Fakultät für Informatik
Technische Universität München

Modeling Epistemic and Aleatoric Uncertainty with Bayesian Neural Networks and Latent Variables

Stefan Depeweg

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:
Prof. Dr. Nils Thuerey

Prüfende der Dissertation:
1. Hon.-Prof. Dr. Thomas A. Runkler
2. Assistant Prof. Dr. Laura Leal-Taixé
3. Prof. José Miguel Hernández-Lobato, Ph.D.,
   University of Cambridge

Die Dissertation wurde am 10.04.2019 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 25.07.2019 angenommen.

Zusammenfassung

Maschinelle Lernverfahren, insbesondere neuronale Netze, sind Teil vieler datengetriebener Lösungen für reale Probleme. Während diese Methoden in vielen Fällen eine hohe Vorhersagequalität erreichen, ist dies nicht das einzig wichtige Qualitätsmerkmal. Im Anwendungsfall ist es auch notwendig die Unsicherheit über die Vorhersage des Modells zu schätzen. Hier ist zwischen epistemischer und aleatorischer Prognoseunsicherheit zu unterscheiden. Der epistemische Teil entsteht aus mangelndem Wissen über die wahre datengenerierende Funktion und ergibt sich aus dem Umfang der gegebenen Daten. Der aleatorische Teil ist nicht reduzierbar, er ist das Ergebnis von intrinsischen Zufall oder begrenzter Beobachtungsfähigkeit des datengenerierenden Prozesses. Moderne Bayes'sche Methoden, wie z.B. Bayes'sche neuronale Netze, sind skalierbare Black Box Algorithmen, die die Universalität neuronaler Netze mit dem prinzipiellen probabilistischen Ansatz der Bayes'schen Inferenz kombinieren. Diese Methoden modellieren jedoch nur den epistemischen Teil der Prognoseunsicherheit.

In dieser Arbeit entwickeln wir ein neues probabilistisches Modell, genannt Bayes'sche neuronale Netze mit latenten Variablen (BNN+LV). Mithilfe von latenten Variablen kann dieses Modell stochastische Muster in den Daten beschreiben und gleichzeitig die epistemische Unsicherheit mittels einer Wahrscheinlichkeitsverteilung der Netzwerkgewichte modellieren. Wir zeigen wie diese beiden Formen der Unsicherheit aus der Prognoseverteilung des Modells extrahiert werden können und entwickeln neuartige Inspektionsverfahren, um die beobachtete Unsicherheit basierend auf den Eingangsvariablen zu erklären.

Empirisch zeigen wir, dass BNN+LV in vielen Regressionsproblemen über eine hohe Prognosequalität und Unsicherheitsgüte verfügen. Ferner zeigen wir, dass die Zerlegung der Unsicherheit in epistemische und aleatorisch Kompenenten aussagekräftige Ergebnisse liefert und den generativen Prozess modellieren kann. Für die Anwendung in Entscheidungsproblemen untersuchen wir aktive Lernszenarien und zeigen, dass die Nutzung einer Zerlegung der Unsicherheit in BNN+LV zu einer effizienteren Akquisestrategie führt. Im Gebiet des modellbasierten Verstärkungslernen (RL) untersuchen wir, wie BNN+LV als Modelle für stochastische dynamische Systeme verwendet werden können. Hier zeigen wir, dass BNN+LV die Systemdynamiken akkurat modellieren können. Aus dem Modell können Kontrollstrategien abgeleitet werden, die eine bessere Performance vorweisen als Vergleichsmethoden. Auch hier entwickeln wir unter Ausnutzung der Zerlegung der Unsicherheit ein neuartiges Kriterium für risikosensitives RL, das es in der Anwendung ermöglicht eine Risikoabwägung zu spezifizieren, die zwischen Minimierung von Kosten, Vermeidung von Stochastizität und das Risiko eines Modellfehlers abwägt.

Abstract

Supervised learning methods, especially neural networks, are part of many data-driven solutions to real-world problems. While these methods can provide high prediction quality, simply making predictions is often not enough. For decision making, it is also necessary to estimate the confidence, or uncertainty, in the prediction of the model. Here, we can distinguish between epistemic and aleatoric predictive uncertainty. The epistemic part originates from lack of knowledge about the true data-generating function and will reduce the more data we collect. The aleatoric part is irreducible; it is the result of intrinsic randomness, or partial observability, of the data-generating process. Modern Bayesian methods, such as Bayesian neural networks, are scalable black-box tools that combine the universality of neural networks with the principled probabilistic reasoning of Bayesian inference. These methods, however, only model the epistemic part of predictive uncertainty.

In this thesis, we develop a novel probabilistic model, called Bayesian neural networks with latent variables (BNN+LV). This model class can describe complex stochastic patterns in the data via a distribution over latent input variables (aleatoric uncertainty), while, at the same time, account for model uncertainty via a distribution over weights (epistemic uncertainty). We show how these two forms of uncertainty can be extracted from the predictive distribution of the model and develop novel model inspection techniques to explain the observed uncertainty based on the input features.

Empirically, we show that BNN+LV provide high uncertainty quality over a wide range of regression tasks. In a series of model inspection studies, we show that the decomposition of uncertainty provides meaningful results and models the data generating process accurately. For decision making, we study active learning scenarios and show that utilizing a decomposition of uncertainty in BNN+LV leads to a more efficient data acquisition strategy. In model-based reinforcement learning (RL) we study how BNN+LV can be used as approximate models for stochastic dynamic systems. Here, we show that BNN+LV serve as powerful models, that give rise to policies that produce lower costs than baseline methods. Again, utilizing the decomposition of uncertainty we develop a novel risk-sensitive criterion for risk-sensitive RL, that enables a practitioner to specify its preference for policies that minimize costs, avoid stochasticity and minimize the risk for model-bias.

Acknowledgements

I am very grateful to my supervisor Thomas Runkler for his guidance over the years. I appreciate the freedom he has given me to shape my research direction and his support to structure this endeavor. I would like to thank José Miguel Hernández-Lobato for the close cooperation and supervision. He became a supervisor of my PhD very early in the process and provided intensive support and guidance ever since. I would also like to thank Laura Leal-Taixé for joining my PhD project as a supervisor and her feedback about the thesis.

I express my gratitude to Finale Doshi-Velez for her feedback and the successful collaboration that resulted in several publications. For helpful discussions during my time at the Harvard university, I also want to thank David Duvenaud and Matthew Johnson.

At Siemens, I want to thank all my coworkers. Steffen Udluft's expertise in reinforcement learning and industrial problems was a great help for which I am very grateful. I thank Hans-Georg Zimmermann for his guidance and inspiring discussions about neural networks, and what modeling time-series has to do with Salvador Dali. I want to thank Volkmar Sterzing, the head of the research group I worked in, for his support and enabling me to experience both research and industry over the course of my PhD. I want to thank Daniel Hein and Markus Kaiser for the countless discussions about science and life in general.

Lastly, I want to thank Esmeralda Ramić and Esther Gabrovsek for the friendship and support in Munich. Your support made me feel home and inspired me when progress stalled.

Contents

List of Abbreviations

BNN	Bayesian neural network
BNN+LV	Bayesian neural network with latent variables
CDF	Cumulative distribution function
CNN	Convolutional neural network
EP	Expectation propagation
GP	Gaussian process
HMC	Hamiltonian Monte Carlo
IB	Industrial benchmark
MC-VB	Monte Carlo variational Bayes
MCMC	Markov chain Monte Carlo
MDP	Markov decision process
MLP	Multilayer perceptron
POMDP	Partially-observable Markov decision process
RL	Reinforcement learning
RMSE	Root mean squared error
RNN	Recurrent neural network
ROC	Receiver operating characteristic
VB	Variational Bayes
VI	Variational inference

1 Introduction

Supervised learning is one major area of machine learning. Here we are interested in learning the relationship between input ($\mathbf{X}$) and output ($\mathbf{Y}$) variables based on data. In almost all areas of science, we can encounter supervised learning problems. For instance, in medical science we may be interested in detecting cancer cells based on medical images, in natural language processing we may want to translate a sentence from one language into another. Advancements in computational power, as well as data storage capabilities, have greatly increased the applicability of modern supervised learning techniques.

Neural networks form one popular class of algorithms that see widespread practical use, such as in computer vision, natural language processing or reinforcement learning (Krizhevsky et al., 2012; Sutskever et al., 2014; Mnih et al., 2015). Two factors that make neural networks attractive are their scalability and universality: these methods can handle large amounts of data and show competitive performance over a wide area of tasks.

While methods such as neural networks can provide high prediction quality, to achieve this, they are dependent on a sufficient amount of data. If data, however, is limited, perhaps due to limited observations or poor feature quality, we cannot expect an accurate prediction, irrespective of the algorithm used. In this case, there is not enough information in the data set $\mathcal{D}$ that would allow modeling the function of interest accurately. The result of limited information is *uncertainty* over what the correct relationship between inputs and outputs may be.

Ideally, we want a method that can express such uncertainty over its prediction. For instance, when detecting cancer cells the degree of uncertainty in a prediction can play a large part in diagnosis as a misclassification can be potentially harmful. Another example is active learning, where the task is to iteratively select data such that the agent learns the most about the problem at hand. Choosing these data by utilizing uncertainty is a successful and principled strategy (MacKay, 1992).

Bayesian inference provides the calculus to formally reason about and to quantify uncertainty. Specifically, using the methodology from probability theory, we can reason about how likely each model instance is, given data. Such a distribution over model instances can then be used to make predictions and provide uncertainty estimates. While traditionally Bayesian approaches to supervised learning were limited to models with low complexity, modern approximation techniques, such as variational inference, enable scalable inference in complex models. Two prominent examples of this are Gaussian processes (GP) and Bayesian neural networks (BNN). A GP is a non-parametric method that can express multivariate Gaussian uncertainty whereas a BNN is a parametric model that places the flexibility of neural networks in a Bayesian framework (Blundell et al., 2015; Gal, 2016).

These methods, however, are not without limitations. One limitation is that they assume a deterministic relationship between the inputs and the outputs. In many supervised learning problems, this assumption does not hold. The feature set may be limited and thereby insufficient to accurately predict the output $\mathbf{y}$ based on $\mathbf{x}$, irrespective of how much training examples are available. The effect of this are noise patterns in $\mathbf{y}$: for similar inputs, we get varying outputs which can form complex patterns of noise, such as bimodality or heteroscedasticity. The existence of such noise patterns is in literature referred to as *aleatoric* uncertainty: variations in the outcome of the process that occur every time we acquire data (Matthies, 2007; Der Kiureghian and Ditlevsen, 2009). By contrast, uncertainty in Bayesian modeling reasons about the likelihood of each model instance. This form of uncertainty is referred to as *epistemic* uncertainty and is reducible, the more data we collect, the more certainty we acquire about what model instance is the right one.

Preferably, we would like to have a machine learning algorithm that can express both epistemic and aleatoric uncertainty with a high level of generality. For decision-making knowing where a method is uncertain because the data is limited, and knowing where a method is uncertain because of randomness (or partial observability) in the data can be highly beneficial. Following the example given in Senge et al. (2014) (Section 1), "A medical doctor, for example, who knows that his uncertainty about the illness of a patient is caused by a lack of knowledge about the disease in question, may decide to consult the literature or ask a colleague before making a decision." By contrast, if the uncertainty in the diagnosis would be due to partial observability, the doctor may decide to do additional diagnostic tests.

The existing methods can only model one, that is the epistemic form of uncertainty. Both BNNs as well as GPs assume that the underlying ground truth function is deterministic; they are merely uncertain about the form or the parameters of said function. In short, we can identify three desirable properties a supervised learning technique should have: a scalable black-box approach for function approximation, uncertainty-awareness in a Bayesian framework and lastly modeling stochastic effects of the data.

Contributions The centerpiece of this thesis is to extend BNNs with a latent variable model (BNN+LV). The latent variables allow this method to model the noise patterns of the data, while still maintaining uncertainty over its parameters. By that, this method can express both epistemic and aleatoric uncertainty and shares the scalability and universality of standard BNNs. Using this novel method, we then investigate how useful the awareness to these two forms of uncertainty is for decision-making in supervised learning. We further develop methods for model inspection and analysis, to address how to extract and interpret the results of such decomposition of uncertainty. Below we provide a list of the published articles until completion of this thesis:

- Stefan Depeweg, José Miguel Hernández-Lobato, Finale Doshi-Velez, and Steffen Udluft. Learning and policy search in stochastic dynamical systems with Bayesian neural networks. *International Conference on Learning Representations (ICLR)*, 2017a

- Stefan Depeweg, José Miguel Hernández-Lobato, Finale Doshi-Velez, and Steffen Udluft. Uncertainty decomposition in Bayesian neural networks with latent variables. *Workshop on Reliable Machine Learning in the Wild, ICML*, 2017b

- Stefan Depeweg, José Miguel Hernández-Lobato, Steffen Udluft, and Thomas Runkler. Sensitivity analysis for predictive uncertainty in Bayesian neural networks. *European Symposium on Artificial Neural Networks, (ESANN)*, 2017c

- Daniel Hein, Stefan Depeweg, Michel Tokic, Steffen Udluft, Alexander Hentschel, Thomas Runkler, and Volkmar Sterzing. A benchmark environment motivated by industrial control problems. In *IEEE Symposium Series on Computational Intelligence (SSCI)*, 2017a

- Stefan Depeweg, Jose Miguel Hernandez-Lobato, Finale Doshi-Velez, and Steffen Udluft. Decomposition of uncertainty in Bayesian deep learning for efficient and risk-sensitive learning. In *International Conference on Machine Learning (ICML)*, pages 1192–1201, 2018

Thesis Outline We introduce the concept of Bayesian modeling for supervised learning in Chapter 2. We highlight methods for approximate inference and introduce GPs and BNNs, two prominent methods for supervised learning. In Chapter 3 we introduce BNN+LV, a novel Bayesian method that is able to both model the noise in the data and express uncertainty over its parameters. We explain key properties of this model and show how epistemic and aleatoric uncertainties can be decomposed from its predictions (Section 3.2). Here we also present a novel model inspection method to explain the quantities of uncertainties based on the input features (Section 3.3). We introduce data sets and benchmarks in Chapter 4. These include standard regression problems and both toy- and real-world stochastic dynamic systems. We perform a set of regression experiments in Chapter 5 using a broad set of benchmarks and data sets. Here we compare BNN+LV to a set of baselines and investigate both the uncertainty quality and accuracy. In Chapter 6 we perform model inspection studies to better understand what kind of epistemic and aleatoric uncertainties we observe in different kinds of problems and study how reasonable the results of the decomposition of uncertainty are. Here, we also apply the aforementioned method of sensitivity analysis to a broad set of problems (Section 6.3) and consider active learning problems to study temporal decision making (Section 6.4). Chapter 7 studies the applicability of BNN+LV for reinforcement learning. In particular in Section 7.1 we investigate how BNN+LV can be used to approximate dynamic systems for model-based RL. Building on this in Section 7.2 we develop a novel risk criterion for risk-sensitive RL, that is obtained by decomposing risk into epistemic and aleatoric uncertainty.

2 Modeling Uncertainty in Supervised Learning

In this chapter, we introduce the fundamentals of modeling uncertainty in the context of supervised learning. Here, we focus on a Bayesian perspective of uncertainty. We will start with the basics of supervised learning and Bayesian Modeling. Building on this we will discuss the core concepts of variational inference as one form of approximate Bayesian modeling. At the end of this chapter, we will introduce Gaussian processes and Bayesian neural networks, which are two supervised learning methods from the Bayesian modeling framework.

In supervised learning we are given a data set $\mathcal{D} = \{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N = (\mathbf{X}, \mathbf{Y})$ formed by feature vectors $\mathbf{x}_n \in \mathbb{R}^D$ and targets $\mathbf{y}_n$. $\mathbf{Y}$ may be real-valued or categorical. The former is referred to as regression and the latter as a classification problem. We assume there exists an unknown function $\mathbf{y} = f(\mathbf{x})$ that generated the data set.

An algorithm for supervised learning uses the data set $\mathcal{D}$ to make an estimate $\hat{f}$ of the true function. This estimation process based on data is called training, and consequently we refer to $\mathcal{D}$ as the training set. At test time we use $\hat{f}$ to make prediction for test data points $\mathbf{x}_\star \in \mathcal{D}_{\text{test}}$. Depending on the kind of supervised learning problem different error metrics can be used to assess the quality of the estimation. Ideally, the algorithm should generalize, that means it can accurately predict new data points that differ from those that were used during training.

Because the ground-truth function is unknown, the central question of supervised learning is: given the data set $\mathcal{D}$, what estimate $\hat{f}$ is plausible? In many cases, different estimates may seem equally likely. Some which model the data very closely can be complex, while other solutions are simple, but may have lower accuracy. Following Jaynes (2003), Bayesian modeling allows us to express the degree of plausibility over the estimated functions $\hat{f}$ using the framework of probability theory. We can express the plausibility in the form of uncertainty over $\hat{f}$ which we will discuss in the following section.

2.1 Bayesian Modeling

To introduce the concepts of Bayesian modeling we will start with parametric functions of the form $\mathbf{y} = f(\mathbf{x}; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is the set of parameters. At the beginning of the inference process in parametric functions, we need to choose a particular family of functions f, which are parameterized by $\boldsymbol{\theta}$, for instance polynomials, periodic functions, or, as we will see in Section 2.4.3, neural networks. We will study a non-parametric modeling

approach, the Gaussian process (GP), in more detail in Section 2.3. We assume that the data set is independent and identically distributed (i.i.d.).

The central question in Bayesian modeling is: What parameters $\boldsymbol{\theta}$ likely have generated the outputs $\mathbf{y}$ given the inputs $\mathbf{x}$? In Bayesian modeling the two ingredients that answer this question are the *prior* and the *likelihood*. Their combination will result in the well-known Bayes theorem.

Without taking data into account, we may have a belief that some parameter values are more likely than others. For instance, in the case of polynomials, we may prefer coefficients that are close to zero, which will typically induce smoothness in the resulting function. We can formulate this as the *prior* $p_0(\boldsymbol{\theta})$. More complex priors are possible; in fact the process of incorporating prior knowledge in the inference process is one core concept in Bayesian modeling (Jaynes, 2003).

The second ingredient in Bayesian modeling is how well a certain parameter set explains the observed data $(\mathbf{X}, \mathbf{Y})$. We call this the *likelihood* function $p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta}) = \prod_{n=1}^{N} p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta})$, which is a product of individual terms because of the i.i.d assumption of the data set. This function measures how likely the outputs $\mathbf{Y}$ are given the inputs $\mathbf{X}$ and a particular set of parameters $\boldsymbol{\theta}$. Note that the likelihood is a function over $\boldsymbol{\theta}$ and not a distribution because we consider the data set $\mathcal{D}$ to be fixed beforehand.

The likelihood and the prior together form the *posterior* over the parameter given the data, which is Bayes theorem:

$$p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{Y}) = \frac{p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta})p_0(\boldsymbol{\theta})}{p(\mathbf{Y}|\mathbf{X})} \,. \tag{2.1}$$

The denominator in Eq. (2.1) is called the *marginal likelihood* or *evidence*. Formally, it is a scalar because $\mathcal{D}$ is fixed and ensures that the posterior is normalized and by that a proper distribution. The evidence is obtained by integrating, or marginalizing, over all parameter configurations $\boldsymbol{\theta}$:

$$p(\mathbf{Y}|\mathbf{X}) = \int p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta})p_0(\boldsymbol{\theta})d\boldsymbol{\theta} \,. \tag{2.2}$$

We see that the evidence is an integral over the likelihood and prior for all possible parameter configurations of $\boldsymbol{\theta}$. The evidence plays a central part in model selection. In the beginning we stated that in parametric models we have to decide about a particular function class of f. We can use the evidence to compare different classes of functions: using the so-called *Bayes factor* which is

$$K = \frac{p(\mathbf{Y}|\mathbf{X}, M_1)}{p(\mathbf{Y}|\mathbf{X}, M_2)} \,, \tag{2.3}$$

where by M_1 and M_2 we denote the respective model classes we wish to compare.

Given the posterior, we can make predictions by integrating over all possible parameter sets, weighted by their posterior probability. This is called the predictive distribution:

$$p(\mathbf{y}_\star|\mathbf{x}_\star, \mathbf{X}, \mathbf{Y}) = \int p(\mathbf{y}_\star|\mathbf{x}_\star, \boldsymbol{\theta})p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{Y})d\boldsymbol{\theta} \,. \tag{2.4}$$

Eq. (2.4) provides a distribution over output values $\mathbf{y}_\star$ given input features $\mathbf{x}_\star$. In standard supervised learning problems we can take the expected value, that is $E_{\boldsymbol{\theta}}[\mathbf{y}_\star|\mathbf{x}_\star, \boldsymbol{\theta}]$, to do predictions on the test set. Importantly however, a Bayesian modeling approach provides us with a full distribution over possible output values which enables us to model *predictive uncertainty* in the context of supervised learning.

Depending on the application context we can quantify the predictive uncertainty using a variety of metrics. For instance, for safety-critical systems, having minimum and maximum values of possible outputs may be highly useful for worst-case error estimations.

Two metrics that summarize the overall uncertainty in the prediction are the entropy and variance. The differential entropy of a random variable with density $p(x)$ is given by:

$$H(X) = -\int_{\mathbb{X}} p(x) \log p(x)\, dx. \tag{2.5}$$

The entropy is maximal for a uniform distribution, which means that all outcomes are equally likely, and minimal for a delta distribution, where all mass is centered on one particular point. Let $\mathrm{H}(\cdot)$ compute the differential entropy of a probability distribution. The total uncertainty present in Eq. (2.4) can then be quantified as $\mathrm{H}(y_\star|\mathbf{x}_\star)$. While we usually cannot compute the entropy analytically for complicated distributions, the entropy can be approximated using standard estimators based on numerical sampling, such nearest-neighbor methods (Kozachenko and Leonenko, 1987; Kraskov et al., 2004), histograms or kernel density estimators (Beirlant et al., 1997). The variance of a random variable x is given by:

$$Var(X) = \sigma^2 = \int (x-\mu)^2 f(x)\, dx\ . \tag{2.6}$$

The variance measures the average 'spread' of a distribution. If analytical solutions are not available, it can be estimated by numerical sampling using the empirical variance of the samples. Both the variance and the entropy are intimately related, for instance in a Gaussian distribution we have that $H(X) = \frac{1}{2}\log(2\pi e\sigma^2)$ and in general both behave very similarly for unimodal distributions. However, for more complex distributions, especially in the presence of multi-modalities the variance will fail to capture such structural information, as it is not sensitive to multiple modes, whereas the entropy is.

We have described the fundamentals of Bayesian modeling in the context of supervised learning. The concepts lay the groundwork for the methods that follow in this thesis. The next sections will introduce approximate techniques for Bayesian modeling.

2.2 Variational Inference

The fundamental drawback in Bayesian modeling is that there are in general no closed-form solutions to the distributions of interests. This is because the process of integration is often not solvable analytically. For instance, integration appears in the the marginal likelihood that we show in Eq. (2.2) over all possible parameter values $\boldsymbol{\theta}$ and in the predictive distribution in Eq. (2.4). Historically this implied that Bayesian modeling

was restricted to modeling less complex models which resulted in limited practical use. This however changed with recent advancements in approximate inference together with increased computational capabilities. Later in this chapter, we will introduce Bayesian neural networks, a class of complex probabilistic models. There we will see how VI provides computationally feasible approximations for complex models.

Approximate inference can be grouped into two subfields: variational inference (VI) and sampling methods (Bishop, 2007). In this thesis we focus on VI; we will explain the fundamentals and highlight relevant literature in this section. To justify this choice, we will introduce sampling methods in Section 2.2.3 and discuss advantages and disadvantages of both approximate inference techniques.

In VI we want to approximate the potentially intractable posterior $p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{Y})$ by a simpler distribution $q(\boldsymbol{\theta})$ which we call the variational distribution. We can define the variational distribution such that it has some desirable properties, such as obtaining analytical solutions to the main integrals in Bayesian modeling, or to easily obtain samples to form an empirical distribution. This approximation process can be formulated as an optimization problem:

$$q^*(\boldsymbol{\theta}) = \arg\min_q \mathrm{D}[q(\boldsymbol{\theta})||p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{Y})] \,, \tag{2.7}$$

where D measures the discrepancy between q and p. When working with probability densities a discrepancy between two densities is measured by a divergence.

Definition 2.2.1. (Divergence) For any two probability density functions $p, q \in \mathcal{P}$ of random variable $\boldsymbol{\theta}$, a divergence D is a function $\mathcal{P} \times \mathcal{P} \to \mathbb{R}$ such that $\mathrm{D}[p||q] = 0$ if and only if $p = q$ and $\mathrm{D}[p||q] \geq 0$ everywhere else.

The most prominent divergence measure is the *Kullback-Leibler* (KL) divergence, which is widely used in machine learning and statistics. It is given by the following formula:

$$\mathrm{KL}[p||q] = \int p(\boldsymbol{\theta}) \log \frac{p(\boldsymbol{\theta})}{q(\boldsymbol{\theta})} d\boldsymbol{\theta} \,. \tag{2.8}$$

The KL divergence is not symmetric, we have that $\mathrm{KL}[q||p] \neq \mathrm{KL}[p||q]$. The former divergence, that is $\mathrm{KL}[q||p]$, is referred to as *exclusive*, whereas the later, that is $\mathrm{KL}[p||q]$, is referred to as *inclusive* KL divergence. Consider we minimize $\mathrm{KL}[p||q]$ by adjusting the parameters of distribution q. Inside the integral the KL divergence in Eq. (2.8) will increase if $p(\boldsymbol{\theta})$ is high, but $q(\boldsymbol{\theta})$ is small, because it appears in the denominator. This would mean that q does not place probability mass in areas where there exists mass in the true density p. Because of this the approximation q tends to cover all the area where $p(\boldsymbol{\theta}) > 0$, even at the cost of putting mass in areas where $p(\boldsymbol{\theta}) = 0$, thereby being *inclusive*. In contrast, when we minimize $\mathrm{KL}[q||p]$ the quantity inside the integral increases if $q(\boldsymbol{\theta})$ is large and $p(\boldsymbol{\theta})$ is small. This happens if q places mass on areas when there is no mass in the true posterior p. Therefore under $\mathrm{KL}[q||p]$ the variational distribution q will tend to cover only the area, where there is also mass in the true posterior, even at the cost of putting no mass in areas where $p(\boldsymbol{\theta}) > 0$, thus being *exclusive*.

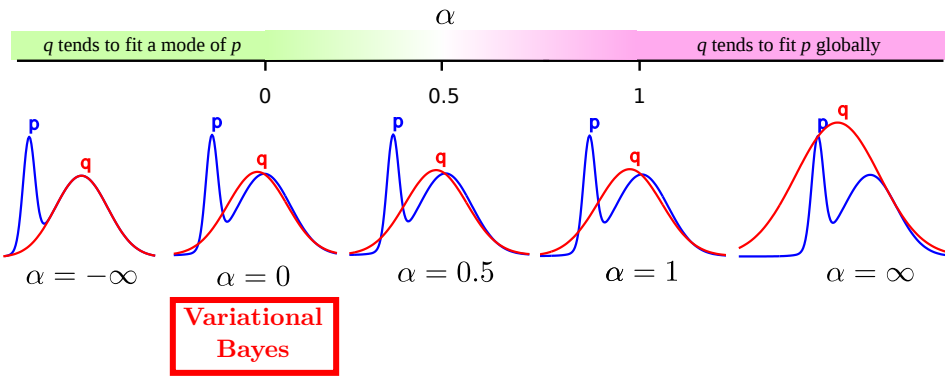


Figure 2.1: Solution for the minimization of the α-divergence between the posterior p (in blue) and the Gaussian approximation q (in red and unnormalized). Figure source Minka (2005).

The α-divergence generalizes over both KL divergences. It has a free parameter α where the two KL divergences are special cases. The α-divergence is given by:

$$\mathrm{D}_\alpha[p||q] = \frac{1}{\alpha(1-\alpha)}\left(1 - \int p(\boldsymbol{\theta})^\alpha q(\boldsymbol{\theta})^{1-\alpha} d\boldsymbol{\theta}\right). \tag{2.9}$$

Here we used the definition given by Tsallis (Tsallis, 1988), there exist alternative formulation of α-divergences, such as Rényi's and Amari's α-divergence (Amari, 1982; Rényi, 1961; Li, 2018). We identify three special cases in the α-divergence:

$$\mathrm{D}_1[p||q] = \lim_{\alpha\to 1} \mathrm{D}_\alpha[p||q] = \mathrm{KL}[p||q]\,, \tag{2.10}$$

$$\mathrm{D}_0[p||q] = \lim_{\alpha\to 0} \mathrm{D}_\alpha[p||q] = \mathrm{KL}[q||p]\,, \tag{2.11}$$

$$\mathrm{D}_{\frac{1}{2}}[p||q] = 2\int \left(\sqrt{p(\boldsymbol{\theta})} - \sqrt{q(\boldsymbol{\theta})}\right)^2 d\boldsymbol{\theta} = 4\mathrm{Hel}^2[p||q]\,. \tag{2.12}$$

We see that when α approaches 1 the inclusive and when α approaches 0 the exclusive KL divergence is obtained. In Eq. (2.12) $\mathrm{Hel}^2[p||q]$ is the *Hellinger distance* between two distributions, it is the only α-divergence that is symmetrical in p and q.

Figure 2.1 illustrates the properties of the three special cases of α-divergences for the one-dimensional case. This figure shows an approximation of a Gaussian mixture with a Gaussian distribution. When $\alpha \geq 1$, q tends to cover the whole posterior distribution p. When $\alpha \leq 0$, q tends to fit a local mode in p. The value $\alpha = 0.5$ is expected to achieve a balance between these two tendencies.

Depending on the divergence measure used, different approximation schemes exist. The most general is arguably one that can minimize arbitrary α-divergences which will be the subject of Section 2.2.2. In the next section we will derive the variational minimization of the exclusive KL divergence. This particular divergence is most commonly used in the field of VI, because of certain mathematical properties that simplify the derivation.

2.2.1 Variational Bayes

Variational Bayes (VB) is a technique of VI that uses the exclusive KL divergence as divergence measure. The technique was proposed at the beginning of the new century (Attias, 1999; Ghahramani and Beal, 2000) and has seen increasing interest in the recent literature for complex models in supervised and unsupervised learning (Kingma and Welling, 2013; Blundell et al., 2015; Blei et al., 2017).

Let Z be the short-hand of the normalization constant $p(\mathbf{Y}|\mathbf{X})$ and $\tilde{p}(\boldsymbol{\theta})$ be the unnormalized posterior distribution, such that $\tilde{p}(\boldsymbol{\theta}) = p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta})p_0(\boldsymbol{\theta})$. We can rewrite the minimization problem when using the exclusive KL divergence as:

$$
\begin{aligned}
q^*(\boldsymbol{\theta}) &= \arg\min_q \mathrm{KL}[q(\boldsymbol{\theta})||p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{Y})] && (2.13)\\
&= \arg\min_q \left(\mathrm{KL}[q(\boldsymbol{\theta})||p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{Y})] - \log Z\right) && (2.14)\\
&= \arg\min_q \int q(\boldsymbol{\theta}) \log \frac{q(\boldsymbol{\theta})}{\tilde{p}(\boldsymbol{\theta})} d\boldsymbol{\theta} && (2.15)\\
&= \arg\max_q \int q(\boldsymbol{\theta}) \log \frac{\tilde{p}(\boldsymbol{\theta})}{q(\boldsymbol{\theta})} d\boldsymbol{\theta} && (2.16)\\
&= \arg\max_q E_{q(\boldsymbol{\theta})}[\log \tilde{p}(\boldsymbol{\theta})] + H(q(\boldsymbol{\theta})) && (2.17)\\
&= \arg\max_q \underbrace{E_{q(\boldsymbol{\theta})}[\log p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta})] - KL[q(\boldsymbol{\theta})||p_0(\boldsymbol{\theta})]}_{\mathcal{L}_{\mathrm{VB}}} . && (2.18)
\end{aligned}
$$

In Eq. (2.14) we added $\log Z$ to the right-hand side of the equation, this is valid, because the normalization constant is independent of the variational distribution q and therefore the addition of this term will not affect the minimization problem. In literature Eq. (2.16) is referred to as *variational free energy* , and the result of our derivation given by Eq. (2.18) is called the *variational lower bound*. The variational lower bound gets its name from the fact that is is a lower bound to the logarithm of the evidence. We can derive it alternatively starting with the evidence of the posterior we wish to approximate:

$$
\begin{aligned}
\log Z &= \log \int p(Y|\mathbf{X}, \boldsymbol{\theta}) p_0(\boldsymbol{\theta}) d\boldsymbol{\theta} && (2.19)\\
&= \log \int \tilde{p}(\boldsymbol{\theta}) d\boldsymbol{\theta} && (2.20)\\
&= \log \int q(\boldsymbol{\theta}) \frac{\tilde{p}(\boldsymbol{\theta})}{q(\boldsymbol{\theta})} d\boldsymbol{\theta} && (2.21)\\
&\geq \int q(\boldsymbol{\theta}) \log \frac{\tilde{p}(\boldsymbol{\theta})}{q(\boldsymbol{\theta})} d\boldsymbol{\theta} , && (2.22)
\end{aligned}
$$

where in the last step we used Jensens's inequality for concave functions.

Theorem 2.2.1 (Jensen's inequality)**.** *For any concave function f:*

$$E[f(X)] \leq f(E[X]) .$$

The relationship between $\mathcal{L}_{\text{VB}}$ and the evidence is relevant for two reasons. As we pointed out in Section 2.1, the evidence integrates over the likelihood and prior for all configurations of parameter $\boldsymbol{\theta}$. The evidence, therefore, depends on the hyper-parameters, the assumptions we make about the model at hand, which can be used for model selection. Secondly, if we can include the hyper-parameters of our model in the maximization process of $\mathcal{L}_{\text{VB}}$ itself, we can potentially increase the evidence of the model. This is because an increase in the lower bound can only increase, but never decrease the evidence. This process is called type II maximum likelihood (or empirical Bayes): whereas in classical maximum likelihood we want to find the parameters $\boldsymbol{\theta}$ for which $p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta})$ is maximal, in type II maximum likelihood we want to find the hyper-parameters for which the evidence Z is maximal.

The derivations leading to Eq. (2.18) have simplified the problem greatly. Earlier we pointed out that the main problem in Bayesian modeling is solving the integrals, specifically integrating over the posterior $p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{Y})$ that appears both in the evidence, as well as in the predictive distribution. In contrast the lower bound in Eq. (2.18) only contains an expectation and entropy term with respect to the variational distribution $q(\boldsymbol{\theta})$. We recall that the main idea in VI is to choose a variational distribution q with desirable properties: for instance, if we choose a Gaussian distribution, the entropy can be calculated in closed form.

For practical applications, two issues are remaining. First,the expectation over q may not be solvable analytically. Here, we can approximate it via Monte Carlo by taking K samples $\theta_1, \ldots, \theta_K \sim q(\boldsymbol{\theta})$. We can then substitute the expectation by an empirical estimation. Second, assume that we cannot solve the maximization step analytically, but require an iterative optimization process, such as gradient ascent with respect to the parameters of the variational distribution. The log-likelihood $\log p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta})$ that appears in Eq. (2.18) is a sum over all available data. This would require us to re-evaluate the current estimate of the parameters of $q(\boldsymbol{\theta})$ over the full data set, which can be expensive. Instead, we can use mini-batches, a standard technique in optimization, where we do not use the full data set, but instead only consider a small subset $\mathcal{S}$ in each iteration. This subset is chosen randomly at each iteration step. These adjustments give rise to the so-called Monte Carlo variational Bayes (MC-VB) approach (Ranganath et al., 2014), the Monte Carlo estimator is given by:

$$\mathcal{L}_{\text{VB}}^{\text{MC}} = \frac{N}{|\mathcal{S}|} \sum_{n \in \mathcal{S}} \frac{1}{K} \sum_{k=1}^{K} \log p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta}_k) - KL[q(\boldsymbol{\theta})||p_0(\boldsymbol{\theta})] \,. \tag{2.23}$$

In principle any optimization method can be used to maximize Eq. (2.23). One approach is to perform gradient ascent with respect to the parameters ϕ of the variational

distribution $q(\boldsymbol{\theta})$. We can estimate the gradient in the following way:

$$\nabla_\phi \mathcal{L}_{\text{VB}} = \nabla_\phi E_{q(\boldsymbol{\theta})}[\log p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta})] - \nabla_\phi KL[q(\boldsymbol{\theta})||p_0(\boldsymbol{\theta})] \tag{2.24}$$

$$= E_{q(\boldsymbol{\theta})}[\log p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta}) \nabla_\phi \log q(\boldsymbol{\theta}|\phi)] - \nabla_\phi KL[q(\boldsymbol{\theta})||p_0(\boldsymbol{\theta})] \tag{2.25}$$

$$\approx \frac{1}{K} \sum_{k=1}^{K} \frac{N}{|\mathcal{S}|} \sum_{n \in \mathcal{S}} \log p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta}_k) \nabla_\phi \log q(\boldsymbol{\theta}_k|\phi)] - \nabla_\phi KL[q(\boldsymbol{\theta})||p_0(\boldsymbol{\theta})] \ , \tag{2.26}$$

where first step leading to Eq. (2.25) can be obtained with a few steps of algebra (see Eq. (5) in Paisley et al. (2012)). These are the same steps of algebra that give rise to the policy gradient theorem in reinforcement learning (see Section 7.1.2 for more details on this). However, this approach leads to very high variance and difficult optimization, because each gradient is obtained by a random sample from q. A lower variance estimator can be obtained using the so-called local reparameterization trick (Kingma and Welling, 2013). For certain distributions we can decompose the sampling process $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K \sim q(\boldsymbol{\theta})$ into a deterministic and stochastic part. For instance, let us assume q is a multivariate Gaussian with diagonal covariance matrix $q(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$ such that $\phi = (\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$, then

$$\boldsymbol{\theta}_k \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2) = \underbrace{\boldsymbol{\epsilon} \circ \boldsymbol{\sigma} + \boldsymbol{\mu}}_{g(\boldsymbol{\epsilon}, \phi)} \ , \tag{2.27}$$

where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$. We can then rewrite

$$\nabla_\phi E_{q(\boldsymbol{\theta})}[f(\boldsymbol{\theta})] = E_{p(\boldsymbol{\epsilon})}[\nabla_\phi f(g(\boldsymbol{\epsilon}, \phi))] \tag{2.28}$$

$$\approx \frac{1}{K} \sum_{k=1}^{K} \nabla_\phi f(g(\boldsymbol{\epsilon}_k, \phi)) \ , \tag{2.29}$$

where we used $f(\cdot)$ as a shorthand for the terms inside the expectation of Eq. (2.25). With this reparameterization we have achieved that each gradient $\nabla_\phi f(g(\boldsymbol{\epsilon}_k, \phi))$ depends on the variational parameters ϕ in a deterministic way, whereas the randomness induced by the estimation of $E_{p(\boldsymbol{\epsilon})}$ is unrelated to the parameters ϕ of interest. This can result in a gradient estimator with significantly lower variance than the one shown in Eq. (2.26) (Kingma and Welling, 2013).

2.2.2 α-divergence Minimization

In the last section we showed that when using the exclusive KL divergence, we obtain a solution were efficient approximation techniques exist. Unfortunately, when using α-divergences the derivation are not as straightforward. To get there, we will start with an alternative method for variational inference, called power expectation propagation (power EP) (Minka, 2004), where, with certain simplifying assumptions, we can derive energy function for α-divergence minimization that shares some similarities with the MC-VB in Eq. (2.23).

Power EP is a generalization of expectation propagation (EP) and very similar in its structure: Classical EP will minimize the inclusive KL divergence $KL(p||q)$, whereas power EP extend this towards general α-divergences. This also implies that the exclusive KL-divergence seems to be a special case, where approximate inference simplifies, whereas for general α-divergences, including the inclusive KL divergence, this does not seem to be the case.

For much of the derivations in this section, we will follow Hernández-Lobato et al. (2016). As stated above we assume that the true posterior is given by a product of likelihood terms $p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta}) = \prod_{n=1}^{N} p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta})$ (which originated from the i.i.d assumption) in addition to a prior. Note that power EP is applicable to more complicated factorization of p: we are given some factorization of p in a composition of functions.

