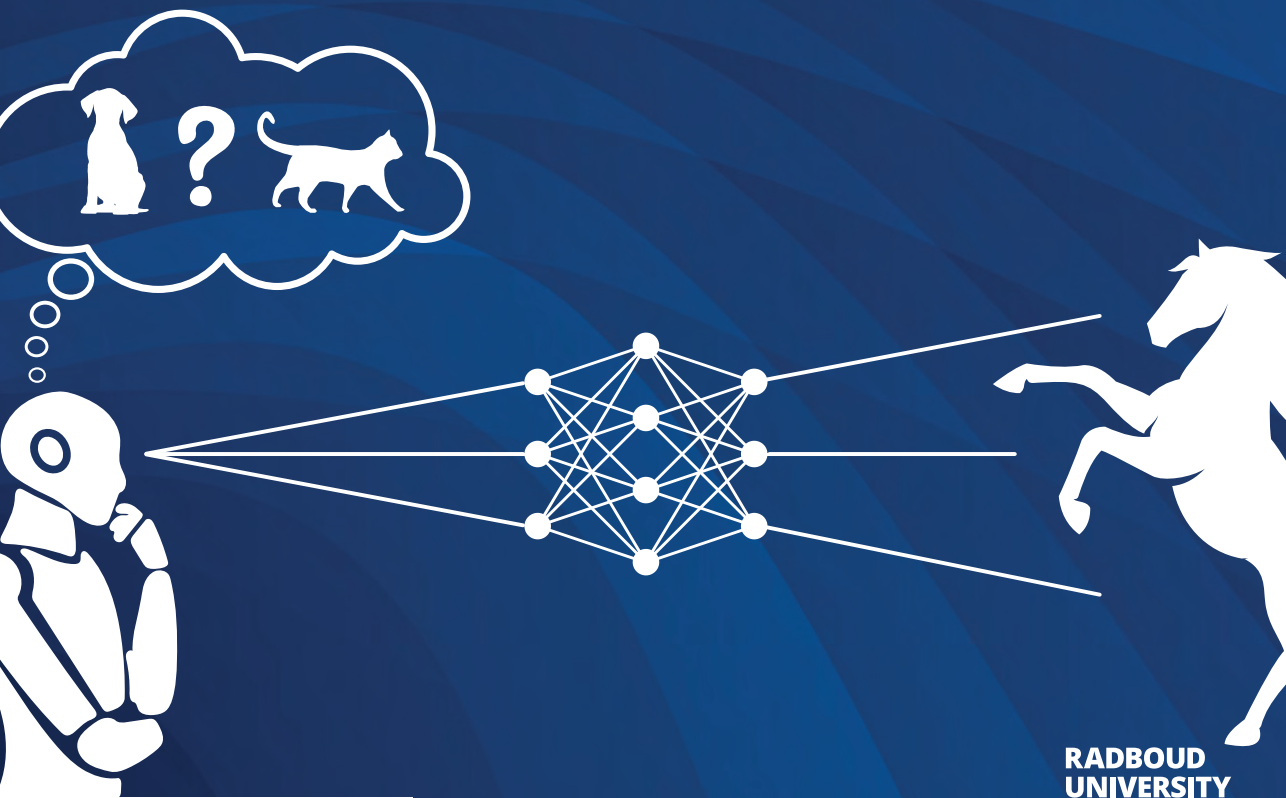
Uncertainty Quantification of Machine Learning Models

L. A. Æ. Sluijterman



Institute for Mathematics,
Astrophysics and Particle Physics

RADBOUD UNIVERSITY PRESS

Radboud Dissertation Series

Uncertainty Quantification of Machine Learning Models

Laurens Sluijterman

Laurens Sluijterman
Uncertainty Quantification of Machine Learning Models

Radboud Dissertations Series
ISSN: 2950-2772 (Online); 2950-2780 (Print)

Published by RADBOUD UNIVERSITY PRESS
Postbus 9100, 6500 HA Nijmegen, The Netherlands
www.radbouduniversitypress.nl

Design: **Laurens Sluijterman**
Cover: **Marjolein van Borselen**
Printing: DPN Rikken/Pumbo

ISBN: 9789465150475
DOI: 10.54195/9789465150475
Free download at: https://doi.org/10.54195/9789465150475

RADBOUD
UNIVERSITY
PRESS

Uncertainty Quantification of Machine Learning Models

Proefschrift ter verkrijging van de graad van doctor
aan de Radboud Universiteit Nijmegen
op gezag van de rector magnificus prof. dr. J.M. Sanders,
volgens besluit van het college voor promoties
in het openbaar te verdedigen op

woensdag 26 maart 2025
om 16.30 uur precies

door

Laurens Auke Æmiel Sluijterman
geboren op 14 november 1995
te Eindhoven

Promotoren:

Prof. dr. E.A. Cator

Prof. dr. T.M. Heskes

Manuscriptcommissie:

Prof. dr. J.J. Houwing-Duistermaat

Prof. dr. J.M. Hernández-Lobato (University of Cambridge, Verenigd Koninkrijk)

Prof. dr. W. Waegeman (Universiteit Gent, België)

Contents

Chapter 1

Introduction

Machine learning has seen an enormous rise over the past decades. Due to the exponential growth in computing power, machine-learning models have evolved from basic neural networks and decision trees, capable of performing straightforward tasks, to vastly complex architectures that may have billions of parameters.

As the capabilities of machine learning grow, so does its integration into safety-critical applications such as medical-image analysis (Varoquaux and Cheplygina, 2022), self-driving cars (Miglani and Kumar, 2019; Parekh et al., 2022), and the prediction of natural disasters (Hernández et al., 2022; Bentivoglio et al., 2022). For these applications, it is essential that these models are trustworthy.

A trustworthy model requires trustworthy uncertainty estimates (Gal, 2016). Relying merely on predictive performance is insufficient. However, producing these uncertainty estimates is far from trivial. Modern models can easily have millions of parameters, making the direct use of many classical techniques difficult or outright impossible. It is this problem of developing new methods to quantify the uncertainty in the predictions of machine-learning models that this thesis contributes to.

This introduction, which also serves as the background chapter, is structured as follows. First, we provide a broad overview of the different types of machine learning. We then focus on neural networks, arguably the currently most popular type of model, providing necessary details on the training process and various architectures. The remainder of the chapter discusses what causes the

uncertainties in the predictions and presents some of the popular approaches that currently exist to tackle these various sources of uncertainty. Lastly, we provide an overview of the structure of the rest of this thesis.

1.1 Machine Learning

Machine learning can be described as the practice of letting a computer learn from data. Imagine the task of distinguishing between images of cats and dogs. In a classical approach, we would think of rules such as whether it has large whiskers, pointy ears, or a long nose. With machine learning, we would give a model many examples of cats and dogs without explicitly telling the model how it has to make a distinction.

Machine learning is typically divided into three areas: reinforcement learning, unsupervised learning, and supervised learning. Figure 1.1 illustrates the division.

Reinforcement learning: The model can interact with the environment and learns from the consequences of its actions. An example is a model playing a video game where it is rewarded for achieving a better time or surviving longer. Another example is a robot that learns to perform a certain task, such as picking up an object, by being rewarded for successful actions.

Unsupervised learning: The model is trained on unlabeled data. Typical examples include clustering, dimensionality reduction, and finding association rules. Clustering methods aim to group data based on similarities or differences, association models search for relations between features in a given data set, and dimensionality reduction approaches try to represent the data in a lower dimension without losing too much information. This is useful either as a pre-processing step or to visualize the data.

Supervised learning: The model is trained on labeled data. The goal is to predict a label or target for a given input or covariate. A distinction is typically made between regression, where the target is continuous, and classification where the label represents a class, for instance cats or dogs. The work in this thesis falls inside this category and we will therefore explore supervised learning in greater detail.

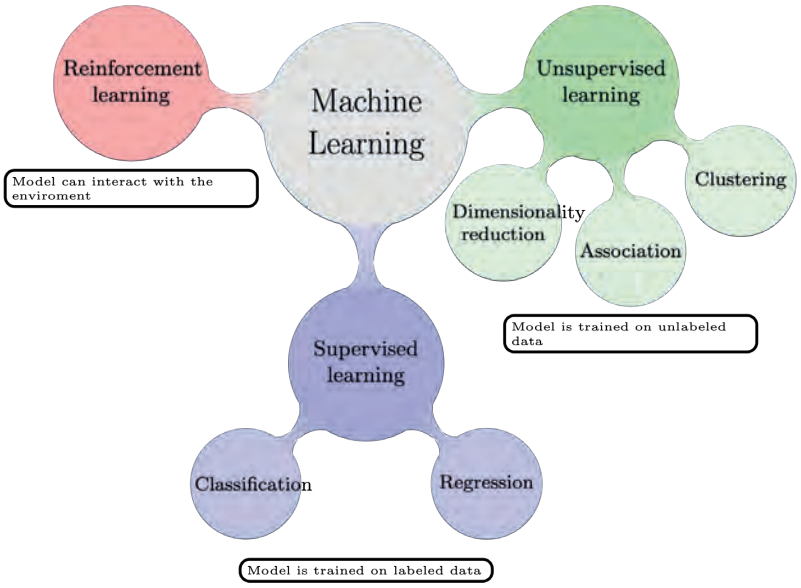


Figure 1.1: The three broad types of machine learning. Reinforcement learning, where the model can interact with the environment; unsupervised learning, where the model has access to unlabeled data; and supervised learning, where the model has access to labeled data.

We consider the situation where we have a data set $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \ldots (\boldsymbol{x}_n, y_n)\}$, where the pairs $(\boldsymbol{x}_i, y_i)$ – with $\boldsymbol{x}_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$ – are independent realizations of the random variable pair (X, Y). Additionally, our modeling choices and assumptions define a hypothesis class, $\mathcal{H}$, that contains the functions from $\mathcal{X}$ to $\mathcal{Y}$ that can be made with the models that we use. When using linear models, for example, the hypothesis class contains all linear functions.

The general idea in supervised learning is to define a loss function and to then find the model that minimizes that loss function. The loss function,

$$l : \mathcal{H} \times \mathcal{X} \times \mathcal{Y} \to \mathbb{R} : (f, \boldsymbol{x}, y) \mapsto l(f(\boldsymbol{x}), y),$$

calculates the loss for a model by measuring the discrepancy between the predictions of the model and the actual observations.

Within the structural risk minimization framework, the goal is to find the optimal model in our hypothesis class, f^*, defined as the model that minimizes

the expected loss:

$$f^* := \arg\min_{f \in \mathcal{H}} \int l(f(X), Y) d\mathcal{P}_{X,Y}. \tag{1.1}$$

However, we cannot evaluate this expectation directly, since $\mathcal{P}_{X,Y}$ is unknown. An alternative is therefore to minimize the empirical loss function of our training data, possibly with a regularization term that favors simpler models. The obtained function, $\hat{f}$, is defined as

$$\hat{f} := \arg\min_{f \in \mathcal{H}} \left(\sum_{i=1}^{n} l(f(\boldsymbol{x}_i), y_i) \right). \tag{1.2}$$

Correctly defining the loss function is crucial, and it should align well with the desired objective. In a regression setting, for instance, optimizing the mean squared error as the loss function penalizes outliers more than optimizing the mean absolute error. Whether having a model that is sensitive to outliers is desirable, depends on the problem.

The specific choice of loss function yields distinct advantages and challenges and directly determines how the model learns from the data. It is therefore important that this choice reflects the specific nuances and demands of the problem at hand.

1.2 Neural Networks

Among the many different machine-learning models, neural networks are arguably the most popular and the most successful. Examples of applications using these models include modern large-language models and self-driving cars.

The basics of a neural network are straightforward. In a standard feed-forward network, the individual operations consist of matrix multiplications and additions. The input is multiplied by a weight matrix, followed by the addition of a bias vector. The elements of this vector are then fed through a non-linear activation function, σ, to produce the output of the hidden layer. This process – multiplication by a weight matrix, addition of a bias vector, and application of an activation function – is repeated multiple times across subsequent layers to produce the final output of the network.

The activation functions facilitate the modeling of non-linear functions. Without them, the output would simply be a linear combination of the input. The activation function of the final layer can be used to enforce the output to be in a certain domain. When modeling a probability, a sigmoid activation function, $\sigma(x) = \frac{1}{1+e^{-x}}$, can be used to restrict the output to the interval $[0, 1]$.

1.2.1 Training

Neural networks are parametrized by weight matrices and bias vectors. We denote the parametrized network with $f_{\boldsymbol{\theta}}$, where $\boldsymbol{\theta}$ represents the full set of model parameters. Optimizing a network involves a three-step gradient back-propagation process: a forward pass, loss calculation, and a backward pass.

During the forward pass, the inputs are fed through the network to obtain predictions. The empirical loss, $\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{n} l(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), y_i)$, is then evaluated. Finally, the gradients with respect to the individual parameters, θ_j, are computed in a backward manner – hence the name back-propagation – using the chain rule, starting from the output layer and working backwards to the input.

This method enables the computation of the gradients for all parameters with only a single forward pass. Considering that modern neural networks can easily have millions of parameters, this efficiency is crucial.

Intuitively, we would want to calculate the gradients using the entire data set. However, especially for larger data sets, it is not feasible to store the entire data set in memory. The common solution is to randomly divide the data set into smaller parts, called *batches*. Gradients are computed using a single batch, after which the model is updated before proceeding to the next batch. Training typically spans multiple *epochs*, each representing a full pass over the data set. For instance, with a data set of 100 data points, a batch size of ten, and training for three epochs, the model undergoes 30 updates, using each data point three times.

Unlike most classical models, the optimization of neural networks is inherently stochastic. Training the model multiple times results in slightly different models each time. This randomness is mainly due to the random initialization of weights and biases prior to training, and due to the random ordering of the batches during training.

1.2.2 Architectures

While the basic operations are straightforward, many intricate models can be made from them, capable of carrying out a broad variety of tasks. We briefly introduce a few key architectures to illustrate the vast range of possibilities these networks can offer.

Convolutional Neural Networks (CNNs, LeCun et al. 1989) use trainable convolutional filters combined with pooling layers – layers that reduce the dimensionality by taking the average or maximum of a group of pixels – to effectively handle images, which typically have very large input dimensions.

Autoencoders (Hinton and Salakhutdinov, 2006) first reduce, or encode, the input to a significantly lower dimension, and then attempt to recreate, or decode, the original input. This architecture can be used for dimensionality reduction, as a generative model, or to detect anomalies – the idea being that unfamiliar inputs will not be reconstructed as well.

Generative Adversarial Networks (GANs, Goodfellow et al. 2014a) are designed as a pair of competing networks: a generator that creates images aiming to be indistinguishable from real images, and a discriminator that tries to distinguish between real and generated images. This competitive process improves the quality of the generated images over time, making GANs powerful tools for image generation.

Long Short-Term Memory (LSTM, Hochreiter and Schmidhuber 1997) networks are specifically designed to retain long-term dependencies in the input sequence. This mechanism is useful in cases such as natural language processing where the start of a sentence can influence its end.

Recently, the performance of LSTMs has been surpassed by transformers (Vaswani et al., 2017). This type of network, which forms the basis of popular large language models like GPT (Brown et al., 2020), is highly effective due to its ability to process entire sequences of data in parallel instead of sequentionally. Consider the sentence *"Sarah likes her hat very much"*. A transformer is able to process each word in this sentence in parallel. This facilitates training on substantially larger data sets. Additionally, an attention mechanism allows the model to focus on different parts of the input depending on the context. For instance, in the example sentence, the model is able to learn that the word *"her"* refers to *"Sarah"*.

This modest list of architectures illustrates that machine learning, and partic-

ularly some variation of a neural network, is extensively used for a wide variety of tasks. Additionally, these models differ significantly from classical models in both size and optimization: neural networks can have millions or even billions of parameters, and their optimization is often non-deterministic. These challenges have led to the development of a research field focused on novel uncertainty-quantification methods for these models.

1.3 Uncertainty Quantification

The remainder of this chapter focuses on introducing the main topic of this thesis: uncertainty quantification. We adopt a source-based approach to this topic. By exploring the entire process from data acquisition to prediction and questioning what sources of uncertainties arise and what solutions exist, we provide a broad overview of various existing uncertainty-quantification techniques.

We do not aim to provide a complete survey of the field; for a comprehensive review, we refer the reader to Gawlikowski et al. (2023), He and Jiang (2023), and Abdar et al. (2021). More attention is given to certain approaches, such as ensembling, that are featured more prominently in the remainder of this thesis, while other approaches, such as conformal prediction, are only discussed briefly. Additionally, various approaches, such as MC-dropout, are explained in more detail in the individual chapters.

The first source of uncertainty is found before the building of the model begins, during the data acquisition (Van Giffen et al., 2022). No matter how sophisticated the uncertainty-estimation method of the model is, if the training data contains biases or errors, the predictions will contain unforeseen errors.

This following list of typical issues during the data selection process is by no means exhaustive, as it is not the main focus of this thesis. For a complete overview, we refer the reader to Mehrabi et al. (2022). However, it is important to acknowledge the importance of the data acquisition process and the potential biases that may arise from it. It is easy to get lost in – or be convinced by – the intricate mathematics and the thousands of lines of code found in modern machine-learning models. However, no model can perfectly correct for biased data.

Selection bias occurs when the data is not representative of the population for which the model is intended. A face recognition program that is trained on

images from people from a certain region may fail on people from a different region. Alternatively, a data set may be selected by sending out questionnaires to users of an application. However, since these people already use the application, they are more likely to have a favorable opinion.

Another typical example is *survivorship bias*, where the data set contains an over-representative number of survivors. The classical example dates back to the Second World War (Mangel and Samaniego, 1984). Returning aircrafts were examined to determine where they should be reinforced. Initially, it might seem logical to reinforce areas most frequently hit on aircraft returning from missions. However, these were aircraft that made it back and the locations of the bullet holes do therefore not accurately expose the weaknesses of the plane.

When analyzing online reviews, there is often a surprisingly large number of one- and five-star reviews, an example of *reporting bias*. People are less likely to put in the effort to share that they found a certain product *"perfectly average"*. If a model is trained on this data, it will not adequately predict three-star ratings.

A notable example of biased data leading to a biased model is Amazon's recruitment tool (Dastin, 2022). In 2014, the company began developing a model to automate parts of the recruitment process. The model was trained on actual resumes from the previous ten years. However, since more men than women choose working in the tech industry, the model primarily received male resumes. This caused the model to inadvertently learn that male candidates were preferable. Although the model did not have direct access to the candidates' gender, it could infer this from specific word choices more prevalent among male candidates and the presence of the words *woman* or *women* in the resume. Consequently, the model rated equivalent female candidates lower.

This example underscores the importance of high-quality training data. However, even with a perfect, unbiased data set, uncertainties remain. The uncertainty in the predictions of a model is called the *predictive uncertainty*. This uncertainty is a combination of three other sources: *data uncertainty*, *model uncertainty*, and *distributional uncertainty*. Figure 1.2 illustrates these sources of uncertainty. In the remainder of this section, we delve deeper into these terms and discuss several existing methods to address them.

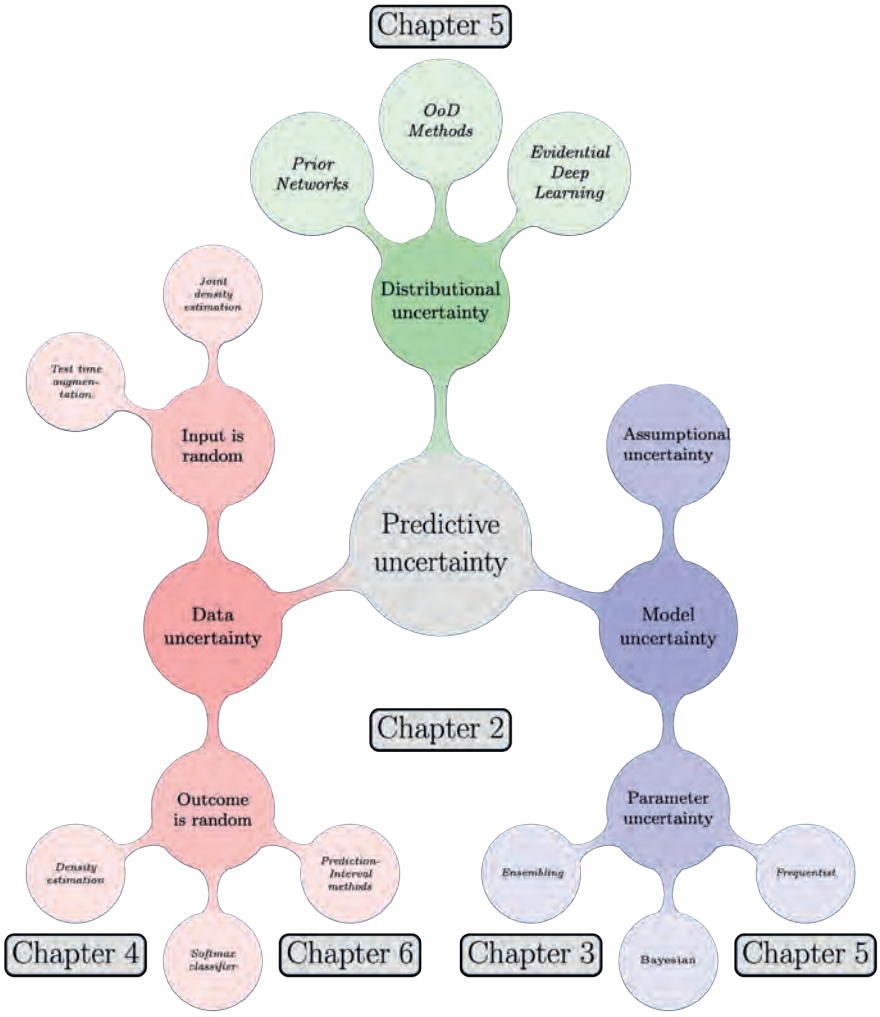


Figure 1.2: This figure illustrates the various sources of uncertainty that jointly make up predictive uncertainty as well as various popular techniques (in italic font) to account for these sources. Firstly, there is *model uncertainty*, both because of the assumptions that are made and because the parameters are determined on a finite data set. However, even with a perfect model, we still have *data uncertainty* because the problem in question may be inherently stochastic. Lastly, we have *distributional uncertainty* due to a mismatch between the data that the model is trained on and the data that the model is applied to. The grey boxes illustrate the specific areas of the field to which the different chapters of this thesis contribute.

1.3.1 Data Uncertainty

There is inherent uncertainty in predictions because the underlying problem is typically stochastic. The data is random due to the variability in both the covariates and the outcomes. Consider a model predicting an individual's height based solely on their weight. Even with data on the heights and weights of millions of people and an ideal model, absolute certainty in the predictions for a new individual is unattainable because the distribution of $Y \mid X$ inherently has variance.

This type of uncertainty is generally referred to as *aleatoric* or *irreducible* uncertainty (Hüllermeier and Waegeman, 2021; Kendall and Gal, 2017). Collecting more data does not diminish this uncertainty. However, adding more covariates can generally reduce it. For instance, if age data were collected, the conditional random variable $Y \mid X, A$, where A is a random variable representing age, has a lower variance.

Since this uncertainty in the outcome is irreducible, a common approach is to model the density of $Y \mid X$ directly. In classification problems, a categorical distribution is typically assumed, and the neural network predicts the probabilities of each class. In a regression context, a certain distribution (usually a normal distribution) is assumed, and the network estimates the parameters of that distribution.

More specifically, we assume that the density of $Y \mid X = \boldsymbol{x}^*$ is given by

$$p(y \mid \boldsymbol{x}^*) = p(y \mid \boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{x}^*)),$$

where $\boldsymbol{\mu}_{\boldsymbol{\theta}} : \mathbb{R}^p \to \mathbb{R}^q : \boldsymbol{x} \mapsto \boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{x})$ represents the network, which is parametrized by $\boldsymbol{\theta}$ and maps the p-dimensional input $\boldsymbol{x}$ to the q-dimensional distributional parameter vector. Concretely, in a classification setting the network may output a 10-dimensional vector containing the class probabilities, and in a regression setting, the 2-dimensional vector containing the mean and variance that parametrize a normal distribution. This latter type of network is referred to as a Mean-Variance-Estimation (MVE) network (Nix and Weigend, 1994; Seitzer et al., 2021; Skafte et al., 2019). In Chapter 4, we present improvements to this type of network.

Other methods do not make any distributional assumptions but directly output a prediction interval. A notable example is the method Quality-Driven ensembles (Pearce et al., 2018), which optimizes a specific loss function aiming for correct marginal coverage – ensuring that, on average, the appropriate fraction

of targets falls inside the prediction intervals – while also being as narrow as possible. Other works use neural networks for quantile regression (Koenker and Bassett Jr, 1978) by optimizing a smooth approximation of the pinball loss (Cannon, 2011; Xu et al., 2017). In Chapter 6, we present a novel smooth approximation that is tailored for models that use the second derivative of the loss during the optimization process.

The randomness of the input also affects the predictive uncertainty. For instance, a measurement sensor may introduce noise, resulting in slightly varied covariates upon repeated measurements. For images, capturing the picture at a slightly different angle can affect the input.

Test-time augmentation is a practical approach to quantify how random input variations affect predictive uncertainty. Instead of feeding the network a single input, multiple perturbed versions of this input are processed through the network. An early example of this method is provided by Ayhan and Berens (2022), who employed basic transformations such as reflections, random crops, random resizing, and adjustments in settings such as brightness and saturation, among others. It is important that the perturbations are realistic (Shanmugam et al., 2021) and there is a significant amount of research on finding good perturbations (Lyzhov et al., 2020; Kim et al., 2020; Cubuk et al., 2019).

A different approach is to explicitly learn the joint density $p(y, \boldsymbol{x})$ instead of the conditional density $p(y \mid \boldsymbol{x})$. Popular approaches include normalizing flows (Kobyzev et al., 2021), masked autoregressive flows (Papamakarios et al., 2017), and variational autoencoders (Kingma et al., 2015; Suzuki et al., 2016).

1.3.2 Model Uncertainty

When the model outputs a prediction interval, class probability, or distributional parameter, we do not know if the model is correct; we refer to this type of uncertainty as model uncertainty, *epistemic* uncertainty, or reducible uncertainty.

The model uncertainty can be further decomposed into two parts, as illustrated in Figure 1.3: *assumptional uncertainty*, which is a consequence of the various modeling choices and assumptions that are made; and *parameter uncertainty*, which is a consequence of fitting parameters on a finite set of random data.

Various choices or assumptions are made, either implicitly or explicitly. When choosing a specific architecture and training procedure, we implicitly assume

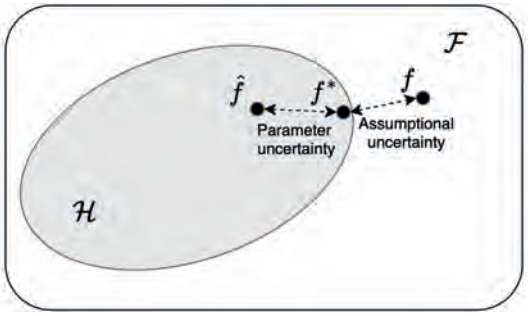


Figure 1.3: This figure, inspired by Figure 4 in Hüllermeier and Waegeman (2021), illustrates the different components that make up the model uncertainty. All possible functions that our neural network can make are denoted with $\mathcal{H}$. This $\mathcal{H}$ is contained in $\mathcal{F}$, all possible functions from $\mathcal{X}$ to $\mathcal{Y}$. On the one hand, we do not know if the true function, f, is inside our hypothesis class. We refer to this as *assumptional* uncertainty. On the other hand, we do not know if the $\hat{f}$ (Equation 1.2) that we arrived at after training is even the best function in our hypothesis class, f^* (Equation 1.1). We refer to this as *parameter* uncertainty.

a certain hypothesis class containing all possible networks that can be made given these modeling assumptions. As an example, think of a simple linear model. We make the assumption that the true function is linear and attempt to find the optimal parameters within our hypothesis class. Additionally, we may assume that $Y \mid X$ follows a normal distribution, which is not necessarily correct. In this work, we refer to this source as *assumptional* uncertainty.

Hüllermeier and Waegeman (2021) refer to this term as *model* uncertainty. However, the term model uncertainty is very frequently used to denote either the entire epistemic uncertainty or the part that we call parameter uncertainty. Therefore, we chose to introduce the new term assumptional uncertainty.

This uncertainty can be substantial. The true function, f, may not be within our hypothesis class. We might use a linear model when the true function is quadratic. Therefore, even if we have the perfect parameters within our hypothesis class, we still may have a large error.

It is very difficult to quantify, or even capture, this assumptional uncertainty (Hüllermeier and Waegeman, 2021). An argument can be made that since neural networks are universal approximators (Lu et al., 2017), meaning that suffi-

ciently large networks can fit virtually any function, the true function should typically be within our hypothesis class. While this argument may be somewhat over-optimistic, neural networks are very flexible, and a pathology such as using a linear function for a quadratic problem is unlikely.

Even if the true function is contained in the hypothesis class, we still have parameter uncertainty. Both because the optimization procedure is random for neural networks and because we are training on a finite set of random data, the model parameters that we end up with are likely not the optimal ones.

There is an interplay between parameter uncertainty and assumptional uncertainty. It is always possible to make a more flexible model by adding more parameters and increasing the training time, thereby increasing the hypothesis class and making more likely to contain the true function f. However, this increases the parameter uncertainty since it becomes more challenging to find to optimal hypothesis in this larger hypothesis class.

There exists a wide range of methods that aim to quantify parameter uncertainty. The majority of these methods can be classified as a Bayesian method, frequentist method, or ensembling method. The latter can arguably also be motivated from a Bayesian or frequentist perspective but due to their popularity, we discuss it here as a separate class.

Bayesian methods: The general idea behind Bayesian uncertainty-estimation methods is to place a prior distribution, $\pi(\boldsymbol{\theta})$, on the parameters of the neural network (MacKay, 1992a; Neal, 2012). Subsequently, Bayes' formula is applied to obtain the posterior distribution of the weights given the data set $\mathcal{D}$:

$$\pi(\boldsymbol{\theta} \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \boldsymbol{\theta})\pi(\boldsymbol{\theta})}{p(\mathcal{D})}, \tag{1.3}$$

where the term $p(\mathcal{D} \mid \boldsymbol{\theta})$ is calculated using the conditional density that the neural network provides. The posterior density is then used to calculate the posterior predictive density:

$$p(y \mid \boldsymbol{x}^*, \mathcal{D}) = \int p(y \mid \boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{x}^*))\pi(\boldsymbol{\theta} \mid \mathcal{D})d\boldsymbol{\theta}.$$

Unfortunately, the denominator in Equation (1.3) is typically intractable, making it impossible to evaluate the posterior exactly. Several solutions exist to combat this problem.

A first approach is to sample from the posterior. Although the denominator is intractable, the posterior is known up to a constant, which makes it possible to sample from it. Examples of sampling algorithms are importance sampling, rejection sampling, and MCMC sampling (Bishop, 2006).

Another approach is to use a second-order approximation of the posterior (MacKay, 1992b; Kass et al., 1991). This so-called Laplace approximation requires the inversion of the high-dimensional Hessian, which is a considerable task. Several works focus on improving this approximation (Martens and Grosse, 2015; George et al., 2018; Lee et al., 2020).

A slightly different approach, called variational inference (VI) (Hinton and Van Camp, 1993), is to approximate the true posterior $p(\boldsymbol{\theta} \mid \mathcal{D})$ with a variational approximation $q_{\boldsymbol{m}}(\boldsymbol{\theta})$, parametrized by $\boldsymbol{m}$. The network is given the learning objective to find the approximate posterior distribution closest to the true posterior distribution by minimizing $\mathrm{KL}(q_{\boldsymbol{m}}(\boldsymbol{\theta})||p(\boldsymbol{\theta} \mid \mathcal{D}))$.

The most prominent VI method is Monte-Carlo dropout (MC-dropout) (Gal and Ghahramani, 2016). Dropout is a regularization technique where the weights of the network are set to zero with a given probability during each forward pass. The authors showed that, when putting a specific prior on the weights, training a network with dropout enabled and while using a specific form a regularization is equivalent to minimizing $\mathrm{KL}(q_{\boldsymbol{m}}(\boldsymbol{\theta})||p(\boldsymbol{\theta} \mid \mathcal{D}))$, where $q_{\boldsymbol{m}}(\boldsymbol{\theta})$ is the density of the distribution that is implied by randomly dropping the weights of the network. By keeping the random dropping of the weights enabled after training, multiple forward passes through the network result in samples from the approximate posterior.

Frequentist methods: Instead of Bayesian statistics, classical parametric statistics can also be leveraged to obtain estimates of the parameter uncertainty. The model can be seen as a parametric model, and we can therefore evaluate the likelihood as a function of the model parameters, $\boldsymbol{\theta}$. Subsequently, we can maximize the likelihood to find maximum-likelihood estimator, $\hat{\boldsymbol{\theta}}_{\mathrm{MLE}}$. Asymptotic theory gives us, under various conditions, the variance of $\hat{\boldsymbol{\theta}}_{\mathrm{MLE}}$ (e.g., see Seber and Wild 2003). This variance can then be converted to a variance of the model predictions using the delta method. Various approaches rely on this basic idea (Kallus and McInerney, 2022; Nilsen et al., 2022; Deng et al., 2023; Khosravi et al., 2011). In Chapter 5, we leverage the likelihood ratio to obtain confidence intervals with desirable qualitative properties.

Ensembling methods: The idea behind ensembling is straightforward: multiple networks, or ensemble members, are trained and the variance in their predictions is used as an uncertainty estimate. This comes with the added benefit that the average of these ensemble members typically gives a more stable estimator.

Early work in fact introduced ensembling as a method to improve model performance (Hansen and Salamon, 1990). However, later works also utilized the potential to obtain uncertainty estimates from the ensemble members (Heskes, 1997), with the currently most popular method being Deep Ensembles (Lakshminarayanan et al., 2017).

It is crucial that the individual ensemble members have enough diversity. Popular approaches to achieve this include training various members with different initializations, training on shuffled versions of the data, training on resampled data (Bishop, 2006), or even using entirely different architectures (Wenzel et al., 2020; Herron et al., 2020). In Chapter 3, we demonstrate how incorporating a missing source of variance into the ensemble members improves the resulting confidence and prediction intervals.

A clear downside of ensembling is the computational cost. Multiple networks need to be trained, which may present a bottleneck both during training and when making predictions. Various works therefore focus on reducing this computational cost. Approaches include pruning, where redundant ensemble members are removed (Guo et al., 2018); distillation, where a single network is trained to represent the same information as the entire ensemble (Hinton et al., 2015); and techniques such as sub-ensembles (Valdenegro-Toro, 2023) and batch-ensembles (Wen et al., 2020) where parts of the network are shared between the ensemble members.

1.3.3 Distributional Uncertainty

The final source of uncertainty results from a potential mismatch between the distribution of the training data and the data of interest. This mismatch is typically referred to as a *data set shift* (Quiñonero-Candela, 2009) and the resulting uncertainty is called *distributional uncertainty* (Malinin and Gales, 2018). Some of the biases during data acquisition, discussed earlier in this section, can cause such a mismatch.

Multiple methods aim to tackle this problem. Notable examples include prior networks (Malinin and Gales, 2018) and evidential networks (Sensoy et al.,

2018). Both papers start from a Bayesian classification setting where the goal is to predict the class probabilities of a multinomial distribution. However, instead of directly predicting the class probabilities, the network outputs the parameters of a Dirichlet distribution. This Dirichlet distribution is interpreted as a distribution over the class probabilities. The density of the posterior predictive distribution in this framework is given by

$$p(y \mid \mathcal{D}, \boldsymbol{x}^*) = \int \int \underbrace{p(y \mid \boldsymbol{\mu})}_{Data} \underbrace{g(\boldsymbol{\mu} \mid \boldsymbol{\alpha}_{\boldsymbol{\theta}}(\boldsymbol{x}^*))}_{Distributional} \underbrace{\pi(\boldsymbol{\theta} \mid \mathcal{D})}_{Parameter} d\boldsymbol{\mu} d\boldsymbol{\theta},$$

where $\boldsymbol{\alpha}_{\boldsymbol{\theta}}$ represents the network that outputs the parameters of the Dirichlet distribution with density function g. The three different densities account for the three different sources of uncertainty.

This extra distribution provides a new layer of flexibility. Consider a classification problem with three classes. A $\text{Dir}(0, 0, 0)$ distribution corresponds to a scenario where every combination of classes is equally likely. In contrast, a $\text{Dir}(10, 10, 10)$ distribution corresponds to a scenario where all three classes are equally likely. In this case, the model knows that all class probabilities should be $1/3$. In the first scenario, it knows that it does not know what the probabilities should be.

Although prior networks and evidential networks may seem similar, the optimization process is in fact quite different. Evidential networks are trained by optimizing the likelihood of the data plus a regularization term. This regularization term forces the model towards a more uncertain state. One could argue, however, that this mainly estimates parameter uncertainty and recent work indicates that it is not extremely accurate at that task (Juergens et al., 2024).

Prior networks, on the other hand, make use of Out-of-Distribution (OoD) data. For example, pictures of farm animals when classifying between cats and dogs. When the network is fed a regular input, a cat or a dog, the loss function is the KL divergence between the predicted Dirichlet distribution and a Dirichlet distribution that is concentrated on the correct class. When the network is fed a farm animal, the loss function is the KL divergence between the predicted Dirichlet distribution and a flat Dirichlet distribution. In other words, the model is taught to recognize scenarios that it is not familiar with and to communicate this via a flat Dirichlet distribution. Both prior and evidential networks also have extensions for regression (Amini et al., 2020; Malinin et al., 2020).

There are many more methods besides these two approaches. OoD detection is an entire field with numerous different methods (Lee et al., 2018; Hendrycks and Gimpel, 2018; Hendrycks et al., 2018). OoD methods do not necessarily incorporate the distributional uncertainty into their estimate, like prior and evidential networks, but often simply detect when a new input is unfamiliar. This information can then be used to either withhold predictions or issue warnings, improving the model's reliability in practical applications. Additionally, incorporating OoD detection can improve system safety, particularly in applications where unexpected inputs can lead to harmful outcomes.

1.3.4 Post-processing

After having obtained a predictive uncertainty estimate by accounting for the various sources of uncertainty, this uncertainty estimate may not be perfectly calibrated. In fact, most modern neural networks are overconfident in both regression (Dheur and Taieb, 2023) and classification tasks (Guo et al., 2017).

Several post-processing techniques are available to recalibrate the predictive uncertainty estimates (Guo et al., 2017). In classification, an example is temperature scaling, a variant of Platt scaling (Platt et al., 1999), where the predicted logits are divided by a constant. This adjustment fine-tunes the probabilities, forcing them closer to 0.5 when the constant is greater than 1, or toward 0 or 1 when the constant is less than 1. In regression, a concrete example is quantile recalibration (Kuleshov et al., 2018), which involves learning a transformation for the quantiles. This transformation ensures that the recalibrated quantiles achieve correct empirical coverage on a validation set.

Conformal prediction (Shafer and Vovk, 2008) is related to calibration but fundamentally different. Unlike methods that calibrate an uncertainty estimate using a validation set, the objective of conformal inference is to construct a prediction interval directly from the validation set. In this approach, nonconformity scores, which measure the deviation between the predictions and the observations, are calculated for each point in the validation set. These scores are then used to construct prediction intervals that come with finite-sample coverage guarantees.

1.4 Structure of This Thesis

In this introduction, we provided a broad overview of the field of uncertainty quantification for machine learning. We divided the modeling process into three parts. First, during the data acquisition phase, various biases in the data can result in a biased model. Next, a model is built based on assumptions and modeling choices, which makes predictions. The uncertainty in these predictions arises from various sources, each requiring different solutions. Finally, the predictive uncertainty estimate can be calibrated in a post-processing step.

The rest of this thesis contributes to various aspects of the second step: obtaining the uncertainty estimates. In Figure 1.2, we indicate where the following chapters of this thesis are situated within the field.

Chapter 2: This chapter discusses the current methodology of testing uncertainty estimates in a regression setting. We identify several flaws. First of all, the current approaches do not easily allow the comparison of methods that output a likelihood with methods that directly output prediction intervals. Secondly, only predictive uncertainty can be evaluated, making it impossible to evaluate confidence intervals directly. Finally, the manner in which prediction intervals are evaluated is marginal instead of conditional. We present a simulation-based evaluation procedure that remedies these shortcomings.

Chapter 3: The following chapter improves upon a highly popular ensembling approach, Deep Ensembles (Lakshminarayanan et al., 2017), in a regression setting. The original authors mentioned that a source of uncertainty is missing in their approach but that incorporating it resulted in a worse performance. We present a method to incorporate this missing source without affecting the performance of the model. By using the evaluation procedures presented in Chapter 2, we demonstrate that this results in improved uncertainty estimates.

Chapter 4: Where the previous chapter focused on improving the model uncertainty estimate, this chapter focuses on improving the data uncertainty estimate. The individual ensemble members used in the previous chapter are MVE networks. The predicted variance is used as a data uncertainty estimate. Especially for noisier data sets, this estimate can be the dominant part in the

predictive uncertainty, and it is therefore crucial that this predicted variance is accurate. Novel ways to improve the training of these MVE networks are discussed. Most notably, we demonstrate that the parts of the networks that estimate the variance and the mean should be regularized separately, and we experimentally verify the claim that a warm-up period – a period where the predicted variance is kept fixed while the part of the network that estimates the mean is trained – leads to significant improvements.

Chapter 5: In Chapter 3, we noticed that the coverage would sometimes be extremely poor in certain regions of the data. Upon further inspection, we noticed that the networks were biased in those regions. Since the prediction and confidence intervals are centered around the prediction, this results in very low coverage. We also noticed that the intervals would not always expand sufficiently in regions where the data was more sparse, especially when interpolating. To address these issues, we present an entirely new type of approach to obtain the model uncertainty estimates by applying the likelihood-ratio testing procedure to neural networks. This new approach asks the intuitive question *"at this new location, what values can be reached while still explaining the data well?"*. By answering this question, we obtain confidence intervals with very desirable qqualitative properties.

Chapter 6: The Dutch power-grid operator Alliander approached us with an interesting case. The company was using an XGBoost model (Chen and Guestrin, 2016) to predict the loads on substations of the grid. However, using this model for quantile regression is non-trivial due to the incompatibility between the second-order approximation of the loss function in XGBoost and the pinball loss used in quantile regression, which has a second derivative of zero. We present a novel smooth approximation of the pinball loss that is specifically tailored to the needs of XGBoost.

Chapter 2

Evaluating Uncertainty Estimates for Regression

This chapter is based on the paper entitled *"How to Evaluate Uncertainty Estimates in Machine Learning for Regression?"* (Sluijterman et al., 2024a), which has been published in *Neural Networks*. Contrary to the other chapters in this thesis, this chapter does not focus on any specific method to produce uncertainty estimates but rather on how to evaluate these uncertainty estimates in a regression setting.

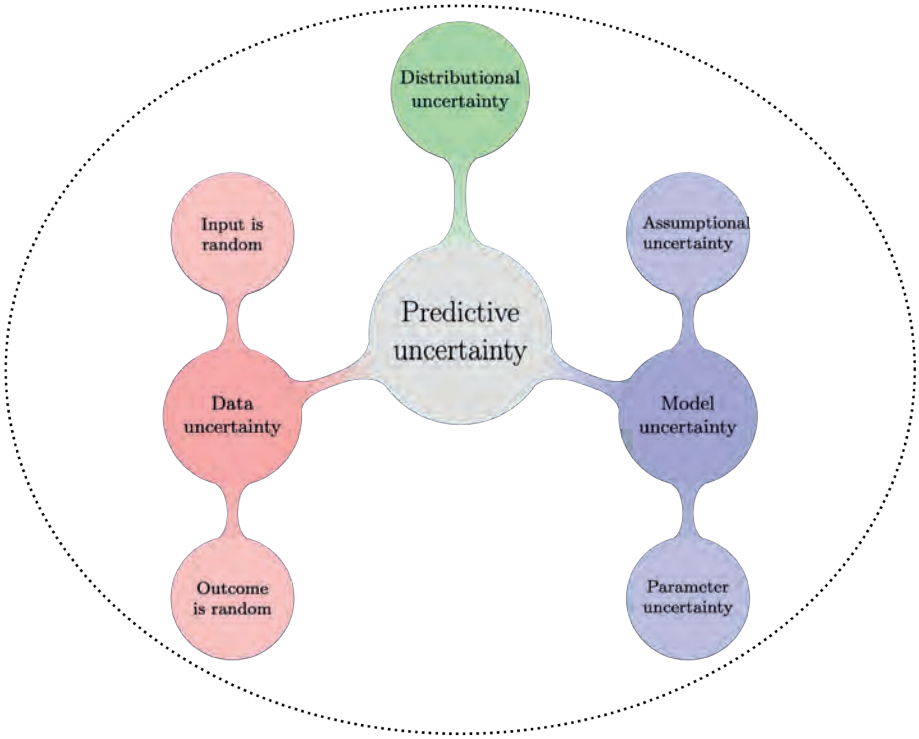


Figure 2.1: The scope of Chapter 2. Rather than focussing on individual uncertainty-quantification methods, this chapter focusses on how to *evaluate* uncertainty estimates in a regression setting.

2.1 Introduction

Neural networks are, among other things, currently being used in a wide range of regression tasks, covering many different areas. It has become increasingly clear that it is essential to have uncertainty estimates to come with the predictions (Gal, 2016; Pearce, 2020). Uncertainty estimates can be used to make confidence intervals or predictions intervals at a given $100 \cdot (1-\alpha)\%$ confidence level. We define these intervals more precisely in Section 2.2, but the intuition is as follows. The probability that the true function value falls inside a confidence interval (CI) should be $100 \cdot (1-\alpha)\%$. A $100 \cdot (1-\alpha)\%$ prediction

interval (PI) is constructed such that the probability that an observation falls inside this interval is $100 \cdot (1 - \alpha)\%$. The two desirable characteristics of a PI or CI are that they cover the correct fraction of the data while being as small as possible (Khosravi et al., 2011). At the moment, a common approach to test a PI is to use a previously unused part of the data and then check which fraction of the observations falls inside the corresponding PIs. This fraction is called the Prediction Interval Coverage Probability (PICP) and is widely used to asses the quality of prediction intervals (Pearce et al., 2018; Kabir et al., 2023; Khosravi et al., 2011; Pearce et al., 2020; Su et al., 2018; Lai et al., 2022; Zhang and Fu, 2023; Chen et al., 2023; Dewolf et al., 2023; Van Beers and De Visser, 2023; Zhang et al., 2023; Zheng and Zhang, 2023).

Where some methods, such as quantile regression, output PIs directly, others output a density. The testing procedure for these methods is largely influenced by the article of Hernández-Lobato and Adams (2015). To compare their proposed method, probabilistic backpropagation, with existing alternatives, they came up with a novel testing procedure. Their testing procedure uses ten publicly available real-world data sets and evaluates the loglikelihood on an unseen test set using a fixed training procedure. We explain this procedure in detail in Section 2.2.3. This setup allowed for an effective way of comparing different methods. Many authors subsequently used this setup as a benchmark to compare their uncertainty estimation methods with those of others (Gal and Ghahramani, 2016; Lakshminarayanan et al., 2017; Mancini et al., 2020; Liu and Wang, 2016; Salimbeni and Deisenroth, 2017). Recent works on PIs also use these data sets to calculate the PICP score, see for example Pearce et al. (2018, 2020); Su et al. (2018); Lai et al. (2022); Nourani et al. (2023).

In this chapter, we demonstrate, both theoretically and through simulation experiments, that both testing methodologies fail to accurately determine the quality of a prediction or confidence interval. First of all, both approaches share the problem that while a good predictive performance may imply a good PI, we have no insight in the performance of the CIs. When using real-world data sets, it is impossible to directly evaluate the CI since the true function that generated the data is not known. Additionally, when using the loglikelihood, it is impossible to compare methods that output a density with methods that directly output a PI. Lastly, covering the correct fraction of points in a test set does not test coverage correctly and even if it did, it does not guarantee that the PIs are correct for individual data points. As a result, it is possible to select the wrong method, that may not produce sensible PIs or CIs for individual points, as the best.

The fact that an average coverage does not guarantee individual level coverage was also noted by various other authors (Kuleshov et al., 2018; Zhao et al., 2020; Chung et al., 2021). In this chapter, we will see that the individual level coverage can indeed be very poor, even with a good coverage on a test set, and we discuss testing approaches to test the individual level coverage.

This chapter consists of six sections, this introduction being the first. Section 2.2 gives the theoretical framework that is needed to properly discuss uncertainty. We precisely define what we mean with coverage and explain the loglikelihood and PICP testing approach. Section 2.3 gives theoretical drawbacks of these current testing methodologies. In Section 2.4, we propose a simulation-based approach to combat these drawbacks. Section 2.5 verifies these concerns using simulation results by comparing the PICP approach to a simulation-based one. Finally, in Section 2.6, we summarize the conclusions and give suggestions for future work.

2.2 Defining the Uncertainty Framework and Testing Methodology

This section consists of three parts. First, we go through the terminology necessary to properly discuss uncertainty. In the second and third part, we explain the two most popular testing procedures, evaluating the loglikelihood and evaluating PICP. For an overview of the various methods to obtain uncertainty estimates, we refer to Khosravi et al. (2011) for early work, and to the reviews by He and Jiang (2023); Hüllermeier and Waegeman (2021); Gawlikowski et al. (2023); Kabir et al. (2018) for more recent contributions.

2.2.1 Defining Uncertainty

Before we can talk about quantifying uncertainty, we need to define precisely what we mean with the term. Throughout this chapter we assume a regression setting. We have a data set $\mathcal{D} = ((\boldsymbol{x}_1, y_1), \dots (\boldsymbol{x}_n, y_n))$ that is a set of n independent realizations of the random variable pair (X, Y). We assume that $\boldsymbol{x} \in \mathbb{R}^d$ and $y \in \mathbb{R}$. The regression situation we are considering is such that

$$Y \mid X = \boldsymbol{x} \sim \mathcal{N}\left(f(\boldsymbol{x}), \sigma^2(\boldsymbol{x})\right),$$

where $f(\boldsymbol{x})$ is the true regression function and $\sigma^2(\boldsymbol{x})$ is the variance of the additive noise. This is equivalent to the typical description of a regression setting where we assume that our observations are a combination of an unknown function and a (normally distributed) noise term:

$$y_i = f(\boldsymbol{x}_i) + \epsilon_i.$$

Suppose we train a neural network (or any other type of model) to approximate $f(\boldsymbol{x})$ with $\hat{f}(\boldsymbol{x})$. Assuming that our data is representative, we have two types of uncertainty. In the first place, we are unsure about the quality of our estimate $\hat{f}(\boldsymbol{x})$. We refer to this as the *model* uncertainty (in other works sometimes referred to as *epistemic* uncertainty). On the other hand, if we want to predict a new y-value, we face an additional source of uncertainty due to the inherent randomness of ϵ_i. If the distribution of ϵ_i given $\boldsymbol{x}_i$ does not depend on $\boldsymbol{x}_i$, we have *homoscedastic* noise. If the distribution of ϵ_i given $\boldsymbol{x}_i$ depends on $\boldsymbol{x}_i$, we have *heteroscedastic* noise. The uncertainty due to ϵ_i is often referred to as the *aleatoric uncertainty*, *irreducible variance*, or *data noise variance*. In this chapter, we use the terminology *data noise variance*.

For an application of a model in practice, it is often necessary to quantify this uncertainty. Ultimately, one may want to do this by giving an accompanying confidence or prediction interval. We now take some time to define these concepts more precisely as this will be the cornerstone of the discussion in the rest of this chapter.

We take a fixed-covariates viewpoint. With $\mathscr{Y}$, we denote the random variables whose realization is an entirely new set of targets $(y_1, \ldots, y_n)$. This is equivalent to realizations of $Y \mid X = \boldsymbol{x}_i$. We add this extra notation to distinguish taking an expectation over an entire new set of targets and taking an expectation over a single observation pair (X, Y). For a given set of covariates, we have an estimator that is a function of the targets. In this case, we may want to take the expectation over $\mathscr{Y}$. In a machine learning context, the predictor $\hat{f}$ often does not only depend on the data set but also on random effects, such as the weight initialisation of a neural network and the ordering in which the training examples are presented. With U, we denote a random variable that expresses randomness in a training process. We are now ready to define a confidence and prediction interval.

Definition 1. *A* $(1-\alpha)\cdot 100\%$ *conditional confidence interval for* f *is a random mapping,* $\text{CI}^{(\alpha)}(\mathscr{Y}, U, \cdot) : \mathbb{R}^d \to \mathcal{P}(\mathbb{R}) : \boldsymbol{x} \mapsto \text{CI}^{(\alpha)}(\mathscr{Y}, U, \boldsymbol{x})$*, such that*

$$\mathbb{E}_{\mathscr{Y},U}\left[\mathbb{1}_{\{f(\boldsymbol{x})\in \text{CI}^{(\alpha)}(\mathscr{Y},U,\boldsymbol{x})\}}\right] = 1-\alpha \quad \forall \boldsymbol{x}. \tag{2.1}$$

Here, $\mathcal{P}(\mathbb{R})$ is the power set of $\mathbb{R}$. With $(\mathscr{Y}, U)$, we explicitly denote that the construction of the interval depends on the specific realization of the targets and of a random effect. For each realization of these random variables, we get a different confidence interval. We drop this extra notation later on and simply write $\text{CI}^{(\alpha)}(\boldsymbol{x})$. Intuitively, this says that, given our set of covariates, if we randomly sample the targets and create a confidence interval, that the probability that $f(\boldsymbol{x})$ falls inside that interval is $1-\alpha$ for all values of $\boldsymbol{x}$. We refer to this type of coverage as *conditional coverage*. We also define *marginal coverage*. It is less desirable, but implicitly often used (see Section 2.3).

Definition 2. *A* $(1-\alpha)\cdot 100\%$ *marginal confidence interval for* f *is a random mapping,* $\text{CI}^{(\alpha)}(\mathscr{Y}, U, \cdot) : \mathbb{R}^d \to \mathcal{P}(\mathbb{R}) : \boldsymbol{x} \mapsto \text{CI}^{(\alpha)}(\mathscr{Y}, U, \boldsymbol{x})$*, such that*

$$\mathbb{E}_{\mathscr{Y},U}\left[\mathbb{E}_X\left[\mathbb{1}_{\{f(X)\in \text{CI}^{(\alpha)}(\mathscr{Y},U,X)\}}\right]\right] = 1-\alpha. \tag{2.2}$$

Marginal coverage states that the probability that, for a random realization of X and a random realization of the confidence interval (which is random because the specific training set and training process is random), the function value falls inside the CI is $(1-\alpha)\cdot 100\%$. The inner expectation gives the probability that a function value for a random realization $\boldsymbol{x}$ falls in *that* specific confidence interval. The outer expectation averages over all possible targets and random training effects, each resulting in slightly different intervals. We emphasize that in this definition the coverage may be very different across different values of $\boldsymbol{x}$. It is immediately clear from the definitions that conditional coverage is much stronger than marginal coverage[1].

Analogously, we define a conditional and marginal prediction interval as follows.

[1] An even stronger notion of coverage would be *simultaneous* coverage, where the function values must fall in the corresponding intervals for all x-values *at the same time* with probability $1-\alpha$. There is very little work on simultaneous intervals within the machine learning community so we do not elaborate on it further. We refer to Degras (2017) for an example in a related field.

Definition 3. *A* $(1-\alpha)\cdot 100\%$ *conditional prediction interval is a random mapping,* $\mathrm{PI}^{(\alpha)}(\mathscr{Y}, U, \cdot) : \mathbb{R}^d \to \mathcal{P}(\mathbb{R}) : \boldsymbol{x} \mapsto \mathrm{PI}^{(\alpha)}(\mathscr{Y}, U, \boldsymbol{x})$*, such that*

$$\mathbb{E}_{\mathscr{Y},U}\left[\mathbb{E}_{Y|X=\boldsymbol{x}}\left[\mathbb{1}_{\{Y\in \mathrm{PI}^{(\alpha)}(\mathscr{Y},U,\boldsymbol{x})\}}\right]\right] = 1-\alpha \quad \forall \boldsymbol{x}. \tag{2.3}$$

Definition 4. *A* $(1-\alpha)\cdot 100\%$ *marginal prediction interval is a random mapping,* $\mathrm{PI}^{(\alpha)}(\mathscr{Y}, U, \cdot) : \mathbb{R}^d \to \mathcal{P}(\mathbb{R}) : \boldsymbol{x} \mapsto \mathrm{PI}^{(\alpha)}(\mathscr{Y}, U, \boldsymbol{x})$*, such that*

$$\mathbb{E}_{\mathscr{Y},U}\left[\mathbb{E}_{X,Y}\left[\mathbb{1}_{\{Y\in \mathrm{PI}^{(\alpha)}(\mathscr{Y},U,X)\}}\right]\right] = 1-\alpha. \tag{2.4}$$

The difference between the two can be exemplified with a weather forecast. Suppose a weatherman gives a 90% prediction interval for the temperature tomorrow. If this is a conditional prediction interval, then the probability that the true temperature tomorrow falls inside that interval is 90%. If it is a marginal interval, however, then the weatherman says that averaged over all possible days, the temperature will fall in those intervals in 90% of the time, but there is no real guarantee for tomorrow. In the second case, the weatherman is allowed to be 85% correct in winter and 95% in the summer. The weatherman could even simply pick 3 days each month to give a point estimate - thus aways being wrong - and give the interval from -100 to 100 degrees Celsius for the rest of the month. Since these intervals are not very useful, we favor smaller intervals. We note that, for some applications, marginal coverage may be adequate and that it sometimes is explicitly the goal (e.g., with split-conformal inference).

