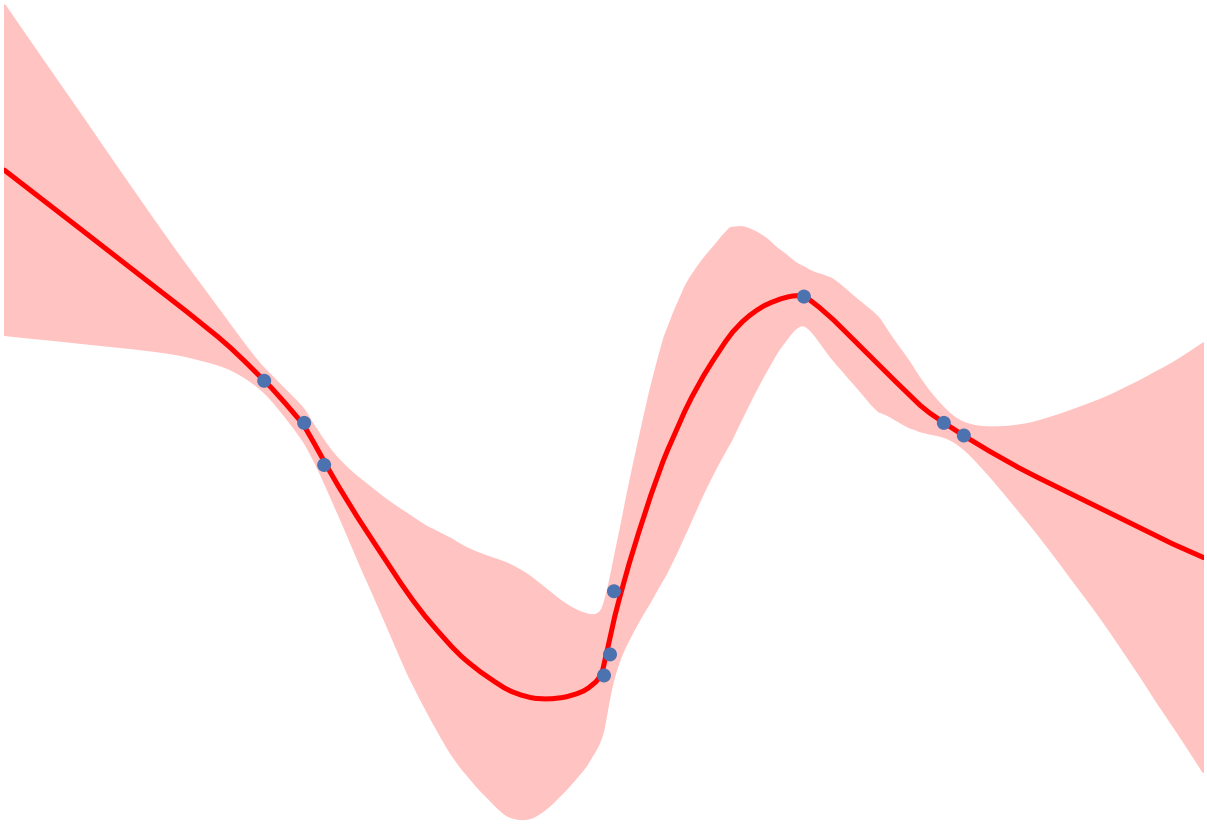


Uncertainty-Aware Models for Deep Reinforcement Learning

Master's thesis in Engineering Mathematics and Computational Science

JOHN MOBERG

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Master's thesis 2019

Uncertainty-Aware Models for Deep Reinforcement Learning

John Moberg

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2019

Uncertainty-Aware Models for Deep Reinforcement Learning
John Moberg

Supervisor and Examiner: Lennart Svensson, Department of Electrical Engineering

Master's Thesis 2019
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The posterior predictive distribution of Bayesian linear regression on the representation of the data in the last hidden layer of a neural network trained for the regression task, as described in Section 3.4.

Typeset in LaTeX
Gothenburg, Sweden 2019

Uncertainty-Aware Models for Deep Reinforcement Learning
John Moberg
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Tractable methods for obtaining uncertainty-aware neural networks (NNs), NNs that *know what they don't know*, have only recently been proposed and no winner has been crowned yet. The ability to take predictive uncertainty into account may be critical for real-world applications, and has been shown to improve performance in some decision-making applications. For instance, recent work in reinforcement learning (RL) indicates that uncertainty-aware models may play a central role in getting model-based RL to work in complex environments where any model will be imperfect.

In this thesis, we study five methods for obtaining uncertainty-aware NNs: Monte Carlo dropout (MC-dropout), probabilistic ensembles (PE), cyclical stochastic gradient Markov chain Monte Carlo (cSGMCMC), Bayesian linear regression (BLR) on the final layer, and BLR ensembles. We propose using the variance prediction of the underlying NN to allow heteroscedastic variance estimation with BLR. The methods' uncertainty estimates are visually compared on toy 1-D and 2-D regression datasets, and quantitatively compared on standard regression datasets in terms of predictive root mean square error and log-likelihood. We also evaluate the methods' uncertainty estimates for out-of-distribution inputs.

Finally, we compare the downstream RL performance of PE, BLR, and BLR ensembles when used as models in a recent model-based RL algorithm. Our results indicate that BLR is competitive with ensembles. Furthermore, BLR ensembles may outperform ensembles, although further research is needed.

Keywords: uncertainty-aware, model-based, exploration, reinforcement learning, Bayesian neural network, Bayesian linear regression, MC-dropout, probabilistic ensemble, cSGMCMC.

Acknowledgements

First of all, I would like to thank my supervisor Lennart Svensson for his continual support, enthusiasm and advice throughout the project. I would also like to thank Juliano Pinto and Henk Wymeersch for being a part of our regular research meetings and exposing me to many interesting discussions and ideas.

Finally, I want to thank my family and friends for their everlasting support. In particular, I am grateful to Joel Sjögren for his enthusiastic assistance whenever my tensors grew too large.

John Moberg, Gothenburg, August 2019

Contents

List of Abbreviations **xi**

List of Figures **xiii**

List of Tables **xv**

1 Introduction **1**
1.1 Objective and method . . . 2
1.2 Limitations . . . 2
1.3 Outline and contributions . . . 3

2 Background **5**
2.1 Reinforcement learning . . . 5
2.1.1 Popular introduction . . . 5
2.1.2 Formalizing RL . . . 6
2.2 Types of uncertainty . . . 7
2.3 Artificial neural networks . . . 8
2.3.1 Artificial neural networks . . . 8
2.3.2 Activation functions . . . 8
2.3.3 Loss functions and probabilistic NNs . . . 9
2.3.4 Optimization . . . 10
2.3.5 Regularization . . . 10
2.3.6 Software libraries . . . 11
2.4 Bayesian neural networks . . . 11
2.4.1 The Bayesian framework . . . 11
2.4.2 Bayesian linear regression . . . 12
2.4.3 Bayesian neural networks . . . 13
2.5 Gaussian processes . . . 14
2.5.1 Gaussian processes . . . 14
2.5.2 Relation to Bayesian neural networks . . . 15

3 Uncertainty-aware Neural Networks **17**
3.1 Monte Carlo dropout (MC-dropout) . . . 17
3.2 Probabilistic Ensembles (PE) . . . 18
3.3 Cyclical Stochastic Gradient MCMC (cSGMCMC) . . . 19
3.3.1 Technical details . . . 20
3.4 Deep Bayesian Linear Regression (Deep BLR) . . . 21

3.4.1 Technical details . . . 22
3.4.2 Deep BLR ensemble . . . 22
3.4.3 Implementation details . . . 23
3.4.4 The limited applicability of Deep BLR . . . 23
3.5 Discussion: Computational footprint . . . 24
3.6 Intermission . . . 24
3.7 Survey of other methods . . . 25

4 Evaluation in Supervised Learning 27
4.1 Toy regression . . . 27
4.1.1 One-dimensional regression . . . 28
4.1.2 Two-dimensional regression . . . 31
4.1.3 Discussion . . . 32
4.2 *UCI* regression . . . 34
4.2.1 Experimental setup . . . 34
4.2.2 Training and hyperparameters . . . 34
4.2.3 Results . . . 35
4.2.4 Conclusions and further work . . . 37
4.3 Out-of-distribution inputs . . . 38
4.3.1 Simulated data and experimental setup . . . 38
4.3.2 Methods and hyperparameters . . . 39
4.3.3 Results . . . 39
4.3.4 Conclusions . . . 39

5 Evaluation in Reinforcement Learning 41
5.1 Deep BLR in PETS . . . 41
5.1.1 The algorithm . . . 42
5.1.2 Experimental details . . . 43
5.1.3 Results . . . 44
5.1.4 Discussion and further work . . . 44

6 Conclusions and Further Research 47
6.1 Conclusions . . . 47
6.2 Further research . . . 48

Bibliography 49

A Experimental details I
A.1 Toy regression . . . I
A.2 Trick for bounded variance in PETS . . . I

List of Abbreviations

- **BLR** — Bayesian linear regression.
- **BNN** — Bayesian neural network.
- **cSGMCMC** — Cyclical Stochastic Gradient MCMC (Zhang et al., 2019).
- **GP** — Gaussian process.
- **HMC** — Hamiltonian Monte Carlo.
- **KL divergence** — Kullback-Leibler divergence.
- **MBRL** — Model-based reinforcement learning.
- **MCMC** — Markov chain Monte Carlo.
- **MSE** — Mean-squared error.
- **NLL** — Negative log-likelihood.
- **NN** — Neural network.
- **PE** — Probabilistic ensemble (Lakshminarayanan et al., 2017).
- **ReLU** — Rectified linear unit.
- **RL** — Reinforcement learning.
- **RMSE** — Root-mean-square error.
- **SGD** — Stochastic gradient descent.

List of Figures

2.1 Plots of the Sigmoid, ReLU, and Swish activation functions discussed in Section 2.3.2. 9

2.2 The resulting posterior predictive distribution after applying Bayesian linear regression to a 1-D regression problem. The gray lines are generated by sampling from the posterior for $\mathbf{w}$. 13

2.3 Demonstration of Gaussian process regression with mean zero and a squared exponential kernel with $\ell = 1.5$ and $\sigma = 1$. The light-red region is a 95% confidence interval. 15

3.1 Illustration of the cyclical stepsize schedule used in cSGMCMC (red) and the traditional decreasing stepsize schedule (blue) in SGMCMC algorithms. In this case, $\beta = 0.5$. Figure from Zhang et al. (2019). . . 20

3.2 High-level illustration of Deep BLR as described in Section 3.4. A probabilistic NN is trained by maximzing the normal log-likelihood. The representation $\phi(x)$, extracted from the NN before the linear output layer, and the predicted variance $\sigma^2(x)$ are then used in Bayesian linear regression, producing posterior and posterior predictive distributions with closed-form expressions. 21

4.1 Approximate posterior predictive distributions on **D1** for each method. Each method outputs a Gaussian mixture which we approximate with a single Gaussian distribution. The shaded area covers two standard deviations of that approximate posterior predictive distribution. See A.1 for the full set of hyperparameters. 29

4.2 Approximate posterior predictive distributions on **D2** for each method, including a reference distribution in 4.2a. Each method outputs a Gaussian mixture which we approximate with a single Gaussian distribution. The shaded area covers two standard deviations of that approximate posterior predictive distribution. See A.1 for the full set of hyperparameters. 30

4.3 Effect of the prior weight variance g on uncertainty estimates produced by Deep BLR. It is clear that higher prior variance leads to higher predictive uncertainty. Best viewed on a computer. 31

4.4 Effect of ensemble size on uncertainty estimates on **D2** for the ensemble methods. On this simple problem, Probabilistic ensembles perform well with only 3 NNs. Deep BLR performs well with a single NN, and it is unclear if the computational trade-off is worth ensembling in this case. Best viewed on a computer. 31
4.5 Approximate posterior predictive distributions on the two-dimensional dataset **D3** for each method. Each method outputs a Gaussian mixture which we approximate with a single Gaussian distribution. The heatmap is constructed from the normalized variance of this distribution. See A.1 for the full set of hyperparameters. 33
4.6 Illustration of the experimental setup used in Section 4.3. The figure depicts the dataset (with reduced dimension) with the two test points and the line segment highlighted. Dimensionality reduction was done using the nonlinear dimensionality reduction algorithm t-SNE (van der Maaten and Hinton, 2008). 38
4.7 Uncertainty estimates along the line segment between two points in the training set, as described in Section 4.3, for different methods for obtaining uncertainty-aware NNs. The uncertainty is represented by the standard deviation of the predictive distributions. For each point, the distance to the closest point in the training set is plotted. 40