Let us further assume that the prior $p_0(\boldsymbol{\theta}) = \exp\{\mathbf{s}(\boldsymbol{\theta})^T \boldsymbol{\lambda}_0 - \log Z(\boldsymbol{\lambda}_0)\}$ belongs to the family of exponential distributions, where $\boldsymbol{\lambda}_0$ and $\mathbf{s}(\boldsymbol{\theta})$ are vectors of natural parameters and sufficient statistics and by $Z(\boldsymbol{\lambda}_0)$ we denote the normalization constant, or partition function, of the prior, ensuring it is a probability distribution. A large set of basic probability distributions, including the Bernoulli, Poisson or Gaussian distribution can be expressed as members of the exponential family. For instance, in a one dimensional Gaussian distribution we have that $\boldsymbol{\lambda} = (\mu/\sigma^2, -1/2\sigma^2)$ are the natural parameters and $\mathbf{s}(\boldsymbol{\theta}) = (\theta, \theta^2)$ are the sufficient statistics.

In power EP we start by forming a variational distribution. In contrast to variational Bayes we approximate each of the n-th likelihood factors $p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta})$ by having an explicit counterpart f_n in the variational distribution q. We define the variational distribution as a set of site approximations from the same exponential family as the prior $p_0(\boldsymbol{\theta})$:

$$f_n(\boldsymbol{\theta}) = \exp\{\mathbf{s}(\boldsymbol{\theta})^T \boldsymbol{\lambda}_n\} \tag{2.30}$$

The variational distribution is then the product of all site approximations with the prior

$$q(\boldsymbol{\theta}) \propto \prod_{n=1}^{N} \exp\{\mathbf{s}(\boldsymbol{\theta})^T \boldsymbol{\lambda}_n\} \exp\{\mathbf{s}(\boldsymbol{\theta})^T \boldsymbol{\lambda}_0\} \tag{2.31}$$

$$= \exp\{\mathbf{s}(\boldsymbol{\theta})^T (\sum_{n} \boldsymbol{\lambda}_n + \boldsymbol{\lambda}_0)\} \; . \tag{2.32}$$

We have a set of N site approximations and a set of N likelihood factors of the distribution we wish to approximate. One straightforward approach would be to optimize each site approximation independently and in parallel. This process however would result in a poor approximation of the full posterior $p(\mathcal{W}|\mathbf{X}, \mathbf{Y})$ (Bishop, 2007). While power EP updates each site approximation sequentially, it does so by taking into account the current state of the full approximation.

As a shorthand we will use $\boldsymbol{\lambda}_q$ for the natural parameters of $q(\boldsymbol{\theta})$. To approximate the target distribution p with the variational distribution q power EP uses a message passing algorithm. Message passing (or belief propagation) is a class of algorithms from the field of graphical models that work by iteratively updating nodes in a local way, based on incoming information. The algorithm repeatedly applies the following four steps for every site approximation f_n:

1 Compute the cavity distribution: $q^{\setminus n}(\boldsymbol{\theta}) \propto q(\boldsymbol{\theta})/f_n(\boldsymbol{\theta})^\alpha$, in terms of natural parameters: $\boldsymbol{\lambda}^{\setminus n} \leftarrow \boldsymbol{\lambda}_q - \alpha\boldsymbol{\lambda}_n$;

2 Compute the "tilted" distribution by inserting the exact likelihood factor raised to the power α: $\tilde{p}_n(\boldsymbol{\theta}) \propto q^{\setminus n}(\boldsymbol{\theta})p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta})^\alpha$;

3 Adjust q by matching moments: $E_q[\boldsymbol{s}(\boldsymbol{\theta})] \leftarrow E_{\tilde{p}_n}[\boldsymbol{s}(\boldsymbol{\theta})]$;

4 Recover the site approximation $f_n(\boldsymbol{\theta})$ by setting $\boldsymbol{\lambda}_n \leftarrow \boldsymbol{\lambda}_q - \boldsymbol{\lambda}^{\setminus n}$, and compute the final update for $q(\boldsymbol{\theta})$ by $\boldsymbol{\lambda}_q \leftarrow \sum_n \boldsymbol{\lambda}_n + \boldsymbol{\lambda}_0$.

The *cavity* distribution in step 1 is the original variational distribution with a fraction of α of one site approximation removed (an alternative description is leave-one-out distribution). In step 2 we insert the corresponding likelihood factor $p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta})$ of the original distribution into the cavity distribution, where the parameter α again determines the fraction of the factor. In result the first two steps replaced a fraction of one site approximation with the original likelihood factor, giving the *tilted* distribution. In step 3 we then project the variational distribution q on the tilted distribution by matching moments. In the final step we recover the adjusted site approximation f_n from the projected q, giving us the update for the site approximation f_n.

The moment matching in step 3 can be shown to minimize the α-divergence from every tilted distribution to the variational distribution, by zeroing the gradient of the α-divergence with respect to the natural parameters (Minka, 2004; Bishop, 2007). In contrast to variational Bayes (Eq. (2.18)) however only a local α-divergence is minimized, because in each step only one factor is included.

Power EP has a corresponding energy function, which was derived in Minka (2004); Seeger (2005):

$$\begin{aligned} E(\boldsymbol{\lambda}_0, \{\boldsymbol{\lambda}_n\}) &= \log Z(\boldsymbol{\lambda}_0) + \left(\frac{N}{\alpha} - 1\right) \log Z(\boldsymbol{\lambda}_q) \\ -\frac{1}{\alpha}\sum_{n=1}^{N} \log \int p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta})^\alpha \exp\{\mathbf{s}(\boldsymbol{\theta})^T(\boldsymbol{\lambda}_q - \alpha\boldsymbol{\lambda}_n)\} d\boldsymbol{\theta}\,. \end{aligned} \tag{2.33}$$

The energy function however, is not yet suitable for scalable approximation. This is because we need to store the natural parameters for each site approximation, to calculate the respective cavity distributions. In large data settings, this leads to prohibitive memory requirements.

To obtain a scalable energy function like the variational bound from Eq. (2.18) an approach to use is factor tying constraint as proposed by Li et al. (2015); Dehaene and Barthelmé (2018). The main idea is that instead of having one set of natural parameters $\boldsymbol{\lambda}_n$ for each site approximation f_n, the parameters are shared (tied) across all f_n. This

leads to a great simplification of the relevant terms:

$$\begin{aligned} \boldsymbol{\lambda}_n = \boldsymbol{\lambda} \quad \forall n \in \{1, ..N\} \, , & \qquad \text{(constrain natural parameters)} \\ \boldsymbol{\lambda}^{\backslash n} = (N - \alpha)\boldsymbol{\lambda} + \boldsymbol{\lambda}_0 \, , & \qquad \text{(cavity distribution)} \\ \boldsymbol{\lambda}_q = N\boldsymbol{\lambda} + \boldsymbol{\lambda}_0 & \qquad \text{(parameters of approximate posterior)} \\ f_n(\boldsymbol{\theta}) = f(\boldsymbol{\theta}) = \exp\{\boldsymbol{s}(\boldsymbol{\theta})^T \boldsymbol{\lambda}\} \, . & \qquad \text{(average site approximation)} \end{aligned}$$

Factor tying can be seen as performing power EP but instead of having N site approximation, we only have a single one for all likelihood factors to approximate. Under the factor tying constraint the energy function now simplifies to:

$$E(\boldsymbol{\lambda}_0, \boldsymbol{\lambda}) = \log Z(\boldsymbol{\lambda}_0) - \log Z(\boldsymbol{\lambda}_q) - \frac{1}{\alpha} \sum_{n=1}^{N} \log \mathbf{E}_q \left[\left(\frac{p(\mathbf{y}_n | \mathbf{x}_n, \boldsymbol{\theta})}{f(\boldsymbol{\theta})} \right)^{\alpha} \right] . \tag{2.34}$$

Eq. (2.34) has now similar desirable properties as the variational lower bound we have shown in Eq. (2.18). We can now apply the same two optimization steps we have outlined in the previous section: enabling mini-batch training and allowing gradient descent by approximating the expectation by Monte Carlo and using the reparameterization trick. This leads to the stochastic estimate of the energy function, which is referred to as black-box α:

$$\hat{E}(\boldsymbol{\lambda}_0, \boldsymbol{\lambda}_q) = \log Z(\boldsymbol{\lambda}_0) - \log Z(\boldsymbol{\lambda}_q) - \frac{1}{\alpha} \frac{N}{|\mathbf{S}|} \sum_{n \in \mathbf{S}} \log \frac{1}{K} \sum_{k} \left(\frac{p(\mathbf{y}_n | \mathbf{x}_n, \boldsymbol{\theta}_k)}{f(\boldsymbol{\theta}_k)} \right)^{\alpha} . \tag{2.35}$$

However, we note that the approximation in Eq. (2.35) will be biased. This is because the expectation in Eq. (2.34) is inside of the logarithm, and because the logarithm is a nonlinear function, the Monte Carlo estimate will underestimate the true energy. This bias will decrease, the more samples K we use in the approximation. Similar as with the lower bound, this objective licenses optimization by gradient descent. In Hernández-Lobato et al. (2016) a study was performed to investigate the bias in the gradient of this estimator. The main finding is that when using 15 samples or more, the bias in the gradient becomes neglectable.

More recently, the work in Li and Gal (2017) derived a further simplification of Eq. (2.35). The result is a new lower bound of the form:

$$\mathcal{L}_{\alpha}^{\mathrm{MC}} = \frac{1}{\alpha} \frac{N}{|\mathcal{S}|} \sum_{n \in \mathcal{S}} \log \frac{1}{K} \sum_{k=1}^{K} p(\mathbf{y}_n | \mathbf{x}_n, \boldsymbol{\theta}_k)^{\alpha} - KL[q(\boldsymbol{\theta}) || p_0(\boldsymbol{\theta})] \, . \tag{2.36}$$

2.2.3 Comparison between VI and Sampling Methods

In the previous sections, we introduced in greater detail VI as a technique for approximate inference. An alternative approach to VI is to use sampling methods. We will now introduce its fundamentals and then discuss advantages and disadvantages of both approximation techniques.

One way to address the problem of integration in Bayesian modeling is by numerical sampling. Assume that we are unable to derive the posterior in Eq. (2.1) in analytical form, however if we can draw samples $\boldsymbol{\theta}_k \sim p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{Y})$ from the posterior, we can utilize the resulting empirical distribution to estimate all quantities of interest, such as the expected value of the predictive distribution:

$$E_{\boldsymbol{\theta}}[\mathbf{y}_\star|\mathbf{x}_\star, \boldsymbol{\theta}] \approx \frac{1}{K}\sum_{k=1}^{K} \mathbf{y}_\star|\mathbf{x}_\star, \boldsymbol{\theta}_k \ . \tag{2.37}$$

Sampling methods are a class of methods to obtain the aforementioned empirical distribution. While computing the posterior of a particular value of $\boldsymbol{\theta}$ is typically infeasible due to the normalization constant in Eq. (2.1), the unnormalized posterior

$$\tilde{p}(\boldsymbol{\theta}) = p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta})p_0(\boldsymbol{\theta}) \tag{2.38}$$

usually can be evaluated for a particular value of $\boldsymbol{\theta}$.

Numerous methods exist that utilize this property, such as rejection sampling and Markov chain Monte Carlo (MCMC) methods. In rejection sampling we define a proposal distribution $q(\boldsymbol{\theta})$ from which we can draw a sample $\boldsymbol{\theta}_k$ in an iterative process. We accept this sample if

$$u < \frac{\tilde{p}(\boldsymbol{\theta}_k)}{Mq(\boldsymbol{\theta}_k)} \ , \tag{2.39}$$

where $u \sim \mathcal{U}(0, 1)$ and otherwise reject it. M is a hyper-parameter of this method and it must hold that $\tilde{p}(\boldsymbol{\theta}) \leq Mq(\boldsymbol{\theta})$ for all values of $\boldsymbol{\theta}$. On average it will take M iterations until a sample is accepted.

Methods such as rejection sampling however are severely limited when $\boldsymbol{\theta}$ is high-dimensional, which is often the case in supervised learning (Bishop, 2007). This is because $Mq(\boldsymbol{\theta})$ must be an upper bound to the target distribution and $q(\boldsymbol{\theta}) > 0$ whenever $\tilde{p}(\boldsymbol{\theta}) > 0$. To satisfy this M will increase leading to more and more rejections as the dimensionality increases.

MCMC is an additional class of sampling methods that is able to work in high-dimensional problems. The main idea is to again utilize the unnormalized posterior $\tilde{p}(\boldsymbol{\theta})$ but use a Markov chain $q(\boldsymbol{\theta}_t|\boldsymbol{\theta}_{t-1})$ as a proposal distribution. This chain can be designed in such a way that its stationary distribution converges to the desired posterior distribution $p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{Y})$. Because of this MCMC methods are random walks, where the next proposed sample $\boldsymbol{\theta}_{t+1}$ depends on the current sample $\boldsymbol{\theta}_t$. Examples of MCMC methods are Metropolis-(Hastings), Gibbs sampling and Hamiltonian Monte Carlo (HMC). HMC in particular has seen widespread use, the main advantage of this method is that it utilizes gradient information in the proposal distribution (Brooks et al., 2011).

What approximate inference technique should we use? The following list compares a set of properties in VI and sampling methods.

1. Bias: Sampling methods are asymptotically unbiased and can therefore produce more accurate posterior approximations than variational approaches providing

enough computation time is available. Variational methods are by contrast biased: the more the variational distribution $q(\boldsymbol{\theta})$ and the true posterior $p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{Y})$ differ, the higher this bias will be. Often times, when working with complicated models, $q(\boldsymbol{\theta})$ will need to have a greatly simplified structure compared to $p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{Y})$ with potentially large bias.

2. Computational cost: There is a higher computational cost when working with sampling methods. This is because of the random walk process: for each sample the likelihood and prior needs to be evaluated to estimate the unnormalized posterior probability. Furthermore, each region in parameter space $\boldsymbol{\theta}$ has to be visited numerous times before and after convergence of the chain to approximate the density.

3. Parameter sensitivity: Sampling methods usually include many hyper-parameters that are highly data and model dependent and that require fine-tuning. For instance, which proposal distribution to use for the random walk process or how long the burn-in process should be until it is assured that the chain converged. Each algorithm again has its own hyper-parameters and the convergence process is highly sensitive for this.

4. Convergence: In sampling methods it is very difficult to determine when it has converged and is drawing samples from the correct stationary distribution. By contrast, it is easy to determine when a variational approach has converged. For example, when the energy function in blackbox-α in Eq. (2.34) does not improve any more beyond a specific threshold.

5. Adaptability: In variational methods the posterior approximation is compactly represented by a collection of parameters. These parameters can be easily updated when new data are available. Updating the samples generated by sampling methods is by contrast very challenging.

2.3 Gaussian Processes

In Section 2.1 we have introduced the concept of Bayesian modeling in the context of parametric models. One drawback of parameterized models is that we have to define prior knowledge in parameter space $\boldsymbol{\theta}$ as opposed to function space f. For instance, let us assume we want to use a prior that prefers functions that are smooth and slow-changing. Depending on the parametric model used, it is not obvious how to formulate such a prior belief in the space of $\boldsymbol{\theta}$.

Gaussian processes belong to the family of non-parametric models. The central idea is that given a sequence of inputs $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ we are interested in the joint distribution of output values $\{f(\mathbf{x}_1), \ldots, f(\mathbf{x}_N)\}$, without reasoning about any parameterization of f. Considering distributions over arbitrary functions f is infeasible in practice, however evaluating such a distribution on a finite set of points what makes this process feasible.

A multivariate Gaussian distribution $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is defined over a d dimensional space, the dimensionality of $\mathbf{x}$. The main idea of GPs is to generalize this to infinitely many dimensions. For this we define a stochastic process, a function that maps from an index t to a random variable $\mathbf{X}_t$. For a Gaussian process, a particular stochastic process, it holds that for every subset T of indexes, the resulting random variables $\{\mathbf{X}_{t\in T}\}$ are jointly Gaussian distributed.

In one-dimensional regression, we are interested in learning a function $y = f(\mathbf{x})$. The aforementioned indeces are now a set of $\mathbf{x}_i$ and in a GP we assume that all subsets of outputs $\{f(\mathbf{x}_1), \ldots, f(\mathbf{x}_N)\}$ are jointly Gaussian distributed. A multivariate Gaussian is completely described by the second-order statistics, the mean vector $\boldsymbol{\mu}$ and a covariance matrix $\boldsymbol{\Sigma}$. In order to define these statistics for arbitrary subsets in input space we define a mean function $\mu_f(\mathbf{x})$ and a *kernel*, or covariance function, $\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) = \mathrm{cov}[f(\mathbf{x}_i), f(\mathbf{x}_j)]$. For any particular set of of $\mathbf{x}$ applying the mean function and the covariance function for every element of the covariance matrix will define the full multivariate Gaussian.

In a GP a standard approach is to assume a mean function of constant zero. Defining the kernel then defines the behavior of the GP completely. For a kernel to be valid it has to hold that for any finite set of random variables their covariance matrix, which can be obtained by applying the kernel function, is positive definite. Positive definiteness of a matrix M means that for every vector $\mathbf{x}$ it holds that $\mathbf{x}^T M \mathbf{x} > 0$.

By choosing a kernel we can implement the assumptions we make about f. This is because the kernel defines the metric to measure similarity between two data points $(\mathbf{x}_i, \mathbf{x}_j)$. For instance, when we have data that appears to be periodic with varying frequency, we may want to use a different measure of similarity than in structured data in the form of trees. As we will see, the kernel plays a central role for making predictions on test data $\mathbf{x}_\star$. In a GP the predictions of $\mathbf{x}_\star$ are influenced by the similarity (defined by the kernel) to the known training data. One important property of kernels is if they are stationary or non-stationary: in a stationary kernel we have that $\mathcal{K}(\mathbf{x}, \mathbf{y}) = \mathcal{K}(\mathbf{x}-\mathbf{y})$. Therefore stationary kernels are translation invariant, the closeness between two points is given solely by their relative, and not absolute, position to another. For instance, the most classical and widely-used kernel, the radial-basis function kernel (RBF) is a stationary kernel of the form:

$$k_{\mathrm{RBF}}(\mathbf{x}_i, \mathbf{x}_j) = \sigma^2_{\mathrm{RBF}} \exp\left(-\frac{(\mathbf{x}_i - \mathbf{x}_j)^2}{2\ell^2}\right) , \tag{2.40}$$

where σ^2_{RBF} and ℓ^2 are hyper-parameters of this kernel. By specifying a mean and kernel function, we have defined our model, a Gaussian process prior $f \sim \mathcal{GP}(\mathbf{0}, \mathcal{K})$. By specifying a set of points $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ we can obtain predictions from the GP prior by sampling from $\mathcal{N}(\mathbf{0}, \mathbf{K}_N)$ where $\mathbf{K}_N$ is the covariance matrix obtained by applying the kernel function for all pairs of $(\mathbf{x}_i, \mathbf{x}_j)$.

In regression we are given a data set $\mathcal{D} = \{\mathbf{x}_n, y_n\}_{n=1}^N = \{\mathbf{X}, \mathbf{y}\}$ formed by feature vectors $\mathbf{x}_n \in \mathbb{R}^D$ and targets y_n. We assume that the data was generated by a function $y_n = f_{\mathrm{true}}(\mathbf{x}_n) + \mathcal{N}(\mathbf{0}, \sigma^2)$, where f_{true} is an (unknown) sample from the GP prior. We further assume that our observations are corrupted by independent Gaussian noise with

unknown noise level σ^2, which is a hyper-parameter in our inference process. Using the GP prior we want to incorporate the data set to obtain a posterior GP. This posterior has a closed form solution, for a derivation we refer to Rasmussen (2004). The GP posterior is given by:

$$\mu_{\text{post}}(\mathbf{x}) = \mathcal{K}(\mathbf{x}, \mathbf{X})(\mathbf{K}_N + \sigma^2\mathbf{I})^{-1}\mathbf{y} \tag{2.41}$$

$$\mathcal{K}_{\text{post}}(\mathbf{x}_i, \mathbf{x}_j) = \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) - \mathcal{K}(\mathbf{x}_i, \mathbf{X})(\mathbf{K}_N + \sigma^2\mathbf{I})^{-1}\mathcal{K}(\mathbf{X}, \mathbf{x}_j)\ , \tag{2.42}$$

where $\mathcal{K}(\mathbf{x}, \mathbf{X})$ is a $1 \times N$ vector of applying the kernel function of $\mathbf{x}$ with all elements of the training data in $\mathcal{D}$ and similarly $\mathbf{K}_N$ is a $N \times N$ matrix of applying the kernel function for all pairs of the training set.

We note that $(\mathbf{K}_N + \sigma^2\mathbf{I})^{-1}\mathbf{y}$ only needs to be computed once, it is constant w.r.t. making a prediction for $\mathbf{x}$. Nevertheless, we need a pass through all training data to compute the mean which is $O(n)$ in time complexity and $O(n^2)$ to compute the predictive variance. On the other hand, the basic GP as in the above equation does not need any "training" like other machine learning methods, because predictions and uncertainty estimates can be obtained in closed form. Having said that the model does have hyper-parameters that one may want to optimize. First, the observation noise σ^2 that appears in the predictive posterior and secondly, possible hyper-parameters in of the kernel function. The standard way to do this is type II maximum likelihood using the model evidence, a process we already discussed in Section 2.2 in the context of variational inference.

Because of the aforementioned time complexity it can be computationally prohibitive to apply GPs to large data sets, in practice GPs work with hundreds, but not thousands of data points. A recent method to overcome this is to use sparse pseudo input GPs (SPGP) (Snelson and Ghahramani, 2006). Here, instead of working on the original data set $\mathcal{D}$ the method builds a pseudo data set $\hat{\mathcal{D}}$ of size n, where n are the number of inducing points. These positions of pseudo data points are then additional hyper-parameters which can be optimized using type II maximum likelihood.

Improving and extending GPs is an active area of research with many different sub-fields. Applications include computer vision domains (He and Siu, 2011), reinforcement learning (Deisenroth and Rasmussen, 2011), or, with the recent interest in composite (deep) architectures, deep GPs (Hensman and Lawrence, 2014; Kaiser et al., 2017). While GPs work well in practice with default settings, such as using a radial-basis kernel and standard optimization techniques, the two key advantages this method has is that it is a principled method for Bayesian modeling and it is much easier to incorporate prior knowledge into the inference process than in parametric models.

2.4 Modeling Uncertainty in Neural Networks

In this section, we review existing approaches on how to model uncertainty in neural networks for supervised learning. In particular we investigate three approaches: How to estimate predictive uncertainty in standard neural networks, how to utilize ensembles of

neural networks to model predictive uncertainty and how to use neural networks in a Bayesian framework approach that we introduced in Section 2.1.

2.4.1 Neural Networks

Neural Networks are popular parametric models for machine learning. Influenced by early research in neuroscience, a neural network consists of connected processing units, called neurons in analogy and inspired by the type of cells in the brain (Rosenblatt, 1958; Hebb, 1949). The neurons together form a network: input neurons are excited by some input signal and this signal gets propagated through weighted connections to subsequent network layers where the last set of neurons together form an output signal.

Nowadays neural networks see widespread use and provide impressive practical results. Examples include computer vision applications (e.g. Krizhevsky et al. (2012)), natural language processing (e.g. Sutskever et al. (2014)) or reinforcement learning (e.g. Mnih et al. (2015)). For a full history and overview, we refer to Schmidhuber (2015).

Depending on the application numerous types of neural networks exist. The three most basic families are feed-forward neural networks (often referred to as multi-layer perceptrons (MLP)), recurrent neural networks (RNN) and convolutional neural networks (CNN). Recurrent neural networks model the temporal structure of the data and are therefore often used when working with time-series. For instance, Sundermeyer et al. (2012) use a special type of RNN, the long short-term memory unit (LSTM) (Hochreiter and Schmidhuber, 1997), for language modeling applications. Convolutional neural networks model spatial relationship in the data, their most prominent application is computer vision; for instance in Krizhevsky et al. (2012) use this class of models to classify real-world images.

The classical type of neural network is the MLP. Throughout this thesis we will only use this type of network, in the following we will start by defining its key properties. Let us assume we are given a data set $\mathcal{D} = \{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N$, formed by feature vectors $\mathbf{x}_n \in \mathbb{R}^D$ and targets which can be continuous $\mathbf{y}_n \in \mathbb{R}^K$ or, in the case of classification, discrete $y_n \in \{1, \dots, K\}$. We try to approximate a function that maps from the input $\mathbf{x}$ to the outputs $\mathbf{y}$.

A MLP with $L = 1$ hidden layer of size H specifies a parametric function of the form:

$$f(\mathbf{x}; \mathcal{W}) = \mathbf{W}_2 \varphi(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2 \ . \tag{2.43}$$

The parameters of the MLP are: the $H \times D$ weight matrix $\mathbf{W}_1$, the H dimensional bias vector $\mathbf{b}_1$, the $K \times H$ weight matrix $\mathbf{W}_2$ and the D dimensional bias vector $\mathbf{b}_2$. We summarize these parameters to the collection of weights $\mathcal{W} = \{\mathbf{W}_l, \mathbf{b}_l\}_{l=1}^L$. $\varphi(.)$ is a nonlinear function, called the activation function. Popular choices are the hyperbolic tangent $\varphi(x) = \tanh(x)$ or the linear rectified unit $\varphi(x) = \max(x, 0)$. Using multiple hidden layers ($L > 1$) is what is referred to as deep neural networks: the formula given by Eq. (2.43) can easily be extended by adding further compositions.

The parameters $\mathcal{W}$ are optimized by minimizing an error function $\mathcal{L}(\mathcal{W}|\mathbf{X}, \mathbf{Y})$ with respect to the weight set $\mathcal{W}$. A popular choice for regression is the Euclidean loss, which

measures the quadratic distance between the output of the network, that is $f(\mathbf{x}_n; \mathcal{W})$, and the corresponding target vector $\mathbf{y}_n$. Averaging over the full training set results in

$$\mathcal{L}(\mathcal{W}|\mathbf{X}, \mathbf{Y}) = \frac{1}{N} \sum_{n=1}^{N} (f(\mathbf{x}_n; \mathcal{W}) - \mathbf{y}_n)^2 \ . \tag{2.44}$$

There exist many different error functions, often used in specific use cases. For instance, in classification a standard approach is to predict a K dimensional output vector $\hat{\mathbf{p}}$ representing probabilities for each class, which is implemented via the softmax activation function:

$$\hat{p}_{n,i} = \frac{e^{\hat{y}_{n,i}}}{\sum_{k=1}^{K} e^{\hat{y}_{n,k}}} \ . \tag{2.45}$$

In this case optimization is performed using the cross entropy error function:

$$\mathcal{L}(\mathcal{W}|\mathbf{X}, \mathbf{Y}) = -\frac{1}{N} \sum_{n=1}^{N} \log \hat{p}_{n,y_n} \ , \tag{2.46}$$

where $\hat{p}_{n,y_n}$ will provide the probability of the true class y_n under the model. The cross entropy $H[p, q]$ is related to the KL divergence:

$$\mathrm{H}[p, q] = \mathrm{H}[p] + \mathrm{KL}[p||q] \ , \tag{2.47}$$

and since $\mathrm{H}[p]$, the entropy of the true distribution, is fixed, minimizing the cross entropy is equal to minimizing the KL divergence. One may wonder why, unlike in regression, the prediction of the network is a probability vector $\hat{\mathbf{p}}$ and not a hard class assignment. This step is necessary, to make the loss $\mathcal{L}(\mathcal{W}|\mathbf{X}, \mathbf{Y})$ differentiable w.r.t. the network parameters $\mathcal{W}$, a property that enables using gradient information for training which we will discuss next.

Neural networks are nonlinear models with respect to their parameters: looking at the basic form of a one layer MLP in Eq. (2.43), we see that $\mathbf{W}_1$ and $\mathbf{b}_1$ appear inside the non-linearity. While the parameters of linear models can be optimized in closed form, using the Moore-Penrose pseudo-inverse (Bishop, 2007), for these models no closed form solution exists. For this reason, minimization is standardly performed using gradient descent: because both the error function $\mathcal{L}$ and the network are differentiable, we can calculate the gradient $\frac{\partial \mathcal{L}}{\partial \mathcal{W}}$ given input data $(\mathbf{X}, \mathbf{Y})$. A neural network is a composite function: the output of one layer is propagated as input to the next layer. Consequently, for the gradient a similar relationship holds: a gradient is obtained by first performing a forward pass to estimate the current error with respect to the loss function $\mathcal{L}$, and then, starting from the error the gradient signal is propagated back through the individual layers in a sequential manner. This process is called back-propagation, and it follows as a direct application of the chain rule.

Gradient descent updates the weights $\mathcal{W}$ by taking a step in the direction of greatest descent w.r.t. the error function:

$$\mathcal{W}_{\text{new}} = \mathcal{W} - \eta \frac{\partial \mathcal{L}}{\partial \mathcal{W}} \, , \tag{2.48}$$

where η is the learning rate. When the dataset is large, instead of using the full data set for each gradient update, we can use mini-batches by only evaluating the gradient on a small subset $\mathcal{S}$ of the training data, which is chosen randomly for each gradient step. This method is called stochastic gradient descent. There exist more sophisticated learning schemes that extend standard stochastic gradient descent such as using momentum (Nesterov, 1983), or using adaptive learning rates (Kingma and Ba, 2014).

Neural networks are universal functional approximators (Hornik et al., 1989) in the sense that, under mild assumptions, for any continuous function f and any error threshold ϵ, there exists a MLP with a specific set of parameters with an approximation error smaller than ϵ. This universality theorem suggests that neural networks can represent large classes of functions, in the beginning of this section we highlighted different areas of science where neural networks have successfully been applied.

Having said that one major challenge in training neural networks is their tendency to overfit on data: when the data set is limited and the network has many parameters, we observe that the error on the training dataset D continuously decreases during training whereas the performance on test data D_{Test} deteriorates. Different techniques exist that try to mitigate this issue of overfitting (or over-training), we will discuss now two approaches: early stopping and weight decay.

In early stopping we split from the training data set D a validation set D_{val} and use only the remaining examples for training, while continuously monitoring the performance of the network on D_{val} during training. After training, we use the weight configuration that resulted in the lowest validation error as the final weight set. On the one hand, this method is promising in keeping the performance similar between training and testing, on the other hand, we use fewer data during training because we do not train on the validation set.

An alternative method, called weight decay, instead adds a penalty to the loss function based on the size of the weights. The reasoning is that for most activation functions small weight values imply a smooth and slow changing function to be preferred. One cause of overfitting is that the network memorizes, instead of generalizes, the training data, which gives rise to very complicated but inaccurate functions (see e.g. Zhang et al. (2016)). By adjusting the error function to prefer network weights of small value, we can expect that the function the network has learned after training is more smooth and less prone to memorization. A standard way to achieve this is to use the L2-norm, for instance in classification the loss function may be:

$$\mathcal{L}(\mathcal{W}|\mathbf{X},\mathbf{Y}) = -\frac{1}{N}\sum_{n=1}^{N}\log(\hat{p}_{n,y_n}) + \lambda(\sum_{l=1}^{L}||\mathbf{W}_l||^2 + ||\mathbf{b}_l||^2)\ , \tag{2.49}$$

where λ is the free parameter setting the strength of the weight decay.

Neural networks by default do not model uncertainty over their parameters. This is because we only train a single set of weights to estimate a function. In the framework of Bayesian Modeling that we introduced in Section 2.1 we perform maximum likelihood over the parameters $\mathcal{W}$. A loss function can alternatively be interpreted as a

log-likelihood function (LeCun et al., 2006; Gal, 2016):

$$p(\mathbf{y}_n|\mathbf{x}_n, \mathcal{W}) \propto \exp[-\mathcal{L}(\mathcal{W}|\mathbf{x}_n, \mathbf{y}_n)] \; . \tag{2.50}$$

When using the Euclidean loss $\exp[-\mathcal{L}(\mathcal{W}|\mathbf{x}_n, \mathbf{y}_n)]$ is proportional to a Gaussian likelihood function. Minimizing the loss then corresponds to maximizing the equivalent likelihood and when using weight decay we obtain a maximum posterior (MAP) estimate with a Gaussian prior on the weights.

In classification, the softmax output function, given by Eq. (2.45), will provide a probability for every class, which can be used to model predictive uncertainty. This uncertainty, however, is not w.r.t. to the parameters of the model, but only to the insecurity about which class K an input $\mathbf{x}$ belongs to under a single model. In regression, a similar way to estimate uncertainty is to use the average squared error on the training data to obtain a constant (homoscedastic) estimate of uncertainty. However, as we discussed earlier, neural networks tend to overfit, therefore the training error often severely underestimates the test error. If we use early stopping, we can instead use the error on the validation set as an estimate of uncertainty. The predictive distribution of the neural network is then

$$p(\mathbf{y}_\star \,|\, \mathbf{x}_\star, \mathcal{D}) = \mathcal{N}(\mathbf{y}_\star | f(\mathbf{x}_\star; \mathcal{W}_{\text{val}}), \sigma^2_{\text{val}}) \; , \tag{2.51}$$

$$\sigma^2_{\text{val}} = \frac{1}{N_{\text{val}}} \sum_{n=1}^{N_{\text{val}}} (f(\mathbf{x}_n; \mathcal{W}_{\text{val}}) - \mathbf{y}_n)^2 \; . \tag{2.52}$$

There exist many alternative ways to make a single neural network capable to express uncertainty in regression. For instance, Nix and Weigend (1994) suggest to have a network with two outputs, one for the mean $\mu(\mathbf{x}; \mathcal{W})$ and another output for the variance $\sigma^2(\mathbf{x}; \mathcal{W})$. This approach enables a neural network to model state-dependent (heteroscedastic) Gaussian uncertainty. In this case the network is trained by minimizing the the negative log-likelihood

$$-\log p(\mathbf{y}_n|\mathbf{x}_n) = \frac{\log(\sigma^2(\mathbf{x}_n; \mathcal{W}))}{2} + \frac{(\mathbf{y}_n - \mu(\mathbf{x}_n; \mathcal{W}))^2}{2\sigma^2(\mathbf{x}_n; \mathcal{W}))} \; . \tag{2.53}$$

We note that both approaches however, do not model uncertainty in the same way as Bayesian modeling, because these approaches do not consider uncertainty over the parameters. We will discuss the difference between these forms of uncertainty in more detail in Chapter 3.

Lakshminarayanan et al. (2017) extend the approach to heteroscedastic Gaussian output nodes using ensembles and adversarial learning. Another technique is dropout (Srivastava et al., 2014), where hidden activations are multiplied by Bernoulli random noise, or Gaussian in the case of Gaussian dropout. The work in Gal and Ghahramani (2016) shows that using Monte Carlo dropout (MC-dropout) is a particular form of Bayesian modeling. This insight enables using the Monte Carlo samples of the output variables, which are generated by different dropout masks, to estimate estimate predictive uncertainty.

In the next section we will discuss ensembling as an additional way to model predictive uncertainty in neural networks.

2.4.2 Ensembling/Bootstrapping Neural Networks

The goal of ensembling is to use a collection of base learners to obtain better predictive performance than would be given by individual base learners alone. Traditionally, the most popular form ensembles are formed using decision trees as base learners, however in principle any learning algorithm, or combinations of these, can be used. Here we discuss ensembles of neural networks.

We can cluster existing ensembling methods in two categories (Lakshminarayanan et al., 2017): in randomization-based approaches (such as random forests) we optimize each base learner in isolation, potentially in parallel without any information exchange between each of them. The other category is formed by boosting-based approaches. In boosting, we fit the base learner in a stage-wise fashion. For instance, in gradient boosting each base learner is trained on the residual of the base learner in the previous stage. By this we can expect the function approximation to be gradually more accurate in each iteration (Friedman, 2001).

Here we focus on the randomization-based approach in neural networks, because of the capability of parallelization. Randomization can happen at multiple steps in the training process. The most basic step is to initialize the parameters of each base learner $\mathcal{W}_k$ at random. Because training neural networks is a nonlinear and typically stochastic process, heterogeneity over the resulting models can be expected due to the varying starting states. An additional strategy is bootstrapping, also referred to as bagging, where each base learner is trained on different bootstrap samples from the original training data set.

If we use base learners of neural networks trained by early stopping and running K instance of it in parallel, we have achieved both randomization and bootstrapping. The former is achieved by initializing all parameters $\mathcal{W}_k$ at random, whereas the latter is achieved by using a different validation set (chosen at random) for each base learner. This way every base learner will have a slightly different training set to learn. Following the constant uncertainty estimate from the previous section in regression, each base learner has its own constant uncertainty estimate. In addition we now have a spread in the predictions in the ensemble. The predictive distribution of the ensemble is now a mixture of Gaussians with equal weights $1/K$:

$$p(\mathbf{y}_\star \mid \mathbf{x}_\star, \mathcal{D}) = \frac{1}{K} \sum_{k=1}^{K} \mathcal{N}(\mathbf{y}_\star | f(\mathbf{x}_\star; \mathcal{W}_{\text{val},k}), \sigma^2_{\text{val},k}) \ , \tag{2.54}$$

$$\sigma^2_{\text{val},k} = \frac{1}{N_{\text{val}}} \sum_{n=1}^{N_{\text{val}}} (f(\mathbf{x}_n; \mathcal{W}_{\text{val},k}) - \mathbf{y}_n)^2 \ . \tag{2.55}$$

There exist many alternative ways of utilizing ensembling to model predictive uncertainty. For instance, Chua et al. (2018) use ensembles of neural networks, where each network is trained using Eq. (2.53).

In the previous section we have connected the process of training neural networks to Bayesian modeling. When using ensembling, the relationship to Bayesian modeling is not as clear. Formally, we conduct multiple MAP estimates under the equivalence

described in Eq. (2.50). At first glance, there is no justification to consider the resulting sets of weights $\{\mathcal{W}_1, \ldots, \mathcal{W}_K\}$ as samples from the true posterior $p(\mathcal{W} \mid \mathcal{D})$. One approach described in Zimmermann et al. (2012) suggests training an ensemble of over-parameterized neural networks so that each member has a training error of zero. The authors then suggest using the ensemble spread as predictive uncertainty. In this case, a connection to Bayesian modeling can be made: under a delta function as likelihood function $p(\mathbf{y}_n|\mathbf{x}_n, \mathcal{W})$ and a flat prior $p(\mathcal{W})$, repeated MAP estimates only need to find the different peaks of the posterior $p(\mathcal{W} \mid \mathcal{D})$, which can be achieved by different weight initialization.

2.4.3 Bayesian Neural Networks

In Section 2.1 we have introduced the concepts of Bayesian modeling and in Section 2.2 the fundamentals of variational inference. We will now apply these concepts to neural networks giving rise to Bayesian neural networks (BNN).

We will begin by defining the model in the context of regression. We have i.i.d. data $\mathcal{D} = \{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N$, formed by feature vectors $\mathbf{x}_n \in \mathbb{R}^D$ and targets $\mathbf{y}_n \in \mathbb{R}^K$. We assume that $\mathbf{y}_n = f(\mathbf{x}_n; \mathcal{W}) + \boldsymbol{\epsilon}_n$, where $f(\cdot, \cdot; \mathcal{W})$ is the output of a neural network with weights $\mathcal{W}$ and $\boldsymbol{\epsilon}_n \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$ is additive Gaussian noise with a diagonal covariance matrix $\boldsymbol{\Sigma}$.

We further assume a fixed network topology. The network is a MLP with L layers, with V_l hidden units in layer l, and $\mathcal{W} = \{\mathbf{W}_l\}_{l=1}^L$ is the collection of $V_l \times (V_{l-1} + 1)$ weight matrices. The $+1$ is introduced here to account for the additional per-layer biases. In principle, we can maintain uncertainty over the network topology by defining a hyper-prior over the topology components. For the rest of this thesis, however, we will assume that the data was generated by a network with a fixed topology.

The first step in Bayesian modeling is to define a prior over the parameters of the model. For instance, we can specify a Gaussian prior distribution for each entry in each of the weight matrices in $\mathcal{W}$:

$$p_0(\mathcal{W}) = \prod_{l=1}^{L} \prod_{i=1}^{V_l} \prod_{j=1}^{V_{l-1}+1} \mathcal{N}(w_{ij,l} \mid 0, \lambda)\,. \tag{2.56}$$

This means that before seeing any data, we assume weights that are close to 0 are more likely. This prior is similar to weight-decay, one technique we introduced in the previous section to regularize neural networks. More complex prior distributions are possible: Blundell et al. (2015) show how to use Gaussian mixture priors for more flexibility, Ghosh et al. (2018) develop so-called horseshoe priors on the weights to induce sparsity for better model selection and Flam-Shepherd et al. (2017) show how to map GP priors to BNN priors.