In the following two subsections, we examine the most popular testing procedures for uncertainty estimates. We identify two different strategies, which we will explain in order:

1. The method outputs a prediction interval and the relevant metrics are the average width of the intervals and the fraction of test points that fall inside the prediction intervals (PICP).

2. The method outputs a density $p(y|\boldsymbol{x})$, generally $\mathcal{N}\left(\hat{f}(\boldsymbol{x}), \hat{\sigma}^2(\boldsymbol{x})\right)$, and the relevant metrics are the loglikelihood of a test set and the root-mean-squared error, RMSE.

Table 2.1: The ten different regression data sets that are currently being used as a benchmark for estimating the quality of uncertainty estimates. The number of data points, N, dimensionality, d, and a short description are given.

Data set	N	d	Description
Boston Housing	506	13	Housing prices in suburbs of Boston as a function of covariates such as crime rates and mean number of rooms
Concrete Compression Strength	1030	8	Concrete compressive strength as a function of covariates such as temperature and age
Energy Efficiency	768	8	The energy efficiencies of buildings as a function of covariates such as wall area, roof area, and height
Kin8nm	8192	8	The forward kinematics of an 8 link robot arm
Naval propulsion	11,934	16	A simulated data set giving the propulsion behaviour of a naval vessel
Combined Cycle Power Plant	9568	4	The net hourly electrical energy output as a function of temperature, ambient pressure, relative humidity, and exhaust vacuum
Protein Structure	45,730	9	Physicochemical properties of protein tertiary structure
Wine Quality Red	1599	11	Wine quality as a function of physicochemical tests such as density, pH, and sulphate levels
Yacht Hydrodynamics	308	6	Air resistance of sailing yachts as a function of covariates such as length-beam ratio, prismatic coefficient, or beam-draught ratio
Year Prediction MSD	515,345	90	The release year of a song based on audio features

2.2.2 The PICP Testing Procedure

We first explain a popular method to test prediction intervals directly. The idea is to take a real-world data set, split it in a training and test set, create prediction intervals using the training set, and calculate the fraction of test points that fall inside the prediction intervals. The relevant metric in this case is the PICP.

Definition 5. *The Prediction Interval Coverage Probability, or* PICP*, is the fraction of observations in a test set that fall inside the corresponding prediction intervals:*

$$\text{PICP} := \frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} \mathbb{1}_{\{y_i \in \text{PI}(\boldsymbol{x}_i)\}}.$$

Note that every method can be tested this way. Methods that output a density can create a PI using that density. This PI can be compared with the PI of a method that directly outputs one (such as quantile regression for instance). We can define the same measure for a confidence interval.

Definition 6. *The Confidence Interval Coverage Probability is the fraction of function values that fall inside the corresponding confidence intervals.*

$$\text{CICP} := \frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} \mathbb{1}_{\{f(x_i) \in \text{CI}(\boldsymbol{x}_i)\}}.$$

To be able to compute the CICP, one needs to have access to the true function $f(\boldsymbol{x})$. The average width of the prediction intervals is usually also reported since we prefer intervals that capture the correct fraction of the data while being as narrow as possible.

2.2.3 The Loglikelihood Testing Procedure

The loglikelihood testing approach assumes the uncertainty estimation method outputs a density $p(y \mid \boldsymbol{x})$, usually a normal distribution with mean $\hat{f}(\boldsymbol{x})$ and standard deviation $\hat{\sigma}_{\text{predictive}}$. The objective is to get the highest average loglikelihood on the test set:

$$\text{LL} = \frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} \log \left[\frac{1}{\sqrt{2\pi\sigma^2_{\text{predictive}}(\boldsymbol{x}_i)}} \exp\left(-\frac{1}{2} \left(\frac{y_i - \hat{f}(\boldsymbol{x}_i)}{\sigma_{\text{predictive}}(\boldsymbol{x}_i)} \right)^2 \right) \right].$$

Evaluating the likelihood tests how well the predicted density matches the true data generating density. This density could subsequently be used to make prediction intervals.

This approach has been popularised by Hernández-Lobato and Adams (2015). In their paper, they tested their method, probabilistic backpropagation, on ten publicly available real-world data sets (see Table 2.1). Using these data sets, they carried out the following procedure.

Step 1: Standardize the data so that it has zero mean and unit variance.

Step 2: Split the data into a training and test set. They apply a 90/10 split.

Step 3: Train the network using 40 epochs and update the weights of the network after each data point. They use a network with one hidden layer containing 50 hidden units. For the two largest data sets, *Protein Structure* and *Year Prediction MSD*, they chose 100 hidden units.

Step 4: Repeat steps 2 and 3 a total of 20 times and report the average root mean squared error ($RMSE$) and loglikelihood (LL) on the test set and their standard deviations. They undo the standardization before calculating the $RMSE$ and LL.

This setup has subsequently been used in multiple articles (Gal and Ghahramani, 2016; Lakshminarayanan et al., 2017; Mancini et al., 2020; Liu and Wang, 2016; Salimbeni and Deisenroth, 2017). Recently, these data sets have also been used to calculate the PICP metric (Khosravi et al., 2011; Pearce et al., 2020; Su et al., 2018; Lai et al., 2022). The following section discusses shortcomings of both the PICP and loglikelihood approach.

2.3 Theoretical Shortcomings of the Current Testing Methodology

In this section, we discuss four problems with the aforementioned methods of testing the quality of uncertainty estimates on a single test set with the PICP or the loglikelihood.

2.3.1 Predictive Performance Does Not Guarantee Good Model Uncertainty Estimates

For some applications, the predictive uncertainty is the only relevant quantity. For the prices on the stock market, it does not matter what the underlying function was; the actual observation is what counts. For a physicist trying to measure a constant or functional relation, however, the model uncertainty may be much more relevant. This model uncertainty is often used for out-of-distribution detection. The reasoning is that in an area that is previously unseen by the model, the model uncertainty is likely to be high. It is therefore crucial to know if the model uncertainty estimate is correct or not.

It is implicitly assumed that methods that have a better predictive performance on a test set also estimate the model uncertainty better. This need not be

the case. Suppose we have separate estimates for the data noise variance and model uncertainty. These two estimates can be combined to obtain a predictive uncertainty estimate. We compare these two methods, A and B, by carrying out the tests as described in the previous section, either the PICP or the loglikelihood. If method A gets a better score than method B, it is unclear why. It is possible that the estimate for the model uncertainty in method A was much worse than for method B, but that this was compensated by a superior estimate of the data noise variance. Yet another possibility is that both estimates are incorrect but result, more or less, in the correct total uncertainty. We provide empirical support in Section 2.5.

2.3.2 Coverage Cannot Be Tested on a Single Data Set

A confidence or prediction interval is a random variable because it depends on the specific realization of the targets. In the context of a neural network, there is also an added random training aspect. It is therefore necessary to test these intervals by repeating the entire process multiple times. This means that new targets need to be collected, a new network needs to be trained, and a new interval needs to be constructed. For a PI, for instance, the marginal coverage can be approximated by evaluating the PICP multiple times:

$$\frac{1}{L}\sum_{l=1}^{L}\left(\frac{1}{n_{\text{test}}}\sum_{i=1}^{n_{\text{test}}}\mathbb{1}_{\{y_i\in \text{PI}_l(\boldsymbol{x}_i)\}}\right) = \frac{1}{L}\sum_{l=1}^{L}\left(\text{PICP}_l\right),$$

where L is the total number of repeated experiments and the subscript l indicates that the prediction intervals will be different in each simulation. In fact, if we do this infinitely many times with an infinitely large test set, this approximation becomes exact since by the law of large numbers

$$\lim_{L\to\infty}\lim_{n_{\text{test}}\to\infty}\frac{1}{L}\sum_{l=1}^{L}\left(\frac{1}{n_{\text{test}}}\sum_{i=1}^{n_{\text{test}}}\mathbb{1}_{\{y_i\in \text{PI}_l(\boldsymbol{x}_i)\}}\right) = \mathbb{E}_{\mathscr{Y},U}\left[\mathbb{E}_{X,Y}\left[\mathbb{1}_{\{Y\in \text{PI}^{(\alpha)}(\mathscr{Y},U,X)\}}\right]\right].$$

By examining the previous equation, we observe that the PICP in fact only gives a single approximation of the inner expectation, $\mathbb{E}_{X,Y}\left[\mathbb{1}_{\{Y\in \text{PI}^{(\alpha)}(\mathscr{Y},U,X)\}}\right]$.

To exemplify this problem, we simulated an example where we fitted a linear model on 25 data points. The x-values were simulated uniformly between -2 and 2, and we used $Y \mid X = x \sim \mathcal{N}\left(x, 0.1^2\right)$, a straight line with some noise. With these 25 data points, we constructed an 80% prediction interval using classical theory and computed the PICP using a test set containing 500 data points.

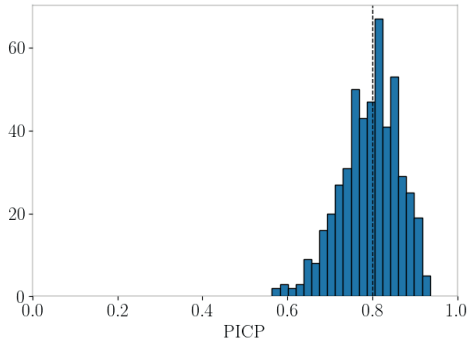


Figure 2.2: The PICP values of 500 simulations. In each simulation, new data was generated, a new linear model was fit, and a new prediction interval was created.

We repeated this process 500 times to demonstrate that a single evaluation of the PICP is not indicative of the quality of a prediction interval. The PICP values are shown in Figure 2.2. We have a perfect prediction interval, but the individual PICP values range between 0.58 and 0.92.

Analogously, we see that the CICP gives a single realization of the inner expectation in the definition of marginal coverage:

$$\lim_{L\to\infty}\lim_{n_{\text{test}}\to\infty}\frac{1}{L}\sum_{l=1}^{L}\left(\frac{1}{n_{\text{test}}}\sum_{i=1}^{n_{\text{test}}}\mathbb{1}_{\{f(\boldsymbol{x}_i)\in \text{CI}_l(\boldsymbol{x}_i)\}}\right) = \mathbb{E}_{\mathscr{Y},U}\left[\mathbb{E}_X\left[\mathbb{1}_{\{f(X)\in \text{CI}^{(\alpha)}(\mathscr{Y},U,X)\}}\right]\right].$$

Looking at a linear model illustrates this can be even more problematic. Consider a linear model without an intercept term,

$$y_i = ax_i + \epsilon_i.$$

The model uncertainty considers the uncertainty in our estimate $\hat{a}$. For this example, we assume that the true function is of the form $f(x) = ax$, meaning that the true function is within our hypothesis class. Suppose we have a perfectly calibrated procedure to construct a 95% CI for a. This means that an interval constructed by that procedure will contain the true value of a in 95% of the experiments (collecting data, fitting the model, creating the CI).

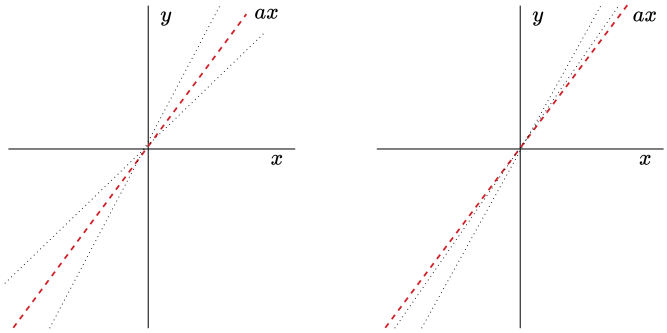


Figure 2.3: Illustration of the CICP for a linear model. The dashed red line gives the true function $f(x) = ax$. The two dotted black lines give the confidence interval. In the left figure, the true function falls inside our confidence interval; in the right figure it does not. It is clear that the measure CICP will either give 1 or 0, even when we have a method that gives a perfect 95% confidence interval: It is impossible to test the coverage of our CI on a single test set.

We can translate the CI of a to a CI of $f(x_i) = ax_i$. However, since the true a is either inside our CI or not, $f(x_i)$ is for *all* x_i either inside the CI or not. This results in a CICP of either 0 or 1, even though the uncertainty estimate is perfectly calibrated. This illustrates that we cannot test the quality of our CI by simply looking at the fraction of points in a single test set that are in our interval. This example is illustrated in Figure 2.3. It is simply not possible to test the quality of a prediction or confidence interval on a single data set.

2.3.3 A Good PICP Score Does Not Guarantee Conditional Coverage

The PICP score estimates marginal coverage and not conditional coverage. It is desirable to have prediction and confidence intervals that have the correct coverage for each specific value of $\boldsymbol{x}$ and not merely on average.

Marginal and conditional coverage can be related as follows:

$$\int_{\mathcal{X},\mathcal{Y}} \mathbb{1}_{y \in \mathrm{PI}(\boldsymbol{x})} \pi(\boldsymbol{x}, y) d\boldsymbol{x} dy = \int_{\mathcal{X}} \left(\int_{\mathcal{Y}} \mathbb{1}_{y \in \mathrm{PI}(\boldsymbol{x})} \pi(y \mid \boldsymbol{x}) dy \right) \pi(\boldsymbol{x}) d\boldsymbol{x}. \tag{2.5}$$

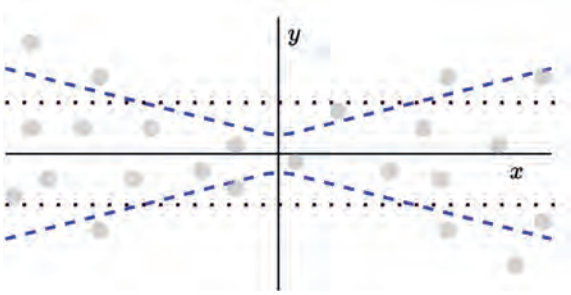


Figure 2.4: This figure illustrates that estimating the data noise variance correctly can result in a PICP close to the chosen confidence level. We assume that the model uncertainty is comparably very small. The true function, $f(x)$, is the constant zero function. The dashed blue line gives $\pm 1\sigma(x)$. The dotted black line gives $\pm 1\hat{\sigma}(x)$. On average, the data noise variance estimate is correct and its corresponding PI captures the correct fraction of the data in this case. Using PICP in this example, we do not notice that our uncertainty estimate is wrong.

This illustrates that a good marginal coverage does not imply a good conditional coverage. It is possible to get a good PICP score (which approximates the left integral in Equation (2.5)) while only estimating the predictive uncertainty correctly *on average*. We illustrate this in Figure 2.4. Assume that the true function is the constant zero function, $f(x) = 0$, and that our estimate of the function is very good, $\hat{f}(x) \approx 0$. The dashed blue lines give plus and minus one time the true standard deviation of the data. Suppose that we use a homoscedastic estimate of this standard deviation, the dotted black line, and that this estimate has on average the correct size. The PICP score of the 68% PI made with our homoscedastic estimate is 0.65 in this example. The intervals are too wide for some x and too small for others but *on average* capture the correct fraction of the data. We are not able to see that our estimate is wrong by simply evaluating the PICP on a single test set.

We also want to stress that, with a one-dimensional output, it is often possible to get close to the desired PICP value on an unseen part of the data set by tuning the hyperparameters. Monte Carlo dropout, for instance, has the data noise variance as a hyperparameter. If the PICP value is too low, it is possible to tune this parameter until it is correct. If the test set resembles the training set, it is unsurprising that a good PICP score can also be achieved on this set. We elaborate on this in Section 2.5.

2.3.4 Shortcomings of the Loglikelihood

The main downside of the loglikelihood is the inability to compare methods that output a prediction interval directly with methods that output a density. However, we argue that even when comparing methods that output a density, the loglikelihood has shortcomings and should be supplemented with other metrics, which are discussed in Section 2.4.

Before addressing the downside of the loglikelihood, we need to mention the argument in favor of using it. If the goal is to find the density that is closest to the true density $\pi(y \mid \boldsymbol{x})$, then the loglikelihood is optimal in the following sense. Suppose that the outputted density is parametrized by $\boldsymbol{\theta}$ and we find $\hat{\boldsymbol{\theta}}$ such that the loglikelihood is maximal on a test set:

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{n_{\text{test}}} \log(p_{\boldsymbol{\theta}}(y_i \mid \boldsymbol{x}_i)).$$

Akaike (1973) showed that (under some assumptions) $\hat{\boldsymbol{\theta}}$ is a natural estimator for the $\boldsymbol{\theta}$ that minimises the KL divergence between the true density and the outputted density. In this sense, the loglikelihood seems to be the obvious metric to measure the quality of an uncertainty estimate. However, if the eventual goal is to make a prediction interval or a confidence interval, which is often the case in applications, then the loglikelihood could favor the method that produces worse prediction intervals or confidence intervals. This is because, although loglikelihood depends on the quality of the predicted variance, it also highly depends on the quality of the fit.

A higher loglikelihood can therefore be the result of a better fit or of a better estimate of the predictive variance. For a single model, improving the loglikelihood will generally improve the quality of the prediction intervals. The problem arises when comparing drastically different models that give very different predictions for varying regions of the data.

This ambiguity can be problematic since the method with the higher loglikelihood can produce significantly worse prediction intervals. To show this, we go through a quick example where two methods are compared by using the loglikelihood on a test set. Suppose we have n data points, x_i, from a $\mathcal{N}\left(100, 5^2\right)$ distribution. We want to compare two blackbox methods that output a mean estimate, $\hat{\mu}$, and an uncertainty estimate, $\hat{\sigma}^2$. The blackbox methods arrive at

the following estimates:

$$\hat{\mu}_1 = \frac{1}{n}\sum_{i=1}^{n} x_i, \quad \hat{\sigma}_1 = 0.9\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{\mu}_1)^2},$$

and

$$\hat{\mu}_2 = 1.05 \cdot \frac{1}{n}\sum_{i=1}^{n} x_i, \quad \hat{\sigma}_2 = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{\mu}_2)^2}.$$

The mean estimate of model 1 is better than that of model 2, but its uncertainty estimate is too small. With these estimates, we create 68% prediction intervals for both models:

$$\text{PI}_i = \hat{\mu}_i \pm \hat{\sigma}_i.$$

We ran 10000 simulations to compute the coverage of the prediction intervals. This means simulating new data, getting new estimates, and creating a new PI. In this example, model 1 has a slightly better loglikelihood (-3.109 versus -3.110) but a considerably worse coverage (0.57 versus 0.67).

This demonstrates that it is important to clearly have in mind what the end goal of the uncertainty quantification method is. If the end goal is to find the density closest to the true density, then the loglikelihood is a useful metric. If, however, the eventual goal is to construct prediction or confidence intervals, solely using loglikelihood could be misleading.

A practical example can be found in the paper on Monte Carlo dropout (Gal and Ghahramani, 2016). The authors compare their method to an, at that time, popular variational inference method (Graves, 2011) and probabilistic backpropagation (Hernández-Lobato and Adams, 2015). Monte Carlo dropout achieves superior or comparable loglikelihood scores on the test sets of all ten data sets in Table 2.1. However, MC dropout also obtains the lowest RMSE on the test sets for nine of the ten data sets. It is not immediately clear if the better loglikelihood is the result of more precise predictions or of a better uncertainty estimate. Subsequently, the paper on Deep Ensembles (Lakshminarayanan et al., 2017) outperformed MC dropout in terms of both loglikelihood and RMSE, leaving the question open if the uncertainty estimate is actually better.

We therefore argue that if the eventual goal is to make prediction or confidence intervals, the loglikelihood should be supplemented by testing **conditional coverage** directly, both for the prediction and confidence intervals. A good conditional coverage ensures good marginal coverage as well. In the following

section, we explain an approach that allows us to test conditional coverage: simulation-based testing.

2.4 Simulation-Based Testing

In this section, we propose a new testing approach that addresses the issues that were raised in the previous section. More specifically, we give a testing procedure that tests conditional coverage in a correct manner. As we argued in the previous section, it is impossible to test coverage correctly on a single data set. It is necessary to repeat the entire experiment multiple times, with new data sets, and then measure the conditional coverage. We therefore propose a simulation-based setup, in which we are able to simulate new data sets and know the true data generating distribution. This has two advantages:

1. The experiment can be repeated multiple times. This allows us to test coverage in the correct sense.
2. The true function, $f(\boldsymbol{x})$, is known. This allows us to directly test the quality of a confidence interval, and not only a prediction interval.

The metrics we propose are the Prediction Interval Coverage Fraction (PICF), the Confidence Interval Coverage Fraction (CICF), and the average width of the intervals. The PICF is defined as follows.

$$\text{PICF}(\boldsymbol{x}) := \frac{1}{n_{\text{sim}}} \sum_{s=1}^{n_{\text{sim}}} \mathbb{E}_{Y|X=\boldsymbol{x}} \left[\mathbb{1}_{\{Y \in \text{PI}_s(\boldsymbol{x})\}} \right], \tag{2.6}$$

where Y is the random variable of which the realizations are the observations, n_{sim} the number of simulations, and $\text{PI}_s(\boldsymbol{x})$ the prediction interval for $\boldsymbol{x}$ in simulation s. Our proposed approach effectively gives a Monte Carlo approximation of the conditional coverage. For a large number of simulations, this precisely becomes the definition of conditional coverage by the law of large numbers:

$$\lim_{n_{\text{sim}} \to \infty} \frac{1}{n_{\text{sim}}} \sum_{s=1}^{n_{\text{sim}}} \mathbb{E}_{Y|X=\boldsymbol{x}} \left[\mathbb{1}_{\{Y \in \text{PI}_s(\boldsymbol{x})\}} \right] = \mathbb{E}_{\mathscr{Y},U} \left[\mathbb{E}_{Y|X=\boldsymbol{x}} \left[\mathbb{1}_{\{Y \in \text{PI}(\mathscr{Y},U,\boldsymbol{x})\}} \right] \right].$$

This setup forces us to manually define the data generating process. If we use

$$\mathcal{P}_{Y|X=\boldsymbol{x}} = \mathcal{N}\left(f(\boldsymbol{x}), \sigma^2(\boldsymbol{x})\right),$$

then the expectation in Equation (3.6) can be calculated as

$$\mathbb{E}_{Y|X=\boldsymbol{x}}\left[\mathbb{1}_{\{Y\in \mathrm{PI}(\boldsymbol{x})\}}\right] = \Phi\left(\frac{R^{(s)}(\boldsymbol{x}) - f(\boldsymbol{x})}{\sigma(\boldsymbol{x})}\right) - \Phi\left(\frac{L^{(s)}(\boldsymbol{x}) - f(\boldsymbol{x})}{\sigma(\boldsymbol{x})}\right), \quad (2.7)$$

where Φ is the CDF of a standard normal Gaussian and $L^{(s)}(\boldsymbol{x}), R^{(s)}(\boldsymbol{x})$ are the lower and upper bounds respectively of the PI in simulation s. The superscript s indicates that these intervals are different for each simulation. Analogously, we define the Confidence Interval Coverage Fraction:

$$\mathrm{CICF}(\boldsymbol{x}) := \frac{1}{n_{\mathrm{sim}}} \sum_{s=1}^{n_{\mathrm{sim}}} \mathbb{1}_{\{f(\boldsymbol{x})\in \mathrm{CI}_s(\boldsymbol{x})\}},$$

where $f(\boldsymbol{x})$ is the true function value. Note that if we have n_{test} observations in our test set, then we obtain n_{test} different computations of $\mathrm{PICF}(\boldsymbol{x})$ and $\mathrm{CICF}(\boldsymbol{x})$. These evaluations can be plotted in a histogram to see if the coverage fractions match the chosen confidence levels. In a one-dimensional setting, we can also plot the PICF and CICF as a function of $\boldsymbol{x}$.

If we construct a $100 \cdot (1-\alpha)\%$ PI or CI, then we would ideally want the $\mathrm{PICF}(\boldsymbol{x})$ and the $\mathrm{CICF}(\boldsymbol{x})$ to be $1-\alpha$ for every $\boldsymbol{x}$. As a quantitative measure for the quality of the PI and CI we propose to use the Brier score, where lower is better. For the PICF, this yields

$$\mathrm{BS} = \frac{1}{n_{\mathrm{test}}} \sum_{i=1}^{n_{\mathrm{test}}} \left(\mathrm{PICF}(\boldsymbol{x}_i) - (1-\alpha)\right)^2. \quad (2.8)$$

A Brier score for the PICF for instance is lower if the average is close to the desired value of $(1-\alpha)$ while having a low variance. We observe this by looking at the bias-variance decomposition

$$\mathbb{E}_X\left[\left(\mathrm{PICF}(X) - (1-\alpha)\right)^2\right] = \mathbb{E}_X\left[\mathrm{PICF}(X) - (1-\alpha)\right]^2 + \mathbb{V}_X\left[\mathrm{PICF}(X)\right].$$

This means that simply being correct on average instead of for all $\boldsymbol{x}$ results in a worse score when using the Brier score of the PICF. We can use the same measure to quantify the quality of a CI. The suggested simulation-based approach can be summarized as follows.

Step 1: Choose a distribution $\mathcal{P}_{X,Y}$ to simulate data sets from.

Step 2: Simulate a test set. This test set does not change and must be the same when comparing different methods.

Step 3: Simulate a training set and use the uncertainty estimation method to obtain the PIs and CIs at different confidence levels, for instance 95, 90, 80, and 70%. Repeat this 100 times[2].

Step 4: Calculate the PICF and CICF for each $\boldsymbol{x}$-value in the test set.

Step 5: Evaluate the relevant metrics, for instance the Brier score and the average width.

To simulate data, we propose three possibilities. The first option is to take a known test function and a noise term. The advantage of this is having total control over the setup. The disadvantage is that it may not be representative of real-world situations. A second option is to use simulations that are based on real-world data sets. The current benchmark data sets are good candidates. In the last part of the next section, we demonstrate a simulation setup for the popular Boston Housing data set. A third option is to use an extremely large data set. This data set can be split in 1 test set and 100 distinct training sets. The previous procedure can now be repeated but instead of simulating data we can use the real data. The disadvantage is that only prediction intervals can be tested in this way since the true underlying function is unknown.

2.5 Experimental Demonstration of the Advantages of Simulation-Based Testing

In this section, we experimentally demonstrate the theoretical shortcomings raised in Section 2.3 and show how these issues can be resolved by using a simulation-based approach. As an illustration, we use two methods that are easy to implement: the naïve bootstrap (Heskes, 1997), and concrete dropout (Gal et al., 2017), an improvement of the popular Monte Carlo dropout method (Gal and Ghahramani, 2016).

With these two approaches, we first show that a good PICP does not imply that the individual estimates for the data noise variance and model uncertainty are correct. Since we know the true underlying function, we can look at the CICP directly and verify its correctness. Secondly, we demonstrate the advantage

[2]Of course, more is better, but we found that this works well enough for a comparison.

of our proposed procedure to average over simulations per $\boldsymbol{x}$-value, instead of averaging over $\boldsymbol{x}$-values in a single test set. We demonstrate that this is useful by observing that having the desired CICP gives no guarantees that the CIs are correct for an individual $\boldsymbol{x}$-value. Most importantly, we demonstrate that it is possible to favor the wrong method when using the predictive performance (such as the PICP or loglikelihood) on a test set as the measure. We end this section by setting up a simulation based on the Boston Housing data set.

2.5.1 Concrete Dropout and the Naive Bootstrap

The uncertainty estimation methods used in this section assume that both the data noise variance and model uncertainty are normally distributed:

$$y_i = f(\boldsymbol{x}_i) + \epsilon_i, \quad \text{with } \epsilon_i \sim \mathcal{N}\left(0, \sigma^2(\boldsymbol{x}_i)\right)$$

and

$$\hat{f}(\boldsymbol{x}) = f(\boldsymbol{x}) + \epsilon_{\omega,i}, \quad \text{with } \epsilon_{\omega,i} \sim \mathcal{N}\left(0, \sigma_\omega^2(\boldsymbol{x}_i)\right).$$

These two uncertainties can be combined to jointly make up the total predictive uncertainty. The methods additionally assume that both uncertainty estimates are independent. This independence allows us to add up both the variances to obtain the variance of the predictive uncertainty.

We use the bootstrap setup from Heskes (1997). This setup outputs a CI as described in Algorithm 1.

In this algorithm, $t^M_{1-\alpha/2}$ is the $1-\frac{\alpha}{2}$ quantile of a t-distribution with M degrees of freedom. We note that the variance in line 5 technically gives the variance of an individual ensemble member and not of the average. As we will see in Section 5, this results in confidence intervals that are often too large.

We train networks with 40, 30, and 20 neurons respectively and *ReLU* activation functions for 80 epochs. In order to arrive at a prediction interval we also need an estimate of the data noise variance. Assuming homoscedastic noise, we take

$$\hat{\sigma}^2 = \frac{1}{n_{\text{val}}} \sum_{j=1}^{n_{\text{val}}} \max\left(\left(y_j - \hat{f}(\boldsymbol{x}_j)\right)^2 - \hat{\sigma}_\omega^2(\boldsymbol{x}), 0\right). \tag{2.9}$$

Algorithm 1 Pseudocode to obtain a CI for $f(\boldsymbol{x})$ using an implementation of the naïve bootstrap approach as described by Heskes (1997).

for i in 1:M **do**
  Resample $\mathcal{D} = ((\boldsymbol{x}_1, y_1), \dots (\boldsymbol{x}_n, y_n))$ pairwise with replacement. Denote this sample with $\mathcal{D}^{(i)}$;
  Train an ANN on $\mathcal{D}^{(i)}$ that outputs $\hat{f}_i(\boldsymbol{x})$;
end for
Define $\hat{f}(\boldsymbol{x}) = \frac{1}{M} \sum_{i=1}^{M} \hat{f}_i(\boldsymbol{x})$;
Calculate $\hat{\sigma}^2_\omega(\boldsymbol{x}) = \frac{1}{M-1} \sum_{i=1}^{M} \left(\hat{f}_i(\boldsymbol{x}) - \hat{f}(\boldsymbol{x}) \right)^2$;
$\mathrm{CI}(\boldsymbol{x}) = [\hat{f}(\boldsymbol{x}) - t^M_{1-\alpha/2} \hat{\sigma}_\omega(\boldsymbol{x}),\ \hat{f}(\boldsymbol{x}) + t^M_{1-\alpha/2} \hat{\sigma}_\omega(\boldsymbol{x})]$;
return $\mathrm{CI}(\boldsymbol{x})$;

Note that we use a small additional validation set to determine $\hat{\sigma}$. With both $\hat{\sigma}$ and $\hat{\sigma}_\omega(\boldsymbol{x})$, we construct the prediction interval

$$\mathrm{PI}(\boldsymbol{x}) = \left[\hat{f}(\boldsymbol{x}) - t^M_{1-\alpha/2} \sqrt{\hat{\sigma}^2_\omega(\boldsymbol{x}) + \hat{\sigma}^2},\ \hat{f}(\boldsymbol{x}) + t^M_{1-\alpha/2} \sqrt{\hat{\sigma}^2_\omega(\boldsymbol{x}) + \hat{\sigma}^2} \right].$$

A different approach to obtain uncertainty estimates is Monte Carlo dropout (Gal and Ghahramani, 2016). The easy implementation makes this method very popular. The idea is to train a network with dropout enabled and then keep dropout enabled while making predictions. The article shows that, under certain conditions, this is equivalent to sampling from an approximate posterior. The standard deviation of these forward passes is used as an estimate of the model uncertainty. The inverse of the hyperparameter τ gives the estimate of the variance of the noise. A follow-up paper (Gal et al., 2017) further refines the method. This so called *concrete* dropout does not rely on a hyperparameter to estimate the data noise variance but outputs a heteroscedastic estimate directly. Additionally, the dropout probability is tuned as a part of the training process. Algorithm 2 describes the procedure in more detail.

2.5.2 A Toy Example with Homoscedastic Noise

In this subsection, we demonstrate that a good on average performance on a single test set gives no guarantees for the actual quality of the uncertainty estimate. We simulated data from $y = f(x) + \epsilon$ with $f(x) = (2x - 1)^3$, and $\epsilon \sim \mathcal{N}\left(0, (0.2)^2\right)$. The training and test set both contain 1000 data points. The

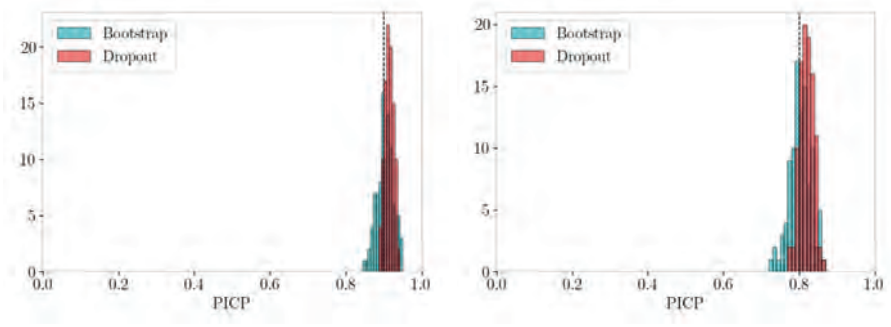
Algorithm 2 Pseudocode to obtain a confidence and prediction interval using concrete dropout.

1: Train an ANN with dropout enabled on $\mathcal{D}_{\text{train}}$. This network has two output neurons corresponding to $\hat{f}_b(\boldsymbol{x}_i)$ and $\hat{\sigma}_b^2(\boldsymbol{x}_i)$ and is trained by maximizing the loglikelihood assuming a normal distribution. The b subscript indicates that each forward pass gives a different result.;
2: Define $\hat{f}(\boldsymbol{x}) := \frac{1}{B}\sum_{b=1}^{B} \hat{f}_b(\boldsymbol{x})$;
3: Define $\hat{\sigma}^2(\boldsymbol{x}) := \frac{1}{B}\sum_{b=1}^{B} \hat{\sigma}_b^2(\boldsymbol{x})$;
4: Define $\hat{\sigma}_\omega^2(\boldsymbol{x}) := \frac{1}{B-1}\sum_{b=1}^{B} \left(\hat{f}(\boldsymbol{x}) - \hat{f}_b(\boldsymbol{x})\right)^2$;
5: $\text{CI}(\boldsymbol{x}) = [\hat{f}(\boldsymbol{x}) - z_{1-\alpha/2}\hat{\sigma}_\omega(\boldsymbol{x}),\ \hat{f}(\boldsymbol{x}) + z_{1-\alpha/2}\hat{\sigma}_\omega(\boldsymbol{x})]$;
6: $\text{PI}(\boldsymbol{x}) = [\hat{f}(\boldsymbol{x}) - z_{1-\alpha/2}\sqrt{\hat{\sigma}_\omega^2(\boldsymbol{x}) + \hat{\sigma}^2(\boldsymbol{x})},\ \hat{f}(\boldsymbol{x}) + z_{1-\alpha/2}\sqrt{\hat{\sigma}_\omega^2(\boldsymbol{x}) + \hat{\sigma}^2(\boldsymbol{x})}]$;
7: **return** $\text{CI}(\boldsymbol{x})$, $\text{PI}(\boldsymbol{x})$;

bootstrap method has an additional 150 validation data points to determine the data noise variance. The x-values are drawn uniformly from the interval $[-0.5, 0.5]$. A total of $M = 50$ bootstrap networks are trained in order to construct a CI and PI for each x-value in the test set using the bootstrap approach. For the dropout approach, we used $B = 100$ forward passes through the network. We repeated these procedures for estimating prediction and confidence intervals for 100 simulations of randomly drawn training sets.

In Figure 2.5, we see that both methods give a PICP that is very close to the desired values of 0.9 and 0.8 in each of the 100 simulations. Note that when simply using one data set, we would only have 1 PICP value. Furthermore, as we argued in Section 2.3.1, it is not clear that a good PICP indicates that the CIs are good.

In a real-world scenario, we would not have access to the true function, $f(x)$, and we would not be able to calculate the CICP. In this case, however, we are. In Figure 2.6, we can see that the CICP values were not that great for most simulations. We see that there is a lot of variance between the CICP values of individual simulations. This exemplifies why using a single test set is not sufficient, as we argued in Section 2.3.2. Additionally, we observe that the confidence intervals of the bootstrap method are often too large, which we expected since the estimated model uncertainty uses the variance of an individual ensemble member and not of the average of the ensemble members. However, even *if* these CICP values would have been perfect, it is still possible that this happens because the CIs are only correct on average. CIs that are



(a) $\alpha = 0.1$. (b) $\alpha = 0.2$.

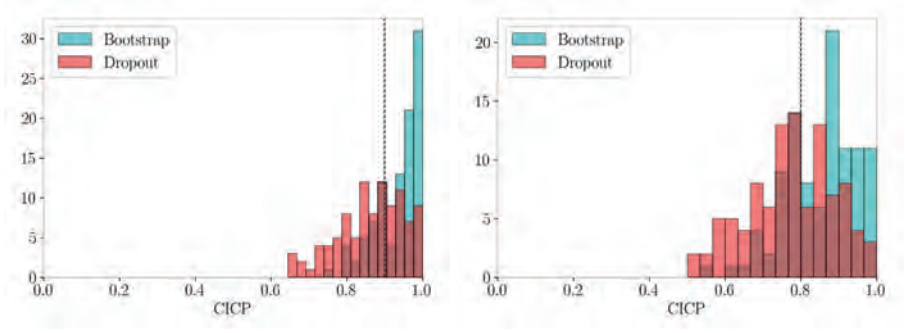
Figure 2.5: These histograms give 100 evaluations of the PICP, at a $(1 - \alpha)$ confidence level using both the bootstrap and dropout approach. The PICP captures the fraction of data points in the test set for which the observations y falls inside the corresponding prediction interval. The data is simulated from $y = f(x) + \epsilon$ with $f(x) = (2x - 1)^3$, and $\epsilon \sim \mathcal{N}\left(0, (0.2)^2\right)$. The details of the construction of the PIs can be found in Section 2.5.1. From these histograms we can see that in a single simulation we would have a good performance of the PI on the test set with either method.

much too large for some x can be countered by CIs that are much too small for other x.

This effect can be seen when we actually look at the coverage fraction per value of x calculated over all the simulations, the CICF, as we proposed in the previous section. To reiterate, for the PICP and CICP we average over test data points and then provide a histogram over simulations, whereas for the PICF and CICF values we average over simulations. In Figure 2.7(b) we can see that for some values of x the CIs contained the true value $f(x)$ in every simulation while hardly ever for other values of x. In Figure 2.7(a) we see that in this set of simulations the PIs are relatively accurate for most values of x and not only on average.

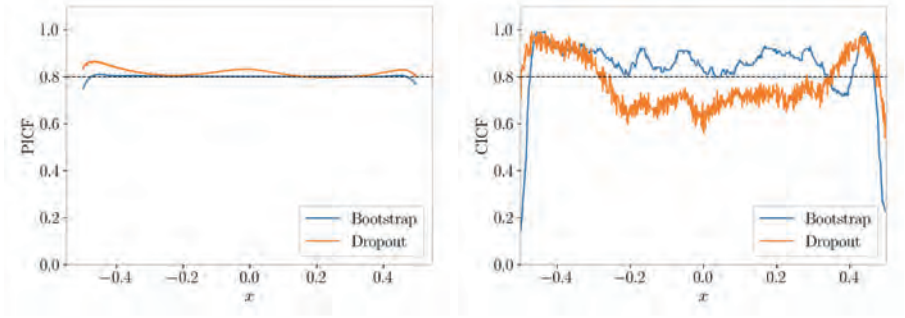
2.5.3 A Toy Example with Heteroscedastic Noise

In the previous example, the PIs behaved well. It is, however, also possible for the PICP to be good only because the PIs are correct on average. Figure 2.8



(a) $\alpha = 0.1$ (b) $\alpha = 0.2$

Figure 2.6: These histograms give 100 evaluations of CICP, at a $(1 - \alpha)$ confidence level. Each point in the histograms gives the fraction of datapoints in the test set of a new simulation for which the true function value $f(x)$ falls inside the corresponding confidence interval. Each simulation has its own CICP value. The same setup was used as in Figure 2.5. We observe that the good PICP values from Figure 2.5 do not translate to good CICP values.

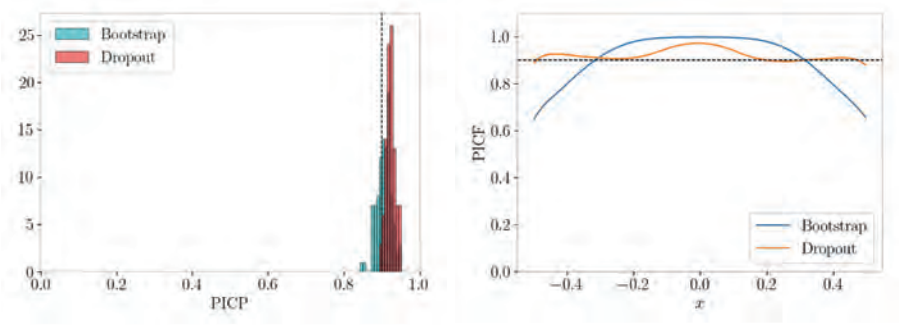


(a) PICF, $\alpha = 0.2$. (b) CICF, $\alpha = 0.2$.

Figure 2.7: The PICF and CICF plotted as a function of x. The PIs appear to be correct for most x while the CIs are often too large or too small. The PIs and CIs were constructed at an 80% confidence level.

illustrates this point. We repeat the simulation but now using a noise term with a standard deviation of $0.1+x^2$. The bootstrap method assumes homoscedastic

noise, while concrete dropout does not. Figure 2.8 illustrates that the PICP score does not show us that the data noise variance estimate of the bootstrap method is wrong. Even worse, it can point in the wrong direction. We can favor the worse method if we would use the coverage fraction on a test set as our metric. Note that, in Figure 2.8(a), we can see that, in most simulations, the bootstrap approach had a comparable or better PICP score compared to the dropout method. On average, the PICP score of the bootstrap method was even slightly closer to the desired value of 0.9. Figure 2.8(b), however, shows that this is only the case because for some values of x the coverage fraction was too high and for others too low, resulting in a good performance on average. According to the Brier score, dropout performed much better in this specific case.



(a) PICP, $\alpha = 0.1$.

(b) PICF, $\alpha = 0.1$.

Figure 2.8: These histograms give the different PICP and PICF values using the bootstrap and dropout approach. The PICP is obtained by calculating the coverage fraction of the PIs on the test set in each simulations. The PICF is obtained by calculating the coverage fraction of the PIs taken over all the simulations. In this simulation we constructed 90% PIs. Bootstrap has a Brier score of 0.011, dropout has a Brier score of 0.0011. We see that a good PICP score does not imply that the PIs are sensible for individual values of x.

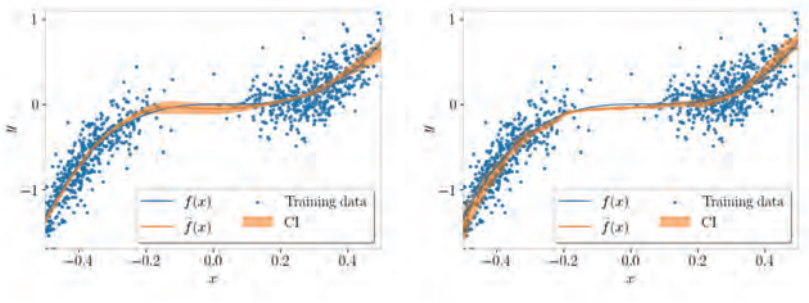
2.5.4 Out-of-Distribution Detection

A desirable property of a confidence interval is that it gets larger in areas where there is a limited amount of data. It is not evident that a good performance on a test set guarantees this effect. In Figure 2.9, we use the same function

as in our example with homoscedastic noise, but simulate our x values from a bimodal distribution instead of uniformly. Both models are trained using 1000 data points and are evaluated on a test set of size 1000. When making 80% CIs and PIs, the bootstrap and dropout methods give a PICP score of 0.75 and 0.83 respectively. The PICP score of the dropout method is closer to desired value of 0.8 and one might conclude from this that this method is better able to construct CIs. Additionally, the average loglikelihood was higher for the dropout method (0.14 versus 0.12). If we actually look at the CIs, however, we see that the behaviour is not as desired and that the bootstrap approach created more sensible CIs. The CIs of the bootstrap method get larger in the area around 0 where there is a limited amount of data and smaller around -0.4 and 0.3 where there is more data. The intervals created by using Monte Carlo dropout do the exact opposite, even though the performance on a single test set was better when using the PICP or loglikelihood as a metric.

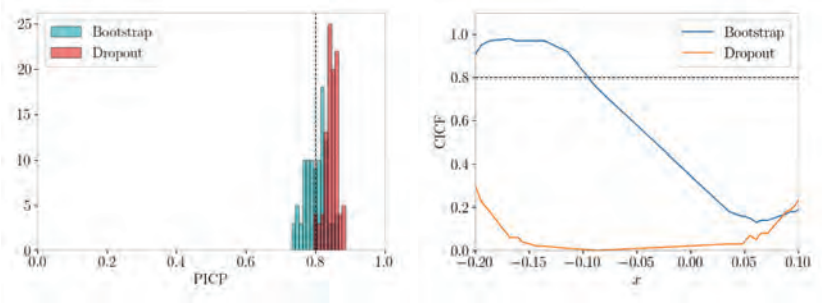
We simulated the data a total of 100 times to further demonstrate this behaviour. We can see in Figure 2.10(a) that in all 100 simulations both methods got a reasonable PICP score. If we look, however, at the CICF for x values between -0.2 and 0.1 we notice that dropout was not able to determine the uncertainty accurately[3]. Additionally, both methods had a substantial bias in the area with fewer data points. This information is only available if we use simulated data where the true function is known. Since both methods assume that the model is unbiased, it is unsurprising that the coverage is not perfect. As a side-note, some other methods, such as Zhou et al. (2018) do explicitly take this into account.

[3]We suspect that the behaviour of dropout is a result of the interpolation. When extrapolating, the ReLu activation functions cause the function values to increase. This gives rise to a large variance in the forward passes through the network. In the region around $x = 0$, the function values are almost zero, likely resulting in less variance and thus a smaller confidence interval. This explanation ignores subtleties with bias terms and it may be interesting to investigate this type of behaviour of dropout further.



(a) Bootstrap (b) Dropout

Figure 2.9: These two figures give 80% CIs using the naïve bootstrap (a) and Monte Carlo dropout (b). The blue line is the true function and blue dots give the training data. The same function and noise were used as when making Figure 2.5 with the difference that the covariates are not uniformly sampled. Even though the dropout approach gave a slightly better PICP score (0.83 versus 0.75) and higher average loglikelihood (0.14 versus 0.12), the CIs do not demonstrate better behaviour than those made with the bootstrap.



(a) PICP (b) CICF

Figure 2.10: These figures demonstrate that a good PICP value (a) does not guarantee desirable behaviour of the model uncertainty estimates. The region between -0.2 and 0.1 contained fewer data points. This figure, combined with Figure 2.9, demonstrates that both methods behave very differently in areas of limited data, something that is not detectable by merely evaluating the predictive uncertainty on a test set.

2.5.5 Boston Housing

So far, we only used a simple one-dimensional simulation in our testing procedure. A good testing procedure should: 1 - be representative for real-world problems, and 2 - allow easy comparison between different methods. The data sets that are currently being used, listed in Table 2.1, meet these criteria. As we just demonstrated, however, it is desirable to simulate the data to get more insights in the accuracy of the uncertainty estimates. A solution would be to set up simulations based on data sets that are currently being used. In Algorithm 3 we give a suggestion how we could set up a simulation that resembles the Boston Housing data set.

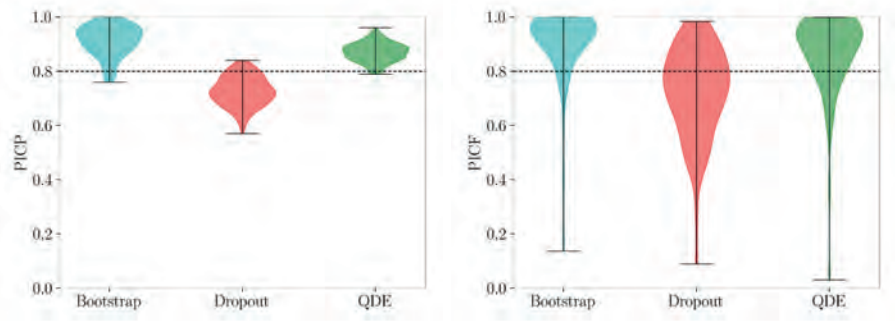
As an illustration, we implemented this idea using two random forests with 100 trees, and max depth 15. These hyperparameters resulted from a manual grid search. We do not attempt to simulate new $\boldsymbol{x}$-values as it would be difficult to get the dependencies between the covariates correct. We divided the generated data set in a train, test, and validation set of sizes 366, 100, and 40. The validation set is used by the bootstrap to determine the estimate of the data noise variance. The same bootstrap and dropout procedures as in the previous subsections were used to obtain PIs and CIs. To illustrate another advantage of our method, namely that we can directly compare methods that output a density with methods that directly output a prediction interval, we also implemented the Quality Driven Ensembles method (QDE) (Pearce et al., 2018). This approach directly outputs a prediction interval by optimizing a loss function that aims for the correct PICP on the training set, while being as narrow as possible.

Algorithm 3 Pseudo-code to simulate data based on the Boston Housing data set

Require: A real world data set $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \dots (\boldsymbol{x}_n, y_n)\}$;

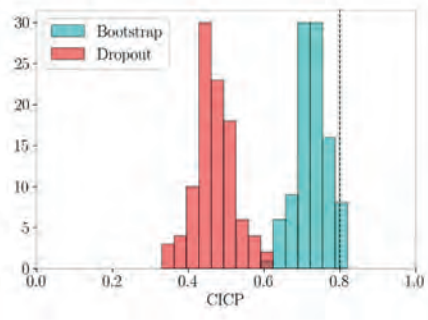
1: Train a random forest on $\mathcal{D}$ and use this predictor as the true function $f(\boldsymbol{x})$;
2: Calculate the residuals, $(y_i - f(\boldsymbol{x}_i))$;
3: Train a second random forest that predicts the residuals squared as a function of $\boldsymbol{x}$ and use this predictor as the true variance $\sigma^2(\boldsymbol{x})$;
4: Simulate a new data set $\mathcal{D}_{\text{new}} = \{(\tilde{\boldsymbol{x}}_1, \tilde{y}_1), \dots (\tilde{\boldsymbol{x}}_n, \tilde{y}_n)\}$, where $\tilde{\boldsymbol{x}}_i = \boldsymbol{x}_i$, and $\tilde{y}_i \sim \mathcal{N}\left(f(\tilde{\boldsymbol{x}}_i), \sigma^2(\tilde{\boldsymbol{x}}_i)\right)$;
5: **return** $\mathcal{D}_{\text{new}}$;

In Figure 2.11(a), we can see that the PICP was a little too high on average for the bootstrap method and QDE, a little too low for the dropout method, but overall quite close to the desired value of 0.8 in most of the simulations. In Figure 2.11(b), we can see, however, that for almost every $\boldsymbol{x}$, the prediction intervals were either too large or too small. We see a similar trend if we look at the CICP in Figure 2.11(c). Both the bootstrap and dropout method (QDE does not provide a confidence interval) consistently had a CICP that was too low. This enforces the argument that simply looking at the performance on a single test set is far from optimal. It also shows that it is possible to apply this simulation-based testing procedure to representative data sets.

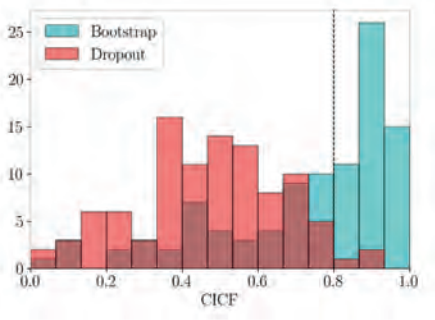


(a) PICP

(b) PICF



(c) CICP



(d) CICF

Figure 2.11: These violin plots and histograms give the different PICP, PICF, CICP, and CICF values using the bootstrap, dropout, and QDE approach on the Boston Housing simulation. For the QDE approach, only the PICP and PICF values are available as this method does not provide CIs. The PICP and CICP values are obtained by calculating the coverage fraction of the PIs and CIs on the test set in each simulation. The PICF and CICF values are obtained by calculating the coverage fraction of the PIs and CIs for each test data point taken over all the simulations. In this simulation we constructed 80% PIs and CIs. For the PICF, bootstrap has a Brier score of 0.027 and average width of 12.3. Dropout has a Brier score of 0.032 and an average width of 7.2. QDE has a Brier score of 0.024 and an average width of 9.6. For the CICF, bootstrap has a Brier score of 0.147 and average width of 4.92. Dropout has a Brier score of 0.15 and average width of 3.04. We once more observe that a (relatively) good PICP value gives no guarantees for the actual performance of either the PI or CI on individual data points.

2.5.6 Time series

While not the main focus of this chapter, our methodology of using a random forest to construct multiple simulated data sets can also be applied to time series. Algorithm 4 describes one of the possible ways in which this can be done.

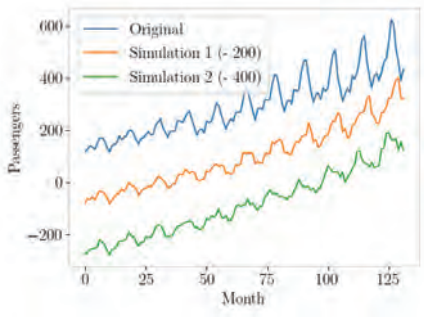
Algorithm 4 Pseudo-code to simulate a new time series

Require: A real-world time series $\{y_1, \dots y_n\}$, look-back time l, and variance σ^2;

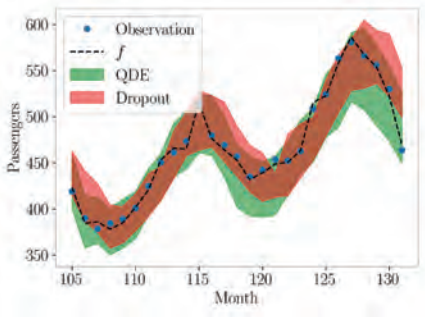
1: $\mathcal{D} = \{(\boldsymbol{x}_{l+1}, y_{l+1}), \dots (\boldsymbol{x}_n, y_n)\}$, where $\boldsymbol{x}_i = (y_{i-1}, \dots y_{i-l})$;
2: Train a random forest on $\mathcal{D}$ and use this predictor as the true function $f(\boldsymbol{x})$;
3: **for** i in 1:l **do**
4: $\quad \tilde{y}_i = y_i$;
5: **end for**
6: **for** i in $l+1$:n **do**
7: $\quad \tilde{\boldsymbol{x}}_i = (\tilde{y}_{i-1}, \dots, \tilde{y}_{i-l})$
8: $\quad \tilde{y}_i \sim \mathcal{N}(f(\tilde{\boldsymbol{x}}_i), \sigma^2)$
9: **end for**
10: $\mathcal{D}_{\text{new}} = \{(\tilde{\boldsymbol{x}}_{l+1}, \tilde{y}_{l+1}), \dots (\tilde{\boldsymbol{x}}_n, \tilde{y}_n)\}$;
11: **return** $\mathcal{D}_{\text{new}}$;

We start from a time series $(y_1, \dots y_n)$. Specifically, we will use the well-known air-passenger data set which has airline passenger data per month. We will consider the task of predicting the number of passengers in the following month based on the previous l months. We first create a baseline data set $\mathcal{D} = \{(\boldsymbol{x}_{l+1}, y_{l+1}), \dots (\boldsymbol{x}_n, y_n)\}$, where $\boldsymbol{x}_i = \{(y_{i-1}, \dots y_{i-l})\}$. With this data set, we train a random forest that predicts y_i based on $\boldsymbol{x}_i$ (line 3). Using this trained random forest, we simulate new data sets as follows. We fix the first l observations, y_i (lines 4 and 5). In other words, $\tilde{y}_i = y_i$ for all $i \leq l$. For $i > l$, we then iteratively define $\tilde{\boldsymbol{x}}_i = (\tilde{y}_{i-1}, \dots, \tilde{y}_{i-l})$ and sample a new observation, $\tilde{y}_i \sim \mathcal{N}(f(\tilde{\boldsymbol{x}}_i), \sigma^2)$ (lines 6 through 8). We used a constant variance of 25. This results in an entirely new time series each time, always starting from the same initial point.

