps that exhibit non-Gaussian behavior. For this exercise skewness of data is calculated per timestep as skewness is a statistical measure that quantifies the degree of asymmetry in the distribution of a real-valued random variable about its mean. It provides insight into the shape of the probability distribution, particularly how much and in which direction the data deviates from a normal (symmetric) distribution.

Mathematically, the skewness γ of a dataset with n observations $x_1, x_2, .., x_n$, mean μ, and standard deviation σ, is defined as:

$$\gamma = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i-\mu}{\sigma}\right)^3 \quad (7.1)$$

For Gaussian distribution $\gamma = 0$ while for positively skewed distribution $\gamma > 0$ and negatively skewed distribution $\gamma < 0$. Figure 7.16 shows the skewness calculated across all 120 timesteps and Figure 7.17 shows the timesteps that exhibit the most non-Gaussian behavior.

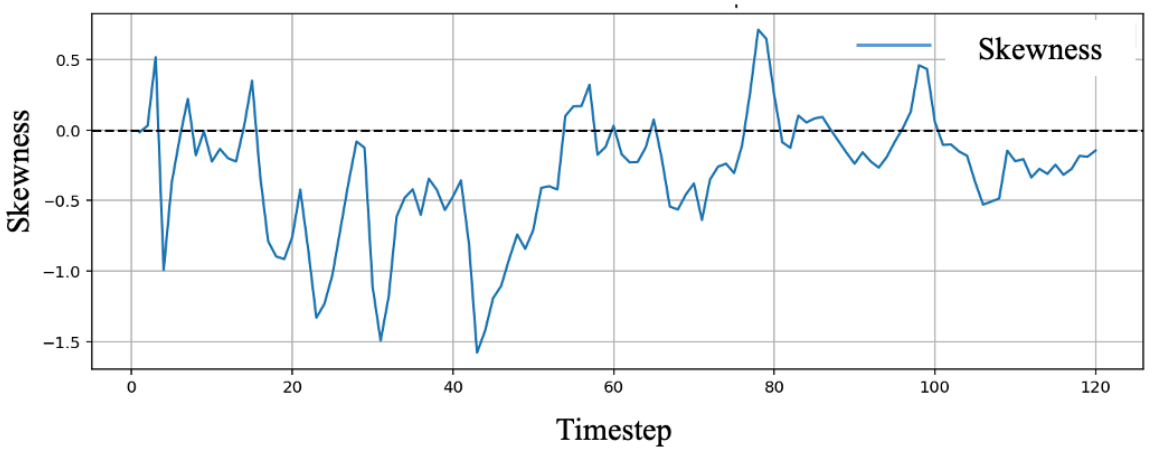

Figure 7.16 Skewness across all time steps

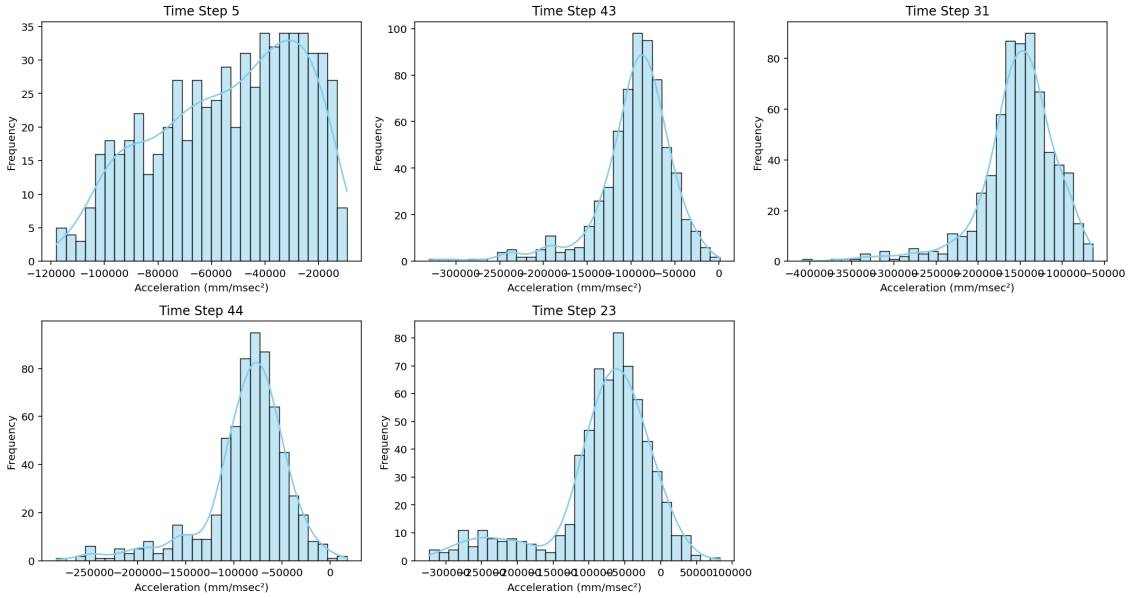

Figure 7.17 Timesteps with non-Gaussian distribution

As shown in Figures 7.11 to 7.15, the proposed methodology demonstrates accurate quantification of predictive uncertainty, even in scenarios where the underlying data distribution is non-Gaussian.

CHAPTER 8

AUTOENCODERS BASED ANAMOLY DETECTION

The reliability of the proposed uncertainty quantification framework is intrinsically linked to the fidelity of the model ensemble in capturing the ground truth response. As demonstrated in Chapter 7, the method effectively characterizes the uncertainty arising from variability in model weights. However, it is crucial to underscore that this performance is contingent upon the ensemble's ability to encompass the true output behavior. To further illustrate this point, the same dataset and experimental case (Test 3) discussed in Chapter 7 is revisited. Figure 8.1 presents the predictions from an ensemble of 500 neural network models, wherein the true response — shown in red — is nearly indistinguishable due to its close alignment with the ensemble predictions. This indicates that the ground truth is well enveloped within the ensemble range. Figure 8.2 provides a comparative visualization using only 10 ensemble members, with the ground truth again marked in red. Despite the reduced number of models, the predictions still encapsulate the actual response with reasonable accuracy. These visual comparisons affirm that, for Test 3, the ensemble of neural networks — whether large or modest in size — accurately captures the underlying system behavior, thereby validating the efficacy of the proposed uncertainty quantification method.

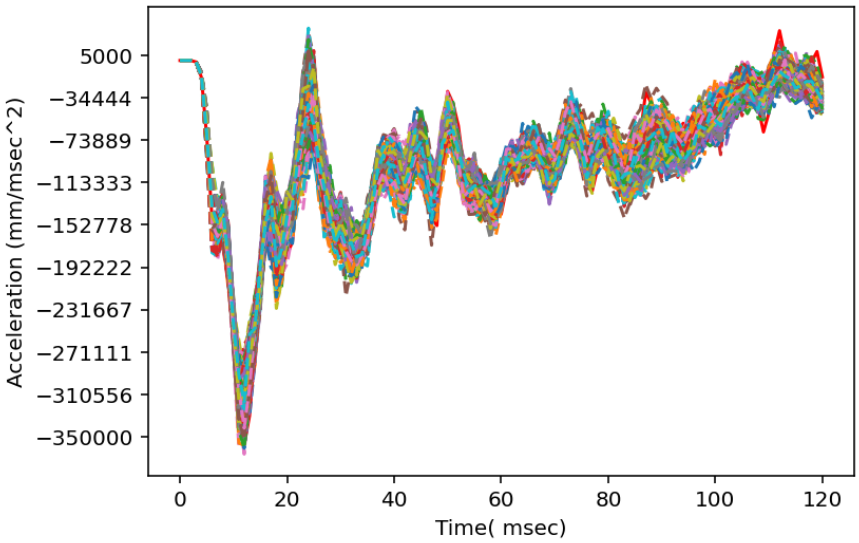

Figure 8.1 Ensemble of 500 Neural Network Models for test 3

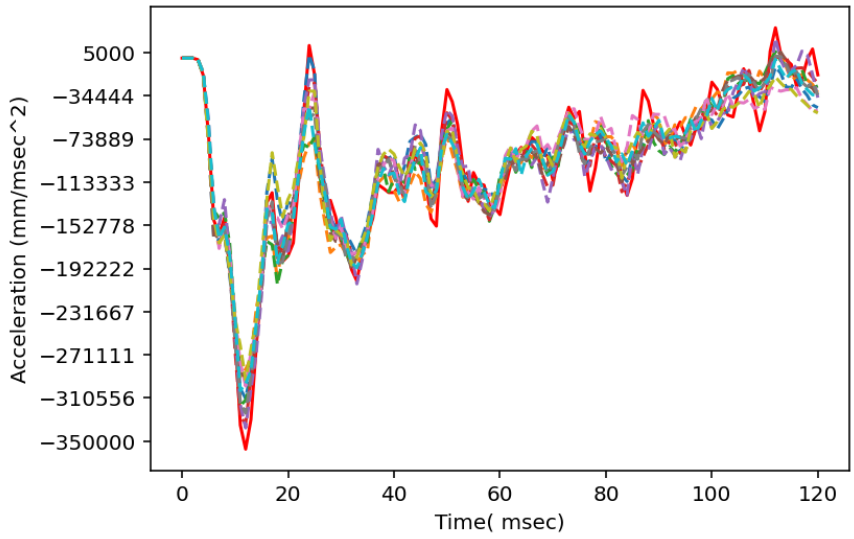

Figure 8.2 Ensemble of 10 Neural Network Models for test 3

However, when evaluating the generalizability of the proposed method on a different unseen data instance (Test 5) a contrasting behavior is observed. As shown in Figure 8.3, the ground truth trajectory (highlighted in red) deviates noticeably from the

predictions generated by the ensemble of 500 neural network models. This deviation indicates that the ensemble fails to fully encompass the true response in this case. As a direct consequence, the uncertainty quantification produced by the proposed framework becomes less reliable, as shown in Figure 8.4. The error in uncertainty estimation is particularly pronounced in regions where the ensemble fails to capture the ground truth. This example underscores a critical limitation: the effectiveness of the uncertainty quantification is inherently dependent on the representational accuracy of the ensemble. When the ensemble does not adequately approximate the true behavior, the resulting uncertainty bounds may be misleading or incomplete.

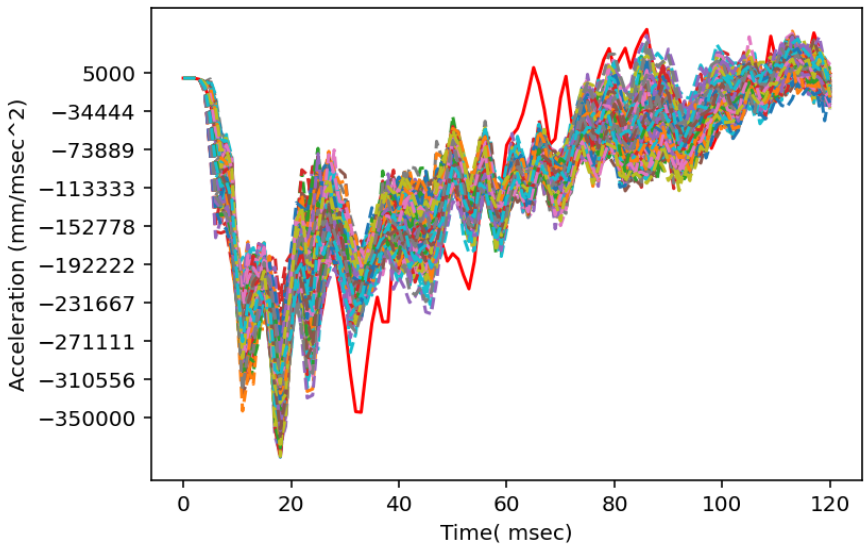

Figure 8.3 Ensemble of 500 neural network models for test 5

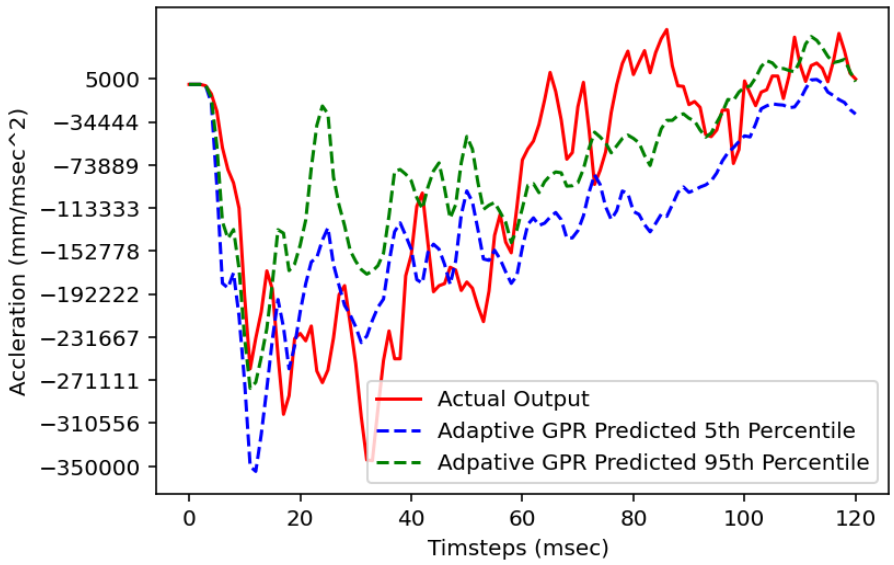

Figure 8.4 Uncertainty estimation using the proposed method for test 5

This observation necessitates the development of a mechanism to flag instances where the ensemble of neural networks may fail to adequately capture the ground truth, thereby compromising the reliability of the proposed uncertainty quantification framework. To address this critical need, this chapter introduces a novel Autoencoder-based trust index method, designed to assess the credibility of ensemble predictions in real time. Before delving into the formulation and implementation of the trust index, let us discuss the concept of autoencoders, which form the foundational basis for this method. The following section provides a brief overview of autoencoders, outlining their structure, functionality, and relevance to the proposed approach.

8.1 Introduction to Autoencoders

An autoencoder is a feedforward neural network where the output is ideally the same as the input. It serves as a nonlinear dimensionality reduction tool by learning a compressed representation of the input data through reconstruction. Unlike conventional Principal Component Analysis (PCA), which performs linear dimensionality reduction by projecting data onto orthogonal principal components, autoencoders allow for nonlinear mappings between the input and output. As such, they can be viewed as a generalization of Kernel PCA, capable of capturing complex, nonlinear structures in the data.

Structurally, an autoencoder consists of two main components: an encoder and a decoder. The encoder is a function $f_\theta: \mathbb{R}^d \rightarrow \mathbb{R}^k$ that maps the input vector $x \in \mathbb{R}^d$ into a latent representation $z \in \mathbb{R}^k$ where k < d. The decoder is a function $g_\phi: \mathbb{R}^k \rightarrow \mathbb{R}^d$ that reconstructs the input as $\hat{x} = g_\phi(f_\theta(x))$.

The network is trained to minimize the reconstruction loss, typically defined as the mean squared error (MSE) between the original input and the reconstructed output

$$\mathcal{L}(\theta, \phi) = \frac{1}{N}\sum_1^N \left| x_i - g_\phi\left(f_\theta(x_i)\right)\right|^2 \tag{8.1}$$

Each neuron in the input and output layers represents one component of the multidimensional input/output. The number of neurons in the hidden layer(s) — often much smaller than the input dimension — enforces a bottleneck, encouraging the model to learn a compressed, information-rich representation.

Although the basic architecture includes a single hidden layer, deeper autoencoders with multiple hidden layers on both encoder and decoder sides are also common. These architectures enable the model to capture more abstract and hierarchical

features. The autoencoder approach is applicable to both static data and dynamic (time series) problems, making it a versatile technique for tasks such as anomaly detection, feature extraction, signal compression, and input familiarity assessment.