Because we assume Gaussian output noise the likelihood of the data given weights $\mathcal{W}$ is:

$$p(\mathbf{Y} \mid \mathcal{W}, \mathbf{X}) = \prod_{n=1}^{N} p(\mathbf{y}_n \mid \mathcal{W}, \mathbf{x}_n) = \prod_{n=1}^{N} \prod_{k=1}^{K} \mathcal{N}(y_{n,k} \mid f(\mathbf{x}_n; \mathcal{W}), \boldsymbol{\Sigma})\,. \tag{2.57}$$

With the prior and the likelihood defined, we can now formulate the posterior of the BNN, that is

$$p(\mathcal{W} \mid \mathcal{D}) = \frac{p(\mathbf{Y} \mid \mathcal{W}, \mathbf{X}) p_0(\mathcal{W})}{p(\mathbf{Y} \mid \mathbf{X})} \,. \tag{2.58}$$

Given a new input vector $\mathbf{x}_\star$, we can then make predictions for $\mathbf{y}_\star$ using the predictive distribution

$$p(\mathbf{y}_\star \mid \mathbf{x}_\star, \mathcal{D}) = \int \mathcal{N}(\mathbf{y}_\star \mid f(\mathbf{x}_\star; \mathcal{W}), \mathbf{\Sigma}) p(\mathcal{W} \mid \mathcal{D}) \, d\mathcal{W} \,. \tag{2.59}$$

In classification we would instead use a softmax activation of the outputs given by Eq. (2.45) an a corresponding likelihood function of $p(\mathbf{Y} \mid \mathcal{W}, \mathbf{X}) = \prod_{n=1}^{N} \hat{p}_{n,y_n} \mid \mathcal{W}, \mathbf{x}_n$ where, as discussed in Section 2.4.1 , $\hat{p}_{n,y_n}$ is the probability of the true label y_n under the model, for a particular set of weights $\mathcal{W}$. The predictive distribution in this case is the expected probability vector integrated over all weight configuration according to the posterior $p(\mathcal{W} \mid \mathcal{D})$.

Unfortunately, the exact computation of (2.59) is intractable and we have to use approximations. Early work used sampling methods to obtain an empirical distribution over weight samples (Neal, 1996). More recent work showed that augmenting standard gradient descent with Gaussian noise can also be interpreted as a particular form of MCMC (Welling and Teh, 2011; Balan et al., 2015; Mandt et al., 2017). Furthermore, we already identified in the previous section that using dropout has a corresponding interpretation in Bayesian modeling (Gal and Ghahramani, 2016).

Recently, variational inference has seen increasing interest and here we want to outline this particular approach towards BNNs (Blundell et al., 2015; Hernández-Lobato and Adams, 2015; Hernández-Lobato et al., 2016). We approximate the exact posterior distribution $p(\mathcal{W} \mid \mathcal{D})$ with a simpler distribution, typically a factorized Gaussian distribution:

$$q(\mathcal{W}) = \left[\prod_{l=1}^{L} \prod_{i=1}^{V_l} \prod_{j=1}^{V_{l-1}+1} \mathcal{N}(w_{ij,l} | m^w_{ij,l}, v^w_{ij,l}) \right] \,, \tag{2.60}$$

where $m^w_{ij,l}$ and $v^w_{ij,l}$ are respectively the means and variance of each weight. This approach of approximating a complex distribution into independent factors is called mean-field approach, in this case the graph is fully-factored. A recent extension to this method is given by Louizos and Welling (2016), that uses a matrix variate Gaussian distribution (Gupta and Nagar, 1999).

The parameters $m^w_{ij,l}$, $v^w_{ij,l}$ are determined by minimizing a divergence between $p(\mathcal{W} \mid \mathcal{D})$ and the approximation q.

We can directly apply the Monte Carlo variational Bayes bound in Eq. (2.23) or the α-divergence minimization given by Eq. (2.35) for optimization. For instance the MC-VB

when using BNNs in regression would be

$$\mathcal{L}_{\text{VB}}^{\text{MC}} = \frac{1}{K}\sum_{k=1}^{K}\frac{N}{|\mathcal{S}|}\sum_{n\in\mathcal{S}}\log p(\mathbf{y}_n|\mathbf{x}_n,\mathcal{W}_k) - KL[q(\mathcal{W})||p_0(\mathcal{W})]\ . \tag{2.61}$$

$$= \frac{1}{K}\sum_{k=1}^{K}\frac{N}{|\mathcal{S}|}\sum_{n\in\mathcal{S}}\log \mathcal{N}(\mathbf{y}_n|f(\mathbf{x}_n;\mathcal{W}_k),\mathbf{\Sigma}) - KL[q(\mathcal{W})||p_0(\mathcal{W})]\ . \tag{2.62}$$

where the entropy term can be computed in closed form, because both the prior and the variational distribution is Gaussian. For instance for $\lambda = 1$ we have:

$$KL[q(\mathcal{W})||p_0(\mathcal{W})] = -\frac{1}{2}\sum_{l=1}^{L}\sum_{i=1}^{V_l}\sum_{j=1}^{V_{l-1}+1}(1+\log(v_{ij,l}^w) - (m_{ij,l}^w)^2 - v_{ij,l}^w)\ , \tag{2.63}$$

which was derived in Kingma and Welling (2013). The work in Blundell et al. (2015) (Eq. (2)) further derives an approximation to the entropy term in Eq. (2.62) that enables modeling more complex prior distributions such as Gaussian mixtures.

Minimization of this bound is performed with respect to both the variational parameters $m_{ij,l}^w$,$v_{ij,l}^w$ and the parameters of the Gaussian observation noise Σ, which can be performed by gradient descent and, because the variational distribution is Gaussian, utilizing the reparameterization trick.

For black-box α minimization we obtain the following formula:

$$\hat{E}(\boldsymbol{\lambda}_0,\boldsymbol{\lambda}_q) = -\log Z(\boldsymbol{\lambda}_q) - \frac{1}{\alpha}\frac{N}{|\mathbf{S}|}\sum_{n\in\mathbf{S}}\log\frac{1}{K}\sum_k\left(\frac{p(\mathbf{y}_n|\mathbf{x}_n,\boldsymbol{\theta}_k)}{f(\boldsymbol{\theta}_k)}\right)^\alpha \tag{2.64}$$

$$= -\log Z(\boldsymbol{\lambda}_q) - \frac{1}{\alpha}\frac{N}{|\mathbf{S}|}\sum_{n\in\mathbf{S}}\log\frac{1}{K}\sum_k\left(\frac{\mathcal{N}(\mathbf{y}_n|f(\mathbf{x}_n;\mathcal{W}_k),\mathbf{\Sigma})}{f(\mathcal{W}_k)}\right)^\alpha\ , \tag{2.65}$$

$$\tag{2.66}$$

where we dropped the normalization constant of the prior because it does not affect the minimization and we have:

$$\log Z_q = \sum_{l=1}^{L}\sum_{i=1}^{V_l}\sum_{j=1}^{V_{l-1}+1}\left[\frac{1}{2}\log\left(2\pi v_{i,j,l}^w\right) + \frac{\left(m_{i,j,l}^w\right)^2}{v_{i,j,l}^w}\right]\ , \tag{2.67}$$

$$f(\mathcal{W}) = \exp\left\{\sum_{l=1}^{L}\sum_{i=1}^{V_l}\sum_{j=1}^{V_{l-1}+1}\frac{1}{N}\left(\frac{v_{i,j,l}^w-\lambda}{2\lambda v_{i,j,l}^w}w_{i,j,l}^2 + \frac{m_{i,j,l}^w}{v_{i,j,l}^w}w_{i,j,l}\right)\right\} \propto \left[\frac{q(\mathcal{W})}{p(\mathcal{W})}\right]^{\frac{1}{N}}\ . \tag{2.68}$$

$$\tag{2.69}$$

BNNs gained a lot of attention in research in the recent years. While much of the work is focused on theory and improving the inference techniques there is also exciting research focusing on practical applications. For instance Kendall and Gal (2017) used this class

of models for semantic segmentation in computer vision domains. In Houthooft et al. (2016) the authors utilize the uncertainty over dynamics models to improve exploration in reinforcement learning. In McAllister et al. (2017) the authors discuss the possible advantage of using Bayesian neural network for autonomous vehicle safety.

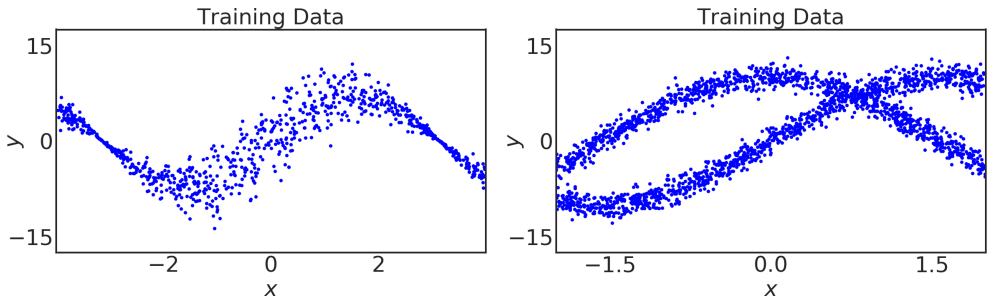
3 Modeling Epistemic and Aleatoric Uncertainty

In this chapter we introduce the concepts of epistemic and aleatoric uncertainty as two forms of uncertainty in the context of supervised learning. As the centerpiece of this chapter we present Bayesian neural networks with latent variables (BNN+LV) in Section 3.1, a novel probabilistic model that, unlike standard methods from Bayesian modeling, can model both of the aforementioned forms of uncertainty. Section 3.2 will show how to extract and decompose predictive uncertainty into epistemic and aleatoric parts; later in this thesis, we will show application scenarios where we use this decomposition for decision-making. Lastly in Section 3.3 we will present a novel model inspection method, the sensitivity analysis of epistemic and aleatoric uncertainty, to relate the quantities of uncertainties we see in the predictions to the input features. The methods we develop in this chapter were previously published in Depeweg et al. (2017a,b,c, 2018).

In the previous chapter, we have introduced the fundamentals of Bayesian modeling to reason about uncertainty in parametric and non-parametric models in supervised learning. In particular, we highlighted Gaussian processes and Bayesian neural networks as two modern methods that see widespread practical use. The premise these methods share is that an unknown function $\mathbf{y} = f(\mathbf{x}) + \epsilon$ generated the data and we can reason about this function in a parametric (such as in BNNs) or non-parametric (such as in GPs) way. This assumption implies a deterministic (functional) relationship between the inputs and outputs, with the addition of some constant Gaussian observation noise.

There are many situations where this assumption will not hold. Take for example the problem of predicting the amount of traffic at some intersection ($\mathbf{y}$) based on the current day of the week ($\mathbf{x}$). While we certainly expect some relationship between the input and the output—Mondays are expected to be busier than Sundays—, there are many exceptions such as public holidays. These phenomena give rise to a multimodal distribution of traffic intensity on this day of the week. Would we use one of the aforementioned models for such a task they will however assume that a deterministic function exists and model a coarse relationship with a high constant noise level ϵ. Many of the patterns in the data, such as the multimodal distribution in this example, will not be modeled and expressed by constant noise. Because of this little information would be gained by such a model.

In many supervised learning problems there exists no fully functional relationship between the features and the outputs. As in the example above, the feature set may be limited and thereby insufficient to predict output $\mathbf{y}$ based on input features $\mathbf{x}$. We define this situation as *partial observability*: there exist additional features that would enable us to predict $\mathbf{y}$ but that are unobserved. We call these unobserved features *latent*



(a) **(b)**

Figure 3.1: Example data with heteroscedastic (Fig. 3.1a) and bimodal (Fig. 3.1b) noise.

variables **z**. In unsupervised learning, the term "latent variables" is used in the meaning of "underlying" variables that generate the observations **x** in a hierarchical process. In contrast in our line of reasoning "latent variables" are characterized as additional, "hidden" variables that are not part of the feature set **x**.

The effect of **z** are noise patterns in the output variables such as multimodality or heteroscedasticity. Latent variables may also be purely random: in Fig. 3.1 we show data from two examples with heteroscedastic and bimodal noise. While the data here was generated using random numbers, we can alternatively say that for instance, in Fig. 3.1b there exists an unobserved feature, a latent variable, that decides where the output **y** falls into the upper or lower mode.

Partial-observability or randomness will introduce uncertainty in the prediction problem: Given only **x** we will not be able to predict **y** perfectly and ideally, a model should express this uncertainty in its prediction. This uncertainty is *irreducible*: irrespective of how much training data we have available, it will remain because it is an intrinsic property of the data. By contrast, in Bayesian modeling we consider uncertainty about the model, either in parametric or non-parametric form. For instance, in parametric models we expect the posterior $p(\boldsymbol{\theta}|\mathcal{D})$ to shrink, that is to become more certain, the more data we collect. Therefore, this form of uncertainty is *reducible* uncertainty. In literature these two forms of uncertainty are referred to as *epistemic* and *aleatoric* uncertainty (e.g Matthies (2007); Der Kiureghian and Ditlevsen (2009)), where epistemic is the reducible and aleatoric the irreducible part of uncertainty.

Ideally, we would like to have a machine learning algorithm that can express both epistemic and aleatoric uncertainty with a high level of generality. For decision making, knowing where a method is uncertain because data are limited, and knowing where a method is uncertain because of randomness (or partial observability) in the data can be highly beneficial. Following the example given in Senge et al. (2014) (Section 1), "A medical doctor, for example, who knows that his uncertainty about the illness of a patient is caused by a lack of knowledge about the disease in question, may decide to consult the literature or ask a colleague before making a decision." By contrast, if the uncertainty in the diagnosis would be due to partial observability, the doctor may decide

to do additional diagnostic tests. In the chapters to follow, we will show these advantage in a set of empirical studies.

Supervised learning methods that utilize Bayesian modeling, such as GPs or BNNs, only model the epistemic form of uncertainty. While some recent extensions exist, for instance modeling heteroscedastic Gaussian noise (Lázaro-Gredilla and Titsias, 2011; Chua et al., 2018), to our knowledge there is no method that both models epistemic uncertainty via a Bayesian modeling approach and can model complex noise patterns with a high level of generality. The next section will introduce a novel probabilistic method that addresses this issue; we will extend the Bayesian neural networks with a latent variable model.

3.1 Bayesian Neural Networks with Latent Variables

3.1.1 Model Assumption

We want to approximate a parametric function that can express arbitrary noise patterns, such as multimodality or heteroscedasticity. Similar to Section 2.4.3 we will focus our derivations on regression and provide the formulas for classification afterward. Let us assume the unknown function that generated an i.i.d. data set $\mathcal{D} = \{\mathbf{X}, \mathbf{Y}\}$ formed by feature vectors $\mathbf{x}_n \in \mathbb{R}^D$ and targets $\mathbf{y}_n \in \mathbb{R}^K$ is of the form:

$$\mathbf{y}_n = f(\mathbf{x}_n, z_n) + \boldsymbol{\epsilon}_n \ , \tag{3.1}$$

where z_n is some unobserved (latent) random variable from an unknown distribution $p(z)$ and $\boldsymbol{\epsilon}_n \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$ is independent Gaussian noise. We assume here that all z_i, z_j are conditionally independent given $\mathbf{x}_n$. This definition follows Bertsekas (2002) (Eq. (1)) where it is used to model general stochastic time-discrete systems. Note that the definition used in Bertsekas (2002) allows a state dependency in z, where $z_n \sim p(z|\mathbf{x}_n)$, however, because $\mathbf{x}_n$ is already an input of f these two formulations can be transfered into another: the function f in Eq. (3.1) can transform z_n by first applying the cumulative distribution function (CDF) of $p(z)$ and then use the quantile function of $p(z|\mathbf{x}_n)$ to obtain samples from this distribution. This process is called inversion sampling. While we consider z_n as a one-dimensional variable here, the following derivations and discussions can easily extended towards multi-dimensional latent variables.

The modeling assumption in Eq. (3.1) is very general, we can model arbitrary functions in f and the effect of z on the output $\mathbf{y}$ can include any form of stochastic pattern, such as heteroscedastic or multimodal noise. For instance, $y = 10\sin(x)z + 10\cos(x)z + \epsilon$ with $z \sim \text{Bern}(0.5)$ and $\epsilon \sim \mathcal{N}(0, 1)$ models the bimodal data we have seen in Fig. 3.1b.

In Section 2.4.1 we introduced neural networks as parametric models. Let us assume that we can express f, which is deterministic given $\mathbf{x}$ and z, via such a neural network. We can then rewrite:

$$\begin{aligned} \mathbf{y}_n &= f(\mathbf{x}_n, z_n) + \boldsymbol{\epsilon}_n & (3.2) \\ &= f(\mathbf{x}_n, z_n; \mathcal{W}') + \boldsymbol{\epsilon}_n \ , & (3.3) \end{aligned}$$

where $f(\cdot, \cdot; \mathcal{W}')$ is the output of a neural network with weights $\mathcal{W}'$. The network receives as input the feature vector $\mathbf{x}_n$ and the random noise z_n. We assume the network has a feed-forward topology as we described in Section 2.4.1, for instance the activation functions for the hidden layers could be rectifiers: $\varphi(x) = \max(x, 0)$ and for the output layers we have the identity function: $\varphi(x) = x$. The network has L layers, with V_l hidden units in layer l, and $\mathcal{W}' = \{\mathbf{W}_l\}_{l=1}^{L}$ is the collection of $V_l \times (V_{l-1} + 1)$ weight matrices. The $+1$ is introduced here to account for the additional per-layer biases.

So far the density $p(z)$ could be any probability distribution, which makes modeling difficult. We can again apply the concept of inversion sampling: arbitrary probability distributions can be obtained by transforming samples from a Gaussian distribution through non-linearities, e.g. by applying the Gaussian CDF and then the inverse CDF of any desired distribution. We can therefore rewrite:

$$\begin{aligned}
\mathbf{y}_n &= f(\mathbf{x}_n, z_n, \mathcal{W}') + \boldsymbol{\epsilon}_n && && (3.4)\\
&= f(\mathbf{x}_n, g(z_n), \mathcal{W}') + \boldsymbol{\epsilon}_n, && z_n \sim \mathcal{N}(0, \gamma) && (3.5)\\
&= f(\mathbf{x}_n, z_n; \mathcal{W}) + \boldsymbol{\epsilon}_n, && z_n \sim \mathcal{N}(0, \gamma)\ , && (3.6)
\end{aligned}$$

where g computes the Gaussian CDF and then the inverse CDF of the distribution $p(z_n)$, which we assume can be modeled as part of $\mathcal{W}$, and γ is an optional hyper-parameter. For this we define, without loss of generality, $\gamma = d$, where d is the dimensionality of $\mathbf{x}$. This reasoning behind this is to make the effect of z not vanish for high-dimensional problems.

We have now introduced the model architecture: given a data set $\mathcal{D}$ we assume the data was generated by an unknown function given by Eq. (3.6). In the data set we only observe the features $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ and targets $\mathbf{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_N\}$, while the the set of weights $\mathcal{W}$ and the latent variables $\mathbf{z} = \{z_1, \ldots, z_N\}$, that were used to generate the training data, are unknown. Using the methodology of Bayesian modeling we can reason about, that means we can infer a posterior, over both sets of variables $\mathcal{W}, \mathbf{z}$. For the weights, this task is similar to that of ordinary BNNs (see Sec. 2.4.3): the function in Eq. (3.6) is deterministic, given $\mathbf{x}$ and z. Inferring a posterior over $z_1, \ldots, z_N$ means to reason about the values of the latent variables for every individual training data point $(\mathbf{x}_n, \mathbf{y}_n)$. If we once again consider the bimodal data shown in Fig. 3.1b, the model should infer that for all those data that fall in the upper mode a different value for the latent variable is inferred than for those in the lower mode. In this case, the deterministic part of the architecture, which is modeled by $\mathcal{W}$, only needs to express the sine and cosine function, while the inferred latent variables act as a discriminator between both modes.

We note that in principle the constant output noise $\boldsymbol{\epsilon}_n$ is not needed at all in Eq. (3.6) because we are already using the more flexible stochastic model based on z_n. We still use it as part of the model, because in practice we make predictions with the above model by averaging over a finite number of samples of z_n and $\mathcal{W}$. By using $\boldsymbol{\epsilon}_n$, we obtain a predictive distribution whose density is well defined and given by a mixture of Gaussians. If we eliminate $\boldsymbol{\epsilon}_n$, the predictive density is degenerate and given by a mixture of delta functions.

We will now describe the Bayesian modeling approach which gives rise to Bayesian neural network with latent variables (BNN+LV).

Let $\mathbf{Y}$ be a $N \times K$ matrix with the targets $\mathbf{y}_n$ and $\mathbf{X}$ be a $N \times D$ matrix of feature vectors $\mathbf{x}_n$. We denote by $\mathbf{z}$ the N-dimensional vector with the values of the latent variables $z_1, \ldots, z_N$ that were used to generate the data. The likelihood function is

$$p(\mathbf{Y} \,|\, \mathcal{W}, \mathbf{z}, \mathbf{X}) = \prod_{n=1}^{N} p(\mathbf{y}_n \,|\, \mathcal{W}, z_n, \mathbf{x}_n) = \prod_{n=1}^{N} \prod_{k=1}^{K} \mathcal{N}(y_{n,k} \,|\, f(\mathbf{x}_n, z_n; \mathcal{W}), \boldsymbol{\Sigma}) \,. \tag{3.7}$$

The prior for each entry in $\mathbf{z}$ is $\mathcal{N}(0, \gamma)$. As in BNN we specify a Gaussian prior distribution for each entry in each of the weight matrices in $\mathcal{W}$. That is,

$$p(\mathbf{z}) = \prod_{n=1}^{N} \mathcal{N}(z_n | 0, \gamma) \,, \tag{3.8}$$

$$p(\mathcal{W}) = \prod_{l=1}^{L} \prod_{i=1}^{V_l} \prod_{j=1}^{V_{l-1}+1} \mathcal{N}(w_{ij,l} \,|\, 0, \lambda) \,, \tag{3.9}$$

where $w_{ij,l}$ is the entry in the i-th row and j-th column of $\mathbf{W}_l$ and γ and λ are a prior variances. The posterior distribution for the weights $\mathcal{W}$ and the latent variables $\mathbf{z}$ is given by Bayes' rule:

$$p(\mathcal{W}, \mathbf{z} \,|\, \mathcal{D}) = \frac{p(\mathbf{Y} \,|\, \mathcal{W}, \mathbf{z}, \mathbf{X}) p(\mathcal{W}) p(\mathbf{z})}{p(\mathbf{Y} \,|\, \mathbf{X})} \,. \tag{3.10}$$

Given a new input vector $\mathbf{x}_\star$, we can then make predictions for $\mathbf{y}_\star$ using the predictive distribution

$$p(\mathbf{y}_\star \,|\, \mathbf{x}_\star, \mathcal{D}) = \int \left[\int \mathcal{N}(y_\star \,|\, f(\mathbf{x}_\star, z_\star; \mathcal{W}), \boldsymbol{\Sigma}) \mathcal{N}(z_\star | 0, \gamma) \, dz_\star \right] p(\mathcal{W}, \mathbf{z} \,|\, \mathcal{D}) \, d\mathcal{W} \, d\mathbf{z} \,. \tag{3.11}$$

We want to highlight that in the predictive distribution we use $f(\mathbf{x}_\star, z_\star; \mathcal{W})$ with $z_\star$ sampled from the prior $\mathcal{N}(z_\star | 0, \gamma)$. While part of the posterior $p(\mathcal{W}, \mathbf{z} \,|\, \mathcal{D})$ is to infer for every training data point $(\mathbf{x}_n, \mathbf{y}_n)$ a posterior $\mathcal{N}(z_n | \mu_n, \sigma_n^2)$, at test time the $\mathbf{y}_\star$ associated with $\mathbf{x}_\star$ is unknown and consequently, there is no other evidence on $z_\star$ than the one coming from $p(z_\star)$, because we assume independence between all pairs of (z_i, z_j).

Unfortunately, the exact computation of (2.4) is intractable and we have to use approximations. In the following we present a variational approximation based on α-divergences (see Section 2.2.2 for an introduction). We use the α-divergence here, to keep the approximation general over a whole family of divergences.

3.1.2 Variational Approximation

We approximate the exact posterior distribution $p(\mathcal{W}, \mathbf{z} \,|\, \mathcal{D})$ with the factorized Gaussian distribution

$$q(\mathcal{W}, \mathbf{z}) = \left[\prod_{l=1}^{L} \prod_{i=1}^{V_l} \prod_{j=1}^{V_{l-1}+1} \mathcal{N}(w_{ij,l} | m_{ij,l}^w, v_{ij,l}^w) \right] \left[\prod_{n=1}^{N} \mathcal{N}(z_n \,|\, m_n^z, v_n^z) \right] \,. \tag{3.12}$$

The parameters $m^w_{ij,l}$, $v^w_{ij,l}$ and m^z_n, v^z_n are determined by minimizing a divergence between $p(\mathcal{W}, \mathbf{z} \,|\, \mathcal{D})$ and the approximation q. After fitting q, we make predictions by replacing $p(\mathcal{W}, \mathbf{z} \,|\, \mathcal{D})$ with q in Eq. (3.11) and approximating the integrals in with empirical averages over samples of $\mathcal{W} \sim q$.

We aim to adjust the parameters of (3.12) by minimizing the α-divergence between $p(\mathcal{W}, \mathbf{z} \,|\, \mathcal{D})$ and $q(\mathcal{W}, \mathbf{z})$:

$$\mathrm{D}_\alpha[p(\mathcal{W}, \mathbf{z} \,|\, \mathcal{D}) || q(\mathcal{W}, \mathbf{z})] = \frac{1}{\alpha(\alpha - 1)} \left(1 - \int p(\mathcal{W}, \mathbf{z} \,|\, \mathcal{D})^\alpha q(\mathcal{W}, \mathbf{z})^{(1-\alpha)} \right) d\mathcal{W}\, d\mathbf{z}\,, \quad (3.13)$$

which includes a parameter $\alpha \in \mathbb{R}$ that controls the properties of the optimal q.

The direct minimization of (3.13) is infeasible in practice for arbitrary α. Following the methodology we introduced in Section 2.2.2, we optimize an energy function whose minimizer corresponds to a local minimization of α-divergences, with one α-divergence for each of the N likelihood factors in (3.7). Since q is Gaussian and the priors $p(\mathcal{W})$ and $p(\mathbf{z})$ are also Gaussian, we represent q as

$$q(\mathcal{W}, \mathbf{z}) \propto \left[\prod_{n=1}^{N} f(\mathcal{W}) f_n(z_n) \right] p(\mathcal{W}) p(\mathbf{z})\,, \quad (3.14)$$

where $f(\mathcal{W})$ is a Gaussian factor that approximates the geometric mean of the N likelihood factors in (3.7) as a function of $\mathcal{W}$. Each $f_n(z_n)$ is also a Gaussian factor that approximates the n-th likelihood factor in (3.7) as a function of z_n. We adjust $f(\mathcal{W})$ and the $f_n(z_n)$ by minimizing local α-divergences. In particular, we minimize the energy function

$$E_\alpha(q) = -\log Z_q - \frac{1}{\alpha} \sum_{n=1}^{N} \log \mathbf{E}_{\mathcal{W}, z_n \sim q} \left[\left(\frac{p(\mathbf{y}_n \,|\, \mathcal{W}, \mathbf{x}_n, z_n, \boldsymbol{\Sigma})}{f(\mathcal{W}) f_n(z_n)} \right)^\alpha \right], \quad (3.15)$$

where $f(\mathcal{W})$ and $f_n(z_n)$ are in exponential Gaussian form and parameterized in terms of the parameters of q and the priors $p(\mathcal{W})$ and $p(z_n)$, that is,

$$f(\mathcal{W}) = \exp \left\{ \sum_{l=1}^{L} \sum_{i=1}^{V_l} \sum_{j=1}^{V_{l-1}+1} \frac{1}{N} \left(\frac{v^w_{i,j,l} - \lambda}{2\lambda v^w_{i,j,l}} w^2_{i,j,l} + \frac{m^w_{i,j,l}}{v^w_{i,j,l}} w_{i,j,l} \right) \right\} \propto \left[\frac{q(\mathcal{W})}{p(\mathcal{W})} \right]^{\frac{1}{N}}, \quad (3.16)$$

$$f_n(z_n) = \exp \left\{ \frac{v^z_n - \gamma}{2\gamma v^z_n} z^2_n + \frac{m^z_n}{v^z_n} z_n \right\} \propto \frac{q(z_n)}{p(z_n)}\,, \quad (3.17)$$

and $\log Z_q$ is the logarithm of the normalization constant of the exponential Gaussian form of q:

$$\log Z_q = \sum_{l=1}^{L} \sum_{i=1}^{V_l} \sum_{j=1}^{V_{l-1}+1} \left[\frac{1}{2} \log \left(2\pi v^w_{i,j,l} \right) + \frac{\left(m^w_{i,j,l} \right)^2}{v^w_{i,j,l}} \right] + \sum_{n=1}^{N} \left[\frac{1}{2} \log \left(2\pi v^z_n \right) + \frac{(m^z_n)^2}{v^z_n} \right]. \quad (3.18)$$

The scalable optimization of (3.15) is done in practice by using stochastic gradient descent. For this, we subsample the sums for $n = 1, \ldots, N$ in (3.15) and (3.18) using mini-batches and approximate the expectations over q in (3.15) with an average over K samples drawn from q. We can then use the reparametrization trick, first proposed by Kingma et al. (2015), to obtain gradients from the resulting stochastic approximation to (3.15). The hyper-parameters $\mathbf{\Sigma}$, λ and γ can also be tuned by minimizing (3.15), a process called type-II maximum likelihood we discussed in the previous chapter. In practice we only tune $\mathbf{\Sigma}$ and keep $\lambda = 1$ and $\gamma = \sqrt{d}$. The latter means that the prior scale of each z_n grows with the data dimensionality. This guarantees that, a priori, the effect of each z_n in the neural network's output does not diminish when more and more features are available.

The predictive distribution of a BNN+LV for the target variable $y_\star$ associated with the test data point $\mathbf{x}_\star$ is

$$p(y_\star|\mathbf{x}_\star) = \int p(y_\star|\mathcal{W}, \mathbf{x}_\star, z_\star)p(z_\star)q(\mathcal{W})\, dz_\star\, d\mathcal{W}\,, \tag{3.19}$$

where $p(y_\star|\mathcal{W}, \mathbf{x}_\star, z_\star) = \mathcal{N}(y_\star|f(\mathbf{x}_\star, z_\star; \mathcal{W}), \mathbf{\Sigma})$ is the likelihood function, $p(z_\star) = \mathcal{N}(z_\star|0, \gamma)$ is the prior on the latent variables and $q(\mathcal{W})$ is the approximate posterior for $\mathcal{W}$ given $\mathcal{D}$.

3.1.3 Algorithmic Design Decisions

In this section we want to list a set of hyper-parameter and design choices that we observed as relevant for training BNN+LV. Some of these carry over from training ordinary BNN via α-divergence minimization (Hernández-Lobato et al., 2016).

Optimizer The energy function given by Eq. (3.15) can be minimized using standard gradient-based techniques that are also used in training neural networks. Having said that, in applications we notice that the right choice of the learning algorithm and the mini-batch size can improve the minimization process: Firstly, using an optimization algorithm such as Adam (Kingma and Ba, 2014) that is adaptive, in the sense that it learns an individual learning rate for each parameter. In BNN+LV we have a heterogeneous set of parameters, where each set models something qualitatively different:

- the weight means $\mathbf{m}^w$ are responsible for function approximation.
- the weight variances $\mathbf{v}^w$ model the uncertainty over the function approximation.
- the parameters of the latent variables $\mathbf{m}^z, \mathbf{v}^z$ model the effect of randomness in the training data. Each data point $(\mathbf{x}_n, \mathbf{y}_n)$ has an associated pair of parameter (m_n^z, v_n^z).
- Hyper-parameters such as $\mathbf{\Sigma}$ which estimates the constant noise level. Optionally, the parameters of the prior of the weights and latent variables λ, γ can be included in the optimization process.

Because each parameter set is qualitatively different, an algorithm with merely a global learning rate may only focus on a small subset of parameters that is most volatile. This can lead to suboptimal convergence.

The second adjustment is to prefer a large mini-batch size. While the algorithm is defined to also work with small batch sizes, to estimate the noise parameter $\mathbf{m}^z$ a form of self-organization over the parameters $\mathbf{m}^z$ has to take place. For instance, in a bimodal problem one solution would be to have all data that lie in the first mode to have a positive associated latent variable, whereas for data that lies in the second mode the latent variable then should be negative. Because every data point has an independent pair of parameters a large mini-batch size can help facilitate this form of self-organization.

Initialization We use the following starting values: the weight parameters $\mathbf{m}^w, \mathbf{v}^w$ and the output noise $\boldsymbol{\Sigma}$ are set close to zero, for the weight means $\mathbf{m}^w$ we add Gaussian noise with standard deviation of 0.1. The parameters of the latent variables, these are $\mathbf{m}^z, \mathbf{v}^z$, are set close to the prior $\mathcal{N}(z_\star|0, \gamma)$.

With the initialization of these variables we aim to induce a stage-wise behavior in the learning process. Because the weight and latent variable variances are close to zero, as well as the output noise, in the first iterations the BNN+LV is expected to behave similarly to a standard neural network. This is because all sources of uncertainty are removed bar some non-structured input noise coming from z. Therefore in the beginning the training process will focus on learning the functional relationship between the inputs $\mathbf{X}$ and outputs $\mathbf{Y}$. Only then it will start modeling the uncertainty over the weights, via $\mathbf{v}^w$, and model the stochastic effects in the data, via $\mathbf{m}^z, \mathbf{v}^z$.

Parameter Representation The variance parameter $\mathbf{v}^w$ and $\mathbf{v}^z$ by definition have to be positive. Furthermore, looking at Eq. (3.15), the gradients may be unstable if the variational parameters are the same as the prior parameters or if a large gradient step will make the variances go negative. Because of this we represent these two parameters in logit space, optimizing $\text{logit\_}\mathbf{v}^w, \text{logit\_}\mathbf{v}^z$ such that

$$\mathbf{v}^w = \text{sigmoid}(\text{logit\_}\mathbf{v}^w)\lambda \,. \tag{3.20}$$

By this the variance of each weight can only reach the prior asymptotically and never exactly. Furthermore, the optimization of $\text{logit\_}\mathbf{v}^w$ is unconstrained, whereas $\mathbf{v}^w$ would be restricted to $\mathbb{R}^+$.

3.1.4 Amortized Inference

In Eq. (3.12) we approximate a posterior over latent variables for every training data point $(\mathbf{x}_i, \mathbf{y}_i)$. This can be problematic for two reasons:

1. The memory requirement of this algorithm grows linearly with the size of N.
2. In optimization we have a mix of global and local parameters.

The second reason requires an explanation: During training we optimize the parameters of the variational distribution of the weights and the variational distribution over the latent variables, as shown in Eq. (3.14). Any particular data point or mini-batch will potentially influence $q(\mathcal{W})$, but only a single data point will influence each $q(z_i)$. This means if we optimize the energy, given by Eq. (3.15) by stochastic gradient descent, the gradient w.r.t. the parameters of $q(\mathcal{W})$ will be dense, whereas w.r.t. $q(z_i)$ it will be sparse. This can make convergence of the algorithm slow. An additional problem lies in the application to computer vision domains. Here, numerous preprocessing steps are utilized that improve performance in practice. Some of them will generate novel (not part of the original data set) images by applying transformations such as shifting or perturbing pixels. This potentially leads to very large (theoretically infinitely-large) effective data sets.

A solution to this problem is given by *amortized inference*. The main idea is "to solve many similar inference problems, and [..] thus offload part of the computational work to shared precomputation and adaptation over time" (Stuhlmüller et al., 2013). Such a process can be performed by an *inference network* (Kingma and Welling, 2013). In the context of BNN+LV this means, instead of learning the parameters of the posterior $q(z_i)$ of each latent variable, we learn a joint set of parameters $\mathcal{W}_z$ of a function, that given the context $(\mathbf{x}_n, \mathbf{y}_n)$ outputs the values of the two, formerly free, parameter (m_n^z, v_n^z). By that we have "amortized" the process of inferring the latent variables. In particular:

$$q(\mathbf{z}) = \prod_{n=1}^{N} \mathcal{N}(z_n \,|\, m_n^z, v_n^z) \tag{3.21}$$

$$\approx \prod_{n=1}^{N} \mathcal{N}(z_n | m_n^z, v_n^z = f(\mathbf{x}_n, \mathbf{y}_n; \mathcal{W}_z)) \,. \tag{3.22}$$

Consequently, we now only optimize the weights $\mathcal{W}_z$ and the parameters of $q(\mathcal{W})$ in the inference process. This solves the aforementioned problems in the optimization process. However, using amortization introduces an additional source of bias in the variational approximation in addition to the already existing bias originating from mean-field approximation of the posterior distribution. We refer to Cremer et al. (2018) for a more detailed discussion and study of these two biases.

3.2 Uncertainty Decomposition

We have introduced a new probabilistic model by augmenting a Bayesian neural network with latent variables. The model is designed to not only maintain uncertainty over its parameters but also to model the noise in the data. In the predictive distribution of BNN+LV given by Eq. (3.19), the randomness or uncertainty on $\mathbf{y}_\star$ has its origin in $\mathcal{W} \sim q(\mathcal{W})$, $z_\star \sim p(z_\star)$ and $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$.

The role of the latent variables is to model the noise in the data, whereas the uncertainty in the weights given by $q(\mathcal{W})$ represents the uncertainty about the function f. At the beginning of this chapter we introduced the two forms of uncertainty, epistemic,

which is reducible and aleatoric, which is irreducible uncertainty. In the predictions of $\mathbf{y}_\star$ the aleatoric uncertainty originates from the randomness of $z_\star$ and $\boldsymbol{\epsilon}$ and cannot be reduced by collecting more data. By contrast, the epistemic uncertainty originates from the randomness of $\mathcal{W}$ and can be reduced by collecting more data, which will typically shrink the approximate posterior $q(\mathcal{W})$.

Eq. (3.19) is the tool to use when making predictions for $\mathbf{y}_\star$. However, there are many settings in whic for decision-making purposes, we may be interested in separating the two forms of uncertainty present in this distribution. In later chapters of this thesis, we will show a set of tasks where this is the case. The main motivation of this section is: How can we disentangle these different forms of uncertainty?

In Chapter 2 we discussed different metrics to quantify predictive uncertainty. We can extend the metrics to disentangle the epistemic and aleatoric components. Let $\mathrm{H}(\cdot)$ compute the differential entropy of a probability distribution. The total uncertainty present in Eq. (3.19) can then be quantified as $\mathrm{H}(\mathbf{y}_\star|\mathbf{x}_\star)$. Let us assume that we do not integrate $\mathcal{W}$ out in Eq. (3.19) and, instead, we just condition on a specific value of this variable. The result is then

$$p(\mathbf{y}_\star|\mathcal{W}, \mathbf{x}_\star) = \int p(\mathbf{y}_\star|\mathcal{W}, \mathbf{x}_\star, z_\star) p(z_\star)\, dz_\star \,, \tag{3.23}$$

with corresponding uncertainty $\mathrm{H}(\mathbf{y}_\star|\mathcal{W}, \mathbf{x}_\star)$. The expectation of this quantity under $q(\mathcal{W})$, that is, $\mathbf{E}_{q(\mathcal{W})}[\mathrm{H}(\mathbf{y}_\star|\mathcal{W}, \mathbf{x}_\star)]$, can then be used to quantify the overall uncertainty in Eq. (3.19) coming from $z_\star$ and ϵ. Therefore, $\mathbf{E}_{q(\mathcal{W})}[\mathrm{H}(\mathbf{y}_\star|\mathcal{W}, \mathbf{x}_\star)]$, measures the aleatoric uncertainty. We can then quantify the epistemic part of the uncertainty in Eq. (3.19) by computing the difference between total and aleatoric uncertainties:

$$\mathrm{H}[\mathbf{y}_\star|\mathbf{x}_\star] - \mathbf{E}_{q(\mathcal{W})}[\mathrm{H}(\mathbf{y}_\star|\mathcal{W}, \mathbf{x}_\star)] = I(\mathbf{y}_\star, \mathcal{W}) \,, \tag{3.24}$$

which, as indicated, is the mutual information between $\mathbf{y}_\star$ and $\mathcal{W}$.