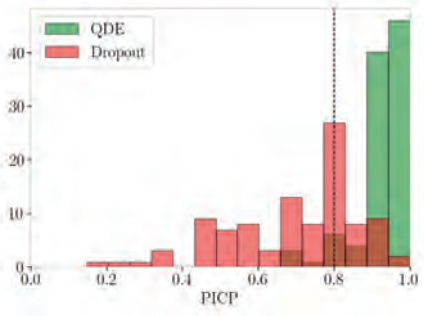
With this simulation, we can apply the simulation-based approach to a time series, as is illustrated in Figure 2.12. We create 100 new time series, see (a) for two examples. On each time series, we train two methods by using the first 80% of the series as the training set. For this example, we used the dropout method



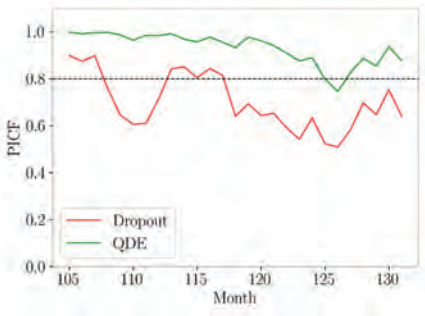
(a) Example of multiple simulated time series



(b) PIs of both methods during one of the simulations



(c) PICP values of the 100 simulations



(d) PICF per month of both approaches

Figure 2.12: This figure illustrates a simulation-based approach for a time series. A total of 100 different time series are simulated, two of which are displayed in (a). For each time series, both methods are trained on the first 80% of the series and evaluated on the final 20%. The resulting PIs on the test set for one of these simulations are displayed in (b). With the 100 simulations, we can compute the PICP and PICF values. We see that the PICP varies massively from simulation to simulation, especially for the dropout method, again highlighting that a single PICP value can give a poor estimate of marginal coverage. For this example, the simulation-based approach also allowed us to observe another interesting effect. For both approaches, the conditional coverage deteriorates further into to future.

and QDE, again illustrating that a method that directly outputs a PI can be compared to a method that outputs a density. Figure 2.12(b) displays the PIs of both methods on the test set, the final 20%, during a single simulation. By repeating this 100 times, we get 100 evaluations of the PICP (Figure 2.12(c)) as well as conditional estimates of the PICF (Figure 2.12(d)).

The results again illustrate the shortcomings of the PICP. Especially for the dropout method, the PICP values vary massively. This effect is particularly strong due to the small size of this data set. An individual PICP value would therefore not give a good estimate of marginal coverage. Additionally, we again see that marginal coverage gives no guarantees for conditional coverage. The QDE approach has very high coverage directly after the training horizon and then goes down further into the future. A similar trend is visible for the dropout method. Such an effect is only visible when evaluating conditional coverage and would be missed when solely evaluating the PICP.

Different approaches are also possible, such as retraining the model every fixed number of months or taking a heteroscedastic estimate of the variance of the noise. We acknowledge that the evaluation of time series is a well-studied field that typically uses slightly different evaluating approaches than the ones discussed in this chapter, see Cerqueira et al. (2020) for a more in-depth discussion on these approaches. Nevertheless, we demonstrated that our simulation-based approach is also applicable to time series and can provide interesting insights into the conditional coverage.

2.6 Conclusion

We conclude that the testing methodology applied to many recent publications for evaluating the quality of uncertainty estimates leaves a lot of room for improvement, especially if the eventual application of a method is the construction of a confidence or prediction interval.

Both the loglikelihood and the PICP evaluate the predictive uncertainty, which is a combination of the data noise variance and model uncertainty, on a previously unseen test set. A good predictive uncertainty overall, however, is not necessarily indicative of a good estimate of the model uncertainty or data noise variance. Since the true function values are unknown, it is impossible to test confidence intervals directly.

Furthermore, we showed that the PICP score has additional problems. A single

PICP score gives a very unreliable estimate of marginal coverage, especially for smaller data sets. To get a good estimate of marginal coverage, it is necessary to repeat the entire experiment. However, marginal coverage is typically not the desired property. A stronger and more useful characteristic is correct conditional coverage. We therefore propose to explicitly evaluate the conditional coverage of both the prediction and confidence intervals.

For the loglikelihood, the main problem is the inability to compare methods that output a PI directly with methods that output a density. Additionally, we demonstrated that a better score may not guarantee better prediction intervals, especially when comparing different models. We therefore propose to supplement the loglikelihood by also evaluating the conditional coverage.

To evaluate conditional coverage, we propose simulation-based testing. We note that we assume that the eventual application of the method is to give accompanying prediction or confidence intervals. In order to properly test coverage, it is necessary to repeat the experiment: create a new data set, train the model, create new intervals. Possible quantitative metrics of the PIs and CIs are the Brier score of the PICF/CICF and the average width of the intervals.

This approach has some downsides. The computational demands for running these tests are higher and there is a need to simulate the data. We propose to set up simulations based on the data sets listed in Table 2.1. It is also possible to set up a simulation based on a data set of particular interest. The additional computational demands only play a role during the testing of these uncertainty quantification methods and not in their usage in practice.

2.6.1 Future Research

In order to compare different uncertainty estimation methods, it is necessary to use the same simulations. It would therefore be useful to create a number of benchmark simulations that can be used to test uncertainty estimates. We propose to base these simulations on the data sets in Table 2.1. In Section 2.5.5 we gave a simple demonstration of such a simulation. In this chapter, we only considered normally distributed noise. Different distributions would allow us to explicitly see what happens if the customary assumption of normality does not hold. The methods in this chapter are based on this assumption but a method like quantile regression, for instance, is not.

We saw in Section 2.5 that a method that performs better on a test set does not

necessarily have better-behaving uncertainty estimates. With new benchmarks, it would be worthwhile to re-evaluate currently available methods for estimating uncertainty.

Chapter 3

Bootstrapped Deep Ensembles

This chapter is based on the preprint entitled *"Confident Neural Network Regression with Bootstrapped Deep Ensembles"* (Sluijterman et al., 2022), which is currently under submission. As is visualized in Figure 3.1, this chapter focuses on the parameter uncertainty. More specifically, we demonstrate that ensembling techniques typically ignore the classical source of parameter uncertainty, and we demonstrate that incorporating this results in improved uncertainty estimates.

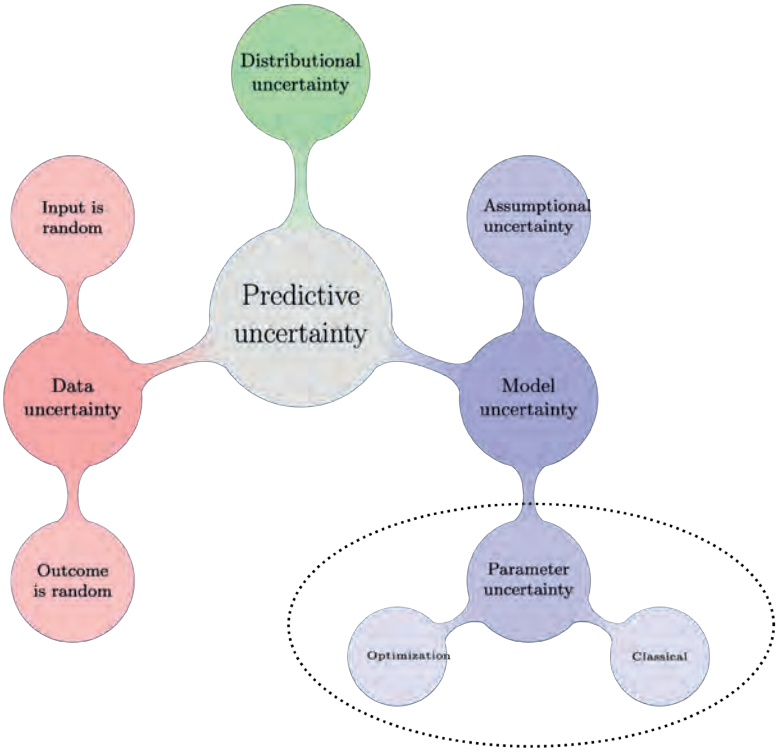


Figure 3.1: The scope of Chapter 3. This chapter focuses on the parameter uncertainty in an ensembling setting, while assuming that the distributional uncertainty and assumptional uncertainty are negligible. The parameter uncertainty is decomposed in an optimization part, a result of the random optimization procedure of modern neural networks; and a classical part, a result of estimating parameters on a finite set of random data.

3.1 Introduction

There has been an enormous interest in uncertainty quantification for machine learning in the past years. Numerous methods, discussed in the next section, have been developed. Of these methods, Deep Ensembles (DE) (Lakshminarayanan et al., 2017), which we explain in detail in the following section, is one of the most popular.

Ensembling methods (see, e.g., Heskes 1997 for an early example) such as DE accomplish two goals at once: The ensemble average reduces some of the variance and then provides a more accurate prediction than a random member, and the variance between the ensemble members can be used to estimate the uncertainty of the prediction.

This uncertainty estimate can only be calibrated if the construction of the ensemble members incorporates all relevant random factors. Firstly, we have the classical source of uncertainty: The model is trained on a finite data set that can be considered a random sample drawn from an unknown distribution. We use the term *classical* since for a model with a deterministic fit (such as, e.g., a linear model), the randomness of the data is the only source of variance in the parameter estimates. Secondly, the optimization procedure of neural networks is random due to random batches, initializations, and optimizers. Both the classical and optimization factor must be incorporated in order to have a calibrated uncertainty estimate.

DE are unable to do this. They can only capture the second source of uncertainty, the random optimization procedure, since all ensemble members are trained on the same data. Lakshminarayanan et al. (2017) were aware of this problem but noted that using a standard bootstrap, where each ensemble member is trained on resampled data, actually decreased performance. Nixon et al. (2020) ascribed this decrease in performance to effectively training on less unique data when bootstrapping.

Contribution: In this chapter, we present an efficient implementation of the parametric bootstrap for a regression setting that incorporates the missing source of uncertainty without affecting accuracy. We demonstrate that this leads to significantly better confidence intervals compared to standard DE and other popular methods.

Scope: We explicitly focus on a regression setting, in which we aim for more accurate confidence and prediction intervals. Our approach makes use of the separate estimates for the mean and variance that are given by Deep Ensembles in a regression setting, which does not translate to classification where only a single probability vector is given.

Organization: Section 3.2 describes the uncertainty framework that we use, gives a short overview of related work, and describes DE in more detail. This leads to Section 3.3 where we introduce our method, *Bootstrapped Deep Ensembles*. In Section 3.4, we experimentally demonstrate the significance of the effect of finite data and show that incorporating this improves the confidence

intervals significantly, which in turn results in improved prediction intervals. Finally, Section 3.5 summarises the conclusions and gives possible avenues for future work.

3.2 Background

3.2.1 Uncertainty Framework

We consider a frequentist regression setting in which a neural network is trained on a data set $\mathcal{D} = ((\boldsymbol{x}_1, y_1), \dots (\boldsymbol{x}_n, y_n))$ consisting of n independent realizations of the random variable pair (X, Y), with input $\boldsymbol{x} \in \mathbb{R}^d$ and target $y \in \mathbb{R}$. Given a new input $\boldsymbol{x}^*$, the neural network outputs a prediction $\hat{f}(\boldsymbol{x})$ for the corresponding target y^*.

We follow the framework that was introduced in Chapter 1 and is illustrated in Figure 3.1. The uncertainty in the prediction consists of three parts; distributional uncertainty, data uncertainty, and model uncertainty. In this chapter, we assume that the data set is representative and that our model class contains the true function. Our focus lies on improving the quantification of the parameter uncertainty.

The certainty in our predictions can be expressed via a confidence and prediction interval. Depending on the task, either the confidence or prediction interval may be of more interest. A confidence interval gives the region that is expected to cover the true function value, $f(\boldsymbol{x}^*)$. The prediction interval gives the region that is expected to cover a new observation, y^*. The confidence interval is determined only by the model uncertainty, whereas the prediction interval also depends on the data uncertainty.

In this chapter, we adhere to the frequentist interpretation of confidence and prediction intervals. A $(1-\alpha)\cdot 100\%$ confidence interval for $f(\boldsymbol{x}^*)$ is a random mapping from $\boldsymbol{x}^*$ to an interval such that if we repeated the entire experiment infinitely many times - that means sampling data, training the network, creating the interval - we would capture the true function value $f(\boldsymbol{x}^*)$ in $(1-\alpha)\cdot 100\%$ of the experiments. A confidence interval with a coverage higher than $(1-\alpha)\cdot 100\%$ is called *conservative*. Additionally, if this holds for all values of $\boldsymbol{x}$, it is called a *conditional* conservative confidence interval. A prediction interval is defined similarly but with y instead of $f(\boldsymbol{x})$. Bayesian methods typically output a credible interval. Although credible regions have fundamental

differences, it is desirable for a credible interval to maintain frequentist properties and it is common to evaluate Bayesian methods frequentistically (Ghosal and Van der Vaart, 2017).

Throughout this chapter, we take the fixed-covariates perspective, meaning that we treat $\boldsymbol{x}$ as being fixed and given and y as the realisation of a random variable. The classical uncertainty due to finite data is therefore the uncertainty due to the randomness of the targets. To make this perspective explicit, we will use the term *random targets* instead of *finite data*.

3.2.2 Related Work and Deep Ensembles

Numerous different methods to obtain uncertainty estimates for neural networks have been developed. We refer the reader to Abdar et al. (2021) for an extensive overview and list a few notable contributions here. **Bayesian Neural Networks** (MacKay, 1992a; Neal, 2012) put a prior distribution over the weights of a network and use the posterior to obtain uncertainty estimates. The calculation of the posterior is often intractable. **Variational Inference** (Hinton and Van Camp, 1993; Jordan et al., 1999) aims to solve this problem by using a tractable approximation of the posterior. **Monte-Carlo Dropout** (Gal and Ghahramani, 2016; Gal et al., 2017) is a notable example of variational inference. Since dropout is already used in many neural networks as a regularization technique, it comes at no extra cost at training time. A downside is that the epistemic uncertainty is only influenced by the dropout rate, thus making it impossible to locally tune the uncertainty estimates to have the correct size (Osband, 2016). **Quantile regression** (Cannon, 2011; Xu et al., 2017; Clements et al., 2019; Tagasovska and Lopez-Paz, 2019) uses a pinball loss to output quantiles directly without the need of any distributional assumptions. Similarly, direct Prediction Interval (PI) methods use a custom loss function that directly outputs PIs with the goal to capture the correct fraction of data points while being as narrow as possible (Pearce et al., 2018). Other methods are focused more on detecting **out-of-distribution** (OoD) samples. These are input values that are very different from the training data. A typical approach is to keep track of the pre-activations, the output of the penultimate layer (or sometimes also other layers) times the weight matrix plus the bias vector, and use some distance measure to determine the level of difference of a new datapoint (Van Amersfoort et al., 2021; Mukhoti et al., 2021; Lee et al., 2018). Similarly, Ren et al. (2019) use likelihood ratios directly on the inputs to detect out-of-distribution samples. OoD detection methods do not aim for calibrated

prediction or confidence intervals.

Deep Ensembles train an ensemble of M networks, each member receiving the same data but in a different order and having a different initialisation. The architecture of the networks is similar to the mean-variance estimation method by Nix and Weigend (1994), where each network outputs a mean, $\hat{f}_i(\boldsymbol{x})$, and variance prediction, $\hat{\sigma}_i^2(\boldsymbol{x})$, for every input. The variance terms $\hat{\sigma}_i^2(\boldsymbol{x})$ estimate the data uncertainty. The networks are trained by minimizing the negative loglikelihood of a normal distribution, which implies the following assumption.

Assumption 1. *The targets, y, are the sum of a function value $f(\boldsymbol{x})$ and normally distributed heteroscedastic noise:*

$$y = f(\boldsymbol{x}) + \epsilon, \quad \textit{with } \epsilon \sim \mathcal{N}\left(0, \sigma^2(\boldsymbol{x})\right).$$

Deep Ensembles assume a Gaussian Mixture of the individual models as the predictive model. In this model, the total mean and variance are defined as

$$\hat{f}_*(\boldsymbol{x}) = \frac{1}{M}\sum_{i=1}^{M} \hat{f}_i(\boldsymbol{x}), \quad \text{and}$$

$$\hat{\sigma}_*^2(\boldsymbol{x}) = \frac{1}{M}\sum_{i=1}^{M} \left(\hat{f}_i(\boldsymbol{x})^2 - \hat{f}_*(\boldsymbol{x})^2 + \hat{\sigma}_i^2(\boldsymbol{x})\right). \tag{3.1}$$

This results in the prediction interval

$$\mathrm{PI}_{\mathrm{DE}} = \hat{f}_*(\boldsymbol{x}) \pm z_{\alpha/2}\sqrt{\hat{\sigma}_*^2(\boldsymbol{x})},$$

where $z_{\alpha/2}$ is the $\alpha/2$ quantile of a standard normal distribution. For later comparison, we construct a confidence interval implied by DE by ignoring the aleatoric variance terms $\hat{\sigma}_i^2(\boldsymbol{x})$ in Equation (3.1) to arrive at

$$\mathrm{CI}_{\mathrm{DE}} = \hat{f}_*(\boldsymbol{x}) \pm t_{\alpha/2}^{(M-1)} \sqrt{\frac{1}{M}\sum_{i=1}^{M} \hat{f}_i(\boldsymbol{x})^2 - \hat{f}_*(\boldsymbol{x})^2},$$

where $t_{\alpha/2}^{(M-1)}$ is the $\alpha/2$ quantile of a t distribution with $M-1$ degrees of freedom.

Deep Ensembles have been shown to clearly outperform Variational Inference and Monte-Carlo Dropout (Lakshminarayanan et al., 2017) and are regarded

the state of the art for uncertainty estimation, both in-distribution (Ashukha et al., 2019) and under distributional shift (Ovadia et al., 2019).

Different explanations for the success of Deep Ensembles have been given. Wilson and Izmailov (2020) relate the method to a form of Bayesian model averaging. They empirically demonstrate that DE are even able to better approximate the predictive distribution than some standard Bayesian approaches. Similarly, Gustafsson et al. (2020) relate the method to sampling from an approximate posterior. Alternatively, Fort et al. (2019) explain the success via the loss landscape. They argue that the different models are able to explore different local minima, where a Bayesian approximation may only explore a single local minimum.

None of these interpretations fully explains why the obtained intervals would be properly calibrated. In fact, as we will also show in our experiments, by training each ensemble member on the same data, DE ignore a significant part of the epistemic uncertainty that is due to finite data (the classical box in Figure 3.1). In the next section, we introduce our method, *Bootstrapped Deep Ensembles*, an easy to implement extension of DE with comparable computational costs that does take this source of epistemic uncertainty into account.

3.3 Bootstrapped Deep Ensembles

Our method can be summarised in two steps. We first train a regular Deep Ensemble, resulting in the exact same predictor $\hat{f}_*(\boldsymbol{x})$ and thus identical accuracy. Secondly, we repeat a small part of the training of these members on new data, more on this shortly, in order to capture the uncertainty that we missed by training the ensemble members on identical data.

As previously stated, we model a neural network as a random predictor with an error that decomposes in a part due to the optimization procedure and a part due to random targets. We formalize this in the following assumption.

Assumption 2. *Let $\hat{f}_i(\boldsymbol{x})$ be the prediction of an ensemble member trained on the same data set $\mathcal{D}$, but with a unique initialization and data ordering, and let $f(\boldsymbol{x})$ be the true value, then*

$$\hat{f}_i(\boldsymbol{x}) = f(\boldsymbol{x}) + \epsilon_{\text{classical}} + \epsilon_{\text{optim},i}, \quad \textit{with}$$

$$\epsilon_{\text{classical}} \sim \mathcal{N}\left(0, \sigma^2_{\text{classical}}(\boldsymbol{x})\right) \textit{ and } \epsilon_{\text{optim},i} \sim \mathcal{N}\left(0, \sigma^2_{\text{optim}}(\boldsymbol{x})\right),$$

where all ϵ are independent. The $\epsilon_{classical}$ term does not have an index i since by definition it is the same for all ensemble members.

The $\epsilon_{\text{classical}}$ and $\sigma^2_{\text{classical}}(\boldsymbol{x})$ terms relate to the uncertainty in box *"Classical"* of Figure 3.1 and the terms $\epsilon_{\text{optim},i}$ and $\sigma^2_{\text{optim}}(\boldsymbol{x})$ to the *"Optimization"* box. From Assumption 2, it follows that the average of ensemble members trained on the same data, $f_*(\boldsymbol{x})$, has variance

$$\mathbb{V}(\hat{f}_*(\boldsymbol{x})) = \sigma^2_{\text{classical}}(\boldsymbol{x}) + \frac{\sigma^2_{\text{optim}}(\boldsymbol{x})}{M}. \tag{3.2}$$

The $\sigma^2_{\text{classical}}(\boldsymbol{x})$ term does not get divided by M since all ensemble members are trained on the same data.

One way to estimate the total variance (Equation 3.2) is to train multiple (deep) ensembles. However, this would become extremely expensive. The contribution of this chapter is that, with only a modest amount of extra work, we are able estimate the total variance. The key ingredient of our approach is to *separately* estimate the two terms $\sigma^2_{\text{classical}}(\boldsymbol{x})$ and $\sigma^2_{\text{optim}}(\boldsymbol{x})$.

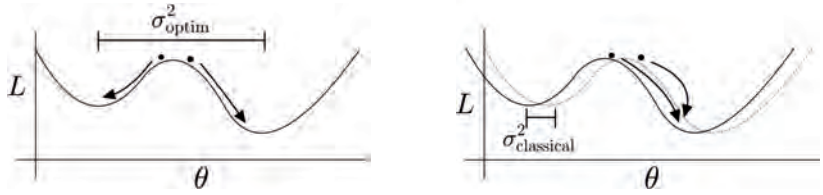
3.3.1 Incorporating the Missing Uncertainty

The estimate for $\sigma^2_{\text{optim}}(\boldsymbol{x})$ is straightforward. Since the M ensemble members are trained on the same data set, we can take the sample variance of these ensemble members as an estimate of the variance due to the random optimization:

$$\hat{\sigma}^2_{\text{optim}}(\boldsymbol{x}) = \frac{1}{M-1}\sum_{i=1}^{M}\left(\hat{f}_i(\boldsymbol{x}) - \frac{1}{M}\sum_{i=1}^{M}\hat{f}_i(\boldsymbol{x})\right)^2.$$

Deep Ensembles, however, fail to measure $\sigma^2_{\text{classical}}(\boldsymbol{x})$. To estimate the missing $\sigma^2_{\text{classical}}(\boldsymbol{x})$, we propose to use an adapted version of the *parametric bootstrap* (Efron, 1982). For a standard parametric model, the parametric bootstrap consists of two steps. A single model is trained on the data, after which B additional models are trained using new data, simulated from the first model. The variance of those extra models is then used to obtain the model uncertainty.

Directly translating the parametric bootstrap to our setup does not work. Training a new network on simulated targets would also capture the variance due to the optimization procedure. As indicated before, a solution could be



(a) Effect of random optimization (b) Effect of random targets

Figure 3.2: Figure (a) sketches the effect of the random optimization procedure. Through random initializations and random orderings of the data, different regions of the loss landscape get explored. We denote the variance that arises from this effect with σ^2_{optim}. With finite training data, our loss landscape itself is subject to randomness. To estimate the resulting uncertainty, we apply the parametric bootstrap, resulting in new targets and a slightly deformed loss landscape. To ensure we end up in the deformed version of the same local minimum, we repeat only a part of the training.

to estimate the entire variance in Equation (3.2) directly by training B entire ensembles on simulated data sets, but this would be far too expensive. We therefore propose an approach to *train additional neural networks while eliminating optimization variability.*

To explain how we do this, we examine the problem from a loss landscape perspective, as sketched in Figure 3.2. The random optimization causes the networks to end up in different local minima, while different targets cause the loss landscape to deform. Starting from a later point in the training cycle - as opposed to starting at initialisation - is much more likely to cause the retrained network to end up in the deformed version of the *same* local minimum, thus eliminating optimization variability.

In order to estimate $\sigma^2_{\text{classical}}(\boldsymbol{x})$, we therefore propose the following procedure. During the training of the original ensemble members, we save a copy of the state of the network after $\lfloor n_{\text{epoch}} \cdot (1-r) \rceil$ epochs, where n_{epoch} is total amount of training epochs and $r \in [0, 1]$ is the retraining fraction. We then repeat the final $r \cdot n_{\text{epoch}}$ epochs, starting from the saved state, with new targets that are simulated from a $\mathcal{N}(\hat{f}_i(\boldsymbol{x}), \hat{\sigma}^2_i(\boldsymbol{x}))$ distribution. We denote this retrained network with $\hat{\hat{f}}_i(\boldsymbol{x})$.

This retraining is meant to capture solely the variance due to the random targets. A standard assumption of the parametric bootstrap is that the distribu-

tions of the difference of the first model and the true model, and the difference of the retrained model and the first model, are similar (Efron, 1982). In our case, this assumption translates to:

Assumption 3. *Let $\hat{\hat{f}}_i(\boldsymbol{x})$ denote the predictions of a retrained ensemble member. The difference between $\hat{f}_i(\boldsymbol{x})$ and $\hat{\hat{f}}_i(\boldsymbol{x})$ is normally distributed with zero mean and variance $\sigma^2_{\text{classical}}(\boldsymbol{x})$:*

$$\hat{\hat{f}}_i(\boldsymbol{x}) = \hat{f}_i(\boldsymbol{x}) + \epsilon_{\text{classical},i} \quad \textit{with} \quad \epsilon_{\text{classical},i} \sim \mathcal{N}\left(0, \sigma^2_{\text{classical}}(\boldsymbol{x})\right).$$

As an estimate for $\sigma^2_{\text{classical}}(\boldsymbol{x})$, we therefore use

$$\hat{\sigma}^2_{\text{classical}}(\boldsymbol{x}) = \frac{1}{M}\sum_{i=1}^{M}\left(\hat{f}_i(\boldsymbol{x}) - \hat{\hat{f}}_i(\boldsymbol{x})\right)^2.$$

In total, we have made three assumptions. Assumption 1 is a modeling assumption that states that we are dealing with additive Gaussian heteroscedastic noise. This assumption is very standard and made by most works on uncertainty estimation (e.g. Deep Ensembles and Concrete Dropout). Assumption 2 states that we assume the model uncertainty to be normally distributed and to consist of a classical part and an optimization part. This normality is also a very standard assumption. The typical reasoning behind this assumption is that it holds asymptotically for a parametric model and is therefore the most sensible choice, also for finite data and non-parametric models (see Appendix 3.B for more details). The same asymptotic normality can be shown for the third assumption, which is a common assumption of the parametric bootstrap (Efron, 1982). We observed that these assumptions hold empirically in most of our simulations, as we demonstrate in Appendix 3.B. The coverage values that we obtained, given in Section 3.4, add to the plausibility of these assumptions.

The entire method is summarised in Algorithm 5. With relatively little extra effort - we only need to train the equivalent of $M(1+r)$ networks, with $r < 1$ - we are able to get uncertainty estimates that translate to confidence and prediction intervals that are better theoretically founded, as is substantiated in the next subsection with a proof that the confidence intervals are guaranteed to be conservative, and empirically result in a better coverage, as is demonstrated in Section 3.4.

Algorithm 5 Pseudo-code to obtain a confidence interval with Bootstrapped Deep Ensembles

Input: M - number of ensembles, n_{epoch} - number of training epochs, r - retrain fraction, $\mathcal{D}$ - data set;
for $i = 1$ **to** M **do**
  Train ensemble member i on $\mathcal{D}$ with random initialisation and data ordering to obtain $\hat{f}_i(\boldsymbol{x})$ and $\hat{\sigma}_i^2(\boldsymbol{x})$, while saving the model and optimizer state after $n_{\text{epoch}}(1-r)$ training epochs;
  Simulate new targets: $\tilde{y}_j \sim \mathcal{N}\left(\hat{f}_i(\boldsymbol{x}_j), \hat{\sigma}_i^2(\boldsymbol{x}_j)\right)$;
  Repeat the final $r \cdot n_{\text{epoch}}$ training epochs on $\tilde{\mathcal{D}} = ((\boldsymbol{x}_1, \tilde{y}_1), \ldots, (\boldsymbol{x}_n, \tilde{y}_n))$, obtaining $\hat{\tilde{f}}_i(\boldsymbol{x})$;
end for
$\hat{f}_*(\boldsymbol{x}) := \frac{1}{M}\sum_{i=1}^{M} \hat{f}_i(\boldsymbol{x})$;
$\hat{\sigma}^2_{\text{classical}}(\boldsymbol{x}) := \frac{1}{M}\sum_{i=1}^{M}\left(\hat{f}_i(\boldsymbol{x}) - \hat{\tilde{f}}_i(\boldsymbol{x})\right)^2$;
$\hat{\sigma}^2_{\text{optim}}(\boldsymbol{x}) := \frac{1}{M-1}\sum_{i=1}^{M}\left(\hat{f}_*(\boldsymbol{x}) - \hat{f}_i(\boldsymbol{x})\right)^2$;
Calculate the $1-\alpha$ confidence interval:
$\text{CI}^{(\alpha)}(\boldsymbol{x}) = \left[\hat{f}_*(\boldsymbol{x}) \pm t_{\alpha/2}^{(M-1)}\sqrt{\hat{\sigma}^2_{\text{classical}}(\boldsymbol{x}) + \frac{\hat{\sigma}^2_{\text{optim}}}{M}}\right]$;

3.3.2 Creating Confidence and Prediction Intervals

The following theorem, proven in Appendix 3.A, states that, under Assumptions 2 and 3, the conditional confidence interval given in Algorithm 5 is conservative.

Theorem 3.3.1. *Following the notation introduced above, let $\hat{f}_*(\boldsymbol{x})$ be the average of the M ensemble members with predictions $\hat{f}_i(\boldsymbol{x})$. Let $\hat{\tilde{f}}_i(\boldsymbol{x})$ be the prediction of ensemble member i after a part of the training is repeated with newly simulated targets. Define $\hat{\sigma}^2_{\text{optim}}(\boldsymbol{x})$ and $\hat{\sigma}^2_{\text{classical}}(\boldsymbol{x})$ as in Algorithm 5. Under Assumptions 2 and 3, with probability at least $(1-\alpha)\cdot 100\%$:*

$$f(\boldsymbol{x}) \in \hat{f}_*(\boldsymbol{x}) \pm t_{\alpha/2}^{(M-1)}\sqrt{\hat{\sigma}^2_{\text{classical}}(\boldsymbol{x}) + \frac{\hat{\sigma}^2_{\text{optim}}(\boldsymbol{x})}{M}}, \tag{3.3}$$

where $t_{\alpha/2}^{(M-1)}$ is the critical value of a t-distribution with $M-1$ degrees of freedom.

Algorithm 6 Pseudo-code to obtain a prediction interval with Bootstrapped Deep Ensembles

Input: Trained and retrained ensemble members, N_t - number of samples to base the prediction interval on.
$\hat{\sigma}^2(\boldsymbol{x}) = \frac{1}{M}\sum_{i=1}^{M}\hat{\sigma}_i^2(\boldsymbol{x})$;
for $j = 1$ **to** N_t **do**
  $t_j \sim t(M-1)$;
  $\mu_j(\boldsymbol{x}) = \hat{f}_*(\boldsymbol{x}) + t_j\sqrt{\hat{\sigma}^2_{\text{classical}}(\boldsymbol{x}) + \frac{\hat{\sigma}^2_{\text{optim}}(\boldsymbol{x})}{M}}$;
  $y_j \sim \mathcal{N}\left(\mu_j(\boldsymbol{x}), \hat{\sigma}^2(\boldsymbol{x})\right)$;
end for
Take the $(1-\alpha/2)$ and $\alpha/2$ empirical quantiles of all y_j as the bounds of the PI

The confidence interval from Theorem 3.3.1 can be easily extended to a prediction interval. The prediction interval combines the aleatoric uncertainty, governed by a normal distribution with variance $\sigma^2(\boldsymbol{x})$, with the epistemic uncertainty, a scaled student distribution with $M-1$ degrees of freedom. Algorithm 6 describes a simple Monte-Carlo sampling procedure to quickly estimate empirical quantiles of the resulting distribution.

3.4 Experimental Results

In this section, we empirically examine the quality of our confidence and prediction intervals. We first explain why and how we simulated data for our experiments. We then go through our three experiments that - 1 - show that the obtained confidence and prediction intervals have a typically better, or in some cases at least similar, coverage compared to other popular methods, - 2 - demonstrate the significant effect of random targets on the total uncertainty to underline the importance of incorporating this effect, and - 3 - show that our method is able to correctly estimate the separate variances due to random optimization and targets.

In the appendix we provide additional experimental results. Specifically, we also test the effect of differently distributed noise, a different simulation method, and different retraining fractions. We observe that our method works well for a variety of data sets using retraining fractions between 0.2 and 0.4, meaning

that we do not need to tune it and can simply pick a default value.

3.4.1 Simulating Data

In order to compare confidence intervals, it is necessary to know the true function values. A simple toy experiment would meet this requirement but is likely not representative for a real-world scenario. To overcome this, we created simulations based on the regression benchmark data sets used in Hernández-Lobato and Adams (2015). These data sets were also used in other works on uncertainty estimation (Gal and Ghahramani, 2016; Lakshminarayanan et al., 2017; Mancini et al., 2020; Liu and Wang, 2016; Salimbeni and Deisenroth, 2017; Khosravi et al., 2011; Pearce et al., 2020; Su et al., 2018) and have become the standard benchmark data sets for regression uncertainty quantification. We take one of these real-world data sets, for instance Boston Housing, and train a random forest to predict y given $\boldsymbol{x}$, and we use this model as the true function $f(\boldsymbol{x})$. We then train a second forest to predict the residuals squared $(y - f(\boldsymbol{x}))^2$ and use this forest as the true variance $\sigma^2(\boldsymbol{x})$. Using these $f(\boldsymbol{x})$ and $\sigma^2(\boldsymbol{x})$, we can simulate new targets from a $\mathcal{N}\left(f(\boldsymbol{x}), \sigma^2(\boldsymbol{x})\right)$ distribution. We used random forests with 100 trees and a max depth of 3. The simulating procedure is summarized in Algorithm 7.

3.4.2 Training Procedure

We used neural networks with three hidden layers having 40, 30, and 20 units respectively, ReLU activations functions in these hidden layers, and a linear activation function in the final layer. To ensure positivity of $\hat{\sigma}$ we used an exponential transformation and added a minimum value of 1e-3 for numerical stability for the ensemble networks. Each network was trained for 80 epochs

Algorithm 7 Pseudo-code to simulate data

1: Train a random forest on $\mathcal{D}$ and use this forest as the true function $f(\boldsymbol{x})$;
2: Calculate the residuals, $(y_i - f(\boldsymbol{x}_i))$;
3: Train a second random forest on the squared residuals and use this function for the true variance $\sigma^2(\boldsymbol{x})$;
4: Simulate new targets: $\tilde{y}_i \sim \mathcal{N}\left(f(\boldsymbol{x}_i), \sigma^2(\boldsymbol{x}_i)\right)$;
5: **Return:** $\mathcal{D}_{\text{new}} = ((\boldsymbol{x}_1, \tilde{y}_1), \ldots, (\boldsymbol{x}_n, \tilde{y}_n))$;

with a batchsize of 32 using the ADAM optimizer. We used the same the train_test_split function from the scikit learn package with random seed 1 for our train/test splits.

This setup is very similar to those used in Gal and Ghahramani (2016) and Hernández-Lobato and Adams (2015) with the exception that we use more than one hidden layer. We do this because we observed a much larger bias when using only one layer. We stress that these are typical architectures and data sets for work on uncertainty quantification in a regression setting. Uncertainty estimation methods for regression are typically evaluated on smaller data sets and with smaller architectures than classification methods.

We used $r = 30\%$ and $M = 5$. We used l_2 regularization with a standard constant of $1/(\#\text{Training Samples})$. The networks used for the Concrete Dropout and Quality Driven Ensembles methods were trained for 240 epochs each since we found that these networks needed longer to converge. All training data was standardized to have zero mean and unit variance before training. All testing was done on the original scales.

3.4.3 Experiment 1: Simulations Based on Benchmark Data Sets

Our first experiment compares the coverage of our confidence and prediction intervals with different popular methods. Our prediction intervals are compared to DE, Concrete Dropout (CD) (Gal et al., 2017), and Quality-Driven Ensembles (QDE) (Pearce et al., 2018). Our confidence intervals are compared to DE, the Naive Bootstrap (NB) (Efron, 1982; Heskes, 1997), and Concrete Dropout.

We test the coverage by using 100 simulated data sets instead of the popular practice of using a single test set. The latter tests empirical coverage of the intervals on a previously unseen part of a single real-world data set (Khosravi et al., 2011; Pearce et al., 2020; Su et al., 2018). However, we argued that it is not sufficient to evaluate the quality of prediction and confidence intervals in this manner on a test set. As discussed in Chapter 2, this only checks the overall coverage, which is relatively easy to tune, and not the quality of the conditional confidence intervals. The proposed alternative is to use simulated data and calculate the coverage per $\boldsymbol{x}$-value over a large number of simulations.

This gives rise to the Confidence Interval Coverage Fraction (CICF):

$$\mathrm{CICF}(\boldsymbol{x}) := \frac{1}{n_{\mathrm{sim}}} \sum_{j=1}^{n_{\mathrm{sim}}} \mathbb{1}_{f(\boldsymbol{x}) \in [LC^{(j)}(\boldsymbol{x}), RC^{(j)}(\boldsymbol{x})]}, \tag{3.4}$$

where n_{sim} is the number of simulations, $f(\boldsymbol{x})$ is the true function value, and $LC^{(j)}(\boldsymbol{x}), RC^{(j)}(\boldsymbol{x})$ are the lower and upper limit of the CI of $f(\boldsymbol{x})$ in simulation j. If our CI has the correct conditional coverage, the $\mathrm{CICF}(\boldsymbol{x})$ should be close to $1-\alpha$ for each value of $\boldsymbol{x}$. The following Brier score – with a perfect score of 0 meaning a CICF of $1-\alpha$ for each individual value of $\boldsymbol{x}$ – captures this:

$$\mathrm{BS} = \frac{1}{n_{\mathrm{test}}} \sum_{i=1}^{n_{\mathrm{test}}} \left(\mathrm{CICF}(\boldsymbol{x}_i) - (1-\alpha)\right)^2. \tag{3.5}$$

Evaluating the quality of the prediction interval is done similarly. We define the Prediction Interval Coverage Fraction:

$$\mathrm{PICF}(\boldsymbol{x}) := \frac{1}{n_{\mathrm{sim}}} \sum_{j=1}^{n_{\mathrm{sim}}} \mathbb{P}\left(Y \in [L^{(j)}(\boldsymbol{x}), R^{(j)}(\boldsymbol{x})]\right), \tag{3.6}$$

and report the resulting Brier scores. Additionally, we report the average widths of the intervals. In case of a comparable Brier score, we favor the method with smaller intervals. In summary, we create 100 new data sets, create 100 prediction and confidence intervals for each $\boldsymbol{x}$-value in the test set, and check how often the intervals contain the true value.

The results are presented in Table 3.1, with Figure 3.4 offering a visual summary of the various methods' performance. The Brier scores of all data sets are plotted against the other methods. BDE perform better for all instances above the dotted diagonal line. Additionally, the distance to the diagonal gives an indication about the difference in performance.

Several trends emerge from these results. Notably, our method clearly improves upon Deep Ensembles, producing superior Brier scores for the confidence intervals in seven of the eight data sets and for the prediction intervals in six of the eight data sets. For the other three data sets, the performance is very similar. The performance gain is often substantial. For example, on the Concrete data set, the Brier scores for the CICF range from 0.026 (BDE) to 0.33 (NB). To illustrate the significance of these differences, we visualised the individual CICF scores as a violin plot in Figure 3.3. We observe that for most values of $\boldsymbol{x}$ the BDE confidence intervals are close to the correct size.

Table 3.1: Results of the comparison of our method Bootstrapped Deep Ensembles (BDE) to Deep Ensembles (DE), the Naive Bootstrap (NB), Concrete Dropout (CD) and Quality Driven Ensembles (QDE). A total of 100 simulated data sets were used to calculate the metrics. On each new data set, new ensembles were trained, and new confidence and prediction intervals were constructed. Brier-CI80 denotes the Brier score of the CICF (Equation (3.4)) of an 80% confidence interval. Brier-PI80 denotes the equivalent quantity for the PICF (Equation (3.6)). We observe comparable Brier scores for most prediction intervals and considerably lower scores for all but two confidence intervals. The confidence intervals of BDE are wider but this is justified by the superior coverage. BDE and DE have identical RMSE since they use the same predictor. The RMSE value of CD was in general extremely similar. NB had a notably larger RMSE (often around 20%). This is to be expected since each ensemble member is effectively being trained on less data due to the resampling. All RMSE values, as well as a further comparison to DE, including training without regularization, with a different base simulation model, and differently distributed noise, can be found in Appendix 3.C. Bold values are used to indicate the overall best brier score and underscored values are used to compare Bootstrapped Deep Ensembles and regular Deep Ensembles.

SIMULATION	**Brier-CI80** ↓				**Brier-PI80** ↓				**Width CI80**				**Width PI80**			
	BDE	DE	NB	CD	BDE	DE	QDE	CD	BDE	DE	NB	CD	BDE	DE	QDE	CD
		$\times 10^{-2}$				$\times 10^{-2}$										
Boston	7.84	11.1	18.1	**7.78**	2.44	3.48	3.13	**1.49**	3.38	2.98	3.95	3.72	7.64	7.15	14.8	8.63
Concrete	**2.65**	8.21	32.5	14.6	**0.803**	1.51	1.47	2.62	8.75	6.39	7.93	6.00	22.7	21.1	39.3	20.1
Energy	3.96	6.67	37.2	**1.54**	0.805	1.32	1.73	**0.587**	2.33	1.84	2.03	2.10	5.55	5.15	17.7	5.40
Kin8nm	0.780	**0.640**	51.3	45.8	**0.127**	0.248	0.928	3.11	0.167	0.141	0.117	0.0237	0.533	0.512	0.687	0.437
Naval	**2.13**	4.59	58.6	29.6	0.244	0.188	1.15	**0.111**	2.84^{1}	1.84^{1}	0.919^{1}	1.64^{1}	0.0317	0.0314	0.0430	0.0312
Power	**8.49**	12.4	32.8	36.3	0.0785	**0.0658**	2.26	0.298	3.06	2.18	2.83	0.780	13.2	12.9	23.1	12.2
Yacht	11.5	14.3	22.3	**10.8**	**2.84**	4.29	12.5	3.72	2.72	2.41	3.49	7.27	5.53	5.05	18.0	14.4
Wine	**0.895**	2.57	8.63	25.9	0.954	1.51	**0.800**	6.18	0.504	0.448	0.685	0.273	1.43	1.36	2.08	1.12

[1] $\times 10^{-3}$

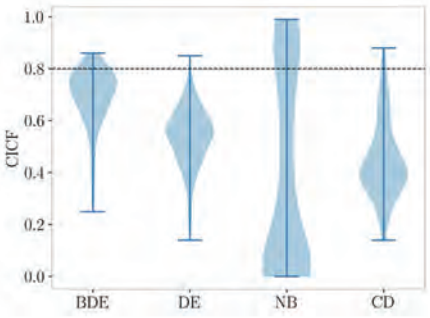


Figure 3.3: Violin plot of the individual CICF($\boldsymbol{x}$) values for 80% CIs, calculated on the test set of the *Concrete* simulation. Each CICF value was obtained using 100 simulations. Violin plots of all other simulations, including for the PICF values, can be found in 3.C. Note that perfect coverage would correspond to a sharp peak at $1 - \alpha$. It can be seen that the confidence intervals for Deep Ensembles (DE) tend to be too optimistic, those for Concrete Dropout (CD) are often far too optimistic, and those for the Naive Bootstrap (NB) are all over the place.

Moreover, our method is very robust. While other methods sometimes perform very poorly, as indicated by the large deviation from the diagonal line in Figure 3.4, our method does not. In particular, our confidence intervals are almost always better than those generated by the other methods and in the few cases that they are not, they still perform almost as well. The confidence intervals of Concrete dropout, for instance, perform similar on one data set and slightly better on two data sets, but dramatically underperform on the remaining five.

To check our assumptions, we kept track of all the predictions of the first ensemble member before and after retraining in each of the 100 simulations of the Boston Housing part of experiment 1. Figure 3.5 shows two typical scenarios. For most values of $\boldsymbol{x}$, we found that the errors are indeed normally distributed and that the variance of $\hat{f}_i(\boldsymbol{x}) - f(\boldsymbol{x})$ is slightly larger than the variance of $\hat{\hat{f}}_i(\boldsymbol{x}) - \hat{f}_i(\boldsymbol{x})$. This is expected, since the former also has variance due to the randomness of the training. We also see, however, that for some values of $\boldsymbol{x}$ we get a large bias term, violating Assumption 2. Figure 3.7 in 3.B shows these plots for the first 28 data points in the Boston Housing test set.

This violation of Assumption 2 explains that some of the BDE confidence intervals have a low coverage. We note that Deep Ensembles have the same problem, and that also on points with high bias, our method has better CICF values.

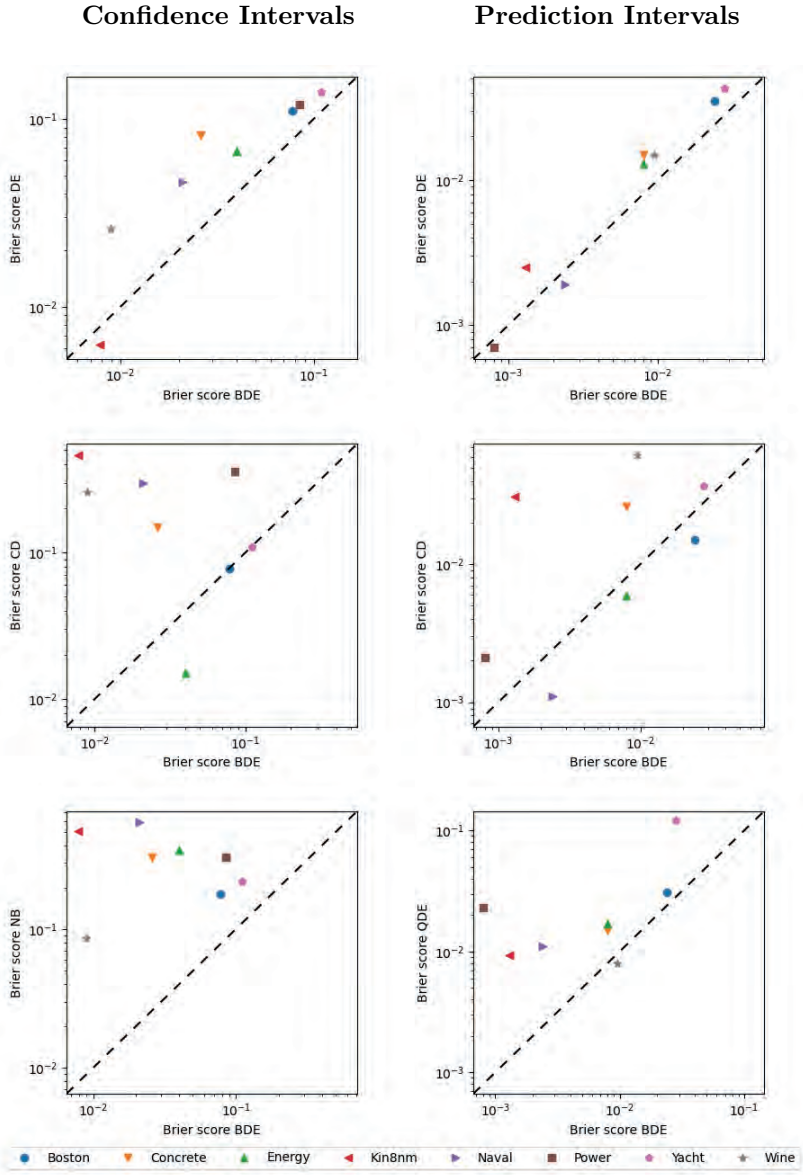
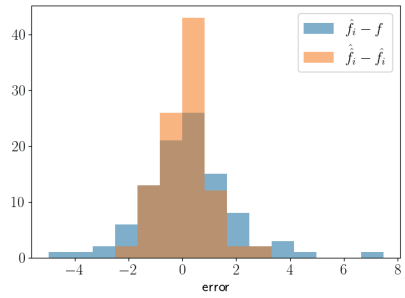
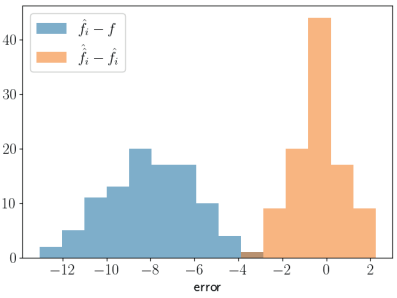


Figure 3.4: Comparison of our method Bootstrapped Deep Ensembles (BDE) to Deep Ensembles (DE), the Naive Bootstrap (NB), Concrete Dropout (CD) and Quality Driven Ensembles (QDE). The Brier scores (see Table 3.1) of BDE are plotted against those of the other methods. Our method has superior performance for all instances above the dotted diagonal. Additionally, the distance to the dotted line gives an indication of the difference in model performance. BDE clearly improves upon DE and is generally very robust.



(a) Data point 13 in the BostonHousing test set



(b) Data point 15 in the BostonHousing test set

Figure 3.5: Empirical example of Assumptions 2 and 3. Each histogram is made by evaluating $\hat{f}_1(\boldsymbol{x}) - f(\boldsymbol{x})$ and $\hat{\hat{f}}_1(\boldsymbol{x}) - \hat{f}_1(\boldsymbol{x})$ for a single value of $\boldsymbol{x}$ on 100 simulated data sets. In (a) we see that the assumptions appear to hold. We have normally distributed errors with a slightly smaller variance after retraining. This should be the case since the aim of the retraining is to only capture the variance due to the random targets and not the random training. In (b), however, we see that for some values of $\boldsymbol{x}$, a large bias can occur.

3.4.4 Experiment 2: Relative Effect of Random Targets

To further motivate the benefit of our method, we compared the relative contributions of random optimization and random data to the total variance of a neural network. As argued above, a neural network can be seen as a random predictor. This randomness is partially a consequence of the random optimization procedure and initialisation, which is captured well by Deep Ensembles. However, especially for small training sets, the classical variance due to random targets is also a significant part of the total variance.

We set up a simulation based on the large Protein-tertiary-structure data set. We trained two sets of 50 networks. The first set was trained with different targets for each network and the second was trained with the same targets for each network. The random targets were simulated using the two random forests. We subsequently examined the average variance of the networks with fixed and random targets on 5000 previously unseen test points. The average variance of the 50 networks trained on random targets gives an estimate of $\sigma^2_{\text{optim}} + \sigma^2_{\text{classical}}$, and the average variance of the 50 networks trained on the

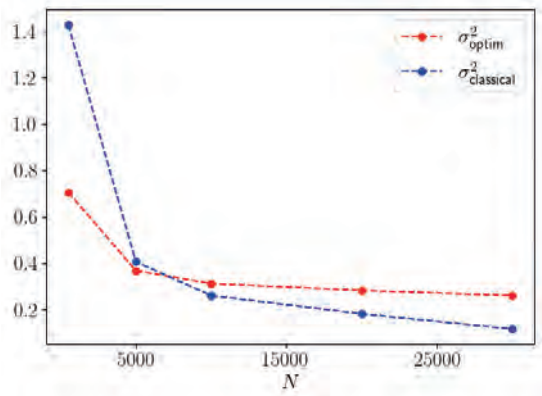


Figure 3.6: The effect of random targets and random optimization on the total variance of the predictor $\hat{f}(\boldsymbol{x})$, trained on the *Protein* data set. For small N, the effect of random targets is dominant, and even for quite a large number of data points, it is still significant. The variance due to random optimization, σ^2_{optim}, is obtained by examining the variance of 50 networks trained on the same data. Another 50 networks are trained on 50 different data sets, each with newly simulated targets. The variance of the second group is an estimate of $\sigma^2_{\text{optim}} + \sigma^2_{\text{classical}}$.

same targets gives an estimate of σ^2_{optim}. We took the difference of the two as an estimate of $\sigma^2_{\text{classical}}$. This process was repeated multiple times on an increasing number of data points n.

Figure 3.6 shows that with less than 5000 data points, the classical variance due to random targets is the dominant part of the total variance, and even with a lot of training data, the effect of random targets is still significant. This significant effect is not taken into account by standard DE, which explains the subpar coverage found in the first experiment (see Table 3.1 and Figure 3.4).

3.4.5 Experiment 3: Examining the Separate Estimates

Our method gives separate estimates for $\sigma^2_{\text{classical}}(\boldsymbol{x})$ and $\sigma^2_{\text{optim}}(\boldsymbol{x})$. To test how well BDE can capture both components, we trained a BDE with $M = 50$ and $r = 30\%$ on a single data set, simulated with the two random forests. The variance of these 50 networks before retraining gives an estimate for $\hat{\sigma}^2_{\text{optim}}(\boldsymbol{x})$. Through our retraining step, we get an estimate for $\hat{\sigma}^2_{\text{classical}}(\boldsymbol{x})$. Subsequently, we trained 50 networks using newly simulated targets for each network. These targets were simulated using the random forests. The variance of the pre-

dictions of those second 50 networks gives an estimate of the ground truth $\sigma^2_{\text{classical}}(\boldsymbol{x})+\sigma^2_{\text{optim}}(\boldsymbol{x})$. If our assumptions are correct, our estimate $\hat{\sigma}^2_{\text{classical}}(\boldsymbol{x})$ should be roughly the difference between $\sigma^2_{\text{classical}}(\boldsymbol{x})+\sigma^2_{\text{optim}}(\boldsymbol{x})$ and $\sigma^2_{\text{optim}}(\boldsymbol{x})$.

Table 3.2 shows that our separate estimates for the variances due to random targets and due to random training sum correctly - within approximately 10% - to the true variance when training with random targets. This shows that our approach of repeating a part of the training on new targets is an effective method to incorporate the uncertainty due to random targets without affecting accuracy.

3.4.6 Limitations

We end the results section by noting some of the limitations of our method. Most notably, it is important to realize that our assumptions will not always hold. The goal of our method was to incorporate the classical uncertainty that is a consequence from the fact that we are training on a random data set. In order to translate the predictions of the ensemble members to a confidence interval, we must make some distributional assumptions. Our assumptions are theoretically motivated by asymptotic analysis for parametric models (see 3.B), but are not guaranteed to always hold in practice.

In particular, the unbiasedness assumption will not always hold. This is a problem with ensembling in general, and not specific to our method. If all ensemble members have a certain bias, then the corresponding confidence interval can easily have a very low coverage. Similarly, if the additive noise is not Gaussian, then the coverage can be imperfect. We investigated this in more detail in 3.C

Table 3.2: The quality of the average estimate for $\sigma^2_{\text{classical}}(\boldsymbol{x})$ on 4 different simulated data sets. The first column gives the ground truth of $\sigma^2_{\text{classical}}+\sigma^2_{\text{optim}}$, obtained by training with random targets. The sum of the estimates $\hat{\sigma}^2_{\text{optim}}$ and $\hat{\sigma}^2_{\text{classical}}$ match the ground truth within roughly 10%.

Simulation	$\sigma^2_{\text{classical}}+\sigma^2_{\text{optim}}$	$\hat{\sigma}^2_{\text{optim}}$	$\hat{\sigma}^2_{\text{classical}}$	$\hat{\sigma}^2_{\text{classical}}+\hat{\sigma}^2_{\text{optim}}$
Boston	2.44	1.52	0.91	2.43
Concrete	13.03	6.45	6.02	12.47
Energy	0.92	0.50	0.45	0.95
kin8nm	2.7e-3	1.7e-3	1.6e-3	3.3e-3

and found this effect to be noticeable but much less substantial than the effect of a large bias.

Our method is 30% more expensive than regular Deep Ensembles. While our tailor-made bootstrapping approach is far more efficient than the alternative of training an ensemble of ensembles, this 30% may be significant depending on the application. Additionally, at inference time, a total of $2M$ forward passes needs to be made, in comparison to M for regular Deep Ensembles.

3.5 Conclusion

In this chapter, we presented our uncertainty estimation method Bootstrapped Deep Ensembles. The BDE confidence intervals have much better coverage than those obtained with standard DE or other popular methods, at a price of just 30% more training time. BDE improves upon DE because it incorporates the epistemic uncertainty due to the randomness of the training targets, where DE only captures the randomness of the optimization procedure. Our simulations show that the randomness of the training targets is substantial, even for larger data sets.