5.1 The two environments used in our evaluation. In Cartpole, the goal is to swing up the pole by controlling the horizontal force applied to the cart. The observation is 4-dimensional and contains the position and velocity of the cart and pole. In Reacher, the goal is to control the robot arm to touch the ball. The action and observation spaces are 7- (the robot has 7 degrees of freedom) and 17-dimensional respectively. 43
5.2 Comparison of 1 probabilistic NN, 5 probabilistic NNs, Deep BLR, and Deep BLR ensemble when used as an uncertainty-aware model for PETS on CartPole. The experiment is repeated 10 times and the mean return for each of the 15 episodes is reported, with the shaded area representing one standard error. 45
5.3 Comparison of 1 probabilistic NN, 5 probabilistic NNs, Deep BLR, and Deep BLR ensemble when used as an uncertainty-aware model for PETS on Reacher. The experiment is repeated 10 times and the mean return for each of the 30 episodes is reported, with the shaded area representing one standard error. 45

List of Tables

4.1 Comparison of MC-dropout, Random ensemble, cSGMCMC ensemble, Deep BLR, and Deep BLR ensemble on a set of standard datasets. We report the mean and standard error of predictive RMSE and NLL across 20 random training-test splits (5 for Protein Structure, 1 for Year Prediction MSD). We rank the methods for each data set (up to a standard error) and sum the ranks to quantify each method's overall performance. 36

4.2 Comparison of MC-dropout with deterministic (minimizing MSE) and probabilistic (minimizing NLL) NNs performed as in 4.1. Results for D-Dropout obtained from Gal and Ghahramani (2016). . . . 36

1

Introduction

Reinforcement learning (RL) has seen great successes in recent years, revitalized with the advent of deep learning. By using neural networks (NNs) as powerful value function approximators, model-free RL algorithms have succeeded at learning a variety of complex tasks. These tasks include continuous and discrete domains like humanoid walking (Schulman et al., 2017) and Atari games (Mnih et al., 2013) respectively, and recently even highly complex video games like Dota 2 (OpenAI, 2019) and StarCraft 2 (DeepMind, 2019). However, solving these problems has required massive amounts of computation and millions of interactions with the true environment. Learning Atari requires millions of frames, and learning to play Dota 2 required weeks of training on 128,000 CPU cores. In the real world, the number of samples required by many contemporary RL algorithms is infeasible and more efficient learning is necessary.

Model-based RL (MBRL) promises increased sample efficiency, but learning a sufficiently accurate model of a complex environment is difficult, and a slightly biased model can lead to seriously erroneous conclusions about optimal behaviour. To date, MBRL hasn't seen the wide array of successes that model-free RL (MFRL) has, but recent work (Chua et al., 2018; Kurutach et al., 2018) indicates that using ensembles of models, inducing uncertainty in the dynamics, can produce MBRL algorithms that perform as well as MFRL algorithms but learn much faster.

Everything boils down to the ability to trust the model. If we are going to base our decisions on extensive planning in our model, it must know what it doesn't know, i.e., keep track of its uncertainty. A common distinction is between *epistemic* and *aleatoric* uncertainty. Epistemic uncertainty is uncertainty resulting from a lack of knowledge about the world, and thus uncertainty that could be eliminated by obtaining more information. Aleatoric uncertainty is uncertainty that is inherent in the environment, e.g., actual stochastic transitions or noisy sensors. While the aleatoric uncertainty isn't at all unimportant, it is the epistemic uncertainty we are most interested in, since that is what is open to improvement. In short, we want a model that is aware of its epistemic uncertainty.

Using uncertainty-aware neural networks to represent approximate dynamics models can enable efficient and safe model-based learning in complex real-world environments. The posterior distribution over models, representing plausible realities, can be used by the agent to learn a policy that is safe and robust with regard to our

often crude knowledge of reality. Beyond better and more efficient learning, uncertainty can be used for efficient exploration (Shyam et al., 2019), and may prove to be of crucial importance for safety. In safety-critical applications such as autonomous driving, we need to know our limitations in order to act safely, and uncertainty quantifies this.

In recent years, many methods to estimate the uncertainty of deep NNs have been proposed, but so far a winner is far from crowned. More research is necessary to understand these methods and the properties of their uncertainty estimates.

1.1 Objective and method

For this thesis, we set out to identify and evaluate state-of-the-art methods for producing uncertainty-aware NNs that can be used in model-based RL, and if possible, improve on them.

In this setting, we desire methods that

- produce reliable uncertainty estimates that encompass both epistemic and heteroscedastic (input-dependent) aleatoric uncertainty, and
- scale to large number of samples and high-dimensional inputs and outputs.

To this end, we begin by investigating the methods' uncertainty estimates on a set of toy examples which highlight our requirements. The methods' predictive performance (which includes the ability to handle uncertainty) are then evaluated on a set of standard regression datasets. Finally, the downstream capability of the methods for producing uncertainty-aware models is evaluated by using them as components in a model-based RL algorithm.

1.2 Limitations

With such a broad objective, we have to make some significant limitations. First of all, we restrict ourselves to the regression case. While classification-type models can be useful in discrete environments, we are primarily interested in the continuous case.

Secondly, we don't concern ourselves with uncertainty calibration (e.g., ensuring that a 90% credible interval contains the true outcome 90% of the time) and instead view uncertainty as relative. However, calibration methods such as the one proposed by Kuleshov et al. (2018) can be applied to any of our uncertainty-aware models.

Our greatest limitation is that we cannot compare *all* promising methods for uncertainty estimation and had to restrict ourselves to an interesting and reasonable

subset. Some of the methods we don't cover are discussed in Section 3.7 and particularly promising ones are highlighted. Of course, hindsight is 20/20.

Finally, there is a vast literature on Bayesian reinforcement learning (Ghavamzadeh et al., 2015) which is theoretically very attractive, but unfortunately intractable in general due to the exponential size of the belief space. While this field is closely related to and very relevant for our work, we decided not to cover it due to time constraints. However, research is ongoing and recent work by, e.g., Lee et al. (2018a) looks promising.

1.3 Outline and contributions

In Chapter 2 we provide the reader with background necessary to understand the rest of the thesis. We introduce concepts such as epistemic and aleatoric uncertainty, reinforcement learning, neural networks, and Bayesian neural networks.

Chapter 3 describes the five methods for uncertainty estimation that are considered in the thesis. The chapter ends with a brief survey of the many alternative methods that aren't covered in this work.

In Chapter 4, the chosen methods are evaluated on progressively more complex regression tasks, starting with 1-D functions which are easily visualized. This is followed by a comprehensive benchmark on a set of standard datasets. The chapter ends with a brief evaluation of the ability to detect out-of-distribution inputs.

Chapter 5 brings us back to our goal of using uncertainty-aware models in RL. A selection of uncertainty-aware NNs are evaluated based on their downstream performance in some continuous environments when used as a component in the uncertainty-aware MBRL algorithm PETS (Chua et al., 2018).

In Chapter 6, we conclude the thesis, discuss our findings, and suggest areas for further research.

Our main contribution is the comparison of Deep Bayesian linear regression (Deep BLR) for uncertainty estimation with some alternatives in the literature. We also propose a slight modification that lets Deep BLR predict heteroscedastic aleatoric uncertainty, which, to the best of our knowledge, is novel. The method is shown to be competitive with the ensemble method that is prominent in recently proposed uncertainty-aware model-based RL algorithms. Furthermore, our results indicate that ensembles of Deep BLR may outperform probabilistic ensembles, although more research is needed.

2

Background

This chapter introduces some central concepts underlying the thesis, such as that of reinforcement learning and uncertainty. We also provide some background theory, covering the foundations of artificial neural networks, Bayesian inference, Bayesian neural networks, and Gaussian processes. Our treatment will by necessity be brief, but we provide references to relevant literature for the interested reader.

2.1 Reinforcement learning

While this thesis doesn't delve into any details of RL algorithms, producing reliable uncertainty-aware models that can be used for efficient RL is our ultimate objective. Chapter 5 is dedicated to comparing the performance of some different uncertainty-aware models when used as a component in a RL algorithm. This section gives a brief introduction to the subject to provide the reader with some context. For a more in-depth treatment, we recommend the book by Sutton and Barto (2018).

2.1.1 Popular introduction

Reinforcement learning is about constructing algorithms that produce agents which take actions to maximize the cumulative reward (referred to as return) in some environment. At each timestep, the agent receives an observation, executes an action, and gets a reward (which may be zero, or even negative.) As time progresses, the agent should learn which actions lead to a high return.

A central concept is the exploration-exploitation dilemma: should you use the strategy you currently believe to be the best, or should you explore the environment to potentially find an even better strategy? This is the trade-off we face when choosing food at a restaurant — do you pick a dish you're familiar with or do you risk it and try something new? A common approach is to use an ϵ-greedy strategy where you act greedily most of the time, but choose a random action with probability $\epsilon > 0$. Usually, ϵ is decreased from 1 to e.g. 0.01 during the course of training.

Note that RL is very general in the sense that many problems can be formulated

as an environment where it can be applied. Some examples are robotics, chess, autonomous driving, data center cooling, and of course, video games. The reward function can either be hand-crafted to simplify learning, or binary, e.g., ± 1 for win/loss in Chess. For "real-life" problems, the difficulty often lies in formulating a reward function that (1) doesn't make learning too hard, and (2) doesn't result in undesired behaviour (e.g., solving a game by finding a software bug (Lehman et al., 2018).)

The resurgence of neural networks had led to great success in RL, creating the subfield of Deep RL. The most prominent success stories are playing Atari (DQN, (Mnih et al., 2013)), Go and Chess (AlphaGo Zero, (Silver et al., 2017), Dota 2 and robotic hand manipulation (Rapid, distributed PPO (OpenAI, 2019)), and StarCraft 2 (DeepMind, 2019). While impressive feats, a common denominator is that all require massive compute, e.g., 128,000 CPUs and 256 GPUs in the case of Dota 2. This is not only expensive, but also potentially infeasible for many real-world tasks we are interested in.

Model-based RL, in which a model of the environment is explicitly learned and used for planning, has shown promise of increased sample efficiency, but has not had the same success as model-free RL in complex environments. Uncertainty-aware models may be one step on the way to successful, sample-efficient model-based RL, by utilizing uncertainty about the environment intelligently. In this context, sample efficiency refers to reducing the number of interactions with the true environment, which may be expensive or time-consuming in real systems.

2.1.2 Formalizing RL

RL is usually formalized in the context of Markov decision processes (MDPs). A MDP is a 5-tuple $(\mathcal{S}, \mathcal{A}, T, r, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ are sets of states and actions, $T : \mathcal{S} \times \mathcal{A} \to P(\mathcal{S})$ is the transition function, $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, and $\gamma \in [0, 1]$ is the discount factor. Each episode starts at some initial state $s_0 \in \mathcal{S}$ (which may be fix or follow some distribution). After observing s_0, an action $a_0 \in \mathcal{A}$ is executed according to some policy $\pi : \mathcal{S} \to P(\mathcal{A})$, leading to the next state $s_1 \sim T(s_0, a_0)$ and a reward $r(s_0, a_0, s_1)$. This continues for N timesteps for some $N \in \mathbb{N}$, where N is the task horizon. Note that T only depends on the current state, explaining the Markov in MDP.

The objective is to learn a policy π that maximizes the expected discounted return

$$\mathop{\mathbb{E}}_{\substack{a_{k+1}\sim\pi(s_k) \\ s_{k+1}\sim T(s_k,a_k)}} \left[\sum_{k=0}^{N-1} \gamma^k r(s_k, a_k, s_{k+1}) \right]$$

when following the policy.

In the model-based RL setting, we aim to learn the transition function T from our interactions with the environment. This reduces to a supervised learning problem:

given the input (s_t, a_t), predict s_{t+1}. It is common to assume that the reward function is known, and we do so in this thesis. Given a perfect model of T, it is comparatively easy to learn an optimal policy. Most importantly, it can be done without additional interaction with the true environment, which may be expensive or unsafe. Model-based RL may thus be able to learn in much fewer samples than model-free methods which are notoriously inefficient.

2.2 Types of uncertainty

In this section, the notions of aleatoric and epistemic uncertainty are introduced as a decomposition of uncertainty that can be important to understand and utilize uncertainty efficiently. We begin with an intuitive description of these uncertainties and how we would expect them to behave, and then try our best to formalize these notions.

Aleatoric uncertainty is irreducible uncertainty that is inherent to the output, e.g. normal sensor noise or throwing a die. Even with infinite data, we cannot expect to predict an outcome accurately in the presence of aleatoric uncertainty. Epistemic uncertainty is subjective uncertainty that is caused by our lack of information. With enough data, we would expect to be able to remove the epistemic uncertainty.