The decomposition can best be understood visually; we show this in the information diagram in Figure 3.2. The entropy of the predictive distribution of $\mathbf{y}_\star$ is the circle on the right side, composed of the blue, cyan, grey and pink areas. The blue area represents $H(\mathbf{y}_\star|\mathcal{W}, z_\star)$ which is a constant for all inputs $\mathbf{x}_\star$. When both $\mathcal{W}$ and z are given, the entropy of y is given by the entropy of the additive Gaussian noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$. $\mathrm{H}(y|\mathcal{W})$ is given by the light and dark blue areas. This is the aleatoric part of the uncertainty of $\mathbf{y}_\star$ because it is unaffected by the (reducible) uncertainty over the network parameters $\mathcal{W}$. On the other hand, the pink and grey area forms the epistemic uncertainty because it is affected by the uncertainty over the network parameters. These two, together form the mutual information between $\mathcal{W}$ and $\mathbf{y}_\star$.

Instead of the entropy, we can use the variance as a measure of uncertainty. Let $\sigma^2(\cdot)$ compute the variance of a probability distribution. The total uncertainty present in Eq. (3.19) is then $\sigma^2(\mathbf{y}_\star|\mathbf{x}_\star)$. This quantity can then be decomposed using the law of total variance:

$$\sigma^2(\mathbf{y}_\star|\mathbf{x}_\star) = \sigma^2_{q(\mathcal{W})}(\mathbf{E}[\mathbf{y}_\star|\mathcal{W}, \mathbf{x}_\star]) + \mathbf{E}_{q(\mathcal{W})}[\sigma^2(\mathbf{y}_\star|\mathcal{W}, \mathbf{x}_\star)] \,, \tag{3.25}$$

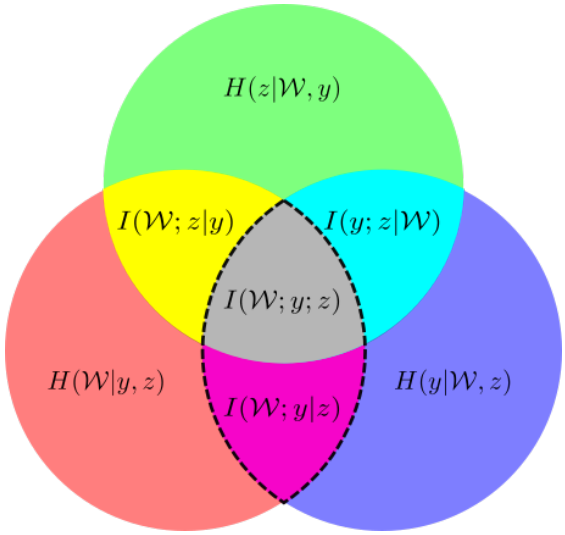


Figure 3.2: Information diagram illustrating quantities of entropy with three variables. The area surrounded by a dashed line indicates the mutual information between $\mathbf{y}_\star$ and $\mathcal{W}$. A conditioning to $\mathbf{x}_\star$ is omitted for readability.

where $\mathbf{E}[\mathbf{y}_\star|\mathcal{W}, \mathbf{x}_\star]$ and $\sigma^2[\mathbf{y}_\star|\mathcal{W}, \mathbf{x}_\star]$ are, respectively, the mean and variance of $\mathbf{y}_\star$ according to $p(\mathbf{y}_\star|\mathcal{W}, \mathbf{x}_\star)$. In the expression above, $\sigma^2_{q(\mathcal{W})}(\mathbf{E}[\mathbf{y}_\star|\mathcal{W}, \mathbf{x}_\star])$ is the variance of $\mathbf{E}[\mathbf{y}_\star|\mathcal{W}, \mathbf{x}_\star]$ when $\mathcal{W} \sim q(\mathcal{W})$. This term ignores any contribution to the variance of $\mathbf{y}_\star$ from $z_\star$ and ϵ and only considers the effect of $\mathcal{W}$. Therefore, it corresponds to the epistemic uncertainty in Eq. (3.19). By contrast, the term $\mathbf{E}_{q(\mathcal{W})}[\sigma^2(\mathbf{y}_\star|\mathcal{W}, \mathbf{x}_\star)]$ represents the average value of $\sigma^2(\mathbf{y}_\star|\mathcal{W}, \mathbf{x}_\star)$ when $\mathcal{W} \sim q(\mathcal{W})$. This term ignores any contribution to the variance of $\mathbf{y}_\star$ from $\mathcal{W}$ and, therefore, it represents the aleatoric uncertainty in Eq. (3.19).

In some cases, working with variances can be undesirable because they have square units. To avoid this problem, we can work with the square root of the previous terms. For example, we can represent the total uncertainty using

$$\sigma(\mathbf{y}_\star|\mathbf{x}_\star) = \left\{\sigma^2_{q(\mathcal{W})}(\mathbf{E}[\mathbf{y}_\star|\mathcal{W}, \mathbf{x}_\star]) + \mathbf{E}_{q(\mathcal{W})}[\sigma^2(\mathbf{y}_\star|\mathcal{W}, \mathbf{x}_\star)]\right\}^{\frac{1}{2}} . \tag{3.26}$$

We have introduced two approaches to decompose predictive uncertainty into epistemic and aleatoric components. In the chapters that follow we will utilize these decompositions for a set of machine learning applications, such as active learning and risk-sensitive reinforcement learning.

3.3 Sensitivity Analysis of Epistemic and Aleatoric Uncertainty

Extracting human-understandable knowledge out of black-box machine learning methods is an important topic of research. One aspect of this is to figure out how sensitive the

model response is to which input variables. This can be useful both as a sanity check, if the approximated function is reasonable, but also to gain new insights about the problem at hand. For neural networks this kind of model inspection can be performed by a sensitivity analysis, a simple method that works by considering the gradient of the network output with respect to the input variables (Fu and Chen, 1993; Montavon et al., 2018).

Here, we want to transfer this idea towards predictive uncertainty: What features impact the uncertainty in the predictions of our model? Using the uncertainty decomposition, that we have outlined in the previous Section, we can further specify this question to: How sensitive is each feature towards epistemic and aleatoric uncertainty?

Answers to this question can provide useful insights about a model at hand. For instance, a feature with high aleatoric sensitivity indicates a strong interaction with other unobserved/latent features. If a practitioner can expand the set of features by taking more refined measurements, it may be advisable to look into variables which may exhibit dependence with that feature and which may explain the stochasticity in the data. Furthermore, a feature with high epistemic sensitivity, suggests careful monitoring or extended safety mechanisms are required to keep this feature values in regions where the model is confident.

We start by briefly reviewing the technique of sensitivity analysis (Fu and Chen, 1993; Montavon et al., 2018), a simple method that can provides insight into how changes in the input affect the network's prediction. Let $\mathbf{y} = f(\mathbf{x}; \mathcal{W})$ be a neural network fitted on a training set $\mathcal{D} = \{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^{N}$, formed by feature vectors $\mathbf{x}_n \in \mathbb{R}^D$ and targets $\mathbf{y}_n \in \mathbb{R}^K$. We want to understand how each feature i influences the output dimension k. Given some test data $\mathcal{D}_{\text{test}} = \{\mathbf{x}_n^\star, \mathbf{y}_n^\star\}_{n=1}^{N_{\text{test}}}$, we use the partial derivate of the output dimension k w.r.t. feature i:

$$I_{i,k} = \frac{1}{N_{\text{test}}} \sum_{n=1}^{N_{\text{test}}} \left| \frac{\partial f(\mathbf{x}_n^\star)_k}{\partial x_{i,n}^\star} \right| . \tag{3.27}$$

In Section 3.2 we have shown how to decompose the variance of the predictive distribution of a BNN+LV into its epistemic and aleatoric components. Our goal is to obtain sensitivities of these components with respect to the input variables. For this we use a sampling based approach to approximate the two uncertainty components and then calculate the partial derivative of these w.r.t. to the input variables. For each test data point $\mathbf{x}_n^\star$, we perform $N_w \times N_z$ forward passes through the BNN. We first sample $w \sim q(\mathcal{W})$ a total of N_w times and then, for each of these samples of $q(\mathcal{W})$, performing N_z forward passes in which w is fixed and we only sample the latent variable z. Then we can do an empirical estimation of the expected predictive value and of the two

components on the right-hand-side of Eq. (3.25):

$$\mathbf{E}[y^\star_{n,k}|\mathbf{x}^\star_n] \approx \frac{1}{N_w}\frac{1}{N_z}\sum_{n_w=1}^{N_w}\sum_{n_z=1}^{N_z} y^\star_{n_w,n_z}(\mathbf{x}^\star_n)_k \tag{3.28}$$

$$\sigma_{q(\mathcal{W})}(\mathbf{E}_{p(z^\star)}[y^\star_{n,k}|\mathcal{W},\mathbf{x}^\star_n]) \approx \hat{\sigma}_{N_w}(\frac{1}{N_z}\sum_{n_z=1}^{N_z} y^\star_{n_w,n_z}(\mathbf{x}^\star_n)_k) \tag{3.29}$$

$$\mathbf{E}_{q(\mathcal{W})}[\sigma^2_{p(z^\star)}(y^\star_{n,k}|\mathcal{W},\mathbf{x}^\star_n)]^{\frac{1}{2}} \approx \big(\frac{1}{N_w}\sum_{n_w=1}^{N_w}\hat{\sigma}^2_{N_z}(y^\star_{n_w,n_z}(\mathbf{x}^\star_n)_k)\big)^{\frac{1}{2}}\,. \tag{3.30}$$

where $y^\star_{n_w,n_z}(\mathbf{x}^\star_n)_k = f(\mathbf{x}^\star_n, z^{n_w,n_z}; \mathcal{W}^{n_w})_k$ and $\hat{\sigma}^2_{N_z}$ ($\hat{\sigma}^2_{N_w}$) is an empirical estimate of the variance over N_z (N_w) samples of z ($\mathcal{W}$). We have used the square root of each component so all terms share the same unit of $y^\star_{n,k}$. Now we can calculate the sensitivities:

$$I_{i,k} = \frac{1}{N_{\text{test}}}\sum_{n=1}^{N_{\text{test}}}\big|\frac{\partial \mathbf{E}[y^\star_{n,k}|\mathbf{x}^\star_n])}{\partial x^\star_{i,n}}\big| \tag{3.31}$$

$$I^{\text{epistemic}}_{i,k} = \frac{1}{N_{\text{test}}}\sum_{n=1}^{N_{\text{test}}}\big|\frac{\partial \sigma_{q(\mathcal{W})}(\mathbf{E}_{p(z^\star)}[y^\star_{n,k}|\mathcal{W},\mathbf{x}^\star_n])}{\partial x^\star_{i,n}}\big| \tag{3.32}$$

$$I^{\text{aleatoric}}_{i,k} = \frac{1}{N_{\text{test}}}\sum_{n=1}^{N_{\text{test}}}\big|\frac{\partial \mathbf{E}_{q(\mathcal{W})}[\sigma^2_{p(z^\star)}(y^\star_{n,k}|\mathcal{W},\mathbf{x}^\star_n)]^{\frac{1}{2}}}{\partial x^\star_{i,n}}\big|\,, \tag{3.33}$$

where Eq. (3.31) is the standard sensitivity term. We also note that the general drawbacks of the sensitivity analysis, such as considering every variable in isolation, arise due to its simplicity (Montavon et al., 2018). These will also apply when focusing on the uncertainty components.

4 Data Sets and Benchmarks

In the upcoming chapters we will shift the focus from a purely methodological base towards specific application scenarios in machine learning. A part of this is to evaluate the performance of different methods on data sets and benchmarks. In this chapter we want to provide an overview about these and shortly describe their properties and domains associated with each task.

In Section 4.3.2 we introduce the industrial benchmark (IB), which was previously published in Hein et al. (2017a).

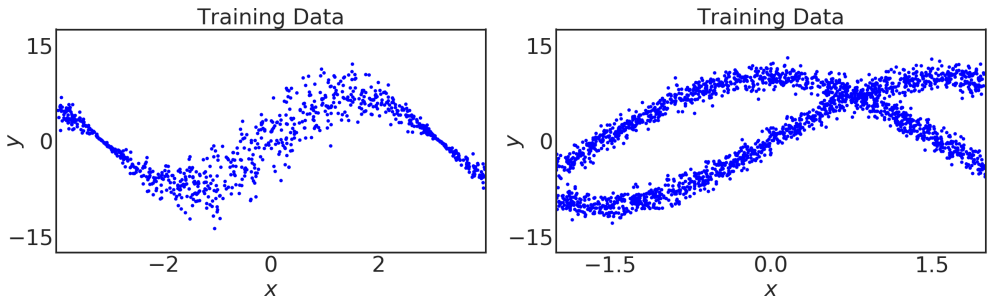
4.1 Standard Regression Benchmarks

Dataset	N	d
Boston Housing	506	13
Combined Power Plant	9568	4
Concrete Strength	1,030	8
Energy Efficiency	768	8
Kin8nm	8,192	8
Naval Propulsion	11,934	16
Wine Quality Red	1,599	11

Table 4.1: Description of the 7 selected data sets. N is is the number of examples and d is the dimensionality of input features.

We obtain a set of standard regression benchmarks from the UCI machine learning repository (Lichman, 2013), a platform that provides a rich pool of data sets. From this set we choose 6 data sets for regression with real-valued inputs and outputs and add another one, the "Kin8nm" data set, from the Delve data set repository (Group, 2019). In Table 4.1 we list these and provide summary statistics for each problem at hand. From the table we can see that the data sets have varying sizes and number of features. We now provide a short description of each, which we obtained from Lichman (2013):

- Boston Housing: predict housing values in suburbs of Boston, based on features such as the pupil-teacher ratio in the area or the accessibility to radial highways.
- Combined Power Plant: predict the hourly electrical energy output if a power plant based on features based on ambient sensors such as temperature or relative humidity.



(a) (b)

Figure 4.1: Example data with heteroscedastic (Fig. 3.1a) and bimodal (Fig. 3.1b) noise.

- Concrete Strength: predict the compressive strength of concrete based on physical measurements such as the amount of cement or the age of the concrete.
- Energy Efficiency: predict the heating load requirement of buildings based on building parameters such as the surface or wall area.
- Kin8nm: Given data from a simulation of 8 link robot arm the task is to predict the distance of the end-effector from a target, based on sensors such as the joint positions and twist angles.
- Naval Propulsion: Given data from a simulated gas turbine operated on a naval vessel the task is to predict the decay state coefficient of the turbine based on features such as the ship speed or temperature measurements of the turbine.
- Wine Quality Red: predict the quality of red wine based on features such as the acidity or pH score of the wine.

4.2 Artificial Benchmark Problems

We consider two artificial benchmark problems with heteroscedastic and bimodal noise patterns. In the heteroscedastic toy problem we sample from

$$y = 7\sin(x) + 3|\cos(x/2)|\epsilon \,, \tag{4.1}$$

where $\epsilon \sim \mathcal{N}(0,1)$ and $x \in [-4,4]$. In Fig. 4.1a we show example data that is sampled from this function uniformly in input space, the plot shows how the noise level varies as a function of x.

In the bimodal toy problem we define

$$y = z10\cos(x) + (1-z)10\sin(x) + \epsilon \,, \tag{4.2}$$

where $\epsilon \sim \mathcal{N}(0,1)$, $z \sim \text{Bern}(0.5)$ and $x \in [-0.5,2]$. 4.1b we show example data that is sampled from this function uniformly in input space, the plot shows how the function varies between a bi- and unimodal noise pattern as a function of x.

4.3 Dynamics Systems

Here we introduce a set of 4 different dynamics systems. These are benchmarks where an agent can potentially interact with by executing actions. These make these benchmarks suitable to study reinforcement learning problems, which we will discuss in more detail in Chapter 7.

These benchmarks were chosen based on two criteria. First, we want to have a set of easily reproducible, artificial benchmarks and a set of real-world scenarios. Secondly, these benchmarks should be expected to have some form of stochasticity as part of their dynamics.

4.3.1 Wet-chicken Benchmark

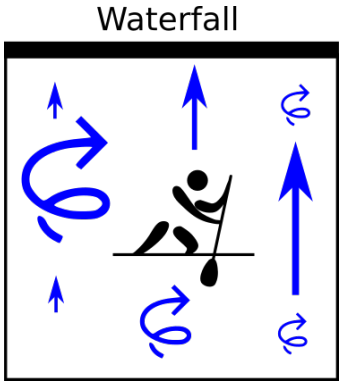


Figure 4.2: Illustration of the wet-chicken problem.

The wet-chicken benchmark (Tresp, 1994) is a stochastic system used for reinforcement learning that presents both bi-modal and heteroscedastic transition dynamics. We use the two-dimensional version of the problem (Hans and Udluft, 2009) and extend it to the continuous case.

In this problem, a canoeist is paddling on a two-dimensional river. The canoeist's position at time t is (x_t, y_t). The river has width $w = 5$ and length $l = 5$ with a waterfall at the end, that is, at $y_t = l$. The canoeist wants to move as close to the waterfall as possible because at time t he gets reward $r_t = -(l - y_t)$. However, going beyond the waterfall boundary makes the canoeist fall down, having to start back again at the origin $(0, 0)$. At time t the canoeist can choose an action $(a_{t,x}, a_{t,y}) \in [-1, 1]^2$ that represents the direction and magnitude of his paddling. The river dynamics have stochastic turbulences s_t and drift v_t that depend on the canoeist's position on the x axis. The larger x_t, the larger the drift and the smaller x_t, the larger the turbulences. The underlying dynamics are given by the following system of equations. The drift and the turbulence magnitude are given by $v_t = 3x_t w^{-1}$ and $s_t = 3.5 - v_t$, respectively. The new location (x_{t+1}, y_{t+1}) is given by the current location (x_t, y_t) and current action

$(a_{t,x}, a_{t,y})$ using

$$x_{t+1} = \begin{cases} 0 & \text{if} \quad x_t + a_{t,x} < 0 \\ 0 & \text{if} \quad \hat{y}_{t+1} > l \\ w & \text{if} \quad x_t + a_{t,x} > w \\ x_t + a_{t,x} & \text{otherwise} \end{cases}, \qquad y_{t+1} = \begin{cases} 0 & \text{if} \quad \hat{y}_{t+1} < 0 \\ 0 & \text{if} \quad \hat{y}_{t+1} > l \\ \hat{y}_{t+1} & \text{otherwise} \end{cases}, \tag{4.3}$$

where $\hat{y}_{t+1} = y_t + (a_{t,y} - 1) + v_t + s_t\tau_t$ and $\tau_t \sim \text{Unif}([-1, 1])$ is a random variable that represents the current turbulence. In Fig. 4.2 we provide an illustration of the benchmark and show visually how turbulence and drift change according to the position of the canoeist.

The wet-chicken dynamics result in complex transition distributions depending on the position. As the canoeist moves closer to the waterfall, the distribution for the next state becomes increasingly bimodal, because when he is close to the waterfall, the change in the current location can be large if the canoeist falls down the waterfall and starts again at $(0, 0)$. The distribution may also be truncated uniform for states close to the borders, because the canoeist may never leave the boundaries $[0, l]^2$. Furthermore the system has heteroscedastic noise, the smaller the value of x_t the higher the noise variance. Because of these properties, the wet-chicken benchmark is especially difficult to approximate using supervised learning techniques.

4.3.2 Industrial Benchmark

The industrial benchmark (IB) is a recently introduced benchmark that includes a variety of aspects found to be vital in industrial applications (Hein et al., 2017a). An implementation of this benchmark is publicly available[1]. While this benchmark is not a direct approximation of any real system, it was designed to pose the same hardness and complexity. Furthermore, the process of searching for an optimal action policy on the IB is supposed to resemble the task of finding optimal valve settings for gas turbines or optimal pitch angles and rotor speeds for wind turbines. The state and action spaces of the IB are continuous and high-dimensional, with a large part of the state being latent to the observer. Formally this makes the IB an instance of a partially observable MDP (POMDP). The dynamical behavior includes heteroscedastic noise and a delayed reward signal that is composed of multiple objectives. The IB is designed such that the optimal policy will not approach a fixed operation point in the three steerings. All of these design choices were driven by experience with industrial challenges.

In this benchmark the hidden Markov state space $\mathbf{s}_t$ consists of 27 variables, whereas the observable state $\mathbf{o}_t$ is only 6 dimensional. This observable state consists of 3 adjustable steering variables A_t: the velocity $v(t)$, the gain $g(t)$ and the shift $s(t)$. We also observe the fatigue $f(t)$ and consumption $c(t)$ that together form the (known) reward function $r(t) = -(3f(t) + c(t))$. The final observable variable of the IB, setpoint p, influences the dynamical behavior of the environment but can never be changed by

[1] `http://github.com/siemens/industrialbenchmark`

actions. An analogy to such a setpoint is, for example, the demanded load in a power plant or the wind speed actuating a wind turbine. Different values of setpoint p will induce significant changes to the dynamics and stochasticity of the benchmark.

An action $a_t \in [0,1]^3$ consists of proposed changes to each of the three steerings variables. Each steering is limited to $[0, 100]$ as follows:

$$a_t = (\Delta v_t, \Delta g_t, \Delta h_t), \tag{4.4}$$

$$v_{t+1} = \max(0, \min(100, v_t + d^{\mathrm{v}} \Delta v_t)), \tag{4.5}$$
$$g_{t+1} = \max(0, \min(100, g_t + d^{\mathrm{g}} \Delta g_t)), \tag{4.6}$$
$$h_{t+1} = \max(0, \min(100, h_t + d^{\mathrm{h}} \Delta s_t)), \tag{4.7}$$

with scaling factors $d^{\mathrm{v}} = 1$, $d^{\mathrm{g}} = 10$, and $d^{\mathrm{h}} = 5.75$. The step size for changing shift is calculated as $d^{\mathrm{h}} = 20 \sin(15^0)/0.9 \approx 5.75$.

After applying action a_t, the environment transitions to the next time step $t + 1$ in which it enters internal state s_{t+1}. State s_t and successor state s_{t+1} are the Markovian states of the environment that are only partially observable to the agent.

4.3.3 Gas Turbine Data

We have a set of 40,000 observations of a 30-dimensional time-series of sensor recordings from a real gas turbine and a cost function that evaluates the performance of the current state of the turbine. The features in the time-series are grouped into three different sets: a set of environmental variables E_t (e.g. temperature and measurements from sensors in the turbine) that cannot be influenced by an agent, a set of variables relevant for the cost function N_t (such as the turbines current pollutant emission) and a set of steering variables A_t that can be manipulated to control the turbine. In particular the steering variables manipulate the combustion valves of the turbine where the air and gas mix is lead into the chamber.

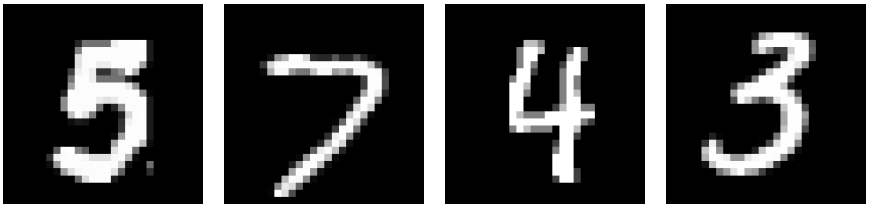
4.3.4 Wind Turbine Simulator

We have a modified version of the HAWC2 wind turbine simulator (Larsen and Hansen, 2007), which is widely used for the study of wind turbine dynamics (Larsen et al., 2015).

In this problem we observe the turbine state $s(t)$, with features such as direction and speed of the wind, temperatures, electric currents and voltages, as well as vibrations produced by major components such as the generator and the rotor blades. The system can be influenced via actions $a(t)$ that adjust the turbine's behavior, with known upper and lower bounds. The goal is to maximize energy output over a one-step horizon.

The system is expected to be highly stochastic due to the unpredictability of future wind dynamics. Furthermore the dimensionality of state observation is much higher than the action dimensionality.

4.4 MNIST Handwritten Digit Data



(a) **(b)** **(c)** **(d)**

Figure 4.3: Four Example images of MNIST handwritten digit data.

The MNIST benchmark is a data set of handwritten digits (LeCun et al., 1998). The data set consists of 60,000 training examples, and a separate test set of 10,000 examples, of grayscale 28×28 images. Each image has an associated label, which are digits from 0 to 9.

In Fig. 4.3 we show 4 example images. We see that for a human observer the correct class of the image is immediately obvious. However, because each image is handwritten, we can expect significant deviations in the images in each class in shape, size, rotation or minor mistakes in the drawing process such as the small white dot in Fig. 4.3a.

5 Accuracy and Uncertainty Calibration in Regression

In this chapter we investigate the predictive performance of BNN+LV in regression. For this we will consider a wide range of tasks, including a set of standard regression benchmarks (see Section 4.1), artificial benchmark problems (see Section 4.2) and data from dynamics systems (see Section 4.3). We will compare the performance of BNN+LV against several baselines, such as neural networks and Gaussian processes. Moreover, we further study the effect of the divergence method used for optimizing BNN+LV, on predictive performance. As a metric we will evaluate both the quality of function approximation using the standard squared error and the quality of uncertainty estimates via the log-likelihood.

This chapter is structured as follows: We start by defining the regression problem and evaluation criteria. We will then specify the model and baselines we consider in this study including hyper-parameters. We then will report the results of our studies and summarize our main findings.

5.1 Problem Description

In regression we are interested in estimating a function from data. We are given data $\mathcal{D} = \{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N$, formed by feature vectors $\mathbf{x}_n \in \mathbb{R}^D$ and targets $\mathbf{y}_n \in \mathbb{R}^K$. The problems we consider here have scalar output, that is $K = 1$.

We split the data set randomly into a training and test set, where we use 90% for training and the remaining 10% of data for testing. For training only the training data set is visible to each learning algorithm and at evaluation we use the test data to measure performance. We repeat this process five times to obtain summary statistics. These statistics are the average performance under a particular metric and the standard error, which is the standard deviation divided by the square root of the number of repetitions. As preprocessing we perform the standard normalization, that is transforming the training data to have unit variance and zero mean. Reported evaluation criteria are computed in unnormalized space.

In Chapter 4 we provided an overview about all data sets and benchmarks we use in this thesis. For this study we will now provide the specific configuration.

1. **Standard Regression Benchmarks**: We use the full data sets that are publicly available[1].

[1] `https://archive.ics.uci.edu`

2. **Artificial Benchmark Problems**: We sample uniformly in input space $n = 1000$ data points for the heteroscedastic and $n = 2500$ data points for the bimodal problem, respectively.

3. **Wet-chicken**: We generate $n = 2500$ state transitions using random actions and predict from the current state $\mathbf{s}(t)$ and action $\mathbf{a}(t)$ the y-dimension of successor state $\mathbf{s}(t+1)$.

4. **Industrial Benchmark**: For each setpoint $p \in \{10, 20, \ldots, 100\}$ we generate 5 trajectories of length 1000, with random exploration. This totals a data set of size $n = 50000$. We predict the reward from a history of observable states, that is:

$$r(t) = f(p, v(t-14), \ldots, v(t), \tag{5.1}$$
$$g(t-14), \ldots, g(t), \tag{5.2}$$
$$h(t-14), \ldots, h(t), , \tag{5.3}$$
$$r(t-15), \ldots, r(t-1)) \ , \tag{5.4}$$

   where v, g, h are the three steering variables, the velocity $v(t)$, gain $g(t)$ and shift $h(t)$. This time-embedding of input features is done to lower the effect of temporal dependencies in this benchmark (see e.g. Bush and Pineau (2009) for a discussion on this). By that the total input dimension is $d = 61$.

5. **Wind Turbine**: We have a data set available of $n = 10000$ observations with random exploration. We predict the reward signal from the current state $\mathbf{s}(t)$ and current action $\mathbf{a}(t)$.

6. **Gas Turbine**: We have a data set available of $n = 38000$ observations, the visible state has $d = 135$ features. We predict the reward signal from the current state $\mathbf{s}(t)$ and current action $\mathbf{a}(t)$.

Evaluation Criteria

The models we consider in this study provide a distribution over output variables $p(y_\star|\mathbf{x}_\star)$. To assess the quality of this distribution we will use two metrics: the root mean squared error (RMSE) between the true target y_n and the expected value of the model predictive distribution and the log-likelihood (LL) of the data under the model.

Root mean squared error The root mean squared error (RMSE) for a one dimensional target variable y is defined as:

$$E = \sqrt{\frac{1}{N_{\text{test}}} \sum_{n=1}^{N_{\text{test}}} (y_n - E[p(\hat{y}_n|\mathbf{x}_n])^2} \ . \tag{5.5}$$

This metric measures how close the expected value of the predictive distribution of the model is to the data of the test set. We note that for multivariate regression problems

this can be problematic because the output variables can have different scales. In this study this does not pose a problem, because we only consider one-dimensional output variables.

Log-likelihood In the previous chapters we discussed different methods how to model uncertainty in supervised learning. For some of these methods the predictive distribution is a Gaussian. The log-likelihood of a test data point $(\mathbf{x}_\star, y_\star)$ under the model therefore is:

$$\log p(y_\star|\mathbf{x}_\star) = -\frac{1}{2}\log(2\pi\sigma^2(\mathbf{x}_\star)) - \frac{(y_\star - \mu(\mathbf{x}_\star))^2}{2\sigma^2(\mathbf{x}_\star)} \,, \tag{5.6}$$

where $\mu(\mathbf{x}_\star), \sigma^2(\mathbf{x}_\star)$ are, respectively, the expected value and variance given by the model. We see in Eq. (5.6) that the squared error also appears in this quantity, the data point becomes more likely if the prediction and the target value are close, weighted by the how certain or uncertain the model is in its prediction. In other methods the predictive distribution is a Gaussian mixture. In this case the log-likelihood is:

$$\ln p(y_\star|\mathbf{x}_\star) = \ln \frac{1}{K}\sum_{k=1}^{K} \exp\big(\frac{1}{2}\log(2\pi\sigma_k^2(\mathbf{x}_\star)) - \frac{(y_\star - \mu_k(\mathbf{x}_\star))^2}{2\sigma_k^2(\mathbf{x}_\star)}\big) \,, \tag{5.7}$$

where K is the number of samples. For every test data point $(\mathbf{x}_n, y_n)$ we will compute the log-likelihood and then average over the test set. This gives us the average test log-likelihood of the test set under the model.

The predictive log-likelihood measures the calibration of uncertainty estimates, and is widely used in standard literature as the primary metric for uncertainty quality. Its main advantage is that this metric does not make assumptions about the shape of the target distribution. By contrast, the RMSE will give non-meaningful results if the test data is highly stochastic, for instance in bimodal cases.

5.2 Model & Baseline Specification

The main method we investigate in this experiment is a **BNN+LV** trained via α-divergence minimization with default $\alpha = 1$. Here, we will compare this method to a set of baselines. We will first describe the individual baseline methods and then specify hyper-parameters we use for training. The list of baselines are:

1. Standard neural networks (**MLP**): As outlined in Section 2.4.1 we train a MLP using early stopping. We form the predictive distribution using the validation error as we described in the aforementioned section. By that the predictive distribution of the MLP is a homoscedastic Gaussian given by Eq. (5.6).

2. MLP Ensemble (**MLP Ens.**): The ensemble consists of 25 neural networks by running the training process of the MLP 25 times in parallel. The predictive distribution of the MLP ensemble is then a mixture of Gaussians (see Eq. (2.55)) and we obtain the log-likelihood via Eq. (5.7), with $K = 25$.

3. Gaussian processes (**GP**): We use GPs with a standard RBF kernel. We refer to Section 2.3 for more details on GPs. For large data sets (these are: Industrial Benchmark, Kin8nm and Naval Propulsion) standard GPs will not work, due to their limited scalability. We then use sparse GPs as described, where we set the number of inducing points to 100.

4. Bayesian neural networks (**BNN**): BNN is trained with $\alpha = 1$ but does not have a latent variable model, it is trained via the mechanism described in Section 2.4.3, in particular using Eq. (2.65).

In a second study we will also investigate different choices of divergences in training BNN+LV. In particular we will compare choices for $\alpha \in \{10^{-6}, 0.5, 1.0\}$. The most prominent approach in training modern BNNs is to optimize the variational lower bound as described in Section 2.4.3, in particular by MC-VB given by Eq. (2.62). In practice we can approximate this using α-divergence minimization when $\alpha \to 0$ (Hernández-Lobato et al., 2016). In our experiments we use BNN+LV with $\alpha = 10^{-6}$ and call this method variational Bayes (VB). We further consider $\alpha = 0.5$ because is a special case in the α-divergence: it is the only metric that is symmetric and is expected to achieve a balance between the exclusive (VB) and inclusive KL-divergence.

For all architectures based on neural networks, that is the MLP, the ensemble of MLPs, BNNs and BNN+LV we use two hidden layers with linear rectifier as activation function. We use 20 hidden units per layer for the artificial benchmark problems and the wet-chicken benchmark and 50 hidden units everywhere else. For these methods we train for 5000 epochs with a learning rate 0.001 for BNN(+LV) and 0.0001 for the MLP and MLP ensemble. As an optimizer we use Adam (Kingma and Ba, 2014). For the GP we use the standard implementation provided by GPy[2].

5.3 Experiments

We show the results for test log-likelihood and RMSE in Table 5.1. Overall we see that BNN+LV perform best in terms of log-likelihood whereas ensembles of neural networks perform best in terms of RMSE.

Looking into test log-likelihood first, we find the advantage of BNN+LV is most prominent in dynamic systems and artificial benchmark problems, where there is a clear improvement over all baselines. In the standard regression benchmarks, the MLP ensemble, BNN and BNN+LV all seem perform equally on average (in fact their average rank, would we only consider this set of benchmarks has the same value of 2.43). We believe that most of these problems are largely deterministic, except for the "Wine Quality Red" data set. Presumably, the features in this problem that describe a given wine are not sufficient to predict the quality, resulting in stochasticity in the data. On the other hand in the artificial benchmark problems and dynamic systems, there is a high degree of stochasticity involved, and consequently, BNN+LV outperform all baselines, and in particular the BNN, which has the same properties bar the latent variables.

[2] `https://github.com/SheffieldML/GPy`

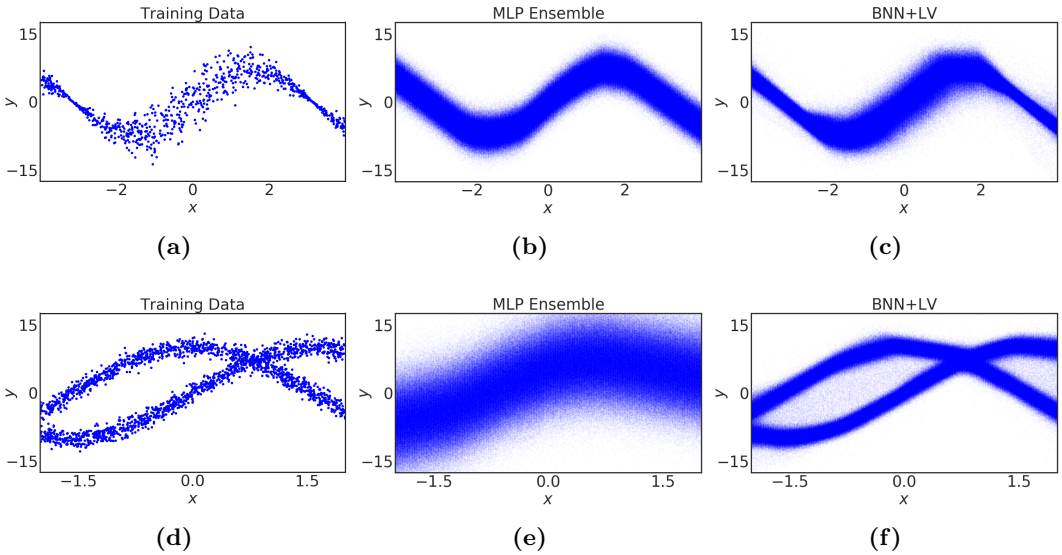


Figure 5.1: Predictive Distribution of MLP ensemble and BNN+LV on artificial benchmark problems.

Considering the test RMSE in Table 5.1, we find that the MLP ensemble is performing best throughout most of the tasks. The improvement over the standard MLP confirms an insight into the advantage of ensembling in supervised which is known since decades (e.g Hansen and Salamon (1990)). Under this metric, also BNN and BNN+LV outperform the standard MLP, albeit not by a wide margin. We note, however, that for problems with strong stochasticity, the RMSE alone is not a good metric quantify to assess predictive performance. For instance, in the heteroscedastic problem the ensemble of MLPs outperforms the BNN+LV in terms of test error. However, when looking at their predictive distributions, shown in Figure 5.1, we see that the BNN+LV has learned both the heteroscedastic (Fig. 5.1c) and the bimodal (Fig. 5.1f) structure of the noise. By contrast, the ensemble in both problems did not capture heteroscedasticity (Fig. 5.1b) or bimodality (Fig. 5.1e).

As a second study we investigate the effect of the divergence measure used in α-divergence minimization for BNN+LV, which is parameterized by the hyper-parameter α. The results are summarized in Table 5.2, where we included the results for the MLP ensemble for better comparison. Overall we see that the divergence affects the results on both RMSE and log-likelihood. In particular, it seems that for lower α (such as VB) the RMSE decreases, whereas the log-likelihood decreases as well. Similarly, for $\alpha = 1.0$ we obtain a higher RMSE but better log-likelihood values. $\alpha = 0.5$ seems to obtain a favorable trade-off is achieved that performs quite well in both scenarios. Overall, the relationship of BNN+LV towards ensembles of neural networks with respect to the test metrics is relatively constant for all values of α. MLP ensembles still outperform

BNN+LV under the RMSE metric, while for log-likelihood BNN+LV perform better (except for VB).

5.4 Discussion

Overall, we found that BNN+LV showed a significant increase in predictive performance in the presence of noise. The improvement in real-world dynamic systems suggests that these systems have stochastic state transitions, making BNN+LV promising candidates as a model for these systems. In terms of approximation quality in terms of RMSE, BNN+LV are similar to GPs and BNNs, whereas ensembles of neural networks perform better. We believe ensembles outperform BNN(+LV) in terms of approximation due to higher diversity. The variational distribution of BNN(+LV) consists of only a pair parameters, namely the means and variances for each weight $w_{i,j,l}$, whereas for the ensemble we have a total of $K = 25$ parameters for each weight. This means in weight space, more diverse solutions can be modeled and higher diversity in ensembles is linked to better generalization (Brown, 2004).

Comparing different α-divergences we found that the choice of α seems to act as a trade-off between the quality of function approximation, measured by RMSE, and uncertainty calibration, measured by the log-likelihood. This finding has also been reported in recent publications, such as Hernández-Lobato et al. (2016) or Li and Gal (2017). Setting $\alpha = 0.5$ appears to be a promising candidate as default setting for the divergence to minimize in variational inference for supervised learning.

	Dataset	MLP	MLP Ens.	Method GP	BNN	BNN+LV
	RMSE					
Standard Regression Benchmarks	Boston Housing	2.90±0.30	**2.50±0.34**	2.68±0.34	3.14±0.33	3.33±0.35
	Combined Power Plant	3.91±0.07	**3.78±0.07**	4.05±0.07	4.06±0.10	3.97±0.07
	Concrete Strength	4.98±0.35	**4.58±0.32**	5.56±0.23	4.94±0.18	5.18±0.21
	Energy Efficiency	0.57±0.02	0.50±0.02	**0.48±0.02**	0.52±0.02	0.52±0.03
	Kin8nm	0.07±0.00	**0.07±0.00**	0.08±0.00	0.07±0.00	0.07±0.00
	Naval Propulsion	0.01±0.00	0.01±0.00	**0.00±0.00**	0.08±0.01	0.25±0.11
	Wine Quality Red	0.64±0.02	**0.61±0.02**	0.62±0.02	0.67±0.02	0.66±0.02
Artificial Problems	Heteroscedastic	1.83±0.10	1.82±0.11	**1.82±0.11**	1.83±0.12	1.83±0.11
	Bimodal	5.20±0.07	5.19±0.06	5.18±0.06	5.20±0.06	**5.18±0.06**
Dynamic Systems	Wet-chicken	1.40±0.02	**1.39±0.02**	1.40±0.02	1.43±0.02	1.40±0.02
	Industrial Benchmark	1.87±0.02	1.77±0.02	1.86±0.01	1.77±0.02	**1.71±0.01**
	Wind Turbine	0.10±0.00	**0.06±0.00**	1.01±0.00	0.07±0.00	0.07±0.00
	Gas Turbine	0.26±0.00	**0.23±0.00**	0.30±0.00	0.26±0.00	0.27±0.00
	Avg. Rank	3.38±0.15	**1.75±0.14**	3.46±0.18	3.23±0.14	3.17±0.17
	Log-Likelihood					
Standard Regression Benchmarks	Boston Housing	-2.52±0.10	**-2.39±0.06**	-2.45±0.14	-2.47±0.04	-2.54±0.04
	Combined Power Plant	-2.78±0.02	-2.74±0.02	-2.82±0.02	-2.72±0.02	**-2.68±0.02**
	Concrete Strength	-3.09±0.10	**-2.92±0.04**	-3.12±0.03	-2.95±0.04	-2.95±0.03
	Energy Efficiency	-0.88±0.05	-0.76±0.02	-0.71±0.05	**-0.60±0.05**	-0.60±0.05
	Kin8nm	1.20±0.01	1.27±0.01	1.02±0.01	1.31±0.01	**1.31±0.01**
	Naval Propulsion	7.58±0.08	7.73±0.01	**9.44±0.01**	5.87±0.14	5.46±0.39
	Wine Quality Red	-0.98±0.03	-0.92±0.03	-0.94±0.04	-0.57±0.09	**2.22±0.09**
Artificial Problems	Heteroscedastic	-2.05±0.04	-2.02±0.05	-2.03±0.05	-1.69±0.05	**-1.67±0.06**
	Bimodal	-3.07±0.01	-3.07±0.01	-3.07±0.01	-2.39±0.17	**-2.07±0.02**
Dynamic Systems	Wet-chicken	-1.75±0.01	-1.75±0.01	-1.76±0.01	-0.89±0.02	**-0.62±0.02**
	Industrial Benchmark	-4.35±0.01	-4.26±0.01	-4.33±0.01	-3.66±0.00	**-3.59±0.01**
	Wind Turbine	0.90±0.01	1.08±0.00	-1.43±0.00	1.36±0.04	**1.40±0.04**
	Gas Turbine	-0.05±0.01	0.04±0.00	-0.21±0.00	0.08±0.01	**0.17±0.01**
	Avg. Rank	4.14±0.11	2.91±0.10	3.92±0.16	2.25±0.12	**1.78±0.16**

Table 5.1: Test RMSE and log-likelihood in regression tasks. Reported are average values over 5 independent runs with standard errors. For the naval propulsion task the RMSE is multiplied by a factor of 100 and for the industrial benchmark by a factor of 0.1 to improve readability. All values are rounded to two positions behind the decimal point.