8.2 Development of Trust Index

In the context of the problem formulation presented in Chapter 7, the use of autoencoders is extended to model the input design space, consisting of 33 design variables. An autoencoder architecture is developed wherein the input layer corresponds directly to the dimensionality of these design variables. The encoder portion of the network begins with a fully connected dense layer comprising 64 neurons activated by the ReLU function, which serves to project the input into a higher-dimensional nonlinear feature space. This is followed by a bottleneck layer with 16 ReLU-activated neurons, which encodes the data into a compressed latent representation. This latent space is intended to capture the most informative patterns within the input data, effectively reducing redundancy and noise. The decoder mirrors the structure of the encoder, starting with a dense layer of 64 ReLU-activated neurons to reconstruct the latent representation, and ending with an output layer of 33 neurons using a linear activation function to restore the original input dimensions. The architecture of the autoencoder is shown in Fig.8.5. The autoencoder is trained using the Mean Squared Error (MSE) loss function and optimized with the Adam optimizer at a learning rate of 0.001. To mitigate overfitting, a validation split was utilized along with early stopping. Prior to training, all input features were standardized to have zero mean and unit variance using the scalar transformation.

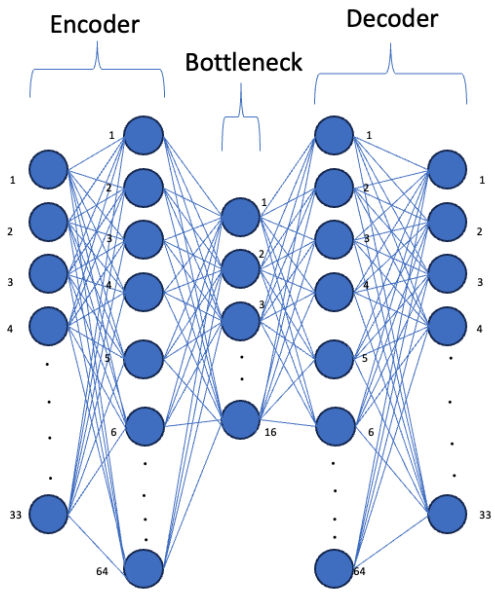

Figure 8.5 Autoencoder architecture

The autoencoder is trained using the same dataset introduced in Chapter 7, consisting of 650 design points. However, in this context, only the input data, comprising 33 design variables, is utilized for training. The objective of the autoencoder is to learn an efficient latent representation that allows accurate reconstruction of the original inputs. The training process minimizes the reconstruction error between the input and the decoded output, ensuring that the autoencoder captures the essential structure of the input space. Ideally, the decoded outputs should closely approximate the original inputs. Figures 8.6 to 8.8 present representative comparisons for 3 random cases (#633, #481, #124) from training set between the original input data and their corresponding reconstructions produced by the autoencoder. These results demonstrate that the

autoencoder is well-tuned and achieves a level of accuracy suitable for use in the proposed Trust Index framework.

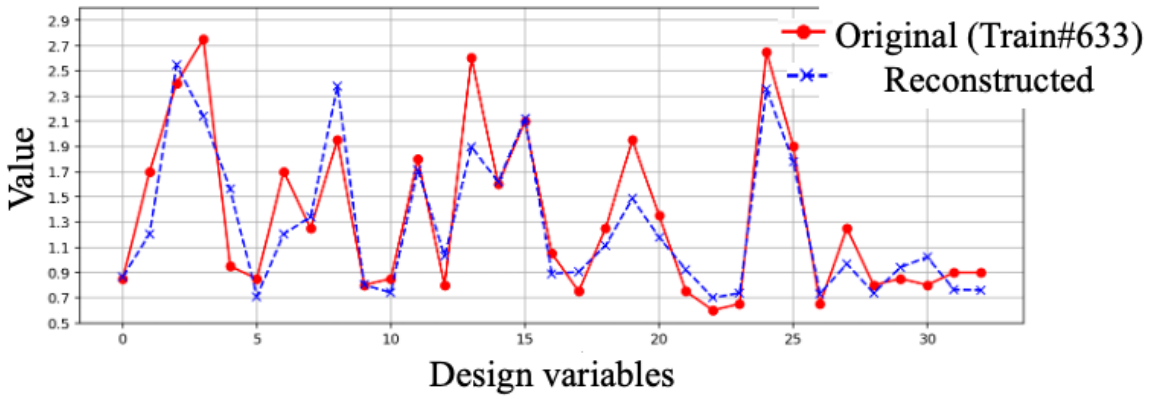

Figure 8.6 Autoencoder reconstruction and actual input for test 633

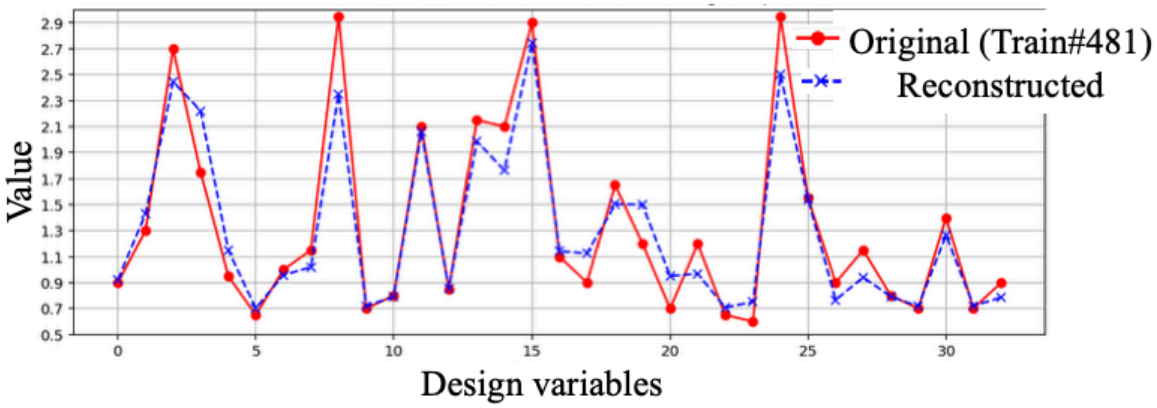

Figure 8.7 Autoencoder reconstruction and actual input for test 481

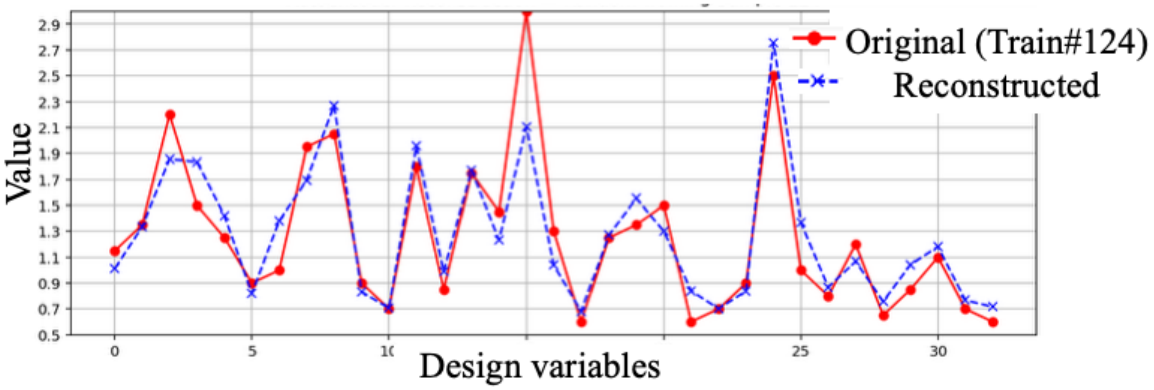

Figure 8.8 Autoencoder reconstruction and actual input for test 124

The trained autoencoder is now applied on unseen test points that were previously used for uncertainty quantification in Chapter 7. Specifically, Figures 8.9 and 8.10 illustrate the input reconstructions for test #1 and test #10, respectively. As discussed earlier, test #1 (also included in the evaluation) exhibits high-fidelity reconstruction, aligning closely with the original input vector. Notably, this corresponds with the observation that the uncertainty quantification for test #1 was highly accurate. In contrast, the reconstruction of test #10 reveals significant deviations from the actual input, as shown in Figure 8.10. This deviation correlates with reduced accuracy in uncertainty estimation for test #10. A plausible explanation is that the combination of input features in test #10 lies outside the distribution learned from the training set of 650 design points, indicating that the model has limited generalization to this unfamiliar input configuration.

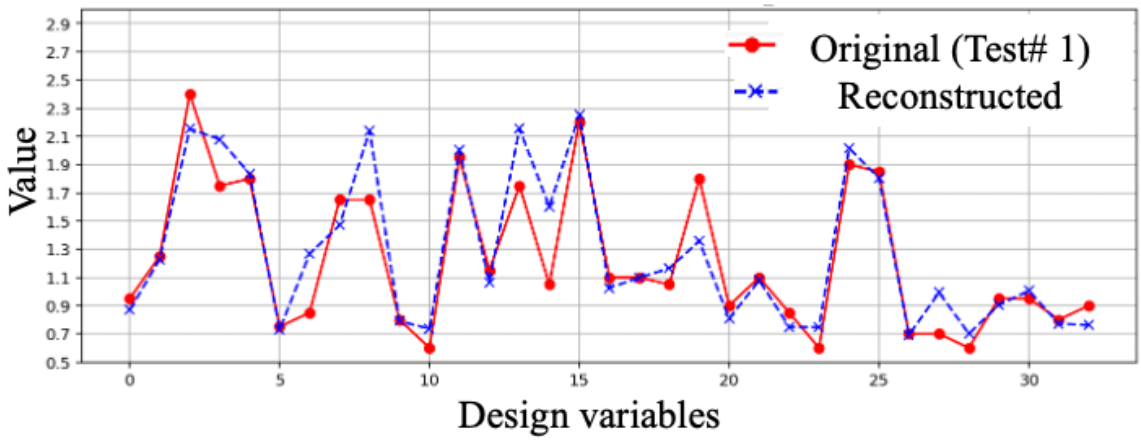

Figure 8.9 Reconstruction of test 1

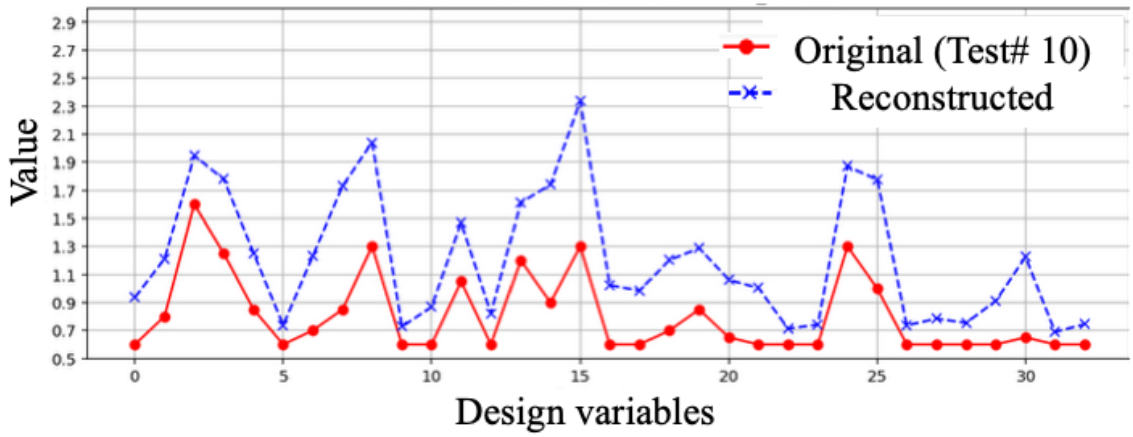

Figure 8.10 Reconstruction of test 10

Figure 8.11 presents a histogram of reconstruction errors (measured as Mean Squared Error, MSE) across all test samples. The distribution exhibits a bell-shaped curve, with a pronounced peak in the range of 0.25 to 0.35 MSE. The red dashed line represents the mean reconstruction error for the training data, which is approximately 0.32, further indicating that the model generalizes well to input patterns that are similar to those seen during training. These results confirm that the developed autoencoder

effectively compresses and reconstructs inputs that are consistent with the training distribution, thereby serving as a foundation for assessing the trustworthiness of model predictions in unfamiliar regions of the input space.

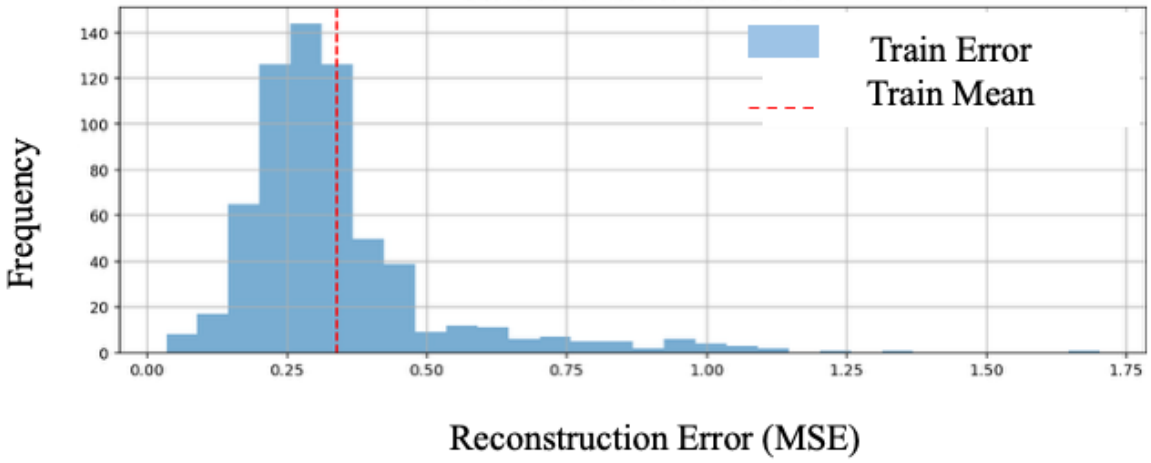

Figure 8.11 Histogram of *r*econstruction error

To systematically quantify the familiarity (similarity) of a new input sample, we propose a *Trust Index* derived from the reconstruction error of an autoencoder *trained solely on the input distribution*. The fundamental premise underpinning this approach is that an autoencoder, when exposed exclusively to a particular dataset during training, becomes adept at reconstructing inputs that lie within the learned distribution. Conversely, when presented with inputs that deviate significantly from the training distribution (i.e., out-of-distribution samples) the reconstruction error increases markedly. This behavior serves as the basis for defining the Trust Index so that inputs yielding low reconstruction error are deemed familiar and trustworthy, while those with high reconstruction error are considered novel or unfamiliar (not similar) in the context of the

model's prior exposure. The Trust Index thus offers a mechanism for evaluating the credibility of ensemble predictions and their associated uncertainty quantification in scenarios involving previously unseen data.

We define the Trust Index for a given input x and its prediction $\hat{x}$ as:

$$\text{Trust Index} = \frac{1}{1+\frac{1}{n}\sum_1^n(x_i-\hat{x}_i)^2} \tag{8.2}$$

where $\frac{1}{n}\sum_1^n(x_i - \hat{x_i})^2$ is the average MSE (Mean Square Error).

The proposed Trust Index offers an interpretable measure of input familiarity, based on the reconstruction error from an autoencoder trained on the *input* distribution. This index is designed to complement traditional output-based uncertainty metrics by focusing on whether the input itself falls within the domain of the model's prior experience. The approach possesses several desirable properties. First, it offers an interpretable scale. Inputs that produce reconstruction errors close to the training mean, yield Trust Index values near 1, indicating a high degree of familiarity. In contrast, inputs with significantly unfamiliar inputs produce values near zero. This makes the Trust Index directly useful as a confidence proxy.