The ability to separate aleatoric and epistemic uncertainty can be very helpful in applications. For a concrete example, a RL agent that attempts to learn by taking actions to minimize its future uncertainty (i.e. some sort of curiosity) may end up repeatedly taking actions with high aleatoric uncertainty, thus learning nothing of value. Minimizing the epistemic uncertainty would clearly be much more reasonable in this scenario.

To formalize these notions, we follow the work of Depeweg et al. (2018). The total uncertainty of some real-valued random variable $\mathbf{Y}$ is given by its variance. By the law of total variance, we can decompose this as

$$\underbrace{\mathrm{Var}(\mathbf{Y})}_{\text{total uncertainty}} = \underbrace{\mathrm{Var}(\mathbb{E}[\mathbf{Y} \mid \mathbf{W}])}_{\text{epistemic uncertainty}} + \underbrace{\mathbb{E}[\mathrm{Var}(\mathbf{Y} \mid \mathbf{W})]}_{\text{aleatoric uncertainty}}$$

where the expectations are taken over the posterior of the parameter vector $\mathbf{W}$. As we gather more data, the posterior for $\mathbf{W}$ will concentrate and the variance will go to zero, so the epistemic uncertainty goes away as we want. Simultaneously, the second term will approach the variance of $\mathbf{Y}$ as expected. For many of our models, we are able to approximate the terms by Monte Carlo integration.

Another important distinction is between homoscedastic and heteroscedastic variance (inducing aleatoric uncertainty). A model with homoscedastic variance assumes equal variance σ^2 for all inputs, while heteroscedastic variance permits input-dependent variance $\sigma^2(\mathbf{x})$. While homoscedasticity can be a reasonable assumption in many cases, it can also be entirely unreasonable, especially in the context of

model-based RL where stochasticity may be present only in some states. For this reason, the capability of uncertainty-aware NNs to model heteroscedasticity is important.

2.3 Artificial neural networks

Artificial neural networks (NNs) form the backbone of the field called deep learning, which is a fuzzy term that refers to NNs with "many" layers. For a comprehensive introduction to deep learning, we point the reader to the book by Goodfellow et al. (2016). In this chapter, we briefly introduce the concepts necessary for the thesis.

2.3.1 Artificial neural networks

A neural network consists of an input layer, an output layer, and some number of hidden layers in between. Each input is successively transformed to some output by passing through each layer, finally producing an output. A layer consists of some number of units (also called neurons), each of which produces a scalar as a linear combination of its inputs and applies a nonlinear differentiable activation function $\sigma(\cdot)$ to it.

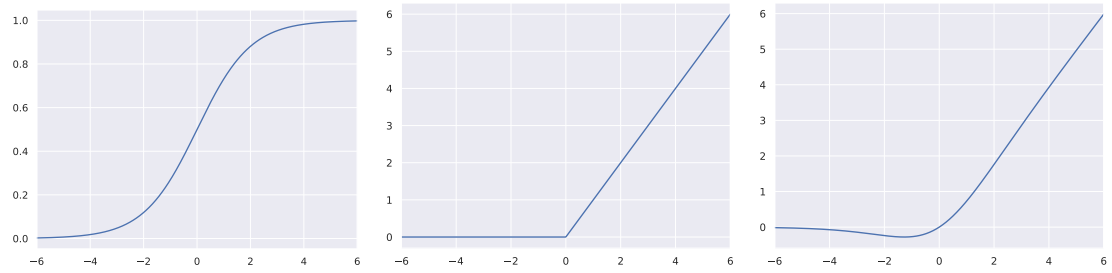
More precisely, given an input $\mathbf{x} \in \mathbb{R}^n$, the output of a unit is given by $\sigma(\mathbf{w}^T\mathbf{x})$ for some weights $\mathbf{w} \in \mathbb{R}^n$ that are specific to that unit. Extending this to a whole layer consisting of p units, the layer will output $\mathbf{o}(\mathbf{x}) \in \mathbb{R}^p$, where each $\mathbf{o}_i = \sigma(\mathbf{w}_i^T\mathbf{x})$. The resulting $\mathbf{o}(\mathbf{x})$ is then fed as input to the next layer. The structure of the output layer will depend on the desired output, but in regression a linear output layer is commonly used, i.e., without a nonlinear transformation.

Once the architecture is set, the NN is trained by minimizing a suitable loss function, e.g., mean-squared error for regression. The next few sections cover loss functions, optimization, and regularization.

2.3.2 Activation functions

There are many activation functions to choose from, and the choice is often based on empirical results rather than theory. A common choice is the rectified linear unit, ReLU, given by $\text{ReLU}(x) = \max(x, 0)$, and this is what will be used for most NNs throughout the thesis. A less common choice is the sigmoid, given by $\text{Sigmoid}(x) = \frac{1}{1+e^{-x}}$. In Section 5.1 we follow prior work and use the fairly recent Swish activation function (Ramachandran et al., 2017):

$$\text{Swish}(x) = x \cdot \text{Sigmoid}(x).$$



(a) Sigmoid. **(b)** ReLU. **(c)** Swish.

Figure 2.1: Plots of the Sigmoid, ReLU, and Swish activation functions discussed in Section 2.3.2.

It can be seen as a smooth version of ReLU that doesn't become identically zero for $x \leq 0$ which may aid learning by preventing units from "dying". Figure 2.1 contains plots of the three activation functions described.

2.3.3 Loss functions and probabilistic NNs

As is common in machine learning, learning to model a problem with a NN is viewed as an optimization problem where some loss function is minimized. For regression tasks, the most common loss function is the mean-squared error (MSE) loss which, given some targets $\mathbf{y} = (y_1, \ldots, y_N)$ and a NN outputting some point estimates $\hat{\mathbf{y}} = (\hat{y}_1, \ldots, \hat{y}_N)$ is defined by

$$\text{MSE}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2.$$

However, this approach cannot take aleatoric uncertainty into account and the loss will only be able to converge to the mode of the true output distribution, giving the maximum a posteriori (MAP) estimate.

A simple way to obtain a NN that estimates aleatoric uncertainty is to let it output the mean and standard deviation $(\hat{\mu}, \hat{\sigma})$ of a normal distribution rather than just $\hat{\mu}$. An appropriate loss function is then the negative log-likelihood of the normal distribution, which is easily found to be

$$\text{NLL}(\mathbf{y}, (\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\sigma}})) = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{1}{2} \log \hat{\sigma}_i^2 + (y_i - \hat{\mu}_i)^2 / (2\hat{\sigma}_i^2) \right).$$

Note the resemblance to the MSE. We call a NN trained in this fashion a probabilistic NN. To ensure positivity of $\hat{\sigma}$, the NN is parameterized to output $\log \hat{\sigma}_i$ which is subsequently transformed to $\hat{\sigma}_i$ with $\exp(\cdot)$. Of course, the NN can parameterize any distribution, but we will stick with normal distributions for simplicity.

2.3.4 Optimization

As a composition of differentiable functions, a NN is differentiable. Given a differentiable loss function, the derivative of the loss can be efficiently computed with regards to each weight in the NN using the chain rule and memoization. Modern deep learning libraries provide automatic differentiation which makes this straightforward.

Given the gradient containing the derivatives of the loss function with respect to each weight, gradient descent methods can be applied to minimize the loss. For large data sets, it is infeasible to compute the gradient using the entire data set, which gives rise to the idea of stochastic gradient descent[1] (SGD) in which only a small subset (a mini-batch) of the training examples is used for each step. Using small batches not only makes training tractable, but may also improve generalization (Keskar et al., 2017). NNs are trained for some number of epochs, where an epoch refers to the whole dataset having been used in batches for optimization. For this reason, it can be a good idea to prefer small mini-batches (e.g., 32 rather than 512) even if large mini-batches makes for faster epochs.

There are many variants of SGD in use, but Adam (Kingma and Ba, 2015) is a common choice and all NNs in this thesis are trained with Adam, with the exception of cSGMCMC (to be introduced in Chapter 3) which uses a simpler variation. In short, Adam adapts the step size automatically and introduces something called momentum which avoids too erratic steps in the presence of noise.

2.3.5 Regularization

Overfitting, which means that the model adapts too closely to the particular data, can lead to poor generalization to unseen data. There is a range of regularization techniques that can be used to combat this, and in this section we cover some of the most basic ones: L_1, L_2, and dropout.

L_1 and L_2 regularization is one of the most common techniques in all of machine learning (e.g., corresponding to lasso and ridge regression for linear models.) It is done by adding a penalty on the norm of the weights to the loss, with the L_1 (absolute value) or L_2 (squared) norm respectively. The terms are weighted by some parameter $\lambda > 0$, so the regularization term for L_2 will look like $\lambda \sum_i w_i^2$, where w_i denotes the weights of the NN. The effect is that the weights are encouraged to be small, resulting in a "simpler" model that may generalize better.

Dropout (Srivastava et al., 2014) is another common technique to prevent overfitting and improve generalization. It means that, with some probability $p \in (0, 1)$, the output of a unit is set to zero. It can be applied to some layer or all layers. Dropout

[1] Strictly speaking, SGD refers to computing the gradient using only a single example, and the algorithm described is called mini-batch gradient descent. Still, the family of methods is most often referred to as just SGD.

discourages weights from coadapting to the data, and intuitively creates a more robust NN that is forced to utilize all information optimally. In effect, utilizing dropout during training amounts to concurrently training an ensemble of NNs that have different architectures and adapt to the data in different ways. Note that dropout is usually turned off when the NN is used for inference, but as we will see, it is possible to use dropout during test time to obtain uncertainty estimates.

2.3.6 Software libraries

We use the deep learning library PyTorch (Paszke et al.) in Python for implementation of all methods in this thesis.

2.4 Bayesian neural networks

Bayesian neural networks (BNNs, (Neal, 1996)) constitute the gold standard of uncertainty-aware neural networks. Unfortunately, exact inference is intractable even for NNs considered small by today's standards. However, research into approximate inference for BNNs is ongoing and many methods for uncertainty estimation frame it as attempting to approximate BNNs. In this section, we introduce BNNs, starting with an introduction to the Bayesian framework.

2.4.1 The Bayesian framework

The framework of Bayesian inference provides a systematic approach to statistical inference and dealing with uncertainty. Given some prior distribution $p(\theta)$ on the sought parameter θ, encoding prior knowledge, and a likelihood function $p(x \mid \theta)$ which describes how the observed data $x \in \mathbf{X}$ is generated, Bayes' rule tells us how to compute the posterior distribution over the parameter space, for $\theta \in \Theta$:

$$p(\theta \mid \mathbf{X}) = \frac{p(\mathbf{X} \mid \theta)p(\theta)}{p(\mathbf{X})}.$$

The posterior distribution represents our belief about plausible parameters after observing the data. To make predictions for a test point x', the posterior predictive distribution is used. It is obtained by averaging over the space of all possible parameters $\theta \in \Theta$:

$$p(x' \mid \mathbf{X}) = \int_{\Theta} p(x' \mid \theta)p(\theta \mid \mathbf{X})\, d\theta.$$

In cases where we cannot perform the integration analytically, we can use Monte Carlo integration wherein we sample $\theta_i \sim p(\theta \mid \mathbf{X})$ for $i = 1, 2, \ldots, N$ and approximate

$$p(x' \mid \mathbf{X}) \approx \frac{1}{N} \sum_{i=1}^{N} p(x' \mid \theta_i).$$

Unfortunately, life is not as easy as it may seem. We glossed over an important part: computing $p(\mathbf{X})$ (which occurs in the denominator of the posterior) analytically is often difficult or impossible, even for simple-looking models. Unless we restrict ourselves and design our model to have a simple posterior, we must resort to methods for approximate inference.

Two common approaches are variational inference (VI) and Markov chain Monte Carlo (MCMC). In VI, we postulate an analytical form for the posterior and use optimization techniques to get it as close as possible to the true posterior, e.g., by minimizing the Kullback-Leibler divergence between the distributions. In MCMC, a Markov chain is cleverly constructed so that it has the posterior as its stationary distribution, allowing us to sample from the exact posterior after a warmup stage. However, both methods have their drawbacks: VI might result in a poor approximation to the often very complex posterior, and MCMC doesn't scale to large datasets or models, just to name a few.

2.4.2 Bayesian linear regression

In this section, we introduce Bayesian linear regression as a concrete example of Bayesian inference that also will be used later as a component of an attempt to construct an uncertainty-aware neural network.

Assume that we have observed N real-valued outputs y_i, each with an associated input $\mathbf{x}_i \in \mathbb{R}^p$, giving us the data $\mathbf{y} \in \mathbb{R}^N$ and $\mathbf{X} \in \mathbb{R}^{N \times p}$. For simplicity, suppose the data is centered to have mean zero.