Where, based on asymptotic statistical theory, one would expect this variance to be inversely proportional to the number of data points, we observed a slower decay, closer to $\frac{1}{\sqrt{N}}$. It would be interesting to study the (asymptotic) behavior of these two components in more detail, also to be able to judge when one can indeed be neglected compared to the other. As a potential bonus, to be investigated in more detail in future work, our method appears to better detect overfitting than standard DEs. Arguments and initial empirical evidence can be found in 3.D.

In regions with relatively little data, confidence intervals tend to get larger. In general, however, we would like to discourage the use of confidence intervals for out-of-distribution detection: confidence intervals may get wider for quite different reasons, in particular when we allow for heteroscedastic noise, and, perhaps more importantly, they simply cannot be trusted in regions with little training data, since the underlying assumptions on which they are based are doomed to be violated. A more promising avenue for future work is to combine our method with an orthogonal approach, specifically for OoD detection (such as, for example, Ren et al. 2019).

APPENDIX CHAPTER 3

This appendix consists of four parts. In Section 3.A, we provide the proof of Theorem 3.3.1. In Section 3.B, we motivate the assumptions on which Bootstrapped Deep Ensembles rely. We provide additional experimentation in Section 3.C and briefly investigate the possibility to detect overfitting in Section 3.D.

3.A Proof of Theorem 3.3.1

Proof The result follows by evaluating

$$T^2 = \frac{(f(\boldsymbol{x}) - \hat{f}_*(\boldsymbol{x}))^2}{\hat{\sigma}^2_{\text{classical}}(\boldsymbol{x}) + \frac{\hat{\sigma}^2_{\text{optim}}(\boldsymbol{x})}{M}}. \tag{3.7}$$

We recall that our estimates for $\sigma^2_{\text{optim}}(\boldsymbol{x})$ and $\sigma^2_{\text{classical}}(\boldsymbol{x})$ are given by:

$$\hat{\sigma}^2_{\text{optim}}(\boldsymbol{x}) = \frac{1}{M-1}\sum_{i=1}^{M}\left(\hat{f}_i(\boldsymbol{x}) - \frac{1}{M}\sum_{i=1}^{M}\hat{f}_i(\boldsymbol{x})\right)^2,$$

and

$$\hat{\sigma}^2_{\text{classical}}(\boldsymbol{x}) = \frac{1}{M}\sum_{i=1}^{M}\left(\hat{f}_i(\boldsymbol{x}) - \hat{\hat{f}}_i(\boldsymbol{x})\right)^2.$$

Assumption 2 tells us that

$$(f(\boldsymbol{x}) - \hat{f}_*(\boldsymbol{x}))^2 = \left(\sigma^2_{\text{classical}}(\boldsymbol{x}) + \frac{\sigma^2_{\text{optim}}}{M}\right)\epsilon_0^2, \quad \text{with} \quad \epsilon_0 \sim \mathcal{N}(0,1),$$

and

$$\hat{\sigma}^2_{\text{optim}}(\boldsymbol{x}) = \frac{\sigma^2_{\text{optim}}(\boldsymbol{x})}{M-1}\zeta_{\text{o}}, \quad \text{with} \quad \zeta_{\text{o}} \sim \chi^2(M-1).$$

Assumption 3 implies

$$\hat{\sigma}^2_{\text{classical}}(\boldsymbol{x}) = \frac{\sigma^2_{\text{classical}}(\boldsymbol{x})}{M}\zeta_{\text{c}}, \quad \text{with} \quad \zeta_{\text{c}} \sim \chi^2(M).$$

This enables us to rewrite equation (3.7) to

$$T^2 = \frac{\epsilon_0^2}{\gamma W_{\text{o}} + (1-\gamma)W_{\text{c}}}, \tag{3.8}$$

with

$$W_{\mathrm{o}} = \frac{\zeta_{\mathrm{o}}}{M-1}, \quad \text{and,} \quad W_{\mathrm{c}} = \frac{\zeta_{\mathrm{c}}}{M},$$

and

$$\gamma = \frac{\frac{\sigma^2_{\mathrm{optim}}(\boldsymbol{x})}{M}}{\sigma^2_{\mathrm{classical}}(\boldsymbol{x}) + \frac{1}{M}\sigma^2_{\mathrm{optim}}(\boldsymbol{x})},$$

where we dropped the dependence of γ on $\boldsymbol{x}$ to simplify notation. Our goal is to bound the following probability:

$$\mathbb{P}\left(\frac{\epsilon_0^2}{\gamma W_{\mathrm{o}} + (1-\gamma)W_{\mathrm{c}}} > F_{1-\alpha}(1, M-1)\right), \tag{3.9}$$

which we can rewrite as

$$\int\int \left(\mathbb{P}\left(\frac{\epsilon_0^2}{\gamma w_{\mathrm{o}} + (1-\gamma)w_{\mathrm{c}}} > F_{1-\alpha}(1, M-1)\right)\right) dG_{\mathrm{o}}(w_{\mathrm{o}}) dG_{\mathrm{c}}(w_{\mathrm{c}}),$$

where $G_{\mathrm{o}}(w_{\mathrm{o}})$ is the cumulative distribution function of W_{o} and $G_{\mathrm{c}}(w_{\mathrm{c}})$ is the cumulative distribution function of W_{c}. We define the conditional probability in the integral as $\phi(\gamma)$:

$$\phi(\gamma) := \mathbb{P}\left(\frac{\epsilon_0^2}{\gamma w_{\mathrm{o}} + (1-\gamma)w_{\mathrm{c}}} > F_{1-\alpha}(1, M-1)\right). \tag{3.10}$$

The next step is to show that $\phi(\gamma)$ is convex. Let H be the CDF of ϵ_0^2, which has a $\chi^2(1)$ distribution, then $h = H'$ is strictly decreasing, which implies $h' < 0$. We can rewrite $\phi(\gamma)$ as

$$\phi(\gamma) = 1 - H((\gamma w_{\mathrm{o}} + (1-\gamma)w_{\mathrm{c}})F_{1-\alpha}(1, M-1)),$$

which gives

$$\phi'(\gamma) = -(w_{\mathrm{o}} - w_{\mathrm{c}})F_{1-\alpha}(1, M-1)h((\gamma w_{\mathrm{o}} + (1-\gamma)w_{\mathrm{c}})F_{1-\alpha}(1, M-1)),$$

and

$$\begin{aligned}\phi''(\gamma) = &-(w_{\mathrm{o}} - w_{\mathrm{c}})^2 F_{1-\alpha}(1, M-1)^2 \\ &\cdot h'((\gamma w_{\mathrm{o}} + (1-\gamma)w_{\mathrm{c}})F_{1-\alpha}(1, M-1)) > 0.\end{aligned}$$

This means that $\phi(\gamma)$ is convex, which implies that equation (3.9) is convex. The maximum of equation (3.9) is therefore either at $\gamma = 0$ or $\gamma = 1$. Evaluating

equation (3.8) shows that taking $\gamma = 0$ gives T^2 an $F(1, M)$ distribution and taking $\gamma = 1$ gives T^2 an $F(1, M-1)$ distribution. Since $F_\alpha(1, M) < F_\alpha(1, M-1)$ for all α, we get

$$\mathbb{P}\left(\frac{(f(\boldsymbol{x}) - \hat{f}_*(\boldsymbol{x}))^2}{\hat{\sigma}^2_{\text{classical}}(\boldsymbol{x}) + \frac{\hat{\sigma}^2_{\text{optim}}(\boldsymbol{x})}{M}} > F_{1-\alpha}(1, M-1)\right) \leq \alpha.$$

□

3.B Motivation of Assumptions

Our method relies on three assumptions. We will first provide a theoretical motivation of these assumptions and then provide some additional empirical support.

3.B.1 Theoretical Motivation of Assumptions

Assumption 1 is a common modeling assumption that may or may not hold depending on the data. The second assumption claims that the output of the neural network is normally distributed. This normality is a very standard assumption. The typical reasoning is that in a deterministic parametric model without regularization this assumption holds asymptotically. The same asymptotic normality can be shown for the third assumption, which is a common assumption of the parametric model.

We now provide the supoprt of these statements for a parametric model. Here, there is no variance due to training and we need to show - 1 - that the output of the model is normally distributed and - 2 - that if we train the model again on simulated targets, that the output will still be normally distributed with roughly equal variance. We stress that this is not a proof that our assumptions hold, which is impossible to prove for a neural network, but a proof of the result for a parametric model, which motivates the assuptions.

Let $\boldsymbol{\theta}$ be the parameters that parametrize our network. Let $p_{\boldsymbol{\theta}}(\mathcal{D})$ be the likelihood of the data given $\boldsymbol{\theta}$. Our setup corresponds to finding the $\boldsymbol{\theta}$ that maximizes $p_{\boldsymbol{\theta}}(\mathcal{D})$:

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\mathcal{D}),$$

and subsequently finding $\hat{\hat{\boldsymbol{\theta}}}$ that maximises $p_{\boldsymbol{\theta}}(\mathcal{D}_{\text{new}})$:

$$\hat{\hat{\boldsymbol{\theta}}} = \arg\max_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\mathcal{D}_{\text{new}}),$$

Furthermore, we define

$$I(\boldsymbol{\theta}_0) := \text{Cov}_{\boldsymbol{\theta}_0} \frac{\partial}{\partial \boldsymbol{\theta}} \log(p_{\boldsymbol{\theta}}(\boldsymbol{x}_1, y_1))\Big|_{\boldsymbol{\theta}_0}. \tag{3.11}$$

Under certain consistency and regularity conditions it is possible to show that

$$\sqrt{n}(\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}_0) \to \mathcal{N}\left(0, I(\boldsymbol{\theta}_0)^{-1}\right),$$

and

$$\sqrt{n}(\hat{\hat{\boldsymbol{\theta}}} - \hat{\boldsymbol{\theta}}) \to \mathcal{N}\left(0, I(\hat{\boldsymbol{\theta}})^{-1}\right).$$

For a proof and clarification of the assumed consistency and regularity see Van der Vaart (2000) or Seber and Wild (2003).

With $\hat{f}_i$ and $\hat{\hat{f}}_i$, we denote the output of the mean prediction of an ensemble member before and after repeating part of the training. The delta method gives the variance of $\hat{f}_i$ and $\hat{\hat{f}}_i$:

$$\mathbb{V}\left[\hat{f}_{\hat{\boldsymbol{\theta}}}(\boldsymbol{x})\right] = \frac{\partial}{\partial \boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\boldsymbol{x})\Big|_{\boldsymbol{\theta}_0} \mathbb{V}\left[\hat{\boldsymbol{\theta}}\right] \left(\frac{\partial}{\partial \boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\boldsymbol{x})\Big|_{\boldsymbol{\theta}_0}\right)^T,$$

and

$$\mathbb{V}\left[\hat{\hat{f}}_{\hat{\hat{\boldsymbol{\theta}}}}(\boldsymbol{x})\right] = \frac{\partial}{\partial \boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\boldsymbol{x})\Big|_{\hat{\hat{\boldsymbol{\theta}}}} \mathbb{V}\left[\hat{\hat{\boldsymbol{\theta}}}\right] \left(\frac{\partial}{\partial \boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\boldsymbol{x})\Big|_{\hat{\hat{\boldsymbol{\theta}}}}\right)^T.$$

Under the assumed consistency, $\hat{\boldsymbol{\theta}}$ and $\hat{\hat{\boldsymbol{\theta}}}$ will be close and thus $\frac{\partial}{\partial \boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\boldsymbol{x})\big|_{\hat{\boldsymbol{\theta}}}$ will be close to $\frac{\partial}{\partial \boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\boldsymbol{x})\big|_{\hat{\hat{\boldsymbol{\theta}}}}$. By the same consistency, $I(\boldsymbol{\theta}_0)$ and $I(\hat{\boldsymbol{\theta}})$ will be close.

3.B.2 Empirical Assessment of Assumptions

During the Boston Housing part of Experiment 1, we kept track of all the predictions of the first ensemble member before and after retraining in each of the 100 simulations. For most values of $\boldsymbol{x}$, we found that the errors are indeed normally distributed and that the variance of $\hat{f}_i(\boldsymbol{x}) - f(\boldsymbol{x})$ is slightly larger than the variance of $\hat{\hat{f}}_i(\boldsymbol{x}) - \hat{f}_i(\boldsymbol{x})$. This is expected, since the former also has variance due to the randomness of the training. We also see, however, that for some values of $\boldsymbol{x}$, we get a large bias term, violating Assumption 2. Figure 3.7 shows these plots for the first 28 data points in the Boston Housing test set.

3.C Additional Experimentation

In this section, we provide additional experimental results. We repeated parts of Experiment 1 in Section 3.4 with the following alterations:

1. We used differently distributed noise – violating Assumption 1 – in order to see how this affects the confidence intervals.
2. We removed all regularization in the neural networks.
3. We used a different simulation model. Recall that we used a random forest that was trained on real-world data sets to be able to simulate data for our experiments. We replaced the random forest with a neural network as the true function, $f(\boldsymbol{x})$.
4. We used different retraining fractions.

We also give the RMSE values from Experiment 1 and provide the violin plots for all the prediction and confidence intervals.

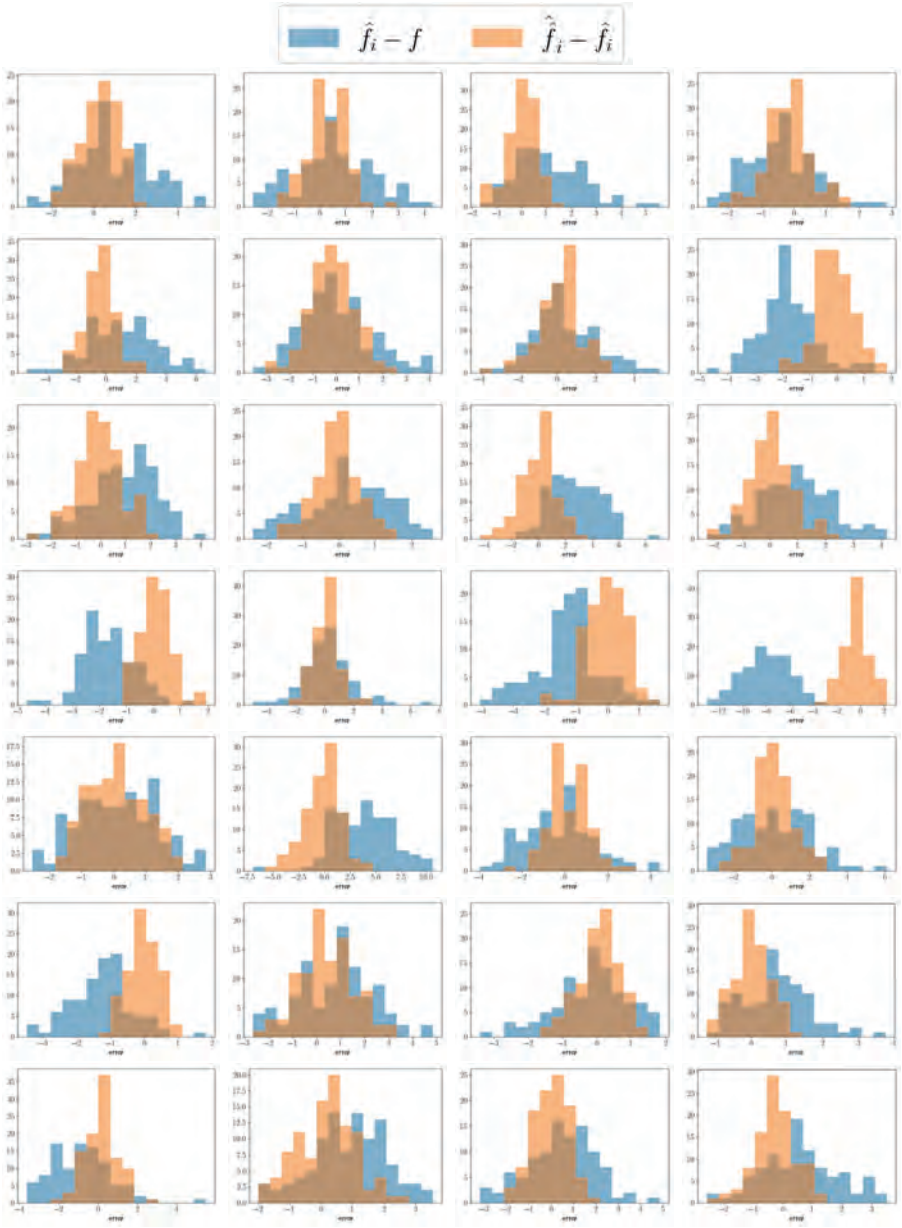


Figure 3.7: The same error plots as in Figure 3.5 for the first 28 data points in the Boston Housing test set. The assumed normality seems to hold well. However, the original predictor is often biased.

3.C.1 Differently Distributed Noise

We study what would happen if we misspecified our model. In order to test this, we simulated data with additive $t(3)$ and $\Gamma(1/10, \sqrt{10})$ distributed noise, denoted with ϵ. In order to make the experiments comparable to the earlier ones, we used the variance $\sigma^2(\boldsymbol{x})$ from the random forest and a scaling factor, C, to obtain a comparable size heteroscedastic variance:

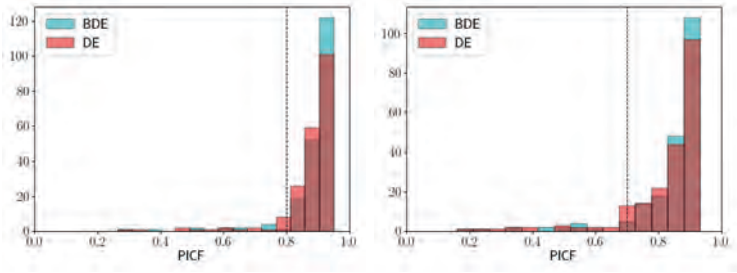
$$y = f(\boldsymbol{x}) + C\sigma(\boldsymbol{x})\epsilon$$

For the $t(3)$ distribution we have $C = \sqrt{1/3}$ and for the $\Gamma(1/10, \sqrt{10})$ distribution we have $C = 1$.

Table 3.3 illustrates that BDE still produce better confidence intervals than DE, but that the performance is affected by the violation of Assumption 1. In the case of the, purposely very skewed, gamma distribution, we also see that the prediction intervals are no longer calibrated, as is illustrated in Figure 3.8.

Table 3.3: Results of the comparison of our method Bootstrapped Deep Ensembles (BDE) to Deep Ensembles (DE) on the Boston simulation using differently distributed additive noise. A total of 100 simulated data sets were used to calculate the metrics. Brier-CIA denotes the Brier score of the CICF of an A% confidence interval.

Noise	**Brier-CI90** ↓		**Brier-CI80** ↓		**Brier-CI70** ↓		**Width CI90**	
	BDE	DE	BDE	DE	BDE	DE	BDE	DE
	$\times 10^{-2}$		$\times 10^{-2}$		$\times 10^{-2}$			
$\mathcal{N}(0,1)$	**6.15**	9.74	**7.88**	11.1	**7.67**	10.5	4.70	4.13
$t(3)$	**6.28**	9.45	**8.07**	10.9	**7.69**	10.1	4.48	3.93
$\Gamma(1/10, \sqrt{10})$	**9.63**	11.8	**11.6**	13.0	**10.9**	11.9	2.66	2.55



(a) $\alpha = 0.2$ (b) $\alpha = 0.3$

Figure 3.8: Histogram of the individual PICF($\boldsymbol{x}$) values calculated on the test set of the Boston simulation. Each point in the histogram represents the fraction of times the true function value $f(\boldsymbol{x})$ was inside the confidence interval calculated over 100 simulations. The skewness of the gamma distribution shows in the error of the PICF.

3.C.2 No Regularization

To examine the effect of regularization, we repeated a part of Experiment 1 without any regularization. The results are given in Table 3.4. We note that the regularization seems to have hardly any effect on the outcome. This could be the result of the fixed training time of 80 epochs and a relatively simple neural network architecture preventing overfitting.

Table 3.4: Results of the comparison of our method Bootstrapped Deep Ensembles (BDE) to Deep Ensembles (DE) without any regularization. A total of 100 simulated data sets were used to calculate the metrics. On each new data set, new ensembles were trained, and new confidence and prediction intervals were constructed. Brier-CI90 denotes the Brier score of the CICF of a 90% confidence interval. Brier-PI90 denotes the equivalent quantity for the PICF. RMSE gives the root mean squared error of the predictions with respect to the targets. Since the predictor is identical for both methods, there is only one value.

SIMULATION	**Brier-CI90** ↓		**Brier-PI90** ↓		**RMSE**	**Width CI90**		**Width PI90**	
	BDE	DE	BDE	DE	BDE/DE	BDE	DE	BDE	DE
	$\times 10^{-2}$		$\times 10^{-3}$						
Boston	**8.0**	11	**8.9**	13	3.82	4.32	3.88	11.2	10.7
Concrete	**4.1**	8.6	**3.9**	6.8	10.3	10.7	8.13	30.6	29.0
Energy	**9.0**	12	**6.2**	8.3	2.87	2.98	2.43	4.98	4.73

3.C.3 Different Simulation Method

Instead of a random forest, we used a neural network in order to simulate data (see Algorithm 8). The network has the same architecture and training procedure as the ones used for the experiment.

We see in Table 3.5 that we get better results for both BDE and DE, although BDE still perform better. A likely explanation is that this task is easier, as is indicated by the significantly lower RMSE. The model we are simulating targets from is identical to the model we are using for the experiment.

3.C.4 Different Retraining Fractions

As expected, we observe from Table 3.6 that the widths of the confidence intervals get larger with an increasing retraining fraction. Trivially, setting the retraining fraction to 0 would yield zero variance and setting it to 1 would also capture the uncertainty due to random training.

Algorithm 8 Pseudo-code to simulate data based on a real-world data set $\mathcal{D}$ using a neural network.

1: Train a neural network on $\mathcal{D}$ that outputs $f(\boldsymbol{x})$ and $\sigma^2(\boldsymbol{x})$;
2: Simulate new targets: $\tilde{y}_i \sim \mathcal{N}\left(f(\boldsymbol{x}_i), \sigma^2(\boldsymbol{x}_i)\right)$;
3: **Return:** $\mathcal{D}_{\text{new}} = ((\boldsymbol{x}_1, \tilde{y}_1), \ldots, (\boldsymbol{x}_n, \tilde{y}_n))$;

Table 3.5: Results of the comparison of our method Bootstrapped Deep Ensembles (BDE) to Deep Ensembles (DE) when using a neural network to simulate data. A total of 100 simulated data sets were used to calculate the metrics. On each new data set, new ensembles were trained, and new confidence and prediction intervals were constructed. Brier-CI90 denotes the Brier score of the CICF of a 90% confidence interval. Brier-PI90 denotes the equivalent quantity for the PICF. RMSE gives the root mean squared error of the predictions with respect to the targets. Since the predictor is identical for both methods.

SIMULATION	**Brier-CI90** ↓		**Brier-PI90** ↓		**RMSE**	**Width CI90**		**Width PI90**	
	BDE	DE	BDE	DE	BDE/DE	BDE	DE	BDE	DE
	$\times 10^{-2}$		$\times 10^{-3}$						
Boston	**2.8**	3.1	**6.3**	8.8	2.92	3.31	3.27	7.86	7.50
Concrete	**2.4**	3.3	**4.9**	6.9	6.00	7.81	7.12	17.7	16.6
Energy	**3.7**	3.8	**8.5**	8.9	2.64	3.10	1.99	5.82	5.53

Table 3.6: The effect of the training fraction on BDE. A total of 100 simulated data sets were used to calculate the metrics. On each new data set, new ensembles were trained, and new confidence and prediction intervals were constructed. Brier-CI80 denotes the Brier score of the CICF of an 80% confidence interval. Brier-PI80 denotes the equivalent quantity for the PICF.

Retraining fraction	**Brier-CI80** ↓ $\times 10^{-2}$	**Brier-PI80** ↓ $\times 10^{-2}$	**Width CI80**	**Width PI80**
0.1	11.5	3.03	2.84	7.37
0.2	9.07	2.70	3.16	7.54
0.3	7.78	2.45	3.38	7.64
0.4	6.93	2.34	3.60	7.78

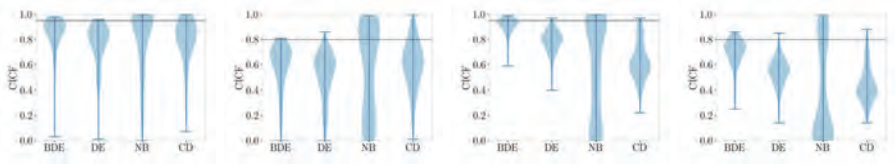
3.C.5 Additional Results Experiment 1

Table 3.7 gives the RMSE values of all methods during Experiment 1 of this chapter. Figures 3.9 and 3.10 give the violin plots of the CICF and PICF values

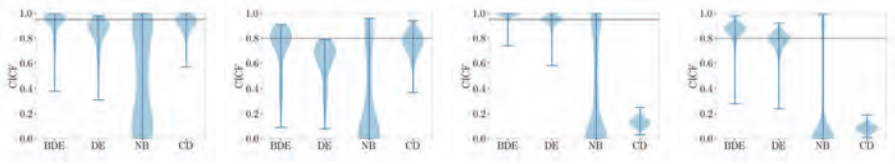
from the same experiment. These figures demonstrate that bootstrapped DE are able to provide reliable confidence and prediction intervals. We also note that high-quality prediction intervals do not guarantee high-quality confidence intervals.

Table 3.7: The RMSE values of the methods during the simulations of Experiment 1. Each value is calculated with respect to the targets of the test set and are averaged over the 100 simulations that were used for each data set. Bootstrapped Deep Ensembles and Deep Ensembles have the same score since they use the exact same predictor. Concrete Dropout has a very comparable score. The Naive Bootstrap has significantly larger errors. This is to be expected since each ensemble member is effectively being trained on less data due to the resampling.

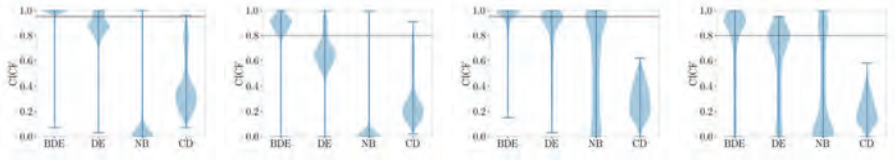
SIMULATION	(Bootstrapped) Deep Ensembles	Naive Bootstrap	Concrete Dropout
Boston	3.92	4.03	3.95
Concrete	10.4	12.5	10.8
Energy	2.73	3.38	2.67
Kin8nm	0.216	0.286	0.238
Naval	0.0128	0.0180	0.0128
Power-Plant	5.01	5.66	5.08
Yacht	3.26	3.86	2.68
Wine	0.654	0.677	0.709



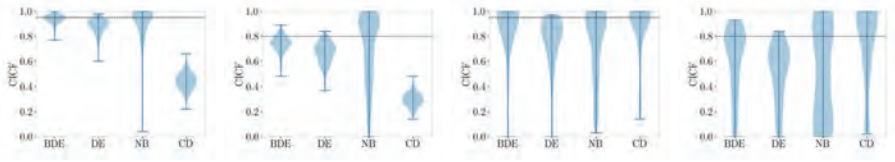
(a) Boston, $\alpha = 0.05$ (b) Boston, $\alpha = 0.2$ (c) Concrete,$\alpha = 0.05$ (d) Concrete, $\alpha = 0.2$



(e) Energy, $\alpha = 0.05$ (f) Energy, $\alpha = 0.2$ (g) kin8nm, $\alpha = 0.05$ (h) kin8nm, $\alpha = 0.2$

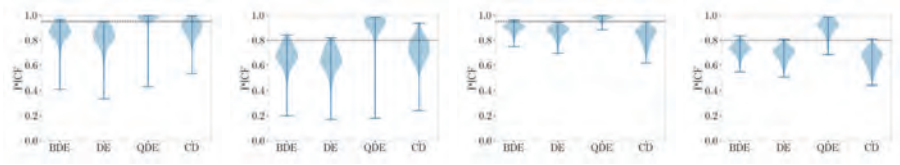


(i) Naval, $\alpha = 0.05$ (j) Naval, $\alpha = 0.2$ (k) Power-plant, $\alpha = 0.05$ (l) Power-plant, $\alpha = 0.2$

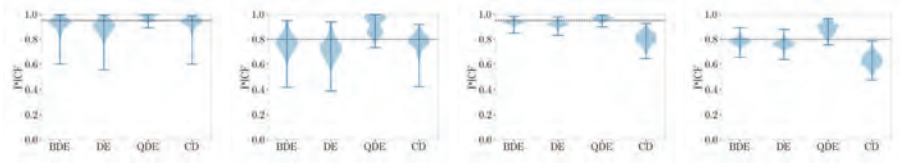


(m) Wine, $\alpha = 0.05$ (n) Wine, $\alpha = 0.2$ (o) Yacht, $\alpha = 0.05$ (p) Yacht, $\alpha = 0.2$

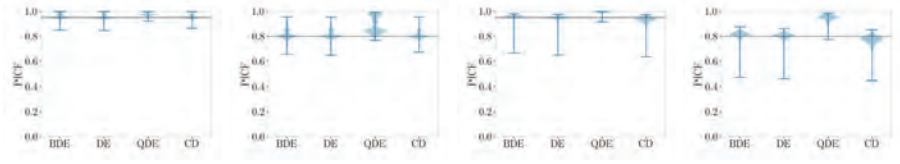
Figure 3.9: Violin plots of the CICF values for all 8 simulations of Experiment 1. For each simulation, we give the CICF values for the 95% and 80% confidence intervals. Each plot is made using the CICF scores of each data point in the test sets. The CICF scores are calculated using 100 simulations. The confidence intervals of Bootstrapped Deep Ensembles have better coverage than the other methods in most simulations.



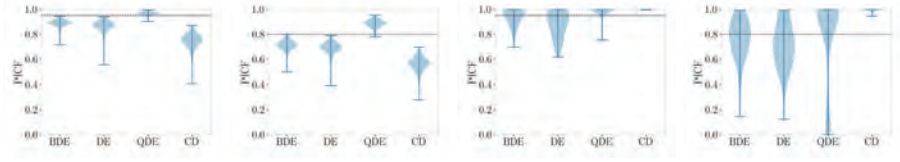
(a) Boston, $\alpha = 0.05$ (b) Boston, $\alpha = 0.2$ (c) Concrete,$\alpha = 0.05$ (d) Concrete,$\alpha = 0.2$



(e) Energy, $\alpha = 0.05$ (f) Energy, $\alpha = 0.2$ (g) kin8nm, $\alpha = 0.05$ (h) kin8nm, $\alpha = 0.2$



(i) Naval, $\alpha = 0.05$ (j) Naval, $\alpha = 0.2$ (k) Power-plant, $\alpha = 0.05$ (l) Power-plant, $\alpha = 0.2$



(m) Wine, $\alpha = 0.05$ (n) Wine, $\alpha = 0.2$ (o) Yacht, $\alpha = 0.05$ (p) Yacht, $\alpha = 0.2$

Figure 3.10: Violin plots of the PICF values for all 8 simulations of Experiment 1. For each simulation, we give the PICF values for the 95% and 80% prediction intervals. The prediction intervals have better coverage than the confidence intervals. Quality-Driven Ensembles give prediction intervals that are too large in most simulations.

3.D Detecting Overfitting

While not the primary goal of our method, a potential added bonus is the possibility to detect overfitting. Suppose that we are in a situation where the networks are overfitting the noise. The ensemble members of standard DE, that are trained on the exact same targets, will tend to provide the same predictions when overfitting on the targets, yielding very small confidence intervals. The retrained ensemble members of Bootstrapped DE, on the other hand, are trained on different targets and hence will tend to provide quite different predictions from their original counterparts, leading to relatively large confidence intervals. With extreme overfitting, to the point that $\hat{\sigma}^2(\boldsymbol{x})$ gets close to zero, this advantage of bootstrapped DE over DE will vanish and both methods will fail to detect overfitting.

We provide short a motivating example by comparing BDE and DE in a scenario in which we know that the network will overfit: a complex network with only 7 data points and no regularization. The targets were simulated from a $\mathcal{N}\left(0, 0.2^2\right)$ distribution, pure noise. Each ensemble member had three hidden layers containing 400, 200, and 100 hidden layers and was trained for 80 epochs.

Figure 3.11 illustrates that BDE is better able to detect overfitting. The confidence intervals of our method increase at the location of the data points, indicating overfitting, whereas those of DE almost vanish. The reason is as follows. The original ensemble members $\hat{f}_i(\boldsymbol{x})$ are the same for both BDE and DE. In the retraining step of BDE, however, the ensemble members will overfit to new targets, resulting in a large estimate for $\sigma^2_{\text{classical}}(\boldsymbol{x})$. This overfitting is even more apparent by the fact that confidence intervals of BDE sometimes actually increase at the locations of the training data. We only investigated this briefly and it may be worthwhile to investigate this further.

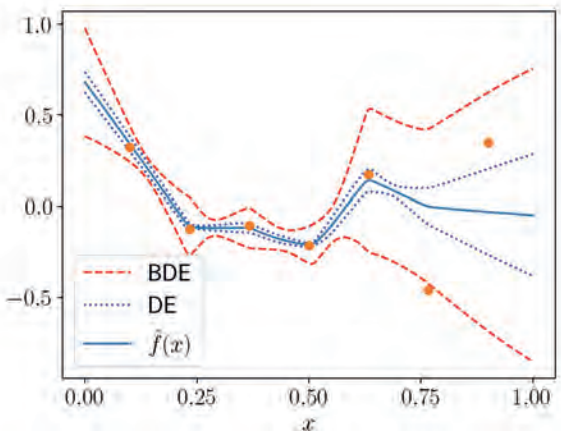


Figure 3.11: The 90% confidence intervals of Bootstrapped Deep Ensembles (BDE) and Deep Ensembles (DE). The original ensemble members were trained long enough to overfit on the data. DE are unable to detect this since all ensemble members behave roughly the same. The variance of the networks after retraining on new targets, however, is much larger since each network will overfit on *different* targets. BDE are therefore better able to detect overfitting.

Chapter 4

Optimal Mean-Variance Estimation

This chapter is based on the paper entitled *"Optimal Training of Mean Variance Estimation Neural Networks"* (Sluijterman et al., 2024b), which has been published in *Neurocomputing*. Where the previous chapter mainly focused on the model uncertainty, this chapter focuses on the data uncertainty, see Figure 4.1. Specifically, we focus on improving the type of networks that were used as the individual ensemble members in the previous chapter.

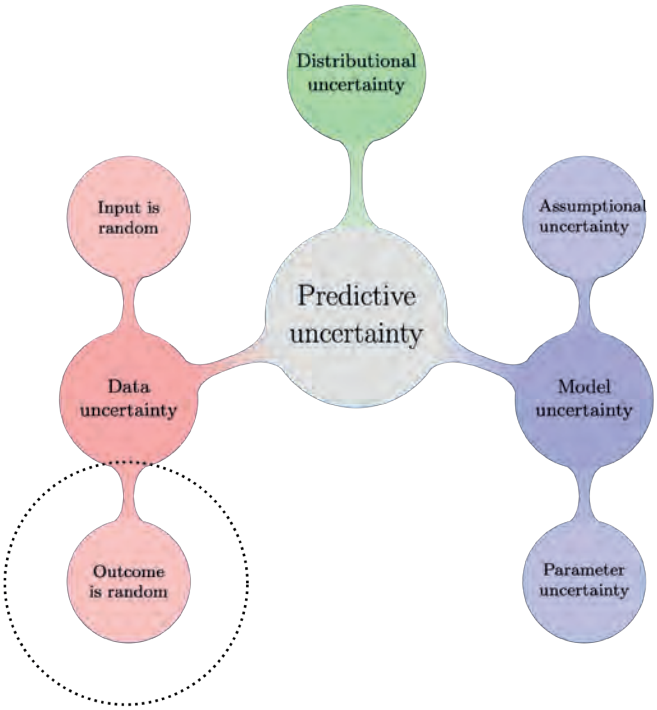


Figure 4.1: The scope of Chapter 4. This chapter focuses on the data uncertainty in a regression setting. We focus on methods to improve the neural networks that were used as ensemble members in the previous chapter.

4.1 Introduction

Neural networks are gaining tremendous popularity both in regression and classification applications. In a regression setting, the scope of this chapter, neural networks are used for a wide range of tasks such as the prediction of wind power (Khosravi and Nahavandi, 2014), bone strength (Shaikhina and Khovanova, 2017), and floods (Chaudhary et al., 2022).

Due to the deployment of neural networks in these safety-critical applications, uncertainty estimation has become increasingly important (Gal, 2016). Assuming a representative data set, the uncertainty in the prediction can be roughly

decomposed into two parts: epistemic or model uncertainty, the reducible uncertainty that captures the fact that we are unsure about our model, and aleatoric uncertainty, the irreducible uncertainty that arises from the inherent randomness of the data (Hüllermeier and Waegeman, 2021; Abdar et al., 2021). In this chaper, we refer to the latter as the variance of the noise, to avoid any confusion or philosophical discussions. The variance of the noise can be *homoscedastic* if it is constant, or *heteroscedastic* if it depends on the input $\boldsymbol{x}$.

There is a vast amount of research that studies the model uncertainty. Notable approaches include Bayesian neural networks (MacKay, 1992a; Neal, 2012), dropout (Gal and Ghahramani, 2016; Gal et al., 2017), and ensembling (Heskes, 1997). Conversely, a lot less emphasis is often placed on the estimation of the variance of the noise. Monte-Carlo dropout, for example, simply uses a single homoscedastic hyperparameter. Some other methods, such as concrete dropout (Gal et al., 2017) and the hugely popular Deep Ensembles (Lakshminarayanan et al., 2017), use a Mean Variance Estimation (MVE) network (Nix and Weigend, 1994). While acknowledging that there are alternative approaches for uncertainty quantification, in this chapter we therefore focus on optimal training of MVE networks.

An MVE network, see Figure 4.2, works as follows. We assume that we have a data set consisting of n pairs $(\boldsymbol{x}_i, y_i)$, with $y_i \sim \mathcal{N}\left(\mu(\boldsymbol{x}_i), \sigma^2(\boldsymbol{x}_i)\right)$. An MVE network consists of two sub-networks that output a prediction for the mean, $\mu_{\boldsymbol{\theta}}(\boldsymbol{x})$, and for the variance, $\sigma^2_{\boldsymbol{\theta}}(\boldsymbol{x})$. These sub-networks only share the input layer and do not have any shared weights or biases. In order to enforce positivity of the variance, a transformation such as a softplus or an exponential is used. The network is trained by using the negative loglikelihood of a normal distribution as the loss function:

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{n} \frac{1}{2} \log(\sigma^2_{\boldsymbol{\theta}}(\boldsymbol{x}_i)) + \frac{1}{2} \frac{(y_i - \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_i))^2}{\sigma^2_{\boldsymbol{\theta}}(\boldsymbol{x}_i)}. \tag{4.1}$$

Since the MVE network is often used as the building block for complex uncertainty estimation methods, it is essential that it works well. Multiple authors have noted that the training of an MVE network can be unstable (Seitzer et al., 2021; Skafte et al., 2019; Takahashi et al., 2018). The main argument, elaborated on in the next section, is that the network will start focusing on areas where the network does well at the start of the training process while ignoring poorly fitted regions.

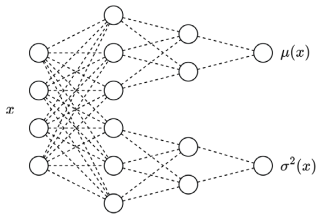


Figure 4.2: Original MVE architecture

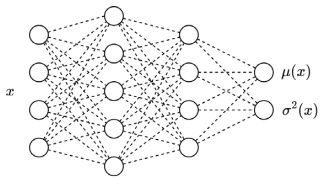


Figure 4.3: Modern MVE architecture

However, Nix and Weigend (1994) already warned for the possibility of harmful overfitting of the variance and gave the solution:

The training of an MVE network should start with a warm-up period where the variance is fixed and only the mean is optimized.

Additionally, the variance is initialized at a constant value in order to make all data points contribute equally to the loss. Nix and Weigend (1994) did not demonstrate the importance of this warm-up period in the original paper. In this chapter, we empirically demonstrate that using a warm-up period can greatly improve the performance of MVE networks and fixes the instability noted by other authors.

A limited amount of research has investigated possible improvements of the MVE network (Seitzer et al., 2021; Skafte et al., 2019). Most improvements require a significant adaptation to the training procedure such as a different loss function or locally aware mini-batches. One of the most prominent approaches is the β-NLL loss (Seitzer et al., 2021), which multiplies the loss in Equation (4.1) by a factor $\sigma_{\boldsymbol{\theta}}^{2\beta}(\boldsymbol{x}_i)$ while disabling gradient back-propagation for this factor. This induces a combination of the original NLL loss ($\beta = 0$) and the mean-squared error loss ($\beta = 1$). However, to the best of our knowledge, none have investigated our proposed easy-to-implement improvement:

The mean and variance in an MVE network should be regularized separately.

Most modern methods (Jain et al., 2020; Egele et al., 2022; Gal et al., 2017; Lakshminarayanan et al., 2017) appear to use the same regularization for both the mean and the variance. In fact, the current use of the MVE network often does not even easily allow for different regularizations. Typically, only a second output node is added to represent the variance, instead of an entire separate sub-network (see Figure 4.3). As we will demonstrate in this chapter, separate regularization can be very beneficial to the predictive results.

Contributions:

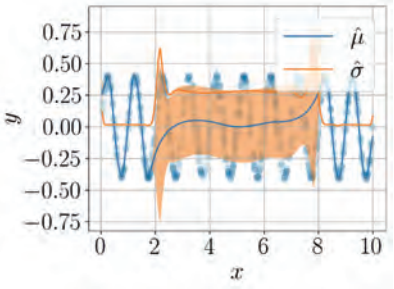
- We provide experimental results that demonstrate the importance of a warm-up period as suggested by Nix and Weigend (1994).
- We explore the benefits of updating the mean and variance simultaneously after the warm-up versus solely learning the variance while keeping the mean fixed.
- We offer a theoretical motivation for why distinct regularization of the mean and variance is essential for an MVE network. We back up our claims with real-world evidence, demonstrating how this approach can lead to significant enhancements.

Organization:

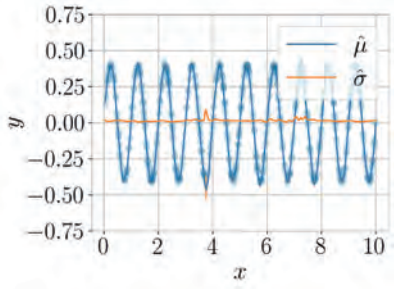
This chapter consists of 6 sections, this introduction being the first. In Section 4.2, we go through the problems with MVE networks that have recently been reported in the literature. In the same section, we show that these problems can be resolved by following the recommendation of using a warm-up period. We also provide an additional theoretical motivation in favor of updating both the mean and the variance after the warm-up as opposed to keeping the mean fixed and only learning the variance. Section 4.3 explains, both intuitively and using classical theory, why we expect to need different amounts of regularization for the mean and the variance estimates. Both the effect of the warm-up and of separate regularization are experimentally examined in Sections 4.4 and 4.5. The final section summarizes the results, gives a list of recommendations when training an MVE network, and provides possible avenues for future work.

4.2 Difficulties With Training MVE Networks

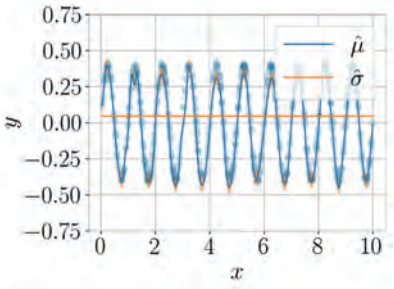
It is known that the training of an MVE network can be unstable (Seitzer et al., 2021; Skafte et al., 2019; Takahashi et al., 2018). The main argument is that the network may fail to learn the mean function for regions where it initially has a large error. In these regions, the variance estimate will increase, which implies that the residual does not contribute to the loss as much. The network will start to focus more on regions where it is performing well, while increasingly ignoring poorly fit regions. For a more in-depth explanation of this effect, we refer to Nix and Weigend (1994).



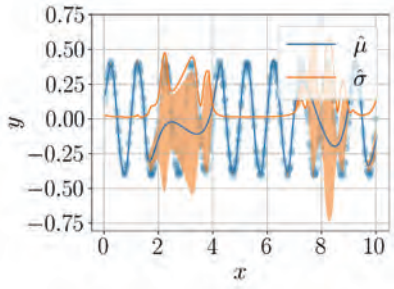
(a) Regular



(b) Warm-up



(c) Separate regularization



(d) β-NLL

Figure 4.4: The effect of various MVE training strategies. The orange area gives plus or minus a single standard deviation. Without a warm-up or separate regularization, the network fails to learn the mean function when simultaneously updating the mean and the variance. The use of a β-NLL loss with a β of 0.5 does not resolve the convergence issues.

To illustrate what can happen, we reproduced an experiment from Seitzer et al. (2021). We sampled 1000 covariates, x_i, uniformly between 0 and 10, and subsequently sampled the targets, y_i, from a $\mathcal{N}\left(0.4\sin(2\pi x_i), 0.01^2\right)$ distribution. Figure 4.4(a) shows that the MVE network is unable to properly learn the mean function. Increasing training time does not solve this. A network with a similar architecture that was trained using the mean-squared error loss *was* able to learn the mean function well.

We provide a second explanation for this behavior by noting that the loss landscape is likely to have many local minima. We already encounter this in a very simple example. Suppose we have a data set consisting of two parts: 100 data

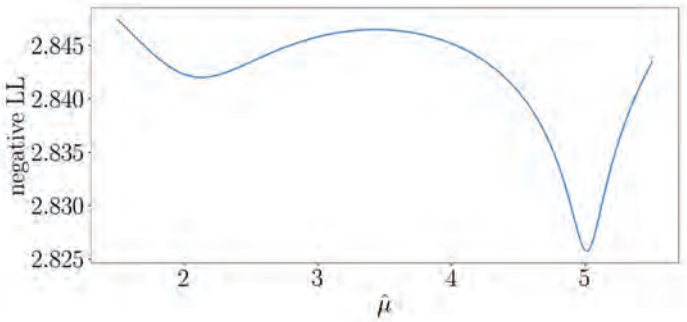


Figure 4.5: A simple example of local minima in the negative loglikelihood. The data consist of two parts: 100 data points from a $\mathcal{N}\left(2, 0.5^2\right)$ distribution and 100 data points from a $\mathcal{N}\left(5, 0.1^2\right)$ distribution. The graph shows the negative loglikelihood as a function of $\hat{\mu}$ where we take the optimal variance estimates for each value of $\hat{\mu}$. The loss has a positive definite 3×3 Hessian at both of the locations, which means they are genuine minima in $\mathbb{R}^3$.

points from a $\mathcal{N}\left(2, 0.5^2\right)$ distribution and 100 data points from a $\mathcal{N}\left(5, 0.1^2\right)$ distribution. For each part, we are allowed to pick a separate variance estimate, $\hat{\sigma}_1^2$ and $\hat{\sigma}_2^2$, but we can only pick a single estimate for the mean. In this situation, there are two local minima of the negative loglikelihood (see Figure 4.5): we can set $\hat{\mu}$ to approximately 2 with a small $\hat{\sigma}_1^2$ and large $\hat{\sigma}_2^2$ or set $\hat{\mu}$ to 5 with a large $\hat{\sigma}_1^2$ and small $\hat{\sigma}_2^2$. The negative loglikelihood has a positive definite 3×3 Hessian at both locations, which are thus minima in $\mathbb{R}^3$. This implies that it is not possible to get out of the local minima illustrated in Figure 4.5 by varying $\hat{\sigma}_1$ and $\hat{\sigma}_2$.

While this simplified setting is of course not a realistic representation of a neural network, it does illustrate that there can easily be many local minima when dealing with complex functions for the mean and the variance. When we start from a random estimate for the mean, it is therefore not unlikely to end up in a bad local minimum.

4.2.1 The Solution: Warm-up

The original authors advised to use a warm-up, which indeed alleviates most problems. After initialization, the variance is fixed at a constant value and the mean estimate is learned. In a second phase, the mean and variance are updated simultaneously.

We can motivate why a warm-up is beneficial, both from the loss-contribution perspective and from the local minima perspective. From the loss-contribution perspective, when keeping the variance fixed during the warm-up, we do not have the problem that regions that are predicted poorly initially fail to learn. Because the variance estimate at initialization is constant, all residuals contribute to the loss equally. From the loss-landscape perspective, we are less likely to end up in a bad local minima if we start from a sensible mean function. Figure 4.4(b) shows that adding a warm-up period indeed solves the convergence problem shown in Figure 4.4(a).

The use of a warm-up period shares similarities with a technique proposed by Kabir et al. (2021) for directly estimating prediction intervals. In this method, the network is initially trained on approximate targets to steer the prediction intervals toward the correct direction. It is then trained on the exact targets in a subsequent phase. This both improved performance and decreased the required training time.

4.2.2 What to Do After the Warm-up?

After the warm-up period, we could either update the variance while keeping the mean estimate fixed or update both simultaneously. In the original MVE paper, the authors argue that simultaneously estimating the mean and the variance is also advantageous for the estimate of the mean. The reasoning is that the model will focus its resources on low noise regions, leading to a more stable estimator.

From a general theoretical perspective, there are clear advantages to optimizing the full likelihood. The resulting maximum-likelihood-estimate is consistent and asymptotically efficient (see any standard textbook on statistics, for instance DeGroot 1986, chapter 7). No other consistent estimator can asymptotically have a lower variance. For a linear model, a similar result even holds for the non-asymptotic regime: Taking the variance of the noise into account leads to an estimator with lower variance. We provide additional details on these statements in Appendix 4.A.

This lower variance in turn results in improved metrics such as RMSE. We demonstrate this both for the general case and for a linear model in Appendices 4.A.1 and 4.A.2. These theoretical results suggest that, besides the obvious benefit of having an estimate of the variance, it is also beneficial for the mean estimate to take the variance into account. Even if we measure performance

on unseen data with the mean-squared error, there are valid arguments to take the variance of the residuals into account when estimating the mean.

In summary, focusing on low noise regions is beneficial. However, the estimate of the variance of the noise strongly depends on the quality of the mean predictor. If the mean predictor is bad, the estimation will not focus on low noise regions but on high accuracy regions, which can be very detrimental. We therefore need a warm-up period, after which classical theory would suggest that estimating the mean and variance simultaneously has advantages. In Section 4.4, we test whether estimating the mean and variance simultaneously is indeed beneficial for the mean estimate.

4.3 The Need for Separate Regularization

In this section, we give a theoretical motivation for the need for a different regularization of the parts of the network that give the mean and variance estimate. The amount of regularization that is needed when estimating a function depends on the smoothness and there is no reason to assume that the mean function and the variance function are equally smooth. If one function is much smoother than the other, we should not regularize them the same way. For instance, in the case of homoscedastic additive noise, the variance function is a constant function, whereas the mean function is likely not.

We can make this argument explicit for a classical linear model. We do this by considering two linear models that most closely resemble the scenario of an MVE network. The first model will estimate the mean while knowing the variance and the second model will estimate the log of the variance[1] while knowing the mean. Both models will in general have a different optimal regularization constant. All derivations for the properties of linear models used in this chapter can be found in Van Wieringen (2015).

We acknowledge that a neural network is much more intricate than a linear model. However, since even for a simple linear model it is essential to have different regularization, the same applies to more complex models like neural networks that have a linear model as a special case. The empirical results that follow later corroborate this.

[1] An MVE network often uses an exponential transformation in the output of the variance neuron to ensure positivity. The network then learns the log of the variance.

4.3.1 Scenario 1: Estimating the Mean With a Known Variance

We consider a linear model with unknown homoscedastic noise. Specifically, we assume that we have a data set consisting of n data points $(\boldsymbol{x}_i, y_i)$, with $\boldsymbol{x}_i \in \mathbb{R}^p$ and $y \in \mathbb{R}$. With X, which we assume to be of full rank, we denote the $n \times p$ design matrix which has the n covariate vectors $\boldsymbol{x}_i$ as rows. With Y, we denote the $n \times 1$ vector containing the observations y_i. We assume that the true relation between the covariates and the observations is linear, so we consider the following model:

$$Y = X\boldsymbol{\beta} + U, \quad U \sim \mathcal{N}\left(0, \sigma^2 I_n\right), \tag{4.2}$$

where $\boldsymbol{\beta}$ is the true linear relation. The goal is to find the $\hat{\boldsymbol{\beta}} \in \mathbb{R}^p$ that minimizes the total squared error plus a regularization term:

$$\sum_{i=1}^{n}(y_i - \boldsymbol{x}_i^T\boldsymbol{\beta})^2 + \lambda\sum_{j=1}^{p}\beta_j^2.$$

Different values of λ result in different estimators $\hat{\boldsymbol{\beta}}(\lambda)$. In 4.B, we show that optimal regularization constant, λ^*, satisfies

$$\lambda^* \propto p(\boldsymbol{\beta}^T\boldsymbol{\beta})^{-1},$$

where we defined optimal as the λ for which

$$\mathrm{MSE}(\hat{\boldsymbol{\beta}}(\lambda)) := \mathbb{E}\left[|\boldsymbol{\beta} - \hat{\boldsymbol{\beta}}(\lambda)|^2\right]$$

is minimal.

4.3.2 Scenario 2: Estimating the Log-Variance With a Known Mean

Next, we examine a linear model that estimates the logarithm of the variance. We again have n datapoints $(\boldsymbol{x}_i, y_i)$ and we assume the log of the variance to be a linear function of the covariates:

$$y_i = \mu_i + \epsilon, \quad \epsilon \sim \mathcal{N}\left(0, e^{\boldsymbol{x}_i^T\tilde{\boldsymbol{\beta}}}\right)$$

We use the same covariates and for the targets we define:

$$z_i := \log((y_i - \mu)^2) - C, \quad \text{with} \quad C = \psi(1/2) + \log(2),$$

where ψ is the digamma function. This somewhat technical choice for C is made such that

$$z_i = \log(\sigma^2(\boldsymbol{x}_i)) + \tilde{\epsilon},$$

where $\tilde{\epsilon}$ has expectation zero and a constant variance. The details can be found in 4.B. In the same appendix we repeat the same procedure, i.e. minimizing the mean-squared error with a regularization term, and demonstrate that the optimal regularization constant, λ^*, satisfies

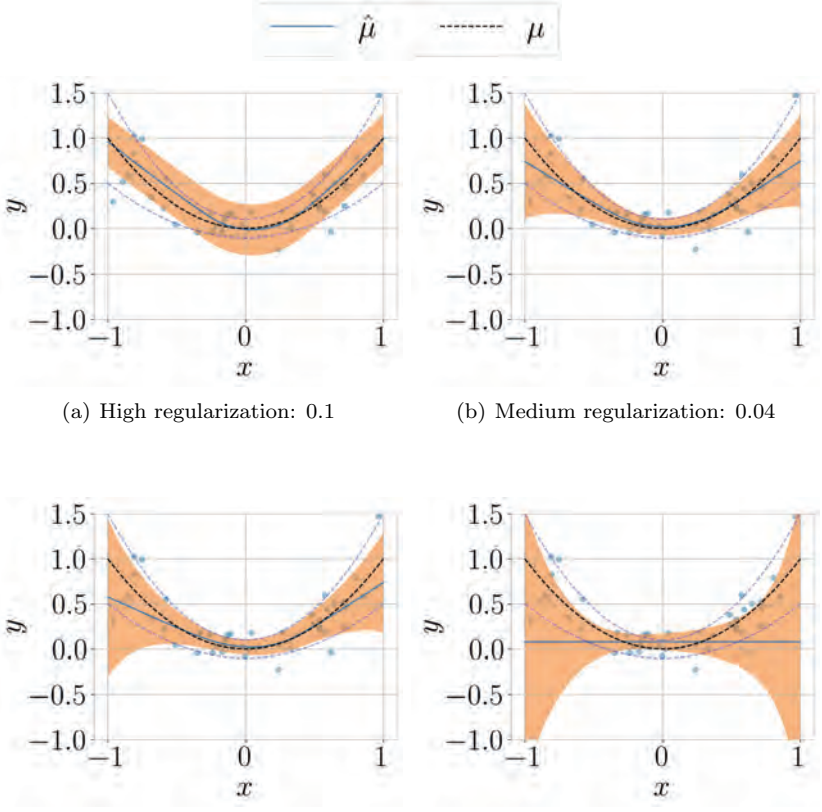
$$\tilde{\lambda}^* \propto p(\tilde{\boldsymbol{\beta}}^T \tilde{\boldsymbol{\beta}})^{-1}.$$

The conclusion is that for these two linear models, that most closely resemble the scenario of regularized neural networks that estimate the mean and log-variance, the optimal regularization constants rely on the true underlying parameters $\boldsymbol{\beta}$ and $\tilde{\boldsymbol{\beta}}$. Since there is no reason to assume that these are similar, there is also no reason to assume that the mean and variance should be similarly regularized.