The Trust Index is computationally efficient, requiring only a single forward pass through the autoencoder followed by a closed-form expression. It is also model-agnostic and fully unsupervised, requiring no labeled output data or task-specific fine-tuning. As such, it can be easily integrated into a broad range of machine learning predictors to identify potentially out-of-distribution inputs and improve the reliability of downstream predictions.

Using Equation (8.2), the Trust Index for all 11 unseen test inputs that are used for uncertainty estimation in Chapter 7 are calculated and presented in Table 8.1.

Table 8.1 Trust Index for test data

Test Data	Trust Index
0	0.5991
1	0.5730
2	0.5182
3	0.6233
4	0.6648
5	0.4420
6	0.5653
7	0.4181
8	0.4310
9	0.6604
10	0.3602

When applied across all eleven unseen test cases used for uncertainty quantification, it was observed that the predictive uncertainty estimates for Test Cases 5 (Figure 8.4) , and 8 (Figure 8.5) were noticeably less accurate compared to the remaining cases. This discrepancy can be attributed to the corresponding Trust Index values, which were relatively low for these cases.

Figures 8.12 and 8.13 highlight the low accuracy of uncertainty quantification for test #8 and test #10 which exhibit the lowest trust index values (0.431 and 0.362, respectively) among the 11 cases of Table 8.1. The reduced Trust Index indicates that the combinations of input variables in these test cases were not well represented within the original design space defined by the training data. In other words, the autoencoder

identified these inputs as being less familiar, thereby signaling limited accuracy of the associated uncertainty estimates.

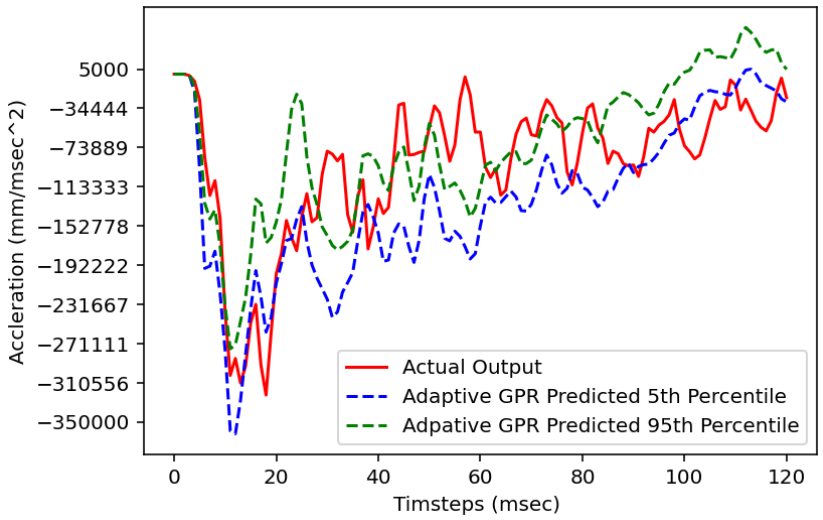

Figure 8.12 Uncertainty quantification for test 8 (trust index 0.431)

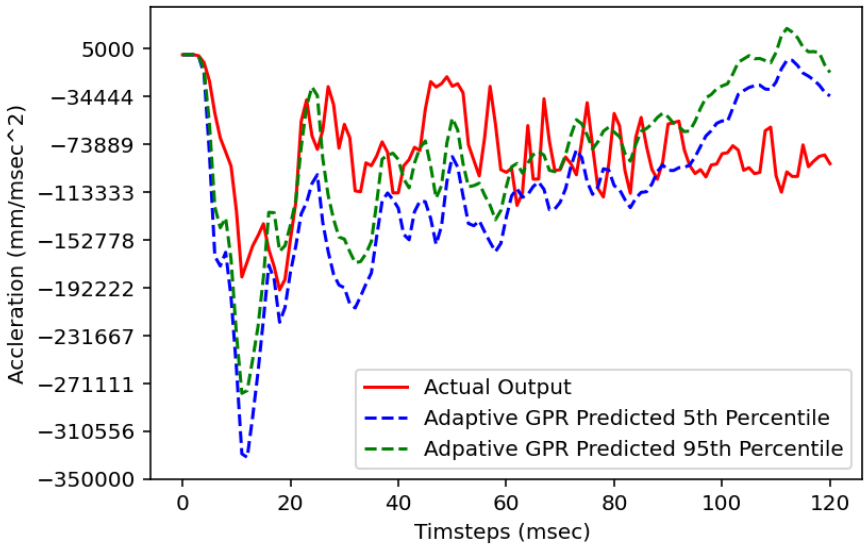

Figure 8.13 Uncertainty quantification for test 10 (trust index 0.362)

The proposed Trust Index framework is generalizable and can be applied to any modeling scenario. The threshold or cutoff value for determining “low trust” is not universal and should be calibrated using appropriate validation datasets for each specific application. Nevertheless, a guiding principle is that a higher Trust Index corresponds to greater confidence in the accuracy of the uncertainty quantification. It is important to emphasize that the Trust Index should be interpreted as a cautionary indicator rather than an absolute measure of predictive confidence. It offers a valuable warning mechanism to flag potentially out-of-distribution inputs and inform users when uncertainty estimates may be inaccurate, thereby enhancing the interpretability and robustness of data-driven modeling frameworks.

CHAPTER 9

MEAN OF ENSEMBLES USING A MOST PROBABALE MODEL

This section proposes an approach called the Most Probable Model approach to estimate the mean of ensemble models without using ensembles during the inference phase. By leveraging ensemble learning, the method identifies the most frequently occurring network parameters, leading to a robust and computationally efficient predictive framework.

To explain the approach, we first consider the following representative function

$$f(s) = 3x(s) - 5y(s) + z(s) - 6 \tag{9.1}$$

where *x(s)*, *y(s)*, and *z(s)* represent independent realizations of random processes X(s), Y(s) and Z(s) respectively, defined over a spatial domain $s \in [S_{min}, S_{max}]$. The goal is to characterize the probabilistic behavior of *f(s)* using the ensemble-based approach and to compare different approaches for estimating the most probable value (mode) of the response at each space location.

Each underlying process (X(s), Y(s), and Z(s)) is assumed stationary characterized by its power spectral density (PSD) which is chosen to reflect typical engineering random processes encountered in road surface stochastic modeling. The used PSD is parametrized using the variance (σ^2), a spatial or temporal scaling coefficient (α), and a characteristic velocity or correlation length (v), in line with standards ISO 8608 for random road profiles (Table 9.1).

Table 9.1 PSD parameters for X(s), Y(s) and Z(s)

Quantity	Description	Value
σ^2	Variance of Road Profile	0.001 m^2
α	Road Profile Coefficient	0.3375 rad/m
v	Vehicle Speed	60 km/h

The Karhunen-Loeve Expansion (KLE) method is used to generate realizations of each random process that follow the given Power Spectral Density (PSD). The first step involves calculating its autocorrelation function using the inverse Fourier transform of the PSD. The autocorrelation function is then used to build a covariance matrix that represents the correlation structure of the discretized process. By performing eigen decomposition of the covariance matrix, a set of orthogonal eigenfunctions (modes) and their corresponding eigenvalues are obtained. These eigenfunctions act as basis functions, and the random process is expressed as a weighted sum of them, where the weights are uncorrelated standard normal random variables. To keep the model efficient while still capturing the main features of the process, the expansion is truncated by including only the modes whose eigenvalues contribute significantly to the total energy. This results in a compact yet accurate representation of the stochastic process. A large ensemble (e.g., 500 realizations) of each process is generated by sampling independent standard normal variables for each retained mode and forming the weighted sum over the grid. The ensemble of all generated realizations is used for all subsequent probabilistic analyses. Figure 9.1 shows a few representative realizations of $X(s)$, $Y(s)$ and $Z(s)$ and Figure 9.2 shows a representative ensemble of function $f(s)$ according to Equation (9.1).

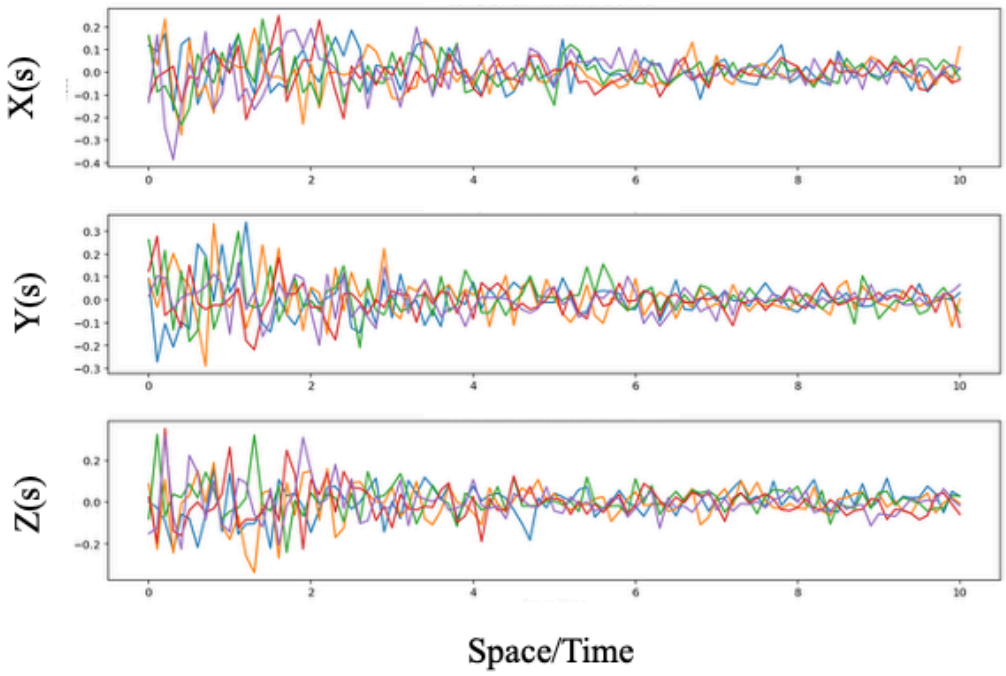

Figure 9.1 Sample realizations (out of 500) of X(s), Y(s) and Z(s)

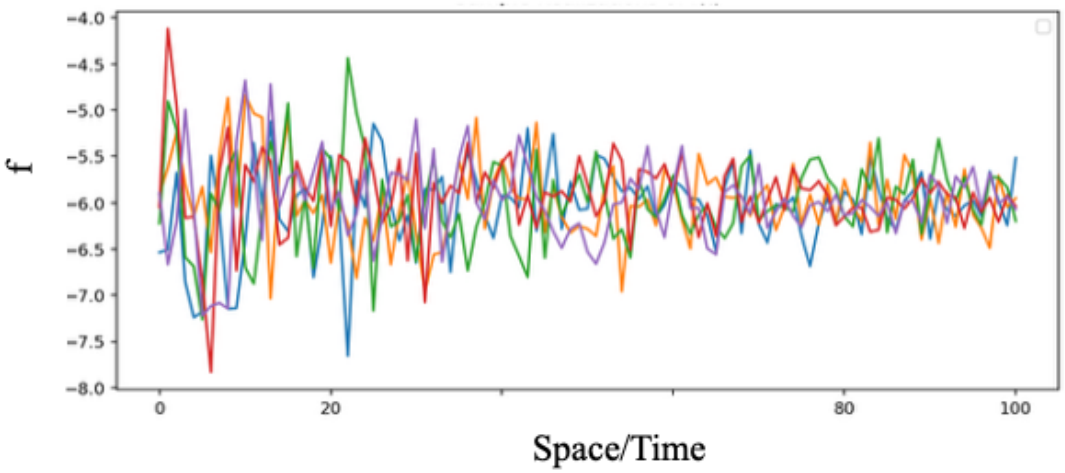

Figure 9.2 Representative ensemble of function *f(s)*

Equation (9.2) shows the value of *f(s)* evaluated at the most probable (frequent) values of X(s), Y(s), and Z(s).

$$f_{MPM}(s) = 3x_{MPM}(s) - 5y_{MPM}(s) + z_{MPM}(s) \; - 6 \tag{9.2}$$

where $x_{MPM}(s)$, $y_{MPM}(s)$, $z_{MPM}(s)$ are the histogram modes of X(s), Y(s), and Z(s) obtained by taking the mean of the highest bin (most probable bin) at each location *s* (mean of random variable at location *s*) of the respective input process.

To assess the accuracy of the MPM approach, both the empirical mode of *f(s)* at each location *s* (obtained directly from the histogram of the ensemble), and the Most Probable Model are computed at each grid location as shown in Figure 9.3. The plot shows that the MPM approach aligns well with the actual mode derived from the ensemble histogram of *f(s)* as evidenced by the close tracking of the blue and red curves. This indicates that the MPM method can effectively represent the actual mode of the function across the domain, capturing the dominant trends in the most probable response.

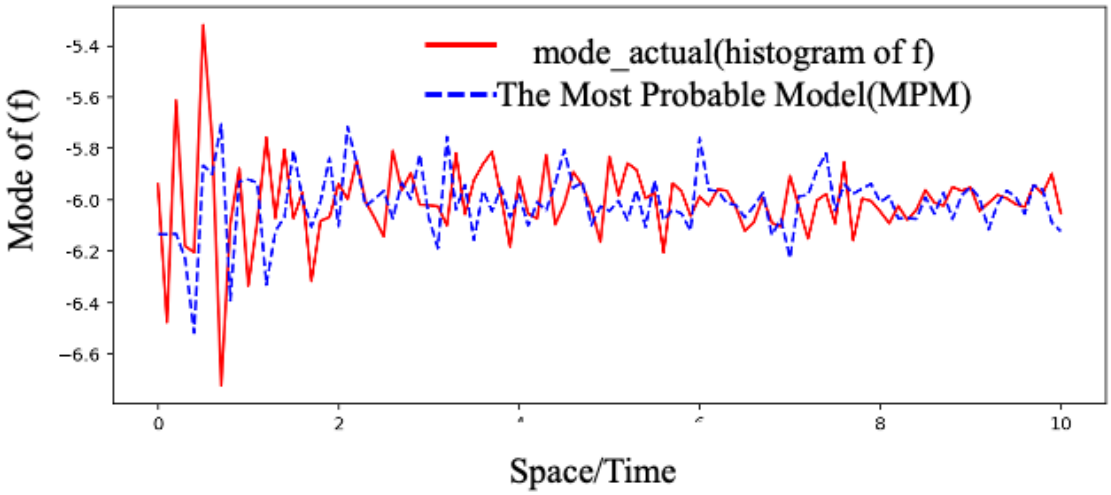

Figure 9.3 Comparison between the MPM of *f(s)* and the actual modes of *f(s)*

9.1 Application of Most Probable Model (MPM) Approach

We consider here the example of feed forward neural network architecture described in Chapter 4 and outlined in Figure 9.4.