Now, we posit a normal likelihood with known variance σ^2. This amounts to assuming the output y_i is produced as a linear transformation of $\mathbf{x}_i$ with an additional noise term distributed as $\mathcal{N}(0, \sigma^2)$. Thus our likelihood is

$$p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}, \sigma^2) = \mathcal{N}(\mathbf{y} \mid \mathbf{X}\mathbf{w}, \sigma^2 \mathbf{I}_N)$$

It remains to define a prior for $\mathbf{w}$. If we choose a normal prior, the posterior distribution will also be normal and simple to compute (we say that a normal prior is *conjugate prior* to a normal likelihood.) So let

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w} \mid \mathbf{w}_0, \mathbf{V}_0)$$

for some prior mean $\mathbf{w}_0 \in \mathbb{R}^p$ and covariance $\mathbf{V}_0 \in \mathbb{R}^{p \times p}$, e.g., $\mathbf{w}_0 = \mathbf{0}_p$ and $\mathbf{V}_0 = \mathbf{I}_p$ if we don't have any helpful information to encode.

Applying Bayes' rule to compute the posterior (Murphy, 2012), we find

$$p(\mathbf{w} \mid \mathbf{X}, \mathbf{y}, \sigma^2) = \mathcal{N}(\mathbf{w} \mid \mathbf{w}_N, \mathbf{V}_N),$$

where the posterior mean and covariance are given by

$$\begin{aligned} \mathbf{w}_N &= \mathbf{V}_N \mathbf{V}_0^{-1} \mathbf{w}_0 + \frac{1}{\sigma^2} \mathbf{V}_N \mathbf{X}^T \mathbf{y}, \text{ and} \\ \mathbf{V}_N &= \sigma^2 (\sigma^2 \mathbf{V}_0^{-1} + \mathbf{X}^T \mathbf{X})^{-1}. \end{aligned}$$

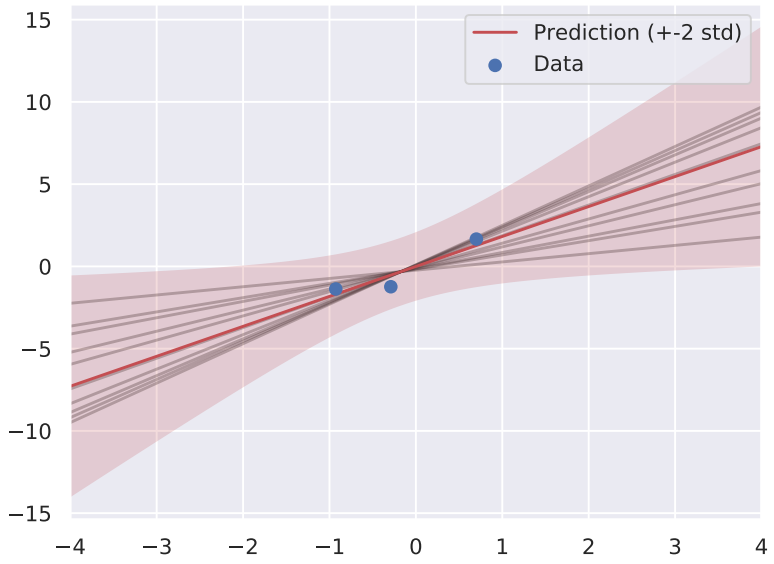


Figure 2.2: The resulting posterior predictive distribution after applying Bayesian linear regression to a 1-D regression problem. The gray lines are generated by sampling from the posterior for $\mathbf{w}$.

Furthermore, the posterior predictive distribution will also be normal:

$$p(y \mid \mathbf{x}, \mathcal{D}, \sigma^2) = \mathcal{N}(y \mid \mathbf{w}_N^T \mathbf{x}, \sigma^2 + \mathbf{x}^T \mathbf{V}_N \mathbf{x}).$$

Figure 2.2 shows the posterior predictive distribution for a simple example where $\sigma^2 = 1$. We see that the uncertainty is small where there is data and high otherwise. The gray lines are generated by sampling $\mathbf{w}$ from the posterior, producing some plausible models explaining the data.

2.4.3 Bayesian neural networks

Applying the Bayesian framework to neural networks amounts to defining a likelihood function and putting a prior distribution on the weights. For example, for a regression problem where we want to predict y given $\mathbf{x}$, we may choose a normal likelihood

$$p(y \mid \mathbf{x}, \mathbf{w}, \sigma^2) = \mathcal{N}(y \mid \mathrm{NN}(\mathbf{x}; \mathbf{w}), \sigma^2),$$

where σ^2 is a known variance and NN is a neural network with weights $\mathbf{w}$. For a prior, we may assume the weights are independently standard normal distributed. Thus

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w} \mid \mathbf{0}, \mathbf{I}).$$

That's it — we have defined a Bayesian neural network! Inference is the tricky part. Obtaining the exact posterior is generally impossible (unless the NN is very simple, in which case MCMC methods may be able to give us samples from it), and even approximating it can be extraordinarily difficult. Further discussion of

contemporary methods for inference in BNNs is left to Section 3.7 where we survey the existing literature.

2.5 Gaussian processes

Gaussian processes are powerful tools for regression and classification that provide natural uncertainty estimates. Unfortunately, exact inference on Gaussian processes has cubical complexity in the number of samples and contemporary approximate inference is not much better (Hensman et al., 2013). This makes Gaussian process regression infeasible even for datasets considered modestly sized in the age of deep learning. However, we can use Gaussian processes as a target and a point of comparison in the cases where it can be used.

In this section, we provide a brief introduction to Gaussian processes. We also demonstrate some connections to Bayesian neural networks. For a much more in-depth treatment of theory and applications, see the classic book by Rasmussen and Williams (2006).

2.5.1 Gaussian processes

Formally, a Gaussian process (GP) is a stochastic process, i.e., a collection of random variables, such that any finite subcollection of them have a multivariate normal distribution. A GP is determined by its mean and covariance functions

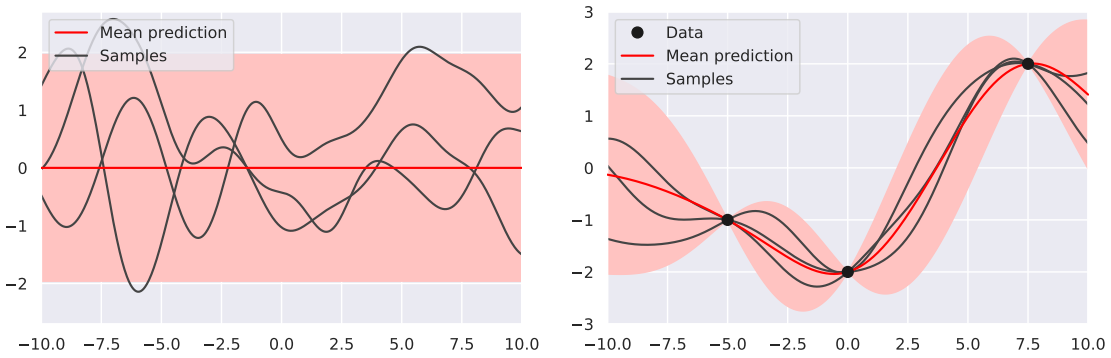
$$\begin{aligned} m(\boldsymbol{x}) &= \mathbb{E}[f(\boldsymbol{x})] \\ k(\boldsymbol{x}, \boldsymbol{x}') &= \mathrm{Cov}(f(\boldsymbol{x}), f(\boldsymbol{x}')) \end{aligned}$$

for $\boldsymbol{x} \in \mathcal{X}$ where $\mathcal{X}$ is some index set, e.g. $\mathcal{X} = \mathbb{R}$. It is common to set $m(x) \equiv 0$, while the choice of kernel depends on the application and is usually where any prior knowledge (e.g., periodicity) is encoded. A simple example is the squared exponential kernel

$$k(\boldsymbol{x}, \boldsymbol{x}') = \sigma^2 \exp\left(-\frac{1}{2\ell^2}|\boldsymbol{x} - \boldsymbol{x}'|^2\right)$$

for which the covariance of the function values for close-by inputs is almost σ^2, while it decays rapidly to zero as the inputs get farther apart. Here ℓ and σ are parameters that must be chosen by the user.

Given some Gaussian process prior $f(\boldsymbol{x}) \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}'))$, we sample functions by selecting a collection of m test points at which we want to evaluate the function. This is done by sampling from the multivariate normal distribution induced by the GP over the function values at the m test points. Figure 2.3a shows three samples, i.e. three different function realizations, from the GP prior with a squared exponential kernel.



(a) Samples from the prior GP.

(b) The GP fits the data while maintaining uncertainty in regions without data.

Figure 2.3: Demonstration of Gaussian process regression with mean zero and a squared exponential kernel with $\ell = 1.5$ and $\sigma = 1$. The light-red region is a 95% confidence interval.

Suppose that we have some training data $\{(\boldsymbol{x}_i, f_i)\}_{i=1}^n$ to which we would like to fit the Gaussian process. This essentially means ensuring that all samples from the GP will coincide with the data. We do this by considering the joint distribution over training and test points and conditioning it on the observed data. Fortunately Gaussian distributions are nice and the resulting posterior distribution has a closed-form formula. Less fortunately it involves the inverse of the covariance matrix which has $(n+m)^2$ entries. Figure 2.3b displays the result of fitting a GP to three data points.

2.5.2 Relation to Bayesian neural networks

It can be shown that a single-layer Bayesian neural network with an i.i.d. prior converges to a Gaussian process as you let the number of neurons go to infinity (Neal, 1996). This is essentially done by noting that the output of each neuron is a sum of independent terms, and so will be normally distributed in the limit according to the central limit theorem from probability theory. This gives us a multivariate normal distribution over the output of all neurons, which is precisely a Gaussian process.

Lee et al. (2018b) extend this to deep BNNs by an induction argument and show that the covariance function for the corresponding GP can be efficiently computed given the prior on the BNN parameters. In the case of ReLU activations, the kernel corresponding to a single-layer BNN was analytically derived by Cho and Saul (2009), and so we can express the kernel analytically in the form of a recurrence

relation

$$K^l(x, x') = \sigma_b^2 + \frac{\sigma_w^2}{2\pi}\sqrt{K^{l-1}(x, x)K^{l-1}(x', x')}\left(\sin\theta_{x,x'}^{l-1} + (\pi - \theta_{x,x'}^{l-1})\cos\theta_{x,x'}^{l-1}\right)$$
$$\theta_{x,x'}^l = \arccos\left(\frac{K^l(x, x')}{\sqrt{K^l(x, x)K^l(x', x')}}\right),$$

which terminates with

$$K^0(x, x') = \sigma_b^2 + \sigma_w^2\left(\frac{x \cdot x'}{d_{in}}\right),$$

where σ_b^2 and σ_w^2 are the variances of the bias and weight priors for the corresponding layer.

To summarize, infinitely wide BNNs of arbitrary depth are equivalent to GPs with covariance functions that we can compute. In the case of ReLU activations, this can be done analytically. Many of the methods investigated in this thesis try to approximate the posterior of BNNs, and so these results motivate using ReLU GPs as a gold standard method.

3

Uncertainty-aware Neural Networks

The area of uncertainty-aware NNs is an active field with an abundance of proposed methods, where even the most prominent methods are fairly recent developments. For this thesis, we had to select a handful of methods we deemed the most interesting. The following methods are considered in this thesis:

- Monte-Carlo dropout (MC-droput, Gal and Ghahramani (2016)).
- Probabilistic ensembles (PE, Lakshminarayanan et al. (2017)).
- Cyclical stochastic gradient MCMC (cSGMCMC, Zhang et al. (2019)).
- Bayesian linear regression on the hidden layer representation. This method has appeared in multiple papers (Snoek et al., 2015; Azizzadenesheli et al., 2018; Riquelme et al., 2018) with no canonical name. We call it Deep BLR. Furthermore, we propose using the variance prediction of the underlying probabilistic NN to allow heteroscedastic variance estimation.
- Deep BLR ensembles, i.e., combining the predictive distributions of independently trained Deep BLR NNs.

The next few sections describe these methods in detail, followed by a brief discussion on their computational footprints. The chapter ends with a short survey of the field and describes some alternative methods that didn't fit in this thesis.

3.1 Monte Carlo dropout (MC-dropout)

Using dropout to obtain uncertainty estimates from NNs was first proposed by Gal and Ghahramani (2016). The NN is trained with dropout as usual, but instead of turning it off for inference, multiple stochastic forward passes are used to obtain the posterior predictive distribution. For this reason, it is called Monte Carlo dropout (MC-dropout). The Bernoulli dropout mask effectively samples from some posterior distribution over NN weights, resulting in an ensemble of NNs. The main appeal of

the method is perhaps how easy and comparatively cheap it is to apply, especially for NN architectures which already utilize dropout.