4.3.3 Separate Regularization of the Variance Alleviates the Variance-Overfitting

While the problem of ignoring initially poorly fit regions is still present, proper regularization of the variance can alleviate the harmful overfitting of the variance. To illustrate this effect, we trained four MVE networks, without a warm-up period, on a simple quadratic function with heteroscedastic noise. The x-values were sampled uniformly from $[-1, 1]$ and the y-values were subsequently sampled from a $\mathcal{N}\left(x^2, (0.1 + 0.4x^2)^2\right)$ distribution. We used the original MVE architecture which has two sub-networks that estimate the mean and the variance. We used separate l_2-regularization constants for both sub-networks in order to be able to separately regularize the mean and the variance. We used the same mean regularization in all networks and gradually decreased the regularization of the variance.

Figure 4.6 demonstrates the effect of different amounts of regularization of the variance. When the variance is regularized too much, the network is unable to learn the heteroscedastic variance. This is problematic both because the resulting uncertainty estimates will be wrong, but also because we lose the



(a) High regularization: 0.1

(b) Medium regularization: 0.04

(c) Low regularization: 0.01

(d) No regularization: 0

Figure 4.6: The effect of the different amounts of regularization of the variance. In all four figures, the mean has the same regularization constant of 0.1. The regularization constants of the variance is given in the subcaptions. The orange area gives plus or minus a single predicted standard deviation. The dotted black line gives the true function and the dotted blue lines illustrate plus or minus the true standard deviation. The blue dots represent the training set. The targets follow a quadratic function with heteroscedastic noise. The covariates are sampled uniformly between -1 and 1. If the variance is regularized too much, the heteroscedasticity is not learned. If the variance is not regularized enough, however, the model fails to learn the mean function.

beneficial effect on the mean that we discussed in the previous subsection. In the second subfigure, the network was able to correctly estimate both the mean and variance. When we decreased the regularization of the variance further, however, we see that the networks starts to increase the variance on the left side instead of learning the function. When we remove the regularization of the variance all together, the network was completely unable to learn the mean function.

Additionally, we repeated the sine experiment while using a higher regularization constant for the variance than for the mean. In Figure 4.4(c), we see that the MVE network is now able to learn the sine function well, even without a warm-up period. We were unable to achieve this when using the same regularization constant for both the mean and the variance or when using the β-NLL loss function with a β of 0.5.

4.4 UCI Regression Experiment

In this section, we experimentally demonstrate the benefit of a warm-up period and separate regularization. In Subsection 4.4.1, we specify the four training strategies that we compare. Subsections 4.4.2 and 4.4.3 give details on the data sets, experimental procedure, and architectures that we use. Finally, the results are given and discussed in Subsection 4.4.4.

4.4.1 Four Approaches

We compare four different approaches:

1. No warm-up: This is the approach that is used in popular methods such as Concrete dropout and Deep Ensembles. The mean and the variance are optimized simultaneously.
2. Warm-up: This is the approach recommended in the original paper. We first optimize the mean and then both the mean and the variance simultaneously.
3. Warm-up fixed mean: We first optimize the mean and then optimize the variance while keeping the mean estimate fixed. We add this procedure to test if optimizing both the mean and the variance after the warm-up further improves the mean estimate.

4. β-NLL: The approach from Seitzer et al. (2021). We use a β of 0.5.

For each approach, we consider two forms of l_2-regularization:

1. Separate regularization: The part of the network that estimates the mean has a different regularization constant than the part of the network that estimates the variance.
2. Equal regularization: Both parts of the network use the same l_2-regularization constant.

4.4.2 Data Sets and Experimental Procedure

We compare the three approaches on a number of regression UCI benchmark data sets. These are the typical regression data sets that are used to evaluate neural network uncertainty estimation methods (Gal and Ghahramani, 2016; Lakshminarayanan et al., 2017; Hernández-Lobato and Adams, 2015; Pearce et al., 2018).

For each data set, we use a 10-fold cross-validation and report the average loglikelihood and RMSE on the validation sets along with the standard errors. For each of the 10 splits, we use another 10-fold cross-validation to obtain the optimal regularization constants. The entire experimental procedure is given in Algorithm 9.

4.4.3 Architecture and Training Details

- We use a split architecture, meaning that the network consists of two sub-networks that output a mean and a variance estimate. Each sub-network has two hidden layers with 40 and 20 hidden units and ELU (Clevert et al., 2015) activation functions. The mean-network has a linear transformation in the output layer and variance-network an exponential transformation to guarantee positivity. We also added a minimum value of 10^{-6} for numerical stability.
- All covariates and targets are standardized before training.
- We use the Adam optimizer (Kingma and Ba, 2014) with gradient clipping set at value 5. We found that this greatly improves training stability in our experiments.

Algorithm 9 Our experimental procedure. A 10-fold cross-validation is used to compare the different methods. In each fold, a second 10-fold cross-validation is used to obtain the optimal regularization constants. We use the same splits when comparing approaches.

Require: Data set $\mathcal{D}$

1: Divide $\mathcal{D}$ in 10 distinct subsets, denoted $D^{(i)}$;
2: **for** i from 1 to 10 **do**
3: $\mathcal{D}_{\text{train}} = \cup_{j \neq i} \mathcal{D}^{(j)}$, $\mathcal{D}_{\text{val}} = \mathcal{D}^{(i)}$;
4: Use 10-fold cross-validation (using only $\mathcal{D}_{\text{train}}$) to find the optimal l_2-regularization constants. This is done by choosing the constants for which the loglikelihood on the left-out sets is highest. The possible regularization constants are $[0.00001, 0.0001, 0.001, 0.01, 0.1]$. Note that for separate regularization, the optimal *combination* of regularization constants is used.;
5: Train a model using the optimal separate regularization constants;
6: Train a model using the optimal equal regularization constant;
7: Evaluate the loglikelihood and RMSE on the validation set;
8: **end for**
9: **return** The average of the 10 loglikelihood and RMSE values along with the standard error;

- We use a default batch-size of 32.
- Within the uncertainty-quantification literature, different approaches are typically compared using a fixed training time (Hernández-Lobato and Adams, 2015). We use 1000 epochs for each training stage. We found that this was sufficient for all networks to converge.
- We set the bias of the variance to 1 at initialization. This makes sure that the variance predictions are more or less constant at initialization.

4.4.4 Results and Discussion

The results are given in Table 4.1 and the optimal regularization constants can be found in 4.C. Bold values indicate that for that specific training strategy (no warm-up, warm-up, or warm-up fixed mean) there is a significant difference between equal and separate regularization. This means that every row can have up to four bold values. Significance was determined by taking the differences

per fold and testing if the mean of these differences is significantly different from zero using a two-tailed t-test at a 90% confidence level.

We see that a warm-up is often very beneficial. For the yacht data set, we observe a considerable improvement in the RMSE when we use a warm-up period. A warm-up also drastically improves the result on the energy data set when we do not allow separate regularization.

Generally, the difference between keeping the mean fixed after the warm-up and optimizing the mean and variance simultaneously after the warm-up is less pronounced. For a few data sets (Concrete, Kin8nm, Protein) we do observe a considerable difference in root-mean-squared error if we only consider equal regularization. If we allow separate regularization, however, these differences disappear.

The use of the β-NLL loss function also appears to yield improvements over the baseline (no warm-up and equal regularization). We also observe some notable improvements when compared to a warm-up with equal regularization (see, for instance, the energy data set). Again, when separate regularization is allowed, these differences mostly disappear.

A separate regularization often drastically outperforms equal regularization. The energy data set gives the clearest example of this. For all three training strategies with the regular loss function, a separate regularization performs much better than an equal regularization of the mean and variance. A similar pattern can be seen for the yacht data set. The optimal regularization for the variance was typically similar or an order of magnitude larger than the optimal regularization of the mean, never lower. We would like to stress that statistically significant results are difficult to obtain with only 5 to 10 folds but that the pattern emerges clearly: separate regularization often improves the results while never leading to a significant decline.

Equal regularization and no warm-up perform as well as the other strategies for some data sets, although never considerably better. For Boston Housing, for example, using a warm-up and separate regularization yields very similar results as the other strategies. This can happen since the problem may be easy enough that the network is able to simultaneously estimate the mean and the variance without getting stuck. Additionally, while there is no reason to assume so a priori, the optimal regularization constant for the mean and the variance can be very similar. In fact, for the Boston Housing experiment, we often found the same optimal regularization constant for the mean and variance during the cross-validation.

Table 4.1: The average loglikelihoods and RMSE-values of the 10 cross-validation splits along with the standard errors. For each split, the optimal regularization constants were obtained with a second 10-fold cross-validation. We used 5-fold cross-validation for the larger Kin8nm and Protein data sets. Bold values indicate a significant difference between equal and separate regularization at a 90% confidence level. Equal regularization never performs significantly better than separate regularization.

Loglikelihood

Data Set	**Training Strategy**							
	No Warm-up		Warm-up		Warm-up fixed Mean		β-NLL	
	Equal regularization	Separate regularization	Equal regularization	Separate regularization	Equal regularization	Separate regularization	Equal regularization	Separate regularization
Boston Housing	-2.61 ± 0.11	-2.61 ± 0.11	-2.59 ± 0.09	-2.59 ± 0.09	-2.67 ± 0.09	-2.60 ± 0.07	-2.67 ± 0.13	-2.58 ± 0.09
Energy	-1.25 ± 0.23	$\mathbf{-0.685 \pm 0.100}$	-1.18 ± 0.47	-0.738 ± 0.117	-1.43 ± 0.44	-0.986 ± 0.178	-0.679 ± 0.116	-0.731 ± 0.120
Yacht	-0.599 ± 0.147	-0.482 ± 0.141	-0.249 ± 0.121	-0.216 ± 0.208	-0.972 ± 0.161	$\mathbf{-0.519 \pm 0.109}$	-0.338 ± 0.126	-0.246 ± 0.160
Concrete	-3.37 ± 0.08	-3.27 ± 0.09	-3.23 ± 0.15	-3.23 ± 0.15	-3.30 ± 0.04	$\mathbf{-3.12 \pm 0.07}$	-3.10 ± 0.03	-3.66 ± 0.43
Wine quality red	-0.994 ± 0.014	$\mathbf{-0.967 \pm 0.007}$	-0.960 ± 0.012	-0.965 ± 0.005	-0.957 ± 0.020	-0.972 ± 0.015	-0.930 ± 0.014	-0.930 ± 0.014
Kin8nm	1.27 ± 0.01	$\mathbf{1.28 \pm 0.01}$	1.25 ± 0.01	1.26 ± 0.01	1.14 ± 0.01	$\mathbf{1.26 \pm 0.01}$	1.29 ± 0.01	1.29 ± 0.01
Protein	-2.83 ± 0.01	-2.84 ± 0.01	-2.82 ± 0.01	$\mathbf{-2.80 \pm 0.01}$	-2.85 ± 0.02	-2.83 ± 0.01	-2.76 ± 0.02	-2.77 ± 0.02

RMSE

Data Set	**Training Strategy**							
	No Warm-up		Warm-up		Warm-up fixed Mean		β-NLL	
	Equal regularization	Separate regularization	Equal regularization	Separate regularization	Equal regularization	Separate regularization	Equal regularization	Separate regularization
Boston Housing	3.56 ± 0.40	3.56 ± 0.40	3.84 ± 0.50	3.84 ± 0.50	4.39 ± 0.53	3.75 ± 0.40	3.74 ± 0.48	3.77 ± 0.49
Energy	2.20 ± 0.15	$\mathbf{0.468 \pm 0.020}$	0.850 ± 0.188	$\mathbf{0.507 \pm 0.023}$	0.813 ± 0.123	0.604 ± 0.034	0.465 ± 0.027	0.480 ± 0.029
Yacht	8.87 ± 0.78	$\mathbf{3.31 \pm 0.70}$	1.00 ± 0.18	0.917 ± 0.203	1.12 ± 0.13	$\mathbf{0.705 \pm 0.084}$	0.515 ± 0.064	0.607 ± 0.105
Concrete	6.41 ± 0.36	6.57 ± 0.25	5.93 ± 0.25	5.83 ± 0.18	7.20 ± 0.27	$\mathbf{5.54 \pm 0.22}$	5.58 ± 0.20	$\mathbf{4.99 \pm 0.20}$
Wine quality red	0.653 ± 0.0092	0.650 ± 0.009	0.641 ± 0.006	0.647 ± 0.009	0.643 ± 0.010	0.638 ± 0.008	0.632 ± 0.012	0.633 ± 0.011
Kin8nm	0.0717 ± 0.0011	0.0706 ± 0.0015	0.0771 ± 0.0010	0.0757 ± 0.0015	0.0834 ± 0.0013	$\mathbf{0.0701 \pm 0.0009}$	0.0697 ± 0.0012	0.0697 ± 0.0012
Protein	4.55 ± 0.02	4.54 ± 0.01	4.47 ± 0.01	$\mathbf{4.42 \pm 0.02}$	4.59 ± 0.06	4.49 ± 0.03	4.24 ± 0.06	4.21 ± 0.02

4.5 UTKFace Age Regression Experiment

In this section, the effects of warm-up and separate regularization are analyzed when using a larger convolutional neural network (CNN). The specific task under consideration is age estimation based on photographic images.

4.5.1 Data Set

We used the aligned and cropped version of the UTKFace data set (Zhang et al., 2017b). The images were downsized from 244×244 to 64×64. The ages are technically discrete, but the range is very large and we therefore treat it is a continuous variable. To achieve a more balanced data set, all individuals over the age of 80, as well as two-thirds of the individuals aged 1, were excluded. The data set was divided into training, validation, and test sets, comprising 3500, 1500, and 1000 samples, respectively. The validation set is used to determine the optimal regularization constants and the test set is used to obtain the final results.

4.5.2 Architecture and Training Details

The CNN is visualized in Figure 4.7. The backbone consists of four pairs of a convolutional layer followed by a max-pool layer. All convolutional layers have 3×3 filters with a stride of one, l_2-regularization with a constant of one, batch-normalization, and ELU activation functions. The first two convolutional layers have four filters and the final two have eight. Each max-pool layer has a 2×2 filter with a stride of 2. The network's second segment consists of two sub-networks that output the mean and the variance predictions, similar to the split architecture used in the previous experiments. Both sub-networks are densely connected networks consisting of four hidden layers with 20, 10, 5, and 2 hidden units respectively. To ensure positivity, an exponential transformation is used for the variance with an added value of 10^{-6} for numerical stability. The two sub-networks have separate l_2-regularization constants.

All targets are standardized prior to training. We use the Adam optimizer with a learning rate of $3 \cdot 10^{-5}$, gradient clipping set at a value of 5, a default batch-size of 32, and 150 epochs. The weights are saved at the end of each epoch and restored to the ones with the highest likelihood on the validation

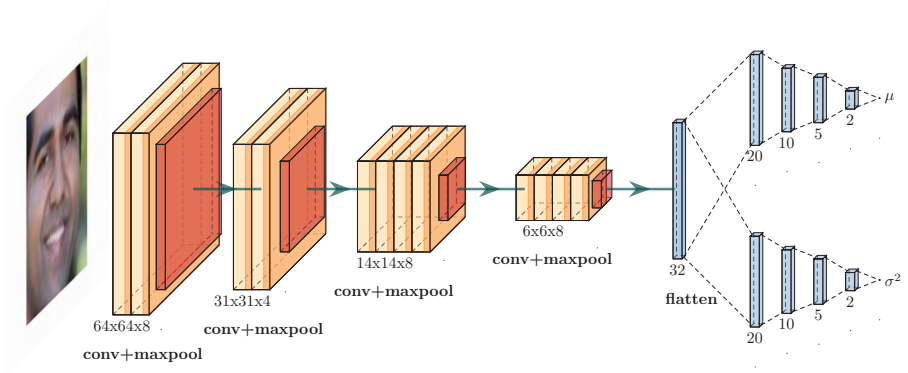


Figure 4.7: The architecture of the CNN used in the UTKFace age regression experiment. The network consists of two segments. We first have 4 pairs of a convolutional layer followed by a max-pool layer. The second segment consist of two densely connected sub-networks that provide the mean and variance predictions.

set. All the results in the following subsections are calculated using the unseen test set.

4.5.3 Separate Regularization

Separate regularization of the mean and variance is more difficult when using a joint backbone. The two sub-networks connected to the backbone can be easily given a different amount of regularization. However, the backbone largely determines what types of functions can be achieved by the network.

We evaluated the effect of different l_2-regularization constants for the two sub-networks (the blue part of Figure 4.7). We started from the same initialization, without a warm-up, and restored the weights to the weights with the highest loglikelihood on the validation set. Figure 4.8 gives the results on the unseen test set.

We observe that for the loglikelihood, the regularization of the mean has a substantial effect. The regularization of the variance is less critical but it appears to be optimal to use a lower regularization constant for the variance. For the RMSE, we also see that a high regularization for the mean is beneficial but we do not observe any clear pattern for the variance.

Although, due to the backbone, it is not possible to fully separate the regu-

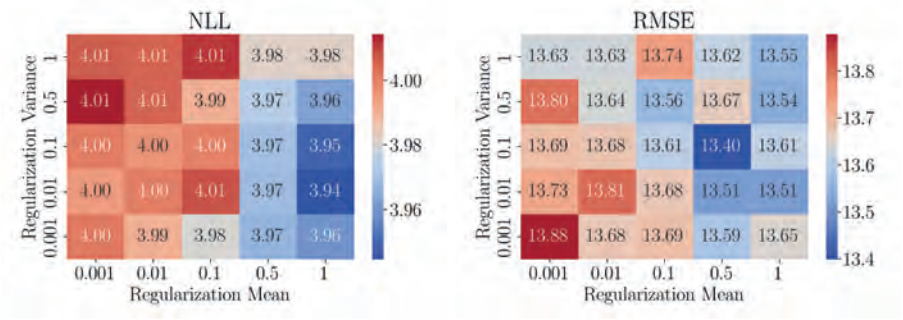


Figure 4.8: The negative loglikelihood and RMSE on the unseen test set for various combinations of l_2-regularization constants. The CNN was trained for 150 epochs without a warm-up and the weights were restored to the epoch with the highest loglikelihood on the validation set.

larization of the mean and variance, regularizing the sub-networks separately still affects the final result. In this example, modest improvements in both likelihood and RMSE could be achieved by using a separate regularization.

4.5.4 A Warm-up When Using a Backbone

An extra step is needed to keep the predicted variance constant during the warm-up. Merely freezing the weights of the variance sub-network is insufficient since the network can still change these predictions through the backbone. The solution is to set the weights of the final layer of the variance sub-network to zero. This ensures that the variance remains constant. Additionally, we initialize the bias of the output layer to zero, which yields unit variance everywhere when using an exponential transformation to ensure positivity.

To demonstrate the effect of a warm-up, we trained two networks from the same initialization, one with and one without a warm-up. We saved the network at the end of each epoch to see the difference between the two strategies and in particular to visualize the transition point from warm-up to full training.

Figure 4.9 gives an illustration of the evolution of the predictions on one of the points from the test set. We observe that the variance remains constant during the warm-up. At the transition, at epoch 50, we observe no instabilities. Other data points have similar smooth transitions.

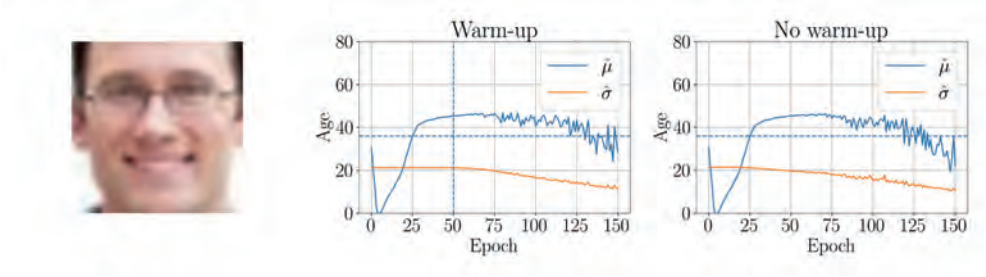


Figure 4.9: The effect of a warm-up on the training process. Both networks had the same initialization and random seed for easier comparison. The vertical dotted line indicates the transition from warm-up to full training and the horizontal dotted lines gives the true age. We see that by initializing the final weights of the variance network to zero, it is possible to keep the predicted variance completely stable during the warm-up. Additionally, we observe no instabilities at the transition point. We evaluated this plot for 100 different inputs and the transition was always smooth.

For this specific example, we did not observe a significant difference in results when using warm-up. This is in line with the UCI experiment, where we also did not always observe an improvement. However, we also see no downside and would therefore still advise to implement it to prevent possible convergence issues.

Additionally, we observe in Figure 4.9 that the manner in which we initialized the network results in an automatic version of a warm-up period. Unit variance is on average the optimal value at initialization since the targets are standardized. On top of that, since the final weights are zero at initialization, changing the backbone does not immediately affect the predicted variance, resulting in a relatively stable variance at the start of training.

4.6 Conclusion

In this chapter, we tested various training strategies for MVE networks. Specifically, we investigated whether following the recommendations of the original authors solves the recently reported convergence problems, and we proposed a novel improvement, separate regularization.

We conclude that the use of a warm-up period is often essential to achieving optimal results and fixes the convergence problems. Without this period, the network can fail to learn regions on which it performs poorly at the start of the training. We empirically observed that not using a warm-up period can lead to highly suboptimal results, both in terms of RMSE and loglikelihood.

We did not find evidence that clearly favors one of the strategies *after* the warm-up, keeping the mean fixed or optimizing the mean and variance simultaneously. In theory, joint maximum likelihood estimation of the mean and the variance is advantageous. In practice, we did not observe significant differences. Optimizing the mean and variance simultaneously after the warm-up was seemingly only beneficial when separate regularization was not allowed.

Current practice - see for instance the implementation of Concrete Dropout (Gal et al., 2017) - is to enforce the same regularization constants for estimating the mean and variance, which implicitly assumes that both functions exhibit a similar degree of smoothness. However, our experiments indicate that this assumption is often violated. Real-world data sets often require a complex mean function that differs significantly from a constant function, while the variance function varies more smoothly, if at all. Therefore, it is more realistic to apply separate regularization constants for the mean and variance functions. Our UCI experiment demonstrate that this approach can indeed lead to significant improvements in model performance across various data sets.

The UTKFace experiments demonstrate that a warm-up is also easy to implement when using a shared backbone. We observed a perfectly constant variance during the warm-up and no instabilities at the transition between the warm-up and full training. Separate regularization is harder to achieve since both the mean and variance sub-networks use the same backbone. Nevertheless, we did observe some moderate improvements when the two sub-networks were given separate regularization constants.

4.6.1 Recommendations

Based on our experiments, we make the following recommendations when training an MVE network:

- Use a warm-up when possible. It is important to initialize the variance such that it is more or less constant for all inputs. Otherwise, some regions may be neglected. This is easily achieved by setting the bias of the variance neuron to 1 at initialization, or by setting the bias of the variance

output and weights that connect to it to zero. For a joint-backbone, this second approach must be used. For some reinforcement learning settings or online learning settings, a warm-up may not be possible.

- Use gradient clipping. We found gradient clipping to yield more stable optimization when optimizing the mean and variance simultaneously.
- Use separate regularization for the mean and variance when possible. If a hyperparameter search is computationally infeasible, the variance should typically be regularized an order of magnitude stronger than the mean. When a backbone is used, the effect of separate regularization is less pronounced.
- Use the β-NLL loss, especially when not using a warm-up or separate regularization. We observed improvements in both RMSE and NLL when compared to the baseline of not using a warm-up and separate regularization. However, we also observed that the convergence issues are not always resolved and therefore advice to not use the β-NLL loss alone, but rather in combination with a warm-up and separate regularization if possible.

4.6.2 Future Work

The results on separate regularization indicate that variance and mean functions are generally not equally smooth. It is therefore likely not optimal to use a similar architecture and training procedure for both. It would be interesting to investigate whether the use of a separate architecture and training procedure leads to further improvements.

APPENDIX CHAPTER 4

4.A Details on the Advantage of Taking the Variance Into Account

4.A.1 General Case

As we stated in the main text, there are advantages to optimizing the full likelihood. The resulting maximum-likelihood-estimate is consistent and asymptotically efficient (DeGroot, 1986, chapter 7). Specifically, no other consistent estimator can asymptotically have a lower variance.

This lower variance results in improved metrics such as RMSE. To see this, we analyze the expected squared error of a new observation: $\mathbb{E}\left[(y_{\text{new}} - \mu_{\hat{\boldsymbol{\theta}}}(\boldsymbol{x}_{\text{new}}))^2\right]$. Here, we assume that there exists a true $\boldsymbol{\theta}_0$ and that $\hat{\boldsymbol{\theta}}$ is the maximum-likelihood-estimate. The expectation is taken over y_{new} and the data with which $\hat{\boldsymbol{\theta}}$ is created. The expected squared error is given by

$$\begin{aligned}
&\mathbb{E}\left[(y_{\text{new}} - \mu_{\hat{\boldsymbol{\theta}}}(\boldsymbol{x}_{\text{new}}))^2\right] \\
&= \mathbb{E}\left[(y_{\text{new}} - \mu_{\boldsymbol{\theta}_0}(\boldsymbol{x}_{\text{new}}) + \mu_{\boldsymbol{\theta}_0}(\boldsymbol{x}_{\text{new}}) - \mu_{\hat{\boldsymbol{\theta}}}(\boldsymbol{x}_{\text{new}}))^2\right] \\
&= \mathbb{E}\left[(y_{\text{new}} - \mu_{\boldsymbol{\theta}_0}(\boldsymbol{x}_{\text{new}}))^2\right] + \mathbb{E}\left[(\mu_{\boldsymbol{\theta}_0}(\boldsymbol{x}_{\text{new}}) - \mu_{\hat{\boldsymbol{\theta}}}(\boldsymbol{x}_{\text{new}}))^2\right] \\
&\approx \mathbb{E}\left[(y_{\text{new}} - \mu_{\boldsymbol{\theta}_0}(\boldsymbol{x}_{\text{new}}))^2\right] + \mathbb{E}\left[(D_{\boldsymbol{\theta}}\mu_{\boldsymbol{\theta}_0}(\boldsymbol{x}_{\text{new}})(\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}_0))^2\right] \\
&= C + \mathbb{E}\left[D_{\boldsymbol{\theta}}\mu_{\boldsymbol{\theta}_0}(\boldsymbol{x}_{\text{new}})(\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}_0)\right]^2 + \mathbb{V}\left[D_{\boldsymbol{\theta}}\mu_{\boldsymbol{\theta}_0}(\boldsymbol{x}_{\text{new}})(\hat{\boldsymbol{\theta}})\right]. \qquad (4.3)
\end{aligned}$$

We now compare this to a different consistent estimator for $\boldsymbol{\theta}_0$, denoted by $\tilde{\boldsymbol{\theta}}$. By following the same derivation, we obtain an expected squared error of

$$C + \mathbb{E}\left[D_{\boldsymbol{\theta}}\mu_{\boldsymbol{\theta}_0}(\boldsymbol{x}_{\text{new}})(\tilde{\boldsymbol{\theta}} - \boldsymbol{\theta}_0)\right]^2 + \mathbb{V}\left[D_{\boldsymbol{\theta}}\mu_{\boldsymbol{\theta}_0}(\boldsymbol{x}_{\text{new}})(\tilde{\boldsymbol{\theta}})\right]. \qquad (4.4)$$

We can show that this is larger than (or equal to) the final line in Equation (4.3). The first term, C, is equal for both (4.3) and (4.4). Since $\hat{\boldsymbol{\theta}}$ was obtained by using maximum likelihood optimization, we know two things. Firstly, we know that the second term in (4.3) is asymptotically of lower order than the third term, and secondly, we know that

$$\mathbb{V}\left[D_{\boldsymbol{\theta}}\mu_{\boldsymbol{\theta}_0}(\boldsymbol{x}_{\text{new}})(\tilde{\boldsymbol{\theta}})\right] \geq \mathbb{V}\left[D_{\boldsymbol{\theta}}\mu_{\boldsymbol{\theta}_0}(\boldsymbol{x}_{\text{new}})(\hat{\boldsymbol{\theta}})\right],$$

which shows that our estimate $\tilde{\boldsymbol{\theta}}$ asymptotically will have a larger expected quadratic error.

4.A.2 Linear Model

In the non-asymptotic regime, a similar result holds for a linear model. Taking the variance into account, leads to a lower-variance estimator. All derivations for the statements in this chapter regarding linear models can be found in Van Wieringen (2015).

We assume that we have a data set consisting of n data points $(\boldsymbol{x}_i, y_i)$, with $\boldsymbol{x}_i \in \mathbb{R}^p$ and $y \in \mathbb{R}$. With X, we denote the $n \times p$ design matrix which has the n covariate vectors $\boldsymbol{x}_i$ as rows. With Y, we denote the $n \times 1$ vector containing the observations y_i. We assume X to be of full rank and consider the linear model:

$$Y = X\boldsymbol{\beta} + U, \quad U \sim \mathcal{N}(0, \Sigma), \tag{4.5}$$

where Σ can be any invertible covariance matrix, possibly heteroscedastic and including interaction terms. Suppose this covariance matrix is known, then classical theory tells us that it is beneficial for our estimate of $\boldsymbol{\beta}$ to take this into account.

To see this, we will compare the linear model in Equation (4.5) with a rescaled version that takes the covariance matrix into account. Since Σ is positive semi-definite, we can write it as BB^T and rescale our model by multiplying with B^{-1}:

$$Z := B^{-1}Y = B^{-1}X\boldsymbol{\beta} + B^{-1}U \tag{4.6}$$

$$= \tilde{X}\boldsymbol{\beta} + V, \quad V \sim \mathcal{N}(0, I_n) \tag{4.7}$$

Both formulations lead to different estimators of $\boldsymbol{\beta}$. The unscaled formulation leads to

$$\hat{\boldsymbol{\beta}} = (X^T X)^{-1} X^T Y,$$

and the second formulation leads to

$$\hat{\boldsymbol{\beta}}^* = (\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T B^{-1} Y.$$

Both estimators are linear unbiased estimators of $\boldsymbol{\beta}$. However, the Gauss-Markov theorem (Gauss, 1823) (see Dodge (2008) for a version that is not in Latin) tells us that the variance of $\hat{\boldsymbol{\beta}}^*$ is lower than the variance of $\hat{\boldsymbol{\beta}}$.

Gauss-Markov Theorem *(Gauss, 1823)*
In the notation introduced above, consider the linear model $Y = X\boldsymbol{\beta}+U$. *Under the following assumptions:*

1. $\mathbb{E}[U] = \mathbf{0}$,
2. $\mathbb{V}[U] = cI_n$,
3. X *is of full rank,*

the ordinary least squares (OLS) estimator for $\boldsymbol{\beta}$, $\hat{\boldsymbol{\beta}} = (X^TX)^{-1}X^TY$, *has the lowest variance of all unbiased linear estimators of* $\boldsymbol{\beta}$, *i.e., the difference of the covariance matrix of any unbiased linear estimator and the covariance matrix of the OLS estimator is positive semi-definite.*

We note that in the second formulation all the conditions of the theorem are met. We therefore know that $\hat{\boldsymbol{\beta}}^*$ has the lowest variance of all unbiased linear estimators of $\boldsymbol{\beta}$ and thus in particular we know that it has a lower variance than $\hat{\boldsymbol{\beta}}$.

We want to emphasize that this leads to improved metrics such as RMSE. Let us for instance look at difference between the expected squared errors of a new pair $(\boldsymbol{x}_{\text{new}}, y_{\text{new}})$ when using $\hat{\boldsymbol{\beta}}$ and $\hat{\boldsymbol{\beta}}^*$:

$$\begin{aligned}
&\mathbb{E}\left[(y_{\text{new}} - \boldsymbol{x}_{\text{new}}^T\hat{\boldsymbol{\beta}})^2 - (y_{\text{new}} - \boldsymbol{x}_{\text{new}}^T\hat{\boldsymbol{\beta}}^*)^2\right] \\
&\qquad = \mathbb{E}\left[y_{\text{new}}^2 - 2y_{\text{new}}\boldsymbol{x}_{\text{new}}^T\hat{\boldsymbol{\beta}} + \boldsymbol{x}_{\text{new}}^T\hat{\boldsymbol{\beta}}\hat{\boldsymbol{\beta}}^T\boldsymbol{x}_{\text{new}}\right] \\
&\qquad\qquad\qquad - \mathbb{E}\left[y_{\text{new}}^2 - 2y_{\text{new}}\boldsymbol{x}_{\text{new}}^T\hat{\boldsymbol{\beta}}^* + \boldsymbol{x}_{\text{new}}^T\hat{\boldsymbol{\beta}}^*\hat{\boldsymbol{\beta}}^{*T}\boldsymbol{x}_{\text{new}}\right] \\
&\qquad = \mathbb{E}\left[\boldsymbol{x}_{\text{new}}^T\hat{\boldsymbol{\beta}}\hat{\boldsymbol{\beta}}^T\boldsymbol{x}_{\text{new}} - \boldsymbol{x}_{\text{new}}^T\hat{\boldsymbol{\beta}}^*\hat{\boldsymbol{\beta}}^{*T}\boldsymbol{x}_{\text{new}}\right] \\
&\qquad = \boldsymbol{x}_{\text{new}}^T\left(\Sigma_{\hat{\boldsymbol{\beta}}} - \Sigma_{\hat{\boldsymbol{\beta}}^*}\right)\boldsymbol{x}_{\text{new}} \geq 0.
\end{aligned}$$

We used that $\hat{\boldsymbol{\beta}}$ and $\hat{\boldsymbol{\beta}}^*$ are both unbiased and independent of y_{new}. In the final line, we applied the Gauss-Markov theorem that guarantees that $\Sigma_{\hat{\boldsymbol{\beta}}} - \Sigma_{\hat{\boldsymbol{\beta}}^*}$ is a positive semi-definite matrix.

4.B Details on the Different Optimal Regularization Constants for Linear Models

We consider two linear models that most closely resemble the scenario of an MVE network. The first model will estimate the mean while knowing the variance and the second model will estimate the log of the variance while knowing the mean. An MVE network often uses an exponential transformation in the output of the variance neuron to ensure positivity. The network then learns the log of the variance. We show that both models will generally have a different optimal regularization constant.

4.B.1 Scenario 1: Estimating the Mean With a Known Variance

We use the same notation as in the previous example and assume a homoscedastic noise with variance σ^2. If we do not consider regularization, the goal is to find the estimator that minimizes the sum of squared errors,

$$\sum_{i=1}^{n}(y_i - \boldsymbol{x}_i^T\boldsymbol{\beta})^2.$$

The solution is given by

$$\hat{\boldsymbol{\beta}} = (X^TX)^{-1}X^TY,$$

for which we know that

$$\mathbb{E}\left[\hat{\boldsymbol{\beta}}\right] = \boldsymbol{\beta} \quad \text{and} \quad \mathbb{V}\left[\hat{\boldsymbol{\beta}}\right] = \Sigma_{\hat{\boldsymbol{\beta}}} = \sigma^2(X^TX)^{-1}.$$

In particular, we used that $\mathbb{E}[\epsilon] = 0$ and $\mathbb{V}[\epsilon] = \sigma^2$.

When we add a regularization constant, λ, the objective becomes to minimize

$$\sum_{i=1}^{N}(y_i - \boldsymbol{x}_i^T\boldsymbol{\beta})^2 + \lambda\sum_{j=1}^{p}\boldsymbol{\beta}_j^2.$$

The solution to this problem is given by

$$\hat{\boldsymbol{\beta}}(\lambda) = (X^TX + \lambda I_p)^{-1}X^TY.$$

For simplicity, we assume that we have an orthonormal basis, in which case $X^T X = I$. This does not change the essence of the argument and makes the upcoming comparison clearer. Our new estimate is no longer unbiased but has a lower variance:

$$\mathbb{E}\left[\hat{\boldsymbol{\beta}}(\lambda)\right] = \frac{1}{1+\lambda}\boldsymbol{\beta} \quad \text{en} \quad \mathbb{V}\left[\hat{\boldsymbol{\beta}}(\lambda)\right] = \sigma^2(1+\lambda)^{-2} I_p$$

Our goal is to answer the question what the optimal value for λ is. We define optimal as the λ for which

$$\text{MSE}(\hat{\boldsymbol{\beta}}(\lambda)) := \mathbb{E}\left[|\boldsymbol{\beta} - \hat{\boldsymbol{\beta}}(\lambda)|^2\right]$$

is minimal.

Theobald (1974) showed that there exists $\lambda > 0$ such that $\text{MSE}(\hat{\boldsymbol{\beta}}(\lambda)) < \text{MSE}(\hat{\boldsymbol{\beta}})$. Typically, the exact value of λ is unknown, but in our controlled example, the optimal value can be derived analytically (Van Wieringen, 2015):

$$\lambda^* = p\sigma^2(\boldsymbol{\beta}^T\boldsymbol{\beta})^{-1}.$$

4.B.2 Scenario 2: Estimating the Log Variance With a Known Mean

Next, we examine a linear model that estimates the logarithm of the variance. We again have n datapoints $(\boldsymbol{x}_i, y_i)$ where we assume the log of the variance to be a linear function of the covariates:

$$y_i = \mu_i + \epsilon, \quad \epsilon \sim \mathcal{N}\left(0, e^{\boldsymbol{x}_i^T \tilde{\boldsymbol{\beta}}}\right)$$

We use the same covariates and for the targets we define:

$$z_i := \log((y_i - \mu)^2) - C, \quad \text{with} \quad C = \psi(1/2) + \log(2),$$

where ψ is the digamma function. This somewhat technical choice for C is made such that

$$z_i = \log(\sigma^2(\boldsymbol{x}_i)) + \tilde{\epsilon},$$

where $\tilde{\epsilon}$ has expectation zero and a constant variance, as can be seen from the following derivation:

$$\begin{aligned}\log((y_i - \mu)^2) &= \log\left(\sigma^2(\boldsymbol{x}_i)\frac{(\boldsymbol{x}_i - \mu)^2}{\sigma^2(\boldsymbol{x}_i)}\right) \\ &= \log(\sigma^2(\boldsymbol{x}_i)) + \log\left(\frac{(\boldsymbol{x}_i - \mu)^2}{\sigma^2(\boldsymbol{x}_i)}\right) \\ &= \log(\sigma^2(\boldsymbol{x}_i)) + \log(\zeta)\,,\ \zeta \sim \chi^2(1) \\ &= \log(\sigma^2(\boldsymbol{x}_i)) + \tilde{\epsilon^*}.\end{aligned}$$

The random variable $\tilde{\epsilon^*}$ has an expectation C and a constant variance that does not depend on μ or $\sigma^2(\boldsymbol{x})$ (Pav, 2015). The key result of this specific construction of z is that we have recovered a linear model with additive noise that has zero mean and a constant variance.

This allows us to repeat the procedure from the previous subsection, i.e. minimizing the sum of squared errors with a regularization term. We obtain the following optimal regularization constant

$$\tilde{\lambda}^* = p\mathbb{V}\left[\tilde{\epsilon}\right](\tilde{\boldsymbol{\beta}}^T\tilde{\boldsymbol{\beta}})^{-1}.$$

The conclusion is that for these two linear models, that most closely resemble the scenario of regularized neural networks that estimate the mean and log-variance, the optimal regularization constants rely on the true underlying parameters $\boldsymbol{\beta}$ and $\tilde{\boldsymbol{\beta}}$. Since, in general, these are different, a different regularization constant should be used.

4.C Optimal Regularization Constants

Table 4.2: The average regularization constant of the 10 cross-validation splits. For each split, the optimal regularization constants were obtained with a second 10-fold cross-validation. We used 5-fold cross-validation for the larger Kin8nm and Protein data sets.

	Loglikelihood							
Data Set	**Training Strategy**							
	No Warm-up		Warm-up		Warm-up fixed Mean		β-NLL	
	Equal regularization	Separate regularization (μ, σ^2)	Equal regularization	Separate regularization (μ, σ^2)	Equal regularization	Separate regularization (μ, σ^2)	Equal regularization	Separate regularization (μ, σ^2)
bostonHousing	0.10	(0.10, 0.10)	0.10	(0.10, 0.10)	0.28	(8.2e-3, 0.23)	0.064	(0.064, 0.055)
energy	9.1e-3	(3.5e-3, 0.46)	4.6e-4	(1.0e-4, 0.22)	3.7e-4	(1.0e-4, 0.14)	3.4e-4	(3.3e-4, 0.01)
yacht	0.014	(3.7e-3, 0.10)	9.0e-3	(6.1e-3, 0.010)	1.8e-3	(1.9e-4, 0.010)	6.31e-4	(5.3e-3, 7.4e-4)
concrete	0.082	(0.082, 0.028)	0.010	(0.010, 0.028)	0.010	(1.0e-3, 0.064)	8.2e-3	(1.8e-3, 0.026)
wine-quality-red	0.10	(0.064, 0.23)	0.064	(0.046, 0.24)	0.010	(0.010, 0.42)	0.010	(0.010, 0.019)
kin8nm	2.8e-3	(1.0e-3, 6.4e-3)	1.0e-3	(1.0e-3, 6.4e-3)	1.0e-3	(1.0e-3, 0.010)	1.0e-3	(1.0e-3, 1.0e-3)
protein-tertiary-structure	4.4e-5	(2.4e-5, 8.2e-6)	6.4e-6	(4.6e-6, 4.4e-5)	2.8e-5	(6.4e-6, 4.4e-5)	2.4e-4	(8.2e-5, 4.2e-4)

Chapter 5

Likelihood-Ratio-Based Confidence Intervals

This chapter is based on the preprint entitled *"Likelihood-ratio-based confidence intervals for neural networks"* (Sluijterman et al., 2023), which is currently under submission. This chapter introduces a novel way to quantify the parameter uncertainty by leveraging the likelihood ratio. The resulting confidence intervals have very desirable qualitative qualities and also incorporate parts of the distributional uncertainty. For out-of-distribution data points and in regions where the data is sparse, the confidence intervals expand significantly.

In this chapter, we will also encounter the connection between assumptional uncertainty and parameter uncertainty that was touched upon in Chapter 1. Choosing a more flexible model increases the hypothesis class, making it more likely to include the true function, at the cost of a higher parameter uncertainty.

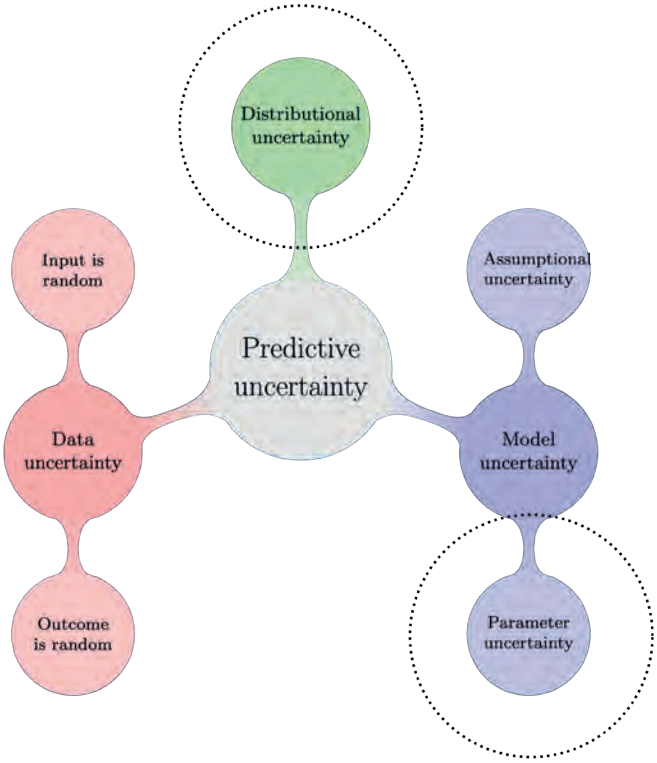


Figure 5.1: The scope of Chapter 5. This chapter focuses on estimating the parameter uncertainty by using a likelihood-ratio-based approach. The resulting confidence intervals have very desirable properties and also incorporate parts of the distributional uncertainty.

5.1 Introduction

Over the past two decades, neural networks have seen an enormous rise in popularity and are currently being used in almost every area of science and industry. In light of this widespread usage, it has become increasingly clear that trustworthy uncertainty estimates are essential (Gal, 2016).

Many uncertainty estimation methods have been developed using Bayesian techniques (Neal et al., 2011; MacKay, 1992a; Gal, 2016), ensembling techniques (Heskes, 1997; Lakshminarayanan et al., 2017), or applications of frequentist techniques such as the delta method (Kallus and McInerney, 2022).

Many of the resulting confidence intervals (for the frequentist methods) and credible regions (for the Bayesian methods) have two common issues. Firstly, most methods result in symmetric intervals around the prediction which can be overly restrictive and can lead to very low coverage in biased regions (Sluijterman et al., 2022). Secondly, most methods rely heavily on asymptotic theorems (such as the central limit theorem or the Bernstein-von-Mises theorem) and can therefore only be trusted in the asymptotic regime where we have many more data points than model parameters, the exact opposite scenario of where we typically find ourselves within machine learning.

Contribution: In this chapter, we demonstrate how the likelihood-ratio test can be leveraged to combat the two previously mentioned issues. We provide a first implementation of a likelihood-ratio-based approach, called DeepLR, that has the ability to produce asymmetric intervals that are more appropriately justified in the scenarios where we have more parameters than data points. Furthermore, these intervals exhibit desirable behavior in regions far removed from the data, as evidenced in Figure 5.5.

Organization: This chapter is structured into four sections, with this introduction being the first. Section 5.2 explains our method in detail and also contains the related work section which is simultaneously used to highlight the advantages and disadvantages of our method. Section 5.3 presents experimental results that illustrate the desirable properties of a likelihood-ratio-based approach. Finally, Section 5.4 summarizes and discusses the results and outlines possible directions for future work.

5.2 DeepLR: Deep Likelihood-Ratio-Based Confidence Intervals

In this section, we present our method, named DeepLR, for constructing confidence intervals for neural networks using the likelihood-ratio test. We first formalize the problem that we are considering in Subsection 5.2.1. Subsection 5.2.2 explains the general idea behind constructing a confidence interval via the likelihood-ratio test. Subsequently, in Subsection 5.2.3, we outline the high level idea for translating this general procedure to neural networks. The details regarding the distribution and the calculation of the test statistic are provided in Subsections 5.2.4 and 5.2.5. Finally, in Subsection 5.2.6, we compare our method to related work while highlighting its strengths and limitations.

5.2.1 Problem Formulation

We consider a data set $\mathcal{D} = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$, consisting of n independent observations of the random variable pair (X, Y). Contrary to previous chapters, we use capital letters to emphasize that the we are dealing with random variables. We consider networks that provide an estimate for the conditional density of $Y \mid X$. This is achieved by assuming a distribution and having the network output the parameter(s) of that distribution.

Three well-known types of networks that fall in this class are: (1) A regression setting where the network outputs a mean estimate and is trained using a mean-squared error loss. This is equivalent to assuming a normal distribution with homoscedastic noise. (2) Alternatively, the network could output both a mean and a variance estimate and be optimized by minimizing the negative loglikelihood assuming a normal distribution. (3) In a classification setting, the network could output logits that are transformed to class probabilities while assuming a categorical distribution.

The network is parametrized by $\boldsymbol{\theta} \in \mathbb{R}^p$, where p is typically much larger than n. With Θ, we denote the set containing all the $\boldsymbol{\theta}$ that are reachable for a network with a specific training process. This includes choices such as training time, batch size, optimizer, and regularization techniques. With $p_{\boldsymbol{\theta}}$, we denote the predicted conditional density. Additionally, we assume that the true conditional density is given by $p_{\boldsymbol{\theta}_0}$ for some $\boldsymbol{\theta}_0$ in Θ. In other words, we assume that our model is well specified.

The objective of our method is to construct a confidence interval for one of the output nodes of the network for a specific input of interest X_0. We denote this output of interest with $f_{\boldsymbol{\theta}_0}(X_0)$ for the remainder of the chapter. In the context of a regression setting, this output of interest is the true regression function value at X_0 and in a classification setting it is the true class probability for input X_0.

We define a $(1 - \alpha) \cdot 100\%$ confidence interval for $f_{\boldsymbol{\theta}_0}(X_0)$ as an interval, $\mathrm{CI}(f_{\boldsymbol{\theta}_0}(X_0))$, which is random since it depends on the random realization of the data, such that the probability (taken with respect to the random data set) that $\mathrm{CI}(f_{\boldsymbol{\theta}_0}(X_0))$ contains the true value $f_{\boldsymbol{\theta}_0}(X_0)$ is $(1 - \alpha) \cdot 100\%$.

5.2.2 Confidence Intervals Based on the Likelihood Ratio

We explain the general idea behind constructing a confidence interval with the likelihood-ratio by working through a well-known example. We consider n observations Y_i that are assumed to be normally distributed with unknown mean μ and unknown variance σ^2. Our goal is to create a confidence interval for μ by using the likelihood-ratio test.

The duality between a confidence interval and hypothesis testing states that we can create a $(1-\alpha)\cdot 100\%$ confidence interval for μ by including all the values c for which the hypothesis $\mu = c$ cannot be rejected at a $(1-\alpha)\cdot 100\%$ confidence level. We must therefore test for what values c we can accept the hypothesis $\mu = c$.

The general approach to test a hypothesis is to create a test statistic of which we know the distribution under the null hypothesis and to reject this hypothesis if the probability of finding the observed test statistic or an extremer value is smaller than α.

As our test statistic, we take two times the log of the likelihood ratio:

$$T(c) := 2\left(\sup_{\Theta}\left(\sum_{i=1}^{n}\log(L(Y_i;\boldsymbol{\theta}))\right) - \sup_{\Theta_0}\left(\sum_{i=1}^{n}\log(L(Y_i;\boldsymbol{\theta}))\right)\right),$$

where L denotes the likelihood function $\boldsymbol{\theta} \mapsto L(Y_i;\boldsymbol{\theta})$, Θ is the full parameter space and Θ_0 the restricted parameter space. In our example, we have

$$\Theta = \{(\mu, \sigma^2) \mid \mu \in \mathbb{R}, \sigma^2 \in \mathbb{R}_{>0}\},$$

and

$$\Theta_0 = \{(c, \sigma^2) \mid \sigma^2 \in \mathbb{R}_{>0}\}.$$

Wilks (1938) proved that $T(c)$ weakly converges to a $\chi^2(1)$ distribution under the null hypothesis that $\mu = c$. We therefore reject if $T(c) > \chi^2_{1-\alpha}(1)$ and our confidence interval for μ becomes the set $\{c \mid T(c) \leq \chi^2_{1-\alpha}(1)\}$, where $\chi^2_{1-\alpha}(1)$ is the $(1-\alpha)$-quantile of a $\chi^2(1)$ distribution.

In our example, this results in the well-known confidence interval for the mean:

$$\bar{Y} \pm z_{1-\alpha/2}\sqrt{\frac{1}{n}\frac{1}{n-1}\sum_{i=1}^{n}(Y_i - \bar{Y})^2},$$

where $z_{1-\alpha/2}$ is the $(1-\alpha/2)$-quantile of a standard-normal distribution.

5.2.3 High-Level Idea of DeepLR

Our goal is to apply the likelihood-ratio testing procedure outlined in the previous subsection to construct a confidence interval for $f_{\boldsymbol{\theta}_0}(X_0)$, the value of one of the output nodes given input X_0. We create this confidence interval by including all the values c for which the hypothesis $f_{\boldsymbol{\theta}_0}(X_0) = c$ cannot be rejected. The testing of the hypothesis is done with the likelihood-ratio test. Specifically, we use two times the log likelihood ratio as our test statistic:

$$\begin{aligned} T(c) &:= 2\Bigg(\sup_{\Theta} \Big(\sum_{i=1}^{n} \log(L(X_i, Y_i; \boldsymbol{\theta})) \Big) - \sup_{\Theta_0(c)} \Big(\sum_{i=1}^{n} \log(L(X_i, Y_i; \boldsymbol{\theta})) \Big) \Bigg) \\ &= 2\Bigg(\sup_{\Theta} \Big(\sum_{i=1}^{n} \log(p_{\boldsymbol{\theta}}(Y_i \mid X_i)) \Big) - \sup_{\Theta_0(c)} \Big(\sum_{i=1}^{n} \log(p_{\boldsymbol{\theta}}(Y_i \mid X_i)) \Big) \Bigg), \end{aligned} \tag{5.1}$$

and we construct a confidence interval for $f_{\boldsymbol{\theta}_0}(X_0)$ by including all values c for which the test statistic is not larger than $\chi^2_{1-\alpha}(1)$:

$$\mathrm{CI}(f_{\boldsymbol{\theta}_0}(X_0)) = \{c \mid T(c) \leq \chi^2_{1-\alpha}(1)\}. \tag{5.2}$$

Here, we consider $\Theta \subset \mathbb{R}^p$ to be the set containing all reachable parameters, and $\Theta_0(c) = \{\boldsymbol{\theta} \in \Theta \mid f_{\boldsymbol{\theta}_0}(X_0) = c\}$. The set Θ is explicitly not equal to all parameter combinations. Due to explicit (e.g., early stopping) and implicit regularization (e.g., lazy training (Chizat et al., 2019)) not all parameter combinations can be reached. The set Θ should therefore be seen as the set containing the parameters of all neural networks that can be found given the optimizer, training time, regularization techniques, and network architecture.

Intuitively, this approach answers the question: *What values could the network have reached at location X_0 while still explaining the data well?* This is a sensible question for a highly flexible and typically overparameterized machine-learning approach. After training, the model ends up with a certain prediction at location X_0. However, since the model is typically very complex, it is likely that the model could just as well have made other predictions at that location while still explaining the data well. Therefore, all those other function values should also be considered as possibilities. Inherently, all modeling choices are taken into account by asking this question. A more flexible model, for instance, is likely able to reach more values without affecting the likelihood of the training data, leading to a larger confidence interval.

The construction of the confidence interval in Equation (5.2) assumes that the test statistic, $T(c)$, has a $\chi^2(1)$ distribution. We discuss this assumption in the

following subsection. The subsection thereafter describes how to calculate the test statistic.

5.2.4 Distribution of the Test Statistic

In the classical setting, Wilks (1938) proved that the likelihood-ratio test statistic asymptotically has a $\chi^2(1)$ distribution when the submodel has one degree of freedom less than the full model. We are, however, not in this classical regime. We have many more parameters than data points and therefore need a similar result for this setting.

It has been shown that the likelihood-ratio test statistic converges to a χ^2 distribution for a wide range of settings, which is referred to as the Wilks-phenomenon by later authors (Fan et al., 2001; Boucheron and Massart, 2011). For a semi-parametric model, which more closely resembles our situation, it has been shown that the test statistic also converges in distribution to a χ^2 distribution under appropriate regularity conditions (Murphy and van der Vaart, 1997).

We prove a similar result for our setting in Appendix 5.A. The theorem states that, under suitable assumptions, our test statistic has a $\chi^2(1)$ distribution. Intuitively, this results from the fact that we added a single constraint, namely that $f_{\boldsymbol{\theta}}(X_0) = c$. We emphasize that even if the test statistic does not exactly follow a $\chi^2(1)$ distribution, the qualitative characteristics of the confidence intervals will remain evident, albeit with inaccurate coverage levels.

5.2.5 Calculating the Test Statistic

Calculating the two terms in equation (5.1) presents certain challenges. The first term, $\sup_\Theta \left(\sum_{i=1}^n \log(p_{\boldsymbol{\theta}}(Y_i \mid X_i)) \right)$, is relatively straightforward. We train a network that maximizes the likelihood, which gives the conditional densities $p_{\hat{\boldsymbol{\theta}}}(Y_i \mid X_i)$.

The second term is substantially more complex. Ideally, we would optimize over the set $\Theta_0(c) = \{\boldsymbol{\theta} \in \Theta \mid f_{\boldsymbol{\theta}}(X_0) = c\}$. This is problematic for two reasons. Firstly, it is unclear how we can add this constraint to the network and secondly, this would necessitate training our network for every distinct value c that we wish to test. Even when employing an efficient bisection algorithm, this could easily result in needing to train upwards of 10 additional networks.