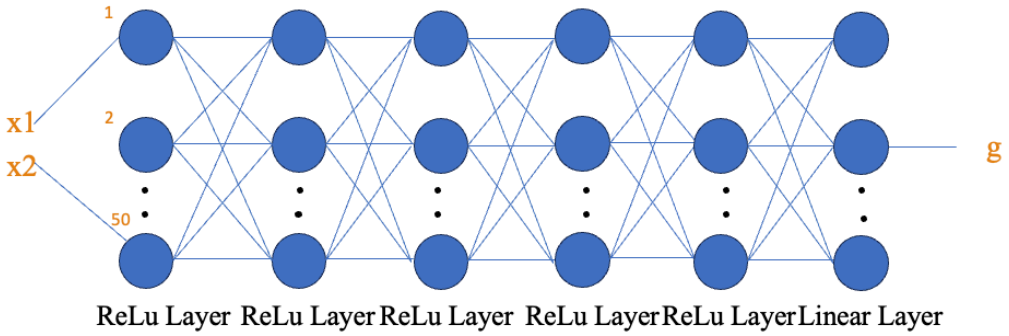

Figure 9.4 A sample neural network model

The network has an input layer and output layer along with 4 hidden layers. Each layer has 50 neurons. We consider Layer 2 for illustration. Considering the weight distribution for all connections in this layer, we identify the most common weights using a histogram-based approach. Layer 2 connects 50 input features to 50 output neurons, resulting in a total of ($50 \times 50 = 2500$) unique connections which play a pivotal role in transforming learned features from earlier layers into representations that the deeper network layers or final outputs can utilize. When training an ensemble of 50 models with the same architecture, each connection in Layer 2 learns slightly different weight values due to variations in initialization, data shuffling, and optimization trajectories.

To understand the shared characteristics across the ensemble, we compute the “most probable weight” for each connection in Layer 0. For an ensemble of 50 models, the

weights for Layer 2 can be represented as a 3D tensor with dimensions ($50 \times 50 \times 50$). The entry tensor [m, i, j] of the tensor represents the weight of the connection between input "i" and neuron "j" for model "m". For example, the connection between input 5 and output neuron 5 will have 50 weight values across the ensemble. These weights might resemble the following distribution -0.17, -0.16, -0.15, -0.14, ...,0.14,0.15]. The variation reflects differences in how each model in the ensemble optimizes this specific connection. The weight distribution for each connection across the ensemble is represented by a histogram. For a specific connection, such as (5, 5), we have a histogram of the 50 weight values. The histogram divides the range of weights into discrete bins and counts the number of weights falling into each bin. The bin with the highest count represents the range where the most weights are concentrated. The center of this bin is considered the *most probable weight* for the connection. For instance, the histogram for the connection (5, 5) in Layer 2 indicates how many of the 50 weights fall into each bin. The method ensures a robust summary of the weight distributions across the ensemble, capturing the most representative weight for each connection in Layer 2. Figure 9.5 shows a representative histogram. the bin with the most frequent weights is highlighted in red. This bin contains 5 weights, making it the most frequent bin. Thus, the most probable weight for the connection (5,5) is -0.109. This value represents the weight that occurred most frequently across the ensemble of models for this specific connection.

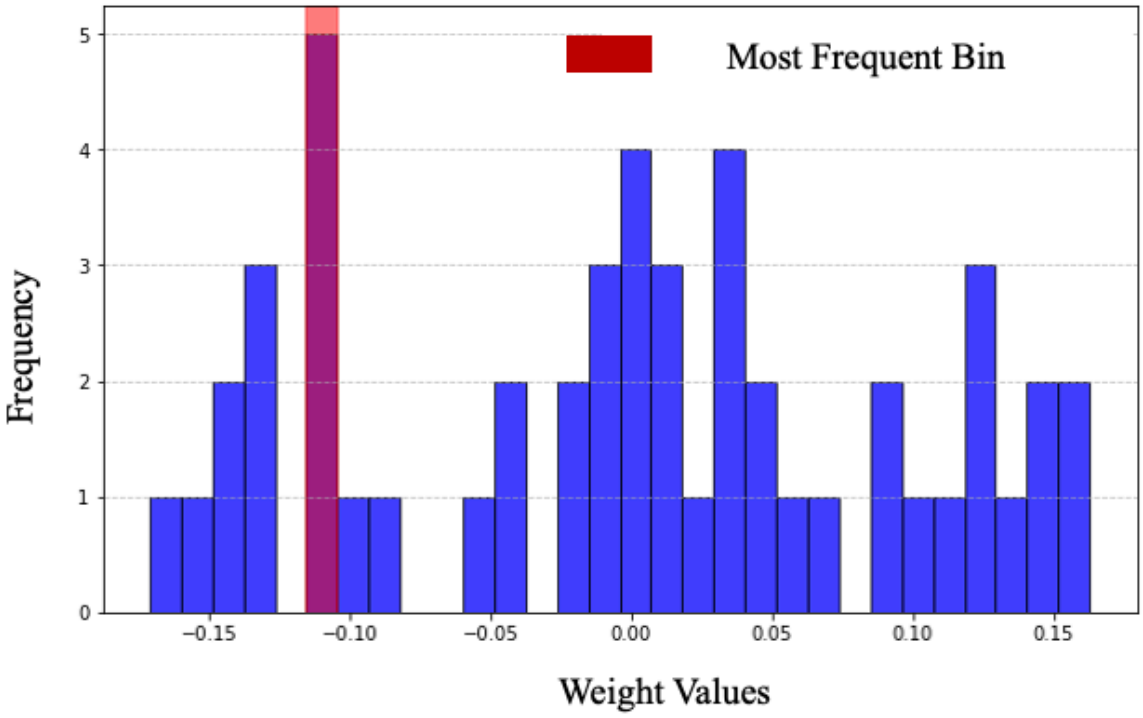

Figure 9.5 Representative histogram of weights for a hidden layer

The proposed method was applied and tested on an unknown test point to evaluate its performance. To compare the effectiveness of the method, an ensemble of predictions was utilized to identify the most probable weights, and the mean of the ensemble predictions was calculated. The results are presented in Figure 9.6 , which compares the actual value (0.578) with the mean ensemble prediction (0.653) and the prediction by the most probable model (0.657). Figure 9.7 illustrates the histogram of the ensemble predictions, highlighting the mean of the ensemble and the prediction from the most probable model. The close agreement between the mean of the ensemble prediction and the prediction by the most probable model demonstrates the accuracy of the proposed method in representing the mean of the ensemble.

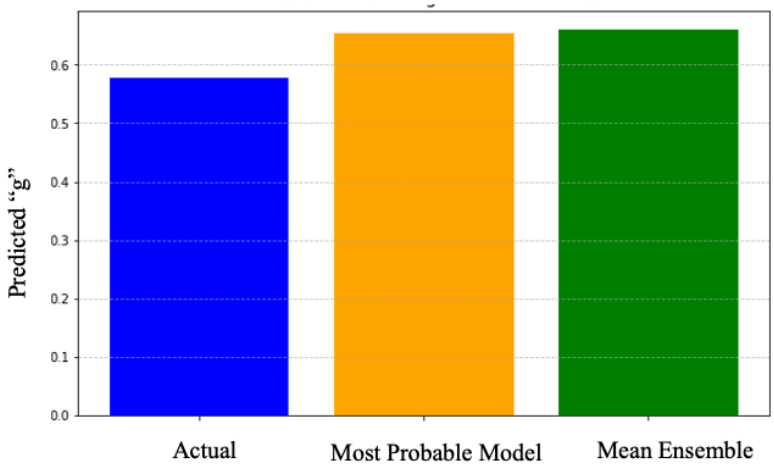

Figure 9.6 Comparison of actual test case with prediction by MPM and mean ensemble

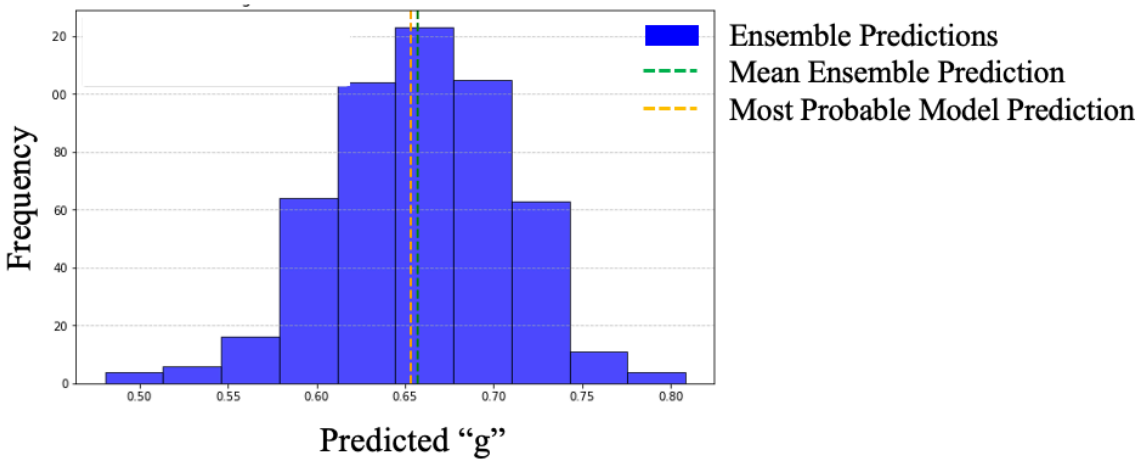

Figure 9.7 Histogram of ensemble prediction with mean ensemble prediction and MPM

9.2 Validation of MPM Approach Using an Automotive Crash Example

In this study, we revisit the automotive crash simulation framework of Chapter 7. However, instead of using the thickness variations of 33 individual structural components as the input design variables, we consider output acceleration curves at a specific location on the vehicle. Specifically, the first half of the acceleration time history (captured up to 60 msec) is utilized as the input, while the latter half of the response (60 to 120 msec)

serves as the output. Figure 9.8 shows the used 662 acceleration curves providing input and output data.

The objective now is to predict the second half of a curve given the acceleration response of the first 60 msec. For unseen data, the crash simulation is thus run up to 60 msec and the rest is predicted by the ML model.

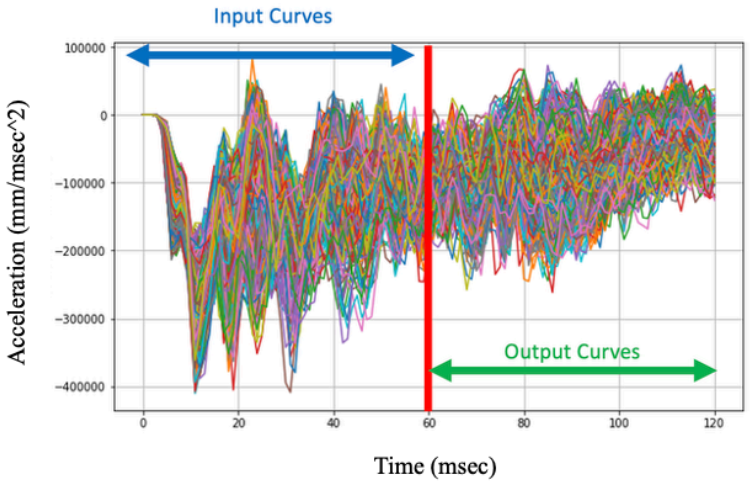

Figure 9.8 Acceleration response split in half for input and output data

For this problem, an autoencoder approach is adopted. Separate input and output autoencoders are first trained for the input data (0-60 msec) and the output data (60-120 msec). Each autoencoder consists of one or more hidden layers, with the number of neurons in each hidden layer automatically determined. The autoencoders provide data compression and are trained using the provided input and output data. The number of hidden neurons in the output autoencoder is typically set to be the same as in the input autoencoder. A mapping model, such as a feedforward neural network, is trained to establish a relationship between the encoded inputs and encoded outputs. The input layer

consists of the encoded representations obtained from the input autoencoder, while the output layer represents the encoded outputs from the output autoencoder. This approach ensures that meaningful latent representations are effectively mapped, enabling robust predictions while reducing the dimensional complexity of the original data. The process is outlined in Figure 9.9.

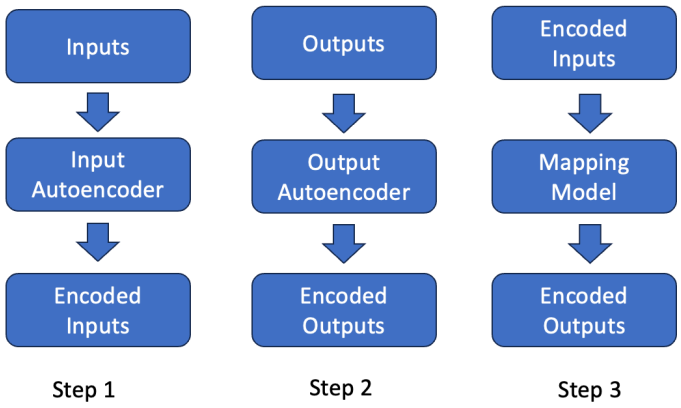

Figure 9.9 An autoencoder-based ML model

Of the total 662 curves, 650 input curves (acceleration response from 0-60 msec) and output curves (acceleration response from 60-120 msec) were used for training. The input and output autoencoders encode the input and output curves, respectively. The latent space dimensions for both autoencoders were set to 32, providing a balance between compression and information retention. While there is potential for further dimensionality reduction, 32 dimensions were found to be sufficient for this example. To map the relationship between the encoded inputs and outputs, a feedforward neural

network is used as the mapping model. The network architecture for the autoencoders is provided in Table 9.2 while that of mapping model is shown in Figure 9.10.

Table 9.2 Autoencoder architecture

	Input Autoencoder	**Output Autoencoder**
Latent Layer Dimension	32	32
Latent Layer Activation Function	Relu	Relu
Output Layer Activation Function	Sigmoid	Sigmoid

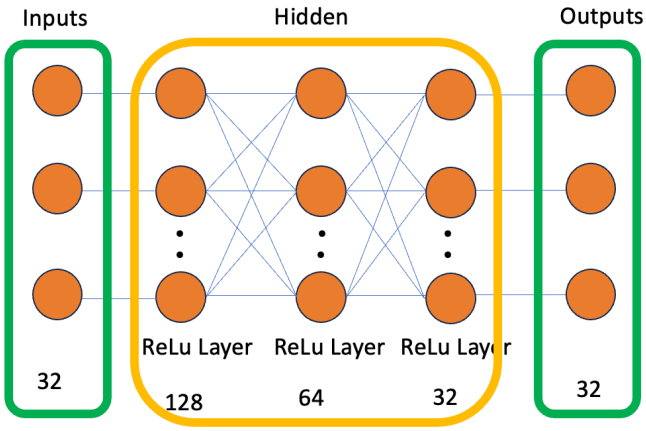

Figure 9.10 Feedforward neural network used as mapping model

An ensemble of input autoencoders was created by training the same network architecture of Table 9.1 100 times and a Most Probable Model (MPM) was developed using all 100 input autoencoders using the procedure of Section 9.2. Similarly, an ensemble of output autoencoders was created with 100 models and the MPM of all output autoencoders was developed.

Using the MPMs of the input and output autoencoders an ensemble of 100 mapping models was developed which were then used to obtain a MPM of the mapping model. We thus obtained three Most Probable Models: one for inputs, one for outputs, and one for the mapping between them. The performance of the models on unseen data is illustrated in Figure 9.11. It is important to note that, given the use of three separate neural network models, the MPM approach is the most efficient. By identifying the most frequent and stable parameter configurations across multiple training runs, this method ensures consistency in predictions while avoiding the computational overhead of the ensemble learning.