Gal and Ghahramani (2016) show that a NN with MC-dropout approximates a deep Gaussian process (Damianou and Lawrence, 2012) in the sense of minimizing the Kullback-Leibler divergence. The predictive distribution is approximated by matching the first two moments with the sample mean and variance over N stochastic forward passes for some sufficiently large N. The version of MC-dropout used in Gal and Ghahramani (2016) uses homoscedastic variance, but is easily extended to the heteroscedastic setting by employing a probabilistic NN and viewing the predictive distribution as a mixture of normals.

While MC-dropout is one of the most prominent methods for uncertainty-aware NNs, several papers indicate that ensembling (to be introduced in the next section) performs better on a range of tasks (Lakshminarayanan et al., 2017; Mcallister et al., 2018; Gustafsson et al., 2019). A weakness of the method is that the dropout rate p has to be tuned, usually with an expensive grid search. In particular, the dropout rate doesn't depend on the data which leads to strange behaviour, for example that the posterior is invariant to duplicates of the dataset (Osband et al., 2018). The tuning process may be problematic in the RL context, where more data is constantly added to the dataset. Despite the criticism, we choose to include MC-dropout since it is, as mentioned, a prominent method.

Gal et al. (2017) proposed Concrete Dropout as an improvement on Dropout, fixing the flaw discussed above by introducing automatic tuning of p. Our preliminary experiments indicated that the method is very sensitive to parameter choice, and even with careful hand-tuning good performance couldn't be attained on simple toy examples. It is possible that the method performs well on large and high-dimensional datasets, but since it doesn't pass a basic sanity check we chose not to continue investigating it.

3.2 Probabilistic Ensembles (PE)

A simple approach to uncertainty estimation is to train M probabilistic NNs on the same data, relying on the random initialization and stochasticity introduced in training to create diversity in the resulting models. As far as we know, this was introduced by Osband et al. (2016b) in the context of Deep Q-networks for reinforcement learning, and later elaborated on by Lakshminarayanan et al. (2017) in a more general context. Lakshminarayanan et al. (2017) refer to it as *Deep Ensembles*, but we opt to call it Probabilistic Ensembles (PE) as in other work.

Osband et al. (2016b) use a variation of Probabilistic Ensembles that gives a bootstrap sample of the data to each NN, i.e., sampling with replacement to obtain a subset of the data. Lakshminarayanan et al. (2017) observed that this deteriorated performance. Intuitively, it seems strange to potentially throw away samples if our

goal is to construct "an accurate posterior", and while this additional stochasticity may avoid overfitting and improve predictive performance, it appears unmotivated for our goal of estimating uncertainty. While research is too sparse for us to be able to reject it entirely, we don't consider it in this thesis.

Another variation adds randomized priors (Osband et al., 2018) and uses it for representing the value function in RL, leading to improved exploration and thus performance. The idea is to add a fixed random prior (represented by an identical NN) to the output and optimizing only the weights of the NN, keeping the prior fixed. While this appears to introduce long-term diversity useful in exploration, it isn't clear to us why this should improve performance in a more general setting. For these reasons, and for simplicity, we choose to not consider this variation.

While not considered in this thesis, Pearce et al. (2018) cast ensembling in a Bayesian light by adding a regularization term that "anchors" the weights to some prior. For wide NNs (50-100 units in the paper), this procedure supposedly results in a good approximation of the true posterior. Their experiments indicate that anchored ensembling performs very well on datasets with high epistemic uncertainty, outperforming PE significantly, but not so well in the presence of high aleatoric uncertainty. This is likely because it does not model heteroscedastic uncertainty. Nevertheless, this is an interesting research direction in that it may put ensembling on solid theoretical ground.

3.3 Cyclical Stochastic Gradient MCMC (cSGMCMC)

Stochastic Gradient MCMC (SGMCMC) constitutes a class of methods that attempt to handle the scaling issue of MCMC by framing it as an optimization problem and using mini-batches of data to compute the gradient. There are many variations, e.g., Stochastic Gradient Langevin Dynamics (SGLD, Welling and Yee Whye (2011)) and Stochastic Gradient Hamiltonian Monte Carlo (SGHMC, Chen et al. (2014)), with varying success. It turns out that there is a very elegant connection to SGD, in which the SGD iterates converge to the posterior distribution when the learning rate is appropriately annealed. For SGLD, this amounts to simply adding some noise to SGD, and for SGHMC, adding noise to SGD with momentum. An uncertainty-aware NN is then constructed as an ensemble of NNs with weights sampled along the trajectory of SGD in parameter space.

Zhang et al. (2019) recently proposed cyclical SGMCMC (cSGMCMC), which is a generalization of SGLD and SGHMC that incorporates a cyclical learning rate schedule. SGLD and SGHMC have trouble exploring multimodal distributions and end up sampling from around a single local minima of the loss function. The idea of cSGMCMC is that a cyclical learning rate helps the SGD iterates jump out of their local minima to explore another mode. Figure 3.1 compares the cSGMCMC stepsize schedule to the traditional decreasing stepsize. The training is divided up

into cycles, each of which begins with an exploration phase (analogous to the burn-in stage in MCMC) and ends with a sampling stage, in which the weights of the NN are sampled.

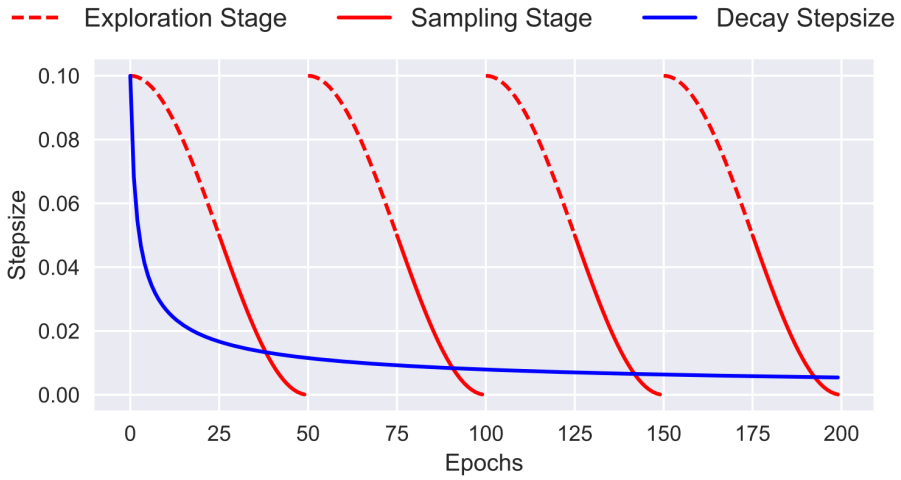


Figure 3.1: Illustration of the cyclical stepsize schedule used in cSGMCMC (red) and the traditional decreasing stepsize schedule (blue) in SGMCMC algorithms. In this case, $\beta = 0.5$. Figure from Zhang et al. (2019).

3.3.1 Technical details

In this section, we provide the necessary ingredients to implement cSGMCMC. The learning rate at iteration k is given by

$$\alpha_k = \frac{\alpha_0}{2}\left[\cos\left(\frac{\pi \bmod(k-1, \lceil K/M \rceil)}{\lceil K/M \rceil}\right) + 1\right],$$

where M is the number of cycles, K the number of total iterations, and α_0 the initial stepsize. The parameter $\beta \in (0, 1)$ controls how much of each cycle is dedicated to exploration, e.g., if $\beta = 0.9$, sampling begins after 90% of the iterations in each cycle. The algorithm uses SGD with momentum, where the final parameter $\alpha \in [0, 1]$ controls the momentum parameter which is given by $1 - \alpha$. During optimization, SGD with momentum is run as usual during the exploration stage, and noise is added during sampling to replicate traditional SGMCMC methods. In this sense, $\alpha = 1$ and $\alpha < 1$ correspond to SGLD and SGHMC respectively.

More precisely, a prior loss and a noise loss are added during the sampling stage. The prior loss is a regularization term that consists of the sum of squares of the NN weights scaled by $1/\sigma_0$ for some prior standard deviation $\sigma_0 > 0$. For the noise loss, each set of weights is scaled by $\epsilon \sim \mathcal{N}(0, \sigma_{\text{noise}})$, sampled independently for each set of weights, where $\sigma_{\text{noise}} = \sqrt{\frac{2\alpha}{\alpha_i}}$. The sum of scaled weights is then added as a penalty term to the loss.

In our implementation, we use a probabilistic NN with a negative log-likelihood loss to estimate aleatoric uncertainty.

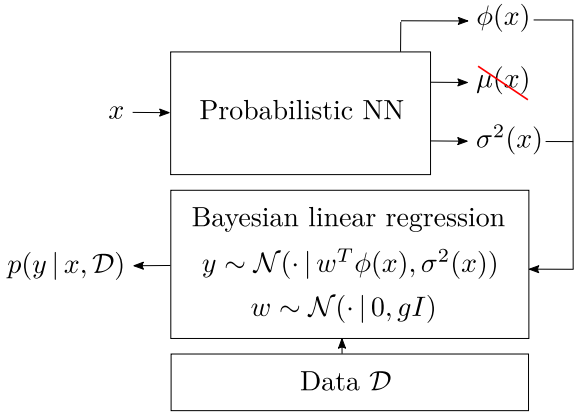


Figure 3.2: High-level illustration of Deep BLR as described in Section 3.4. A probabilistic NN is trained by maximzing the normal log-likelihood. The representation $\phi(x)$, extracted from the NN before the linear output layer, and the predicted variance $\sigma^2(x)$ are then used in Bayesian linear regression, producing posterior and posterior predictive distributions with closed-form expressions.

3.4 Deep Bayesian Linear Regression (Deep BLR)

Given a NN with a linear output layer, as is common for regression problems, a simple way to obtain uncertainty estimates is to do Bayesian linear regression (as shown in Section 2.4.2) on the NN's representation of the input data before the final layer. It is not obvious why this should yield good uncertainty estimates, but the prediction should at least not be worse than the maximum likelihood estimate.

This method was used successfully by Snoek et al. (2015) for Bayesian optimization, where Gaussian processes are traditionally used. Riquelme et al. (2018) evaluated its performance when used as a model for contextual bandits and find it to be competitive with other methods. Azizzadenesheli et al. (2018) apply the method to a Deep Q-network (DQN) and use the posterior for Thompson sampling, showing better performance than Bootstrapped DQN (which uses a bootstrap ensemble (Osband et al., 2016a)) on Atari. Riquelme et al. (2018) call this method *Neural Linear*. In this work, we refer to it as Deep Bayesian Linear Regression (Deep BLR for short) since we are doing BLR on a deep representation.

Unlike prior work, where the variance is either assumed known or homoscedastic (e.g., with an inverse-gamma prior), we incorporate heteroscedastic variance by using the variance prediction of the probabilistic NN. Thus our BLR model uses both a deep representation and the variance predicted in the final layer. This results in an efficient uncertainty-aware NN that can estimate both epistemic and heteroscedastic aleatoric uncertainty. Of course, it remains to see how reliable these estimates are. Figure 3.2 provides a high-level illustration of Deep BLR.

3.4.1 Technical details

Suppose that we begin with some dataset with N q-dimensional inputs stored in $\mathbf{X} \in \mathbb{R}^{N \times q}$ with corresponding one-dimensional and centered outputs $\mathbf{y} \in \mathbb{R}^N$. For multi-output problems, we treat each output independently. We train a probabilistic NN with a linear output layer that outputs $\mu(\mathbf{x})$ and $\sigma^2(\mathbf{x})$ by maximizing the normal log-likelihood. Denote by $\boldsymbol{\phi}(\mathbf{x})$ the hidden representation of $\mathbf{x}$ that is fed to the NN's final layer.

Using the NN, construct a new data matrix in the hidden representation space,

$$\mathbf{Z} = \begin{pmatrix} \boldsymbol{\phi}(\mathbf{x}_1) \\ \boldsymbol{\phi}(\mathbf{x}_2) \\ \vdots \\ \boldsymbol{\phi}(\mathbf{x}_N) \end{pmatrix} \in \mathbb{R}^{N \times p},$$

where p is the number of units in the last hidden layer. Since the NN is trained to be able to do linear regression on this representation, it makes sense to apply Bayesian linear regression. We always center $\mathbf{Z}$ by subtracting the mean of each feature over the training set.