	Dataset	Method VB	$\alpha = 0.5$	$\alpha = 1.0$	MLP Ens.
Standard Regression Benchmarks	**RMSE**				
	Boston Housing	3.34±0.34	3.51±0.48	3.33±0.35	**2.50±0.34**
	Combined Cycle Power Plant	3.98±0.07	4.00±0.07	3.97±0.07	**3.78±0.07**
	Concrete Compression Strength	5.52±0.09	5.40±0.13	5.18±0.21	**4.58±0.32**
	Energy Efficiency	0.51±0.03	0.52±0.04	0.52±0.03	**0.50±0.02**
	Kin8nm	0.07±0.00	0.07±0.00	0.07±0.00	**0.07±0.00**
	Naval Propulsion	0.04±0.01	0.05±0.01	0.25±0.11	**0.01±0.00**
	Wine Quality Red	0.65±0.02	0.68±0.02	0.66±0.02	**0.61±0.02**
Artificial Problems	Heteroscedastic	1.84±0.11	1.83±0.10	1.83±0.11	**1.82±0.11**
	Bimodal	5.19±0.05	5.18±0.05	**5.18±0.06**	5.19±0.06
Dynamic Systems	Wet-chicken	1.41±0.02	1.39±0.02	1.40±0.02	**1.39±0.02**
	Industrial Benchmark	**1.69±0.01**	1.70±0.02	1.71±0.01	1.77±0.02
	Wind Turbine	0.08±0.00	0.07±0.00	0.07±0.00	**0.06±0.00**
	Gas Turbine	0.26±0.00	0.26±0.00	0.27±0.00	**0.23±0.00**
	Avg. Rank	2.80±0.12	2.62±0.11	2.89±0.12	**1.69±0.15**
Standard Regression Benchmarks	**Log-Likelihood**				
	Boston Housing	-2.71±0.05	-2.63±0.02	-2.54±0.04	**-2.39±0.06**
	Combined Cycle Power Plant	-2.72±0.02	-2.70±0.02	**-2.68±0.02**	-2.74±0.02
	Concrete Compression Strength	-3.10±0.01	-3.07±0.02	-2.95±0.03	**-2.92±0.04**
	Energy Efficiency	-0.75±0.03	-0.64±0.09	**-0.60±0.05**	-0.76±0.02
	Kin8nm	1.30±0.01	**1.32±0.01**	1.31±0.01	1.27±0.01
	Naval Propulsion	6.31±0.24	6.09±0.32	5.46±0.39	**7.73±0.01**
	Wine Quality Red	-0.98±0.03	**2.33±0.10**	2.22±0.09	-0.92±0.03
Artificial Problems	Heteroscedastic	-1.71±0.06	-1.68±0.05	**-1.67±0.06**	-2.02±0.05
	Bimodal	-2.12±0.01	-2.07±0.01	**-2.07±0.02**	-3.07±0.01
Dynamic Systems	Wet-chicken	-0.56±0.11	**-0.09±0.04**	-0.62±0.02	-1.75±0.01
	Industrial Benchmark	-3.72±0.03	-3.62±0.01	**-3.59±0.01**	-4.26±0.01
	Wind Turbine	1.18±0.01	1.31±0.01	**1.40±0.04**	1.08±0.00
	Gas Turbine	-0.03±0.00	0.03±0.00	**0.17±0.01**	0.04±0.00
	Avg. Rank	3.11±0.10	2.06±0.10	**1.78±0.11**	3.05±0.15

Table 5.2: Test RMSE and log-likelihood in regression tasks for BNN+LV with different values of α. Reported are average values over 5 independent runs with standard errors. For the naval propulsion task the RMSE is multiplied by a factor of 100 and for the industrial benchmark by a factor of 0.1 to improve readability. All values are rounded to two positions behind the decimal point.

6 Model Inspection & Uncertainty Analysis for BNN+LV

In the previous chapter we have tested the predictive performance of BNN+LV on a set of tasks. Our main findings are that these type of models are capable of expressing stochastic patterns in the data resulting in better test-log-likelihood compared to baseline models.

The primary motivation of introducing BNN+LV is that they are able to express the stochasticity in the data, thereby model the aleatoric uncertainty, while still maintaining uncertainty in its parameters, which models epistemic uncertainty. In this chapter we want to take a more detailed look into these two forms of uncertainty in the prediction of BNN+LV. In particular, our analysis will try to address the following questions:

- How much predictive epistemic and aleatoric uncertainty do we observe for different kinds of problems? How does the decomposition change with respect to the size of the network?
- Is the uncertainty decomposition in BNN+LV meaningful, in the sense that it matches our intuition?
- How calibrated is the predictive uncertainty? Can we make a connection to the risk of making an error?
- Can we interpret epistemic and aleatoric uncertainty based on the input features of the problem?
- Can we utilize the decomposition of uncertainty for decision making in active learning?

To answer these questions we will do a set of experiments. We group these experiments into four categories: In Section 6.1 we focus on analyzing predictive uncertainty. We will inspect the decomposition of uncertainty for a set of tasks, which are designed such that we expect a particular form of decomposition to emerge. Also in this section we compare the decomposition for different model sizes of BNN+LV and different divergence measures. In Section 6.2 we will study the calibration of predictive uncertainty by linking it to the test error in regression and classification domains. In Section 6.3 we will present results of the sensitivity analysis for predictive uncertainty, a novel method for model inspection that we have introduced in Section 3.3. This method aims to explain predictive uncertainty in terms of the contribution of each input feature. Lastly, in Section 6.4 we will consider active learning scenarios, to study if and how the decomposition of uncertainty can be utilized for sequential decision making, in simple toy domains.

The methods and experiments presented in this chapter include published results from Depeweg et al. (2017b, 2018, 2017c).

6.1 Analysis of Predictive Uncertainty

In Chapter 3 we have introduced the concepts of epistemic and aleatoric uncertainty. As part of this chapter we introduced a novel probabilistic model, the BNN+LV, to model and separate these two forms of uncertainty. In this section we want to confirm this property empirically on a set of tasks.

6.1.1 Problem Description

While model inspection studies are to an extent subjective, the approach we take is to define experiments where there are clear sources of uncertainty coming from either the data density or randomness. We can then observe if the resulting uncertainty decomposition matches the intuition behind the generative model. Similar model inspection studies into epistemic and aleatoric uncertainties can be found in Kendall and Gal (2017) (Section 5).

Here, we will use two groups of experiments: The first group consists of low-dimensional regression problems. In particular we use the heteroscedastic and bimodal artificial benchmark problems (Sec. 4.2) and the wet-chicken dynamics (Sec. 4.3.1). All three tasks exhibit intrinsic noise in their dynamics so we expect some aleatoric uncertainty in the predictions of the BNN+LV. We further will make the data density non-uniform, so we ideally see varying epistemic uncertainty as function of the input $\mathbf{x}$.

The second group of tasks are derived from the MNIST data set (Sec. 4.4). We consider a total of 5 different experiments:

1. Standard MNIST
2. Object rotation
3. Subsampling data of a subset of classes
4. Applying label noise to a subset of classes
5. Occlusion of peripheral areas in the image

The details of each experiment will be described in Section 6.1.3. In each experiment we will extract and decompose the predictive uncertainty into epistemic and aleatoric components (see Section 3.2). Specifically, in the predictive distribution of BNN+LV we estimate the two uncertainty components given by Eq. (3.2). These are, for every input $\mathbf{x}_\star$ we estimate $\mathbf{E}_{q(\mathcal{W})}[\mathrm{H}(\mathbf{y}_\star|\mathcal{W}, \mathbf{x}_\star)]$, which is the aleatoric uncertainty and $\mathrm{H}[\mathbf{y}_\star|\mathbf{x}_\star] - \mathbf{E}_{q(\mathcal{W})}[\mathrm{H}(\mathbf{y}_\star|\mathcal{W}, \mathbf{x}_\star)]$ which is the epistemic uncertainty. These we can then visualize and interpret for all experiments.

6.1.2 Model Specification

We will use different model architectures for the two groups of experiments and specify the hyper-parameter we use for training here.

For the experiments based on MNIST we use a BNN+LV trained with bb-α with $\alpha = 1.0$ and for the latent variables we use an inference network as described in Section 3.1.4. Concretely, the BNN has three hidden layers with 150 units each, with linear rectifiers for the hidden layer and the softmax activation function for the output layer. The feed-forward topology is $[784, 150, 150, 150, 10]$. The inference network has a single hidden layer with 100 units, with a topology of $[784 + 10, 100, 2]$. The final two nodes output the mean and variance of the latent variable: m_n^z, v_n^z. We train for 150 epochs using Adam as optimizer with a learning rate of 0.001 and a mini-batch size of 250.

For the regression problems we train for 5000 epochs a BNN+LV with two-hidden layer and 20 hidden units per layer. We use linear rectifier as activation function. For training we use Adam as optimizer with a learning rate of 0.001. These are the same settings that we used in Chapter 5.

To estimate the information-theoretic components we repeatedly sample W and z a total of 250 times. In the case of MNIST we can directly compute all information-theoretic components (see Section 3.2) in closed form because the predictive distribution is a categorical distribution. For the regression problems we estimate these using a nearest neighbor approach (Kozachenko and Leonenko, 1987).

6.1.3 Experiments

Group: Regression Problems

We start by doing a set of qualitative comparisons on artificial benchmark problems. Our goal is to empirically confirm that BNN+LV are able to decompose uncertainty in heteroscedastic and bimodal noise settings. We will further an additional study where we will use different network sizes and compare the decomposition of uncertainty for different values of α.

Heteroscedastic Noise We consider a regression problem with heteroscedastic noise where $y = 7\sin(x) + 3|\cos(x/2)|\epsilon$ with $\epsilon \sim \mathcal{N}(0, 1)$. We sample 750 values of the input x from a mixture of three Gaussians with mean parameters $\{\mu_1 = -4, \mu_2 = 0, \mu_3 = 4\}$, variance parameters $\{\sigma_1 = \frac{2}{5}, \sigma_2 = 0.9, \sigma_3 = \frac{2}{5}\}$ and with each Gaussian component having weight equal to $1/3$ in the mixture. Figure 6.1a shows the data. We have many points at the borders and in the center, but few in between.

Figure 6.1 shows the results obtained (see caption for details). The resulting decomposition of predictive uncertainty is very accurate: the epistemic uncertainty (Figure 6.1f), is inversely proportional to the density used to sample the data (Figure 6.1b). This makes sense, since in this toy problem the most informative inputs are located in regions where data is scarce. However, this may not be the case in more complicated settings. Finally, we note that the total predictive uncertainty (Figure 6.1d) fails to identify in-

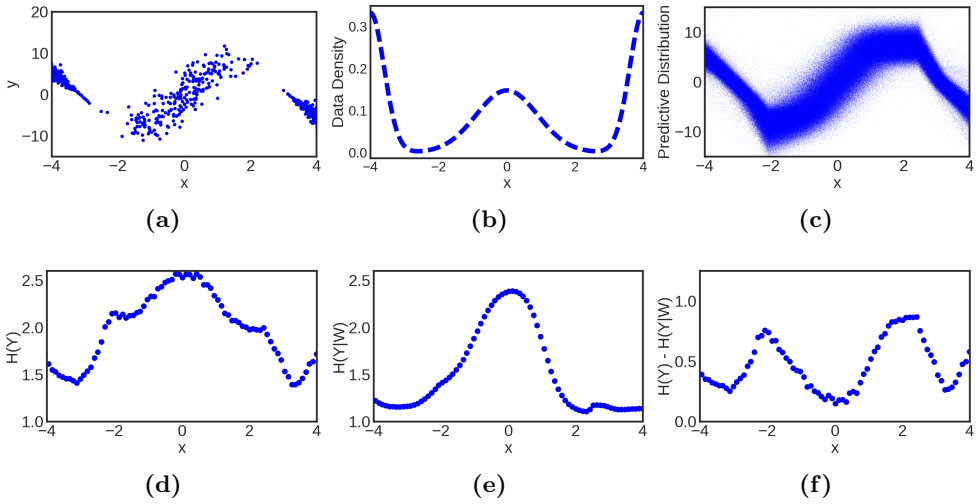


Figure 6.1: Uncertainty decomposition on heteroscedastic data. (a): Raw data. (b): Density of x in data. (c): Predictive distribution $p(y_\star|x_\star)$. (d): Estimate of $\mathrm{H}(y_\star|x_\star)$. (e): Estimate of $\mathbf{E}_{q(\mathcal{W})}\left[\mathrm{H}(y_\star|x_\star, \mathcal{W})\right]$. (f): Estimate of entropy reduction $\mathrm{H}(y_\star|x_\star) - \mathbf{E}_{q(\mathcal{W})}\left[\mathrm{H}(y_\star|x_\star, \mathcal{W})\right]$.

formative regions. In conclusion we observe that the decomposition of uncertainty in BNN+LV allows us to identify informative inputs when the noise is heteroscedastic.

Bimodal Noise Next we consider a toy problem given by a regression task with bimodal data. We define $x \in [-0.5, 2]$ and $y = 10\sin(x) + \epsilon$ with probability 0.5 and $y = 10\cos(x) + \epsilon$, otherwise, where $\epsilon \sim \mathcal{N}(0, 1)$ and ϵ is independent of x. We sample 750 values of x from an exponential distribution with $\lambda = 2$. Figure 6.2a shows the data. We have many points on the left, but few on the right.

Figure 6.2 shows the results obtained (see caption for details). Figure 6.2c shows that the BNN+LV has learned the bimodal structure in the data and Figure 6.2d shows how the total predictive uncertainty increases on the right, where data is scarce. The aleatoric uncertainty (Figure 6.2e), by contrast, has an almost symmetric form around $x = 0.75$, taking lower values at this location. This makes sense since the data generating process is symmetric around $x = 0.75$ and the noise changes from bimodal to unimodal when one gets closer to $x = 0.75$. Figure 6.2f shows an estimate of the epistemic uncertainty, which as expected increases with x. In conclusion we say that the decomposition of uncertainty allows us to identify informative inputs when the noise is bimodal.

Heterogeneous Noise Patterns We consider data sampled from a 2D stochastic system called the wet-chicken (see Section 4.3.1). The wet-chicken transition dynamics exhibit complex stochastic patterns: bimodality, heteroscedasticity and truncation (the agent cannot move beyond the boundaries of state space: $[0, 5]^2$). The data are $7,500$ state transitions collected by random exploration. Figure 6.3a shows the states visited. For

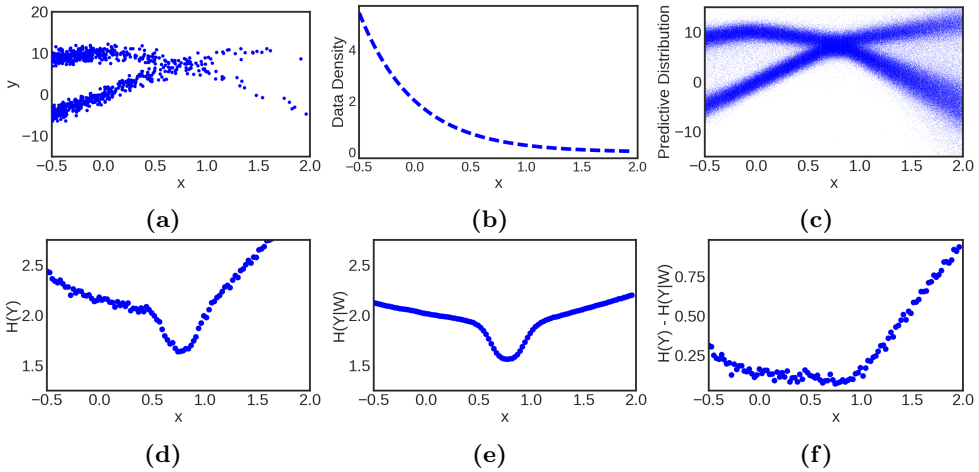


Figure 6.2: Uncertainty decomposition on bimodal data. (a): Raw data. (b): Density of x in data. (c): Predictive distribution: $p(y_\star|x_\star)$. (d): Estimate of $\mathrm{H}(y_\star|x_\star)$. (e): Estimate of $\mathbf{E}_{q(\mathcal{W})}\left[\mathrm{H}(y_\star|x_\star, \mathcal{W})\right]$. (f): Estimate of entropy reduction $\mathrm{H}(y_\star|x_\star) - \mathbf{E}_{q(\mathcal{W})}\left[\mathrm{H}(y_\star|x_\star, \mathcal{W})\right]$.

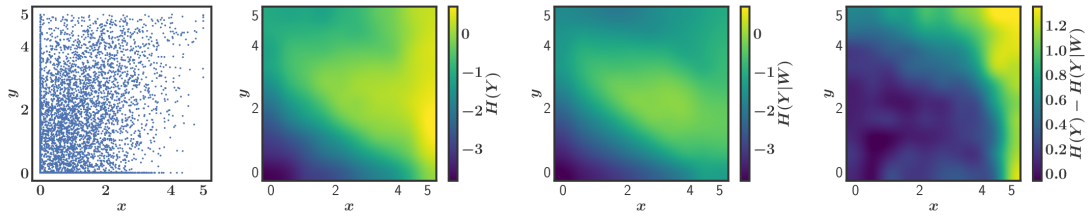


Figure 6.3: Uncertainty decomposition on wet-chicken dynamics. (a): Raw data. (b): Entropy estimate $\mathrm{H}(\mathbf{s}_{t+1}|\mathbf{s}_t)$ of predictive distribution for each $\mathbf{s}_t$ (using $\mathbf{a}_t = \{0, 0\}$). (e): Conditional entropy estimate $\mathbf{E}_{q(\mathcal{W})}\left[\mathrm{H}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathcal{W})\right]$. (f): Estimated entropy reduction.

each transition, the BNN+LV predicts the next state given the current one and the action applied. Figure 6.3d shows that the epistemic uncertainty is highest in the top right corner, while data is most scarce in the bottom right corner. The reason for this result is that the wet-chicken dynamics bring the agent back to $y = 0$ whenever the agent goes beyond $y = 5$, but this does not happen for $y = 0$ where the agent just bounces back. Therefore, learning the dynamics is more difficult and requires more data at $y = 5$ than at $y = 0$. The epistemic uncertainty captures this property, but the total predictive uncertainty (Figure 6.3b) does not.

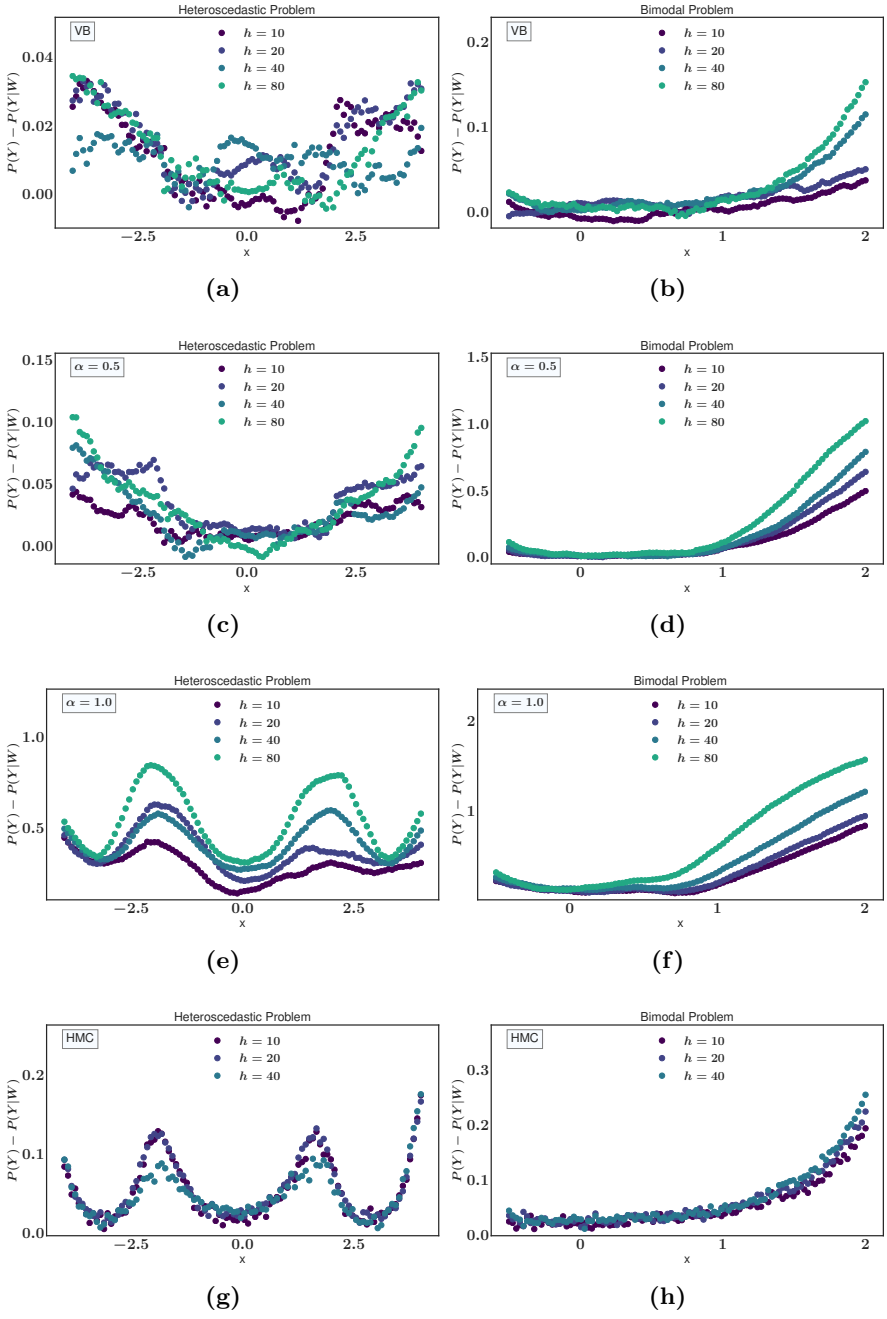


Figure 6.4: Predictive epistemic uncertainty for the heteroscedastic (left) and bimodal (right) problem. Rows 1-3: using variational Bayes ($\alpha = 10^{-6}$) , $\alpha = 0.5$ and $\alpha = 1$ as divergence measure. Row 4: Using Hamiltonian Monte Carlo (HMC), a sampling method, to estimate the unbiased posterior over BNN+LV. In each plot we show the estimated epistemic uncertainty for four different network sizes from smallest (violet) to largest (green). For HMC we only consider network sizes from 10 to 40 due to scalability constraints.

Comparison between network sizes and different values of α In the previous experiments we used a neural network topology with $h = 20$ hidden units that was optimized w.r.t. a α-divergence with $\alpha = 1$. Here, we repeat these experiment for the two stochastic toy problems with varying network sizes and optimization methods. In particular we use $h \in \{10, 20, 40, 80\}$ as number of hidden units and $\alpha = \{10^{-6}, 0.5, 1.0\}$ for different values of α. As in Chapter 5 we use $\alpha = 10^{-6}$ to approximate variational Bayes (VB).

In addition we use the method Hamiltonian Monte Carlo (HMC) as additional baseline. HMC is a sampling method and as we discussed in Section 2.2.3, these methods can be asymptotically unbiased. For the heteroscedastic and bimodal problem, which are small scale, we can use HMC as a ground-truth. After convergence, we can expect that HMC will provide unbiased samples from the true posterior $p(\mathcal{W}, \mathbf{z} \mid \mathcal{D})$ over BNN+LV (that is Eq. (3.10) in Section 3.1), whereas in variational inference we only obtain samples from the approximate posterior $q(\mathcal{W})$. By comparing to this method we can see how large the effect of the variational approximation is on the decomposition of uncertainty. For HMC we use a burn-in of $500k$ and then collect $200k$ samples where we only keep every 10th sample. The decomposition then follows as usual, where instead of a posterior $q(\mathcal{W})$ we have an empirical distribution formed by the samples.

In Figure 6.4 we show the results. We see that for the problem with bimodal noise (right-hand side) the results look very similar across all methods, whereas for the heteroscedastic problem only $\alpha = 1$ and HMC produce a pattern that corresponds to the data density. In addition the epistemic uncertainty in HMC appears to be more peaked and overall lower uncertainty than using $\alpha = 1$, which suggest that while the approximate posterior $q(\mathcal{W})$ leads to a similar decomposition of uncertainty, it seems to overestimates predictive uncertainty.

Group: MNIST Problems

We will now present the result on problems based on the MNIST data set. In each paragraph we will describe the setup of the experiment and then summarize our main findings.

Standard MNIST We consider the MNIST setting, with standard training and test data splits. We find that with a test accuracy is 0.98 this means that BNN+LV give competitive performance on the MNIST data set using a feed-forward architecture. In Fig. 6.5 we show the decomposition of uncertainty for 4 different settings. We see that we have predominately epistemic and a neglectable amount of aleatoric uncertainty. This makes sense because the images of the data set (see Fig. 4.3) in principle do provide enough information about the class; in other words MNIST is not partially observable. Interestingly, for the incorrectly classified data in the test set we have large epistemic uncertainty (see rightmost bar plot in Fig. 6.5). We will investigate this scenario in more detail in Section 6.2.

MNIST with Label Noise In this study we introduce noise on the class labels on the data. In particular, for all training examples with labels of either 1 or 7 we flip the

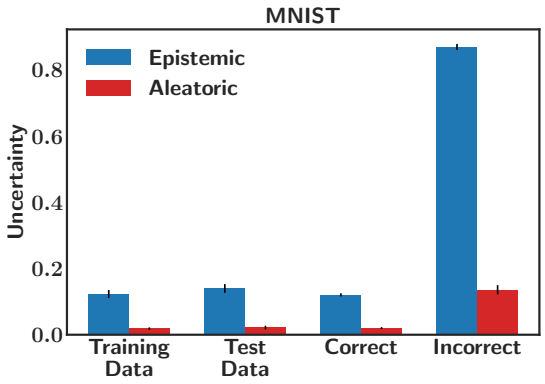


Figure 6.5: Standard MNIST setting. We show the epistemic (blue) and aleatoric (red) predictive uncertainties. From left to right: Uncertainties on training set, uncertainties on test set, uncertainties on correctly classified test images, uncertainties of incorrectly classified test set.

class assignment with a probability of $\frac{1}{3}$, so 33% of images that originally had class 1, now belong to class 7, and vice versa. We choose sevens and ones here, because of their similar visual representation. Note that this noise is heteroscedastic, because it only applies to a subset of classes, all other classes are unaffected. Fig. 6.6 summarizes our

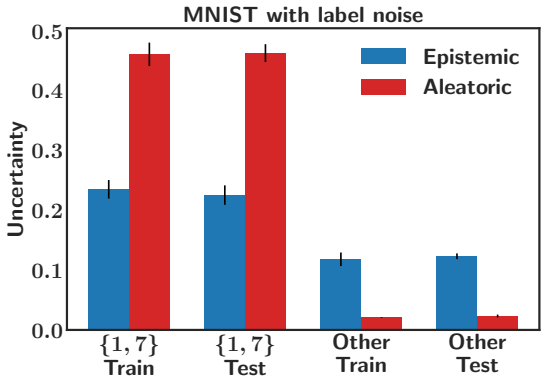


Figure 6.6: MNIST with label noise. We show the epistemic (blue) and aleatoric (red) predictive uncertainties. From left to right: Uncertainties on: training an test set of examples with class 1 or 7, training and test set of examples for all other classes.

findings. We see for those classes affected by label noise we have significantly larger aleatoric uncertainty, whereas in all other classes we have mostly epistemic uncertainty. This corresponds to the design of the experiment: adding heteroscedastic label noise results in a decomposition with increased aleatoric uncertainty.

MNIST: Subsampling Here we subsample particular classes in the training data: for all data with classes of $y \in \{0, 6, 8, 9\}$ we remove 75% of examples from the training set,

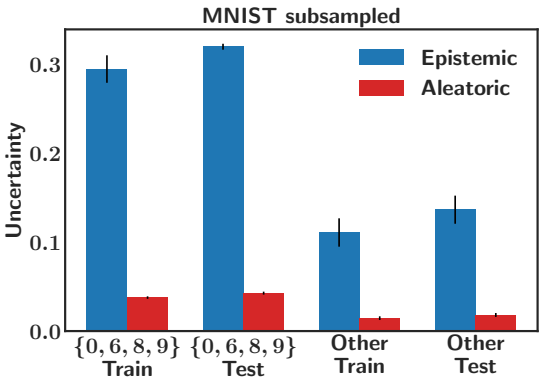


Figure 6.7: MNIST with subsampled classes. We show the epistemic (blue) and aleatoric (red) predictive uncertainties. From left to right: Uncertainties on: training an test set of examples from subsampled classes $y \in \{0, 6, 8, 9\}$, training and test set of examples for all other classes.

whereas all other classes have the original set of examples. Similarly to the previous experiment we choose these classes because of their similar visual representation. The results shown in Fig. 6.7 show that for these classes the epistemic uncertainty is significantly higher than for the remaining ones, which makes sense, as lower data density in these areas leads to higher uncertainty about how the correct function may look like to make the images $\mathbf{x}$ to the classes y.

MNIST: Rotations We consider the effect of rotations on the uncertainties in the predictive distribution. To that end we use the models trained in the standard MNIST setting. Then we evaluate the epistemic and aleatoric uncertainties on test data, where we rotate the objects in 8 steps of 45 degrees. Fig. 6.8 shows two examples of the rotation operation. We show two example results of the decomposition in Fig. 6.9. Predominately, we observe an increase in epistemic uncertainty. This makes sense because a rotation will move a training examples out of the known manifold. In Fig. 6.9b we see a rotation of the digit 9. Here, a rotation by 180 degrees will lead to very low epistemic uncertainty, because in this case the image will look like a 6.

MNIST: Occlusion In this experiment we study the effect of occlusion on the predictive uncertainties. We consider occlusion in the form of masking out the peripheral parts of the image. In particular we consider four occlusion levels and train a BNN+LV and decompose predictive uncertainty in each of them individually. For the original 28 by 28 MNIST image, we restrict the image in both x and y dimension to the following upper and lower bounds of pixels, starting at zero: $\{(.,.), (7, 21), (10, 18), (11, 17), (12, 16)\}$. These occlusion levels are visualized in Fig. 6.10.

In Fig. 6.11a we show the results of the decomposition of uncertainty for various levels of occlusion. We see here that as more and more parts of the images are hidden, the aleatoric uncertainty continuously increases. This makes sense, by the introduction of

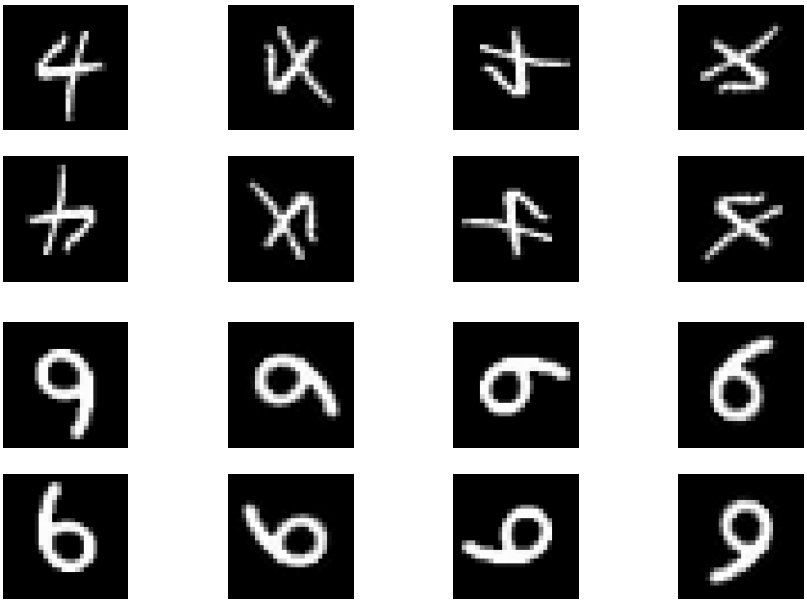
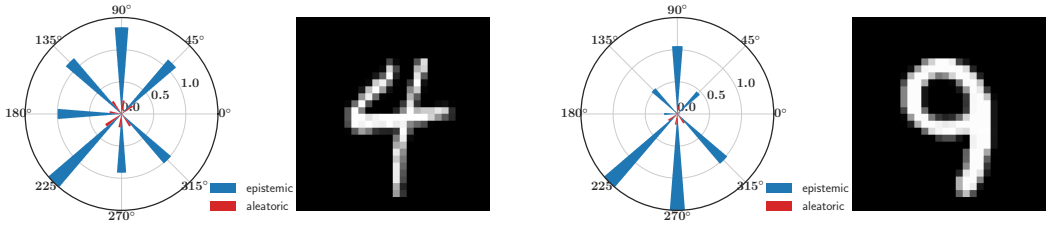


Figure 6.8: Two sets of example images with rotations of 45 degrees in counter-clockwise direction.



(a) **(b)**

Figure 6.9: Epistemic and aleatoric uncertainties for two example images. Left plot: Circular bar plot showing uncertainties as a function of the rotation operation. Right plot: Original image under consideration.

the occlusion the problem becomes gradually more partially observable, the input image does not contain enough features to make a reliable classification. Correspondingly the test accuracy, that we show in Fig. 6.11b decreases as the level of occlusion increases.

6.1.4 Discussion

We studied the decomposition of predictive uncertainty on a set of tasks with varying noise patterns and data densities. Our main finding is that the epistemic and aleatoric uncertainties we extracted from the BNN+LV correspond to the data generating model: the presence of noise patterns in the data results in aleatoric and data scarcity results

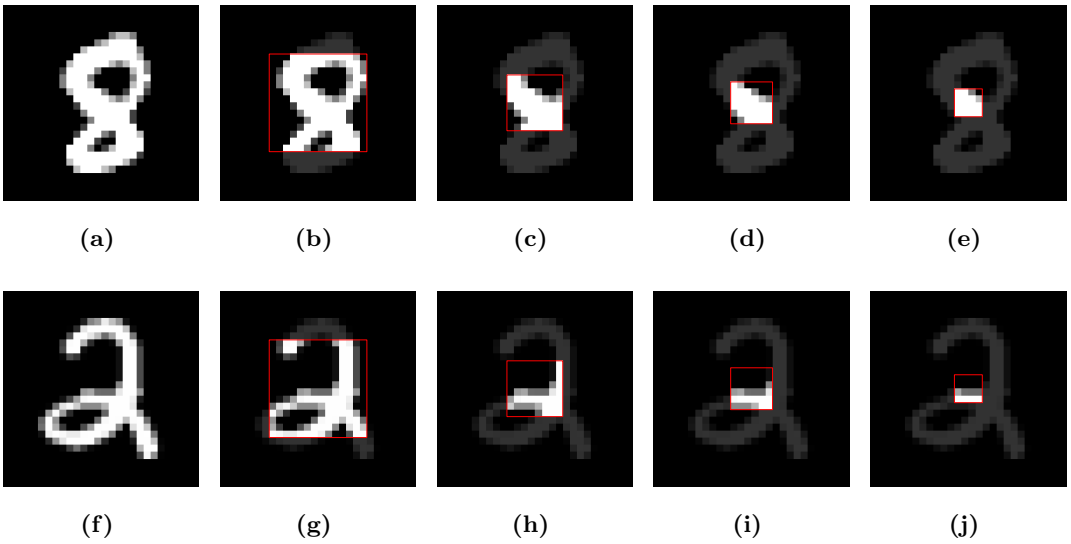


Figure 6.10: Two example images of the MNIST task with different levels of occlusion. From left to right: occlusion level 0 (no occlusion) to level 4 (strongest occlusion). For each level of occlusion the model only receives the area inside the red square as input data.

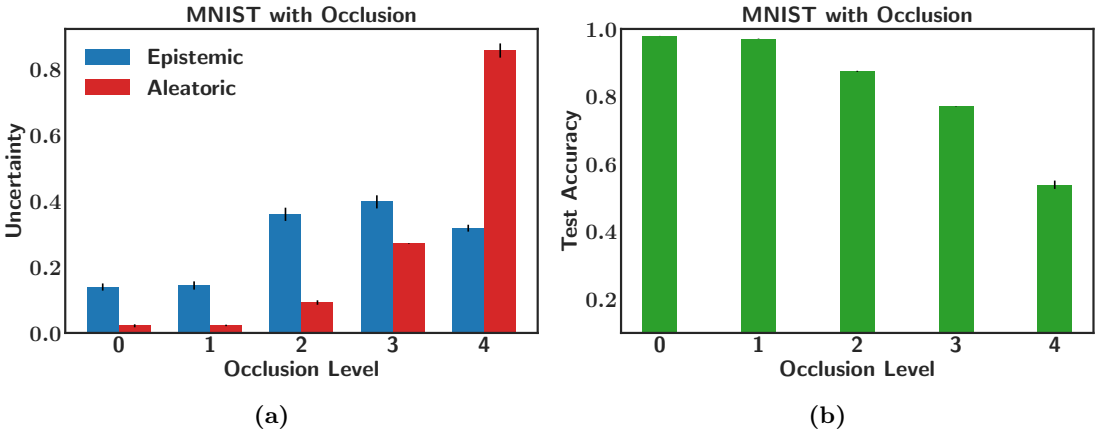


Figure 6.11: MNIST with occlusion. Left plot: epistemic (blue) and aleatoric (red) uncertainties for varying level of image occlusion (x-axis). Right plot: Test accuracy of BNN+LV trained on MNIST with varying level of image occlusion (x-axis).

in epistemic uncertainty. These findings were present both in toy regression and image classification domains. We further observed that the divergence measure we use to train BNN+LV seems to affect these forms of uncertainties. In the default setting of $\alpha = 1$ the decomposition is similar in shape compared to the ground-truth we obtain using HMC in stochastic toy problems. For variational Bayes and $\alpha = 0.5$ this, however, does not

seem to be the case. We will discuss this question further at the outlook of this thesis in Chapter 8.

6.2 Predictive Uncertainty and Test Error

Here we want to study how the uncertainty in the predictive distribution relates to the risk of the model making an error in its predictions at test time. In the previous section we already saw that for the standard MNIST setting, we observed on average high epistemic uncertainty in test data that was classified incorrectly, whereas for correctly classified test data the epistemic uncertainty was low (see Fig. 6.5). This dependency can be beneficial, as predictive uncertainty can be extracted by the model and can, for instance, be used to know beforehand how trustworthy the prediction is going to be. This is related to the concept of *adversarial examples* in computer vision where small change to an input image $\mathbf{x}$, which are non-perceptible to humans, induce a wrong classification with high confidence (Szegedy et al., 2013). Recent work in Gal and Smith (2018) proved that in an idealized, theoretical setting, which includes the absence of any aleatoric uncertainty in the data, a Bayesian neural network cannot have adversarial examples, because the confidence for predicting adversarial examples will always be low due to an increase in epistemic uncertainty.