We address this problem as follows. We first create a network that is perturbed in the direction of a relatively large value ($c_{\max}$) at X_0 and a network that is perturbed in the direction a relatively small value ($c_{\min}$) at X_0 while maximizing the likelihood of the data. We denote the resulting network parameters with $\hat{\boldsymbol{\theta}}_{\pm}$:

$$\hat{\boldsymbol{\theta}}_{+} = \underset{\substack{\theta\in\Theta,\\ f_\theta(X_0)\approx c_{\max}}}{\arg\max}\ L(\mathcal{D};\boldsymbol{\theta}), \quad \text{and} \quad \hat{\boldsymbol{\theta}}_{-} = \underset{\substack{\theta\in\Theta,\\ f_\theta(X_0)\approx c_{\min}}}{\arg\max}\ L(\mathcal{D};\boldsymbol{\theta}).$$

Subsequently, the network that maximizes the likelihood under the constraint $f_{\boldsymbol{\theta}}(X_0) = c$ is approximated using a linear combination. Specifically, suppose we want the network that maximizes the likelihood of the training data while passing through c at X_0. In the case that $c > f_{\hat{\boldsymbol{\theta}}}(X_0)$, we approximate this network by taking a linear combination of the outputs such that

$$(1-\lambda) f_{\hat{\boldsymbol{\theta}}}(X_0) + \lambda f_{\hat{\boldsymbol{\theta}}_+}(X_0) = c,$$

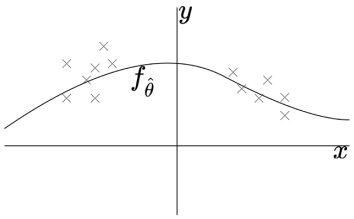
and we define p_c as the density that we get by using the same linear combinations for the distributional parameters that are predicted by the networks parametrized by $\hat{\boldsymbol{\theta}}$ and $\hat{\boldsymbol{\theta}}_+$. We then approximate the second term in equation (5.1) as follows:

$$\sup_{\Theta_0(c)} \Big(\sum_{i=1}^{n} \log(p_{\boldsymbol{\theta}}(Y_i \mid X_i))\Big) \approx \sum_{i=1}^{n} \log(p_c(Y_i \mid X_i)). \tag{5.3}$$

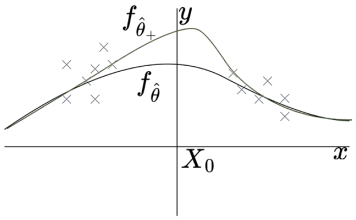
This procedure is visualized in Figure 5.2 for a regression setting. Details on the second step, finding the perturbed networks, are provided below both for a regression and binary-classification setting.

Regression Suppose we completed step 1 and we have a network that maximizes the data's likelihood. For step 2, our objective is to adjust this network such that it goes through a relatively large value at X_0 while continuing to maximize the data's likelihood. Moreover, we aim to achieve this in as stable a manner as possible, given that minor differences can significantly influence the test statistic, particularly for large data sets.

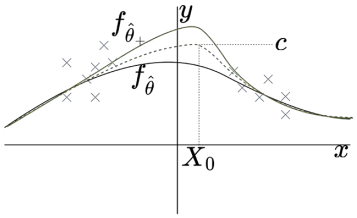
We accomplish this by copying the original network and training it on the objective to maintain the original predictions – as those maximized the likelihood – while predicting $f_{\hat{\boldsymbol{\theta}}}(X_0)+1$ for input X_0. For the perturbation in the negative direction we use -1. These values ± 1 are chosen assuming that the targets are normalized prior to training.



(a) Step 1: Train a network on the data.



(b) Step 2: Copy the resulting network and train it on the objective to have the same predictions at the training locations and a larger prediction at X_0.



(c) Step 3: Choose λ such that $(1-\lambda)f_{\hat{\boldsymbol{\theta}}}(X_0) + \lambda f_{\hat{\boldsymbol{\theta}}_+}(X_0) = c$ and let p_c be the density resulting from the same linear combination of the predicted distributional parameters.

$$2\Bigg(\sum_{i=1}^{n}\log(p_{\hat{\boldsymbol{\theta}}}(Y_i|X_i)) - \sum_{i=1}^{n}\log(p_c(Y_i \mid X_i))\Bigg) \leq \chi^2_{1-\alpha}(1).$$

(d) Step 4: Repeat step 3 for multiple values of c and define the confidence interval for $\text{CI}(f_{\boldsymbol{\theta}_0}(X_0))$ as all c for which:

Figure 5.2: Illustration of the steps of our method for the positive direction in a regression setting. Steps 2, 3, and 4 are also carried out in the negative direction.

We obtain this training objective by using modified training set, $\tilde{\mathcal{D}}$, that is constructed by replacing the targets Y_i in the original training set with the predictions $f_{\hat{\boldsymbol{\theta}}}(X_i)$ of the original network and adding the data point $(X_0, f_{\hat{\boldsymbol{\theta}}}(X_0)+1)$.

The resulting problem is very imbalanced. We want the network to change the prediction at location X_0, which is only present in the data once. This makes the training very unstable since, especially when training for a small number of epochs, it is very influential which specific batch contains the new point. To remedy this, we use a combination of upsampling and downweighting of the new data point $(X_0, f_{\hat{\boldsymbol{\theta}}}(X_0)+1)$. Merely upsampling the new data point – i.e., adding it multiple times – is undesirable as this can introduce significant biases (Van den Goorbergh et al., 2022). Hence, we add many copies of $(X_0, f_{\hat{\boldsymbol{\theta}}}(X_0)+1)$ but we reweigh the loss contributions of these added data points such that

they have a total contribution to the loss that is equivalent to that of a single data point.

We propose to add $2n$/batch size extra data points such that each batch is expected to have 2 new data points. The same training procedure is used as for the original network. We found slightly larger or smaller number of added data points to perform very similarly. This setting worked for a wide variety of data sets and architectures. In Appendix 5.B, we experimentally check the distribution of the test statistic in a controlled simulation experiment.

Binary Classification Consider a data set where the targets are either 1 or 0. Our network outputs logits, denoted with $f_{\hat{\boldsymbol{\theta}}}(X)$, that are transformed to probabilities via a sigmoid function. The procedure is nearly identical for this binary classification setting: We create a positively perturbed network, parametrized by $\hat{\boldsymbol{\theta}}_+$, and a negatively perturbed network, parametrized by $\hat{\boldsymbol{\theta}}_-$.

The only difference is in the construction of the augmented data sets. We again replace the targets Y_i by the predictions of the original network, $f_{\hat{\boldsymbol{\theta}}}(X_i)$, but now we add multiple copies of the data point $(X_0, 1)$ for the positive direction, and multiple copies of the data point $(X_0, 0)$ for the negative direction.

The entire method is summarized in Algorithm 10. In summary, we want to test what values the network can reach while still explaining the data well. We do this by perturbing the network in a positive direction and a negative direction and subsequently testing which linear combinations would still explain the data reasonably well, i.e., linear combinations with a test statistic smaller than $\chi^2_{1-\alpha}(1)$.

Algorithm 10 Pseudocode for the construction of $\mathrm{CI}(f_{\boldsymbol{\theta}_0}(X_0))$ using DeepLR

Require: $\mathcal{D} = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$, $\hat{\boldsymbol{\theta}}$, X_0, α;

$n_{\text{extra}} = 2n/(\text{batch size})$;

For binary classification $c_{\max} = 1$ and $c_{\min} = 0$, and for regression $c_{\max} = f_{\hat{\boldsymbol{\theta}}}(X_0) + \delta$ and $c_{\min} = f_{\hat{\boldsymbol{\theta}}}(X_0) - \delta$;

$\tilde{\mathcal{D}}_+ = \{(X_1, f_{\hat{\boldsymbol{\theta}}}(X_1)), \ldots, (X_n, f_{\hat{\boldsymbol{\theta}}}(X_n)), \overbrace{(X_0, c_{\max}), \ldots, (X_0, c_{\max})}^{n_{\text{extra}} \text{ times}}\}$;

$\tilde{\mathcal{D}}_- = \{(X_1, f_{\hat{\boldsymbol{\theta}}}(X_1)), \ldots, (X_n, f_{\hat{\boldsymbol{\theta}}}(X_n)), \overbrace{(X_0, c_{\min}), \ldots, (X_0, c_{\min}}^{n_{\text{extra}} \text{ times}}\}$;

Make two copies of network parametrized by $\hat{\boldsymbol{\theta}}$ and train them on $\tilde{\mathcal{D}}_+$ and $\tilde{\mathcal{D}}_-$ using the original training procedure. During the training, the loss contribution of the added data points is divided by n_{extra}. Denote the parameters of the resulting networks with $\hat{\boldsymbol{\theta}}_+$ and $\hat{\boldsymbol{\theta}}_-$;

for $c \in \mathbb{R}$ ($\triangleright$ since testing all possible c is impossible, we propose to use some variation of a bisection search algorithm) **do**

  if $c > f_{\hat{\boldsymbol{\theta}}}(X_0)$ **then**

    Pick λ such that $(1-\lambda) f_{\hat{\boldsymbol{\theta}}}(X_0) + \lambda f_{\hat{\boldsymbol{\theta}}_+}(X_0) = c$ with corresponding density p_c; $\triangleright$ The density p_c is obtained by taking the same linear combination of the predicted distribution parameters outputted by the networks parametrized by $\hat{\boldsymbol{\theta}}$ and $\hat{\boldsymbol{\theta}}_+$;

  end if

  if $c < f_{\hat{\boldsymbol{\theta}}}(X_0)$ **then**

    Pick λ such that $(1-\lambda) f_{\hat{\boldsymbol{\theta}}}(X_0) + \lambda f_{\hat{\boldsymbol{\theta}}_-}(X_0) = c$ with corresponding density p_c;

  end if

  if $2\left(\sum_{i=1}^n \log(p_{\hat{\boldsymbol{\theta}}}(Y_i \mid X_i)) - \sum_{i=1}^n \log(p_c(Y_i \mid X_i)) \right) \leq \chi^2_{1-\alpha}(1)$ **then**

    Include c in $\mathrm{CI}(f_{\boldsymbol{\theta}_0}(X_0))$;

  end if

end for

return $\mathrm{CI}(f_{\boldsymbol{\theta}_0}(X_0))$

5.2.6 Related Work

In this subsection, we place our work in context of existing work while simultaneously highlighting the strengths and limitations of our approach. Our aim is not to give a complete overview of all uncertainty quantification methods, for

which we refer to the various reviews and surveys on the subject (Abdar et al., 2021; Gawlikowski et al., 2023; He and Jiang, 2023). Instead, we discuss several broad groups in which most methods can be categorized: Ensembling methods, Bayesian methods, frequentist methods, and distance-aware methods.

Ensembling methods train multiple models and use the variance of the predictions as an estimate for model uncertainty (Heskes, 1997; Lakshminarayanan et al., 2017; Zhang et al., 2017a; Wenzel et al., 2020; Jain et al., 2020; Dwaracherla et al., 2022). While being extremely easy to implement, they can be computationally expensive due to the need to train multiple networks. Moreover, the resulting confidence intervals can behave poorly in regions with a limited amount of data where the predictor is likely biased. Additionally, ensemble members may interpolate in a very similar manner, potentially leading to unreasonably narrow confidence intervals.

Bayesian approaches place a prior distribution on the model parameters and aim to simulate from the resulting posterior distribution given the observed data (MacKay, 1992a; Neal, 2012; Hernández-Lobato and Adams, 2015). Since this posterior is generally intractable, it is often approximated, with MC-Dropout being a notable example (Gal and Ghahramani, 2016; Gal et al., 2017).

A downside is that these methods can be challenging to train, potentially resulting in a lower accuracy. Our proposed approach does not change the optimization procedure and therefore has no accuracy loss. Another downside is that while asymptotically Bayesian credible sets become confidence sets by the Bernstein-von Mises theorem (see Van der Vaart (2000, Chapter 10))¿ However, this theorem generally does not apply for a neural network where the dimension of the parameter space typically exceeds the number of data points. Moreover, the prior distribution is often chosen out of computational convenience instead of being motivated by domain knowledge.

Distance-aware methods have a more pragmatic nature. They use the dissimilarity of a new input compared to the training data as a metric for model uncertainty (Lee et al., 2018; Van Amersfoort et al., 2020; Ren et al., 2021). As we will see in the next section, our method also exhibits this distance-aware property, albeit for a different reason: The further away from the training data, the easier it becomes for the network to change the predictions without negatively affecting the likelihood of the training data.

Frequentist methods use classical parametric statistics to obtain model uncertainty estimates. The typical approach - more elaborately explained in text-

books on parametric statistics, e.g. Seber and Wild (2003) - involves obtaining (an estimate of) the variance of the model parameters using asymptotic theory and then converting this variance to the variance of the model predictions using the delta method. This approach has been used by various authors to create confidence intervals for neural networks (Kallus and McInerney, 2022; Nilsen et al., 2022; Deng et al., 2023; Khosravi et al., 2011).

Confidence intervals of this type are often referred to as Wald-type intervals. These intervals are necessarily symmetric. Various authors have noted that, for classical models, Wald-type intervals often behave worse than likelihood-ratio type intervals in the low-data regime (Hall and La Scala, 1990; Andersen et al., 2012; Murphy, 1995; Murphy and van der Vaart, 1997). Specifically, when the loglikelihood cannot be effectively approximated with a quadratic function, Wald-type intervals may behave very poorly (Pawitan, 2001, Chapter 2). The significant advantage of Wald-type intervals in the classical setting is the easier computation. However, while only a single model needs to be fitted, the necessary inversion of a high-dimensional $p \times p$ matrix and the quadratic approximation of the likelihood strongly rely on being in the asymptotic regime.

Conversely, the construction of the DeepLR confidence interval does not rely on a quadratic approximation of the likelihood, which is in general only valid asymptotically. While we still utilize asymptotic theory to determine the distribution of our test statistic (see our proof of Theorem 5.A.1 in Appendix 5.A), we do impose the extremely strong requirement that the second derivative of the loglikelihood must converge and be invertible. We only require this second derivative to behave nicely in a single direction, which is a much weaker requirement.

Another benefit of likelihood-ratio-based confidence intervals is that they are transformation invariant (Pawitan, 2000). In other words, a different parametrization of the distribution does not alter the resulting confidence intervals. This is not the case for most Wald-type intervals or Bayesian approaches.

The main limitation of DeepLR in its current form is the computational cost. We compare the computational costs of the various approaches in Table 5.1. Our method comes at no extra training cost but requires two additional networks to be trained for every confidence interval. Ensembling methods also need to train multiple networks, typically from five to ten, but these networks can be reused for different confidence intervals. Frequentist methods typically come at no additional training cost but require the inversion of a $p \times p$ matrix to construct the confidence interval. The cost of Bayesian methods varies drastically from method to method. The training process can be substantially

Table 5.1: A comparison of the computational costs of different types of methods that create confidence intervals or credible regions.

Approach	Additional training cost	Additional inference cost
DeepLR	No additional costs	Two additional networks per new input and multiple additional forward passes.
Ensembling	Multiple, typically five to ten, extra networks need to be trained.	A forward pass through each of the ensemble members.
Frequentist	None	Inversion of a $p \times p$ matrix.
Bayesian	Varies. MC dropout requires no additional training costs.	Varies. Typically, every new input requires a large number of forward passes.

more involved and the construction of a credible set requires multiple samples from the (approximate) posterior.

In summary, a likelihood-ratio-based method is distance aware, transformation invariant, has no accuracy loss, and is capable of creating asymmetric confidence intervals. However, it comes with the downside of being computationally expensive. Producing a confidence interval for a single input requires the training of two additional networks.

5.3 Experimental Results

In this section, we present the results of various experiments that showcase the desirable properties of DeepLR, such as its distance-aware nature and capability to create asymmetric confidence intervals. The high computational cost of our method prohibits any large-scale experiments. Nevertheless, the following experiments demonstrate the effectiveness of a likelihood-ratio-based approach.

5.3.1 Toy Examples

To start, we present two one-dimensional toy examples that effectively illustrate the behavior of our method. Specifically, these examples illustrate the capability of our method to produce asymmetric confidence intervals that expand in regions with a limited amount of data points, both during interpolation and extrapolation.

Regression The data set consists of 80 realisations of the random variable pair (X, Y). Half of the x-values are sampled uniformly from the interval $[-1, -0.2]$, while the remaining half are sampled uniformly from the interval $[0.2, 1]$. The y-values are subsequently sampled using

$$Y \mid X = x \sim \mathcal{N}\left(2x^2, 0.1^2\right).$$

On this training set, we train a network by minimizing the mean-squared error, which is equivalent to maximizing the loglikelihood while assuming a normal distribution with homoscedastic variance. We use the mean-squared-error of the residuals as an estimate for the variance (this estimate is updated for every evaluation of the test statistic). The network is trained for 400 epochs, using a default Adam optimizer (Kingma and Ba, 2014) and a batch size of 32. The network consists of 3 hidden layers with 40, 30, and 20 hidden units respectively. All layers have elu activation functions (Clevert et al., 2015) with l_2-regularization applied in each dense layer with a constant value of 1e-4.

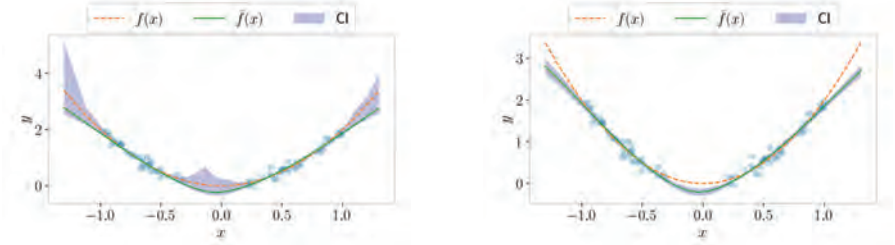
Figure 5.3 gives the 95% confidence intervals for mean predictions. Notably, in the biased region around 0, the intervals become highly asymmetric. In contrast, most other methods produce symmetric interval around the original network. In a region with a bias, this can easily lead to intervals with very poor coverage. This is exemplified by the biased symmetric intervals generated by an ensemble consisting of 10 ensemble members (we used the ensembling strategy employed by Lakshminarayanan et al. (2017)).

In Appendix 5.C, we demonstrate that our likelihood-ratio-based approach also works for this example with the popular XGBoost model (Chen and Guestrin, 2016). While our chapter focuses on applying the methodology to neural networks, this highlights that likelihood-ratio-based uncertainty quantification methods could also be developed for other types of models.

Binary classification The data set consists of 60 realisations of the random variable pair (X, Y), where half of the x-values are sampled uniformly from the interval $[0, 0.2]$ and the other half are sampled uniformly from the interval $[0.8, 1]$. The y-values are subsequently simulated using

$$Y \mid X = x \sim \text{Ber}(p(x)), \quad \text{with } p(x) = 0.5 + 0.4\cos(6x).$$

On this training set, we train a fully connected network with three hidden layers consisting of 30 hidden units with elu activations functions. The final layer



(a) DeepLR (b) Ensemble

Figure 5.3: This figure illustrates DeepLR for a regression problem. The blue dots indicate the locations of the training points, the dotted orange line represents the true function, and the solid green line represents the predicted regression function of the network. The shaded blue region gives the 95% CI of the regression function. DeepLR exhibits two desirable properties when compared to an ensemble approach. Firstly, the intervals expand in regions where data is sparse. Secondly, the intervals can be asymmetric, allowing for the compensation of potential bias.

outputs a logit that is transformed using a sigmoid to yield a class probability. The network is trained for 300 epochs using a binary-crossy-entropy loss function and the Adam optimizer with a batch size of 32.

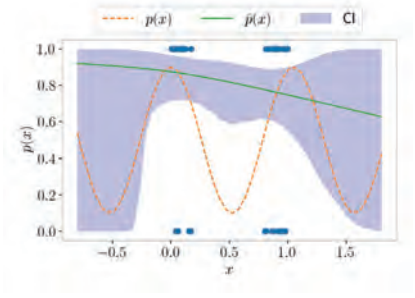
The resulting 95% confidence intervals of the predicted probability of class 1 are given in Figure 5.4. We carried out the experiment with two amounts of regularization to illustrate how this affects the result. For comparison, we also implemented an ensembling approach and MC-Dropout (see Lakshminarayanan et al. (2017) and Gal and Ghahramani (2016) for details). We used ten ensemble members and a standard dropout rate of 0.2. All networks were trained using the same training procedure.

We observe the same desirable properties as in the regression example. The intervals get much larger in regions with a limited amount of data, also when interpolating, and can become asymmetrical. Additionally, the intervals get smaller when we increase the amount of regularization. The model class becomes smaller (fewer parameters can be reached) making it more difficult for the model to change the predictions without affecting the likelihood. This, in turn, leads to smaller confidence intervals. If the model is overly regularized, it will become miss specified ($\boldsymbol{\theta}_0 \notin \Theta$) and the resulting confidence intervals will not be correct.

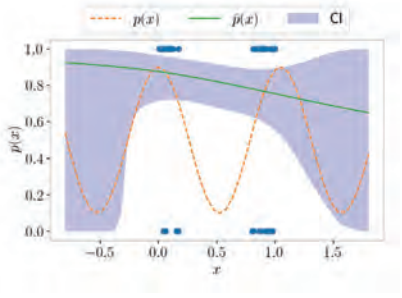
This effect illustrates the trade-off between assumptional uncertainty and parameter uncertainty that we explained in Chapter 1. A more flexible model

is more likely to contain the true function but it is more difficult to find the optimal parameters, resulting in a larger confidence interval.

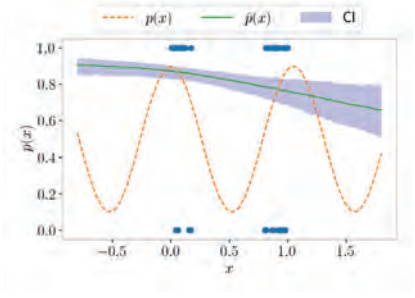
The other approaches do not share the same qualitative properties. The ensembling approach results in confidence intervals that are far too narrow. All ensemble members behave more or less the same, especially when interpolating. The MC-Dropout credible regions do not expand in the regions in between the data and also only moderately expand when extrapolating.



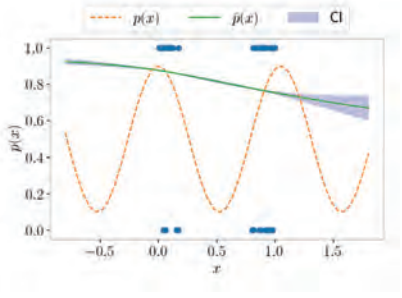
(a) DeepLR, regularization constant 1e-5



(b) DeepLR, regularization constant 1e-4

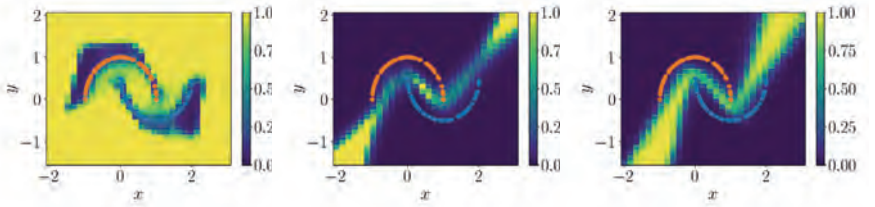


(c) Dropout



(d) Ensemble

Figure 5.4: This figure illustrates DeepLR for a binary classification problem. The blue dots represent the training data, the dotted orange line the true probability of class 1 and the solid green line the predicted probability of class 1. The shaded blue region provides the 95% CI for the predicted probability of class 1. Figures (a) and (b) demonstrate the behavior of DeepLR for varying amounts of regularization. The more regularization, the smaller the class of admissible functions becomes, which naturally results in smaller intervals. Additionally, the top left figure demonstrates that the intervals expand when interpolating, a feature not shared by the dropout and ensembling approach.



(a) DeepLR (b) Ensembling (c) Dropout

Figure 5.5: A comparison of the confidence intervals of our likelihood-ratio approach (DeepLR), an ensembling approach, and MC-Dropout on the two-moon data set. The colorbar represents the width of 95% confidence intervals, where yellow indicates greater uncertainty. The orange and the blue circles indicate the location of the data points of the two different classes. Crucially, DeepLR presents high levels of uncertainty in regions far away from the data, generating confidence intervals of [0.00, 1.00], unlike the other methods that display extreme certainty in those regions.

5.3.2 Two-moon Example

The data set consists of 80 data points, generated using the `make_moons` function from the `scikit-learn` package, which creates a binary classification problem with two interleaving half circles (Pedregosa et al., 2011).

We utilize the same network architecture as in the toy classification example. The network is trained for 500 epochs using the Adam optimizer with default learning rate and batch size of 32, while also applying l_2-regularization in each layer with a constant value of 1e-3.

Figure 5.5 presents the 95% confidence intervals for the predicted class probabilities. The results illustrate that DeepLR becomes extremely uncertain in regions farther from the data (i.e., the confidence interval for the class probability spans the full range of $[0, 1]$).

In contrast, both an ensembling approach and MC-Dropout report excessively high certainty in the upper left and lower right regions. The ensemble's behavior can be attributed to all ensemble members extrapolating in the same direction causing all ensemble members to report more or less the same class probability. For MC-dropout, a saturated sigmoid is causing the narrow credible intervals.

This comparison underscores the unique capability of DeepLR to provide more accurate uncertainty estimates in regions less well represented by data – a crucial capability in practical applications.

5.3.3 MNIST Binary Example

For a more difficult task, we train a small convolutional network on the first two classes of the MNIST data set, consisting of handwritten digits. In this binary classification task, the 0's labeled as class 0 and the 1's as class 1.

The CNN architecture consists of two pairs of convolutional layers (with 28 filters and 3x3 kernels) and max-pooling layers (2x2 kernel), followed by a densely connected network with two hidden layers with 30 hidden units each and elu activation functions.

The network is trained for 10 epochs, using the SGD optimizer with a batch size of 32, default learning rate, and l_2 regularization with a constant value of 1e-5, and binary cross-entropy loss function. The amounts of training time and regularization were determined from a manual grid search using an 80/20 split of training data. For the actual experiment, the entire training set was utilized.

Figure 5.6 presents 95% CIs for a number of different training points, test points, and OoD points. As shown, the OoD points have wider confidence intervals than the training and test points, reflecting greater uncertainty.

In an additional experiment, we rotated one of the test-points and created confidence intervals for the rotated images. As Figure 5.7 illustrates, increasing the rotation angle results in larger confidence intervals. This behavior is also seen with the ensembling and MC-dropout, but to a significantly lesser extent.

These larger intervals can be explained as follows: Our approach essentially asks the intuitive question how much the network can change the prediction for this new input without overly affecting the likelihood of the training data. A heavily rotated image of a number 1 deviates greatly from the typical input, utilizing pixels that are almost never used by the training inputs. It is therefore relatively straightforward for the network to change the prediction of this rotated input without dramatically lowering the likelihood of the training data. This, in turn, results in very large confidence intervals, accurately indicating that it is a very unfamiliar input.

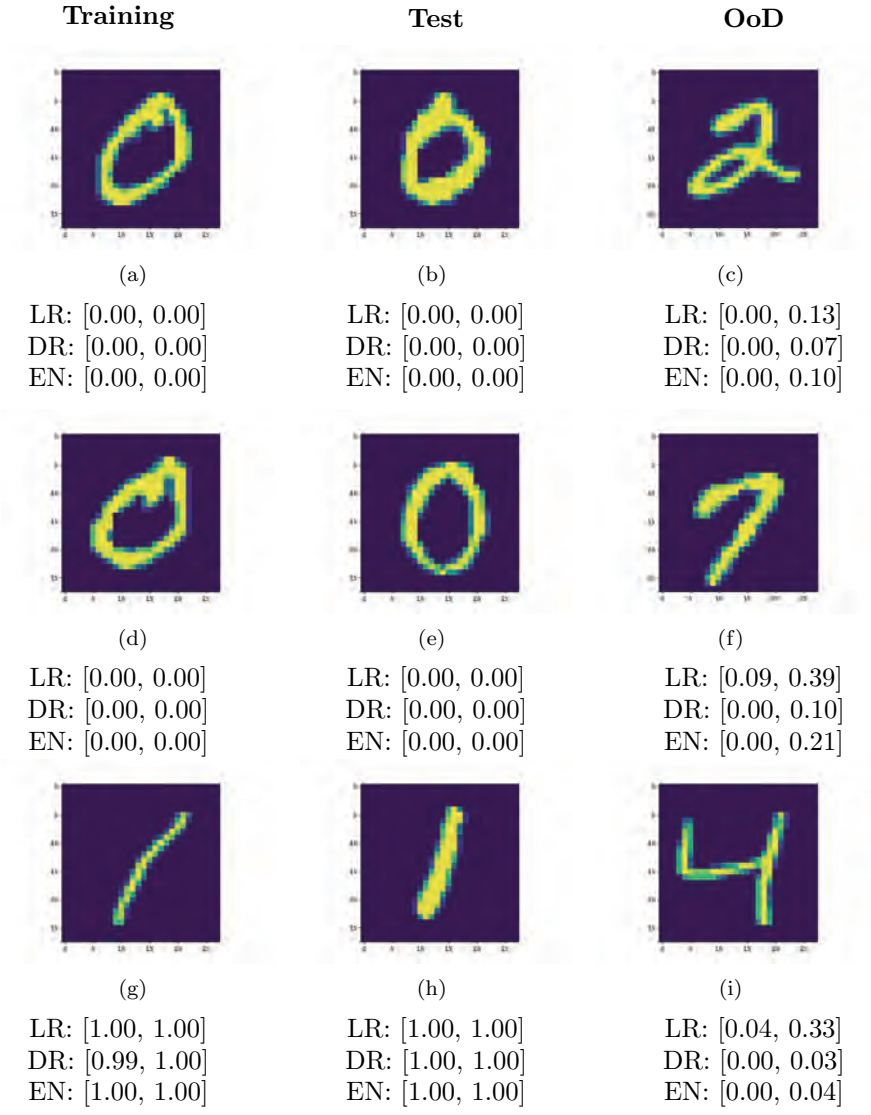


Figure 5.6: 95% CIs for different training points (first column), test points (second column), and OoD points (third column). Each subcaption displays the 95% CI for the probability of class 1 made with DeepLR (LR), MC-dropout (DR), and ensembling (EN). For in-distribution points (zeros and ones), all three methods provide extremely narrow intervals. For the OoD points, our method provides much larger CIs, a property also present in the other methods, albeit to a lesser extent.

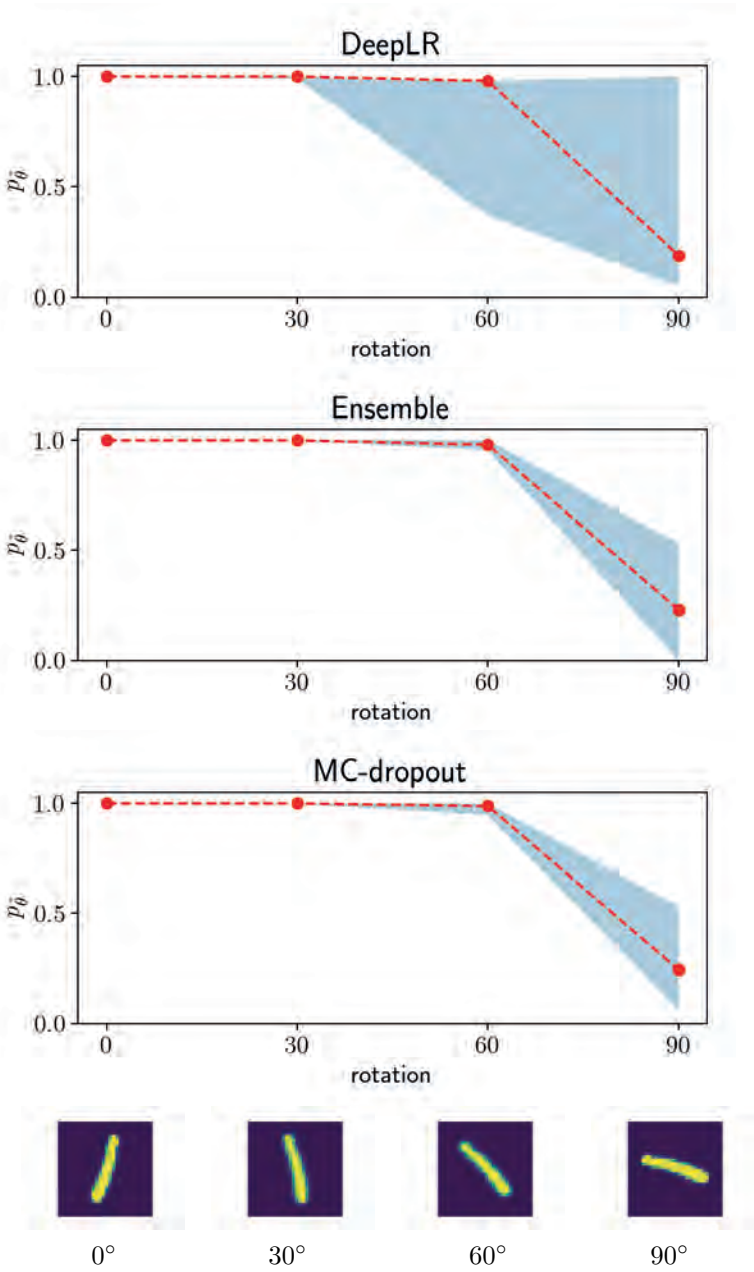


Figure 5.7: This figure provides 95% CIs for different amounts of rotation. For rotations of 0 and 30 degrees, all methods produce very narrow confidence intervals, implying high certainty. At 60 degrees of rotation, DeepLR outputs high uncertainty whereas the ensembling approach and MC-dropout remain fairly certain. At 90 degrees of rotation, DeepLR outputs very high uncertainty, a confidence interval of [0.05, 1.00], a behavior also observed to a lesser extent in both ensembling and MC-dropout.

5.3.4 CIFAR Binary Example

We extend the previous experiment to the CIFAR10 data set, using the first two classes – planes and cars – as a binary classification problem.

We use a CNN consisting of two pairs of convolutional layers (with 32 filters and 3x3 kernels) and max-pooling layers (2x2 kernel), followed by a densely connected network with three hidden layers with 30 hidden units each and elu activation functions.

The CNN is trained for 15 epochs using the SGD optimizer with default learning rate, a batch size of 32, and l_2-regularization with a constant value of 1e-5. The training time and regularization are determined the same way as for the MNIST experiment, using an 80/20 split of the training data.

Figure 5.8 presents 95% confidence intervals for several training, test, and OoD points. Our method is uncertain for out-of-distribution inputs. However, contrary to the MNIST example, we also see that the model is uncertain for various in-distribution points. Figures 5.8(e) and 5.8(g) provide examples of such uncertain predictions. The exact reason why the model is uncertain for those inputs remains speculation. Possible explanations might be the open hood of the car in (e) or the large amount of blue sky in (g). An interesting avenue for future work would be to investigate what specific features cause DeepLR to output greater uncertainty.

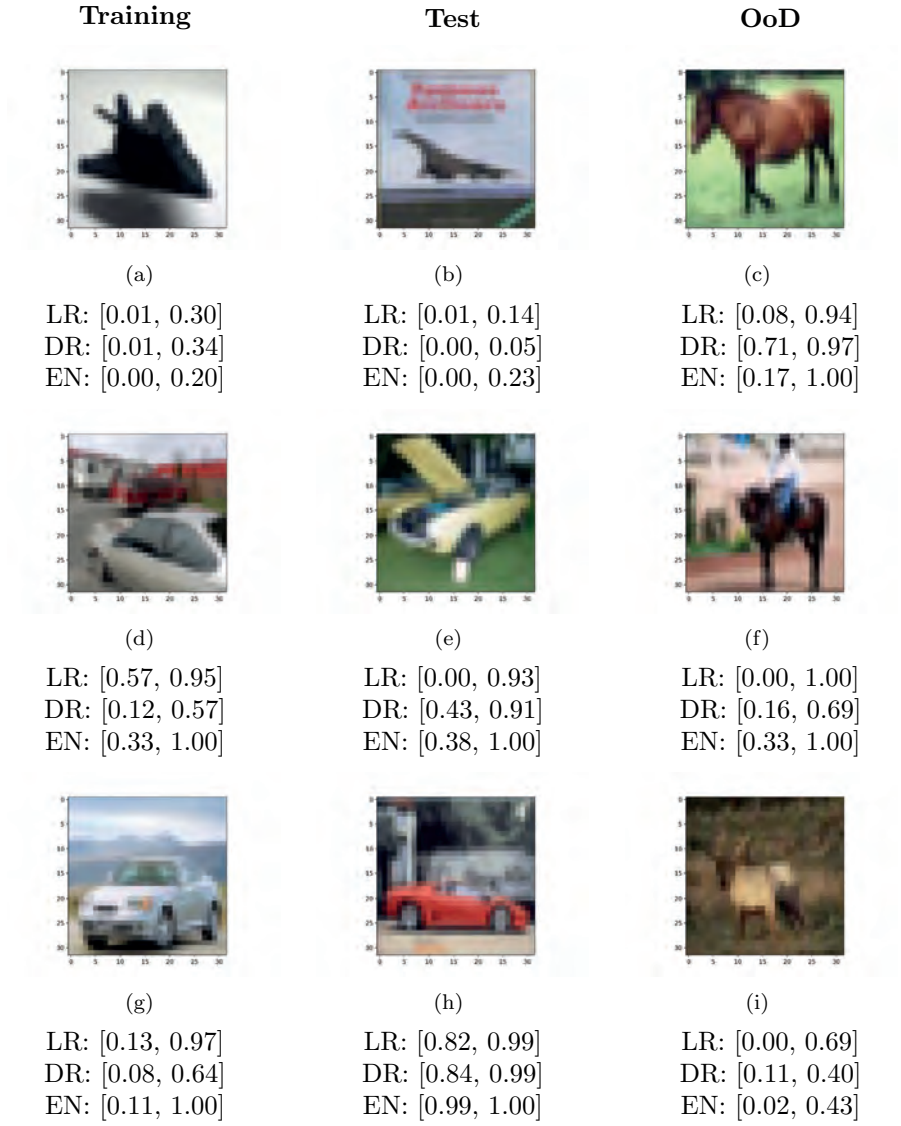


Figure 5.8: This figure gives 95% CIs for various training points (first column), test points (second column), and OoD points (third column). Each subcaption provides the CIs for the probability of class 1 (cars) made with DeepLR (LR), MC-dropout (DR), and ensembling (EN). All methods demonstrate greater uncertainty than in the MNIST example, which is sensible as the CIFAR data set is significantly harder. Our method is very uncertain for all OoD points, which is not always the case for MC-dropout (c and i) and ensembling (i). We also observe relatively uncertain predictions by all methods for some in-distribution points, notably (e) and (g).

5.3.5 Brain Tumor Example

As we mentioned before, there are scenarios where the high computational cost can be justified. As an example, we consider the task of detecting brain tumors in MRI images. Compared to the very high costs of making the MRI image, the extra computational cost of retraining the network in order to get an uncertainty estimate is justified.

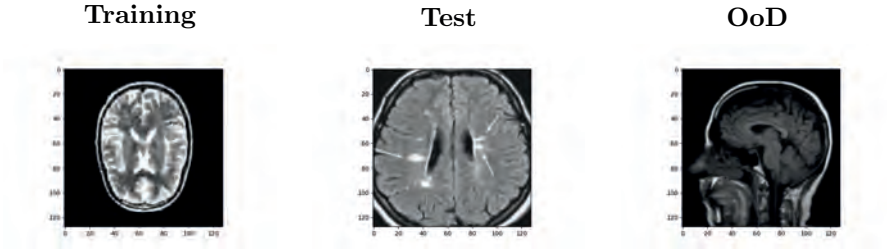
As a training set, we use the Br35h data set (Hamanda, 2020), conisting of 1500 MRI images that contain a tumor and 1500 MRI images that do not contain a tumor. All images, see Figure 5.9 for examples, are taken in the axial (horizontal) plane above the level of the eyes. As OoD inputs, we take two images from a second MRI data set (Sartaj, 2020). The first OoD image is in the sagittal (vertical) plane. The second OoD image is in the axial plane but contains the eyes.

For this experiment, we use a slightly larger CNN consisting of four blocks of convolutional layers with max-pooling and batch-normalization – the first two blocks have 3x3 kernels with 8 filters, and the last two have 3x3 kernels with 16 filters – followed by three densely connected layers with 20, 10, and 5 hidden units respectively, batch-normalization, and elu activation functions.

The CNN is trained for 30 epochs using the SGD optimizer with default learning rate, a batch size of 32, and l_2-regularization with a constant value of 1e-5. The training time and regularization are determined in the same way as during the MNIST and CIFAR experiment. The resulting model has roughly 95% accuracy on the unseen test set.

Contrary to the previous experiments, we keep the convolutional layers fixed during the retraining to reduce the computational cost. This illustrates that it is possible to use cheaper procedures to approximate the test statistic.

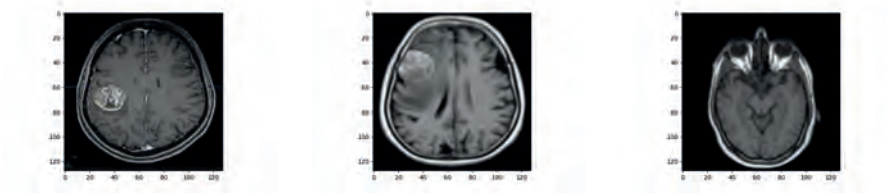
The results for six different inputs are visualized in Figure 5.9. DeepLR produces extremely narrow intervals for all four in-distributions points and very wide intervals for both OoD points. While the ensembling approach also produces large intervals for the OoD points, the intervals are also wide for the test points and to a lesser extent for the training points. MC-dropout produces wide intervals for all inputs except for one of the two training inputs (a). Notably, MC-dropout the intervals are not wider for the OoD inputs than for the test inputs.



(a)
LR: [0.00, 0.03]
DR: [0.00, 0.19]
EN: [0.00, 0.02]

(b)
LR: [0.00, 0.02]
DR: [0.00, 0.72]
EN: [0.00, 0.62]

(c)
LR: [0.02, 0.82]
DR: [0.00, 0.25]
EN: [0.00, 1.00]



(d)
LR: [0.99, 1.00]
DR: [0.68, 0.99]
EN: [0.97, 1.00]

(e)
LR: [0.94, 0.99]
DR: [0.53, 1.00]
EN: [0.86, 1.00]

(f)
LR: [0.00, 0.93]
DR: [0.00, 0.43]
EN: [0.00, 1.00]

Figure 5.9: 95% CIs for different training points (first column), test points (second column), and OoD points (third column). Each subcaption displays the 95% CI for the probability of class 1 (a tumor) made with DeepLR (LR), MC-dropout (DR), and ensembling (EN). DeepLR produces very narrow CIs both for the two training and the two test inputs. For the OoD inputs, the intervals become substantially wider. MC-dropout produces wide inputs for all three types of inputs. Ensembling produces very narrow intervals for both training inputs but relatively large interval for one of the test inputs (b). Both OoD inputs have a very wide CI.

5.3.6 Adversarial Example

In addition to the previous experiments, we briefly tested how the method deals with adversarial examples (Goodfellow et al., 2014b). Adversarial examples are modified inputs that are specifically designed to mislead the model, typically by adding small perturbations to the input. While virtually imperceptible to humans, these perturbations can dramatically alter the model's prediction.

We constructed adversarial versions of the two most confident test-inputs in Figure 5.8 using the FGSM method (Goodfellow et al., 2014b), which works by using the gradients of the networks' loss function with respect to the input to create an input that maximizes the loss.

As illustrated in Figure 5.10, DeepLR demonstrates a higher uncertainty for the two adversarial inputs. This effect can be explained as follows. Although very similar to a human observer, an adversarial example is significantly different to a neural network. The difference was so large that both adversarial examples were wrongly classified by the network. Since these adversarial examples differ significantly from the training data, the network can more easily change the prediction at this location without changing the other predictions and thus the likelihood of the training data too much. This results in much larger confidence intervals.

These results provide an encouraging sign that our method could also be able to handle adversarial examples, offering the unique capability to create confidence intervals, detect OoD examples, and offer robustness against adversarial examples, all with a single method. Unfortunately, the high computational cost of the method prevents more large-scale comparisons with other methods to more firmly establish this potential.

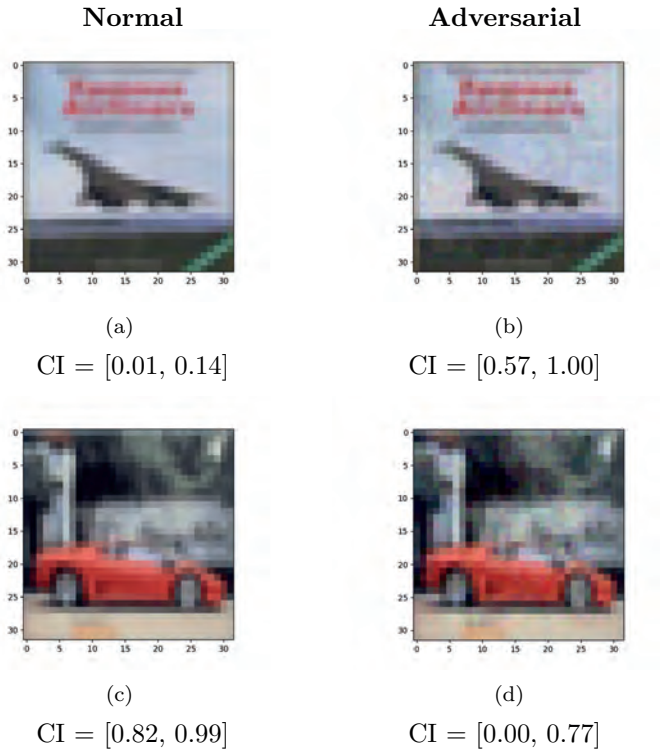


Figure 5.10: Adversarial inputs generated by the FGSM method result in more uncertain predictions. The same network was used as for the CIFAR example illustrated in Figure 5.8. The intuition behind the larger CIs is as follows. While very similar to humans, the adversarial examples are significantly different to a neural network. This allows the network to change the prediction for the adversarial example without changing the predictions of the training data, resulting in larger confidence intervals.

5.4 Discussion and Conclusion

In this chapter, we demonstrated the potential of a likelihood-ratio-based uncertainty estimate for neural networks. This approach is capable of producing asymmetric confidence intervals that are better motivated in cases where we have fewer data points than parameters, i.e. most deep learning applications.

The experimental results verify the theoretical advantages of a likelihood-ratio-based approach. The intervals are larger in regions with fewer data points, can get asymmetric in biased regions, and get larger for OoD and adversarial outputs. However, not being specifically designed for OoD detection or robustness against adversarial attacks, we do not claim it to be competitive in this regard against tailor-made alternatives.

While we made an effort to reduce it, our method still has some variance and can produce slightly different intervals upon repetition due to the randomness of the optimization procedure. This effect is greater for larger data sets where small differences can have a large effect on the test statistic.

Furthermore, it is essential that the model is well specified. This is the case for any model and not specific to our method. If the true density $p_{\boldsymbol{\theta}_0}$ cannot be reached, the resulting confidence intervals will surely be wrong. A model can be miss specified if it is overly regularized or if incorrect distributional assumptions are made (e.g. incorrectly assuming Gaussian noise).

In its current form, the high computational cost makes DeepLR unsuitable for many deep learning applications. A self-driving car that is approaching a cross-section cannot stop and wait for an hour until it has an uncertainty estimate. Nevertheless, a trustworthy uncertainty estimate may be critical in certain situations, or only a limited number of confidence intervals may be required. For instance, for medical applications, the extra computational time may be worthwhile. Alternatively, for some applications within astrophysics, only very few confidence intervals may be needed. If only a single interval is needed, for instance, our method is cheaper than an ensemble.

5.4.1 Future Work

Overall, our findings highlight the potential of a likelihood-ratio-based approach as a new branch of uncertainty estimation methods. We hope that our work will inspire further research in this direction. Several areas for improvement include:

- Reduced computational cost: A clear limitation of the current implementation is the cost. For every confidence interval, we need to train two additional networks. We acknowledge that this is infeasible for many - although not all - applications and we hope that the proof of concept in this chapter motivates further research in this direction that may result in reduced computational cost.

- Improved approximation of the test statistic: The calculation of the test statistic uses an approximation for the second term in equation (5.1). Further research could focus on finding better and possibly cheaper approximations. It may also be worthwhile to investigate the use of a Bartlett correction. There are multiple ways to approximate the test statistic and we do not claim that our proposal is the optimal one. In fact, in this chapter we use three slightly different approaches. For the brain-tumor example, we only retrained parts of the network. For the XGBoost example in Appendix 5.C, we simply retrained the entire model from scratch, and for the other examples, we perturbed the networks starting from the original one. This new branch of uncertainty-estimation methods should focus on improving the approximation of the test statistic.
- Application to other machine-learning models: We applied this approach to neural networks. However, the methodology should also be applicable to other models, for example random forests. Especially for models that are relatively cheap to train, this approach could be very promising. In Appendix 5.C, we demonstrated that a similar approach is also viable for the XGBoost model.
- Extension beyond binary classification: We currently approximate the test statistic by retraining the network two times and using a linear combination of the original and the perturbed network. This does not easily extend to multiple classes. In that case, we would need to retrain the network in many more directions and we would no longer have a one-dimensional interpolation problem.
- Development of the theory on the distribution of the test statistic: It would be interesting to develop the theory surrounding the distribution of the test statistic in greater generality, possibly also when explicitly considering a regularization term. We constructed a reparametrization that showed that, under some assumptions, the test statistic has a $\chi^2(1)$ distribution. It would be interesting to study these assumptions further.
- A better understanding of what causes DeepLR to become uncertain: We saw, for example, that various planes and cars had rather large accompanying confidence intervals. It would be interesting to study what causes certain input to become more uncertain than others.

APPENDIX CHAPTER 5

5.A Proof of Theorem 5.A.1

Theorem 5.A.1. *Assume that Θ contains an open subset around $\boldsymbol{\theta}_0$ in $\mathbb{R}^p$. Let $l(\mathcal{D};\boldsymbol{\theta})$ be the loglikelihood of the data given $\boldsymbol{\theta}$. Define $\Theta_0(c) = \{\boldsymbol{\theta} \in \Theta \mid f_{\boldsymbol{\theta}}(X_0) = c\}$ and denote the tangent space of $\Theta_0(c)$ at $\boldsymbol{\theta}_0$ with $T_{\boldsymbol{\theta}_0}\Theta_0(c)$. Also define $\hat{\boldsymbol{\theta}}$ as the Maximum Likelihood Estimator (MLE) of $\boldsymbol{\theta}_0$ and $\hat{\boldsymbol{\theta}}_0$ as the MLE when we restrict our parameterspace to $\Theta_0(c)$.*

We assume:

A1: $l(\mathcal{D};\boldsymbol{\theta})$ is three times continuously differentiable in a neighbourhood of $\boldsymbol{\theta}_0$,

A2: $\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}_0 + o_p(n^{-1/4})$ and $\hat{\boldsymbol{\theta}}_0 = \boldsymbol{\theta}_0 + o_p(n^{-1/4})$,

A3: There exists a vector $\hat{n} \in \mathbb{R}^p$ such that $h_0^T \ddot{l}(\boldsymbol{\theta}_0)\hat{n} = O_p(n\|h_0\|^2)$ for all $h_0 \in (\Theta_0(c) - \boldsymbol{\theta}_0)$. Also, $\hat{\boldsymbol{n}}$ is transversal to $T_{\boldsymbol{\theta}_0}\Theta_0(c)$.

A4: There exists a constant $i > 0$ such that $\frac{1}{n}\hat{\boldsymbol{n}}^T \ddot{l}(\boldsymbol{\theta}_0)\hat{n} = -i + o_p(1)$.

Under these assumptions, the test statistic

$$T(c) = 2\left(\sup_{\Theta} l(\mathcal{D};\boldsymbol{\theta})) - \sup_{\Theta_0(c)} l(\mathcal{D};\boldsymbol{\theta})\right)$$

converges in distribution to a $\chi^2(1)$ distribution.

We will first prove this theorem and then comment on the assumptions we make.

Proof. Our strategy is to construct a reparametrization

$$\Psi : \Theta_0(c) \times \mathbb{R} \to \Theta : (\tilde{\boldsymbol{\theta}}, t) \mapsto \Psi(\tilde{\boldsymbol{\theta}}, t), \quad \text{such that}$$

$$L(\mathcal{D}; \Psi(\tilde{\boldsymbol{\theta}}, t)) \approx L(\mathcal{D}; \tilde{\boldsymbol{\theta}})\phi(\mathcal{D}; t),$$

with high probability for $\tilde{\boldsymbol{\theta}} \in \Theta_0$ close to $\boldsymbol{\theta}_0$, for some function ϕ. In other words, we use a parametrization such that the likelihood factorizes in a part that depends on $\tilde{\boldsymbol{\theta}} \in \Theta_0(c)$, and a part that depends on $t \in \mathbb{R}$. If we can construct such a parametrization, then $T(c)$ will reduce to 2 times the loglikelihood-ratio of a one-dimensional model, which is known to converge in distribution to a $\chi^2(1)$ distribution.