As in Section 2.4.2, we posit a normal likelihood and a normal prior on the weights:

$$\begin{aligned} p(\mathbf{y} \mid \mathbf{Z}, \mathbf{w}, \boldsymbol{\Sigma}) &= \mathcal{N}(\mathbf{y} \mid \mathbf{Z}\mathbf{w}, \boldsymbol{\Sigma}), \\ p(\mathbf{w}) &= \mathcal{N}(\mathbf{w} \mid \mathbf{w}_0, \mathbf{V}_0). \end{aligned}$$

For prior parameters, $\mathbf{w}_0 = \mathbf{0}_p$ and $\mathbf{V}_0 = g\mathbf{I}_0$ are used, where $g > 0$ controls the prior variance and has an effect analogous to L_2 regularization. The covariance matrix used in the likelihood, $\boldsymbol{\Sigma} = \boldsymbol{\Sigma}(\mathbf{X})$, is constructed with the NN's predicted variance for each input, i.e.,

$$\boldsymbol{\Sigma} = \operatorname{diag}(\sigma^2(\mathbf{x}_1), \sigma^2(\mathbf{x}_2), \ldots, \sigma^2(\mathbf{x}_N)).$$

Using Bayes' rule for linear Gaussian systems (Murphy, 2012), we find the posterior distribution $p(\mathbf{w} \mid \mathbf{X}, \mathbf{y}, \boldsymbol{\Sigma}) = \mathcal{N}(\mathbf{w} \mid \mathbf{w}_N, \mathbf{V}_n)$ with parameters

$$\begin{aligned} \mathbf{V}_n &= (\mathbf{V}_0^{-1} + \mathbf{Z}^T\boldsymbol{\Sigma}^{-1}\mathbf{Z})^{-1} \\ \mathbf{w}_N &= \mathbf{V}_n(\mathbf{V}_0^{-1}\mathbf{w}_0 + \mathbf{Z}^T\boldsymbol{\Sigma}^{-1}\mathbf{y}). \end{aligned}$$

Furthermore, the posterior predictive distribution at a test point $\mathbf{x}$ is given by

$$p(y \mid \mathbf{x}, \mathbf{X}, \mathbf{y}, \boldsymbol{\Sigma}) = \mathcal{N}(y \mid \mathbf{w}_N^T\boldsymbol{\phi}(\boldsymbol{x}), \sigma^2(\mathbf{x}) + \boldsymbol{\phi}(\boldsymbol{x})^T\mathbf{V}_N\boldsymbol{\phi}(\boldsymbol{x})).$$

3.4.2 Deep BLR ensemble

It is straightforward to combine PE and Deep BLR by simply adding Deep BLR on top of each NN. Since the predictive distribution of each NN is normal, we can easily replace it with the BLR posterior predictive distribution. This can be viewed either as improving BLR by introducing more variety in the representations, or as "squeezing out" more uncertainty from each NN.

3.4.3 Implementation details

The method is simple to implement: we just train a NN as usual, obtain the final hidden layer representation for each sample, and use the equations described above. A naive implementation will scale poorly in the number of samples since we must store and invert a $N \times N$ covariance matrix, but making use of libraries supporting sparse matrices (we use `scipy.sparse.diags` from SciPy (Jones et al., 2001)) and noting that the covariance matrix is diagonal circumvents these issues.

3.4.4 The limited applicability of Deep BLR

A major weakness of Deep BLR is that it, in practice, only works for architectures with a fully connected, linear output layer. This essentially means that it cannot be used for architectures with a convolutional output layer, which usually is the case when the output is an image. Some examples of this are predicting the next frame in Atari games for model-based RL (Leibfried et al., 2017; Kaiser et al., 2019) and predicting the depth of images (Ma et al., 2018). As far as we know there is no way to efficiently replicate convolutions in the framework of linear regression, but we describe one attempt.

It is possible to view a convolutional layer as linear regression with some data augmentation to enable parameter sharing. We illustrate this with an example. The final convolutional layer in Ma et al. (2018) applies an 1×1 convolution to each input channel, where the input has shape $(128, 352, 1216)$. Each filter activation can be viewed as a linear regression over the input channels, but we must construct $352 \times 1216 = 428,032$ examples for each input image[1] given that we want to share parameters as in a convolutional layer. While no expensive matrix inversions are necessary, we must now compute, e.g., $\boldsymbol{\phi}(\boldsymbol{x})^T \mathbf{V}_N \boldsymbol{\phi}(\boldsymbol{x})$ where $\boldsymbol{\phi}(\boldsymbol{x})$ is 468032×128 and $\mathbf{V}_N$ is 128×128, to get the predictive distribution for a single input $\boldsymbol{x}$. Similar computation is needed to get the posterior distribution itself. While the computation can be done, it is far too demanding for any real-time application.

Despite the limited applicability, Deep BLR can be useful for the many problems where the output *is* linear and fully connected. All hope is not lost for image-based RL either. An alternative approach to directly dealing with images in RL is to transform the observations to a low-dimensional latent space using, for example, a variational autoencoder (as done by Ha and Schmidhuber (2018)), and do RL in that space instead. Deep BLR can then be used to obtain uncertainty in our latent space predictions. Another approach to solving this is to do approximate inference (e.g., by variational inference), but this is not so appealing since it takes us away from simple, closed-form updates.

[1] Each example corresponds to the filter being applied to one pixel across the input channels.

3.5 Discussion: Computational footprint

MC-dropout and Deep BLR are the cheapest in terms of training: we only need to train a single NN, and the additional computation for Deep BLR will be negligible in most cases. Probabilistic ensembles and cSGMCMC ensembles require training M NNs which may add a significant footprint, but in the case of Probabilistic ensembles training can be parallelized, while cSGMCMC permits sampling multiple NNs in each cycle which may decrease the number of cycles necessary.

When it is time for inference, Deep BLR only requires a single forward-pass, but computing the posterior predictive distribution may add some overhead for high-dimensional representations. MC-dropout only requires storing a single NN, but many forward passes are necessary. Probabilistic ensembles and cSGMCMC ensembles both require storing M NNs and doing one forward pass through each.

The size of the NN is crucial to this discussion, especially in comparing PE with Deep BLR. For large NNs with millions of parameters, ensembling can often only be done by either training sequentially (which is time-consuming) or by using parallel GPUs (which is expensive), while Deep BLR only requires training a single NN. However, in the context of transition models in RL, the NNs tend to be small (unless images are involved) and so the whole ensemble can be trained simultaneously. In effect, the power of modern GPUs makes it so that training M NNs often takes no longer than one NN. The same principle holds for inference. Thus, while ensembling appears demanding at first glance, Deep BLR can be slower due to the additional overhead.

3.6 Intermission

We have described five methods for constructing uncertainty-aware NNs, all of which, with the exception of BLR, are ensemble-based. The difference lies in how the NNs making up the ensemble are obtained: PE relies on random initialization, cSGMCMC takes snapshots of the weights during training with a modified learning procedure, and MC-dropout randomly drop outs units. Each of these can be viewed as sampling weights from some posterior distribution and constructing a Monte Carlo estimate of the predictive distribution by averaging out over a finite set of possible models explaining the data. This is not necessary for BLR since we have closed-form solutions for the posterior and predictive distributions.

Since each NN outputs a normal distribution, the output of the ensemble will be distributed as a mixture of normals. In Section 4.1 we approximate this mixture with a normal distribution by matching the first two moments, as done by Lakshminarayanan et al. (2017), to obtain confidence intervals. In Section 4.2 we use the full distribution to compute the predictive negative log-likelihood.

3.7 Survey of other methods

The most prominent set of methods that we haven't considered in this thesis are those that directly attempt approximate inference for BNNs by variational inference. In this section, we discuss some alternatives from the ever-growing literature on uncertainty-aware NNs.

Hamiltonian Monte Carlo (HMC, Neal (2011)) is a MCMC method often mentioned as the "gold standard" for exact inference in BNNs. While accurate, it doesn't scale. We put significant effort in attempting exact inference for BNNs for the toy 1-D regression examples seen in Chapter 4 with limited success for NNs with more than 20 hidden units. In particular, inference in probabilistic NNs (i.e., with heteroscedastic variance) seems problematic.

Bayes-by-Backprop (BBB, (Blundell et al., 2015)) approximates the posterior distribution over weights with a diagonal normal distribution by minimizing the Kullback-Leibler (KL) divergence to the true posterior, i.e. variational inference. The KL divergence is approximated with Monte Carlo samples from the approximate posterior (using the reparametrization trick (Kingma and Welling, 2013)) and is minimized with SGD.

Hernández-Lobato and Adams proposed Probabilistic Backpropagation (PBP), which also approximates the posterior with a factored distribution, but applies something called assumed density filtering (which we will not attempt to explain) instead of VI. Their experiments indicate that it outperforms the VI approach of Graves (2011) (which is what BBB is based on and improves upon.)

Louizos and Welling (2016) and Sun et al. (2017) extend VI and PBP respectively to allow posterior correlations between the weights in the same layer by using normal distributions over matrices. This seems to perform better than the factored univariate distributions previously used.

Hernández-Lobato et al. (2016) proposed black-box alpha-divergence minimization (BB-α) in which the α-divergence is minimized, of which KL is a special case. Minibatch SGD is used to minimize a Monte Carlo estimate of the resulting energy function. It is shown that BB-α becomes VI when $\alpha \to 0$. The posterior is approximated with a factorized normal distribution (as in BBB and PBP), but the method supposedly transfers easily to more complex models. Depeweg et al. (2017) use BB-α in the context of model-based RL with some success.

An interesting recent development is the functional variational BNNs (fvBNN) proposed by Sun et al. (2019). The idea is to do approximate inference on distributions over functions rather than distributions in parameter space. With appealing theory and promising results on regression datasets and the bandit benchmark of Riquelme et al. (2018), we think this is the most promising candidate that we unfortunately did not have time to evaluate.

Another line of research has produced Noisy K-FAC (Kronecker-factored Approx-

imate Curvature, (Zhang et al., 2018)), in which an approximation of the Fisher information matrix is used for natural gradient descent. It is shown that this approximates VI with appropriately added noise. As in (Louizos and Welling, 2016; Sun et al., 2017), matrix variate Gaussian posterior approximations are used. Like fvBNN, good empirical results are shown and it seems promising.

A clear disadvantage of many of the methods mentioned is that they are complicated to derive and implement, as acknowledged by Korattikara et al. (2015). We spent some time playing around with BBB, PBP, and BB-α, but had little success in getting them to work. This contributed significantly to our choice of focusing on other methods. Ideally, we would have liked to try more methods, especially fvBNN and Noisy K-FAC, but of course, time is limited. We discuss the potential for future research in Chapter 6.

4

Evaluation in Supervised Learning

In this chapter, we study the performance of the methods on successively more difficult regression tasks. We begin with some toy regression datasets and visually evaluate the resulting posterior distributions. For a more quantitative comparison, we proceed with some real and more complex datasets from the *UCI Machine Learning Repository* (Dua and Graff, 2017). The chapter ends with a short study of the uncertainty estimates for out-of-distribution inputs.

4.1 Toy regression

In this section, the chosen methods for estimating uncertainty are evaluated visually on two 1-D toy regression datasets, followed by a simple 2-D regression dataset. On one of the datasets, we provide the "true" uncertainty as computed with a ReLU GP (which, as we saw in Section 2.5.2, is the limit of a BNN.) While this type of evaluation is subjective and possibly of little relevance for large-scale use, we believe that it acts as a sanity check: can the methods do well on very simple problems? It also allows us to familiarize ourselves with, e.g., hyperparameters in a setting where the results can be visualized.

In these experiments, we visually judge whether a posterior is "good" or not, but that is, of course, subjective. Given some prior and likelihood, Bayes' rule gives us the true posterior (although it is often difficult or impossible to compute), but the choice of model is subjective and for most methods we cannot explicitly specify a prior. This makes evaluating the success of each method difficult and inherently subjective. What can be said is that we expect low uncertainty in regions with data (unless there is aleatoric uncertainty), high uncertainty where there is none, and smooth interpolation in between.

All nets were trained with Adam (Kingma and Ba, 2015) to convergence. See Appendix A.1 for a complete list of hyperparameters used for each experiment. The hyperparameters for each method were handpicked since selecting a good metric for optimization is difficult. ReLU activations were used for all NNs. The chosen hyperparameters represent our best effort to showcase representative performance and are the same for each problem in this section. We used the library Pyro (Bingham

et al., 2018) for HMC.

4.1.1 One-dimensional regression

In this section, we apply the methods to two 1-D regression tasks, **D1** and **D2**, and study the resulting posterior predictive distributions visually. For the slightly more complicated **D2**, we also study the effect of ensemble size where applicable. Hyperparameters were chosen to perform well across both tasks.

D1 is generated by evaluating

$$f_i(x) = x^3 + \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(0, \sigma_i^2), \quad i \in \{1, 2, 3\}$$

where $\sigma_1 = \sigma_3 = 0.01$ and $\sigma_2 = 1$, on 50 samples of x from each region $\mathcal{R}_i$, where $\mathcal{R}_1 = [-1.5, -0.4]$, $\mathcal{R}_2 = [-0.5, 0.5]$, and $\mathcal{R}_3 = [0.6, 1.5]$. This task constitutes a simple sanity check and also evaluates the ability to represent heteroscedastic aleatoric uncertainty.