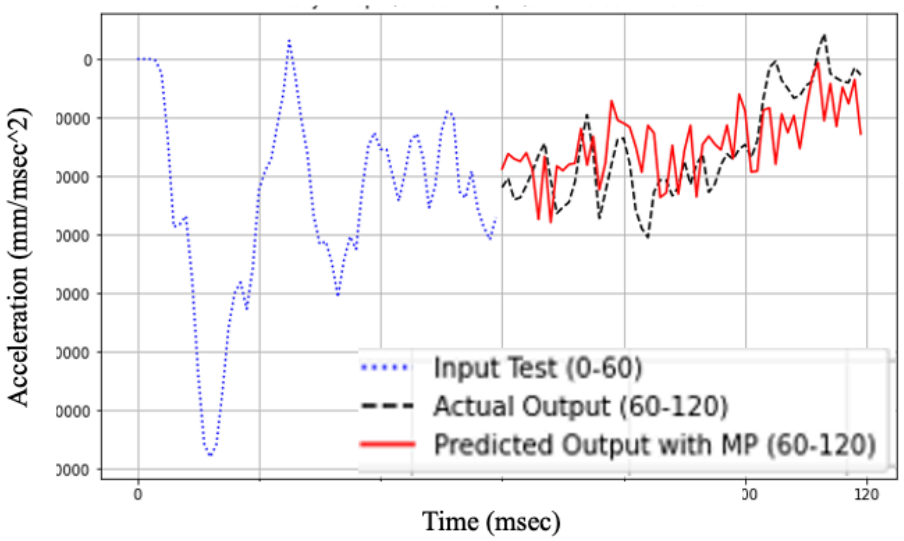

Figure 9.11 Comparison of actual output vs prediction by the MPM

As another illustration, the Most Probable Model approach was also applied to the problem described in Chapter 7. Figure 9.12 compares the prediction by the MPM approach for unseen data point along with mean ensemble prediction and the actual

output. The MPM prediction closely corresponds to the mean of the ensemble indicating its efficiency over the ensemble method, as the latter runs multiple models in order to calculate their mean. In contrast, the MPM approach requires only one model run for prediction.

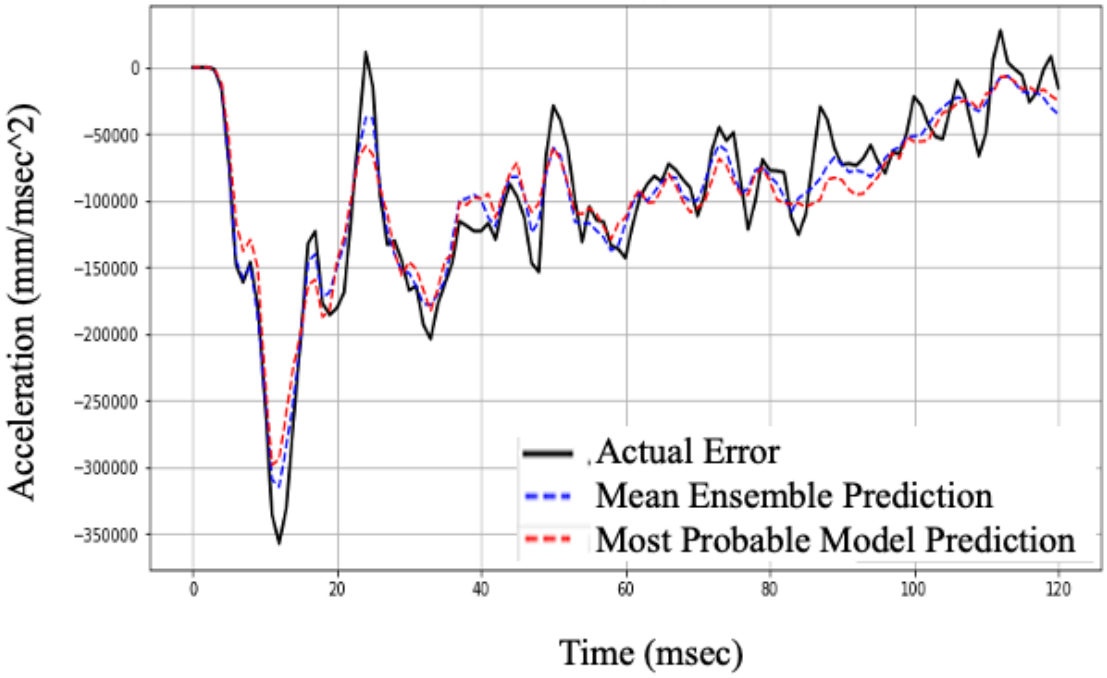

Figure 9.12 Comparison of the MPM prediction , the mean ensemble and actual output

CHAPTER 10

SYNTHETIC DATA GENERATION USING VARIATIONAL AUTOENCODERS

This dissertation primarily addresses the challenge of model uncertainty in machine learning, with a particular focus on ensemble-based approaches. Traditional ensemble methods require the training and deployment of multiple models to quantify prediction uncertainty, which, while effective, can be computationally expensive and resource intensive. Chapter 5 introduced a novel methodology to efficiently estimate uncertainty bounds by utilizing only two models (the 5th and 95th percentile Adaptive Gaussian Process Regression (GPR) models), instead of relying on a full ensemble of predictions. This substantially reduces the computational burden typically associated with ensemble methods during inference step, while still providing meaningful measures of uncertainty. In Chapter 9, the "Most Probable Model" approach was presented, which further enhances efficiency by offering reliable mean predictions derived from the ensemble, circumventing the need to evaluate all models for every prediction.

Despite these advancements, all ensemble-based methods and the proposed extensions ultimately depend on the initial training of multiple machine learning models. This initial stage, albeit performed only once, remains a significant bottleneck in terms of computational time and resource consumption. Accordingly, there is a compelling need to develop approaches that maintain the advantages of ensemble-based uncertainty quantification while improving efficiency. To address this, this chapter proposes a novel methodology based on Variational Autoencoders (VAE). This approach enables the

generation of multiple plausible predictions by leveraging a small subset of ensemble models, thereby significantly reducing the computational demand without compromising the quality of uncertainty quantification.

10.1 Review of Variational Autoencoders (VAE)

A Variational Autoencoder (VAE) is a class of deep generative models designed to learn probabilistic representation of complex datasets such as images, signals, or sequences by mapping them into a structured, low-dimensional latent space. In contrast to the deterministic autoencoders described in Chapter 8, VAEs introduce a probabilistic framework that enables both reconstruction of observed data and generation of new samples, making them particularly suitable for unsupervised learning and generative modeling tasks.

The structure of a Variational Autoencoder (VAE) is schematically illustrated in Figure 10.1 which provides a visual representation of the model's architecture and information flow. The VAE consists of three main stages: the encoder, the latent space, and the decoder. The process begins with the observed input data, denoted by **y**, which is mapped by the encoder into a latent space **z**. The encoder, represented as $q_{\emptyset}(\boldsymbol{z}|\mathbf{y})$ is a probabilistic model that approximates the posterior distribution over the latent variable **z** given the observed input. This distribution is typically assumed to be Gaussian, with its parameters (mean and variance) learned from the data. The triangular shape in the figure reflects the compression operation, reducing the dimensionality of the data and enforcing a bottleneck that captures only the most salient features. The middle block, labeled **z** represents the latent space where the input is encoded into a lower-dimensional,

structured representation. This latent space is regularized to follow a prior distribution $p(\mathbf{z}) = \mathcal{N}(0, I)$ ensuring continuity and smoothness that are essential for generative tasks. Following this, the decoder, depicted by $p_\theta(\mathbf{y}|\mathbf{z})$, maps the latent variable $\mathbf{z}$ back into the data space, reconstructing the output $\hat{\boldsymbol{y}}$. The decoder mirrors the encoder's structure, gradually expanding the compressed representation to produce a sample that ideally resembles the original input. The objective during training is to minimize the difference between $\mathbf{y}$ and $\hat{\boldsymbol{y}}$ while simultaneously encouraging the latent space $\mathbf{z}$ to follow the prescribed prior distribution. This encoder–latent–decoder pipeline, as shown in Figure 10.1, captures the core operational flow of a VAE which is to encode high-dimensional data into a compressed latent form, sample from the latent distribution, and decode back to reconstruct the original input.

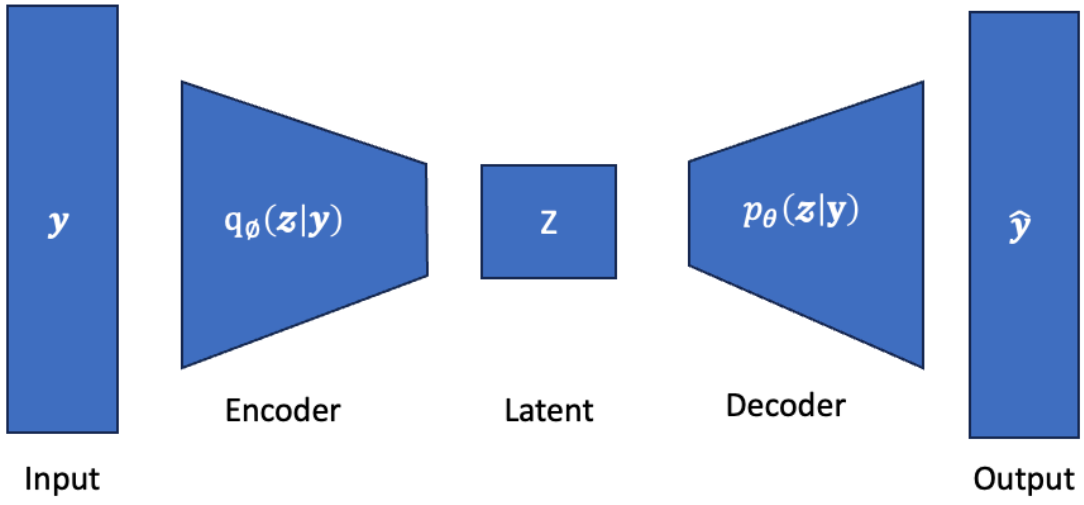

Figure 10.1 Variational autoencoder information flow

The VAE is trained to maximize the likelihood of the observed data under the generative model. However, direct computation of the marginal likelihood $\mathrm{p}_\theta(\mathbf{y})$ is generally intractable due to the integration over the latent variables. Instead, the model is trained by maximizing the variational lower bound of Equation 10.1, known as the Evidence Lower Bound (ELBO), which serves as a surrogate objective.

$$\log \mathrm{p}_\theta(\mathbf{y}) \geq \mathbb{E}\mathrm{q}_\emptyset(\boldsymbol{z}|\mathbf{y})[\log \mathrm{p}_\theta(\mathbf{y}|\mathbf{z})] - D_{KL}(\mathrm{q}_\emptyset(\boldsymbol{z}|\mathbf{y}))||\mathrm{p}_\theta(\boldsymbol{z}|\mathbf{y}) \tag{10.1}$$

The first term in Equation (10.1) is the expected log-likelihood of the reconstruction, encouraging the decoder to generate outputs that closely match the inputs. The second term is the Kullback–Leibler (KL) divergence between the approximate posterior $\mathrm{q}_\emptyset(\boldsymbol{z}|\mathbf{y})$ and the prior $\mathrm{p}_\theta(\mathbf{y}|\mathbf{z})$, which acts as a regularizer that constrains the latent representation to remain close to the prior distribution, typically chosen as $\mathcal{N}(0, \mathrm{I})$. For the commonly used Gaussian latent distribution with diagonal covariance, the KL divergence term admits a closed-form expression as

$$D_{KL}(\mathrm{q}_\emptyset(\boldsymbol{z}|\mathbf{y}))||\mathrm{p}_\theta(\boldsymbol{z}|\mathbf{y}) = \frac{1}{2}\sum_{\mathrm{j}=1}^{\mathrm{d}}(\mu_\mathrm{j}^2 + \sigma_\mathrm{j}^2 - \log(\sigma_\mathrm{j}^2) - 1) \tag{10.2}$$

where μ_j, σ_j denote the mean and standard deviation of the decoder. The D_{KL} divergence is a measure of how the learned distribution $\mathrm{q}_\emptyset(\boldsymbol{z}|\mathbf{y})$ (i.e. the encoder's output) differs from the prior distribution $\mathrm{p}(\boldsymbol{z})$. Penalizing KL encourages the encoder's output to stay close to the simple prior, making the latent space well-behaved for sampling and interpolation. Figure 10.2 provides a graphical illustration of the KL Divergence and ELBO.

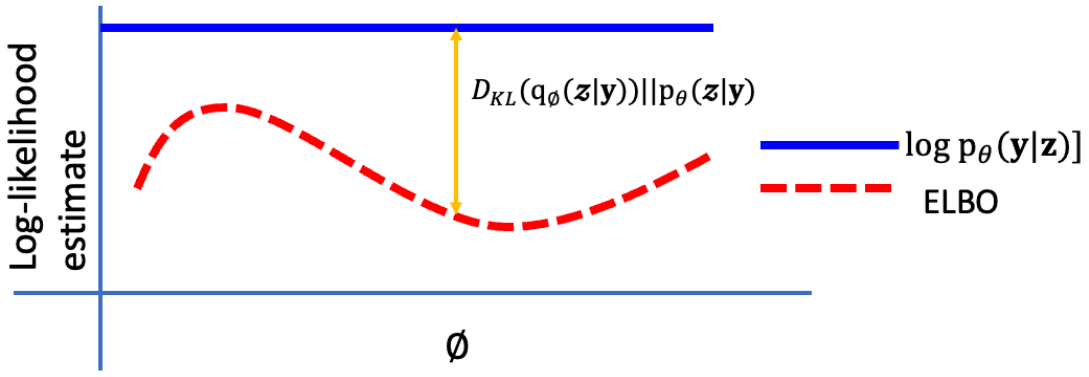

Figure 10.2 Illustration of KL Divergence and ELBO

To allow backpropagation using a stochastic model, the VAE uses

$$z = \mu(y) + \sigma(y).\epsilon \tag{10.3}$$

where $\epsilon \sim \mathcal{N}(0, I)$. This allows stochastic sampling while using the conventional backpropagation algorithm.

A central challenge in training VAEs is that sampling from a probability distribution (such as $q_\phi(z|y)$) is inherently a non-differentiable operation, making it difficult to update the encoder's parameters using gradient-based optimization. The reparameterization trick solves this by expressing the random variable z as a deterministic function of the encoder's outputs and an independent source of noise. Specifically, instead of sampling z directly from $(\mathcal{N}(\mu(y), \sigma^2(y)))$, we sample an auxiliary variable ϵ from a standard normal distribution $\mathcal{N}(0, I)$ and compute Equation (10.3). This simple transformation makes the sampling process differentiable with respect to the encoder's parameters, enabling end-to-end training of the entire VAE model using the standard backpropagation. The reparameterization trick is a key innovation that

allows VAEs to be trained efficiently and has made probabilistic deep generative modeling practical.

During VAE training, all inputs **y** is first passed through the encoder, which outputs the mean and variance parameters $\mu(\mathbf{y})$ and $\sigma(\mathbf{y})$ for the approximate posterior distribution $q_\emptyset(\boldsymbol{z}|\mathbf{y})$. Using these parameters, a latent variable z is sampled via the reparameterization trick to allow gradients to flow through the sampling process. This sampled z is then passed through the decoder to produce a reconstruction $\hat{y}$ of the original input. The model's objective, known as the Evidence Lower Bound (ELBO), combines a reconstruction loss (measuring how well $\hat{\mathbf{y}}$ matches **y**) and a KL divergence loss which regularizes the learned latent distribution towards the prior. Finally, the encoder and decoder parameters are updated via gradient descent to maximize the ELBO, thereby improving both reconstruction fidelity and latent space regularity.

10.2 Conditional VAE (C-VAE)

In earlier sections, the Variational Autoencoder (VAE) was introduced as a generative model capable of learning the distribution of high-dimensional outputs (such as time-series curves) by mapping them into a lower-dimensional latent space. The latent space is regularized through the Kullback-Leibler (KL) divergence, which encourages the encoded representations to follow a standard normal distribution, facilitating interpolation and sampling. While the VAE offers substantial benefits in terms of computational efficiency and generative flexibility, it is inherently limited in one crucial aspect: it learns a marginal distribution, p(**y**), over the outputs without explicitly accounting for the corresponding inputs or design variables, **x**. Consequently, the generative process in a

standard VAE is unconditional. That is, although one can sample diverse outputs, there is no mechanism to condition these outputs on a specific input vector. This becomes problematic in engineering design and simulation contexts, where the relationship between input design parameters and system response is deterministic or at least input dependent.