We use the following parametrization:

$$\boldsymbol{\theta} = P(\boldsymbol{\theta}) + t\hat{\boldsymbol{n}},$$

with $P(\boldsymbol{\theta})$ being the projection onto Θ_0. In a neighbourhood of $\boldsymbol{\theta}_0$ this projection (i.e., the closest point in Θ_0 to $\boldsymbol{\theta}$) is uniquely defined and since $\hat{\boldsymbol{n}}$ is not tangent to $T_{\boldsymbol{\theta}_0}\Theta_0(c)$, t is also unique. We define $\hat{\boldsymbol{h}}_0$ and $\hat{\boldsymbol{h}}_0^{(0)}$ such that

$$\hat{\boldsymbol{h}}_0 = P(\hat{\boldsymbol{\theta}}) - \boldsymbol{\theta}_0 \quad \text{and} \quad \hat{\boldsymbol{h}}_0^{(0)} = \hat{\boldsymbol{\theta}}_0 - \boldsymbol{\theta}_0.$$

See Figure 5.11 for a visualization of the notation. Condition A2 implies that $\|\hat{\boldsymbol{h}}_0\| = o_p(n^{-1/4})$ and $\|\hat{\boldsymbol{h}}_0^{(0)}\| = o_p(n^{-1/4})$. We also define $\hat{t}$ such that $\hat{\boldsymbol{\theta}} = P(\hat{\boldsymbol{\theta}}) + \hat{t}\hat{\boldsymbol{n}}$. Finally, we define $\bar{t}$ by

$$\bar{t} = \operatorname*{arg\,max}_{t\in\mathbb{R}} \left(\dot{l}(\boldsymbol{\theta}_0)t\hat{\boldsymbol{n}} + \frac{1}{2}t^2\hat{\boldsymbol{n}}^T\ddot{l}(\boldsymbol{\theta}_0)\hat{\boldsymbol{n}} \right) = -\frac{\dot{l}(\boldsymbol{\theta}_0)\hat{\boldsymbol{n}}}{\hat{\boldsymbol{n}}^T\ddot{l}(\boldsymbol{\theta}_0)\hat{\boldsymbol{n}}}.$$

By condition A4, this is indeed a maximum for large enough n. The loglikelihood is a sum of n i.i.d. random variables, and the expectation has its maximum at $\boldsymbol{\theta}_0$. Therefore, $l(\boldsymbol{\theta}_0)$ and its derivatives are all $O_p(n)$, but since $\dot{l}(\boldsymbol{\theta}_0)$ is a sum of i.i.d. random variables with expectation 0, we see that $\dot{l}(\boldsymbol{\theta}_0) = O_p(n^{1/2})$; this implies that $\bar{t} = O_p(n^{-1/2})$. In what follows, we drop the dependence of the loglikelihood on the data for notational simplicity. A Taylor expansion of the loglikelihood and using the fact that $\hat{\boldsymbol{\theta}}$ and $\hat{\boldsymbol{\theta}}_0$ are MLE's yield the following inequalities:

$$\begin{aligned}
l(\hat{\boldsymbol{\theta}}) &= l(\boldsymbol{\theta}_0 + \hat{\boldsymbol{h}}_0 + \hat{t}\hat{\boldsymbol{n}}) \\
&\geq l(\boldsymbol{\theta}_0 + \hat{\boldsymbol{h}}_0^{(0)} + \bar{t}\hat{\boldsymbol{n}}) \\
&= l(\boldsymbol{\theta}_0 + \hat{\boldsymbol{h}}_0^{(0)}) + \dot{l}(\boldsymbol{\theta}_0 + \hat{\boldsymbol{h}}_0^{(0)})\bar{t}\hat{\boldsymbol{n}} + \frac{1}{2}\bar{t}^2\hat{\boldsymbol{n}}^T\ddot{l}(\boldsymbol{\theta}_0 + \hat{\boldsymbol{h}}_0^{(0)})\hat{\boldsymbol{n}} + O_p(n\bar{t}^3) \\
&= l(\boldsymbol{\theta}_0 + \hat{\boldsymbol{h}}_0^{(0)}) + \dot{l}(\boldsymbol{\theta}_0)\bar{t}\hat{\boldsymbol{n}} + \hat{\boldsymbol{h}}_0^{(0)\,T}\ddot{l}(\boldsymbol{\theta}_0)\bar{t}\hat{\boldsymbol{n}} + \frac{1}{2}\bar{t}^2\hat{\boldsymbol{n}}^T\ddot{l}(\boldsymbol{\theta}_0)\hat{\boldsymbol{n}} \\
&\qquad + O_p(n\bar{t}^3 + n\bar{t}^2||\hat{\boldsymbol{h}}_0^{(0)}|| + n\bar{t}||\hat{\boldsymbol{h}}_0^{(0)}||^2), \qquad (5.4)
\end{aligned}$$

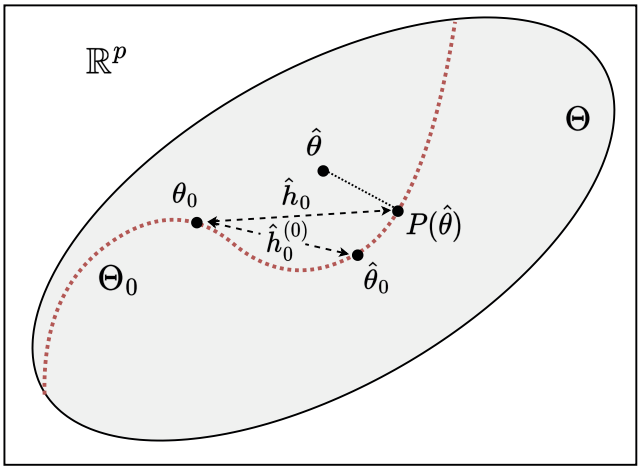


Figure 5.11: Visualization of the notation used in the proof. The shaded area represents Θ and the dotted red line $\Theta_0(c)$.

and

$$
\begin{aligned}
l(\hat{\boldsymbol{\theta}}) &= l(\boldsymbol{\theta}_0 + \hat{\boldsymbol{h}}_0 + \hat{t}\hat{\boldsymbol{n}}) \\
&= l(\boldsymbol{\theta}_0 + \hat{\boldsymbol{h}}_0) + \dot{l}(\boldsymbol{\theta}_0 + \hat{\boldsymbol{h}}_0)\hat{t}\hat{\boldsymbol{n}} + \frac{1}{2}\hat{t}^2\hat{\boldsymbol{n}}^T\ddot{l}(\boldsymbol{\theta}_0 + \hat{\boldsymbol{h}}_0)\hat{\boldsymbol{n}} + O_p(n\hat{t}^3) \\
&\leq l(\boldsymbol{\theta}_0 + \hat{\boldsymbol{h}}_0^{(0)}) + \dot{l}(\boldsymbol{\theta}_0)\hat{t}\hat{\boldsymbol{n}} + \hat{\boldsymbol{h}}_0^T\ddot{l}(\boldsymbol{\theta}_0)\hat{t}\hat{\boldsymbol{n}} + \frac{1}{2}\hat{t}^2\hat{\boldsymbol{n}}^T\ddot{l}(\boldsymbol{\theta}_0)\hat{\boldsymbol{n}} \\
&\qquad + O_p(n\hat{t}^3 + n\hat{t}^2||\hat{\boldsymbol{h}}_0|| + n\hat{t}||\hat{\boldsymbol{h}}_0||^2) \qquad (5.5) \\
&\leq l(\boldsymbol{\theta}_0 + \hat{\boldsymbol{h}}_0^{(0)}) + \dot{l}(\boldsymbol{\theta}_0)\bar{t}\hat{\boldsymbol{n}} + \hat{\boldsymbol{h}}_0^T\ddot{l}(\boldsymbol{\theta}_0)\hat{t}\hat{\boldsymbol{n}} + \frac{1}{2}\bar{t}^2\hat{\boldsymbol{n}}^T\ddot{l}(\boldsymbol{\theta}_0)\hat{\boldsymbol{n}} \\
&\qquad + O_p(n\hat{t}^3 + n\hat{t}^2||\hat{\boldsymbol{h}}_0|| + n\hat{t}||\hat{\boldsymbol{h}}_0||^2). \qquad (5.6)
\end{aligned}
$$

Combining (5.5) and (5.4), we see that $\hat{t} = O_p(\bar{t}) = O_p(n^{-1/2})$. Using A2, we also see that all terms in the remainder are $o_p(1)$. Because of A3, this is also true for $\hat{\boldsymbol{h}}_0^{(0)\,T}\ddot{l}(\boldsymbol{\theta}_0)\bar{t}\hat{\boldsymbol{n}}$ and $\hat{\boldsymbol{h}}_0^T\ddot{l}(\boldsymbol{\theta}_0)\hat{t}\hat{\boldsymbol{n}}$. Using (5.4) and (5.6), our test statistic

therefore reduces to

$$\begin{aligned} T(c) &= 2\left(\sup_{\boldsymbol{\theta}\in\Theta} l(\boldsymbol{\theta}) - \sup_{\boldsymbol{\theta}\in\Theta_0(c)} l(\boldsymbol{\theta}) \right) \\ &= 2(l(\hat{\boldsymbol{\theta}}) - l(\hat{\boldsymbol{\theta}}_0)) \\ &= 2\dot{l}(\boldsymbol{\theta}_0)\bar{t}\hat{\boldsymbol{n}} + \bar{t}^2\hat{\boldsymbol{n}}^T\ddot{l}(\boldsymbol{\theta}_0)\hat{\boldsymbol{n}} + o_p(1) \\ &= \frac{(\dot{l}(\boldsymbol{\theta}_0)\hat{\boldsymbol{n}})^2}{n \cdot i} + o_p(1). \end{aligned}$$

By the central limit theorem and a well-known property of the loglikelihood relating the variance of the derivative to the expectation of the second derivative, $\frac{1}{\sqrt{n}}\frac{d}{dt}l(\boldsymbol{\theta}_0 + t\hat{\boldsymbol{n}})|_{t=0}$ converges in distribution to a normal distribution with mean zero and covariance $i \in \mathbb{R}$. We conclude that $T(c)$ weakly converges to a $\chi^2(1)$ distribution. □

Assumption A1 is necessary for the Taylor expansion, and A2 is important for localization, but notice that the rate of convergence in directions other than $\hat{\boldsymbol{n}}$ can be slower than parametric. A4 mainly implies that the Fisher information of the relevant one-dimensional model is not 0. The more technical A3 requires more explanation: a natural choice would be to consider the expected $\ddot{l}(\boldsymbol{\theta}_0)$ and pick $\hat{\boldsymbol{n}}$ such that $\ddot{l}(\boldsymbol{\theta}_0)\hat{\boldsymbol{n}}$ is perpendicular to $\Theta_0(c)$ at $\boldsymbol{\theta}_0$. This does not require that the full second derivative matrix is invertible (a strong requirement when the parameter space is very high dimensional), since it only needs to solve one vector equation. The intuition is that the second derivative only needs to behave well in a direction transversal to $\Theta_0(c)$.

We acknowledge that we cannot prove that these assumptions hold in practice. It could happen, for instance, that $T_{\boldsymbol{\theta}_0}\Theta_0(c) = \Theta$. In this case, we are overparameterized to such an extent that the restriction $f_{\boldsymbol{\theta}}(X_0) = c$ has no effect on the likelihood, leading to a test statistic of 0, which clearly is not $\chi^2(1)$ distributed. Our resulting confidence interval, however, is still conservative in this case since we will not reject the null hypothesis when the test statistic is zero.

5.B Empirical Distribution of Test Statistic in Toy Experiment

It is not possible to explicitly verify the distribution of $T(c)$ since it is not possible to perform the constrained optimization that would be required to obtain it. However, we can evaluate the distribution of our approximation of the test statistic.

For the toy regression example, we know the true function and we can check the distribution of the test statistic under the null hypothesis. This corresponds to the perturbation at X_0 that goes through $f(X_0)$.

We simulated a new data set 100 times and for each data set trained a network and calculated the test statistic. We did this for four values of X_0 $(-0.9, -0.5, 0.5, 0.9)$. It is possible that the test statistic becomes negative, in which case we simply set it to zero. This corresponds to a test statistic that would result in exactly the same CIs. The negative values have no influence on the actual coverage of the interval since a hypothesis only gets rejected when the test statistic is too large. The results are given in Figure 5.12.

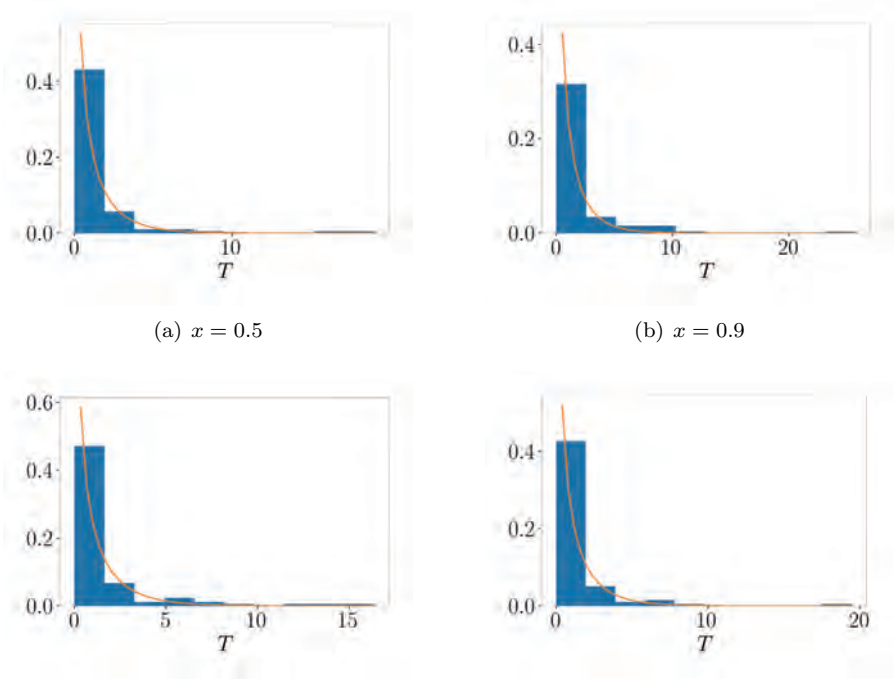
5.C Application of Methodology to XGBoost

While our chapter focuses on neural networks, the likelihood-ratio methodology presented can also be applied to other types of models. To illustrate this, we repeated the toy regression example from Section 5.3 with the popular XGBoost model (Chen and Guestrin, 2016).

XGBoost is a highly popular gradient boosting approach. Weak learners, typically tree models, are trained iteratively on the residuals of all previous trees while using a quadratic approximation of the loss function.

We used the default implementation of XGBoost from the `sklearn` API. XGBoost models can be regularized in a variety of ways; a common approach is through the λ and γ parameters. The λ parameter regularizes the size of the outputs of the trees and γ regularizes the number of splits. We constructed 95% CIs for two models: one model with a high amount of regularization ($\lambda = 5$ and $\gamma = 0.4$) and one with a lower amount of regularization ($\lambda = 2.5$ and $\gamma = 0.2$).

Since XGBoost does not have a random initialization and does not use batches,



(a) $x = 0.5$ (b) $x = 0.9$

(c) $x = -0.5$ (d) $x = -0.9$

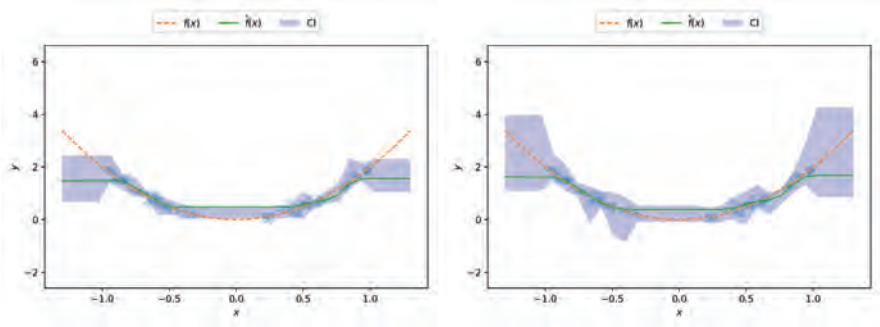
Figure 5.12: These histograms visualize the distribution of the test statistic during our toy regression experiment at four different locations. For each location, we simulated a new data set, trained a model, and calculated the test statistic 100 times. The orange line gives the probability density function of a $\chi^2(1)$ distribution.

we used a slightly different retraining approach. Instead of replacing the targets with the predictions of the model, we simply keep the original network and add the data point $(X_0, f_{\hat{\boldsymbol{\theta}}}(X_0))$. A new model is trained from scratch on this new data set.

Figure 5.13 visualizes the 95% CIs for both models. The greater flexibility of the model with a lower amount of regularization is reflected in the CIs. The intervals are substantially wider than the intervals of the model with a larger amount of regularization. Additionally, we observe the same qualitative

behaviour as we saw with neural networks. The intervals expand in regions where the data is more sparse and can become asymmetric.

This example illustrates that our likelihood-ratio methodology can also be applied to other machine-learning models besides neural networks. It also shows that there are multiple possible approaches to find suitable perturbations. We used a slightly different construction, simply adding a single data point and training from scratch, and still observed the same qualitative behavior. We do not claim that the method we used to find perturbations in this work is the optimal one. Future work should focus on finding optimal perturbation strategies.



(a) High regularization

(b) Low regularization

Figure 5.13: This figure illustrates that the likelihood-ratio methodology can also be applied to other models, such as XGBoost. The resulting 95% CIs inherently take the amount of regularization into account. A model with a lower amount of regularization is more flexible and can therefore reach values more easily without affecting the likelihood of the training data. We also observe the same favorable behaviour as we saw with neural networks. The intervals get larger in regions where the data is more sparse and can become asymmetric in biased regions.

Chapter 6

Quantile Regression with XGBoost

This chapter is based on the preprint entitled *"Composite Quantile Regression With XGBoost Using the Novel Arctan Pinball Loss"* (Sluijterman et al., 2024c), which is currently under review. The chapter focuses on the data uncertainty in a regression setting. Contrary to the rest of this thesis, we do not use neural networks, but instead XGBoost models, explained later in this chapter. A popular approach to deal with data uncertainty in a regression setting is quantile regression. However, XGBoost has specific issues that make it difficult to use for quantile regression. In this chapter, we present solutions to this problem and evaluate the resulting model on data sets of the Dutch energy grid, provided by power-grid operator *Alliander*.

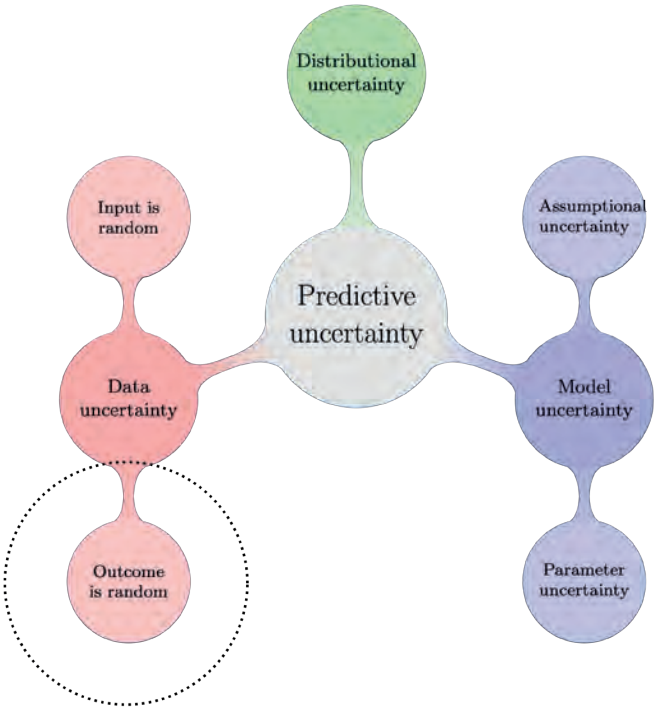


Figure 6.1: The scope of Chapter 6. This chapter focusses on quantile regression using XGBoost models. Quantile regression is a popular approach to quantify the data uncertainty.

6.1 Introduction

Extreme Gradient Boosting (XGBoost, Chen and Guestrin (2016)) is a powerful, open-source software library renowned for its performance in structured or tabular data sets across a wide range of domains, including finance (Gumus and Kiran, 2017; Nobre and Neves, 2019), healthcare (Ogunleye and Wang, 2020; Ramaneswaran et al., 2021; Li et al., 2019), and cybersecurity (Dhaliwal et al., 2018; Jiang et al., 2020). XGBoost is increasingly being used for safety-critical applications, such as predicting floods (Ma et al., 2021).

For these safety-critical applications, it is typically insufficient to rely solely

on predicting a point estimate. When predicting the water level in a river, understanding the potential extreme values is more important than predicting averages. Quantile regression (Koenker and Bassett Jr, 1978) offers an attractive solution. Instead of merely predicting a point estimate, various quantiles are predicted, for instance including the 0.95-quantile of the water levels.

Quantile regression has been carried out with a large variety of models. While originally mainly linear models were used, modern implementations of quantile regression often leverage complex models such as random forest (Meinshausen, 2006) and neural networks (Hatalis et al., 2019). These implementations may also predict multiple quantiles with a single model (Xu et al., 2017), typically referred to as composite quantile regression.

A large advantage of quantile regression is that it makes no distributional assumptions. Many uncertainty estimation methods will typically assume a Gaussian distribution (Lakshminarayanan et al., 2017; Nix and Weigend, 1994; Gal and Ghahramani, 2016), which could lead to subpar prediction intervals if the data is not normally distributed. Given the large popularity of XGBoost and quantile regression, there is a clear appeal to use XGBoost for this task.

Unfortunately, using XGBoost for quantile regression is nontrivial. At its heart, the model uses a quadratic approximation of the loss function during the optimization. However, as we will discuss in more detail later, the objective that is typically used for quantile regression, the pinball loss, is not differentiable everywhere and has a second derivative of zero, which makes this second-order approximation impossible.

Several solutions have been developed. The current implementation in the XGBoost package uses a different type of trees, additive trees, that do not require the second derivative during the optimization. However, this requires the use of separate models for each quantile. This is undesirable both as this can easily result in a very high number of quantile crossings – for example, the 0.45-quantile being larger than the 0.55-quantile – and because it is inefficient (Zou and Yuan, 2008). Another option is to use the regular XGBoost model but with a smooth approximation of the pinball loss that is differentiable everywhere.

While various of these approximations have been used for neural networks (Hatalis et al., 2019; Zheng, 2011; Xu et al., 2017), these approximations typically have a second derivative that is either zero or becomes extremely small. These approximations are unsuitable for XGBoost given its reliance on the second-order approximation of the loss function.

In this chapter, we therefore present a novel smooth approximation, named

the arctan pinball loss, specifically tailored for XGBoost. Crucially, the loss function is differentiable everywhere and has a much larger second derivative than the existing alternatives, making it more suitable for XGBoost. This allows the use of a single model for multiple quantiles, resulting in far fewer crossings and an increased efficiency.

Our chapter is organized as follows. Section 6.2 contains all relevant technical details on XGBoost and quantile regression. Additionally, the existing smooth approximations are discussed. The arctan pinball loss is presented in Section 6.3. In Section 6.4, our implementation of quantile regression with XGBoost is compared to the current implementation. Crucially, our approach has significantly fewer crossings while achieving similar or superior coverage. Final concluding remarks can be found in Section 6.5.

6.2 Background and Related Work

This section consists of three parts. We first provide the details on XGBoost that are necessary for this chapter. We then discuss quantile regression and explain why it is non-trivial to use XGBoost for this task. The third subsection provides current solutions for this problem along with the shortcomings of those solutions.

6.2.1 XGBoost

We provide an introduction to XGBoost at the minimal level that is required for this chapter. For a more in-depth introduction, we refer to Chen and Guestrin (2016), whose notation we have followed here.

XGBoost is a boosting approach (Schapire, 1990) that iteratively trains weak learners, typically tree models, while employing both the first and second derivative of the loss function, hence the name Extreme Gradient.

The eventual output of the model is the sum of the outputs of the K trees:

$$\hat{y}_i = \phi(\boldsymbol{x}_i) = f_0(\boldsymbol{x}_i) + \eta \sum_{k=1}^{K} f_k(\boldsymbol{x}_i), \tag{6.1}$$

where $f_0(\boldsymbol{x}_i)$ is the base score, η is the learning rate, and f_k is a tree with tree structure q_k, a function that maps the inputs to a leaf index, and weights-vector

$\boldsymbol{\omega}_k$, a vector containing the weights of each leaf.

XGBoost iteratively trains the trees with the goal to predict the remaining residual. These individual trees are trained by optimizing a regularized objective:

$$\mathcal{L}(\phi) = \sum_i l(y_i, \hat{y}_i) + \sum_k \Omega(f_k), \quad \text{with } \Omega(f_k) = \gamma T_k + \frac{1}{2}\lambda ||\boldsymbol{\omega}_k||^2. \tag{6.2}$$

The loss function, l, measures the difference between the outputs, $\hat{y}$, and the observations, y. The output could be an estimate of y but it could also be a conditional quantile. The regularization term Ω favors simpler trees with a smaller number of leaves, T_k, and smaller weights.

The model is trained iteratively, one tree at a time. Each tree aims to learn the residual from all the previous trees. Let $\hat{y}_i^{(t)}$ be the i-th prediction after having trained the first t trees. During the training of tree t, the following objective is optimized:

$$\mathcal{L}^{(t)} = \sum_i l(y_i, \hat{y}_i^{(t-1)} + f_t(\boldsymbol{x}_i)) + \Omega(f_t). \tag{6.3}$$

XGBoost uses a quadratic approximation of Equation (6.3) during the optimization:

$$\mathcal{L}^{(t)} \approx \sum_i [l(y_i, \hat{y}_i^{(t-1)}) + f_t(\boldsymbol{x}_i) g_i + \frac{1}{2} f_t^2(\boldsymbol{x}_i) h_i] + \Omega(f_t), \tag{6.4}$$

where $g_i = \frac{\partial l(y_i, \hat{y}_i^{(t-1)})}{\partial \hat{y}_i^{(t-1)}}$, and $h_i = \frac{\partial^2 l(y_i, \hat{y}_i^{(t-1)})}{\partial (\hat{y}_i^{(t-1)})^2}$. Using this equation, the optimal weight of leaf j of tree t can be calculated:

$$\omega_{tj}^* = -\frac{\sum_{i \in I_{tj}} g_i}{\sum_{i \in I_{tj}} h_i + \lambda}, \tag{6.5}$$

where $I_{tj} = \{i | q_t(\boldsymbol{x}_i) = j\}$, are the indices of the data points that end up in leaf j.

By using Equation (6.5), the approximate loss function in Equation (6.4) can be calculated for a specific tree structure. An efficient split-finding algorithm is used to find the optimal tree structure.

XGBoost further distinguishes itself through several key features that enhance its performance and versatility in machine learning tasks. Firstly, it employs a highly efficient split finding algorithm that optimizes the selection of split points

in trees, significantly speeding up the learning process. Secondly, XGBoost has excellent parallelization capabilities, allowing it to utilize multiple cores during the training phase, which greatly reduces the time required to build models. Furthermore, it is adept at handling missing values in the data set. XGBoost automatically learns the best direction to assign missing values during the split, either to the left or right child, depending on which choice leads to the best gain. This ability to deal with incomplete data directly, without needing imputation or dropping rows, makes XGBoost a robust and flexible tool for a wide array of data science and machine learning applications.

Depending on the specific data set, XGBoost can be a good candidate due to these advantages. Depending on the specific task, it may be desirable to obtain quantiles via quantile regression. Given a specific data set and task, e.g., a large tabular data set with the goal to predict water levels, we may therefore ideally want to perform quantile regression using XGBoost. As we will discuss later, however, this is far from trivial.

6.2.2 Quantile Regression

Quantile regression aims to predict a specific quantile of a probability distribution rather than, for instance, predicting the mean. A conditional quantile is defined as:

$$q_\tau(\boldsymbol{x}) := \min\{y | F_{Y|X=\boldsymbol{x}}(y) \geq \tau\}.$$

In other words, it is the smallest value y, such that the probability that Y is smaller than y, given $X = \boldsymbol{x}$, is at least τ. Koenker and Bassett Jr (1978) showed that conditional quantiles can be estimated by minimizing the so-called pinball loss:

$$L_\tau(y_i, \hat{y}_i) = \tau(y_i - \hat{y}_i)\mathbb{I}_{\{\hat{y}_i \leq y_i\}} + (\tau - 1)(y_i - \hat{y}_i)\mathbb{I}_{\{\hat{y}_i > y_i\}}, \tag{6.6}$$

where $\hat{y}_i$ is the predicted quantile, y_i is the observed value, and $\mathbb{I}$ is the indicator function. This loss is also known as the check function, tick function, or quantile loss. The pinball loss is visualized in Figure 6.2 for two different values of τ. For the 0.9-quantile, estimating a quantile smaller than the observation is penalized more than estimating a quantile that is too large[1].

[1]If we predict a single quantile (that does not depend on any input), the pinball loss is minimized when the quantile is such that the number of observations larger than the predicted quantile is equal to $(1 - \tau)n$. This can be seen by differentiating $L_\tau(\hat{y}) = \sum_{i:\hat{y} \leq y_i} \tau(y_i - \hat{y}) + \sum_{i:\hat{y} > y_i} (\tau - 1)(y - \hat{y})$ with respect to $\hat{y}$ and setting it to zero.

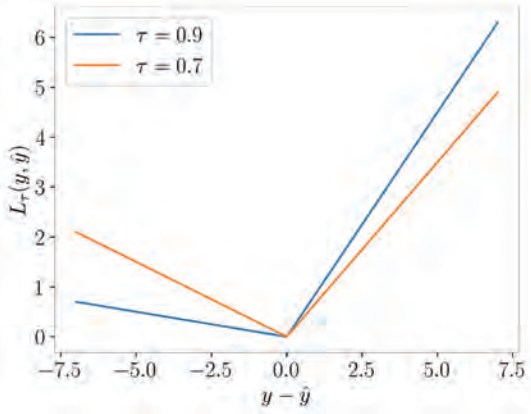


Figure 6.2: The pinball loss for two different values of τ. An observation above the predicted 0.9-quantile leads to a much larger loss than an observation below.

Note that the pinball loss only depends on $y_i - \hat{y}_i$. In the remainder of this chapter, we will therefore use the notation $u := y - \hat{y}$ and present the loss functions in terms of u.

The original paper from Koenker and Bassett Jr (1978) sparked a large amount of further research which developed into an entire subfield within statistics and econometrics. Non- and semiparametric versions were developed (Chaudhuri, 1991; Lee, 2003) as well as extensions to simultaneously predict multiple quantiles (Zou and Yuan, 2008). Adaptions were constructed for time series (Koenker and Xiao, 2006; Chen et al., 2009), causal inference (Chernozhukov and Hansen, 2006), and dealing with censored data (Powell, 1986; Yang et al., 2018). Simultaneously, advances were made on the theory of variable selection and regularization (Belloni and Chernozhukov, 2011). We refer the interested reader to a survey by Koenker (2017) that provides an overview of the important advances in the four decades following the first seminal work.

More recently, there has been a growing interest to perform quantile regression using machine learning models (Hatalis et al., 2019; Zheng, 2011; Xu et al., 2017; Meinshausen, 2006). As we discuss now, using XGBoost to perform quantile regression comes with specific challenges.

6.2.3 Challenges and previous solutions

Ideally, we would want to let XGBoost output the quantiles and use the pinball loss function directly. However, this is not possible for two reasons. First of all, the loss function is not differentiable at $u = 0$. Secondly, the second derivative is zero everywhere. This is problematic for XGBoost since it uses a second order approximation of the loss function during the optimization.

The current solution in the XGBoost package is to use additive trees. These are slightly different trees that do not require the second derivative but rely on an adapted training algorithm that uses line searches. However, using these modified trees requires a separate model for each quantile, which is highly inefficient.

The problem of the differentiability at $u = 0$ can also be overcome by using a smooth approximation of the loss function. Multiple different smooth approximations have been suggested for neural networks.

Hatalis et al. (2019) use the following smooth approximation based on work from Zheng (2011):

$$L_{\tau,s}^{(\exp)}(u) = \tau u + s \log\left(1 + \exp(-u/s)\right), \tag{6.7}$$

where s is a smoothing parameter that determines the amount of smoothing. A smaller value gives a closer approximation to the true pinball loss.

Cannon (2011) and Xu et al. (2017) use the Huber norm to approximate the pinball loss. The Huber norm is given by:

$$n_\delta(u) = \begin{cases} \frac{1}{2}u^2 & \text{for } |u| \leq \delta, \\ |u| - \frac{1}{2}\delta & \text{otherwise.} \end{cases} \tag{6.8}$$

The resulting approximation of the pinball loss is given by:

$$L_{\tau,\delta}^{(\text{Huber})}(u) = \tau n_\delta(u)\mathbb{I}_{\{u>0\}} + (1-\tau)n_\delta(u)\mathbb{I}_{\{u<0\}}. \tag{6.9}$$

This Huber pinball loss has also been applied to XGBoost (Yin et al., 2023). However, since the second derivative of the Huber pinball loss is still zero for $|u| > \delta$, the algorithm requires a large value of λ to properly converge in practice. Being forced to use a large λ is undesirable. The second derivative becomes obsolete and the training in practice reduces to gradient descent with a very low learning rate. This can be seen by evaluating Equation (6.5) for a large value of λ.

To really benefit from the convergence speed caused by the quadratic approximation used by XGBoost, it is essential to use an approximation of the pinball loss that is not only differentiable at zero but also has a non-zero second derivative everywhere. The exponential approximation, $L_{\tau,s}^{(\exp)}(u)$, may therefore seem like a suitable candidate. In fact, the second derivative is strictly positive:

$$\frac{\partial^2 L_{\tau,s}^{(\exp)}(u)}{\partial u^2} = \left(\exp(-\frac{u}{2s}) + \exp(\frac{u}{2s})\right)^{-2} \frac{1}{s}. \tag{6.10}$$

However, when implementing this, we ran into similar problems. Although the second derivative is always positive, it decays exponentially as a function of $|u|$, resulting in a vanishing second derivative. This caused extreme updates and overflow errors, even in simple examples. The only solution was to use a large value of λ or a large value of s. As explained before, being forced to use a large value of λ is undesirable. Similarly, using a large value of s is undesirable as this leads to a very rough approximation of the loss function and overly wide intervals. We elaborate on this last point later. In summary, we need to find a smooth approximation with a reasonably large second derivative.

6.3 The Arctan Pinball Loss

Our goal is to develop a smooth approximation of the pinball loss function that maintains a large second derivative. To achieve this, we introduce the following approximation, named the arctan pinball loss:

$$L_{\tau,s}^{(\arctan)}(u) = \left(\tau - 0.5 + \frac{\arctan(u/s)}{\pi}\right) u + \frac{s}{\pi}, \tag{6.11}$$

where s is a smoothing parameter that controls the amount of smoothing. A smaller value of s results in a closer approximation but, as we will soon see, also a smaller second derivative. The s/π term ensures that the approximation is unbiased for large values of $|u|$. For XGBoost, this term is purely aesthetic, as it does not influence the first or second derivative. However, for other applications or optimisation procedures, it could be useful. We provide more details on the construction and the unbiasedness in Appendix 6.A.

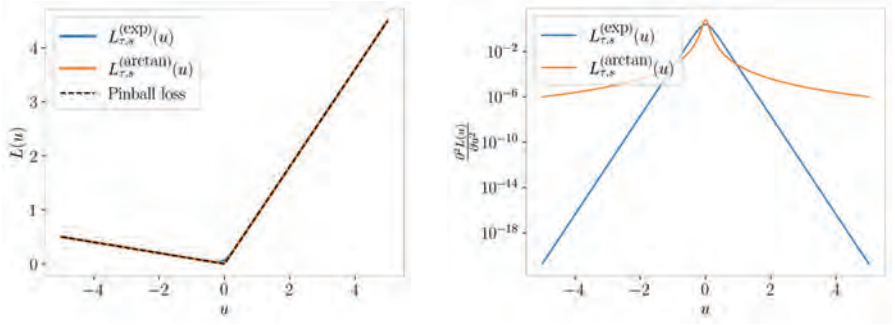


Figure 6.3: A comparison of $L_{\tau,s}^{(\exp)}(u)$ and $L_{\tau,s}^{(\arctan)}(u)$ for $\tau = 0.9$ and $s = 0.1$. Both the exponential approximation and the arctan approximation approximate the pinball loss very closely. However, as is displayed in (b), the second derivative of the arctan pinball loss is much larger.

The second derivative of the arctan pinball loss is given by:

$$\begin{aligned}\frac{\partial^2 L_{\tau,s}^{(\arctan)}(u)}{\partial u^2} &= \frac{2}{\pi s}\left(1+(u/s)^2\right)^{-1} - \frac{2u^2}{\pi s^3}\left(1+(u/s)^2\right)^{-2} \\ &= \frac{2}{\pi s}(1+(u/s)^2)^{-2}.\end{aligned}$$

Crucially, this second derivative is strictly positive and falls off polynomially as opposed to the exponential decay of $\frac{\partial^2 L_{\tau,s}^{(\exp)}(u)}{\partial u^2}$.

Figure 6.3 visualizes this difference in second derivative. Figure 6.3(a) shows that both the exponential pinball loss and the arctan pinball loss approximate the true pinball loss very well when using $s = 0.1$. However, as can be seen in Figure 6.3(b), the arctan pinball loss has a second derivative that is orders of magnitude larger, making it a much better candidate to use with XGBoost.

By using this loss function, we are able to carry out quantile regression while using the default version of XGBoost. One of the advantages of this is that we can predict multiple quantiles with the same model by using multi-output leaves. From a theoretical point of view, using the same model for multiple quantiles is advantageous. The different quantiles can share information, making it more efficient than estimating all the quantiles with separate models (Zou and Yuan, 2008).

A second advantage of using the same model for different quantiles is that all these quantiles share the same splits. This makes it much less likely that quantiles cross. As we will see in the Section 6.4, using separate models for each

quantile results in many more quantiles crossings, which is clearly undesirable.

However, even when using a single model, crossings cannot be entirely prevented. Due to the quadratic approximation, a single update can still result in a crossing. Three scenarios where crossings could occur during an update are visualized in Figure 6.4.

For simplicity, we consider the scenario where there is only a single data point in a leaf. Suppose we predict the 0.95-quantile (red) and the 0.85-quantile (blue). Without any regularization, $\lambda = 0$, the update for both quantiles is proportional to the gradient divided by the second derivative (Equation (6.5)).

In situation 1, the 0.95-quantile is slightly larger than y and the 0.85-quantile substantially smaller than y. The update is proportional to the gradient divided by the second derivative. In the first situation, the gradient for the 0.95-quantile is smaller than for the 0.85-quantile and the second derivative is larger. These resulting updates cause the 0.85-quantile to become substantially bigger and the 0.95-quantile to become slightly smaller. This could result in a crossing.

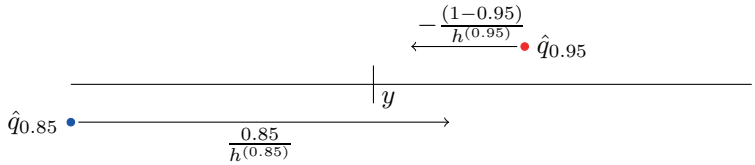
In situation 2, the 0.85-quantile is smaller than y and the 0.95-quantile is larger than y by a similar amount. In this case, the second derivatives for both are equal. However, the gradient for the 0.85 quantile is roughly 0.85 compared to -0.05 for the 0.95 quantile. This could also result in a crossing during this update.

In the final scenario, both quantiles are larger than y. The gradient of the 0.95-quantile is -0.05 and the gradient for the 0.85-quantile is -0.15. At first glance this should not be able to result in a crossing. However, since the second derivative for $\hat{q}_{0.95}$ is smaller, this is still a possibility.

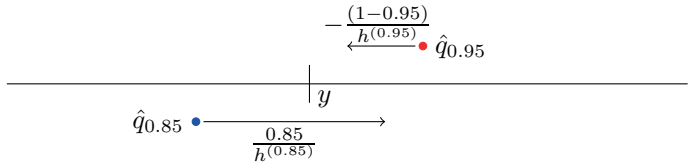
Note that two of the three crossing scenarios were caused by a difference in second derivative. Since the second derivative of our arctan pinball loss is polynomial instead of exponential, we do not suffer from this effect as much. Additionally, using a larger λ would also diminish this effect.

In general, using any approximation of the true loss can result in a slightly biased model. Figure 6.5 illustrates the bias that both approximations of the pinball loss, $L_{\tau,s}^{(\exp)}(u)$ and $L_{\tau,s}^{(\arctan)}(u)$, have near the origin. The optimum for both losses is slightly below $u = 0$ when using a τ larger than 0.5. This causes the predicted quantiles to be slightly larger. This would result in slightly more conservative prediction intervals, especially when using larger values of s. We will observe this behaviour in Section 6.4.

Situation 1: Different second derivative and gradient



Situation 2: Different gradient



Situation 3: Different second derivative

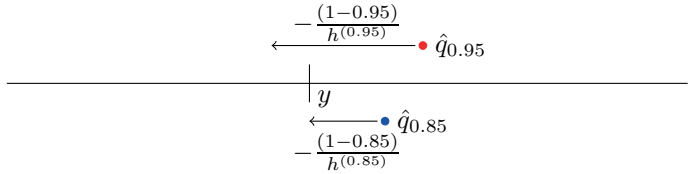


Figure 6.4: Three scenarios where crossings can occur, not at scale. While the optimum of the arctan pinball loss has no crossings, individual updates can result in crossings due to the quadratic approximation of the loss function. The resulting update is proportional to the gradient divided by the second derivative, denoted with h. These second derivatives in particular can vary greatly in size, leading to updates that can result in crossings.

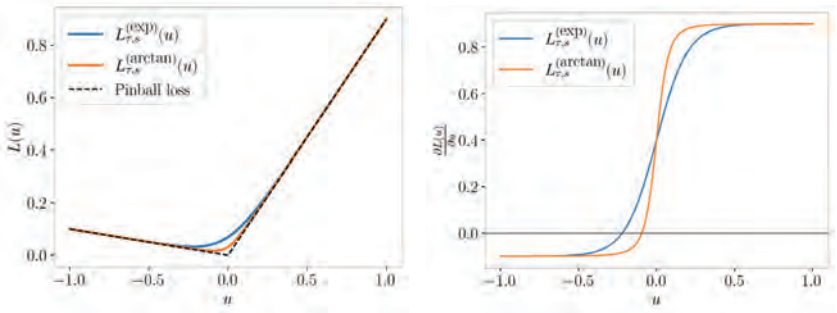


Figure 6.5: A comparison of $L_{\tau,s}^{(\exp)}(u)$ and $L_{\tau,s}^{(\arctan)}(u)$ for $\tau = 0.9$ and $s = 0.1$ at a very small scale. Near the origin, the bias in both approximations is clear. Both approximations have the actual minimum of the loss function slightly below $u = 0$. This results in slightly more conservative quantiles, meaning larger quantiles for $\tau > 0.5$ and smaller quantiles for $\tau < 0.5$ compared to when using the regular pinball loss. This effect is larger when using a larger value of s.

For optimal use of the arctan pinball loss, we recommend the following modeling choices.

1. Always use standardized targets. This allows us to keep certain hyper-parameters, most notably the smoothing parameter s, fixed regardless of the data set. We typically found values between 0.05 and 0.1 to work well. Smaller values result in extremely small second derivatives, and much larger values result in an approximation that is too rough, leading to overly conservative prediction intervals.

2. Set the min-child-weight parameter to zero. This parameter regularizes the trees by requiring a minimum weight in each leaf in order to allow a split. The weight is defined as the sum of the second derivatives of the points in the resulting leaf. This makes sense when using a loss function with a constant second derivative, such as the mean-squared error. In that case, this parameter enforces a minimum number of data points in each leaf to prevent overfitting. However, since the second derivative of our loss function is far from constant, we advise to not use this parameter and set it to zero.

3. Use a slightly smaller learning rate of 0.05 (compared to 0.1 in the stan-

dard implementation). The weights of the new tree are given by Equation (6.5). The outputs of the new tree are multiplied by the learning rate to obtain the actual update. Since the second derivative can be substantially smaller than 1, it is still possible to obtain rather large updates. To make this more stable, we advise to use a slightly smaller learning rate.

4. Set the max-delta parameter to 0.5. This is done for the same reason as the slightly lower learning rate. To prevent overly large updates, this parameter is set to 0.5. During our experiments, we observed no negative effects of using this parameter in terms of coverage or validation loss but it reduced the number of quantile crossings.

6.4 Experimental Results

This section consists of three parts. We first go through the various data sets that were used. Subsequently, we explain our experimental design. This includes the choices of hyper-parameters, the optimization procedure, and the metrics that were used. Finally, the results are given and discussed in the third subsection.

6.4.1 Data Sets

Toy example Our first example is a one-dimensional toy example. This experiment demonstrates the qualitative advantages of our approach. Specifically, we illustrate that the splits for the different quantiles are all located at the same positions, significantly reducing the number of quantile crossings.

The training set consists of 1,000 realizations of the random variable pair (X, Y), where $X \sim U[0, 1]$ and $Y \mid X = x \sim \mathcal{N}(\sin(7x), 0.2^2)$.

UCI Benchmark Data sets Secondly, we examine our method on six publicly available UCI regression data sets: Boston housing, energy, concrete, wine quality, yacht, and kin8nm. These data sets range from a few hundred data points, with yacht being the smallest at 308, to several thousands, kin8nm having over 8,000 data points. The data sets feature between 6 and 13 covariates, encompassing both continuous and categorical variables. Given their high dimensionality and the wide range of tasks they represent, these data

sets are frequently used as benchmark data sets in machine learning research (Hernández-Lobato and Adams, 2015).

Electricity-grid Substations Lastly, we examine the total load on four distinct substations from the Dutch electricity grid. For each substation, three months of data at a temporal resolution of 15 minutes is available. The objective is to predict the load on the substation one day ahead using the 81 available covariates. These covariates comprise a mix of measurements, predictions, and categorical values. Examples include load measurements from the previous day, day-ahead electricity price, predicted amounts of solar radiation and windspeed for the next day, and calendar-derived variables such as whether the day is a weekday or a holiday. These data sets have been provided to us by the distribution system operator Alliander and are publicly available as part of the OpenSTEF package: `https://github.com/OpenSTEF/openstef-offline-example/tree/master/examples/data`.

6.4.2 Experimental Design

For all experiments, we predict 10 different quantiles:

$$[0.05, 0.15, 0.25, \ldots, 0.75, 0.85, 0.95].$$

The following hyper-parameters are optimized:

- The number of estimators: [100, 200, 400].
- The λ regularization parameter: [0.01, 0.1, 0.25, 0.5, 1, 2.5, 5, 10].
- The γ regularization parameter: [0.1, 0.25, 0.5, 1, 2.5, 5, 10].
- The maximum depth of the trees: [2, 3, 4].

For the toy example, we applied 3-fold cross-validation (using the average pinball loss as the criterion) to determine the optimal hyper-parameters and evaluated the resulting model on a separate test set.

For the UCI data sets, we used 3-fold cross-validation to obtain predicted quantiles for every data point in the data set. During each cross-validation, another round of 3-fold cross-validation was used to determine the optimal hyper-parameters.

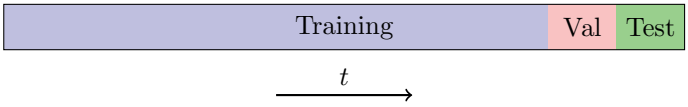


Figure 6.6: Illustration of the train/val/test split used for the Alliander data.

Since the substation data sets are time series, we could not use regular cross-validation. Instead, we used a train/validation/test split where we allocated the first 80% of the time series as the training set, the next 10% as the validation set, and the final 10% as the test set. The optimal hyper-parameters were determined using the validation set, and the actual model was fitted using these parameters on the combined training and validation set. Subsequently, the model was evaluated on the test set. This procedure is visualized in Figure 6.6.

Metrics For the toy-example, which is mainly illustrative, we provide visualizations of the various quantiles. For the UCI data set and the electricity-grid substation data sets, we provide the following quantitative metrics:

1. The marginal coverage percentage and average width of the 90% PI:

$$\text{Coverage} = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}_{\{y_i \in \text{PI}(\boldsymbol{x}_i)\}} \cdot 100\%, \tag{6.12}$$

where $\text{PI}(\boldsymbol{x}_i)$ is the 90% PI that is constructed using the predicted 0.05- and 0.95-quantile. We also report the average width of this interval.

The marginal coverage of an interval, however, does not fully capture the quality of the predicted conditional quantiles. The typical argument is that we want an interval that has the correct marginal coverage while being as narrow, or sharp, as possible (Kuleshov et al., 2018). A similar argument has been made in terms of calibration and refinement (DeGroot and Fienberg, 1983) for a probabilistic classifier. This argument translates well to quantile regression.

Suppose we are predicting a conditional τ-quantile, denoted with $\hat{y}(X)$. The perfect predicted quantile would satisfy:

$$\mathbb{P}\left(Y < \hat{y}(X) \mid X = \boldsymbol{x}\right) = \tau \quad \forall \boldsymbol{x}.$$

The predicted quantile is never perfect and we therefore make the following errors:

$$\begin{aligned}\mathbb{P}\left(Y < \hat{y}(X) \mid X = \boldsymbol{x}\right) = \tau &+ \left(\mathbb{P}\left(Y < \hat{y}(X)\right) - \tau\right) \\ &+ \left(\mathbb{P}\left(Y < \hat{y}(X) \mid X = \boldsymbol{x}\right) - \mathbb{P}\left(Y < \hat{y}(X)\right)\right).\end{aligned} \tag{6.13}$$

The error in the first line of Equation (6.13), $\mathbb{P}\left(Y < \hat{y}(X)\right) - \tau$, is the calibration error. Crucially, this error can be low by having conditional quantiles that are only correct on average and not for individual values of $\boldsymbol{x}$. This can be seen by noting that:

$$\mathbb{P}\left(Y < \hat{y}(X)\right) = \int_{\mathcal{X}} \mathbb{P}\left(Y < \hat{y}(X) \mid X = \boldsymbol{x}\right)\pi(\boldsymbol{x})d\boldsymbol{x},$$

where $\pi(\boldsymbol{x})$ is the density function of the random variable X.

The error term in the second line of Equation (6.13) is the refinement error. This term is large if the coverage of the conditional quantile is substantially larger or smaller than the marginal coverage.

As an example, the empirical CDF would be relatively well-calibrated but would not be practical as it entirely ignores the covariates. A similar point is made by Kuleshov et al. (2018).

2. The average pinball loss:

Because of the limitation of only reporting the marginal coverage, we also report the average pinball loss:

$$\text{Average pinball loss} = \frac{1}{n_\tau n}\sum_{i=1}^{n}\sum_{j=1}^{n_\tau} L_{\tau_j}(y_i, \hat{y}_{ij}),$$

where L_{τ_j} is the pinball loss for quantile τ_j, n_τ is the number of predicted quantiles, n is the number of data points, and $\hat{y}_{ij}$ is the j-th predicted quantile of data point i. The pinball loss is a proper scoring rule for conditional quantiles (Gneiting and Raftery, 2007) and therefore measures both the calibration error and the refinement error.

3. The crossing percentage, which is the percentage of adjacent predicted quantiles that cross:

$$\text{Crossing percentage} = \frac{1}{(n_\tau - 1)n} \sum_{i=1}^{n} \sum_{j=1}^{n_\tau - 1} \mathbb{I}_{\{\hat{y}_{ij} > \hat{y}_{i(j+1)}\}} \cdot 100\%.$$

6.4.3 Results and Discussion

Toy example Figure 6.7 illustrates the difference between our approach and the default implementation of quantile regression in XGBoost. The default implementation uses a separate model for each quantile. This causes the splits to be at different locations, easily resulting in quantile crossings. On the contrary, our approach uses a single model for all ten quantiles and therefore has the splits at the same locations. In this example, our approach had 0 crosses.

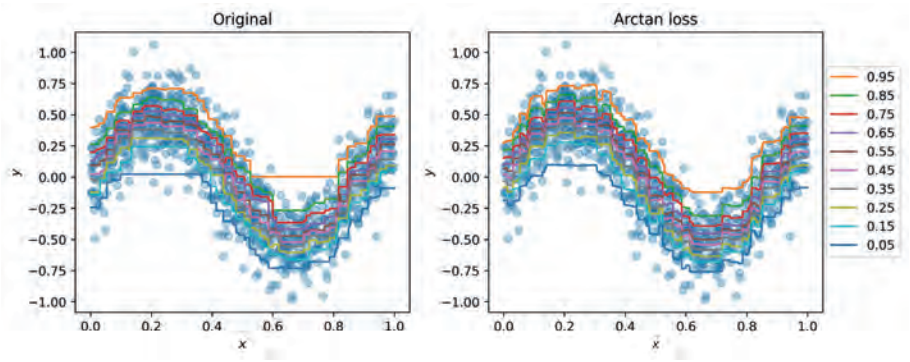


Figure 6.7: A comparison of the original implementation and the arctan loss. By using the arctan loss function, a multi-output tree can be used in XGBoost. This causes the splits to be at the same locations for all quantiles, greatly reducing the number of crossings.

UCI data sets The results on the six UCI data sets are given in Table 6.1. We evaluated two values of s, our smoothing parameter. A larger value means more smoothing.

We observe far fewer crossings for all six data sets. Additionally, while our intervals are typically smaller, our marginal coverage is overall closer to the desired 90%, with the exception of the energy data set. We do not see a

Table 6.1: Results on UCI benchmark data sets. The marginal coverage and the average width are calculated for the 90% PIs that are constructed with the 0.05- and the 0.95 quantiles. Using the arctan pinball loss results in significantly fewer quantile crossings while achieving comparable performance. On 5 of the 6 data sets, energy being the exception, we observe superior coverage and an equal or better pinball loss. We also see the effect that using a larger value of s results in more conservative quantiles and thus wider prediction intervals.

Data set	**Average Pinball loss**			90% PI **coverage**			90% **PI width**			**Crossing percentage**		
	s=0.05	s=0.1	default	s=0.05	s=0.1	default	s=0.05	s=0.1	default	s=0.05	s=0.1	default
Energy	0.22	0.22	0.17	93.9	97.1	88.9	5.4	5.8	4.5	7.1	0.3	20.2
Concrete	1.5	1.4	1.5	81.4	86.1	81.3	15.2	15.8	19.8	5.5	3.2	16.1
Kin8nm	0.040	0.040	0.041	83.1	84.3	82.3	0.44	0.44	0.46	1.1	0.6	11.0
Boston Housing	0.92	0.91	0.95	80.0	83.2	76.1	8.0	8.7	9.9	2.4	0.7	18.5
Yacht	0.21	0.26	0.21	90.9	95.5	69.2	2.8	4.9	5.5	0.5	0.0	29.8
Wine	0.17	0.17	0.17	88.1	88.7	61.2	1.5	1.7	1.8	3.1	2.2	5.3

clear difference in performance in terms of the pinball loss. This illustrates the previously mentioned fact that the coverage is a marginal coverage. A model can be very well calibrated, but not very informative, or it can be very informative yet poorly calibrated.

An insightful way to visualize the marginal coverage is via reliability diagrams (Murphy and Winkler, 1977; Niculescu-Mizil and Caruana, 2005). For each quantile, for instance the 0.05-quantile, the fraction of observations below the corresponding predicted quantiles is plotted. Perfect calibration would result in a diagonal line, a biased model would result in a line above or below the diagonal, and over- or underconfident models result in an (inverted) s-curve.

Figure 6.8 provides the reliability diagrams for both approaches on the energy and wine data sets. The left plot illustrates why the pinball loss of our new approach was worse on the energy data set. We observe that practically all the intervals are overly conservative. This is likely caused by using an s-value that was slightly too large. As we previously argued, this can cause overly conservative quantiles. This is further confirmed by the fact the intervals were even wider when using $s = 0.1$. Additionally, the right plot clearly shows that the extremely low marginal coverage on the wine data set by the original approach was caused by the 0.05-quantiles being too large.

At first glance, the performance of the original approach on the wine data set looks rather dramatic. The 90% PI of the original approach only has a 61.2% marginal coverage. The reliability diagram already revealed that this is caused by the 0.05-quantile that is too large. However, the average pinball loss shows

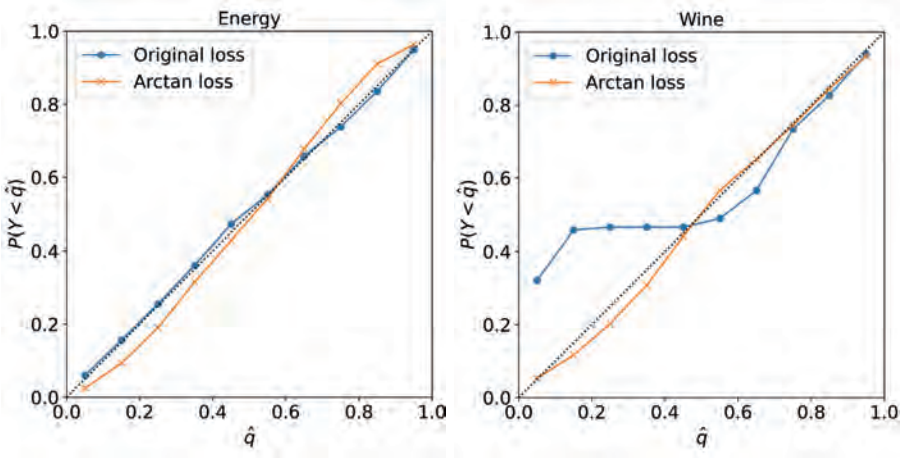


Figure 6.8: Reliability diagrams for the quantiles obtained for the energy (left) and wine (right) data sets using the arctan loss with $s = 0.05$ (orange) or the regular pinball loss (blue). For the wine data set, the lower quantiles of the original approach are calibrated very poorly. For the energy data sets, all the quantiles are too conservative, meaning that the lower ones are too small and the larger ones are too large. This leads to intervals that are slightly too large and a higher pinball loss.

that the actual model is not that much worse than our implementation. When investigating this further, we found that the original 0.05-quantiles were very slightly, but consistently, too large. This resulted in a very low coverage of the 90% PIs even though the intervals were in fact only very slightly too small.

Figure 6.9 illustrates this effect. The observations are plotted against the differences between the observations and the predicted quantiles. A positive value corresponds to an observation above the predicted quantile. For the 0.05-quantiles, 95% of observations should be above the quantiles. As we observe, this is only 67.9%. However, the figure illustrates that this is caused by a substantial number of observations that are only very slightly larger than these quantiles. In fact, when reducing the predicted quantiles by just 1%, the percentage of observations above the corresponding 0.05-quantiles rose to 96.9%.

While the marginal coverage is often closer to 90% with the arctan loss, we also have a number of data sets where the intervals are too narrow, especially when using a smaller s. As mentioned, the original implementation even had a marginal coverage as low as 61.2% for one of the data set. This overconfidence

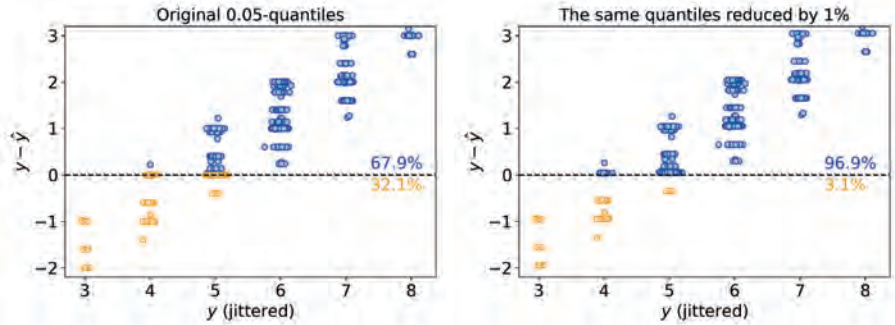


Figure 6.9: Differences between observations and 0.05-quantiles of the default implementation on the wine data set. The values on x-axis are jittered slightly for better visibility. No jitter was used for y-axis. Ideally, 5% of the y-values should be smaller than the predicted quantiles. For the original approach, this was 32.1% of observations (denoted in orange). However, reducing the predicted quantiles by a mere 1% caused the percentage of points below the predicted quantiles to fall to just over 3%. This effect was particularly strong on this data set since the wine scores are not continuous.

is in line with the observation of Guo et al. (2017) who noted that modern machine learning models are often overconfident. The pinball loss depends on both calibration and refinement and therefore the resulting optimal model according to the pinball loss may not be the best calibrated model.

A general approach to improve the calibration is to add a post-hoc calibration step. The PIs are evaluated on a previously unseen part of the data set and are tuned such that they are better calibrated. We advise to always consider using such a post-hoc calibration step when implementing these models in practice. An example of such a procedure can be found in Romano et al. (2019).

Electricity-grid substations The results for the electricity substations are given in Table 6.2. Our approach yields comparable results while having far fewer crossing and requiring only a single model. For two of the four substations, we have a slightly better pinball loss while for two others, we perform slightly worse. A similar pattern is observed for the coverage.

We also observed that the models are sometimes biased, rather than overconfident, causing a subpar coverage. We suspect that this is caused by the varying conditions combined with the fact that trees do not extrapolate well.

Table 6.2: Results on Alliander data sets. We obtain comparable performance with only a single model and have almost zero quantile crossings. The coverage is lower for both approaches, an effect we expect to be the result of the inability of tree based models to extrapolate well.