Intuitively, an error in the model's prediction can occur in three scenarios:

1. The relationship between input and output is stochastic due to aleatoric uncertainty.

2. The model does not know the correct prediction due to limited data, which results in epistemic uncertainty in the predictive distribution.

3. The model is confident in its prediction but it is wrong.

Because the cause of error can be attributed to both epistemic and aleatoric uncertainty, we suggest to use the full predictive uncertainty, that is $\mathrm{H}[\mathbf{y}_\star|\mathbf{x}_\star]$, to connect it to the test error. Measuring calibration of predictive uncertainty already establishes a link between these two concepts. In the regression experiments done in Chapter 5 we measured the log-likelihood of the test data under the model to estimate quality and calibration of uncertainty estimates. For instance, for a Gaussian predictive distribution given by Eq. (5.6) the log-likelihood will be low, if the test error is high, but the model has high confidence (σ^2 is low).

In this section we want to study this relationship in more detail for BNN+LV. We will use the trained models we used in the regression studies from Chapter 5 and from the standard MNIST classification task.

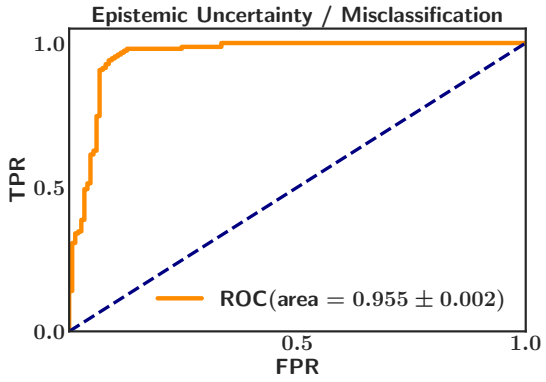


Figure 6.12: ROC curve of predicting misclassification based on epistemic uncertainty in the standard MNIST domain. X-axis: False positive rate. Y-axis: True positive rate.

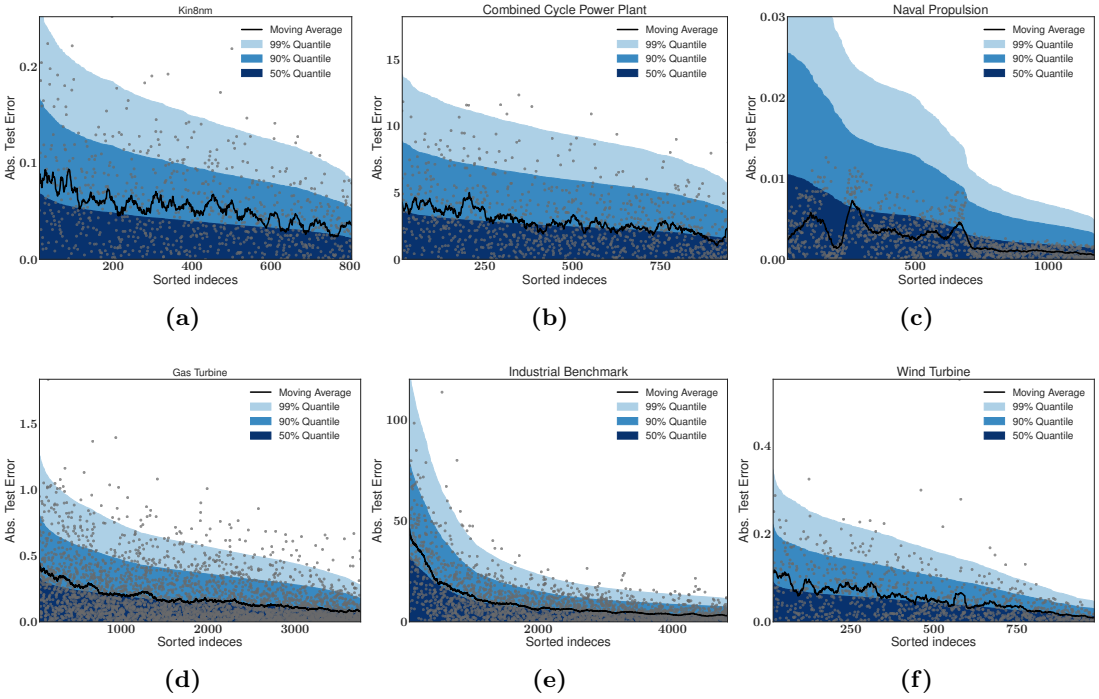


Figure 6.13: Uncertainty quality visualization (see Teye et al. (2018)) for six regression problems. Gray dots are absolute test error of a test data points, sorted by their estimated predictive uncertainty under the BNN+LV. The shaded blue areas show the 50% (dark blue), 90% (blue) and 99% (light blue) quantile of the predictive distribution, i.e (50/90/99)% of errors on test data are expected to fall under this curve. Black curve shows moving average of test error.

To start in Fig. 6.12 we show the receiver operating characteristic (ROC) curve between misclassification and predictive uncertainty in the standard MNIST setting.

The ROC curve measures the false positive rate against the true positive rate of a classifier. In our case the classifier is the amount of predictive uncertainty, whereas the target variable is the occurrence of a misclassification. We obtain an area under curve of 0.98 of the ROC curve; this means that in almost all cases a high predictive uncertainty is an accurate predictor for misclassification. For regression the work in Teye et al. (2018) highlights a novel way to link predictive uncertainty to test error visually. Here, we compute the (50/90/99)% quantiles of the predictive distributions for all test inputs $\mathbf{x}_\star$, this means under the model (50/90/99)% of errors are expected to be higher (lie outside) of the corresponding quantile. We can then compute the actual test errors and see how accurate the quantiles of the predictive distribution of the BNN+LV are. In Fig. 6.13 we show the results for 6 of the regression problems from Chapter 5. We see that in most cases the average test errors follows the 50% quantile very closely. In the Naval Propulsion example, however, the quantiles are much larger than the actual test errors. This indicates that the BNN+LV expects the test errors to be much larger than they actually are. In all plots the gray points, which indicate individual test errors, have considerable variance. A possible reason for this is randomness, which results in aleatoric uncertainty.

6.3 Sensitivity Analysis

In Section 3.3 we introduced a novel method for model inspection, called the sensitivity analysis of predictive uncertainty. The main idea is to examine the effect of each feature on the epistemic and aleatoric uncertainty in the prediction of BNN+LV. We discussed advantages of this method for monitoring and decision making and in this section we want to apply this method to a set of regression data sets and visualize the results.

6.3.1 Problem Description

In this experiment we are interested in the question of how input features impact the uncertainty in the predictions of the BNN+LV. In particular, how sensitive is each feature towards epistemic and aleatoric uncertainty? To address these question we use the gradient of the uncertainty components with respect to the input features. This can be obtained by the mechanism described in Section 3.3. We can quantify the sensitivity using Eq. (3.32) and Eq. (3.33).

We generate an artificial benchmark dataset and additionally use 8 standard regression benchmarks in varying domains and dataset sizes (see Section 4.1). For all experiments, we use a BNN with 2 hidden layer. We first perform model selection on the number of hidden units per layer from $\{20, 40, 60, 80\}$ on the available data.

For the artificial data set we use a function with 2 input features and heteroskedastic noise: $y = 7\sin(x_1) + 3|\cos(x_2/2)|\epsilon$ with $\epsilon \sim \mathcal{N}(0, 1)$. The first input variable x_1 is responsible for the shape of the function whereas the second variable x_2 determines the noise level. We sample 500 data points with $x_1 \sim \text{exponential}(\lambda = 0.5) - 4$ and $x_2 \sim \mathcal{U}(-4, 4)$.

6.3.2 Model Specification

We use the same set of hyper-parameters that we used in Chapter 5. For the sensitivity analysis we will sample $N_w = 200$ $w \sim q(\mathcal{W})$ and $N_z = 200$ samples from $z \sim \mathcal{N}(0, \gamma)$. All experiments were repeated 5 times and we report average results.

6.3.3 Experiments

Artificial Data Fig. 6.14a shows the sensitivities. The first variable x_1 is responsible for the epistemic uncertainty whereas x_2 is responsible for the aleatoric uncertainty. This result corresponds with the generative model for the data: x_2 affects the noise level, whereas x_1 determines the shape of the deterministic part of the function.

Standard Regression Benchmarks In Fig. 6.14 we show the results of all experiments. For some problems the aleatoric sensitivity is most prominent (Fig. 6.14f,6.14g), while in others we have predominately epistemic sensitivity (Fig. 6.14e,6.14h) and a mixture in others. This makes sense, because we have variable dataset sizes (e.g. Boston Housing with 506 data points and 13 features, compared to Protein Structure with 45730 points and 9 features) and also likely different heterogeneity in the datasets.

In the power-plant example feature 1 (temperature) and 2 (ambient pressure) are the main sources of aleatoric uncertainty of the target, the net hourly electrical energy output. The data in this problems originates from a combined cycle power plant consisting of gas and steam turbines. The provided features likely provide only limited information of the energy output, which is subject to complex combustion processes. We can expect that a change in temperature and pressure will influence this process in a complex way, which can explain the high sensitivities we see. The task in the naval-propulsion-plant example, shown in Fig. 6.14i, is to predict the compressor decay state coefficient, of a gas turbine operated on a naval vessel. Here we see that two features, the compressor inlet air temperature and air pressure have high epistemic uncertainty, but do not influence the overall sensitivity much. This makes sense, because we only have a single value of both features in the complete dataset. The model has learned no influence of this feature on the output (because it is constant), but any change from this constant will make the system highly uncertain.

6.3.4 Discussion

We have shown results of the sensitivity analysis for predictive epistemic and aleatoric uncertainty on standard regression and toy data sets. For some cases, such as the toy dataset we observe that the results match the generative model for the data. In others we could interpret the results based on summary statistics of the problem, such as data set size and dimensionality. In general, because the sensitivity analysis is an introspective method, there is no direct way to evaluate the quality of the results.

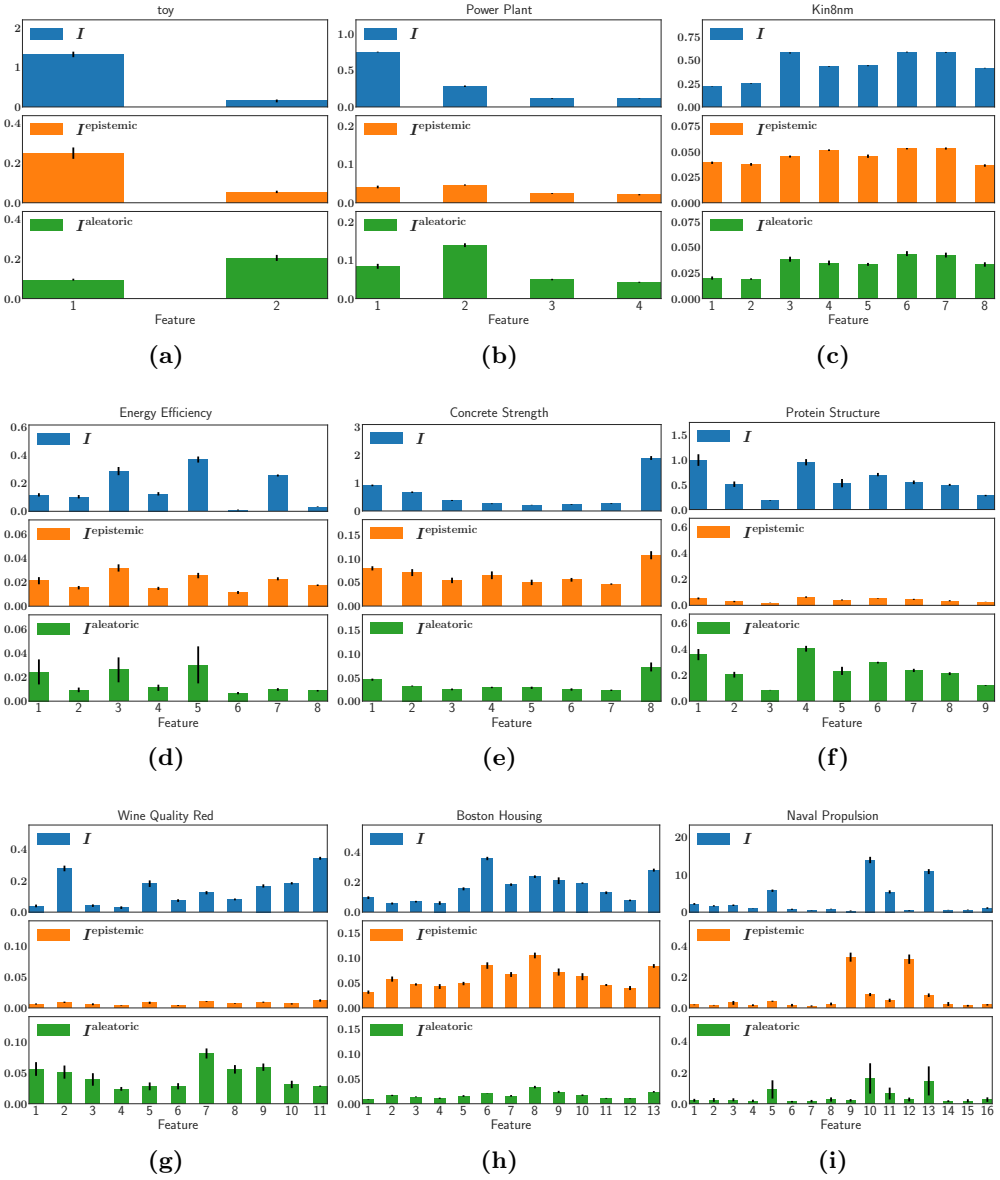


Figure 6.14: Sensitivity analysis for the predictive expectation and uncertainty on toy data (a) and UCI datasets (b)-(i). Top row shows sensitivities w.r.t. expectation (Eq. (3.31)). Middle and bottom row show sensitivities for epistemic and aleatoric uncertainty (Eq. (3.32) and Eq. (3.33)). Error bars indicate standard errors.

6.4 Active Learning

In this section we will do a set of active learning experiments on stochastic problems using BNN+LV. Active learning requires sequential decision making. We will show that if we

follow an information-theoretic approach, based on MacKay (1992), a decomposition of uncertainty in epistemic and aleatoric components will arise naturally.

6.4.1 Problem Description

Active learning is the problem of iteratively collecting data so that the final gains in predictive performance are as high as possible (Settles, 2010). In many practical problems we have a large amount of unlabeled observation data $\mathbf{X}$, whereas obtaining target variables $\mathbf{Y}$ is expensive. This is usually the case if obtaining targets requires manual labeling by human experts. The key goal in active learning is to choose those observations $\mathbf{x}_\star$ for which obtaining labels $\mathbf{y}_\star$ would improve the performance of the learning algorithm the most.

Let us define a training data set $\mathcal{D} = \{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N$ of already known observations and targets. From this data set we can learn the functional relationship between inputs and outputs. In active learning, the algorithm then sequentially decides for which $\mathbf{x}_\star$ the corresponding target $\mathbf{y}_\star$ should be shown and included in the training set. The target is unknown beforehand, the decision is therefore only based on the observation $\mathbf{x}_\star$.

One way to do such selection is to apply the information-theoretic approach for active learning, that was first described by MacKay (1992). The main idea is that using a Bayesian modeling approach we can reason about for which $\mathbf{x}_\star$ the entropy of the posterior $p(\boldsymbol{\theta}|\mathcal{W})$ will decrease the most, if we include $(\mathbf{x}_\star, \mathbf{y}_\star)$ in the training set. Because $\mathbf{y}_\star$ is unknown, we need to integrate over all possible values it can take. The reasoning behind this approach is that the data that maximize the reduction in entropy of $p(\boldsymbol{\theta}|\mathcal{W})$ provides the most knowledge about the data generating process, assuming the modeling assumption is correct.

There exist many approaches in literature that follow this approach (MacKay, 1992; Hernández-Lobato and Adams, 2015; Guo and Greiner, 2007). These approaches however assume a deterministic relationship between inputs and outputs, and consequently use deterministic methods (mostly GPs) as model class. Here, we consider the case of actively learning arbitrary non-linear functions in the presence of complex noise. We will first derive the information-theoretic approach using BNN+LV and then perform a set of active learning experiments.

6.4.2 Uncertainty Decomposition for Active Learning

We assume we have a BNN+LV that is parameterized by weights $\mathcal{W}$. This model is used to describe a batch of training data $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), \cdots, (\mathbf{x}_N, \mathbf{y}_N)\}$, resulting in a approximate posterior over the parameters $q(\mathcal{W}|\mathcal{D})$.

The expected reduction in posterior entropy for $\mathcal{W}$ that would be obtained when collecting the unknown target $\mathbf{y}_\star$ for the input $\mathbf{x}_\star$ is

$$\begin{aligned} &\mathrm{H}(\mathcal{W}|\mathcal{D}) - \mathbf{E}_{\mathbf{y}_\star|\mathbf{x}_\star,\mathcal{D}}\left[\mathrm{H}(\mathcal{W}|\mathcal{D}\cup\{\mathbf{x}_\star,\mathbf{y}_\star\})\right] && (6.1)\\ &\quad = I(\mathcal{W}, \mathbf{y}_\star) && (6.2)\\ &\quad = \mathrm{H}(\mathbf{y}_\star|\mathbf{x}_\star) - \mathbf{E}_{q(\mathcal{W}|\mathcal{D})}\left[\mathrm{H}(\mathbf{y}_\star|\mathcal{W}, \mathbf{x}_\star)\right] . && (6.3)\end{aligned}$$

In standard Bayesian modeling approaches such as GPs or BNNs the right hand side of Eq. (6.3) will be constant, because given a set of weights $\mathcal{W}$ (or f in a GP setting) the function is assumed to be deterministic, the remaining uncertainty $\mathrm{H}(\mathbf{y}_\star|\mathcal{W}, \mathbf{x}_\star)$ is constant with respect to $\mathbf{x}_\star$.

We see that the decomposition of uncertainty that we introduced in Section 3.2 has arisen naturally in this setting: the most informative points $\mathbf{x}_\star$ for which to collect $y_\star$ next is the one for which the epistemic uncertainty in the BNN+LV predictive distribution is the highest.

The epistemic uncertainty in Eq. (6.3) can be approximated using standard entropy estimators, e.g. nearest-neighbor methods Kozachenko and Leonenko (1987); Kraskov et al. (2004); Gao et al. (2016). For that, we repeatedly sample $\mathcal{W}$ and $z_\star$ and do forward passes through the BNN+LV to sample $\mathbf{y}_\star$. The resulting samples of $\mathbf{y}_\star$ can then be used to approximate the respective entropies for each $\mathbf{x}_\star$ using the nearest-neighbor approach:

$$\begin{aligned} &\mathrm{H}(\mathbf{y}_\star|\mathbf{x}_\star) - \mathbf{E}_{q(\mathcal{W})}\left[\mathrm{H}(\mathbf{y}_\star|\mathcal{W}, \mathbf{x}_\star)\right] \\ &\approx \hat{\mathrm{H}}(\mathbf{y}_\star^1, \ldots, \mathbf{y}_\star^L) - \frac{1}{M}\sum_{i=1}^{M}\left[\hat{\mathrm{H}}(\mathbf{y}_\star^{1,\mathcal{W}_i}, \ldots, \mathbf{y}_\star^{L,\mathcal{W}_i})\right] . \end{aligned} \tag{6.4}$$

where $\hat{\mathrm{H}}(\cdot)$ is a nearest-neighbor entropy estimate given an empirical sample of points, $\mathbf{y}_\star^1, \ldots, \mathbf{y}_\star^L$ are sampled from $p(\mathbf{y}_\star|\mathbf{x}_\star)$ according to Eq. (3.19), $\mathcal{W}_1, \ldots, \mathcal{W}_M \sim q(\mathcal{W})$ and $\mathbf{y}_\star^{1,\mathcal{W}_i}, \ldots, \mathbf{y}_\star^{L,\mathcal{W}_i} \sim p(y_\star|\mathcal{W}_i, \mathbf{x}_\star)$ for $i = 1, \ldots, M$.

There are alternative ways to estimate the entropy, e.g. with histograms or using kernel density estimation (KDE) Beirlant et al. (1997). We choose nearest neighbor methods because they tend to work well in low dimensions, are fast to compute (compared to KDE) and do not require much hyper-parameter tuning (compared to histograms). However, we note that for high-dimensional problems, estimating entropy is a difficult problem.

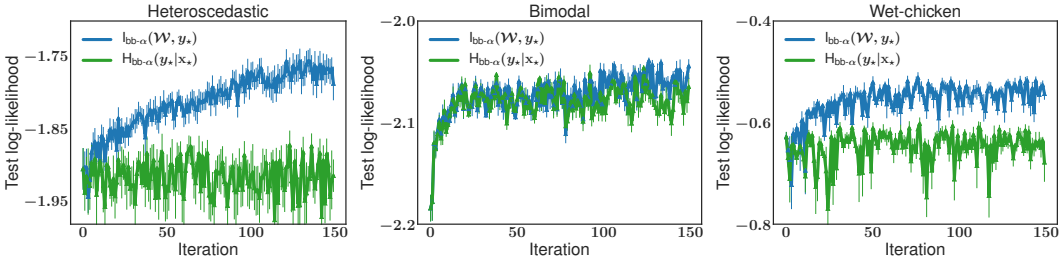
6.4.3 Model & Baseline Specification

We will use the two artificial benchmark problems and wet-chicken dynamics for our experiments. The settings are the same as in Section 6.1.3: that means in each of these three settings we start with a training data which is described in the respective paragraphs of Section 6.1.3. For the wet-chicken dynamics we predict y-dimension of successor state $\mathbf{s}(t+1)$ to make the evaluation comparable to that of Chapter 5.

The general set-up is as follows. We start with the available data described in Section 6.1.3 and train the methods on this data set. At each iteration, we select a batch of data points to label from a pool set which is sampled uniformly at random in input space. The selected data is then included in the training set and the log-likelihood is evaluated on a separate test set. The test set is of size 500 for the bimodal and heteroscedastic problem and 2500 for the wet-chicken problem. and is sampled uniformly in input space. After that, the models are re-trained (from scratch). This process is performed for 150 iterations and we repeat all experiments 5 times.

	Method		
Dataset	$\mathrm{I}_{\text{bb-}\alpha}(\mathcal{W}, y_\star)$	$\mathrm{H}_{\text{bb-}\alpha}(y_\star \mid \mathbf{x}_\star)$	GP
Heteroscedastic	**-1.79±0.03**	-1.92±0.03	-2.09±0.02
Bimodal	**-2.04±0.01**	-2.06±0.02	-2.86±0.01
Wet-chicken	**-0.54±0.02**	-0.68±0.05	-1.80±0.00

Table 6.1: Test log-likelihood in active learning experiments after 150 iterations.



(a) **(b)** **(c)**

Figure 6.15: Test log-likelihoods over active learning iterations for three benchmarks. Error bars indicate standard error over 5 repetitions.

The method we propose is to use a BNN+LV to describe the data and for the data collection process to use the reduction in entropy, given by Eq. (6.3) in each of the 150 iterations. We refer to this method as $\mathrm{I}_{\text{bb-}\alpha}(\mathcal{W}, y_\star)$ and compare it with two baselines. The first baseline also uses a BNN+LV to describe data, but does not perform a decomposition of predictive uncertainty, that is, this method uses the full uncertainty $\mathrm{H}(y_\star|\mathbf{x}_\star)$ instead of Eq. (6.3) for active learning. We call this method $\mathrm{H}_{\text{bb-}\alpha}(y_\star|\mathbf{x}_\star)$. The second baseline is given by a Gaussian process (GP) model which collects data according to $\mathrm{H}(y_\star|\mathbf{x}_\star)$ since in this case the uncertainty decomposition is not necessary because the GP model does not include latent variables. The GP model assumes Gaussian noise and is not able to capture complex stochastic patterns.

We train for 5000 epochs a BNN+LV with two-hidden layer and 20 hidden units per layer. We use Adam as optimizer with a learning rate of 0.001. For Gaussian processes(GPs) we use the standard RBF kernel using the python GPy implementation. For the entropy estimation we use a nearest-neighbor approach as explained in the main document with $k = 25$ and 500 samples of $q(W)$ and 500 samples of $p(z)$.

6.4.4 Experiments

Table 6.1 shows the test log-likelihoods obtained after 150 iterations. Overall, BNN+LV outperform GPs in terms of predictive performance. We also see significant gains of $\mathrm{I}_{\text{bb-}\alpha}(\mathcal{W}, y_\star)$ over $\mathrm{H}(y_\star|\mathbf{x}_\star)$ on the heteroscedastic and wet-chicken tasks, whereas their results are similar on the bimodal task. The reason for this is that, in the latter task, the epistemic and the total uncertainty have a similar behavior as shown in Figures 6.2d

and 6.2f. Finally, we note that heteroscedastic GPs Le et al. (2005) will likely perform similar to BNN+LV in the heteroscedastic task from Figure 6.2, but they will fail in the other two settings considered (Figures 6.2 and 6.3).

In Fig. 6.15 we show the test log-likelihood over the course of the active learning iterations for $\mathrm{I}_{\text{bb-}\alpha}(\mathcal{W}, y_\star)$ and $\mathrm{H}(y_\star|\mathbf{x}_\star)$. We see that when using the full uncertainty, without the decomposition, the predictive performance does barely improve over time in the heteroscedastic and wet-chicken problem. This is because, the data collection focuses on the stochastic areas in this problem and does not explore those areas where data is scarce.

6.4.5 Discussion

We have performed an empirical study in active learning as part of model inspection. Here, we have showed that using the information-theoretic approach for active learning a decomposition of uncertainty naturally occurs. We showed on a set of small-scale benchmarks, that exhibit complex stochasticity, that utilizing this decomposition leads to an improvement in the data collection process, compared to either using the full uncertainty or Gaussian process. These results also imply that for these set of problems the decomposition is meaningful, in the sense that data scarcity affects mostly the weight uncertainty $q(\mathcal{W})$ whereas stochasticity is modeled by the latent variable z.

7 Reinforcement Learning

Reinforcement learning (RL) is a subfield of machine learning. An agent is interacting with an environment by executing actions. Such interaction will lead to a state transition of the environment which returns a cost (or reward) signal back to the agent. The task of the agent is to identify the best possible actions that minimizes the costs, or maximizes the rewards, over a (possibly infinitely long) horizon of steps.

In the standard setting of time-discrete systems at each time step $t = 1, \ldots, T$ an agent receives from the environment E the current state observation $\mathbf{s}_t$ and is tasked to choose an action $\mathbf{a}_t$. The action is then applied to the environment transitioning into the next state $\mathbf{s}_{t+1}$. Upon this transition the agent receives a cost $c_t = c(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$. The Markov decision processes (MDP) is a classical mathematical framework in RL to formalize this situation. A MDP M is 5-tuple:

$$\mathrm{M} \coloneqq (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{C}) \ , \tag{7.1}$$

where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}$ is the transition function $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ indicating the probability of the successor state $\mathbf{s}_{t+1}$ when applying action $\mathbf{a}_t$ in state $\mathbf{s}_t$ and $\mathcal{C}$ is the cost function $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$. Note that $\mathcal{S}$ and $\mathcal{A}$ may be continuous or discrete. The Markov property of an MDP then states that:

$$p(\mathbf{s}_{t+1}|\mathbf{s}_0, \ldots, \mathbf{s}_t, \mathbf{a}_0, \ldots, \mathbf{a}_t) = p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) \ . \tag{7.2}$$

We note that in the framework we described here we assume that the true state $\mathbf{s}_t$ is visible to the agent. If this is not the case, we have a partially observable Markov decision process (POMDP), an extension of a MDP with conditional observations Ω based on observation probabilities $\mathcal{O}$.

In the standard RL case the goal of the agent is to interact with the environment via actions to minimize the cumulative future costs:

$$C = c_0 + \gamma c_1 + \gamma^2 c_2 + \ldots \ , \tag{7.3}$$

where $\gamma \in [0, 1]$ is called the discount factor.

The agents behavior can be described by a *policy*: $\pi : \mathcal{S} \to \mathcal{A}$ which can be deterministic or stochastic. The expected cumulative costs when executing policy π for a given starting state $\mathbf{s}_0$ gives rise to the *value function*:

$$\mathrm{V}^\pi(\mathbf{s}) = E[C|\mathbf{s}_0 = \mathbf{s}] = E[\sum_{t=0}^{T} \gamma^t c_t | \mathbf{s}_0 = \mathbf{s}] \ . \tag{7.4}$$

The value function measures the expected cost of executing the policy π in environment E starting at a particular starting position $\mathbf{s}_0$. The horizon T can be finite or or infinite,

in the latter case a discount factor $\gamma < 1$ is necessary such that the cumulative costs are finite. The optimal policy $\pi^\star$ is such that for all starting positions, the value function is minimal. We have now described the key ingredients of a reinforcement learning problem: an agent that interacts with an environment E, that we describe via a time-descrete MDP, a policy π, a cost signal c_t and a value function V (Sutton and Barto, 1998).

Numerous algorithms exist that try to solve the RL problem often designed for specific application scenarios and with different properties. In most cases we assume that the true MDP is unknown to the agent. In that case one fundamental property that distinguishes RL algorithms is whether to follow a model-based or model-free approach. In model-based RL we first build an approximation $\hat{E}$ of the environment and then utilize this for decision making: by having a model of the environment at hand, the agent can simulate the effect of its actions and thereby decide what the best possible action is. Model-free algorithms, by contrast, do not construct such approximation; instead these methods only focus on finding out what the best sequence of actions is without explicitly understanding the dynamics of the environment. To give an illustrative example: A cat does not need to know how a floor heating system works in order to find the most comfortable place to sit. Similarly, to drive a car one does not need to know the specifics of the physical processes in the car's engine. In this thesis we focus on model-based RL and we will describe the key properties in the next section.

We can further classify RL problems into interactive (or online) and batch RL problems (Lange et al., 2012; Ernst et al., 2005). In interactive RL, which is the classical case in literature, the agent can directly interact with the environment at hand. Over time the agent is expected to improve continuously until converging to the optimal actions in any given situation. Because of this, in an interactive setting, the agent needs to balance exploration and exploitation. Exploration is necessary to gain new information about the behavior of the environment while exploitation makes the agent behave in a way he considers most optimal at the current state of learning. Thus, one important quality metric of RL algorithms is the balance of these two behaviors. One example for this is ϵ-greedy exploration: with probability ϵ the agent chooses the action he considers most optimal at the current stage of learning and with probability $1 - \epsilon$ he executes a random action (Sutton and Barto, 1998). The value of ϵ is scheduled to increase over the course of training.

By contrast in batch RL the agent is not allowed to interact with the system during learning, it has only access to a data set, a batch, $\mathcal{D}$ of observations and applied action, and perhaps insight about the cost function c. Such a setting is often present in safety-critical systems, such as robotics or real-world industry systems, to avoid possible damage to the system caused by exploration behavior (Gordon, 1995; Ernst et al., 2005). While the problem of exploration and exploitation is not present in this setting, the knowledge the agent has about the system dynamics is limited to the available data: some areas in state space may be over- or underrepresented in $\mathcal{D}$. This situation calls for data-efficient methods i.e., how to gain the most insight from the limited data that is available. Furthermore, the agent ideally should incorporate the uncertainty over the system dynamics due to limited data into the learning process. Related, but slightly

different concepts are on-policy and off-policy methods. The former indicates that in order to evaluate and improve a current policy estimate, this policy needs to be applied to the environment. In contrast an off-policy method does not require that: we have a behavior policy interacting with the system, and our current estimate. While off-policy and batch RL are related, in most off-policy settings it is assumed that over time the current policy estimate replaces the behavior policy in an iterative process. The problem setting is still considered "online": interacting with a target system to gradually figure out what the best course of action is. This interactive process is not given in a batch RL setting.

Nowadays especially model-free algorithms see widespread use and achieve impressive practical results in interactive settings. Two famous examples are learning to play Atari games (Mnih et al., 2015) or beating human professional players in the game of Go (Silver et al., 2016). While model-free RL has shown impressive results, these approaches are not without downsides. One important weakness is *data-efficiency*. Even for simple RL benchmarks these methods require large amounts of data, and these data can only come from interacting with the environment. The effects of this are slow and hardware hungry training and the requirement of intensive interaction with the environment E. By contrast, humans can learn to play new games very quickly. While some of this learning speed can be attributed to having more rich prior knowledge (see e.g. Dubey et al. (2018)), another reason could be that we build internal models of the environment we are interacting with, such as a basic model of physics of the outer world. Especially in batch RL settings, where the available data is restricted by default, data-efficiency becomes an important criteria, consequently model-free methods appear to perform poorly (Hein et al., 2017b).

Model-based methods, by contrast, are expected to be more data-efficient. This is because by building a simulation based on the available data of the environment, the model is able to interpolate between data points and therefore extracts much more knowledge of the data. For instance, Deisenroth and Rasmussen (2011) uses a GP model for a real-world cart-pole task and show a data-efficiency an order of magnitude above model-free approaches. On the other hand, the main disadvantage of using model-based methods is the risk of *model-bias*. When we use the model to optimize the policy and if model and environment differ, the policy is vulnerable to "bias" by the model. A policy that exploits dynamics in the model that do not translate to the real environment will give suboptimal turnout and also may be dangerous in safety-critical applications. To quote from Deisenroth et al. (2013): "Dealing with inaccurate dynamics models is one of the biggest challenges in model-based RL since small errors in the model can lead to large errors in the policy". One way to address the issue of model-bias is to maintain uncertainty over the dynamics of the model. For instance, the work in Deisenroth and Rasmussen (2011) uses Gaussian processes as a model to express uncertainty over the dynamics function.

A further issue is that state transitions can be stochastic. In many real-world scenarios stochasticity may often arise due to some unobserved environmental feature that can affect the dynamics in complex ways (such as unmeasured gusts of wind on a boat). Popular methods used for approximating the environment transition functions include

Gaussian processes (Kuss and Rasmussen, 2004; Ko et al., 2007; Deisenroth and Rasmussen, 2011), fixed bases such as Laguerre functions (Wahlberg, 1991), and adaptive basis functions or neural networks (Draeger et al., 1995; Schaefer et al., 2007). All of these methods assume deterministic transition functions, perhaps with some addition of Gaussian observation noise. Thus, they are severely limited in the kinds of stochasticity—or transition noise—they can express.

Modeling stochastic effects in the data, while maintaining parameter uncertainty, is the central motivation for developing BNN+LV. In the previous chapters in regression (Chapter 5) and model inspection (Chapter 6) we also showed empirically that BNN+LV provide competitive predictive performance, has meaningful uncertainty over its parameters and can model stochasticity of the data. In this chapter we develop a novel model-based RL method that utilizes BNN+LV as models. We consider the batch RL scenario due to the relevance of this scenario for industrial RL problems. To study the effectiveness of these models we focus on tasks that possess stochasticity in the dynamics. In Section 7.2 we will study risk-sensitive scenarios and show that BNN+LV enable us to develop a novel risk-sensitive criterion using a decomposition of uncertainty to identify policies that balance expected cost, model-bias and noise aversion.

7.1 Model-based Reinforcement Learning

In model-based reinforcement learning, an agent uses its experience to first learn an approximation of the environment and then uses this model to reason about what action to take next. One central question in model-based RL therefore is: what is a good model? In Chapter 5 we evaluated the predictive performance of BNN+LV on a set of problems, including stochastic dynamic (toy- and real-world) systems. Here we saw that BNN+LV provide superior predictions in the presence of noise. Building on this insight in this chapter we wantto show how BNN+LV can be used as models for RL and evaluate how policies that are trained under a BNN+LV model behave in comparison to standard baseline models. We want to address the following questions:

1. How can we efficiently train policies when using BNN+LV as models for reinforcement learning?
2. Do BNN+LV enable learning accurate policies for stochastic benchmarks?
3. How does the performance of policies under BNN+LV dynamics compare to baseline models?

The methods and experiments presented in this section were previously published in Depeweg et al. (2017a).

7.1.1 Problem Description

In the beginning of this chapter we have introduced the Markov decision process (MDP) as the classical mathematical framework to describe an environment E. In the context of

model-based RL, we can describe an MDP alternatively using the concept of a discrete-time stochastic dynamic systems which is the framework used in classical control theory (Bertsekas, 2002). The general form of a (time-)discrete dynamic system is:

$$\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t, z_t) \,, \tag{7.5}$$

where f is an unknown function that takes as input the current observable state $\mathbf{s}_t$, the control signal (action) $\mathbf{a}_t$, and an unobserved/latent stochastic disturbance z_t from an unknown distribution. All z_i are assumed to be independent random variables. The equivalence of this formulation to the MDP follows from the fact that the next state $\mathbf{s}_{t+1}$ is only dependent on the current state and action (which is the Markov property), the state $\mathbf{s}_t$ is assumed to be observable, and the stochastic disturbance is independent w.r.t. t. Because of this Eq. (7.5) describes a probability distribution $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$. As with MDPs also deterministic systems can be described by Eq. (7.5), with $z_t = 0$.

The environment further has a cost function c:

$$c_t = c(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) \,, \tag{7.6}$$

that returns a scalar cost signal when doing a particular transition.

We consider a batch RL setting, where we are given a batch of state transitions $\mathcal{D} = \{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})\}$ formed by triples containing the current state $\mathbf{s}_t$, the action applied $\mathbf{a}_t$ and the next state $\mathbf{s}_{t+1}$ from such a discrete-time dynamic system. For example, $\mathcal{D}$ may be formed by measurements taken from an already-running dynamic system.

Model-based RL methods include two key parts (Deisenroth et al., 2013): In the first part we learn a model of the dynamics of the environment and in the second part we utilize this model to learn a policy. While in interactive RL these two steps are interwoven, in batch RL we have a simple two-step process: In the first step we want to estimate a model from the available data given by $\mathcal{D}$. This is a standard supervised learning task. From the batch of state transitions we infer the functional relationship between current state and action and successor state given by Eq. (7.5). Such a task can be a regression or classification problem depending if the state and action space is discrete or continuous. Different methods will make different assumptions about the form of the system dynamics. For instance, if a standard neural network is used f is assumed to be deterministic and consequently the stochastic disturbance z is not modeled. If state and action spaces are discrete, estimating the transition function, which reduces to a three dimensional tensor of probabilities $p_{s,a,s'}$ is straightforward and Bayesian modeling over discrete systems has also been studied in the last decades (Shapiro and Kleywegt, 2002; Nilim and El Ghaoui, 2005; Bagnell et al., 2001). In this work however we will focus on continuous state and action sequences.

We identify that Eq. (7.5) has the same form as Eq. (3.1) from Chapter 3. In fact, this class of function was the starting point in the derivation of BNN+LV. By applying the simplification we outlined in the aforementioned chapter we can therefore approximate Eq. (7.5) with:

$$\mathbf{s}_t = f_{\text{true}}(\mathbf{s}_{t-1}, \mathbf{a}_{t-1}, z_t; \mathcal{W}_{\text{true}}) + \boldsymbol{\epsilon}_t \,, \quad z_t \sim \mathcal{N}(0, \gamma) \,. \tag{7.7}$$

where the latent variables $z_t \sim \mathcal{N}(0, \gamma)$ and additive observation noise variable $\boldsymbol{\epsilon}_n \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$ account for the stochasticity in the dynamics. For this reason, BNN+LV seem to be promising candidates as models for time-discrete dynamics systems, because they approximate the general form given by Eq. (7.5). As mentioned earlier, existing model-based methods do not model the stochasticity to such a level of generality (see for instance Deisenroth et al. (2013), Eq. (3.1)).

The second step in model-based RL is to identify the best possible actions under a model M. Most methods usually make a single estimate of a model M (see for instance Schaefer et al. (2007)), whereas methods that utilize Bayesian modeling will instead provide a distribution over models $p(M)$ (Bagnell and Schneider, 2001; Deisenroth and Rasmussen, 2011; Gal et al., 2016).

Let us define a roll-out, or trajectory:

$$\tau = ((\mathbf{s}_0, \mathbf{a}_0), \ldots, (\mathbf{s}_T, \mathbf{a}_T)) \ , \tag{7.8}$$

over a horizon of T. A roll-out is a sequence of steps which result from interacting with a model via actions $\mathbf{a}$ in a recursive way: when in $\mathbf{s}_t$, we apply action $\mathbf{a}_t$ to the model, which leads to a state transition to $\mathbf{s}_{t+1}$. The starting state $\mathbf{s}_0$ may be fixed or could be any state in the state space $\mathcal{S}$ depending on the problem. We define $\tau|M$ as a trajectory under a model M and $p(\tau|M)$ as the (typically intractable) distribution over all possible trajectories under M. Finally, let $p(\tau) = \int p(\tau|M)p(M)dM$ be the marginal distribution over all trajectories.