In the context of this research, each training instance is characterized by design variables $\boldsymbol{x} \in \mathbb{R}^{\boldsymbol{d}}$, and the output is an associated time-series $\boldsymbol{y} \in \mathbb{R}^{\boldsymbol{T}}$. The goal is not merely to sample plausible outputs from the overall output distribution, but rather to model the conditional distribution $p(\boldsymbol{y}|\boldsymbol{x})$ that is, the distribution of outputs given a specific input configuration. The VAE lacks the structural capacity to make such a distinction.

To address this limitation, the Conditional Variational Autoencoder (CVAE) is used. The CVAE extends the VAE architecture by conditioning both the encoder and decoder on the input vector **x**. Specifically, the encoder now models the posterior $\mathrm{q}_{\emptyset}(\boldsymbol{z}|\mathbf{y},\mathbf{x})$, and the decoder reconstructs the output through $\mathrm{p}_{\theta}(\mathbf{y}|\boldsymbol{z},\mathbf{x})$. This conditioning allows the latent space to be structured in a way that reflects input-dependent variations, thereby enabling the generation of multiple plausible outputs for a given input. The remainder of this chapter details the architecture, training process, and validation of the CVAE model tailored for time-series response prediction in the context of simulation-driven design.

The CVAE assumes the following generative process

$$\boldsymbol{z} \sim p(\boldsymbol{z}) = \mathcal{N}(0, \mathbf{I}) \tag{10.4}$$

$$\boldsymbol{y} \sim p_\theta(\boldsymbol{y}|\mathbf{z}, \mathbf{x}) \tag{10.5}$$

Since the true posterior $p(\boldsymbol{z}|\boldsymbol{y}, \boldsymbol{x})$is intractable, we introduce the variational approximation

$$\mathrm{q}_\emptyset(\boldsymbol{z}|\mathbf{y}, \mathbf{x}) = \mathcal{N}(\mu_\theta(\boldsymbol{y}, \boldsymbol{x}), \mathrm{diag}(\sigma^2\emptyset(\mathbf{y}, \mathbf{x}))) \tag{10.6}$$

The model is trained by maximizing the conditional Evidence Lower Bound (ELBO) given as

$$\log \mathrm{p}_\theta(\mathbf{y}|\mathbf{x}) \geq \mathbb{E}\mathrm{q}_\emptyset(\boldsymbol{z}|\mathbf{y}, \mathrm{x})[\log \mathrm{p}_\theta(\mathbf{y}|\mathbf{z}, \mathbf{x})] - D_{KL}(\mathrm{q}_\emptyset(\boldsymbol{z}|\mathbf{y}, \mathbf{x}))||\mathrm{p}_\theta(\boldsymbol{z}|\mathbf{y}, \mathbf{x}) \tag{10.7}$$

The loss function in practice is implemented as

$$\mathcal{L}CVAE = ||\boldsymbol{y} - \widehat{\boldsymbol{y}}|| + \beta . D_{KL}(\mathcal{N}\left(\mu_\theta, \sigma^2\emptyset\right)||\mathcal{N}(0, \mathbf{I})\big) \tag{10.8}$$

where β is user-defined weighting factor.

10.3 Uncertainty Quantification using C-VAE

This section presents the potential use of a Variational Autoencoder (VAE) framework for uncertainty quantification using the crash simulation problem discussed in Chapter 7. This case study involves a frontal vehicle crash model characterized by 33 input design variables across 650 design points. The response of interest is a time-series signal recorded at the toe pan location, comprising 121 discrete time steps (see Figure 7.5 for illustration). To capture the predictive uncertainty, an ensemble of 25 deep neural networks is independently trained, each employing the same architecture as detailed in

Table 7.1. For each of the 650 design points, predictions are obtained from all 25 models, resulting in an ensemble of 650 × 25 = 16,250 output curves. These curves are stacked into a two-dimensional array of shape (16,250 × 121), representing the empirical distribution of ensemble predictions. This array forms the training dataset for the VAE, which aims to learn a low-dimensional latent representation of the ensemble variability.

The Conditional Variational Autoencoder (CVAE) developed in this work is designed to model the conditional distribution of time-series responses given a set of design inputs. The architecture consists of a fully connected encoder and decoder network, both explicitly conditioned on the design vector. The encoder takes as input the concatenation of a normalized output curve (with 121-time steps) and its corresponding input design vector (of dimension 33), resulting in a combined input of dimension 154. This input is passed through two hidden layers: the first dense layer contains 128 ReLU-activated units, followed by a second dense layer with 64 ReLU-activated units. The output of the second hidden layer is then branched into two separate dense layers of size 32, producing the latent mean vector and log variance vector, respectively, which parameterize a multivariate Gaussian posterior in latent space of dimension 32. A sampling layer implements the reparameterization trick to ensure differentiability of the latent space sampling process. During each forward pass, this layer samples a latent vector from the Gaussian distribution defined by the mean and log-variance vectors output by the encoder.

The decoder network mirrors the conditioning structure of the encoder. It takes as input the sampled latent vector (dimension 32) concatenated with the same design vector (dimension 33), resulting in a 65-dimensional input. This input is passed through two

hidden dense layers, with 128 and 64 ReLU-activated units respectively, followed by a final output layer with 121 units to reconstruct the original time-series response. The reconstruction loss is defined as the mean squared error (MSE) between the predicted and actual output curves. The Kullback-Leibler (KL) divergence between the approximate posterior and the standard normal prior is used as a regularization term to shape the latent space. A weighting factor $\beta = 0.05$ is applied to the KL divergence to control the trade-off between reconstruction fidelity and latent space regularity. The model is trained for 500 epochs with a batch size of 32. To prevent numerical instability and warnings related to on-the-fly variable creation, the encoder and decoder are pre-built using a dummy batch prior to training. The training dataset consists of ensemble predictions generated by 25 independently trained forward neural networks for each design point. For 650 training inputs, this results in a total of 16,250 training curves. Each curve is paired with its corresponding input design vector, which is repeated to match the ensemble expansion. All inputs are normalized using standard scaling, and the output predictions are inverse-transformed using the saved scaler after inference. The trained encoder and decoder are stored in separate files for use during sampling and uncertainty quantification.

After the Conditional Variational Autoencoder (CVAE) is trained, it can be utilized as a generative model to predict the distribution of time-series responses corresponding to a specific design input. The inference process leverages only the decoder component of the trained CVAE, along with the input scaler and output scaler saved during training. To begin, the unknown input test is selected from the test dataset. This vector is normalized using the same scalar as used during training to ensure consistency in the latent space representation. This normalized input is passed to two

different inference pipelines: one based on a trained ensemble of forward neural networks, and the other using the CVAE decoder. For the ensemble-based approach, 500 independently trained forward models are loaded sequentially. Each model accepts the scaled test input and predicts a corresponding output curve in the normalized space. These predicted curves are collected and subsequently inverse transformed using the saved output scaler to convert them back to physical units. The 5th and 95th percentile curves are computed across the ensemble to form uncertainty bounds, which serve as a baseline reference.

In the CVAE-based approach, the decoder is used to generate a large number of synthetic output curves (1000 in this case) conditioned on the selected test input. Specifically, the latent variable **z** is sampled repeatedly from a standard multivariate normal distribution of the same dimension as used during training (in this case, 32). For each sample, the decoder receives both the latent vector and the normalized design input and generates a predicted output curve. These curves, initially in normalized space, are inverse transformed using the same output scaler. The ensemble of synthetic curves is then used to compute the 5th and 95th percentile bounds, analogous to the ensemble method.

Figures 10.3 and 10.4 demonstrate the uncertainty quantification capability of CVAE for test points 3 and 2, respectively. They compare its 5th–95th percentile prediction envelope with the 5th and 95th percentiles calculated from the ensemble predictions of 500 models for the acceleration response. This indicates that the CVAE has successfully learned the distribution of ensemble predictions. The variable width of the

shaded region captures time-dependent uncertainty, with broader intervals in regions of higher variability and tighter bounds where predictions are more confident.

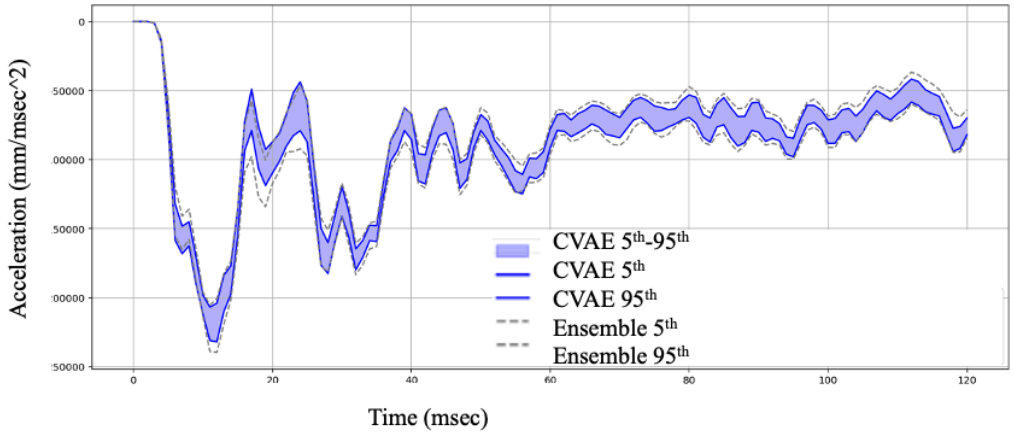

Figure 10.3 CVAE-based 5th and 95th percentiles for test point # 3

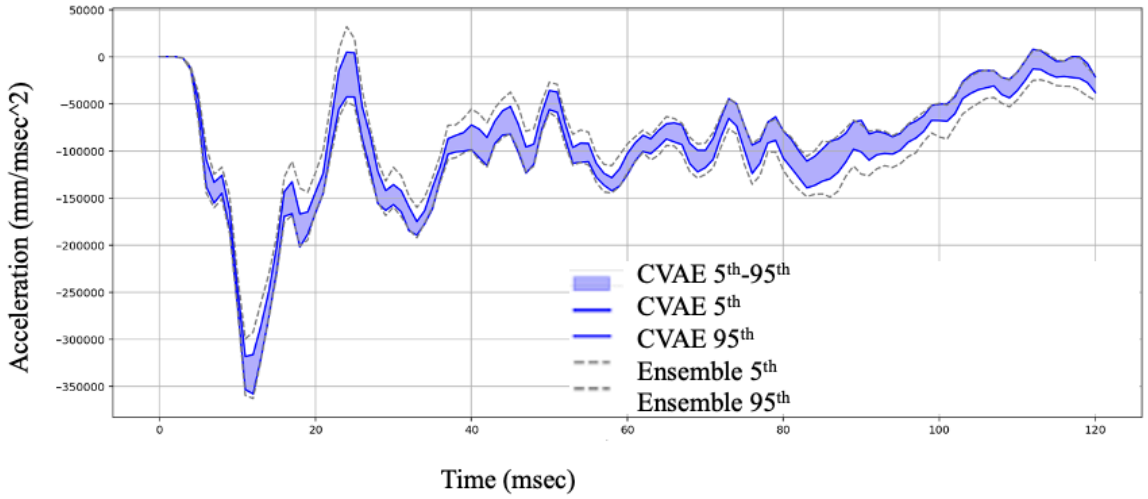

Figure 10.4 CVAE-based 5th and 95th percentiles for test point # 2

This demonstrates the CVAE's effectiveness as a generative model capable of reproducing both the shape and uncertainty of ensemble outputs using only a compact

latent space. The 5th and 95th percentile curves derived from the synthetic data generated by the CVAE can subsequently be utilized to construct an Adaptive Gaussian Process Regression (GPR) model, as detailed in Chapter 5. The primary advantage of the proposed CVAE-based framework lies in its ability to significantly reduce the time and computational resources required for uncertainty quantification. Instead of training and maintaining a large ensemble of models (e.g., 500 networks), the approach uses predictions from a relatively small ensemble (e.g., 25 models) to train a Variational Autoencoder. Once trained, the CVAE acts as a generative AI model capable of producing a large number of synthetic prediction curves that preserve the statistical characteristics of the original ensemble.

CHAPTER 11

DISSERTATION CONTRIBUTIONS

This research introduces a novel framework for quantifying uncertainty in machine learning that is both straightforward to implement and highly effective. The work leverages network ensemble methods and autoencoders to address the persistent challenge of uncertainty quantification in predictive modeling. A significant contribution, detailed in Chapter 5, is the development of an adaptive Gaussian Process Regression (GPR) model, which offers the advantage of being architecture-independent and, therefore, applicable to a broad range of machine learning models. The proposed methodology is versatile, supporting uncertainty quantification for both scalar outputs and time-dependent problems. The effectiveness and generalizability of the approach were demonstrated through its application to a real-world crash simulation case study.

By offering a robust and adaptable solution, this research advances the field of uncertainty quantification in machine learning, providing both researchers and practitioners with a valuable tool for improving the reliability and accuracy of their predictive models. Additionally, a novel autoencoder-based technique, termed the “Trust Index,” was developed to flag unseen or out-of-distribution inputs that may not be well-represented in the training data.

Chapter 9 presented an indirect strategy for uncertainty quantification through the construction of the most probable model, utilizing an ensemble-based approach. Subsequently, Chapter 10 introduced a Conditional Variational Autoencoder (CVAE)

approach to generate synthetic data which can be used in place of predictions generated by ensemble of models. Below are specific contributions:

- Development of an "Adaptive GPR using NN Ensemble Method." This approach introduced a novel method to integrate the strengths of network ensemble methods and Gaussian Process Regression (GPR), effectively mitigating the individual limitations of each technique and providing a more robust framework for uncertainty quantification in machine learning models.

- Creation of a "Trust Index." This is an innovative autoencoder-based method for input data assessment. By training autoencoders on input distributions, this approach enables a reliable identification and flagging of out-of-distribution or previously unseen inputs, thereby enhancing model reliability and safety.

- Introduction of the "Most Probable Model" technique. This new method identifies the most probable model within an ensemble by determining the most frequently occurring weights across network instances. This technique offers a systematic way to extract representative models from ensembles, further improving interpretability and robustness in predictive tasks. It also improves computational efficiency of the mean prediction.

- Conditional Variational Autoencoder (CVAE) for Uncertainty Quantification. We introduced and validated a CVAE approach specifically designed for uncertainty quantification in engineering applications. This method generates input-conditioned synthetic data, substantially reducing the need for computationally expensive ensemble training while maintaining high accuracy and reliability in predictive modeling.

CHAPTER 12

FUTURE WORK

This research primarily focused on utilizing ensemble-based network approaches for uncertainty quantification, where the effectiveness significantly depends on the ensemble's capability to represent the ground truth adequately. Future efforts could concentrate on developing a comprehensive framework or tool capable of integrating all the methodologies presented in this dissertation, including the Adaptive Gaussian Process Regression (GPR), the Trust Index, the Most Probable Model, and the synthetic data generation using Conditional Variational Autoencoders (CVAE). Moreover, the developed "Most Probable Model" approach, which currently serves as an indirect measure for mean predictions, can be expanded to assess uncertainty quantification. Further development and refinement of this approach to facilitate direct uncertainty quantification would considerably enhance its practicality and interpretability in diverse predictive modeling applications.