Data set	**Average Pinball loss**		90% PI **coverage**		90% **PI width**		**Crossing percentage**	
	$s = 0.1$	default	$s = 0.1$	default	$s = 0.1$	default	$s = 0.1$	default
Substation 287	0.19	0.18	80.6	76.7	1.66	1.56	0.4	12.8
Substation 307	0.75	0.77	88.2	84.11	8.24	7.95	0.0	4.2
Substation 435	0.53	0.55	78.8	86.1	4.83	5.88	0.0	2.1
Substation 438	0.69	0.68	84.6	86.1	7.0	7.3	0.0	3.3

For substation 287, the training data (including validation) starts halfway through October and ends in early January. The entire test set consists of days in January. During that time, the loads in the substation were typically lower. Multiple factors might cause this, but a likely explanation is reduced sunlight in January.

Figure 6.10 provides reliability diagrams for both substations. The aforementioned bias of both models for substation 287 is clearly visible. All the quantiles are too large. For substation 307, we observe that both approaches yielded well-calibrated quantiles.

The 90% PIs for substations 287 and 307 are visualised for both approaches in Figure 6.11. For substation 287, we observe that the intervals fail to capture the lowest peaks. Additionally, both models exhibit a bias around the 24th of January. The predicted intervals are often above the actual loads in that region.

In this specific application, which involves a time series with only three months of training data, using tree-based models presents clear disadvantages. Inevitably, the model will encounter unseen scenarios, such as the first significantly sunny day or the first frost period in a three-month period. In these instances, the model may fail since the individual trees cannot extrapolate.

This illustrates that XGBoost may not always be the correct model. We stress that we do not claim this to be the optimal way to perform quantile regression. Other models, such as neural networks or even simple linear models, could work just as well, or better, depending on the specific situation. However, there may be situations where XGBoost is preferred due to its efficient training and its capability to handle missing data. In such cases, using the arctan pinball loss allows for the simultaneous estimation of multiple quantiles, resulting in

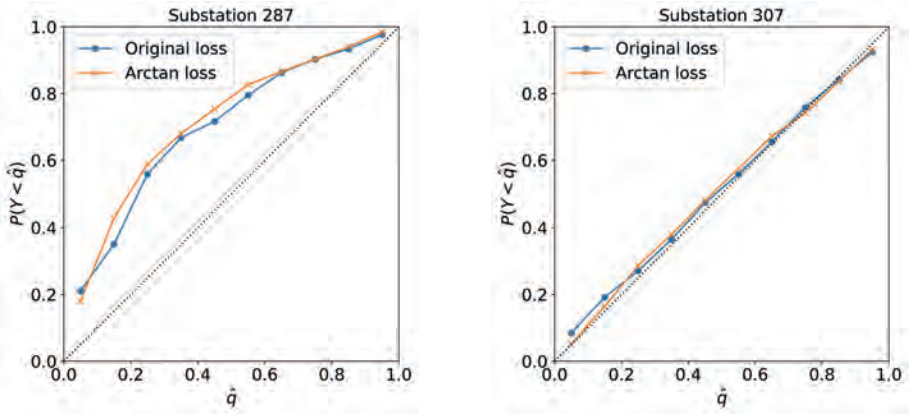


Figure 6.10: Reliability diagrams for the quantiles obtained for substations 287 (left) and 307 (right) using the arctan loss (orange) or the regular pinball loss (blue). For substation 287, both models are biased. All the quantiles are too large, likely the result of a distributional shift between the training and test data. For substation 307, both models are calibrated very well.

substantially fewer crossings.

6.5 Conclusion

This chapter introduced a novel smooth approximation of the pinball loss function, termed the arctan pinball loss, which has been specifically designed to meet the needs of the XGBoost framework. The key advantage of this loss function lies in its second derivative, which decreases significantly more slowly than that of the currently available alternatives.

This arctan loss facilitates the use of a single model for multiple quantiles simultaneously. This is both more efficient and greatly reduces the number of quantile crossings. The experimental results demonstrate that this approach is viable for a wide range of data sets and yields competitive results while using only a single model and while having far fewer quantile crossings.

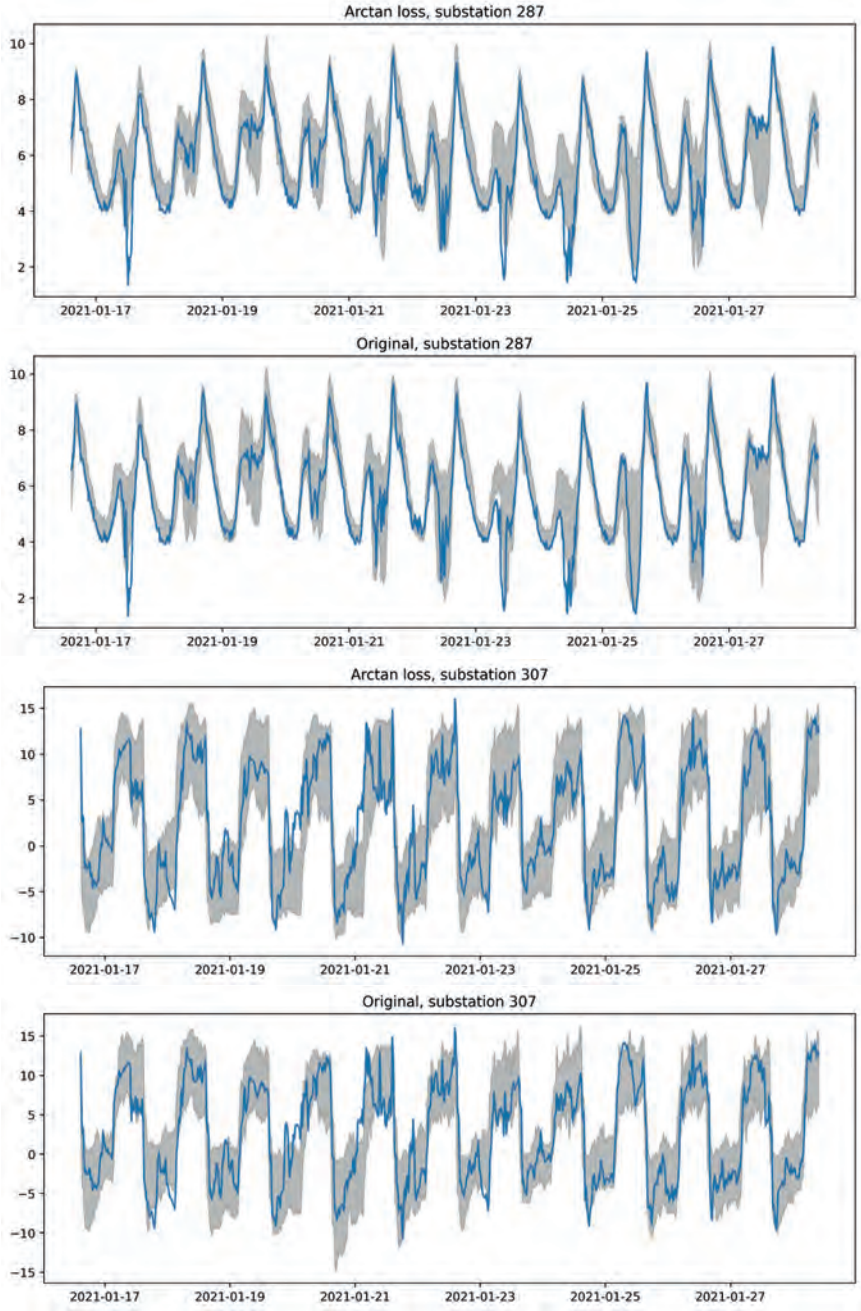


Figure 6.11: A visualization of the time series for substations 287 and 307. The grey area gives the 90% PI that is constructed using the 0.05- and 0.95-quantiles. For substation 287, the values of the lowest peaks are overestimated by both models and there is a substantial bias around the 24th of January.

APPENDIX CHAPTER 6

6.A Constructing the Arctan Pinball Loss

Recall that we defined $u := y - \hat{y}$. We will discuss the pinball loss as a function of u and τ, the desired quantile. The classical pinball loss is $(\tau - 1)u$ for $u < 0$ and τu for $u > 0$. We can place this pinball loss in a larger set of functions, namely $\{L(u) = (\tau - f(u))u\}$. For the pinball loss, $f(u)$ is a stepfunction that goes from 1 to 0 at $u = 0$.

We can also consider other functions for $f(u)$ that go from 1 to 0 to find approximations of the loss function. For a loss function to be suitable for XGBoost we require that $L''(u)$ is non-negligible in the relevant domain of u. If the targets are standardized, this relevant domain is roughly from -10 to 10 for most data sets.

It is immediately clear that the classical pinball loss is not suitable. A simple calculation shows that

$$L''(u) = -2f'(u) - f''(u)u.$$

Writing it out in terms of $f(u)$ allows us to easily check different functions and see how quickly the second derivative goes to zero.

We propose the following function f as a suitable candidate:

$$f_s(u) = 0.5 - \frac{\arctan(u/s)}{\pi}.$$

Using this f would result in the following loss function:

$$L_{\tau,s}(u) = (\tau - 0.5 + \frac{\arctan(u/s)}{\pi})u.$$

However, this loss function is asymptotically biased, as we demonstrate for the

limit $u \to \infty$. The limit u to $-\infty$ is identical and can be obtained similarly.

$$\begin{aligned}
\lim_{u\to\infty} L_{\tau,s}(u) - L_\tau^{\text{pinball}}(u) &= \lim_{u\to\infty} (\tau - 0.5 + \frac{\arctan(u/s)}{\pi})u - \tau \cdot u \\
&= \lim_{u\to\infty} (-0.5 + \frac{\arctan(u/s)}{\pi})u \\
&= \lim_{u\to\infty} \frac{(-0.5 + \frac{\arctan(u/s)}{\pi})}{u^{-1}} \\
&\overset{\text{L'Hôpital}}{=} \lim_{u\to\infty} -\frac{\frac{1}{\pi s}\frac{1}{1+(u/s)^2}}{u^{-2}} \\
&= -\frac{1}{\pi s} \lim_{u\to\infty} \frac{u^2}{1+(u/s)^2} \\
&= -\frac{1}{\pi s} \lim_{u\to\infty} \frac{s^2u^2}{s^2+u^2} \\
&= -\frac{s}{\pi}.
\end{aligned}$$

To obtain an asymptotically unbiased loss function, we therefore add an $\frac{s}{\pi}$ term and end up with our arctan pinball loss:

$$L_{\tau,s}^{\arctan}(u) = (\tau - 0.5 + \frac{\arctan(u/s)}{\pi})u + \frac{s}{\pi}.$$

Crucially, the second derivative of this arctan pinball loss is polynomial:

$$\begin{aligned}
\frac{\partial^2 L_{\tau,s}^{\arctan}(u)}{\partial u^2} &= \frac{2}{\pi s}\left(1 + (u/s)^2\right)^{-1} - \frac{2u^2}{\pi s^3}\left(1 + (u/s)^2\right)^{-2} \\
&= \frac{2}{\pi s}(1 + (u/s)^2)^{-2}.
\end{aligned}$$

Research Data Management

The research in this thesis has been carried out under the institute research data management policy of the Institute for Mathematics, Astrophysics and Particle Physics (IMAPP) of the Radboud University.

The more substantial pieces of code used in the various chapters have been made publicly available on GitHub:

Chapter 3 `https://github.com/LaurensSluyterman/Bootstrapped_Deep_Ensembles`

Chapter 4 `https://github.com/LaurensSluyterman/Mean_Variance_Estimation`

Chapter 5 `https://github.com/LaurensSluyterman/Likelihood_ratio_intervals`

Chapter 6 `https://github.com/LaurensSluyterman/XGBoost_quantile_regression`

Summary

The work in this thesis contributes to the field of uncertainty quantification for machine-learning models, in particular neural networks. The first chapter gives an introduction to the field by first going through the various sources of uncertainty and then discussing popular approaches to quantify each of these uncertainties.

Roughly speaking, there are three sources of uncertainty. Firstly, we are unsure about the optimal model parameters because the model is trained on a finite amount of random data. Secondly, even *if* we were absolutely certain about the model parameters, the quantity that we are trying to predict may be inherently random and thus we can never be entirely sure. Lastly, the data on which we trained could be not entirely representative of the data for which we want to make predictions. Each chapter focuses on improving the estimation of one or multiple of these uncertainty sources.

Chapter 2 proposes an improved approach to evaluate the quality uncertainty estimates in a regression setting. By using a simulation-based approach, methods that output a density can be compared with methods that output prediction intervals. An added benefit of using a simulated data is that the true data-generating process is known. This facilitates explicit testing of conditional prediction and confidence intervals.

The third chapter provides an extension to the popular *Deep Ensembles* approach. As noted by the original authors (Lakshminarayanan et al., 2017), their approach (purposely) does not incorporate the uncertainty in the model parameters that is caused by training on a finite random sample. While it is possible to incorporate this, for example via bootstrapping, they found that

solely incorporating the uncertainty due to the random training produced better results. The reasoning is that, when bootstrapping, the model effectively trains on fewer data points, leading to a lower performance. We provide a parametric bootstrapping approach that allows us to incorporate the missing uncertainty component without affecting accuracy and show that this results in improved coverage of the resulting confidence intervals.

Chapter 4 presents an improvement to Mean-Variance Estimation networks (Nix and Weigend, 1994). These networks estimate both the conditional mean and variance, assuming a normal distribution. We show that recently reported convergence problems with these networks (Skafte et al., 2019) can be relatively easily prevented by using the warm-up that the original authors proposed. We provide experimental results that demonstrate how crucial this step can be. Additionally, we present a novel improvement by demonstrating that separately regularizing the parts of the network that predict the mean and the variance can result in improved performance.

In Chapter 5, we present a likelihood-ratio-based approach to construct a confidence interval for the prediction of a network for a specific new input. Intuitively, this approach works by asking the question: *"For this new input, what predictions can the network make while still explaining the data reasonably well?"* We show that answering this question leads to confidence intervals with very desirable properties such as the ability to become asymmetric in biased regions and greatly expanding in regions where there is less data.

The final Chapter 6 diverges from studying neural networks and discusses a novel loss function that facilitates the use of the popular XGBoost model for quantile regression. This chapter is the result of a collaboration with the Dutch power-grid operator Alliander, who approached us with an interesting case. Their goal is to use XGBoost to predict loads on substations of the grid. Because XGBoost is very efficient and capable of dealing with missing values, this is their model of choice. However, as we see in this chapter, it is a non-trivial task to use this model for quantile regression because the loss function that is typically used for that task, the pinball loss, has a second derivative of zero. This is problematic since XGBoost uses the second derivative in the optimization procedure. We therefore develop a novel loss function that has a second derivative that approaches zero polynomially.

Samenvatting

Dit proefschrift bestudeert onzekerheidskwantificatie voor machine learning modellen, in het bijzonder neurale netwerken. Het eerste hoofdstuk geeft een inleiding tot het veld door de verschillende bronnen van onzekerheid te bespreken en per bron populaire methoden te behandelen om deze te kwantificeren.

Grofweg zijn er drie bronnen van onzekerheid. Ten eerste zijn we, omdat het model is getraind op een eindige hoeveelheid willekeurige data, onzeker over de optimale parameters van het model. Ten tweede, zelfs *als* we de optimale parameters exact zouden weten, kan wat we proberen te voorspellen inherent willekeurig zijn. Hierdoor kunnen we dus nooit helemaal zeker zijn van een nieuwe voorspelling. Ten derde is het mogelijk dat de dataset waarop we ons model baseren niet volledig representatief is voor de data waarvoor we voorspellingen willen maken. Elk hoofdstuk presenteert verbeteringen voor het schatten van één of meerdere van deze bronnen van onzekerheid.

Het tweede hoofdstuk beschrijft een nieuwe aanpak om de kwaliteit van onzekerheidsschattingen te testen in een regressiecontext. Deze aanpak maakt gebruik van simulaties waardoor het mogelijk wordt methoden die een dichtheid voorspellen te vergelijken met methoden die een betrouwbaarheids- of predictie-interval voorspellen. Een bijkomend voordeel is dat het echte datagenererende proces bekend is. Hierdoor is het mogelijk expliciet de conditionele betrouwbaarheids- en predictie-intervallen te testen.

Het derde hoofdstuk beschrijft een uitbreiding op de populaire *Deep Ensembles* techniek. Zoals opgemerkt door de oorspronkelijke auteurs (Lakshminarayanan et al., 2017), houdt hun aanpak (bewust) geen rekening met de onzekerheid in de modelparameters die wordt veroorzaakt door het trainen op een eindige

hoeveelheid willekeurige data. Hoewel het mogelijk is om dit wel te doen, bijvoorbeeld via een bootstrap, merkten zij op dat alleen het meenemen van de onzekerheid door het niet-deterministische trainingsproces tot betere resultaten leidde. De reden is dat bij bootstrappen het model effectief op minder data traint, wat leidt tot een lagere precisie. Wij presenteren een parametrische bootstrap benadering waarmee de ontbrekende onzekerheidscomponent gevangen kan worden zonder dat dit invloed heeft op de nauwkeurigheid. We tonen aan dat dit leidt tot een verbeterde coverage van de resulterende betrouwbaarheidsintervallen.

Hoofdstuk 4 presenteert een verbetering van Mean-Variance Estimation netwerken (Nix and Weigend, 1994). Deze netwerken schatten zowel het conditionele gemiddelde als de variantie, onder aanname van een normale verdeling. We laten zien dat recent gerapporteerde convergentieproblemen met deze netwerken (Skafte et al., 2019) relatief eenvoudig kunnen worden voorkomen door de aanbevelingen van de oorspronkelijke auteurs te volgen. Specifiek blijkt het cruciaal om eerst het gemiddelde te leren voordat de variantie wordt geoptimaliseerd. We laten experimenteel zien hoe cruciaal deze stap kan zijn. Verder zien we dat deze netwerken verbeterd kunnen worden door het apart regulariseren van de delen van het netwerk die respectievelijk het gemiddelde en de variantie voorspellen.

In Hoofdstuk 5 presenteren we een aanpak, gebaseerd op de likelihood-ratio test, om een betrouwbaarheidsinterval te construeren voor de voorspelling van een netwerk voor een specifiek nieuw datapunt. Deze benadering werkt door de intuïtieve vraag te stellen: *"Welke voorspellingen kan het netwerk maken voor dit nieuwe datapunt terwijl het nog steeds de rest van de data goed voorspelt?"* We tonen aan dat het beantwoorden van deze vraag leidt tot betrouwbaarheidsintervallen met zeer wenselijke eigenschappen, zoals de mogelijkheid om asymmetrisch te worden in gebieden met een bias en aanzienlijk groter te worden in gebieden waar minder data beschikbaar is.

Het laatste hoofdstuk, Hoofdstuk 6, gaat als enige niet over neurale netwerken maar bestudeert een nieuwe loss-functie die het gebruik van het populaire XGBoost-model voor kwantielregressie mogelijk maakt. Dit hoofdstuk is het resultaat van een samenwerking met de Nederlandse netbeheerder Alliander, die ons benaderde met een interessant probleem. Hun doel is om XGBoost te gebruiken om de belasting op transformatorstations van het net te voorspellen. Omdat XGBoost zeer efficiënt is en goed met ontbrekende waarden kan omgaan, willen ze graag dit model gebruiken. Echter, zoals we in dit hoofdstuk zien, is het een niet-triviale taak om dit model voor kwantielregressie te

gebruiken. Dit komt doordat de loss-functie die typisch voor die taak wordt gebruikt, de pinball-loss, een tweede afgeleide van nul heeft. Dit is problematisch omdat XGBoost de tweede afgeleide gebruikt tijdens de optimalisatie. We ontwikkelen daarom een nieuwe loss-functie die een tweede afgeleide heeft die op polynomiale wijze naar nul gaat.

Publications

The work during this PhD project has resulted into three published papers, one accepted paper, and three papers that are currently under review.

Published or Accepted

- Sluijterman et al. (2024a): Sluijterman, L., Cator, E., & Heskes, T. *"How to Evaluate Uncertainty Estimates in Machine Learning for Regression?"*, published in *Neural Networks*.
- Sluijterman et al. (2024b): Sluijterman, L., Cator, E., & Heskes, T. *"Optimal Training of Mean Variance Estimation Neural Networks"*, published in *Neurocomputing*.
- Van Borselen et al. (2024)[2]: Van Borselen, M., Sluijterman, L., Greupink, R., & de Wildt, S. *"Towards More Robust Evaluation of the Predictive Performance of Physiologically Based Pharmacokinetic Models: Using Confidence Intervals to Support Use of Model-Informed Dosing in Clinical Care"*, published in *Clinical Pharmacokinetics*.
- Sluijterman et al. (2023): Sluijterman, L., Cator, E., & Heskes, T. *"Likelihood-ratio-based confidence intervals for neural networks"*, accepted for publication in *Machine Learning*.

[2]Not part of this thesis, joint first author

Under Review

- Sluijterman et al. (2022): Sluijterman, L., Cator, E., & Heskes, T. *"Confident Neural Network Regression with Bootstrapped Deep Ensembles"*.
- Sluijterman et al. (2024c): Sluijterman, L., Kreuwel, F., Cator, E., & Heskes, T. *"Composite Quantile Regression With XGBoost Using the Novel Arctan Pinball Loss"*.
- Krebbers et al. (2024)[3]: Krebbers, R., Sluijterman, L., Meurs, J., Khodabakhsh, A., Cator, E., & Critescu, S. *"Optimizing Data Analysis for Broadband Mid-Infrared Absorption Spectroscopy: A Hybrid Dataset Approach"*.

[3]Not part of this thesis, joint first author

Curriculum Vitae

Laurens Sluijterman was born in Eindhoven, the Netherlands, on November 14th 1995. After completing high school, he enrolled at Radboud University in Nijmegen, where he obtained a bachelor's degree in Physics (*cum laude*) and Mathematics in 2018, and a master's degree in Mathematics (*cum laude*) in 2020 with a specialization in Applied Stochastics. That same year, at the same university, he started as a PhD student under the supervision of Eric Cator and Tom Heskes. In July 2024, Laurens began working as a researcher in the biostatistics group of the Radboud University Medical Center.

Acknowledgements

Four years is both a surprisingly short and surprisingly long period. It is short enough that it is over in a heartbeat and long enough that significant life events invariably occur. Both of these applied to the past four years, and there are many people who helped me tremendously to navigate them. Without some, it would have been far more difficult, or at the very least substantially less enjoyable, to finish this project.

First and foremost, I want to thank my supervisors Eric and Tom. In some ways you are very different: One of you is more chaotic, the other more organized; one of you answers emails in thirty minutes, the other somewhere between ten seconds and ten days. In other ways, however, you are the same: both excellent supervisors, and more importantly, both extremely pleasant to be around. Someone once told me that a good supervisor is more important than a good topic and I can hardly imagine better supervision.

To Tom: One of the more satisfying parts of my PhD were the times when I was working through your lists of suggestions and seeing a paper improve significantly in the span of a couple of hours. Knowing how busy you are, I do not know where you found the time to supervise me the way that you did. Sometimes I suspect you must have an identical twin. I also enjoyed our frequent chats about tennis, although I did not particularly enjoy losing to you in straight sets, one of them being 6-1. You would think that being roughly three decades younger would give you an advantage.

To Eric: Our collaboration felt very complementary to me. We both approach problems quite differently: I would quickly try to code out a working example to play around a bit, whereas you were more theoretical, but the combination

of these approaches turned out to work very well. Although I must admit, trying to follow you on a whiteboard was troublesome at times. Partly because of the lightning speed with which you would work out some intricate detail of a linear model, but also because your whiteboard is not really white but rather a colorful mixture of older ideas – some (it is my suspicion) dating back to before the start of my PhD. I hope that your current health issues are over soon so that we can quickly see you back in front of your not-so-whiteboard.

To Greta, Astrid, and Emma: It is easy to forget, especially when everything is going well, how crucial your work is to a nice working environment. You have helped me with countless administrative tasks and the department would surely not be the same without the many outings and lunches that you organized.

To my fellow PhD students: Thanks for all the nice talks, coffee-machine chats, and lunches. Coming to the office was always very enjoyable, for a large part because of you.

A special thanks to Rein, without whom I think there would have been a good chance I never would have started a PhD at the mathematics department. I first got to know you when I saw you studying for a topology exam. I, the expert planner that I am, still had to start two days before the exam so I figured it may be a good idea to join your study sessions. You had worked through *every single* exercise in the lecture notes and explained some of the ones I was struggling with. I think it may be the first time I recall enjoying the struggle of working through something that was not immediately clear to me. During the years that followed, we worked together a lot, although I may have learned a bit more from you than the other way around. I am very grateful that we are still working together at the moment and I can hopefully return the favor over the coming years.

To Francesco: I greatly enjoyed all our nice discussions during our band rehearsals. The word *band* may be a bit too strong though, since at the end it was just the two of us, and the word *rehearsal* is also a bit overblown to describe the 30 minutes of playing preceding the two hours of drinking wine.

To my grandmother Corrie, who sadly passed away during my PhD: I don't think you quite understood what it was I was doing, or how this was any different than high school, but I do know that you were always proud regardless. You always made sure that getting a paper rejected or some other setback was quickly forgotten, just by being who you were. I also want to thank my uncle Michel for taking extremely good care of her during the last decade or so of her life.

To my father Seyno: At a very young age, I got to see you defend your PhD, sitting all the way at the back next to my grandmother who made sure I was behaving, and perhaps that unconsciously inspired me to want to do the same. I also appreciate the many occasions on which you drove all the way from Eindhoven to Nijmegen, just to have lunch.

To Maarten and Esther: Thank you for always watching the dog whenever I was at the office and for celebrating every success, no matter the (lack of) significance, with cake, dinner, or a postcard.

To Marjolein: First of all, thank you for helping me design my cover, which is based on a short example from the introduction. The thinking robot sees a horse via its neural network, but since it is trained on only cats and dogs, it is determining if this is a rather large dog or an enormous cat. I am also grateful for you introducing me to the medical side of the university and allowing me to contribute to your own PhD research. This has surely helped me to get my current job, which I enjoy profoundly. Without you, doing this PhD, or all of life for that matter, would have been substantially less enjoyable.

Bibliography

Abdar, M., Pourpanah, F., Hussain, S., Rezazadegan, D., Liu, L., Ghavamzadeh, M., Fieguth, P., Cao, X., Khosravi, A., Acharya, U. R., et al. (2021). A review of uncertainty quantification in deep learning: Techniques, applications and challenges. *Information Fusion*.

Akaike, H. (1973). Information theory and an extension of the likelihood principle. In *Proceedings of the Second International Symposium of Information Theory*.

Amini, A., Schwarting, W., Soleimany, A., and Rus, D. (2020). Deep Evidential Regression. In *Advances in Neural Information Processing Systems*, volume 33, pages 14927–14937. Curran Associates, Inc.

Andersen, P. K., Borgan, O., Gill, R. D., and Keiding, N. (2012). *Statistical Models Based on Counting Processes*. Springer Science & Business Media.

Ashukha, A., Lyzhov, A., Molchanov, D., and Vetrov, D. (2019). Pitfalls of in-domain uncertainty estimation and ensembling in deep learning. In *International Conference on Learning Representations*.

Ayhan, M. S. and Berens, P. (2022). Test-time Data Augmentation for Estimation of Heteroscedastic Aleatoric Uncertainty in Deep Neural Networks. In *Medical Imaging with Deep Learning*.

Belloni, A. and Chernozhukov, V. (2011). ℓ1-penalized quantile regression in high-dimensional sparse models. *The Annals of Statistics*, 39(1):82–130.

Bentivoglio, R., Isufi, E., Jonkman, S. N., and Taormina, R. (2022). Deep learn-

ing methods for flood mapping: A review of existing applications and future research directions. *Hydrology and Earth System Sciences*, 26(16):4345–4378.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, New York.

Boucheron, S. and Massart, P. (2011). A high-dimensional Wilks phenomenon. *Probability Theory and Related Fields*, 150(3):405–433.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Cannon, A. J. (2011). Quantile regression neural networks: Implementation in R and application to precipitation downscaling. *Computers & Geosciences*, 37(9):1277–1284.

Cerqueira, V., Torgo, L., and Mozetič, I. (2020). Evaluating time series forecasting models: An empirical study on performance estimation methods. *Machine Learning*, 109(11):1997–2028.

Chaudhary, P., Leitão, J. P., Donauer, T., D'Aronco, S., Perraudin, N., Obozinski, G., Perez-Cruz, F., Schindler, K., Wegner, J. D., and Russo, S. (2022). Flood uncertainty estimation using deep ensembles. *Water*, 14(19):2980.

Chaudhuri, P. (1991). Nonparametric Estimates of Regression Quantiles and Their Local Bahadur Representation. *The Annals of Statistics*, 19(2).

Chen, T. and Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA. Association for Computing Machinery.

Chen, X., Koenker, R., and Xiao, Z. (2009). Copula-based nonlinear quantile autoregression. *The Econometrics Journal*, 12(s1):S50–S67.

Chen, Z., Zhang, B., Meng, A., and Li, P. (2023). Prediction interval estimation of dynamic thermal rating considering weather uncertainty. *Electric Power Systems Research*, 214:108927.

Chernozhukov, V. and Hansen, C. (2006). Instrumental quantile regression inference for structural and treatment effect models. *Journal of Econometrics*, 132(2):491–525.

Chizat, L., Oyallon, E., and Bach, F. (2019). On lazy training in differentiable programming. In *NeurIPS 2019-33rd Conference on Neural Information Processing Systems*, pages 2937–2947.

Chung, Y., Neiswanger, W., Char, I., and Schneider, J. (2021). Beyond Pinball Loss: Quantile Methods for Calibrated Uncertainty Quantification. In *Advances in Neural Information Processing Systems*, volume 34, pages 10971–10984. Curran Associates, Inc.

Clements, W. R., Robaglia, B.-M., Van Delft, B., Slaoui, R. B., and Toth, S. (2019). Estimating risk and uncertainty in deep reinforcement learning. *arXiv preprint arXiv:1905.09638*.

Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*.

Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., and Le, Q. V. (2019). AutoAugment: Learning Augmentation Strategies From Data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 113–123.

Dastin, J. (2022). Amazon Scraps Secret AI Recruiting Tool that Showed Bias against Women. In *Ethics of Data and Analytics*, pages 296–299. Auerbach Publications.

Degras, D. (2017). Simultaneous confidence bands for the mean of functional data. *Wiley Interdisciplinary Reviews: Computational Statistics*, 9(3):e1397.

DeGroot, M. H. (1986). *Probability and Statistics*. Addison-Wesley Pub. Co, Reading, Mass, 2nd ed edition.

DeGroot, M. H. and Fienberg, S. E. (1983). The Comparison and Evaluation of Forecasters. *Journal of the Royal Statistical Society. Series D (The Statistician)*, 32(1/2):12–22.

Deng, D., Chen, G., Yu, Y., Liu, F., and Heng, P.-A. (2023). Uncertainty Estimation by Fisher Information-based Evidential Deep Learning. *arXiv preprint arXiv:2303.02045*.

Dewolf, N., Baets, B. D., and Waegeman, W. (2023). Valid prediction intervals for regression problems. *Artificial Intelligence Review*, 56(1):577–613.

Dhaliwal, S. S., Nahid, A.-A., and Abbas, R. (2018). Effective Intrusion Detection System Using XGBoost. *Information*, 9(7):149.

Dheur, V. and Taieb, S. B. (2023). A Large-Scale Study of Probabilistic Calibration in Neural Network Regression. In *Proceedings of the 40th International Conference on Machine Learning*, pages 7813–7836. PMLR.

Dodge, Y. (2008). *The Concise Encyclopedia of Statistics*. Springer Science & Business Media.

Dwaracherla, V., Wen, Z., Osband, I., Lu, X., Asghari, S. M., and Van Roy, B. (2022). Ensembles for uncertainty estimation: Benefits of prior functions and bootstrapping. *arXiv preprint arXiv:2206.03633*.

Efron, B. (1982). *The Jackknife, the Bootstrap and Other Resampling Plans*. SIAM.

Egele, R., Maulik, R., Raghavan, K., Lusch, B., Guyon, I., and Balaprakash, P. (2022). AutoDEUQ: Automated Deep Ensemble with Uncertainty Quantification. In *2022 26th International Conference on Pattern Recognition (ICPR)*, pages 1908–1914.

Fan, J., Zhang, C., and Zhang, J. (2001). Generalized Likelihood Ratio Statistics and Wilks Phenomenon. *The Annals of Statistics*, 29(1):153–193.

Fort, S., Hu, H., and Lakshminarayanan, B. (2019). Deep ensembles: A loss landscape perspective. *arXiv preprint arXiv:1912.02757*.

Gal, Y. (2016). *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge.

Gal, Y. and Ghahramani, Z. (2016). Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning*, pages 1050–1059.

Gal, Y., Hron, J., and Kendall, A. (2017). Concrete dropout. *Advances in Neural Information Processing Systems*, 30.

Gauss, C.-F. (1823). *Theoria Combinationis Observationum Erroribus Minimis Obnoxiae*. Henricus Dieterich.

Gawlikowski, J., Tassi, C. R. N., Ali, M., Lee, J., Humt, M., Feng, J., Kruspe, A., Triebel, R., Jung, P., Roscher, R., Shahzad, M., Yang, W., Bamler, R.,

and Zhu, X. X. (2023). A survey of uncertainty in deep neural networks. *Artificial Intelligence Review*, 56(1):1513–1589.

George, T., Laurent, C., Bouthillier, X., Ballas, N., and Vincent, P. (2018). Fast Approximate Natural Gradient Descent in a Kronecker Factored Eigenbasis. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Ghosal, S. and Van der Vaart, A. (2017). *Fundamentals of Nonparametric Bayesian Inference*, volume 44. Cambridge University Press.

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

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014a). Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014b). Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

Graves, A. (2011). Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, pages 2348–2356. Citeseer.

Gumus, M. and Kiran, M. S. (2017). Crude oil price forecasting using XGBoost. In *2017 International Conference on Computer Science and Engineering (UBMK)*, pages 1100–1103, Antalya. IEEE.

Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. (2017). On calibration of modern neural networks. In *International Conference on Machine Learning*, pages 1321–1330. PMLR.

Guo, H., Liu, H., Li, R., Wu, C., Guo, Y., and Xu, M. (2018). Margin & diversity based ordering ensemble pruning. *Neurocomputing*, 275:237–246.

Gustafsson, F. K., Danelljan, M., and Schon, T. B. (2020). Evaluating scalable Bayesian deep learning methods for robust computer vision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 318–319.

Hall, P. and La Scala, B. (1990). Methodology and Algorithms of Empirical Likelihood. *International Statistical Review / Revue Internationale de Statistique*, 58(2):109–127.

Hamanda, A. (2020). Br35H :: Brain Tumor Detection 2020.

Hansen, L. and Salamon, P. (1990). Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001.

Hatalis, K., Lamadrid, A. J., Scheinberg, K., and Kishore, S. (2019). A Novel Smoothed Loss and Penalty Function for Noncrossing Composite Quantile Estimation via Deep Neural Networks. *arXiv preprint arXiv:1909.12122*.

He, W. and Jiang, Z. (2023). A Survey on Uncertainty Quantification Methods for Deep Neural Networks: An Uncertainty Source Perspective. *arXiv preprint arXiv:2302.13425*.

Hendrycks, D. and Gimpel, K. (2018). A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks. *arXiv preprint arXiv:1610.02136*.

Hendrycks, D., Mazeika, M., and Dietterich, T. (2018). Deep Anomaly Detection with Outlier Exposure. In *International Conference on Learning Representations*.

Hernández, P. D., Ramírez, J. A., and Soto, M. A. (2022). Deep-Learning-Based Earthquake Detection for Fiber-Optic Distributed Acoustic Sensing. *Journal of Lightwave Technology*, 40(8):2639–2650.

Hernández-Lobato, J. M. and Adams, R. (2015). Probabilistic backpropagation for scalable learning of Bayesian neural networks. In *International Conference on Machine Learning*, pages 1861–1869.

Herron, E. J., Young, S. R., and Potok, T. E. (2020). Ensembles of Networks Produced from Neural Architecture Search. In Jagode, H., Anzt, H., Juckeland, G., and Ltaief, H., editors, *High Performance Computing*, pages 223–234, Cham. Springer International Publishing.

Heskes, T. (1997). Practical confidence and prediction intervals. In *Advances in Neural Information Processing Systems*, pages 176–182.

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the Knowledge in a Neural Network. *arXiv preprint arXiv:1503.02531*.

Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(5786):504–507.

Hinton, G. E. and Van Camp, D. (1993). Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the Sixth Annual Conference on Computational Learning Theory*, pages 5–13.

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

Hüllermeier, E. and Waegeman, W. (2021). Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. *Machine Learning*, 110(3):457–506.

Jain, S., Liu, G., Mueller, J., and Gifford, D. (2020). Maximizing overall diversity for improved uncertainty estimates in deep ensembles. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 4264–4271.

Jiang, H., He, Z., Ye, G., and Zhang, H. (2020). Network Intrusion Detection Based on PSO-Xgboost Model. *IEEE Access*, 8:58392–58401.

Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., and Saul, L. K. (1999). An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233.

Juergens, M., Meinert, N., Bengs, V., Hüllermeier, E., and Waegeman, W. (2024). Is Epistemic Uncertainty Faithfully Represented by Evidential Deep Learning Methods? In *Proceedings of the 41st International Conference on Machine Learning*, pages 22624–22642. PMLR.

Kabir, H. D., Khosravi, A., Hosen, M. A., and Nahavandi, S. (2018). Neural network-based uncertainty quantification: A survey of methodologies and applications. *IEEE access*, 6:36218–36234.

Kabir, H. M. D., Khosravi, A., Nahavandi, S., and Srinivasan, D. (2021). Neural Network Training for Uncertainty Quantification Over Time-Range. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 5(5):768–779.

Kabir, H. M. D., Mondal, S. K., Khanam, S., Khosravi, A., Rahman, S., Qazani, M. R. C., Alizadehsani, R., Asadi, H., Mohamed, S., Nahavandi, S., and Acharya, U. R. (2023). Uncertainty aware neural network from similarity and sensitivity. *Applied Soft Computing*, 149:111027.

Kallus, N. and McInerney, J. (2022). The Implicit Delta Method. In *Advances in Neural Information Processing Systems*.

Kass, R., Tierney, L., and Kadane, J. (1991). Laplace's method in Bayesian analysis. *Contemporary Mathematics*, pages 89–135.

Kendall, A. and Gal, Y. (2017). What Uncertainties Do We Need in Bayesian

Deep Learning for Computer Vision? In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Khosravi, A. and Nahavandi, S. (2014). An optimized mean variance estimation method for uncertainty quantification of wind power forecasts. *International Journal of Electrical Power & Energy Systems*, 61:446–454.

Khosravi, A., Nahavandi, S., Creighton, D., and Atiya, A. F. (2011). Comprehensive review of neural network-based prediction intervals and new advances. *IEEE Transactions on Neural Networks*, 22(9):1341–1356.

Kim, I., Kim, Y., and Kim, S. (2020). Learning Loss for Test-Time Augmentation. *Advances in Neural Information Processing Systems*, 33:4163–4174.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P., Salimans, T., and Welling, M. (2015). Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pages 2575–2583.

Kobyzev, I., Prince, S. J. D., and Brubaker, M. A. (2021). Normalizing Flows: An Introduction and Review of Current Methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(11):3964–3979.

Koenker, R. (2017). Quantile regression 40 years on. Technical report, The IFS.

Koenker, R. and Bassett Jr, G. (1978). Regression quantiles. *Econometrica: Journal of the Econometric Society*, pages 33–50.

Koenker, R. and Xiao, Z. (2006). Quantile Autoregression. *Journal of the American Statistical Association*, 101(475):980–990.

Krebbers, R., Sluijterman, L., Meurs, J., Khodabakhsh, A., Cator, E., and Critescu, S. (2024). Optimizing Data Analysis for Broadband Mid-Infrared Absorption Spectroscopy: A Hybrid Dataset Approach. *Paper under submission*.

Kuleshov, V., Fenner, N., and Ermon, S. (2018). Accurate Uncertainties for Deep Learning Using Calibrated Regression. In *Proceedings of the 35th International Conference on Machine Learning*, pages 2796–2804. PMLR.

Lai, Y., Shi, Y., Han, Y., Shao, Y., Qi, M., and Li, B. (2022). Exploring uncertainty in regression neural networks for construction of prediction intervals. *Neurocomputing*.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, pages 6402–6413.

LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., and Jackel, L. (1989). Handwritten Digit Recognition with a Back-Propagation Network. In *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann.

Lee, J., Humt, M., Feng, J., and Triebel, R. (2020). Estimating Model Uncertainty of Neural Networks in Sparse Information Form. In *Proceedings of the 37th International Conference on Machine Learning*, pages 5702–5713. PMLR.

Lee, K., Lee, K., Lee, H., and Shin, J. (2018). A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In *Advances in Neural Information Processing Systems*.

Lee, S. (2003). Efficient Semiparametric Estimation of a Partially Linear Quantile Regression Model. *Econometric Theory*, 19(1):1–31.

Li, W., Yin, Y., Quan, X., and Zhang, H. (2019). Gene Expression Value Prediction Based on XGBoost Algorithm. *Frontiers in Genetics*, 10.

Liu, Q. and Wang, D. (2016). Stein variational gradient descent: A general purpose Bayesian inference algorithm. In *Advances in Neural Information Processing Systems*, pages 2378–2386.

Lu, Z., Pu, H., Wang, F., Hu, Z., and Wang, L. (2017). The expressive power of neural networks: A view from the width. *Advances in Neural Information Processing Systems*, 30:6231–6239.

Lyzhov, A., Molchanova, Y., Ashukha, A., Molchanov, D., and Vetrov, D. (2020). Greedy Policy Search: A Simple Baseline for Learnable Test-Time Augmentation. In *Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 1308–1317. PMLR.

Ma, M., Zhao, G., He, B., Li, Q., Dong, H., Wang, S., and Wang, Z. (2021). XGBoost-based method for flash flood risk assessment. *Journal of Hydrology*, 598:126382.

MacKay, D. J. (1992a). A practical Bayesian framework for backpropagation networks. *Neural Computation*, 4(3):448–472.

MacKay, D. J. C. (1992b). Information-Based Objective Functions for Active Data Selection. *Neural Computation*, 4(4):590–604.

Malinin, A., Chervontsev, S., Provilkov, I., and Gales, M. (2020). Regression Prior Networks. *arXiv preprint arXiv:2006.11590*.

Malinin, A. and Gales, M. (2018). Predictive Uncertainty Estimation via Prior Networks. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Mancini, T., Calvo-Pardo, H., and Olmo, J. (2020). Prediction intervals for deep neural networks. *arXiv preprint arXiv:2010.04044*.

Mangel, M. and Samaniego, F. J. (1984). Abraham Wald's Work on Aircraft Survivability. *Journal of the American Statistical Association*, 79(386):259–267.

Martens, J. and Grosse, R. (2015). Optimizing Neural Networks with Kronecker-factored Approximate Curvature. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 2408–2417. PMLR.

Mehrabi, N., Morstatter, F., Saxena, N., Lerman, K., and Galstyan, A. (2022). A Survey on Bias and Fairness in Machine Learning. *ACM Computing Surveys*, 54(6):1–35.

Meinshausen, N. (2006). Quantile Regression Forests. *The Journal of Machine Learning Research*, 7:983–999.

Miglani, A. and Kumar, N. (2019). Deep learning models for traffic flow prediction in autonomous vehicles: A review, solutions, and challenges. *Vehicular Communications*, 20:100184.

Mukhoti, J., Kirsch, A., van Amersfoort, J., Torr, P. H., and Gal, Y. (2021). Deterministic neural networks with appropriate inductive biases capture epistemic and aleatoric uncertainty. *arXiv preprint arXiv:2102.11582*.

Murphy, A. H. and Winkler, R. L. (1977). Reliability of Subjective Probability Forecasts of Precipitation and Temperature. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 26(1):41–47.

Murphy, S. A. (1995). Likelihood Ratio-Based Confidence Intervals in Survival Analysis. *Journal of the American Statistical Association*, 90(432):1399–1405.

Murphy, S. A. and van der Vaart, A. W. (1997). Semiparametric likelihood ratio inference. *The Annals of Statistics*, 25(4):1471–1509.

Neal, R. M. (2012). *Bayesian Learning for Neural Networks*, volume 118. Springer Science & Business Media.

Neal, R. M. et al. (2011). MCMC using hamiltonian dynamics. *Handbook of markov chain monte carlo*, 2(11):2.

Niculescu-Mizil, A. and Caruana, R. (2005). Predicting good probabilities with supervised learning. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 625–632.

Nilsen, G. K., Munthe-Kaas, A. Z., Skaug, H. J., and Brun, M. (2022). Epistemic uncertainty quantification in deep learning classification by the Delta method. *Neural Networks*, 145:164–176.

Nix, D. A. and Weigend, A. S. (1994). Estimating the mean and variance of the target probability distribution. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, volume 1, pages 55–60. IEEE.

Nixon, J., Lakshminarayanan, B., and Tran, D. (2020). Why are bootstrapped deep ensembles not better? In *"I Can't Believe It's Not Better!"NeurIPS 2020 Workshop*.

Nobre, J. and Neves, R. F. (2019). Combining Principal Component Analysis, Discrete Wavelet Transform and XGBoost to trade in the financial markets. *Expert Systems with Applications*, 125:181–194.

Nourani, V., Zonouz, R. S., and Dini, M. (2023). Estimation of prediction intervals for uncertainty assessment of artificial neural network based wastewater treatment plant effluent modeling. *Journal of Water Process Engineering*, 55:104145.

Ogunleye, A. and Wang, Q.-G. (2020). XGBoost Model for Chronic Kidney Disease Diagnosis. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 17(6):2131–2140.

Osband, I. (2016). Risk versus uncertainty in deep learning: Bayes, bootstrap and the dangers of dropout. In *NIPS Workshop on Bayesian Deep Learning*, volume 192.

Ovadia, Y., Fertig, E., Ren, J., Nado, Z., Sculley, D., Nowozin, S., Dillon, J., Lakshminarayanan, B., and Snoek, J. (2019). Can you trust your model's uncertainty? Evaluating predictive uncertainty under dataset shift. *Advances in Neural Information Processing Systems*, 32:13991–14002.

Papamakarios, G., Pavlakou, T., and Murray, I. (2017). Masked Autoregressive Flow for Density Estimation. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Parekh, D., Poddar, N., Rajpurkar, A., Chahal, M., Kumar, N., Joshi, G. P., and Cho, W. (2022). A Review on Autonomous Vehicles: Progress, Methods and Challenges. *Electronics*, 11(14):2162.

Pav, S. E. (2015). Moments of the log non-central chi-square distribution. *arXiv preprint arXiv:1503.06266*.

Pawitan, Y. (2000). A Reminder of the Fallibility of the Wald Statistic: Likelihood Explanation. *The American Statistician*, 54(1):54–56.

Pawitan, Y. (2001). *In All Likelihood: Statistical Modelling and Inference Using Likelihood*. OUP Oxford.

Pearce, T. (2020). *Uncertainty in Neural Networks; Bayesian Ensembles, Priors & Prediction Intervals*. PhD thesis, University of Cambridge.

Pearce, T., Brintrup, A., Zaki, M., and Neely, A. (2018). High-quality prediction intervals for deep learning: A distribution-free, ensembled approach. In *International Conference on Machine Learning*, pages 4075–4084.

Pearce, T., Leibfried, F., and Brintrup, A. (2020). Uncertainty in neural networks: Approximately Bayesian ensembling. In *International Conference on Artificial Intelligence and Statistics*, pages 234–244. PMLR.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12(85):2825–2830.

Platt, J. et al. (1999). Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74.

Powell, J. L. (1986). Censored regression quantiles. *Journal of Econometrics*, 32(1):143–155.

Quiñonero-Candela, J., editor (2009). *Dataset Shift in Machine Learning*. Neural Information Processing Series. MIT Press, Cambridge, Mass.

Ramaneswaran, S., Srinivasan, K., Vincent, P. M. D. R., and Chang, C.-Y. (2021). Hybrid Inception v3 XGBoost Model for Acute Lymphoblas-

tic Leukemia Classification. *Computational and Mathematical Methods in Medicine*, 2021:1–10.

Ren, J., Fort, S., Liu, J., Roy, A. G., Padhy, S., and Lakshminarayanan, B. (2021). A simple fix to mahalanobis distance for improving near-ood detection. *arXiv preprint arXiv:2106.09022*.

Ren, J., Liu, P. J., Fertig, E., Snoek, J., Poplin, R., Depristo, M., Dillon, J., and Lakshminarayanan, B. (2019). Likelihood ratios for out-of-distribution detection. *Advances in Neural Information Processing Systems*, 32:14707–14718.

Romano, Y., Patterson, E., and Candes, E. (2019). Conformalized Quantile Regression. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Salimbeni, H. and Deisenroth, M. (2017). Doubly stochastic variational inference for deep Gaussian processes. In *Advances in Neural Information Processing Systems*, pages 4588–4599.

Sartaj, B. (2020). Brain Tumor Classification (MRI).

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning*, 5(2):197–227.

Seber, G. and Wild, C. (2003). *Nonlinear Regression*. Wiley.

Seitzer, M., Tavakoli, A., Antic, D., and Martius, G. (2021). On the pitfalls of heteroscedastic uncertainty estimation with probabilistic neural networks. In *International Conference on Learning Representations*.

Sensoy, M., Kaplan, L., and Kandemir, M. (2018). Evidential Deep Learning to Quantify Classification Uncertainty. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Shafer, G. and Vovk, V. (2008). A Tutorial on Conformal Prediction. *The Journal of Machine Learning Research*, 9:371–421.

Shaikhina, T. and Khovanova, N. A. (2017). Handling limited datasets with neural networks in medical applications: A small-data approach. *Artificial Intelligence in Medicine*, 75:51–63.

Shanmugam, D., Blalock, D., Balakrishnan, G., and Guttag, J. (2021). Better Aggregation in Test-Time Augmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1214–1223.

Skafte, N., Jørgensen, M., and Hauberg, S. (2019). Reliable training and estimation of variance networks. *Advances in Neural Information Processing Systems*, 32.

Sluijterman, L., Cator, E., and Heskes, T. (2022). Confident neural network regression with bootstrapped deep ensembles. *arXiv preprint arXiv:2202.10903*.

Sluijterman, L., Cator, E., and Heskes, T. (2023). Likelihood-ratio-based confidence intervals for neural networks. *arXiv preprint arXiv:2308.02221*.

Sluijterman, L., Cator, E., and Heskes, T. (2024a). How to evaluate uncertainty estimates in machine learning for regression? *Neural Networks*, 173:106203.

Sluijterman, L., Cator, E., and Heskes, T. (2024b). Optimal training of Mean Variance Estimation neural networks. *Neurocomputing*, 597:127929.

Sluijterman, L., Kreuwel, F., Cator, E., and Heskes, T. (2024c). Composite Quantile Regression With XGBoost Using the Novel Arctan Pinball Loss. *arXiv preprint arXiv:2406.02293*.

Su, D., Ting, Y. Y., and Ansel, J. (2018). Tight prediction intervals using expanded interval minimization. *arXiv preprint arXiv:1806.11222*.

Suzuki, M., Nakayama, K., and Matsuo, Y. (2016). Joint Multimodal Learning with Deep Generative Models.

Tagasovska, N. and Lopez-Paz, D. (2019). Single-model uncertainties for deep learning. In *Advances in Neural Information Processing Systems*, pages 6417–6428.

Takahashi, H., Iwata, T., Yamanaka, Y., Yamada, M., and Yagi, S. (2018). Student-t variational autoencoder for robust density estimation. In *IJCAI*, pages 2696–2702.

Theobald, C. M. (1974). Generalizations of mean square error applied to ridge regression. *Journal of the Royal Statistical Society: Series B (Methodological)*, 36(1):103–106.

Valdenegro-Toro, M. (2023). Sub-Ensembles for Fast Uncertainty Estimation in Neural Networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4119–4127.

Van Amersfoort, J., Smith, L., Jesson, A., Key, O., and Gal, Y. (2021). Improving deterministic uncertainty estimation in deep learning for classification and regression. *arXiv preprint arXiv:2102.11409*.

Van Amersfoort, J., Smith, L., Teh, Y. W., and Gal, Y. (2020). Uncertainty estimation using a single deep deterministic neural network. In *International Conference on Machine Learning*, pages 9690–9700. PMLR.

Van Beers, J. J. and De Visser, C. C. (2023). Peaking into the Black-box: Prediction Intervals Give Insight into Data-driven Quadrotor Model Reliability. In *AIAA SCITECH 2023 Forum*, National Harbor, MD & Online. American Institute of Aeronautics and Astronautics.

Van Borselen, M. D., Sluijterman, L. A. Æ., Greupink, R., and de Wildt, S. N. (2024). Towards More Robust Evaluation of the Predictive Performance of Physiologically Based Pharmacokinetic Models: Using Confidence Intervals to Support Use of Model-Informed Dosing in Clinical Care. *Clinical Pharmacokinetics*, 63(3):343–355.

Van den Goorbergh, R., Van Smeden, M., Timmerman, D., and Van Calster, B. (2022). The harm of class imbalance corrections for risk prediction models: Illustration and simulation using logistic regression. *Journal of the American Medical Informatics Association*, 29(9):1525–1534.

Van der Vaart, A. W. (2000). *Asymptotic Statistics*, volume 3. Cambridge University Press.

Van Giffen, B., Herhausen, D., and Fahse, T. (2022). Overcoming the pitfalls and perils of algorithms: A classification of machine learning biases and mitigation methods. *Journal of Business Research*, 144:93–106.

Van Wieringen, W. N. (2015). Lecture notes on ridge regression. *arXiv preprint arXiv:1509.09169*.

Varoquaux, G. and Cheplygina, V. (2022). Machine learning for medical imaging: Methodological failures and recommendations for the future. *npj Digital Medicine*, 5(1):1–8.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Wen, Y., Tran, D., and Ba, J. (2020). Batchensemble: An alternative approach to efficient ensemble and lifelong learning. *arXiv preprint arXiv:2002.06715*.

Wenzel, F., Snoek, J., Tran, D., and Jenatton, R. (2020). Hyperparameter Ensembles for Robustness and Uncertainty Quantification. In *Advances in*

Neural Information Processing Systems, volume 33, pages 6514–6527. Curran Associates, Inc.

Wilks, S. S. (1938). The Large-Sample Distribution of the Likelihood Ratio for Testing Composite Hypotheses. *The Annals of Mathematical Statistics*, 9(1):60–62.

Wilson, A. G. and Izmailov, P. (2020). Bayesian deep learning and a probabilistic perspective of generalization. *Advances in neural information processing systems*, 33:4697–4708.

Xu, Q., Deng, K., Jiang, C., Sun, F., and Huang, X. (2017). Composite quantile regression neural network with applications. *Expert Systems with Applications*, 76:129–139.

Yang, X., Narisetty, N. N., and He, X. (2018). A New Approach to Censored Quantile Regression Estimation. *Journal of Computational and Graphical Statistics*, 27(2):417–425.

Yin, X., Fallah-Shorshani, M., McConnell, R., Fruin, S., Chiang, Y.-Y., and Franklin, M. (2023). Quantile Extreme Gradient Boosting for Uncertainty Quantification. *arXiv preprint arXiv:2304.11732*.

Zhang, C. and Fu, Y. (2023). Probabilistic Electricity Price Forecast with Optimal Prediction Interval. *IEEE Transactions on Power Systems*, pages 1–10.

Zhang, H., Cisse, M., Dauphin, Y. N., and Lopez-Paz, D. (2017a). Mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*.

Zhang, J., Liu, Z., and Chen, T. (2023). Interval prediction of ultra-short-term photovoltaic power based on a hybrid model. *Electric Power Systems Research*, 216:109035.

Zhang, Z., Song, Y., and Qi, H. (2017b). Age Progression/Regression by Conditional Adversarial Autoencoder. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5810–5818.

Zhao, S., Ma, T., and Ermon, S. (2020). Individual Calibration with Randomized Forecasting. In *Proceedings of the 37th International Conference on Machine Learning*, pages 11387–11397. PMLR.

Zheng, S. (2011). Gradient descent algorithms for quantile regression with smooth approximation. *International Journal of Machine Learning and Cybernetics*, 2(3):191–207.

Zheng, Z. and Zhang, Z. (2023). A Stochastic Recurrent Encoder Decoder Network for Multistep Probabilistic Wind Power Predictions. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–14.

Zhou, Y., Zhou, Z., and Hooker, G. (2018). Approximation trees: Statistical stability in model distillation. *arXiv preprint arXiv:1808.07573*.

Zou, H. and Yuan, M. (2008). Composite quantile regression and the oracle model selection theory. *The Annals of Statistics*, 36(3):1108–1126.

$$s^2 = \frac{\sum (x_i - \bar{x})^2}{n-1}$$

Radboud Universiteit