One approach to identify the optimal actions is model predictive control (MPC). At each point in time MPC optimizes a sequence of actions $\mathbf{a}_t, \ldots, \mathbf{a}_T$ over a horizon T with respect to

$$\mathbf{a}_t, \ldots, \mathbf{a}_T = \underset{\mathbf{a}_t, \ldots, \mathbf{a}_T}{\arg\min} \, \mathbf{E}[\sum_{t'=t}^{t+T} c_{t'}] \ , \tag{7.9}$$

where the expectation is over all trajectories τ that can be generated under the model (or distribution over models, see for instance Chua et al. (2018)) using the action sequence $\mathbf{a}_t, \ldots, \mathbf{a}_T$ and starting in the current state $\mathbf{s}_t$. After finding the best sequence of actions the MPC controller uses the first and executes it on the ground-truth environment. After selecting an action $\mathbf{a}_t$ the horizon T gets shifted on step in the future and the optimization is repeated, a process called *receding horizon* control.

Because the minimization process in Eq. (7.9) is typically not solvable analytically, there exist numerous approximation techniques. The most simplistic method would be to perform a random search in action space. More sophisticated approaches exist, for instance by assuming linearity in the effect of the actions (Todorov and Li, 2005), or using more efficient search strategies such as particle swarm optimization (PSO) (Hein et al., 2018) . MPC, however, has drawbacks. One is that it can be very slow to use in practice, or relies on strong approximation about the nature of the dynamics (such as the aforementioned linearity). Furthermore, MPC does not estimate a policy π, but needs to repeat the planning process at every point in time. This can also make it slow to apply in practice and maybe even be prohibitive in industrial RL context.

Instead, in batch RL it can be desirable to have a policy π, that can be deployed on the target system, without any learning or optimization taking place afterwards. One method that enables is model-based *policy search* and which will be the topic of the next section.

7.1.2 Model-based Policy Search with BNN+LV

In model-based policy search we try to find a parametric policy $\mathbf{a}_t = \pi(\mathbf{s}_t; \mathcal{W}_\pi)$, which for instance could be modeled by a neural network. In this work we model policies as deterministic functions. While stochastic policies are often used to guide exploration, here we consider the batch RL scenario where exploration is not allowed.

We define $p(\tau|\mathcal{W}_\pi, M)$ as the distribution of all possible trajectories under model M and with a policy parameterized by $\mathcal{W}_\pi$. The objective function to minimize in model-based policy search is given by:

$$J(\mathcal{W}_\pi) = \mathbf{E}_{p(M)}\mathbf{E}_{p(\tau|\mathcal{W}_\pi,M)}\left[\sum_{t=1}^{T} c_t\right] , \tag{7.10}$$

which means we want to find the parameters $\mathcal{W}_\pi$ of policy π, that minimizes the expected cost over all possible models and trajectories for a horizon of size T.

The two challenges in policy search are estimating the objective in Eq. (7.10) and performing the minimization with respect to the policy parameters $\mathcal{W}_\pi$ itself. For certain model classes, such as Gaussian processes, the objective can be estimated by moment matching: because the predictive distribution of a GP is Gaussian, uncertainty can be propagated in every step and by that a distribution over trajectories can be obtained (Deisenroth and Rasmussen, 2011). An alternative method is to sample trajectories: if we are able to sample from the distribution over models, that is $p(M)$, we can generate random roll-outs $\tau|\mathcal{W}_\pi, M$ of interaction between model and current policy (see for instance Gal et al. (2016)). In the case of high variance the work in Ng and Jordan (2000) propose an efficient sampling process by constraining the random number generation.

For minimization we want to find the optimal policy:

$$\mathcal{W}_{\pi_\star} = \arg\min_{\mathcal{W}_\pi} J(\mathcal{W}_\pi) . \tag{7.11}$$

Given we can estimate Eq. (7.10), in principle any gradient-free method can be applied, such as random search in parameter space of $\mathcal{W}_\pi$, or more sophisticated methods such as PSO (Hein et al., 2017b). However, faster convergence and more scalable applicability can be expected by utilizing gradient information for policy updates. This can be achieved in two ways. If every component in J, that is the model, the policy and cost function is differentiable, we can obtain an analytical gradient. This is because the cost c_t is connected to the current state $\mathbf{s}_t$ and action $\mathbf{a}_t$ which themselves are connected to the previous state $\mathbf{s}_{t-1}$ which then enables calculating gradients in the same way as backpropagation in neural networks (see Section 2.4.1) and for which automatic differentiation tools can be used.

If gradients cannot be computed analytically, for instance when the cost signal is sparse, a gradient has to be estimated. This can be done using finite difference methods or policy gradient techniques (Deisenroth et al., 2013). In this thesis, we only consider problems where an analytical gradient is available. For completeness at the end of this section we provide a short digression into model-based policy gradient. This approach is a blueprint that would enable the methodologies and ideas we develop in this work to be applicable to situations where for instance the reward is sparse and therefore not differentiable.

We will now describe the minimization process of the objective in Eq. (7.10) by gradient descent. As model class we will use BNN+LV. In the next section describe the policy search process using alternative methods, that will also serve as baselines for the experiments that follow. As mentioned earlier we assume that given a batch of data $\mathcal{D}$ of state transitions we have trained a BNN+LV. The result of this is a distribution over models, given by the distribution $q(\mathcal{W})$. The transition model in Eq. (7.7) specifies a probability distribution $p(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{a}_{t-1})$ that we approximate by:

$$p(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{a}_{t-1}) \approx \int \mathcal{N}(\mathbf{s}_t|f(\mathbf{s}_{t-1}, \mathbf{a}_{t-1}, z_t; \mathcal{W}), \mathbf{\Sigma}) q(\mathcal{W}) \mathcal{N}(z_t|0, \gamma)\, d\mathcal{W}\, dz_t\,, \tag{7.12}$$

where the feature vectors in our BNN are now $\mathbf{s}_{t-1}$ and $\mathbf{a}_{t-1}$ and the targets are given by $\mathbf{s}_t$. In this expression, the integration with respect to $\mathcal{W}$ accounts for stochasticity arising from lack of knowledge of the model parameters, while the integration with respect to z_t accounts for stochasticity arising from unobserved processes that cannot be modeled. In practice, these integrals are approximated by an average over samples of $z_t \sim \mathcal{N}(0, \gamma)$ and $\mathcal{W} \sim q$.

In the second part of model-based policy search, we optimize the parameters $\mathcal{W}_\pi$ of a policy that minimizes the sum of expected cost over a finite horizon T with respect to the belief $q(\mathcal{W})$. This expected cost is obtained by averaging over multiple virtual roll-outs. For each roll-out we sample $\mathcal{W}_i \sim q$ and then simulate state trajectories using the model $\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t, z_t; \mathcal{W}_i) + \boldsymbol{\epsilon}_{t+1}$ with policy $\mathbf{a}_t = \pi(\mathbf{s}_t; \mathcal{W}_\pi)$, latent variables $z_t \sim \mathcal{N}(0, \gamma)$ and additive noise $\boldsymbol{\epsilon}_{t+1} \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma})$. This procedure allows us to obtain estimates of the policy's expected cost for any particular cost function. If model, policy and cost function are differentiable, we are then able to tune $\mathcal{W}_\pi$ by stochastic gradient descent over the roll-out average.

We approximate the exact costs given by Eq. (7.10) using Monte Carlo, by drawing K samples of models $\{\mathcal{W}^1, \ldots, \mathcal{W}^K\}$ and generating a trajectory:

$$J(\mathcal{W}_\pi) = \mathbf{E}_{p(M)} \mathbf{E}_{p(\tau|\mathcal{W}_\pi, M)} \left[\sum_{t=1}^{T} c_t \right] \tag{7.13}$$

$$= \mathbf{E}_{q(\mathcal{W})} \mathbf{E}_{p(\tau|\mathcal{W}_\pi, \mathcal{W})} \left[\sum_{t=1}^{T} c_t \right] \tag{7.14}$$

$$\approx \frac{1}{K} \sum_{k=1}^{K} \left[\sum_{t=1}^{T} c(\mathbf{s}_t^{\mathcal{W}^k, \{z_1^k, \ldots, z_t^k\}, \{\boldsymbol{\epsilon}_1^k, \ldots, \boldsymbol{\epsilon}_t^k\}, \mathcal{W}_\pi}) \right]\,, \tag{7.15}$$

Algorithm 1 Model-based policy search using BNN+LV.

Input: $\mathcal{D} = \{\mathbf{s}_n, a_n, \mathbf{\Delta}_n\}$ for $n \in 1..N$
Fit $q(\mathcal{W})$ *and* $\mathbf{\Sigma}$ *by optimizing* (3.15).
function UNFOLD($\mathbf{s}_0$)
  sample$\{\mathcal{W}^1, .., \mathcal{W}^K\}$ *from* $q(\mathcal{W})$
  $C \leftarrow 0$
  for $k = 1 : K$ **do**
    for $t = 0 : T$ **do**
      $z_{t+1}^k \sim \mathcal{N}(0, \gamma)$
      $\mathbf{\Delta}_t \leftarrow f(\mathbf{s}_t, \pi(\mathbf{s}_t; \mathcal{W}_\pi), z_{t+1}^k; \mathcal{W}^k)$
      $\boldsymbol{\epsilon}_{t+1}^k \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma})$
      $\mathbf{s}_{t+1} \leftarrow \mathbf{s}_t + \mathbf{\Delta}_t + \boldsymbol{\epsilon}_{t+1}^k$
      $C \leftarrow C + c(\mathbf{s}_{t+1})$
  return C/K
Fit $\mathcal{W}_\pi$ *by optimizing* $\frac{1}{N}\sum_{n=1}^{N}$ UNFOLD($\mathbf{s}_n$)

Figure 7.1: Pseudocode of the algorithm for model-based policy search using BNN+LV.

where $\mathbf{s}_t^{\mathcal{W},\{z_1,\ldots,z_t\},\{\boldsymbol{\epsilon}_1,\ldots,\boldsymbol{\epsilon}_t\},\mathcal{W}_\pi}$ is the state that is obtained at time t in a roll-out generated by using a policy with parameters $\mathcal{W}_\pi$, a transition function parameterized by $\mathcal{W}$ and latent variables $z_1, \ldots, z_t$, with additive noise values $\boldsymbol{\epsilon}_1, \ldots, \boldsymbol{\epsilon}_t$. In the last line we have approximated the integration with respect to $\mathcal{W}, z_1, \ldots, z_T$, $\boldsymbol{\epsilon}_1, \ldots, \boldsymbol{\epsilon}_T$ and $\mathbf{s}_0$ by averaging over K samples of these variables. To sample $\mathbf{s}_0$, we draw this variable uniformly from the available transitions $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$. The gradient can then be obtained by utilizing the Monte Carlo approximation of Eq. (7.15):

$$\nabla_{\mathcal{W}_\pi} J(\mathcal{W}_\pi) \approx \nabla_{\mathcal{W}_\pi} \frac{1}{K} \sum_{k=1}^{K} \left[\sum_{t=1}^{T} c(\mathbf{s}_t^{\mathcal{W}^k,\{z_1^k,\ldots,z_t^k\},\{\boldsymbol{\epsilon}_1^k,\ldots,\boldsymbol{\epsilon}_t^k\},\mathcal{W}_\pi}) \right] . \tag{7.16}$$

Algorithm 1 shows the full computation scheme of model-based policy search using BNN+LV. The gradients can be obtained using automatic differentiation tools such as Theano (Bergstra et al., 2010). Note that Algorithm 1 uses the BNNs to make predictions for the change in the state $\Delta_t = \mathbf{s}_{t+1} - \mathbf{s}_t$ instead of for the next state $\mathbf{s}_{t+1}$ since this approach often performs better in practice (Deisenroth and Rasmussen, 2011).

Digression: Model-based Policy Gradient Policy gradient is one major subfield of reinforcement learning. These techniques are on-policy: to estimate the gradient of J w.r.t. the current policy, parameterized by $\mathcal{W}_\pi$, this policy needs to be applied to the system. However, here we consider the case of applying the policy inside a model M, which makes them applicable to a batch RL scenario. Due to the Markov property of

Eq. (7.5) the distribution over trajectories, that is $p(\tau|\mathcal{W}_\pi, M)$ can be expressed as:

$$p(\tau|\mathcal{W}_\pi, M) = p(\mathbf{s}_0)\prod_{t=0}^{T} p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t = \pi(\mathbf{s}_t; \mathcal{W}_\pi); M) \ . \tag{7.17}$$

The log-derivate, or REINFORCE, trick (Williams, 1992) allows us to rewrite the gradient of this distribution w.r.t. the policy parameters as:

$$\nabla_{\mathcal{W}_\pi} p(\tau|\mathcal{W}_\pi, M) = p(\tau|\mathcal{W}_\pi, M)\nabla_{\mathcal{W}_\pi} \log p(\tau|\mathcal{W}_\pi, M) \tag{7.18}$$

The REINFORCE trick allows us to simplify the derivation of the gradient of the objective J such that:

$$\nabla_{\mathcal{W}_\pi} J(\mathcal{W}_\pi) = \nabla_{\mathcal{W}_\pi} \mathbf{E}_M \mathbf{E}_{p(\tau|\mathcal{W}_\pi, M)} \left[\sum_{t=1}^{T} c(\mathbf{s}_t, \mathbf{a}_t)\right] \ , \tag{7.19}$$

$$= \mathbf{E}_{p(M)} \nabla_{\mathcal{W}_\pi} \int p(\tau|\mathcal{W}_\pi, M) \sum_{t=1}^{T} c(\mathbf{s}_t, \mathbf{a}_t) d\tau \ , \tag{7.20}$$

$$= \mathbf{E}_{p(M)} \int p(\tau|\mathcal{W}_\pi, M) \nabla_{\mathcal{W}_\pi} \log p(\tau|\mathcal{W}_\pi, M) \sum_{t=1}^{T} c(\mathbf{s}_t, \mathbf{a}_t) d\tau \ , \tag{7.21}$$

$$= \mathbf{E}_{p(M)} \mathbf{E}_{p(\tau|\mathcal{W}_\pi, M)} [\nabla_{\mathcal{W}_\pi} \log p(\tau|\mathcal{W}_\pi, M) \sum_{t=1}^{T} c(\mathbf{s}_t, \mathbf{a}_t)] \ , \tag{7.22}$$

$$= \mathbf{E}_{p(M)} \mathbf{E}_{p(\tau|\mathcal{W}_\pi, M)} \sum_{t=1}^{T} \nabla_{\mathcal{W}_\pi} \log(p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t = \pi(\mathbf{s}_t; \mathcal{W}_\pi), M)) c(\mathbf{s}_t, \mathbf{a}_t) \ , \tag{7.23}$$

$$= \mathbf{E}_{p(M)} \mathbf{E}_{p(\tau|\mathcal{W}_\pi, M)} \sum_{t=1}^{T} \nabla_{\mathbf{a}_t} \log(p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, M)) \nabla_{\mathcal{W}_\pi} \pi(\mathbf{s}_t; \mathcal{W}_\pi) c(\mathbf{s}_t, \mathbf{a}_t) \ . \tag{7.24}$$

This result enables estimating the gradient of the objective given by Eq. (7.10), even in the case that the cost function is not differentiable: the outer two expectations $\mathbf{E}_{p(M)}$ and $\mathbf{E}_{p(\tau|\mathcal{W}_\pi, M)}$ can be estimated by sampling trajectories and the the gradients are only w.r.t. to the model and policy. Note that because we require a deterministic policy a gradient w.r.t. the model is needed, whereas for stochastic policies this is not the required (Peters and Schaal, 2006).

7.1.3 Model & Baseline Specification

We will use the same set of parametric models that we used in the regression experiments in Chapter 5. The policy search procedure will differ slightly for different methods and here we want to describe this for each baseline.

1. Standard neural networks (MLP): Similar as in Chapter 5 we train the neural network using early stopping. We model the dynamic system in Eq. (7.5) using:

$$\mathbf{s}_t = f_{\text{true}}(\mathbf{s}_{t-1}, \mathbf{a}_{t-1}, \mathcal{W}_{\text{true}}) + \boldsymbol{\epsilon}_t \ , \tag{7.25}$$

where we approximate $\mathcal{W}_{\text{true}}$ using the set of weights with the lowest validation error. The observation noise $\boldsymbol{\epsilon}_t$ is constant additive Gaussian noise whose variance is given by the minimal squared validation error. The approximation to the objective function $J(\mathcal{W}_\pi)$ then is:

$$J(\mathcal{W}_\pi) \approx \frac{1}{K} \sum_{k=1}^{K} \left[\sum_{t=1}^{T} c(\mathbf{s}_t^{\mathcal{W}^{\text{val}}, \{\boldsymbol{\epsilon}_1^k, \ldots, \boldsymbol{\epsilon}_t^k\}, \mathcal{W}_\pi}) \right] \tag{7.26}$$

2. MLP Ensemble (MLP Ens.): We use an ensemble of 25 neural networks by running the training process of the MLP 25 times in parallel. The model assumption is the same as in the MLP, in addition we have an empirical distribution $p(M) = \{\mathcal{W}_1, \ldots, \mathcal{W}_K$ with corresponding constant output variances $\sigma_1^2, \ldots, \sigma_K^2$. The objective function then is

$$J(\mathcal{W}_\pi) \approx \frac{1}{K} \sum_{k=1}^{K} \left[\sum_{t=1}^{T} c(\mathbf{s}_t^{\mathcal{W}^{\text{val},k}, \{\boldsymbol{\epsilon}_1^k, \ldots, \boldsymbol{\epsilon}_t^k\}, \mathcal{W}_\pi}) \right] \quad . \tag{7.27}$$

3. Gaussian processes (GP): These have recently been used for policy search under the name of PILCO (Deisenroth and Rasmussen, 2011). For each dimension of the target variables, we fit a different sparse GP using the FITC approximation (Snelson and Ghahramani, 2005). In particular, each sparse GP is trained using 150 inducing inputs by using the method stochastic expectation propagation (Bui et al., 2016). After this training process we approximate the sparse GP by using a feature expansion with random basis functions (see supplementary material of Hernández-Lobato et al. (2014)). This allows us to draw samples from the GP posterior distribution over functions, enabling the use of Algorithm 1 for policy training. Note that PILCO will instead moment-match at every roll-out step as it works by propagating Gaussian distributions. However, in our experiments we obtained better performance by avoiding the moment matching step with the aforementioned approximation based on random basis functions.

4. Bayesian neural networks (BNN): The BNN is trained with $\alpha = 1$ but does not have a latent variable model.

As in Chapter 5 all methods based on neural networks share the same topology and hyper-parameters. The specific hyper-parameters differ for each experiment due to scalability and specifics of each benchmark. We will therefore describe them in the subsequent paragraphs.

7.1.4 Experiments

We now evaluate the performance of our algorithm for policy search in three different benchmark problems. These problems are chosen based on two reasons. First, they contain complex stochastic dynamics and second, they represent real-world applications

	Baselines				BNN+LV		
Benchmark	**MLP**	**MLP Ens.**	**GP**	**BNN**	**VB**	$\boldsymbol{\alpha = 0.5}$	$\boldsymbol{\alpha = 1.0}$
Wet-chicken	-2.71±0.09	-2.64±0.06	-3.05±0.06	-2.88±0.06	-2.67±0.10	**-2.37±0.01**	-2.42±0.01
IB	-183.5±4.1	-184.5±7.4	-285.2±20.5	-177.0±3.7	-180.2±0.6	-174.2±1.1	**-171.1±2.1**
Gas	-0.65±0.14	-0.42±0.01	-0.64±0.18	-0.44±0.02	-0.45±0.02	**-0.41±0.03**	-0.55±0.08

Table 7.1: Policy performances over different benchmarks. Printed are average values over 5 runs with respective standard errors.

common in industrial settings. A Theano implementation of algorithm 1 is available online[1].

The first experiments we conduct is on the wet-chicken benchmark. We choose this because the benchmark is very simple but exhibits complex stochastic patterns, making it ideal to empirically evaluate whether the capability to model stochastic patterns in BNN+LV lead to better performing policies than using baselines. The second experiments uses the industrial benchmark, a system with stochasticity and high-dimensionality, which was designed to mimic some properties observed in real-world industrial systems. Lastly, we use data from a real gas turbine and design a task with partial observability.

Wet-Chicken

The neural network models are set to 2 hidden layers and 20 hidden units per layer. We use 2500 random state transitions for training. We found that assuming no observation noise by setting Γ to a constant of 10^{-5} helped the models converge to lower energy values. For policy training we use a horizon of size $T = 5$ and optimize the policy network for 100 epochs, averaging over $K = 20$ samples in each gradient update, with mini-batches of size 10 and learning rate set to 10^{-5}.

The predictive distributions of different models for y_{t+1} are shown in Figure 7.2 for specific choices of (x_t, y_t) and $(a_{x,t}, a_{y,t})$. These plots show that BNNs+LV with $\alpha = 0.5$ are very close to the ground-truth. While it is expected that Gaussian processes fail to model multi-modalities in Figure 7.2c, the FTIC approximation allows them to model the heteroscedasticity to an extent. VB captures the stochastic patterns on a global level, but often under or over-estimates the true probability density in specific regions.

After fitting the models, we train policies using Algorithm 1 with a horizon of size $T = 5$. Table 7.1 shows the average reward obtained by each method. BNN+LV with $\alpha = 0.5$ or $\alpha = 1$ produce the best policies, whereas VB seems to lack robustness and has much larger empirical variance across experiment repetitions.

Figure 7.3 shows three example policies, π_{VB} ,$\pi_{\alpha=0.5}$ and π_{GP} (Figure 7.3a,7.3b and 7.3c, respectively). The policies obtained by BNNs with random inputs (VB and $\alpha = 0.5$) show a richer selection of actions. The biggest differences are in the middle-right regions of the plots, where the drift towards the waterfall is large and the bi-modal transition for y (missed by the GP) is more important.

[1] `https://github.com/siemens/policy_search_bb-alpha`

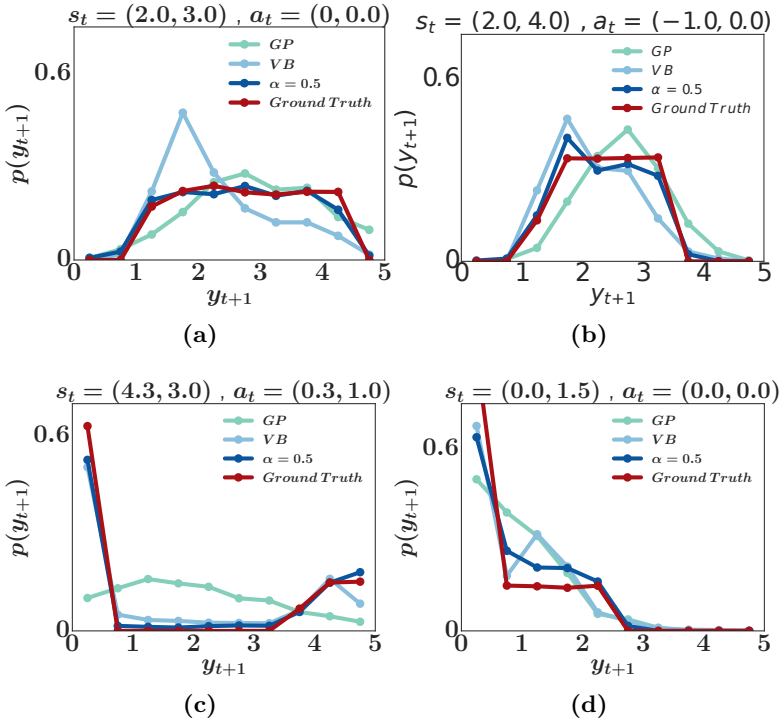


Figure 7.2: Predictive distribution of y_t given by different methods in four different scenarios. Ground truth (red) is obtained by sampling from the real dynamics.

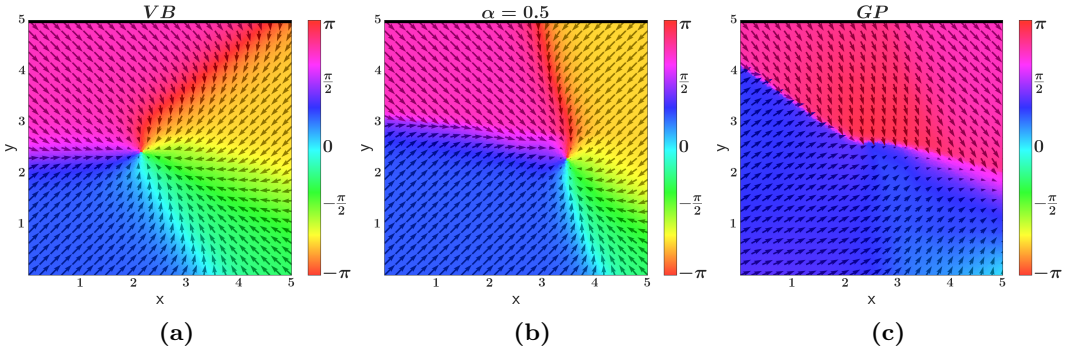


Figure 7.3: Visualization of three policies in state space. Waterfall is indicated by top black bar. Left: policy π_{VB} obtained with a BNN trained with VB. Avg. reward is -2.53. Middle: policy $\pi_{\alpha=0.5}$ obtained with a BNN trained with $\alpha = 0.5$. Avg. reward is -2.31. Right: policy π_{GP} obtained by using a Gaussian process model. Avg. reward is -2.94. Color and arrow indicate direction of paddling of policy when in state $\mathbf{s}_t$, arrow length indicates action magnitude. Best viewed in color.

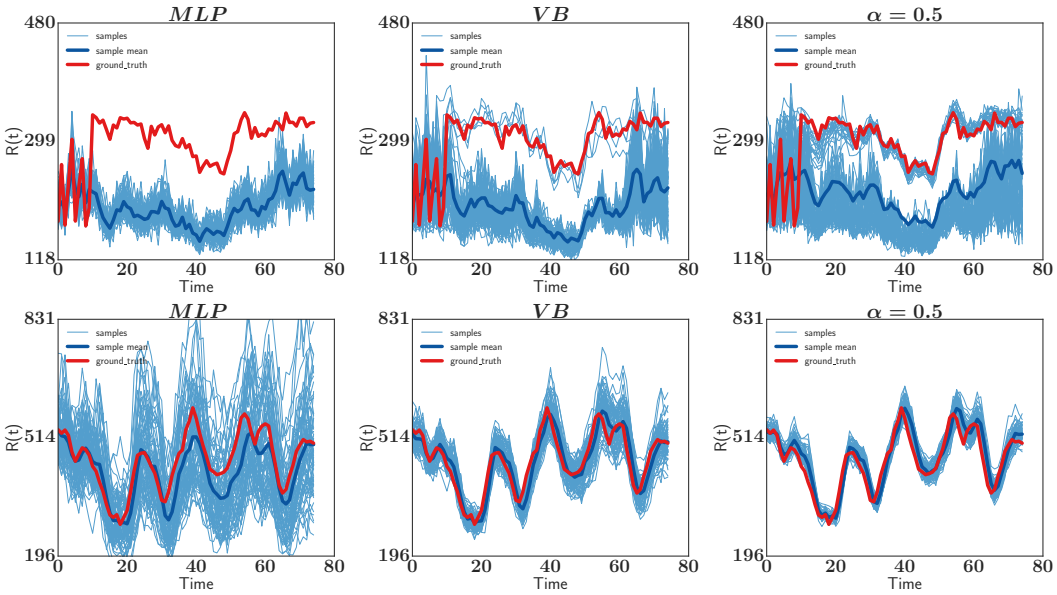


Figure 7.4: Roll-outs of algorithm 1 for two starting states $\mathbf{s}_0$ (top/bottom) using different types of BNNs (left to right) with $K = 75$ samples for $T = 75$ steps. Action sequence $A_0, \cdots, A_{T=75}$ given by data set for each $\mathbf{s}_0$. From left to right: model trained using VB,$\alpha = 0.5$ and $\alpha = 1.0$ respectively. Red: trajectory observed in data set, blue: sample average, light blue: individual samples.

Industrial applications

We now present results on two industrial cases. First, we focus on data generated by a real gas turbine and second, we consider a recently introduced simulator called the industrial benchmark, with code publicly available (Hein et al., 2017a).

Gas turbine data For the experiment with gas turbine data we simulate a task with partial observability. As we discussed in Section 4.3.3 we have 40,000 observations of a 30-dimensional time-series of sensor recordings from a real gas turbine available. We are also given a cost function that evaluates the performance of the current state of the turbine. The features in the time-series are grouped into three different sets: a set of environmental variables E_t (e.g. temperature and measurements from sensors in the turbine) that cannot be influenced by the agent, a set of variables relevant for the cost function N_t (e.g. the turbines current emission) and a set of steering variables A_t that can be manipulated to control the turbine.

We first train a world model as a reflection of the real turbine dynamics. To that end we define the world model's transitions for N_t to have the functional form $N_t = f(E_{t-5}, .., E_t, A_{t-5}, ..A_t)$. The world model assumes constant transitions for the environmental variables: $E_{t+1} = E_t$. To make fair comparisons, our world model is given by a non-Bayesian neural network with deterministic weights and with additive Gaussian output noise.

We then use the world model to generate an artificial batch of data for training the different methods. The inputs in this batch are still the same as in the original turbine data, but the outputs are now sampled from the world model. After generating the artificial data, we only keep a small subset of the original inputs to the world model. The aim of this experiment is to learn policies that are robust to noise in the dynamics. This noise would originate from latent factors that cannot be controlled, such as the missing features that were originally used to generate the outputs by the world model but which are no longer available. After training the models for the dynamics, we use algorithm 1 for policy optimization. The resulting policies are then finally evaluated in the world model.

All neural network architectures have two hidden layers with 50 hidden units each. For policy training and world-model evaluation we perform a roll-out with horizon $T = 20$. For learning the policy we use mini-batches of size 10 and draw $K = 10$ samples from q.

In Table 5.1 from Chapter 5 we already evaluated the predictive performance of different methods on these data. Here we saw that BNN+LV provide the highest test log-likelihood whereas ensembles of MLPs performed best in terms of test RMSE. The results of the policy given by Table 7.1 reflect that, using these two methods as models give rise to the best performing policies.

Industrial benchmark We provide a description of this benchmark in Section 4.3.2. For each setpoint $S \in \{10, 20, \cdots, 100\}$ we generate 7 trajectories of length 1000 using random exploration. This batch with $70,000$ state transitions forms the training set. We use $30,000$ state transitions, consisting of 3 trajectories for each setpoint, as test set.

For data preprocessing, in addition to the standard normalization process, we apply a log transformation to the reward variable. Because the reward is bounded in the interval $[0, R_{max}]$, we use a logit transformation to map this interval into the real line. We define the functional form for the dynamics as $R_t = f(A_{t-15}, \cdots, A_t, R_{t-15}, \cdots, R_{t-1})$. This time-embedding is done to reduce the effect of temporal dependencies due to the partial observability of this benchmark. By this we hope to transform this benchmark which is a POMDP to an equivalent MDP, which is licensed by Takens' theorem (Takens, 1981). For more details on this we refer to Bush and Pineau (2009).

For the neural network models we use two hidden layers with 75 hidden units.We use a horizon of $T = 75$, training for 500 epochs with batches of size 50 and $K = 25$ samples for each roll-out.

Each row in Figure 7.4 visualizes long term predictions of the MLP and BNNs trained with VB and $\alpha = 0.5$ in two specific cases. In the top row we see while all three methods produce wrong predictions in expectation (compare dark blue curve to red curve). However, BNNs trained with VB and with $\alpha = 0.5$ exhibit a bi-modal distribution of predicted trajectories, with one mode following the ground-truth very closely. By contrast, the MLP misses the upper mode completely. The bottom row shows that the VB and $\alpha = 0.5$ also produce more tight confident bands in other settings.

Next, we learn policies using the trained models. Here we use a relatively long horizon of $T = 75$ steps. Table 7.1 shows average rewards obtained when applying the policies

to the real dynamics. Because both benchmark and models have an autoregressive component, we do an initial warm-up phase using random exploration before we apply the policies to the system and start to measure rewards.

We observe that GPs perform very poorly in this benchmark. We believe the reason for this is the long search horizon, which makes the uncertainties in the predictive distributions of the GPs become very large. Tighter confidence bands, as illustrated in Figure 7.4 seem to be key for learning good policies. Overall, $\alpha = 1.0$ performs best with $\alpha = 0.5$ being very close.

7.1.5 Discussion

In this section we have shown how BNN+LV can be used in model-based RL, in particular for gradient-based policy search. We have presented an algorithm that uses random rollouts and stochastic optimization for learning a parameterized policy in a batch scenario. This algorithm is particularly suited for industry domains.

BNN+LV have allowed us to solve the wet-chicken benchmark, which is a challenging problem where model-based approaches usually fail. They have also shown promising results on industry benchmarks including real-world data from a gas turbine. In particular, our experiments indicate that a BNN trained with $\alpha = 0.5$ or $\alpha = 1$ as divergence measure in conjunction with the presented algorithm for policy optimization is a powerful black-box method for policy search.

7.2 Risk-sensitive Reinforcement Learning

In the previous section we studied BNN+LV for model-based RL in a batch scenario. The objective of this study was to minimize expected costs. Here we found that BNN+LV serve as powerful models that give rise to policies that produce low costs on the target system. In Section 3.2 we developed the idea of decomposing predictive uncertainty in the predictions of BNN+LV into epistemic and aleatoric components: the epistemic component originates from lack of knowledge about the model parameters whereas the aleatoric component originates from stochasticity in the data. In Chapter 6 we visualized this on a set of tasks, which resulted in meaningful decompositions. As part of this chapter we further studied active learning scenarios (see Section 6.4) and showed that the decomposition could be used for sequential decision making.

Here, we want to see how we can further utilize the uncertainty decomposition in the field of reinforcement learning, in particular for *risk-sensitive* RL. In risk-sensitive RL the main idea is that the agent, in addition to minimizing costs, should also take into account a particular criterion of risk. One example of this is to avoid (potentially low-probability) worst-case events. This can be realized by augmenting the objective function with an additional objective that is called the *risk-sensitive criterion*. In this study we extend the standard risk-sensitive criterion via an uncertainty decomposition in BNN+LV. The decomposition enables us to define a novel risk-sensitive criterion to identify policies that balance expected cost, model-bias and noise aversion.

There exist numerous work on utilizing model uncertainty for safe or risk-sensitive RL (Mihatsch and Neuneier, 2002; Garcia and Fernandez, 2015). In safe RL numerous other approaches exist often in the context of exploration (Joseph et al., 2013; Hans et al., 2008; Garcia and Fernández, 2012; Berkenkamp et al., 2017). Uncertainties over transition probabilities have been studied in discrete MDPs since a long time (Shapiro and Kleywegt, 2002; Nilim and El Ghaoui, 2005; Bagnell et al., 2001) often with a focus on worst-case avoidance. Our work extends this to continuous state and action space using scalable probabilistic models.

The methods and experiments presented in this section are based on the publications Depeweg et al. (2017b, 2018).

7.2.1 Problem Description

We maintain the scenario of the previous section, we consider model-based RL in a batch scenario: we are given a batch of state transitions $\mathcal{D} = \{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})\}$ formed by triples containing the current state $\mathbf{s}_t$, the action applied $\mathbf{a}_t$ and the next state $\mathbf{s}_{t+1}$ from such a discrete-time dynamic system. We also want to perform policy search by gradient descent.

Following Garcia and Fernandez (2015) we can define a risk-sensitive criterion as:

Definition 7.2.1. (Risk-Sensitive Criterion) In risk-sensitive RL, the objective function includes a scalar parameter β that allows the desired level of risk to be controlled. The parameter β is known as the risk sensitivity parameter, and is generally either positive

or negative: $\beta > 0$ implies risk aversion, $\beta < 0$ implies a risk-seeking preference, and (through a limiting argument) $\beta = 0$ implies risk neutrality.

The classic risk-sensitive criterion penalizes the deviations of the cumulative cost C from the expectation $\mathbf{E}[C]$ for a particular starting state $\mathbf{s}_0$. To keep the notation simple, for the rest of this section we denote by τ roll-outs that all originate from the same starting state $\mathbf{s}_0$. For example, the risk-sensitive objective could be:

$$J(\mathcal{W}_\pi) = \mathbf{E}_{p(\tau|\mathcal{W}_\pi)}[C] + \sigma_{p(\tau|\mathcal{W}_\pi)}(C) \;, \tag{7.28}$$

where $\sigma_{p(\tau|\mathcal{W}_\pi)}(C)$ is the standard deviation of C across all trajectories and model instances under the current policy given parameterized by $\mathcal{W}_\pi$. The risk-sensitive parameter β determines the amount of risk-avoidance ($\beta \geq 0$) or risk-seeking behavior ($\beta < 0$) when optimizing $\mathcal{W}_\pi$.

Instead of working directly with the risk on the final cost C, we can alternatively consider the sum of risks on the individual costs $c_1, \ldots, c_T$.:

$$J(\mathcal{W}_\pi) = \sum_{t=1}^{T} \mathbf{E}_{p(\tau|\mathcal{W}_\pi)}[c_t] + \sigma_{p(\tau|\mathcal{W}_\pi)}(c_t) \tag{7.29}$$

Here we consider these individual risk terms for every time step t instead of the risk of the cumulative cost because the former is a more restrictive criterion since low risk on the c_t will imply low risk on C, but not the other way around.

In the next section we show how we can further decompose the standard risk term, that is $\sigma_{p(\tau|\mathcal{W}_\pi)}(c_t)$ utilizing the results from Section 3.2.

7.2.2 Uncertainty Decomposition for Risk-sensitive RL

The standard deviation $\sigma_{p(\tau|\mathcal{W}_\pi)}(c_t)$ is w.r.t. the distribution over τ, that is all trajectories that the policy π may generate, for a particular starting state $\mathbf{s}_0$. Two causes lead to a possible spread in the trajectories: the system may be stochastic, or we have uncertainty over the system dynamics. Applying the idea of the variance decomposition we introduced in Section 3.2 we can decompose this term into epistemic and aleatoric terms using the law of total variance. In particular,

$$\sigma_{p(\tau|\mathcal{W}_\pi)}(c_t) = \left\{ \sigma^2_{p(M)}(\mathbf{E}_{p(\tau|\mathcal{W}_\pi, M)}[c_t]) + \mathbf{E}_{p(M)}[\sigma^2_{p(\tau|\mathcal{W}_\pi, M)}(c_t)] \right\}^{\frac{1}{2}} \;, \tag{7.30}$$

where $\mathbf{E}_{p(\tau|\mathcal{W}_\pi, M)}[c_t]$ and $\sigma^2_{p(\tau|\mathcal{W}_\pi, M)}(c_t)$ denote the mean and variance of c_t under virtual roll-outs performed with policy $\mathcal{W}_\pi$ and under the dynamics of one particular model M. In a similar manner as in Eq. (3.26), the operators $\mathbf{E}_{p(M)}[\cdot]$ and $\sigma^2_{p(M)}(\cdot)$ in Eq. (7.30) compute the mean and variance of their arguments when $M \sim P(M)$.

In a BNN+LV we have that $p(M) = q(\mathcal{W})$ and the randomness in a model originates from sampling the latent variable z and the additive Gaussian noise $\boldsymbol{\epsilon}$. Here, the two terms inside the square root in Eq. (7.30) have a clear interpretation. The first one, that is $\sigma^2_{q(\mathcal{W})}(\mathbf{E}_{p(\tau|\mathcal{W}_\pi, \mathcal{W})}[c_t])$, encodes the risk originating from the sampling from the

distribution over models $q(\mathcal{W})$ in the virtual roll-outs. Any variation inside one particular model originating from randomness in z or $\boldsymbol{\epsilon}$ will be integrated out by the inner expectation $\mathbf{E}_{p(\tau|\mathcal{W}_\pi,\mathcal{W})}[c_t])$. We call this term the epistemic risk. By contrast the second term, $\mathbf{E}_{q(\mathcal{W})}[\sigma^2_{p(\tau|\mathcal{W}_\pi,\mathcal{W})}(c_t)]$, represents the expected risk originating from the sampling of z and $\boldsymbol{\epsilon}$ in the virtual roll-outs. This is because the outer expectation will integrate all variation that occurs due to uncertainty over the model parameters. Thus this term measures the average variability in each model due to stochastic effects and call this term the aleatoric risk.