Appendix A: Equations of Motion for Vehicle Model

The state-space equations of motion of the 11-DOF full-vehicle model of Figure 16 are

$$[\dot{X}] = [A]\,[X] + [B]\,[U]$$
$$[Y] = [C]\,[X]$$

where

$$[A] = [A1\ \ A2\ \ A3\ \ A4\ \ A5\ \ A6\ \ A7\ \ A8\ \ A9\ \ A10\ \ A11\ \ A12\ \ A13\ \ A14]$$

$$[B] = \left[0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad \frac{K_{t1}}{M_1} \quad 0 \quad \frac{K_{t2}}{M_2} \quad 0 \quad \frac{K_{t3}}{M_3} \quad 0 \quad \frac{K_{t4}}{M_4}\right]$$

$$[X] = [X_1 \quad X_2 \quad X_3 \quad X_4 \quad X_5 \quad X_6 \quad X_7 \quad X_8 \quad X_9 \quad X_{10} \quad X_{11} \quad X_{12} \quad X_{13} \quad X_{14}]^T$$

with

$$Z_{cg} = X_1, \quad \dot{Z}_{cg} = X_2, \quad \Theta = X_3, \quad \dot{\Theta} = X_4,$$
$$\dot{Z}_1 = X_8, \quad Z_2 = X_9, \quad \dot{Z}_2 = X_{10}, \quad Z_3 = X_{11},$$

$$\Phi = X_5, \quad \dot{\Phi} = X_6, \quad Z_1 = X_7,$$
$$\dot{Z}_3 = X_{12}, \quad Z_4 = X_{13}, \quad \dot{Z}_4 = X_{14}$$

The matrices $[U]$ and $[C]$ are

$$[U] = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad Zt_1 \quad 0 \quad Z_{t2} \quad 0 \quad Z_{t3} \quad 0 \quad Z_{t4}]^T$$

$$[C] = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The elements of matrix $[A]$ are provided below.

$$[A1] = [0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]$$
$$[A3] = [0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]$$
$$[A5] = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]$$
$$[A7] = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]$$
$$[A9] = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0]$$
$$[A11] = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0]$$
$$[A13] = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1]$$

$$[A2] = \begin{bmatrix} (-K_1 - K_2 - K_3 - K_4)/\,Mcg \\ (-C_1 - C_2 - C_3 - C_4)\,/\,M_{cg} \\ (a*K_1 + a*K_2 - b*K_3 - b*K_4)/Mcg \\ (a*C_1 + a*C_2 - b*C_3 - b*C_4)/Mcg \\ (c*K_1 - d*K_2 + c*K_3 - d*K_4)/Mcg \\ (c*C_1 - d*C_2 + c*C_3 - d*C_4)/Mcg \\ K_1/Mcg \\ C_1/Mcg \\ K_2/Mcg \\ C_2/Mcg \\ K_3/Mcg \\ C_3/Mcg \\ K_4/Mcg \\ C_4/Mcg \end{bmatrix}^T$$

$$[A4] = \begin{bmatrix} (a*K_1 + a*K_2 - b*K_3 - b*K_4)/I_{yy} \\ (a*C_1 + a*C_2 - b*C_3 - b*C_4)\,/Iyy \\ (-a^2 K_1 - a^2 K_2 - b^2 K_3 - b^2 K_4)/Iyy \\ (-a^2 C_1 - a^2 C_2 - b^2 C_3 - b^2 C_4)/Iyy \\ (-a*c*K_1 + a*d*K_2 + b*c*K_3 - b*d*K_4)/Iyy \\ (-a*c*C_1 + a*d*C_2 + b*c*C_3 - b*d*C_4)/Iyy \\ -a*K_1/Iyy \\ -a*C_1/Iyy \\ -a*K_2/Iyy \\ -a*C_2/Iyy \\ b*K_3/Iyy \\ b*C_3/Iyy \\ b*K_4/Iyy \\ b*C_4/Iyy \end{bmatrix}^T$$

$$[A6] = \begin{bmatrix} (c*K_1 - d*K_2 + c*K_3 - d*K_4)/I_{xx} \\ (c*C_1 - d*C_2 + c*C_3 - d*C_4)/I_{xx} \\ (-a*c*K_1 + a*d*K_2 + b*c*K_3 - b*d*K_4)/I_{xx} \\ (-a*c*C_1 + a*d*C_2 + b*c*C_3 - b*d*C_4)/I_{xx} \\ (-c^2K_1 - d^2K_2 - c^2K_3 - d^2K_4)/I_{xx} \\ (-c^2C_1 - d^2C_2 - c^2C_3 - d^2C_4)/I_{xx} \\ -c*K_1/I_{xx} \\ -c*C_1/I_{xx} \\ d*K_2/I_{xx} \\ d*C_2/I_{xx} \\ -c*K_3/I_{xx} \\ -c*C_3/I_{xx} \\ d*K_4/I_{xx} \\ d*C_4/I_{xx} \end{bmatrix}^T$$

$$[A_8] = \begin{bmatrix} K_1/M_1 \\ C_1/M_1 \\ -a*K_1/M_1 \\ -a*C_1/M_1 \\ -c*K_1/M_1 \\ -c*C_1/M_1 \\ (-\boldsymbol{K_1} - \boldsymbol{K_{t1}})/M_1 \\ -C_1/M_1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}^T \qquad [A_{10}] = \begin{bmatrix} K_2/M_2 \\ C_2/M_2 \\ -a*K_2/M_2 \\ -a*C_2/M_2 \\ d*K_2/M_2 \\ d*C_2/M_2 \\ 0 \\ 0 \\ (-\boldsymbol{K_2} - \boldsymbol{K_{t2}})/M_2 \\ -C_2/M_2 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}^T$$

$$[A_{12}]=\begin{bmatrix} K_3/M_3 \\ C_3/M_3 \\ b*K_3/M_3 \\ b*C_3/M_3 \\ -c*K_3/M_3 \\ -c*C_3/M_3 \\ 0 \\ 0 \\ 0 \\ 0 \\ (-\boldsymbol{K_3}-\boldsymbol{K_{t3}})/M_3 \\ -C_3/M_3 \\ 0 \\ 0 \end{bmatrix}^T \qquad [A_{14}]=\begin{bmatrix} K_4/M_4 \\ C_4/M_4 \\ b*K_4/M_4 \\ b*C_4/M_4 \\ d*K_4/M_4 \\ d*C_4/M_4 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ (-\boldsymbol{K_4}-\boldsymbol{K_{t4}})/M_4 \\ -C_4/M_4 \end{bmatrix}^T$$

REFERENCES

[1] Karniadakis, G.E., Kevrekidis, I.G., Lu, L., et al., 2021, "Physics-informed Machine Learning," *Nat Rev Phys* 3: 422–440. https://doi.org/10.1038/s42254-021-00314-5.

[2] Hexagon "Odyssee" Accessed [04/25/2024]. https://hexagon.com/products/odyssee.

[3] Nemani, V., Biggio, L., Huan, X., et al., 2023, "Uncertainty Quantification in Machine Learning for Engineering Design and Health Prognostics: A Tutorial," *Mechanical Systems and Signal Processing*, 205, 110796.

[4] Rasmussen, C. E., 2006, *Gaussian Processes in Machine Learning*, MIT Press.

[5] Brochu, E., Cora, V. M., de Freitas, N., 2010, "A Tutorial on Bayesian Optimization of Expensive Cost Functions with Application to Active User Modeling and Hierarchical Reinforcement Learning," *Computer Science*.

[6] Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., de Freitas, N., 2016, "Taking the Human Out of the Loop: A Review of Bayesian Optimization," *Proceedings of the IEEE*, 104 (1): 148–175.

[7] Liu, H., Ong, Y.-S., Shen, X., Cai, J., 2020. "When Gaussian Process Meets Big Data: A Review of Scalable GPs," *IEEE Transactions on Neural Networks and Learning Systems,* 31 (11): 4405–4423.

[8] Berger, O., 1985, *Statistical Decision Theory and Bayesian Analysis*, Springer Series in Statistics. Springer New York, New York, NY. ISBN 978-1-4419-3074-3.

[9] Bernardo, J. M., Smith, A. F. M., 2000, *Bayesian Theory*, John Wiley & Sons, New York, NY.

[10] Sivia, D. S., Skilling, J., 2006, *Data Analysis: A Bayesian Tutorial*, 2nd ed. Oxford University Press, New York, NY.

[11] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R., 2014, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *The Journal of Machine Learning Research*, 15 (1): 1929–1958.

[12] Gal, Y., Ghahramani, Z., 2016. "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning," *International Conference on Machine Learning*, 1050–1059. PMLR.

[13] Gal, Y., Ghahramani, Z., 2016, "A Theoretically Grounded Application of Dropout in Recurrent Neural Networks," *Advances in Neural Information Processing Systems*, 1019–1027.

[14] Opitz, D., Maclin, R., 1999, "Popular Ensemble Methods: An Empirical Study," *Journal of Artificial Intelligence Research,* 11: 169–198.

[15] Dietterich, T. G., 2000, "Ensemble Methods in Machine Learning," *International Workshop on Multiple Classifier Systems*, 1–15, Springer.

[16] Breiman, L., 1996, "Bagging predictors," *Machine Learning*, 24 (2): 123–140.

[17] Schapire, R., Yoav F., 2012, *Boosting: Foundations and Algorithms*, Adaptive Computation and Machine Learning Series, The MIT Press.

[18] Thelen, A., Zhang, X., Fink, O., Lu, Y., Ghosh, S., Youn, B. D., Todd, M. D., Mahadevan, S., Hu, C., Hu, Z., 2023, "A Comprehensive Review of Digital Twin–Part 2: Roles of Uncertainty Quantification and Optimization, a Battery Digital Twin, and Perspectives," *Structural and Multidisciplinary Optimization*, 66 (1): 1–43.

[19] Zhang, X., Mahadevan, S., 2019, "Ensemble Machine Learning Models for Aviation Incident Risk Prediction," *Decision Support Systems*, 116: 48–63.

[20] Lakshminarayanan, B., Pritzel, A., Blundell, C., 2017, "Simple and Scalable Predictive Uncertainty Estimation Using Deep Ensembles," *Advances in Neural Information Processing Systems*, 30.

[21] Zhang, X., Chan, F. T., Mahadevan, S., 2022, "Explainable Machine Learning In Image Classification Models: An Uncertainty Quantification Perspective," *Knowledge-Based Systems*, 243: 108418.

[22] Ovadia, Y., Fertig, E., Ren, J., Nado, Z. D., Sculley, S., Dillon, J., Lakshminarayanan, B., & Snoek, J., 2019, "Can You Trust Your Model's Uncertainty? Evaluating Predictive Uncertainty Under Dataset Shift," *Proceedings of the 33rd Conference on Neural Information Processing Systems* (NeurIPS 2019), Vancouver, Canada.

[23] Hullermeier, E., Waegeman, W., 2021, "Aleatoric and Epistemic Uncertainty In Machine Learning: An Introduction To Concepts And Methods," *Machine Learning*, 110 (3): 457–506.

[24] Der Kiureghian, A., Ditlevsen, O., 2009, "Aleatory or Epistemic? Does It Matter?" *Structural Safety*, 31 (2): 105–112.

[25] Gal, Y., Hron, J., Kendall, A., 2017, "Concrete Dropout," *Proceedings of the 31st Conference on Neural Information Processing Systems* (NIPS 2017), Long Beach, CA, USA.

[26] Osband, I., 2016, "Risk Versus Uncertainty In Deep Learning: Bayes, Bootstrap and The Dangers Of Dropout," *NIPS Workshop on Bayesian Deep Learning*, 192.

[27] Alarab, I., Prakoonwit, S., Nacer, M. I., 2021, "Illustrative Discussion of MC-Dropout in General Dataset: Uncertainty Estimation In Bitcoin," *Neural Processing Letters,* 53 (2): 1001–1011.

[28] Caldeira, J., Nord, B., 2020, "Deeply Uncertain: Comparing Methods of Uncertainty Quantification In Deep Learning Algorithms," *Machine Learning: Science and Technology*, 2 (1): 015002.

[29] Pedregosa et al., 2011, "Scikit-learn: Machine Learning in Python," JMLR 12: 2825-2830.

[30] Osband, I., 2016, "Risk Versus Uncertainty In Deep Learning: Bayes, Bootstrap and the Dangers of Dropout," *NIPS Workshop on Bayesian Deep Learning*, 192.

[31] Caldeira, J., Nord, B., 2020, "Deeply Uncertain: Comparing Methods of Uncertainty Quantification in Deep Learning Algorithms," *Machine Learning: Science and Technology*, 2 (1) 015002.

[32] Gal, Y., Ghahramani, Z., 2016, "Bayesian Convolutional Neural Networks with Bernoulli Approximate Variational Inference," *4th International Conference on Learning Representations* (ICLR).

[33] Zhang, X., Mahadevan, S., 2019, "Ensemble Machine Learning Models for Aviation Incident Risk Prediction," *Decision Support Systems*, 116 48-63.

[34] Zhang, X., Chan, F.T., Mahadevan, S., 2022, "Explainable Machine Learning in Image Classification Models: An Uncertainty Quantification Perspective," *Knowledge-Based Systems*, 243, 108418.

[35] Fort, S., Hu, H., Lakshminarayanan, B., 2020, "Deep Ensembles: A Loss Landscape Perspective," 10.48550/arXiv.1912.02757.

[36] Garcia-Pozuelo D, Gauchia A, Olmeda E, Diaz V, 2014, "Bump Modeling and Vehicle Vertical Dynamics Prediction," *Advances in Mechanical Engineering*, 6, doi:10.1155/2014/736576.

[37] Pearce, T., Zaki, M., Brintrup, A., Neely, A., 2020, "Uncertainty in Neural Networks: Approximately Bayesian Ensembling," *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics* (AISTATS) 2020, Palermo, Italy. PMLR: Volume 108.

[38] Pearce, T., Zaki, M., Brintrup, A., Neely, A., 2018, "High-Quality Prediction Intervals for Deep Learning: A Distribution-Free, Ensembled Approach," *Proceedings of the 35th International Conference on Machine Learning*, PMLR 80:4075-4084.

[39] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Computation, vol. 9, no. 8, pp. 1735–1780, 1997. https://doi.org/10.1162/neco.1997.9.8.1735

[40] A. Graves, “Supervised sequence labelling with recurrent neural networks,” Studies in Computational Intelligence, vol. 385, Springer, 2012. https://doi.org/10.1007/978-3-642-24797-2

[41] National Highway Traffic Safety Administration. 2024. Crash Simulation Vehicle Models. U.S. Department of Transportation. https://www.nhtsa.gov/crash-simulation-vehicle-models.

[42] Rasmussen, C. E., and Williams, C. K. I. Gaussian Processes for Machine Learning. MIT Press, 2006

[43] Nemani, V., Biggio, L., Huan, X., et al., 2023. “Uncertainty Quantification in Machine Learning for Engineering Design and Health Prognostics: A Tutorial”, arXiv preprint arXiv:2035.04933.

[44] Koenker, R., and Bassett, G., 1978. “Regression Quantiles”, *Econometrica*, 46(1), pp. 33–50.

[45] Taddy, M., 2013. “Bayesian Inference in Large Scale Structured Models”, Annual Review of Statistics and Its Application, 1, pp. 175–201.

[46] Goodfellow, I., Bengio, Y., and Courville, A., 2016. “Deep Learning”, MIT Press, Cambridge, MA.

[47] Kingma, D.P., and Ba, J., 2015. “Adam: A Method for Stochastic Optimization”, International Conference on Learning Representations (ICLR).

[48] Glorot, X., and Bengio, Y., 2010. “Understanding the Difficulty of Training Deep Feedforward Neural Networks”, Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS).

[49] Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O., 2017. "Understanding Deep Learning Requires Rethinking Generalization", International Conference on Learning Representations (ICLR).

[50] Ananthaswamy, A. 2024. Why Machines Learn. New York: Dutton.