Figure 4.1 displays the approximate posterior predictive distributions given by the five methods trained on **D1**. We note that the distributions are qualitatively similar and all are able to capture the heteroscedastic variance present around the origin.

D2 is generated by evaluating

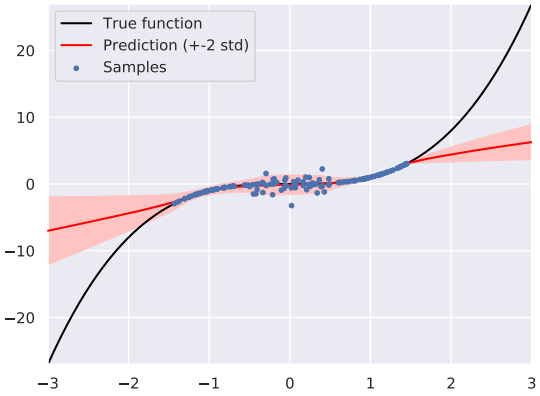
$$g(x) = \sin(x) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, 0.05^2)$$

on 50 samples of x from each of the three regions $\mathcal{R}_1 = [-7, -4]$, $\mathcal{R}_2 = [-2, 0]$, and $\mathcal{R}_3 = [2, 3]$. This task is more complicated than **D1** (but has homoscedastic variance) and also allows us to evaluate intersample uncertainty.

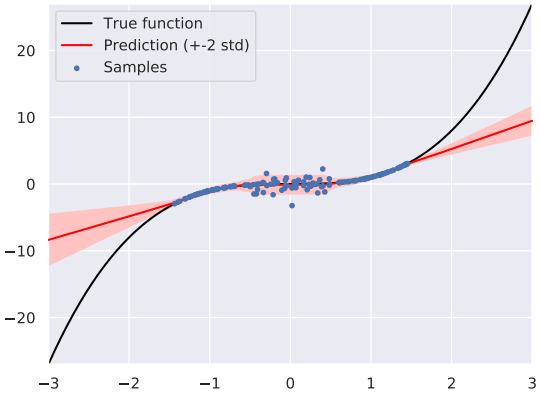
Figure 4.2 displays the approximate posterior predictive distributions given by the five methods trained on **D2**, including a reference result provided by a ReLU GP in 4.2a. Remember that there is no true posterior independent of the prior and likelihood, both of which are different for all methods, and so the GP result doesn't serve as truth, but rather an example of a good result.

We see reasonable results for all methods. PE seems to slightly underestimate the uncertainty between $\mathcal{R}_1$ and $\mathcal{R}_2$, indicating too little variety among the NNs. cSGMCMC does better in this respect. Out of all methods, Deep BLR and Deep BLR ensemble seem to most resemble the ReLU GP uncertainty. Based on these results, all methods seem to produce reasonable uncertainty estimates, and in particular Deep BLR does so despite its naivety.

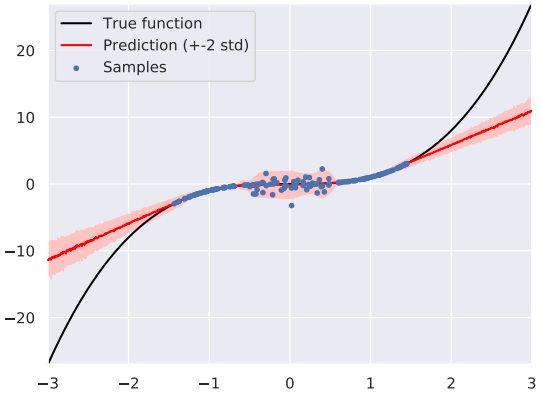
In Figure 4.3 we see the effect of varying the prior weight variance g when using Deep BLR on **D2**. It is clear that higher prior variance leads to higher predictive uncertainty, and that the effect is significant. While this adds another hyperparameter, it makes the subjectivity of uncertainty explicit rather than implicit, which can be seen as an advantage.



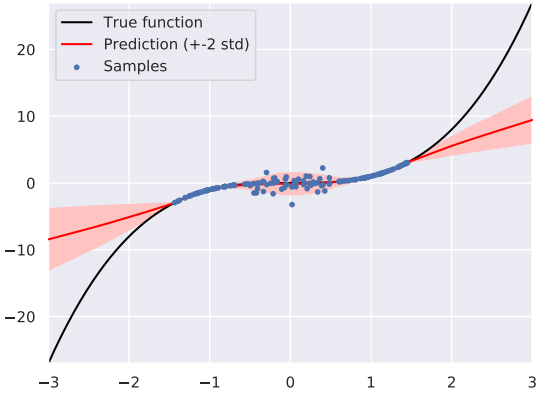
(a) Probabilistic ensemble. 10 NNs with 5 hidden layers, each with 10 ReLU.



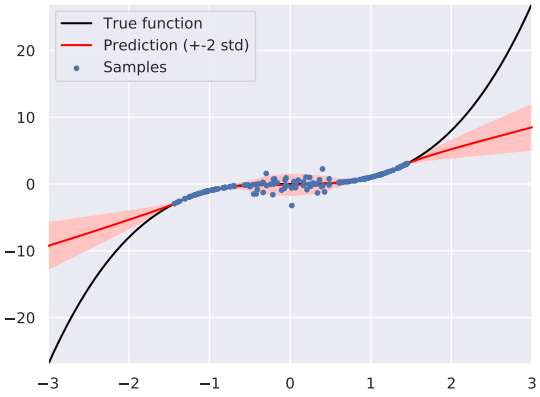
(b) cSGMCMC ensemble. 80 NNs with 5 hidden layers, each with 10 ReLU.



(c) MC-dropout. 100 samples from a NN with 2 hidden layers, each with 10 ReLU, and $p = 0.25$.

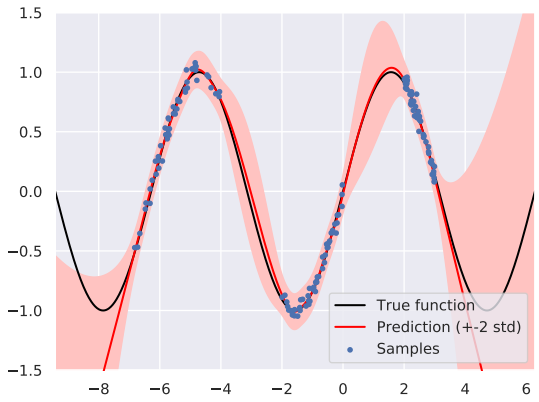


(d) Deep BLR. Prior variance $2\mathbf{I}$ and representation from NN with 3 hidden layers, each with 50 ReLU.

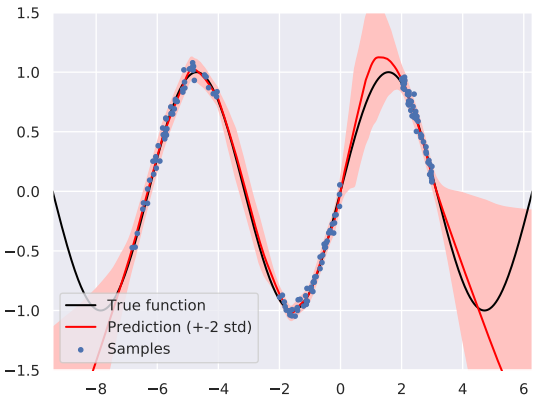


(e) Deep BLR ensemble. 10 instances of Deep BLR with parameters as in 4.1d.

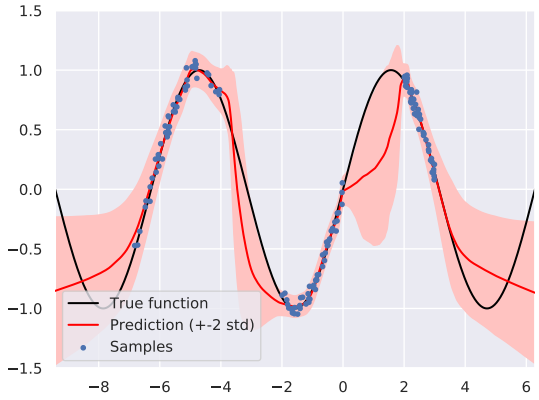
Figure 4.1: Approximate posterior predictive distributions on **D1** for each method. Each method outputs a Gaussian mixture which we approximate with a single Gaussian distribution. The shaded area covers two standard deviations of that approximate posterior predictive distribution. See A.1 for the full set of hyperparameters.



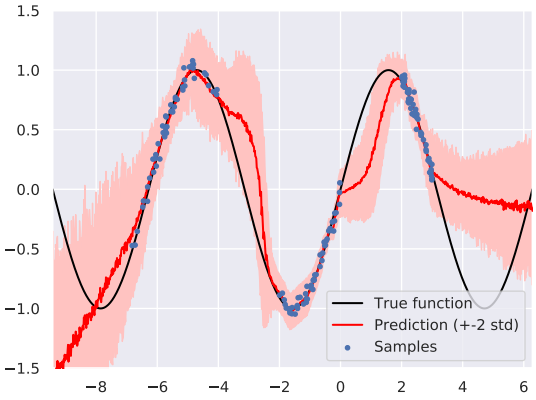
(a) Gaussian process. ReLU kernel with tuned homoscedastic variance.



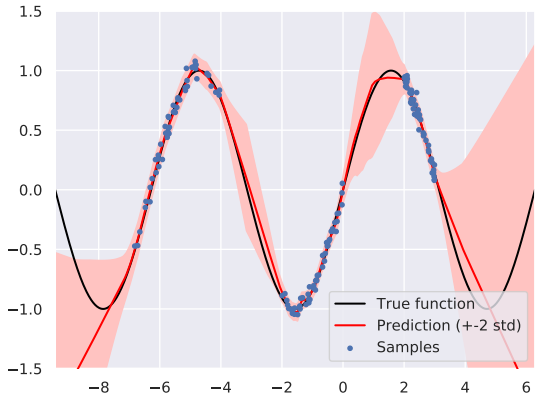
(b) Probabilistic ensemble. 10 NNs with 5 hidden layers with 10 ReLU each.



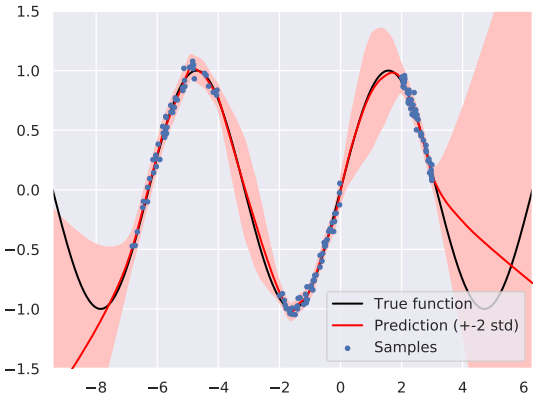
(c) cSGMCMC ensemble. 80 NNs with 5 hidden layers with 10 ReLU each.



(d) MC-dropout. 100 samples from a NN with 2 hidden layers of 1000 ReLU each and $p = 0.25$.

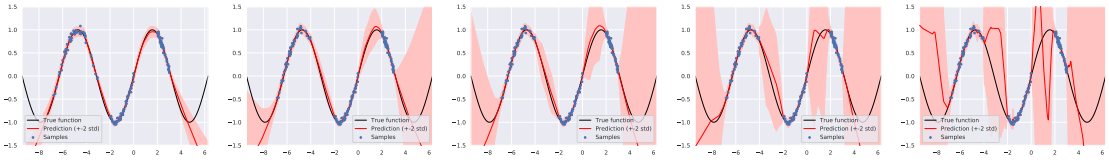


(e) Deep BLR. Prior variance $2\mathbf{I}$ and representation from NN with 3 hidden layers of 50 ReLU each.



(f) Deep BLR ensemble. 10 instances with parameters as in 4.2e.

Figure 4.2: Approximate posterior predictive distributions on **D2** for each method, including a reference distribution in 4.2a. Each method outputs a Gaussian mixture which we approximate with a single Gaussian distribution. The shaded area covers two standard deviations of that approximate posterior predictive distribution. See A.1 for the full set of hyperparameters.

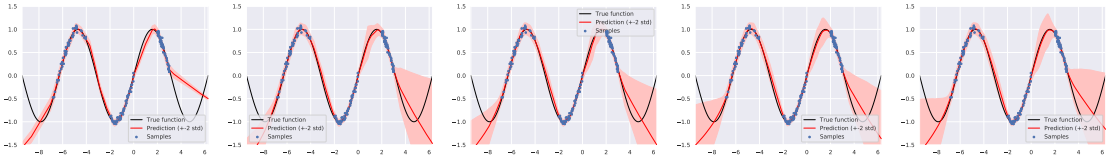


$g = 0.1$ $g = 1$ $g = 10$ $g = 100$ $g = 1000$

Figure 4.3: Effect of the prior weight variance g on uncertainty estimates produced by Deep BLR. It is clear that higher prior variance leads to higher predictive uncertainty. Best viewed on a computer.