We can now extend the objective in Eq. (7.10) with a new risk term that balances the epistemic and aleatoric risks. This term is obtained by first using risk-sensitive parameters β and γ to balance the epistemic and aleatoric components in Eq. (7.30), and then summing the resulting expression for $t = 1, \ldots, T$:

$$\sigma(\gamma, \beta) = \sum_{t=1}^{T} \left\{ \beta^2 \sigma^2_{q(\mathcal{W})}(\mathbf{E}_{p(\tau|\mathcal{W}_\pi,\mathcal{W})}[c_t]) + \gamma^2 \mathbf{E}_{q(\mathcal{W})}[\sigma^2_{p(\tau|\mathcal{W}_\pi,\mathcal{W})}(c_t)] \right\}^{\frac{1}{2}} . \tag{7.31}$$

Therefore, our 'risk-sensitive criterion" uses the function $J(\mathcal{W}_\pi) = \mathbf{E}[C] + \sigma(\gamma, \beta)$, which can be approximated via Monte Carlo and optimized using stochastic gradient descent. The Monte Carlo approximation is generated by performing $M \times N$ roll-outs with starting state $\mathbf{s}_0$ sampled uniformly from $\mathcal{D}$. For this, $\mathcal{W}$ is sampled from $q(\mathcal{W})$ a total of M times and then, for each of these samples, N roll-outs are performed with $\mathcal{W}$ fixed and sampling only the latent variables and the additive Gaussian noise in the BBN+LVs. Let $z_t^{m,n}$ and $\boldsymbol{\epsilon}_t^{m,n}$ be the samples of the latent variables and the additive Gaussian noise at step t during the n-th roll-out for the m-th sample of $\mathcal{W}$, which we denote by $\mathcal{W}^m$. Then

$$c_{m,n}(t) = c(\mathbf{s}_t^{\mathcal{W}^m, \{z_1^{m,n}, \ldots, z_t^{m,n}\}, \{\boldsymbol{\epsilon}_1^{m,n}, \ldots, \boldsymbol{\epsilon}_t^{m,n}\}, \mathcal{W}_\pi}) \tag{7.32}$$

denotes the cost obtained at time t in that roll-out. All these cost values obtained at time t are stored in the $M \times N$ matrix $\mathbf{C}(t)$. The Monte Carlo estimate of $J(\mathcal{W}_\pi)$ is then

$$J(\mathcal{W}_\pi) \approx \sum_{t=1}^{T} \left\{ \frac{\mathbf{1}^{\mathrm{T}} \mathbf{C}(t) \mathbf{1}}{MN} + \left\{ \beta^2 \hat{\sigma}^2 \left[\mathbf{C}(t) \mathbf{1}/N \right] \gamma^2 \frac{1}{M} \sum_{m=1}^{M} \hat{\sigma}^2 \left[\mathbf{C}(t)_{m,\cdot} \right] \right\}^{\frac{1}{2}} \right\} , \tag{7.33}$$

where $\mathbf{1}$ denotes a vector with all of its entries equal to 1, $\mathbf{C}(t)_{m,\cdot}$ is a vector with the m-th row of $\mathbf{C}(t)$ and $\hat{\sigma}^2[\mathbf{x}]$ returns the empirical variance of the entries in vector $\mathbf{x}$.

By setting β and γ to specific values in Eq. (7.33), the user can choose different trade-offs between cost, aleatoric and epistemic risk: for $\gamma = 0$ the term inside the square root is β^2 times $\hat{\sigma}^2 \left[\mathbf{C}(t) \mathbf{1}/N \right]$ which is a Monte Carlo approximation of the epistemic risk in Eq. (7.30). Similarly, for $\beta = 0$, inside the square root we obtain γ^2 times $\frac{1}{M} \sum_{m=1}^{M} \hat{\sigma}^2 \left[\mathbf{C}(t)_{m,\cdot} \right]$ which approximates the aleatoric risk. For $\beta = \gamma$ the standard risk criterion $\sigma(c_t)$ is obtained, weighted by β.

At the beginning of this chapter we identified the *model-bias* as one of the main challenges in model-based RL. We show now that the epistemic risk term in Eq. (7.30) can be connected to this concept. Model-bias occurs because a policy with $\mathcal{W}_\pi$ is optimized on the model but executed on the ground truth system. The more model and ground truth differ, the more the policy may be 'biased' by the model (Deisenroth and Rasmussen, 2011). Given the initial state $\mathbf{s}_0$, we can quantify this bias with respect to the policy parameters $\mathcal{W}_\pi$ as:

$$b(\mathcal{W}_\pi) = \sum_{t=1}^{T} \left(\mathbf{E}_{p(\tau|\mathcal{W}_\pi, M_{\text{true}})}[c_t] - \mathbf{E}_{p(\tau|\mathcal{W}_\pi)}[c_t] \right)^2 , \tag{7.34}$$

where $\mathbf{E}_{p(\tau|\mathcal{W}_\pi, M_{\text{true}})}[c_t]$ is the expected cost obtained at time t across roll-outs starting at $\mathbf{s}_0$, under the ground truth dynamics and with policy $\pi(\mathbf{s}_t; \mathcal{W}_\pi)$. $\mathbf{E}[c_t]$ is the same expectation but under BNN+LV dynamics sampled from $q(\mathcal{W})$ on each individual roll-out.

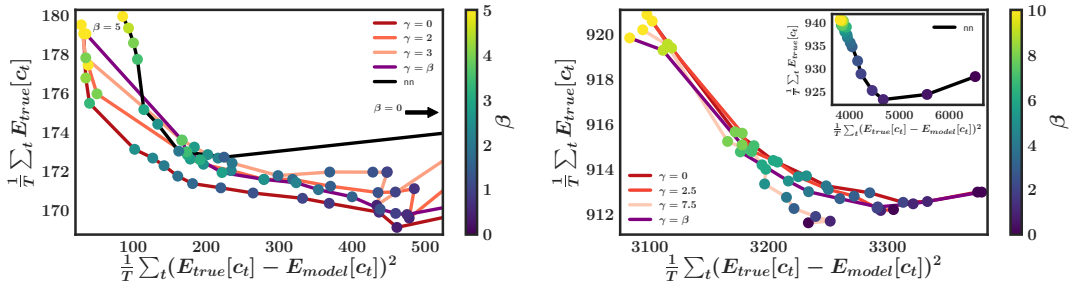
Unfortunately we may not be able to avoid the existence of such discrepancy when data is limited, which is always the case in a batch RL scenario: if the learning method does not have enough data, we can not expect it to accurately model the true dynamics. However, for some areas in state space we may have more data available than in others and consequently, we can expect the risk for model bias to be lower in these areas. Therefore the goal is to guide the policy search towards policies that stay in these areas of the state space where the risk for model-bias is low. Eq. (7.34) is impossible to compute in practice prior to policy execution on the ground-truth system. However, let us assume that the true dynamic can be expressed by a neural network with latent variables and weights $\mathcal{W}_{\text{true}}$. We can then rewrite $\mathbf{E}_{p(\tau|\mathcal{W}_\pi, M_{\text{true}})}[c_t]$ as $\mathbf{E}_{p(\tau|\mathcal{W}_\pi, \mathcal{W}_{\text{true}})}[c_t]$ and since we do not know $\mathcal{W}_{\text{true}}$, we can further assume that $\mathcal{W}_{\text{true}} \sim q(\mathcal{W})$. The expected model-bias is then

$$\mathbf{E}[b(\mathcal{W}_\pi)] = \mathbf{E}_{q(\mathcal{W})} \left\{ \sum_{t=1}^{T} \left(\mathbf{E}_{p(\tau|\mathcal{W}_\pi, \mathcal{W}_{\text{true}})}[c_t] - \mathbf{E}_{p(\tau|\mathcal{W}_\pi)}[c_t] \right)^2 \right\} \tag{7.35}$$

$$= \sum_{t=1}^{T} \sigma^2_{q(\mathcal{W})}(\mathbf{E}_{p(\tau|\mathcal{W}_\pi, \mathcal{W})}[c_t]) \, . \tag{7.36}$$

We see that our definition of epistemic risk also represents an estimate of model-bias in model-based RL. This risk term will guide the policy to operate in areas of state space where model-bias is expected to be low.

The aleatoric risk term in Eq. (7.30) can be connected with the concept of noise aversion. Let $\sigma^2_{p(\tau|\mathcal{W}_\pi, \mathcal{W}_{\text{true}})}(c_t)$ be the variance obtained at time t across roll-outs starting at $\mathbf{s}_0$, under the ground truth dynamics and with policy $\pi(\mathbf{s}_t; \mathcal{W}_\pi)$. Assuming $\mathcal{W}_{\text{true}} \sim q(\mathcal{W})$, the expected variance is then $\mathbf{E}_{q(\mathcal{W})}[\sigma^2_{p(\tau|\mathcal{W}_\pi, \mathcal{W}}(c_t)]$. This term will guide the policy to operate in areas of state space where the stochasticity of the cost is low. Assuming a deterministic cost function this stochasticity is determined by the model's predictions that originate from z_t and $\boldsymbol{\epsilon}_t$.



(a) **(b)**

Figure 7.5: RL experiments. (a): results on Industrial Benchmark. (b): results on wind turbine simulator. Each curve shows average cost (y-axis) against model bias (x-axis). Circle colour correspond to different values of β (model-bias weight) and curve color indicates different values of γ (noise-averseness weight). The purple curve is the baseline $\gamma = \beta$. The black curve is nearest neighbor baseline.

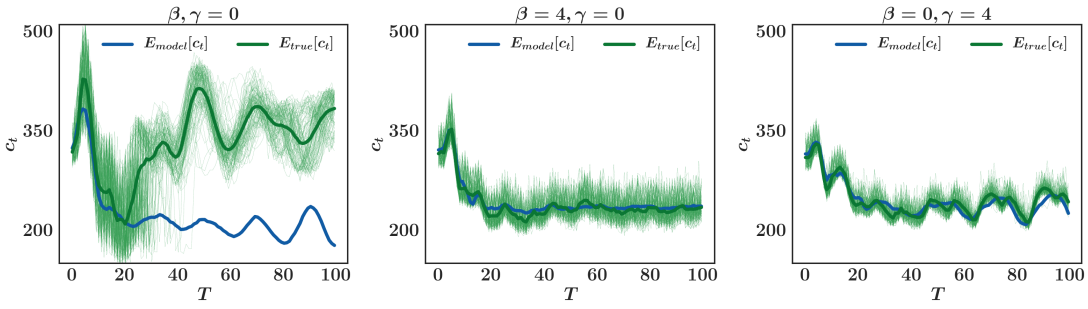
7.2.3 Model & Baseline Specification

We consider the risk-sensitive criterion for different choices of β and γ, comparing it with three baselines. The first baseline is obtained by setting $\beta = \gamma = 0$. In this case, the policy optimization ignores any risk and is therefore the method we used in Section 7.1. The second baseline is obtained when $\beta = \gamma$. In this case, the risk criterion simplifies to $\beta\sigma(c_t)$, which corresponds to the traditional risk-sensitive approach in Eq. (7.28), but applied to the individual costs $c_1, \ldots, c_T$. The last baseline uses a deterministic neural network to model the dynamics and a nearest neighbor approach to quantify risk: for each state $\mathbf{s}_t$ generated in a roll-out, we calculate the Euclidean distance of that state to the nearest one in the training data. The average value of the distance metric for $\mathbf{s}_t$ across roll-outs is then an approximation to $\sigma(c_t)$. To reduce computational cost, we summarize the training data using the centroids returned by an execution of the k-means clustering method. We denote this method as the *nn*-baseline. We will specify hyper-parameters in the following section.

7.2.4 Experiments

We investigate the following questions: To what extent does our new risk criterion reduce model-bias? What trade-offs do we observe between average cost and model-bias? How does the decomposition compare to other simple methods? For this we consider two model-based RL scenarios. The first one is given by the industrial benchmark (see Section 4.3.2) and the second RL scenario is a modified version of the HAWC2 wind turbine simulator. (see Section 4.3.4).

We are given a batch of data formed by state transitions generated by a behavior policy π_b, for example, from an already running system. The behavioral policy has limited randomness and will keep the system dynamics constrained to a reduced manifold



(a) **(b)** **(c)**

Figure 7.6: 100 roll-outs on industrial benchmark ground truth system (light green), their average (dark green), and the average of corresponding roll-outs on the BNN+LVs model (blue) for a fixed value of the initial state $\mathbf{s}_0$. We show results for three policies with different model-bias and noise-averseness trade-offs. Policies are optimized using (a): no risk penalty ($\beta, \gamma = 0$). (b): a penaly on the model-bias only ($\gamma = 0, \beta = 4$). (c): a penalty on the noise risk only ($\gamma = 4, \beta = 0$).

in state space. This means that large portions of state space will be unexplored and uncertainty will be high in those regions.

Industrial Benchmark Policies in the industrial benchmark specify changes Δ_v, Δ_g and Δ_s in three steering variables v (velocity), g (gain) and s (shift) as a function of $\mathbf{s}_t$. In the behavior policy these changes are stochastic and sampled according to

$$\Delta_v = \begin{cases} z_v\,, & \text{if } v(t) < 40 \\ -z_v\,, & \text{if } v(t) > 60 \\ u_v\,, & \text{otherwise} \end{cases} \tag{7.37}$$

$$\Delta_g = \begin{cases} z_g\,, & \text{if } g(t) < 40 \\ -z_g\,, & \text{if } g(t) > 60 \\ u_g\,, & \text{otherwise} \end{cases} \tag{7.38}$$

$$\Delta_s = u_s\,, \tag{7.39}$$

where $z_v, z_g \sim \mathcal{N}(0.5, \frac{1}{\sqrt{3}})$ and $u_v, u_g, u_s \sim \mathcal{U}(-1, 1)$. The velocity $v(t)$ and gain $g(t)$ can take values in $[0, 100]$. Therefore, the data collection policy will try to keep these values only in the medium range given by the interval $[40, 60]$. Because of this, large parts of the state space will be unobserved. After collecting the data, the $30,000$ state transitions are used to train a BNN with latent variables with the same hyperparameters as in Section 7.1. Finally, we train different policies using the Monte Carlo approximation and we set the horizon of $T = 100$ steps, with $M = 50$ and $N = 25$ and a mini-batch size of 1 for 750 epochs. The total training time on a single CPU is around 18 hours.

Figure 7.5a shows results. The y-axis in the plot is the average total cost at horizon T obtained by the policy in the ground truth system. The x-axis is the average model-bias in the ground truth system according to Eq. (7.34). Each individual curve in the plot is

obtained by fixing γ to a specific value (line colour) and then changing β (circle colour). The policy that ignores risk ($\beta = \gamma = 0$) results in both high model-bias and high cost when evaluated on the ground truth, which indicates overfitting. As β increases, the policies put more emphasis on avoiding model bias, but at the same time the average cost increases. The best tradeoff is obtained by the dark red curve with $\gamma = 0$. The risk criterion is then $\beta \sum_{t=1}^{T} \sigma_{q(\mathcal{W})}(\mathbf{E}[c_t|\mathcal{W}])$. In this problem, adding noise risk by setting $\gamma > 0$ decreases performance. The *nn* baseline shows a similar pattern as the BNN+LVs approach, but the trade-off between model-bias and cost is worse.

Figure 7.6 shows roll-outs for three different policies and a fixed initial state $\mathbf{s}_0$. Figure 7.6a shows results for a policy learned with $\gamma = \beta = 0$. This policy ignores risk, and as a consequence, the mismatch between predicted performance on the model and the ground truth increases after $t = 20$. This result illustrates how model bias can lead to policies with high costs at test time. Figure 7.6b shows results for policy that was trained while penalizing model-bias risk ($\beta = 4$, $\gamma = 0$). In this case, the average costs under the BNN+LVs model and the ground truth are similar, and the overall ground truth cost is lower than in Figure 7.6a. Finally, Figure 7.6c shows results for a noise-averse policy ($\beta = 0$, $\gamma = 4$). In this case, the model bias is slightly higher than in the previous figure, but the stochasticity is lower.

Wind Turbine Simulator We are given a batch of around 5,000 state transitions generated by a behavior policy π_b. The policy does limited exploration around the neutral action $a(t) = 0$. The system is expected to be highly stochastic due to the unpredictability of future wind dynamics. Furthermore the dimensionality of state observation is much higher than the action dimensionality, so, with the limited data set that we have, we expect it to be very challenging to learn the influence of the action on the reward accurately. First we train a BNN with two hidden layers and 50 hidden units per layer on the available batch using α-divergence minimization with $\alpha = 1.0$. In the second step, using the model, we train a policy with 20 hidden units on each of the two layers in the usual way, using the Monte Carlo estimate. The total training time on a single CPU is around 8 hours.

The results for wind turbine simulator can be found in Figure 7.5b. As in the previous experiment we see that the epistemic risk weight β acts as a trade-off between model-bias and policy cost: as β increases the model-bias decreases but so does the policy performance. The best trade-offs between expected cost and model bias are obtained by the policies with $\gamma = 7.5$. These policies are noise-averse and will try to avoid noisy regions in state space. This makes sense because in wind turbines, high noise regions in state space are those where the effect of wind turbulence will have a strong impact on the average cost. We believe that in this problem being noise-averse is a good heuristic: choosing actions that make the system behave more noisy are sub-optimal. If the policy that generated the data explored these noisy regions they are considered "safe" in terms of epistemic uncertainty, and hence the epistemic risk term will not penalize these actions in any way.

7.2.5 Discussion

In this study we have utilized the decomposition of predictive uncertainty into its epistemic and aleatoric components for risk-sensitive reinforcement learning. In particular we proposed a novel risk-sensitive criterion for model-based reinforcement learning which decomposes risk into model-bias and noise aversion components. This criterion enables a user to implement more fine-grained preferences for policy search. In particular, the user can specify the weights of the objective function that minimize costs, avoiding stochasticity and minimize the risk for model-bias.

The experiments confirmed that this specification leads to a minimization of these respective quantities and thus how the decomposition of uncertainty is useful for decision making in the context of risk.

8 Conclusions & Outlook

Conclusions

As the centerpiece of this thesis we have developed a novel method for supervised learning, called Bayesian neural network with latent variables (BNN+LV) in Chapter 3. The motivation for BNN+LV was to fuse three properties in one machine learning method: First, the method should be a powerful black-box tool for function approximation. Second, it should be a probabilistic model that can express uncertainty over its parameters. Lastly, the method should be able to model stochastic effects in the data such as heteroscedasticity or multimodality. This combination of properties enables modeling both epistemic and aleatoric uncertainty with high flexibility. The former due to inferring uncertainty estimates over the parameter of the model and the latter due to the addition of latent variables with allows to express the aforementioned stochastic effects in the data. For decision-making we have shown how these two forms of uncertainty can be decomposed in the model's predictive distribution for different metrics to measure uncertainty.

In the second part of the thesis we have investigated the applicability of BNN+LV in different supervised learning scenarios, with a focus to confirm these aforementioned properties of the model empirically.

Regression We have compared the predictive performance of BNN+LV on a wide range of tasks and to several baselines. Overall we found that BNN+LV provide better estimates of uncertainty in terms of test log-likelihood compared to all baselines, especially in settings known to posses stochastic effects. For function approximation, measured in RMSE, BNN+LV outperformed standard neural networks but were inferior to ensembles. We believe one reason for this to be the lower heterogeneity in parameter space compared to ensembles. Furthermore we confirmed empirically that the choice of divergence measure in the variational approximation seems to act as a trade-off between the quality of uncertainty estimates and function approximation.

Model Inspection In this chapter we investigated predictive uncertainty in BNN+LV. We found that on a set of tasks, both artificial and in vision, the uncertainty decomposition provided meaningful results and matched the intuition behind the data generating process: in areas of data scarcity we saw an increase in epistemic, and in areas of increased stochasticity an increase in aleatoric uncertainty. We further found that the choice of divergence measure in BNN+LV affects the decomposition. Using the sensitivity analysis we showed how to measure the impact of each feature to both epistemic

and aleatoric components in standard regression problems. Lastly, we have shown how to utilize the decomposition in BNN+LV for more efficient sequential decision making: in a set of active learning experiments a data-acquisition strategy that utilizes the decomposition of uncertainty outperformed both GPs and a strategy that uses the full uncertainty.

Reinforcement Learning In the field of model-based reinforcement learning (RL) we studied the usefulness of BNN+LV as models for dynamic systems. Here, our main results are that BNN+LV serve as powerful models, especially for stochastic systems, that give rise to policies that produce lower costs than baseline methods. We further developed a novel risk-sensitive criterion in the field of risk-sensitive RL, that enables a practitioner to specify its preference for policies that minimize costs, avoid stochasticity and minimize the risk for model-bias.

Future Work

At the end of this thesis we want to address a set of open problems that pose interesting research directions and extensions to the method proposed in this thesis.

I. Modeling temporal dependencies & POMDPs:
In BNN+LV and in stochastic dynamic systems, as described in Bertsekas (2002), one central assumption is to assume temporal independence in the latent variable z. For standard supervised learning this assumption is reasonable and the latent variable of the BNN+LV enables modeling stochastic effects in the data, which was the main motivation of developing this method in the first place. However, for temporal dynamics, such as constructing roll-outs in model-based RL, this assumption implies that we rely on the MDP assumption of the form of Eq. (7.5). For some applications this MDP assumption will not hold, for instance the industrial benchmark is a partially observable system (POMDP). We approached this problem using a time-embedding of previous observations $\mathbf{o}(t-k), \ldots, \mathbf{o}(t)$ to approximate a MDP state. Ideally, we want a more principled approach, that considers temporal dependencies of the latent variable.

II. Bias of variational inference:
In Section 2.2 we introduced variational inference as one technique that allows more complex probabilistic models, such as the BNN+LV, to be computationally tractable. In Section 3.1.4 we further introduced amortized inference in BNN+LV as another technique for approximation. The effect of these approximations is still an open question in research because obtaining a ground-truth can be difficult in large data settings. We conducted an initial study on this topic in Section 6.1. Here we found that the effect of approximation and divergence measure used can have a large effect on the decomposition of uncertainty in BNN+LV. The work in Cremer et al. (2018) provides a first analysis into the effect of the bias of amortization and the mean-field approach commonly used in VI.

III. Optimal training in BNN+LV: A stage-wise process?
BNN+LV try to solve three problems at once: function approximation of the deterministic structure of the data, uncertainty estimates over this function and modeling stochastic effects in the data by inference of the latent variables. This can in principle lead to difficult training and sub-optimal convergence. In its current state, these problems are solved simultaneously by minimizing the energy function in Eq. (3.15) through gradient descent. Initializing the parameters of the model in a certain way, such as starting with low values of the variance of the weights, is an attempt to encourage a stage-wise training process. However, how to most effectively train this model is still an open question.

IV. Incorporation of prior knowledge & Interpretability
BNN+LV are parametric models based on neural networks. These methods are considered black-box and usually have a large set of parameters. It is therefore not obvious how prior knowledge about a problem can be incorporated in the inference process. By contrast, in a Gaussian processes it is much easier to formalize knowledge about a problem into the prior. Prior knowledge can potentially overcome the tendency of BNN+LV to underfit, an issue we observed for instance in Chapter 5. Here, we want to highlight two recent works that address this issue. The work in Flam-Shepherd et al. (2017) presents a method to transfer a GP prior, which is formulated in function space, into a prior over parameters of BNNs. Ghosh et al. (2018) develop so-called horseshoe priors on the weights to induce sparsity for better model selection.

Furthermore, after training there is no simple way to obtain insight about the nature and properties of the approximated function. We have performed a set of model inspection studies in Chapter 6 and introduced a novel method for interpretability: the sensitivity analysis of predictive uncertainty to approach these questions. However, understanding neural network models is still an ongoing research topic (e.g. Koh and Liang (2017), Montavon et al. (2018)).

Bibliography

Shun-Ichi Amari. Differential geometry of curved exponential families-curvatures and information loss. *The Annals of Statistics*, pages 357–385, 1982.

Hagai Attias. Inferring parameters and structure of latent variable models by variational Bayes. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 21–30. Morgan Kaufmann Publishers Inc., 1999.

J. Andrew Bagnell and Jeff G. Schneider. Autonomous helicopter control using reinforcement learning policy search methods. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*, volume 2, pages 1615–1620. IEEE, 2001.

J. Andrew Bagnell, Andrew Y Ng, and Jeff G Schneider. Solving uncertain markov decision processes. 2001.

Anoop Korattikara Balan, Vivek Rathod, Kevin P Murphy, and Max Welling. Bayesian dark knowledge. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3438–3446, 2015.

Jan Beirlant, Edward J. Dudewicz, László Györfi, and Edward C. Van der Meulen. Nonparametric entropy estimation: An overview. *International Journal of Mathematical and Statistical Sciences*, 6(1):17–39, 1997.

James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: A cpu and gpu math compiler in python. In *Proc. 9th Python in Science Conf*, volume 1, 2010.

Felix Berkenkamp, Matteo Turchetta, Angela P. Schoellig, and Andreas Krause. Safe model-based reinforcement learning with stability guarantees. In *Advances in Neural Information Processing Systems (NIPS)*, pages 908–918, 2017.

Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific optimization and computation series. 2002. ISBN 9781886529083.

Christopher M. Bishop. *Pattern recognition and machine learning, 5th Edition*. Information science and statistics. Springer, 2007. ISBN 9780387310732.

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

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. In *International Conference on Machine Learning (ICML)*, pages 1613–1622, 2015.

Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. *Handbook of markov chain monte carlo*. CRC press, 2011.

Gavin Brown. *Diversity in neural network ensembles*. PhD thesis, Citeseer, 2004.

Thang D. Bui, Daniel Hernández-Lobato, Yingzhen Li, José Miguel Hernández-Lobato, and Richard E. Turner. Deep Gaussian processes for regression using approximate expectation propagation. In *International Conference on Machine Learning (ICML)*, pages 1472–1481, 2016.

Keith Bush and Joelle Pineau. Manifold embeddings for model-based reinforcement learning under partial observability. In *Advances in neural information processing systems*, pages 189–197, 2009.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *arXiv preprint arXiv:1805.12114*, 2018.

Chris Cremer, Xuechen Li, and David Duvenaud. Inference suboptimality in variational autoencoders. *arXiv preprint arXiv:1801.03558*, 2018.

Guillaume Dehaene and Simon Barthelmé. Expectation propagation in the large data limit. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 80 (1):199–217, 2018.

Marc Deisenroth and Carl E. Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning (ICML)*, pages 465–472, 2011.

Marc Peter Deisenroth, Gerhard Neumann, and Jan Peters. A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2:1–142, 2013.

Stefan Depeweg, José Miguel Hernández-Lobato, Finale Doshi-Velez, and Steffen Udluft. Learning and policy search in stochastic dynamical systems with Bayesian neural networks. *International Conference on Learning Representations (ICLR)*, 2017a.

Stefan Depeweg, José Miguel Hernández-Lobato, Finale Doshi-Velez, and Steffen Udluft. Uncertainty decomposition in Bayesian neural networks with latent variables. *Workshop on Reliable Machine Learning in the Wild, ICML*, 2017b.

Stefan Depeweg, José Miguel Hernández-Lobato, Steffen Udluft, and Thomas Runkler. Sensitivity analysis for predictive uncertainty in Bayesian neural networks. *European Symposium on Artificial Neural Networks, (ESANN)*, 2017c.

Stefan Depeweg, Jose Miguel Hernandez-Lobato, Finale Doshi-Velez, and Steffen Udluft. Decomposition of uncertainty in Bayesian deep learning for efficient and risk-sensitive learning. In *International Conference on Machine Learning (ICML)*, pages 1192–1201, 2018.

Armen Der Kiureghian and Ove Ditlevsen. Aleatory or epistemic? does it matter? *Structural Safety*, 31(2):105–112, 2009.

Andreas Draeger, Sebastian Engell, and Horst Ranke. Model predictive control using neural networks. *IEEE Control systems*, 15(5):61–66, 1995.

Rachit Dubey, Pulkit Agrawal, Deepak Pathak, Thomas L. Griffiths, and Alexei A. Efros. Investigating human priors for playing video games. *arXiv preprint arXiv:1802.10217*, 2018.

Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(Apr):503–556, 2005.

Daniel Flam-Shepherd, James Requeima, and David Duvenaud. Mapping Gaussian process priors to Bayesian neural networks. In *NIPS Bayesian deep learning workshop*, 2017.

Jerome H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

Li Fu and Tinghuai Chen. Sensitivity analysis for input vector in multilayer feedforward neural networks. In *Neural Networks, 1993., IEEE International Conference on*, pages 215–218. IEEE, 1993.

Yarin Gal. Uncertainty in deep learning. *University of Cambridge*, 2016.

Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

Yarin Gal and Lewis Smith. Idealised Bayesian neural networks cannot have adversarial examples: Theoretical and empirical study. *arXiv preprint arXiv:1806.00667*, 2018.

Yarin Gal, Rowan McAllister, and Carl E. Rasmussen. Improving PILCO with Bayesian neural networks dynamics models. In *Data-Efficient Machine Learning workshop, ICML*, 2016.

Weihao Gao, Sewoong Oh, and Pramod Viswanath. Breaking the bandwidth barrier: Geometrical adaptive entropy estimation. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2460–2468, 2016.

Javier Garcia and Fernando Fernández. Safe exploration of state and action spaces in reinforcement learning. *Journal of Artificial Intelligence Research*, 45:515–564, 2012.

Javier Garcia and Fernando Fernandez. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 1(0):1437–1480, 2015.

Zoubin Ghahramani and Matthew J. Beal. Variational inference for Bayesian mixtures of factor analysers. In *Advances in Neural Information Processing Systems (NIPS)*, pages 449–455, 2000.

Soumya Ghosh, Jiayu Yao, and Finale Doshi-Velez. Structured variational learning of Bayesian neural networks with horseshoe priors. *arXiv preprint arXiv:1806.05975*, 2018.

Geoffrey J. Gordon. Stable function approximation in dynamic programming. In *Machine Learning Proceedings 1995*, pages 261–268. Elsevier, 1995.

Delve Development Group. Delve Datasets. `https://www.cs.toronto.edu/~delve/data/datasets.html`, 2019. [Online; accessed 09-January-2019].

Yuhong Guo and Russell Greiner. Optimistic active-learning using mutual information. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 823–829, 2007.

Arjun K. Gupta and Daya K. Nagar. *Matrix variate distributions*. Chapman and Hall/CRC, 1999.

Alexander Hans and Steffen Udluft. Efficient uncertainty propagation for reinforcement learning with limited data. In *ICANN*, pages 70–79. Springer, 2009.

Alexander Hans, Daniel Schneegaß, Anton Maximilian Schäfer, and Steffen Udluft. Safe exploration for reinforcement learning. In *ESANN*, pages 143–148, 2008.

Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 12(10):993–1001, 1990.

He He and Wan-Chi Siu. Single image super-resolution using Gaussian process regression. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 449–456. IEEE, 2011.

Donald O. Hebb. The organization of behavior. a neuropsychological theory. 1949.

Daniel Hein, Stefan Depeweg, Michel Tokic, Steffen Udluft, Alexander Hentschel, Thomas Runkler, and Volkmar Sterzing. A benchmark environment motivated by industrial control problems. In *IEEE Symposium Series on Computational Intelligence (SSCI)*, 2017a.

Daniel Hein, Alexander Hentschel, Thomas Runkler, and Steffen Udluft. Particle swarm optimization for generating interpretable fuzzy reinforcement learning policies. *Engineering Applications of Artificial Intelligence*, 65:87–98, 2017b.

Daniel Hein, Alexander Hentschel, Thomas Runkler, and Steffen Udluft. Particle swarm optimization for model predictive control in reinforcement learning environments. In *Critical Developments and Applications of Swarm Intelligence*, pages 401–427. IGI Global, 2018.

James Hensman and Neil D. Lawrence. Nested variational compression in deep Gaussian processes. *arXiv preprint arXiv:1412.1370*, 2014.

José Miguel Hernández-Lobato and Ryan Adams. Probabilistic backpropagation for scalable learning of Bayesian neural networks. In *International Conference on Machine Learning*, pages 1861–1869, 2015.

José Miguel Hernández-Lobato, Matthew W. Hoffman, and Zoubin Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In *Advances in Neural Information Processing Systems (NIPS)*, pages 918–926, 2014.

José Miguel Hernández-Lobato, Yingzhen Li, Mark Rowland, Daniel Hernández-Lobato, Thang Bui, and Richard E. Turner. Black-box α-divergence minimization. In *International Conference on Machine Learning (ICML)*, pages 1511–1520, 2016.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. VIME: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1109–1117, 2016.

Edwin T. Jaynes. *Probability theory: The logic of science*. Cambridge university press, 2003.

Joshua Joseph, Alborz Geramifard, John W. Roberts, Jonathan P. How, and Nicholas Roy. Reinforcement learning with misspecified model classes. In *International Conference on Robotics and Automation (ICRA)*, pages 939–946, 2013.

Markus Kaiser, Clemens Otte, Thomas Runkler, and Carl Henrik Ek. Bayesian alignments of warped multi-output Gaussian processes. *arXiv preprint arXiv:1710.02766*, 2017.

Alex Kendall and Yarin Gal. What uncertainties do we need in Bayesian deep learning for computer vision? In *Advances in Neural Information Processing Systems (NIPS)*, pages 5574–5584, 2017.

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

Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Diederik P. Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2575–2583, 2015.

Jonathan Ko, Daniel J. Klein, Dieter Fox, and Dirk Haehnel. Gaussian processes and reinforcement learning for identification and control of an autonomous blimp. In *IEEE Robotics and Automation*, pages 742–747, 2007.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. *arXiv preprint arXiv:1703.04730*, 2017.

L.F. Kozachenko and Nikolai N. Leonenko. Sample estimate of the entropy of a random vector. *Problemy Peredachi Informatsii*, 23(2):9–16, 1987.

Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. Estimating mutual information. *Physical review E*, 69(6):066138, 2004.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1097–1105, 2012.

Malte Kuss and Carl E. Rasmussen. Gaussian processes in reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 751–758, 2004.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems (NIPS)*, pages 6402–6413, 2017.

Sascha Lange, Thomas Gabel, and Martin Riedmiller. Batch reinforcement learning. In *Reinforcement learning*, pages 45–73. Springer, 2012.

Torben J. Larsen and Anders Melchior Hansen. How 2 hawc2, the user's manual. Technical report, Risø National Laboratory, 2007.

Torben J. Larsen, Gunner Larsen, Helge A Madsen, Kenneth Thomsen, and Søren M Pedersen. Comparison of measured and simulated loads for the siemens swt2.3 operating in wake conditions at the lillgrund wind farm using hawc2 and the dynamic wake meander model. *EWEA offshore 2015*, 2015.

Miguel Lázaro-Gredilla and Michalis Titsias. Variational heteroscedastic Gaussian process regression. 2011.

Quoc V. Le, Alex J. Smola, and Stéphane Canu. Heteroscedastic Gaussian process regression. In *International Conference on Machine Learning (ICML)*, pages 489–496, 2005.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Yann LeCun, Sumit Chopra, Raia Hadsell, M. Ranzato, and F. Huang. A tutorial on energy-based learning. *Predicting structured data*, 1(0), 2006.

Yingzhen Li. *Approximate Inference: New Visions*. PhD thesis, University of Cambridge, 2018.

Yingzhen Li and Yarin Gal. Dropout inference in Bayesian neural networks with alpha-divergences. *arXiv preprint arXiv:1703.02914*, 2017.

Yingzhen Li, José Miguel Hernández-Lobato, and Richard E. Turner. Stochastic expectation propagation. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2323–2331, 2015.

Moshe Lichman. UCI machine learning repository. `http://archive.ics.uci.edu/ml`, 2013.

Christos Louizos and Max Welling. Structured and efficient variational deep learning with matrix Gaussian posteriors. In *International Conference on Machine Learning*, pages 1708–1716, 2016.

David J.C. MacKay. Information-based objective functions for active data selection. *Neural computation*, 4(4):590–604, 1992.

Stephan Mandt, Matthew D. Hoffman, and David M. Blei. Stochastic gradient descent as approximate Bayesian inference. *The Journal of Machine Learning Research*, 18 (1):4873–4907, 2017.

Hermann G. Matthies. Quantifying uncertainty: modern computational representation of probability and applications. In *Extreme man-made and natural hazards in dynamics of structures*, pages 105–135. Springer, 2007.

Rowan McAllister, Yarin Gal, Alex Kendall, Mark Van Der Wilk, Amar Shah, Roberto Cipolla, and Adrian Vivian Weller. Concrete problems for autonomous vehicle safety: Advantages of Bayesian deep learning. International Joint Conferences on Artificial Intelligence, Inc., 2017.

Oliver Mihatsch and Ralph Neuneier. Risk-sensitive reinforcement learning. *Machine learning*, 49(2-3):267–290, 2002.

Thomas Minka. Power ep. Technical report, Technical report, Microsoft Research, Cambridge, 2004.

Tom Minka. Divergence measures and message passing. Technical report, Microsoft Research, 2005.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 73:1–15, 2018.

Radford M. Neal. *Bayesian Learning for Neural Networks*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996. ISBN 0387947248.

Yurii Nesterov. A method for unconstrained convex minimization problem with the rate of convergence o (1/kˆ 2). In *Doklady AN USSR*, volume 269, pages 543–547, 1983.

Andrew Y. Ng and Michael Jordan. Pegasus: A policy search method for large mdps and pomdps. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 406–415. Morgan Kaufmann Publishers Inc., 2000.

Arnab Nilim and Laurent El Ghaoui. Robust control of markov decision processes with uncertain transition matrices. *Operations Research*, 53(5):780–798, 2005.

David A. Nix and Andreas S. Weigend. Estimating the mean and variance of the target probability distribution. In *Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference On*, volume 1, pages 55–60. IEEE, 1994.

John Paisley, David Blei, and Michael Jordan. Variational Bayesian inference with stochastic search. *arXiv preprint arXiv:1206.6430*, 2012.

Jan Peters and Stefan Schaal. Policy gradient methods for robotics. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 2219–2225. IEEE, 2006.

Rajesh Ranganath, Sean Gerrish, and David Blei. Black box variational inference. In *Artificial Intelligence and Statistics*, pages 814–822, 2014.

Carl E. Rasmussen. Gaussian processes in machine learning. In *Advanced lectures on machine learning*, pages 63–71. Springer, 2004.

Alfréd Rényi. On measures of entropy and information. Technical report, Hungarian academy of sciences, 1961.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

Anton M. Schaefer, Steffen Udluft, and Hans-Georg Zimmermann. The recurrent control neural network. In *ESANN*, pages 319–324. Citeseer, 2007.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

Matthias Seeger. Expectation propagation for exponential families. Technical report, 2005.

Robin Senge, Stefan Bösner, Krzysztof Dembczyński, Jörg Haasenritter, Oliver Hirsch, Norbert Donner-Banzhoff, and Eyke Hüllermeier. Reliable classification: Learning classifiers that distinguish aleatoric and epistemic uncertainty. *Information Sciences*, 255:16–29, 2014.

Burr Settles. Active learning literature survey. *University of Wisconsin, Madison*, 52 (55-66):11, 2010.

Alexander Shapiro and Anton Kleywegt. Minimax analysis of stochastic problems. *Optimization Methods and Software*, 17(3):523–542, 2002.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

Edward Snelson and Zoubin Ghahramani. Sparse Gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1257–1264, 2005.

Edward Snelson and Zoubin Ghahramani. Sparse Gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1257–1264, 2006.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Andreas Stuhlmüller, Jacob Taylor, and Noah Goodman. Learning stochastic inverses. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3048–3056, 2013.

Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. Lstm neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*, 2012.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3104–3112, 2014.

Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Floris Takens. Detecting strange attractors in turbulence. In *Dynamical systems and turbulence, Warwick 1980*, pages 366–381. Springer, 1981.

Mattias Teye, Hossein Azizpour, and Kevin Smith. Bayesian uncertainty estimation for batch normalized deep networks. *arXiv preprint arXiv:1802.06455*, 2018.

Emanuel Todorov and Weiwei Li. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *American Control Conference, 2005. Proceedings of the 2005*, pages 300–306. IEEE, 2005.

Volker Tresp. The wet game of chicken. Technical report, 1994.

Constantino Tsallis. Possible generalization of boltzmann-gibbs statistics. *Journal of statistical physics*, 52(1-2):479–487, 1988.

Bo Wahlberg. System identification using Laguerre models. *IEEE Transactions on Automatic Control*, 36:551–562, 1991.

Max Welling and Yee W. Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *International Conference on Machine Learning (ICML)*, pages 681–688, 2011.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

Hans-Georg Zimmermann, Christoph Tietz, and Ralph Grothmann. Forecasting with recurrent neural networks: 12 tricks. In *Neural Networks: Tricks of the Trade*, pages 687–707. Springer, 2012.

Index