[51] Kuleshov, V., Fenner, N., and Ermon, S., 2018. "Accurate Uncertainties for Deep Learning Using Calibrated Regression", Proceedings of the 35th International Conference on Machine Learning, in Proceedings of Machine Learning Research, vol. 80, pp. 2796–2804. PMLR. Available at: https://proceedings.mlr.press/v80/kuleshov18a.html.

[52] Gal, Y., 2016. "Uncertainty in Deep Learning", PhD thesis, University of Cambridge.

[53] Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D., 2015. "Weight Uncertainty in Neural Networks", arXiv preprint arXiv:1505.05424.

[54] Lakshminarayanan, B., Pritzel, A., and Blundell, C., 2017. "Simple and Scalable Predictive Uncertainty Estimation Using Deep Ensembles", arXiv preprint arXiv:1612.01474.

[55] Gal, Y., Hron, J., and Kendall, A., 2017. "Concrete Dropout", Advances in Neural Information Processing Systems, pp. 3584–3593.

[56] Gneiting, T., and Raftery, A.E., 2005. "Weather Forecasting with Ensemble Methods", Science, 310(5746), pp. 248–249.

[57] Platt, J., 1999. "Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods", Advances in Large Margin Classifiers, 10(3), pp. 61–74.

[58] Gneiting, T., Balabdaoui, F., and Raftery, A.E., 2007. “Probabilistic Forecasts, Calibration and Sharpness”, Journal of the Royal Statistical Society: Series B (Statistical Methodology), 69(2), pp. 243–268.

[59] Niculescu-Mizil, A., and Caruana, R., 2005. “Predicting Good Probabilities with Supervised Learning”, Proceedings of the 22nd International Conference on Machine Learning, pp. 625–632.

[60] Gal, Y., and Ghahramani, Z., 2016. “Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning”, Proceedings of the 33rd International Conference on Machine Learning (ICML-16).

[61] Murphy, A.H., 1973. “A New Vector Partition of the Probability Score”, Journal of Applied Meteorology, 12(4), pp. 595–600.

[62] Saxena, A., Sun, M., and Ng, A.Y., 2009. “Make3D: Learning 3D Scene Structure from a Single Still Image”, IEEE Transactions on Pattern Analysis and Machine Intelligence, 31(5), pp. 824–840.

[63] Kendall, A., and Gal, Y., 2017. “What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?”, Advances in Neural Information Processing Systems, pp. 5580–5590.

[64] Ghavamzadeh, M., Mannor, S., Pineau, J., and Tamar, A., 2015. “Bayesian Reinforcement Learning: A Survey”, Foundations and Trends in Machine Learning, 8(5–6), pp. 359–483.

[65] Van Roy, B., Bertsekas, D.P., Lee, Y., and Tsitsiklis, J.N., 1997. “A Neuro-Dynamic Programming Approach to Retailer Inventory Management”, Proceedings of the 36th IEEE Conference on Decision and Control, 4, pp. 4052–4057.

[66] Box, G.E.P., and Hunter, W.G., 1962. “A Useful Method for Model-Building”, Technometrics, 4(3), pp. 301–318.

[67] Gelman, A., and Hill, J., 2007. “Data Analysis Using Regression and Multilevel/Hierarchical Models”, Cambridge University Press, Volume 1.

[68] Kucukelbir, A., Wang, Y., and Blei, D.M., 2017. “Evaluating Bayesian Models with Posterior Dispersion Indices”, Proceedings of the 34th International Conference on Machine Learning, Proceedings of Machine Learning Research, 70, pp. 1925–1934, Sydney, Australia.

[69] Jiang, X., Osl, M., Kim, J., and Ohno-Machado, L., 2012. “Calibrating Predictive Model Estimates to Support Personalized Medicine”, Journal of the American Medical Informatics Association, 19(2), pp. 263–274.

[70] Raftery, A.E., Gneiting, T., Balabdaoui, F., and Polakowski, M., 2005. “Using Bayesian Model Averaging to Calibrate Forecast Ensembles”, Monthly Weather Review, 133(5), pp. 1155–1174.

[71] Nguyen, K., and O’Connor, B., 2015. “Posterior Calibration and Exploratory Analysis for Natural Language Processing Models”, Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 1587–1598.

[72] Zadrozny, B., and Elkan, C., 2002. “Transforming Classifier Scores into Accurate Multiclass Probability Estimates”, Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD), pp. 694–699.

[73] Kuleshov, V., and Liang, P., 2015. “Calibrated Structured Prediction”, Advances in Neural Information Processing Systems (NIPS).

[74] Guo, C., Pleiss, G., Sun, Y., and Weinberger, K.Q., 2017. “On Calibration of Modern Neural Networks”, arXiv preprint arXiv:1706.04599.

[75] Kuleshov, V., and Ermon, S., 2017. “Estimating Uncertainty Online Against an Adversary”, Proceedings of the AAAI Conference on Artificial Intelligence, pp. 2110–2116.

[76] Dawid, A.P., 1984. “Present Position and Potential Developments: Some Personal Views: Statistical Theory: The Prequential Approach”, *Journal of the Royal Statistical Society*, Series A (General), 147, pp. 278–292.

[77] Gneiting, T., and Raftery, A.E., 2007. “Strictly Proper Scoring Rules, Prediction, and Estimation”, *Journal of the American Statistical Association*, 102(477), pp. 359–378.

[78] Hersbach, H., 2000. “Decomposition of the Continuous Ranked Probability Score for Ensemble Prediction Systems”, *Weather and Forecasting*, 15(5), pp. 559–570.

[79] MacKay, D.J.C., 1992. “Bayesian Interpolation”, *Neural Computation*, 4(3), pp. 415–447.

[80] Gal, Y., and Ghahramani, Z., 2016. “A Theoretically Grounded Application of Dropout in Recurrent Neural Networks”, Advances in Neural Information Processing Systems (NIPS 29).

[81] Jégou, S., Drozdzal, M., Vazquez, D., Romero, A., and Bengio, Y., 2017. “The One Hundred Layers Tiramisu: Fully Convolutional DenseNets for Semantic Segmentation”, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), pp. 1175–1183.

[82] Maddison, C.J., Mnih, A., and Teh, Y.W., 2016. “The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables”, arXiv preprint arXiv:1611.00712.

[83] Zhao, S., Song, J., and Ermon, S., 2019. “InfoVAE: Balancing Learning and Inference in Variational Autoencoders”, Proceedings of the AAAI Conference on Artificial Intelligence, 33(1), pp. 5885–5892. https://doi.org/10.1609/aaai.v33i01.33015885.

[84] Rezende, S., and Kingma, D.P., 2019. “Towards a Deeper Understanding of Variational Autoencoding Models”, arXiv preprint arXiv:1907.08956. https://arxiv.org/abs/1907.08956.

[85] Alemi, A.A., Poole, B., Fischer, I., Dillon, J.V., Saurous, R.A., and Murphy, K., 2017. “An Information-Theoretic Analysis of Deep Latent-Variable Models”, arXiv preprint arXiv:1711.00464.

[86] Arjovsky, M., Chintala, S., and Bottou, L., 2017. “Wasserstein GAN”, arXiv e-prints.

[87] Chen, X., Kingma, D.P., Salimans, T., Duan, Y., Dhariwal, P., Schulman, J., Sutskever, I., and Abbeel, P., 2016. “Variational Lossy Autoencoder”, arXiv preprint arXiv:1611.02731.

[88] Dziugaite, G.K., Roy, D.M., and Ghahramani, Z., 2015. “Training Generative Neural Networks via Maximum Mean Discrepancy Optimization”, arXiv preprint arXiv:1505.03906.

[89] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y., 2014. “Generative Adversarial Nets”, *Advances in Neural Information Processing Systems*, pp. 2672–2680.

[90] Gretton, A., Borgwardt, K.M., Rasch, M., Schölkopf, B., and Smola, A.J., 2007. “A Kernel Method for the Two-Sample Problem”, *Advances in Neural Information Processing Systems*, pp. 513–520.

[91] Gulrajani, I., Kumar, K., Ahmed, F., Taiga, A.A., Visin, F., Vázquez, D., and Courville, A.C., 2016. “PixelVAE: A Latent Variable Model for Natural Images”, arXiv preprint arXiv:1611.05013.

[92] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A., 2017. “Improved Training of Wasserstein GANs”, arXiv preprint arXiv:1704.00028.

[93] Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A., 2017. “beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework”, International Conference on Learning Representations (ICLR).

[94] Kingma, D.P., and Welling, M., 2013. “Auto-Encoding Variational Bayes”, arXiv e-prints, arXiv:1312.6114.

[95] Kingma, D.P., Rezende, D.J., Mohamed, S., and Welling, M., 2014. “Semi-Supervised Learning with Deep Generative Models”, CoRR, abs/1406.5298.

[96] Kingma, D.P., Salimans, T., and Welling, M., 2016. “Improving Variational Inference with Inverse Autoregressive Flow”, arXiv preprint arXiv:1606.04934.

[97] Li, Y., Song, J., and Ermon, S., 2017. “InfoGAIL: Interpretable Imitation Learning from Visual Demonstrations”, *Advances in Neural Information Processing Systems*, pp. 3812–3822.

[98] Li, Y., Swersky, K., and Zemel, R., 2015. “Generative Moment Matching Networks”, International Conference on Machine Learning (ICML), pp. 1718–1727.

[99] Liu, Q., and Wang, D., 2016. “Stein Variational Gradient Descent: A General Purpose Bayesian Inference Algorithm”, arXiv preprint arXiv:1608.04471.

[100] Makhzani, A., Shlens, J., Jaitly, N., and Goodfellow, I., 2015. “Adversarial Autoencoders”, arXiv preprint arXiv:1511.05644.

[101] Nowozin, S., Cseke, B., and Tomioka, R., 2016. “f-GAN: Training Generative Neural Samplers Using Variational Divergence Minimization”, *Advances in Neural Information Processing Systems*, pp. 271–279.

[102] Radford, A., Metz, L., and Chintala, S., 2015. “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, arXiv preprint arXiv:1511.06434.

[103] Salakhutdinov, R., and Murray, I., 2008. “On the Quantitative Analysis of Deep Belief Networks”, Proceedings of the 25th International Conference on Machine Learning (ICML), pp. 872–879. ACM.

[104] Shu, R., Bui, H.H., Zhao, S., Kochenderfer, M.J., and Ermon, S., 2018. “Amortized Inference Regularization”, *Advances in Neural Information Processing Systems* (NeurIPS).

[105] van den Oord, A., Kalchbrenner, N., and Kavukcuoglu, K., 2016. “Pixel Recurrent Neural Networks”, CoRR, abs/1601.06759.

[106] Yang, Z., Hu, Z., Salakhutdinov, R., and Berg-Kirkpatrick, T., 2017. “Improved Variational Autoencoders for Text Modeling Using Dilated Convolutions”, CoRR, abs/1702.08139.

[107] Zhao, S., Song, J., and Ermon, S., 2018. “The Information Autoencoding Family: A Lagrangian Perspective on Latent Variable Generative Models”, arXiv preprint arXiv:1806.06514.

[108] Zhu, J., Park, T., Isola, P., and Efros, A.A., 2017. “Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks”, CoRR, abs/1703.10593.

[109] Arjovsky, M., Chintala, S., and Bottou, L., 2017. “Wasserstein GAN”, arXiv e-prints.

[110] Bachman, P., 2016. “An Architecture for Deep, Hierarchical Generative Models”, *Advances in Neural Information Processing Systems*, pp. 4826–4834.

[111] Bordes, F., Honari, S., and Vincent, P., 2016. “Learning to Generate Samples from Noise through Infusion Training”, OpenReview. https://openreview.net/pdf?id=BJAFbaolg

[112] Bowman, S.R., Vilnis, L., Vinyals, O., Dai, A.M., Józefowicz, R., and Bengio, S., 2015. “Generating Sentences from a Continuous Space”, CoRR, abs/1511.06349.

[113] Burda, Y., Grosse, R., and Salakhutdinov, R., 2015. “Importance Weighted Autoencoders”, arXiv preprint arXiv:1509.00519.

[114] Chen, X., Kingma, D.P., Salimans, T., Duan, Y., Dhariwal, P., Schulman, J., Sutskever, I., and Abbeel, P., 2016. “Variational Lossy Autoencoder”, arXiv preprint arXiv:1611.02731.

[115] Chung, J., Kastner, K., Dinh, L., Goel, K., Courville, A.C., and Bengio, Y., 2015. “A Recurrent Latent Variable Model for Sequential Data”, CoRR, abs/1506.02216.

[116] Dosovitskiy, A., and Brox, T., 2016. “Generating Images with Perceptual Similarity Metrics Based on Deep Networks”, CoRR, abs/1602.02644.

[117] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y., 2014. “Generative Adversarial Nets”, *Advances in Neural Information Processing Systems*, pp. 2672–2680.

[118] Gulrajani, I., Kumar, K., Ahmed, F., Taiga, A.A., Visin, F., Vázquez, D., and Courville, A.C., 2016. “PixelVAE: A Latent Variable Model for Natural Images”, CoRR, abs/1611.05013.

[119] Rezende, D.J., Mohamed, S., and Wierstra, D., 2014. “Stochastic Backpropagation and Approximate Inference in Deep Generative Models”, arXiv e-prints.

[120] Sønderby, C.K., Raiko, T., Maaløe, L., Sønderby, S.K., and Winther, O., 2016. “Ladder Variational Autoencoders”, arXiv e-prints.

[121] Kingma, D.P., and Welling, M., 2013. “Auto-Encoding Variational Bayes”, arXiv e-prints.

[122] Kingma, D.P., Salimans, T., and Welling, M., 2016. “Improving Variational Inference with Inverse Autoregressive Flow”, arXiv preprint arXiv:1606.04934.

[123] Lamb, A., Dumoulin, V., and Courville, A., 2016. “Discriminative Regularization for Generative Models”, arXiv preprint arXiv:1602.03220.

[124] Larsen, A.B.L., Sønderby, S.K., and Winther, O., 2015. “Autoencoding Beyond Pixels Using a Learned Similarity Metric”, arXiv preprint arXiv:1512.09300.

[125] Nowozin, S., Cseke, B., and Tomioka, R., 2016. “f-GAN: Training Generative Neural Samplers Using Variational Divergence Minimization”, *Advances in Neural Information Processing Systems*, pp. 271–279.

[126] Radford, A., Metz, L., and Chintala, S., 2015. “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, arXiv preprint arXiv:1511.06434.

[127] Salimans, T., Karpathy, A., Chen, X., and Kingma, D.P., 2017. “PixelCNN++: Improving the PixelCNN with Discretized Logistic Mixture Likelihood and Other Modifications”, arXiv preprint arXiv:1701.05517.

[128] van den Oord, A., Kalchbrenner, N., Espeholt, L., Vinyals, O., Graves, A., et al., 2016a. “Conditional Image Generation with PixelCNN Decoders”, *Advances in Neural Information Processing Systems*, pp. 4790–4798.

[129] van den Oord, A., Kalchbrenner, N., and Kavukcuoglu, K., 2016. “Pixel Recurrent Neural Networks”, CoRR, abs/1601.06759.

PUBLICATIONS

1. Chavare, S. and Mourelatos, Z. P., 2025, "Uncertainty Quantification in Machine Learning using an Ensemble Approach with Gaussian Process Regression," SAE Paper 2025-01-8199, 2024 SAE WCX, Detroit, MI.
2. Chavare, S. and Mourelatos, Z. P., 2025, "Conditional Variational Autoencoder-based Synthetic Ensembles for Efficient Model Uncertainty Quantification," in preparation for submission to ASME *Journal of Mechanical Design*.
3. Chavare, S. and Mourelatos, Z. P., 2025, "Efficient Ensemble-based Model Uncertainty Quantification Using Autoencoder-driven Anomaly Detection and Most Probable Model Estimation," in preparation for submission to ASME *Journal of Mechanical Design*.