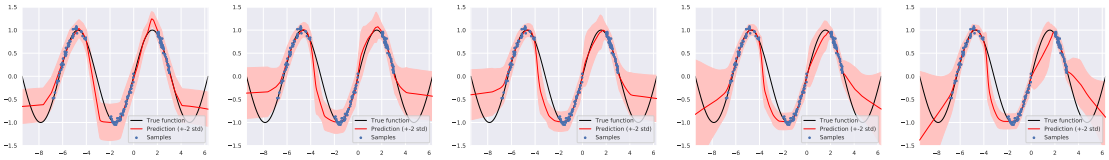
To study the effect of ensemble size on the quality of uncertainty estimates, we trained each ensemble method with $M \in [1, 3, 5, 10, 20, 50]$. The result can be seen in Figure 4.4. Deep BLR doesn't seem to benefit much from ensembling, while $M = 3$ seems sufficient for PE in this case.

Probabilistic ensemble



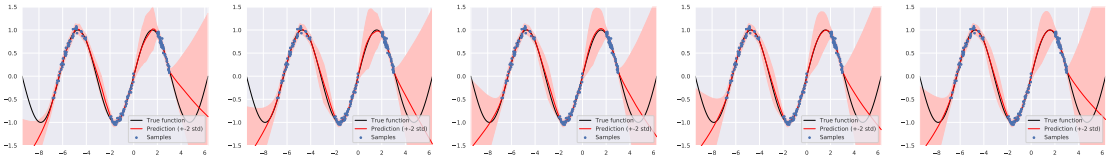
$M = 1$ $M = 3$ $M = 5$ $M = 10$ $M = 20$

cSGMCMC ensemble



$M = 4$ $M = 12$ $M = 20$ $M = 40$ $M = 80$

Deep BLR ensemble



$M = 1$ $M = 3$ $M = 5$ $M = 10$ $M = 20$

Figure 4.4: Effect of ensemble size on uncertainty estimates on **D2** for the ensemble methods. On this simple problem, Probabilistic ensembles perform well with only 3 NNs. Deep BLR performs well with a single NN, and it is unclear if the computational trade-off is worth ensembling in this case. Best viewed on a computer.

4.1.2 Two-dimensional regression

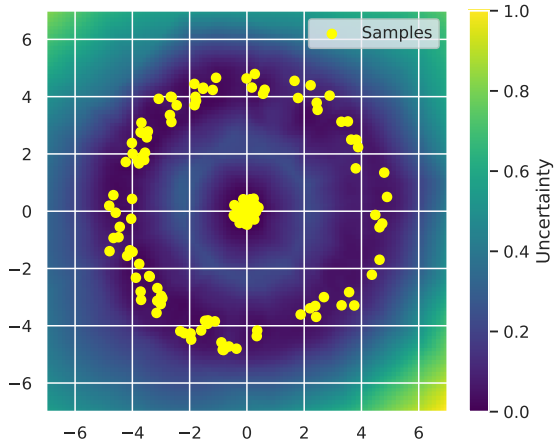
To verify the methods' capacity to handle multiple variables, the methods were also evaluated on a simple two-dimensional regression problem **D3**. The training set

consists of $f(x, y) = x^2 + y^2$ evaluated on random points in the annulus between two concentric circles of radii 4 and 5 respectively and around the origin. We expect the uncertainty to be low in the annulus and around the origin, moderate in between, and increase with the distance from the annulus.

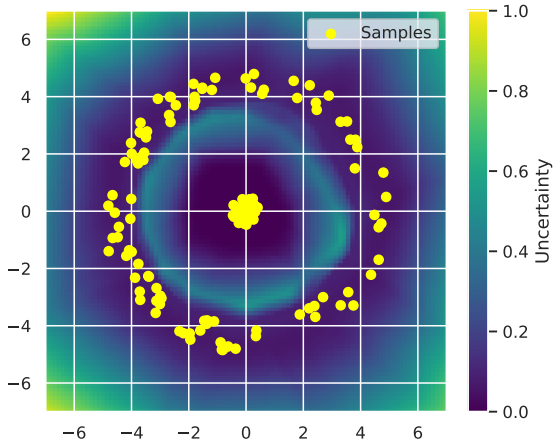
Figure 4.5 displays the results. All methods behave roughly as expected. cSGMCMC estimates an abrupt increase in uncertainty in the middle region, while the other methods have much more smooth uncertainty estimates. Deep BLR has asymmetric uncertainty around the origin which is smoothed out in the Deep BLR ensemble. All methods estimate increased uncertainty outside the annulus.

4.1.3 Discussion

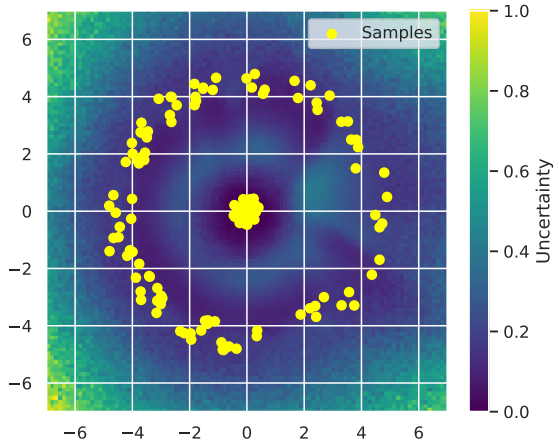
We have seen that all methods provide reasonable uncertainty estimates. Probabilistic ensembles seem promising, but require training many independent NNs. However, this can be parallelized, even on a single GPU if the NNs aren't too large. cSGMCMC ensembles also seem promising, and allow (require!) finer tuning than PE. MC-dropout is simple, but seems less reliable. Deep BLR provides reasonable uncertainty estimates while only requiring training a single network. Also, unlike other methods, it doesn't require multiple forward passes during inference. We have also seen that varying the prior variance g lets us encode some sort of prior uncertainty. This is both a strength and a weakness, but uncertainty is subjective and g makes this explicit, in contrast to PE where no such parameter exists. Figure 4.4 indicates that ensembling BLR is unnecessary for simple 1-D problems, but it may be helpful for more difficult problems.



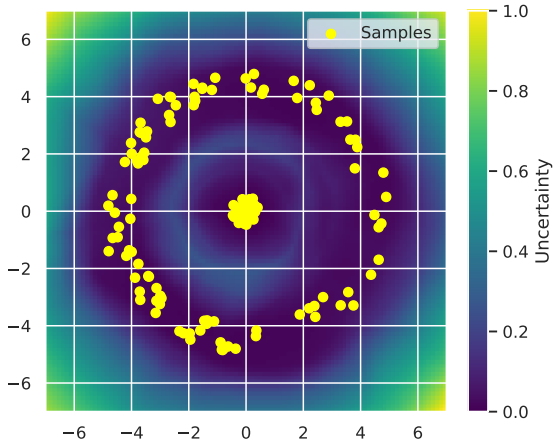
(a) Probabilistic ensemble. 10 NNs with 5 hidden layers with 10 ReLU each.



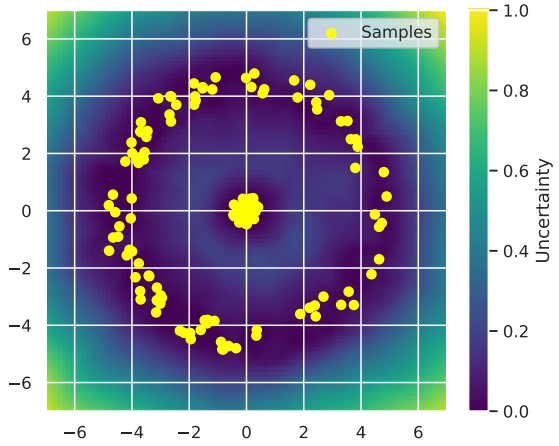
(b) cSGMCMC ensemble. 80 NNs with 5 hidden layers with 10 ReLU each.



(c) MC-dropout. 100 samples from a NN with 2 hidden layers of 1000 ReLU each and $p = 0.25$.



(d) Deep BLR. Prior variance $2\mathbf{I}$ and representation from NN with 3 hidden layers of 50 ReLU each.



(e) Deep BLR ensemble. 10 instances with parameters as in 4.5d.

Figure 4.5: Approximate posterior predictive distributions on the two-dimensional dataset **D3** for each method. Each method outputs a Gaussian mixture which we approximate with a single Gaussian distribution. The heatmap is constructed from the normalized variance of this distribution. See A.1 for the full set of hyperparameters.

4.2 *UCI* regression

In this section, we proceed to compare the methods by testing them on real-world regression datasets from the *UCI Machine Learning Repository* (Dua and Graff, 2017). By looking at the predictive mean-squared error and log-likelihood, we get a quantitative understanding of the methods' performance. We follow the experiment set-up of previous work on uncertainty-aware NNs (Hernández-Lobato and Adams; Gal and Ghahramani, 2016; Lakshminarayanan et al., 2017), but on a subset of the datasets due to limited time and resources.

4.2.1 Experimental setup

Specifically, the methods are evaluated on the following datasets:

- **Boston Housing**, with 506 samples and 13 features,
- **Concrete Strength**, with $1,030$ samples and 8 features,
- **Kin8nm**, with $8,192$ samples and 8 features,
- **Power Plant**, with $9,568$ samples and 4 features,
- **Protein Structure**, with $45,730$ samples and 9 features, and
- **Year Prediction MSD**, with $515,345$ samples and 90 features.

To obtain robust performance metrics, we evaluate each method on 20 random 90/10 training/test splits except for Protein Structure and Year Prediction MSD where 5 and 1 splits are done respectively due to their size. Input and target variables were normalized by the mean and standard deviation based on the training set. Since each method outputs a mixture of normals, the negative log-likelihood is obtained with the mixture density.

4.2.2 Training and hyperparameters

We follow prior work (Hernández-Lobato and Adams; Gal and Ghahramani, 2016; Lakshminarayanan et al., 2017) and use a NN with a single hidden layer with 50 ReLU for all datasets, except for the larger Protein Structure and Year Prediction MSD where 100 ReLUs are used. Each network outputs the mean and variance of a normal distribution and is trained by minimizing the negative log-likelihood, thus estimating the aleatoric uncertainty.

With the exception of the cSGMCMC ensemble, each NN is optimized for 40 epochs using Adam (Kingma and Ba, 2015) with batch size 32 and learning rate 0.01 (0.001

and 0.0001 for Protein Structure and Year Prediction MSD respectively). The cSGMCMC ensemble is trained for $5 \cdot 40$ epochs, following the procedure described in 3.3. In the cases where hyperparameter optimization is necessary, we perform a grid-search to minimize the negative log-likelihood on a 20% validation set contained within the training set. The algorithm-specific hyperparameters are as follows:

- **Probabilistic MC-dropout.** The dropout rate p is tuned by a grid search over $p \in \{0.005, 0.01, 0.05, 0.1\}$ for each split.
- **Probabilistic ensemble.** 5 NNs are used the ensemble, as in Lakshminarayanan et al. (2017).
- **cSGMCMC ensemble.** Trained for 5 cycles of 40 epochs each. Learning proportion $\beta = 0.9$ and 4 NNs sampled during the sampling stage. The initial learning rate α_0 and the prior variance σ are tuned with Bayesian optimization (BO, (Snoek et al., 2012) for 10 iterations with $\alpha_0 \in [0.1, 10]$ and $\sigma \in [0.01, 10]$ on a single random split. For the larger data sets Protein Structure and Year Prediction MSD, BO was replaced by minor hand-tuning, giving $\alpha_0 = 0.1$, $\sigma = 5.0$ and $\alpha_0 = 0.5$, $\sigma = 5.0$ respectively.
- **Deep BLR.** The prior variance g is tuned by a grid search over 50 logarithmically spaced points in $[10^{-2}, 10^4]$ for each split. Note that this is very cheap since the NN doesn't have to be retrained.
- **Deep BLR ensemble.** Deep BLR was applied in the same fashion as above, but on 5 NNs trained as for PE.

4.2.3 Results

The results are contained in Table 4.1, where the first table contains the predictive RMSE and the second the negative log-likelihood which measures the methods' ability to handle predictive uncertainty. The mean and standard error over all splits is reported. For each dataset, the methods are ranked from 1 to 5 (where 1 is best). The sum of ranks across all datasets are displayed below each method with the lowest rank marked in bold.

There is no clear winner of this evaluation. Considering RMSE and NLL separately, we find that cSGMCMC ensemble and Deep BLR ensemble do best respectively. Deep BLR ensemble consistently produces the lowest NLL, which is promising. We also note that, overall, all methods (perhaps with the exception of P-Dropout) perform very similarly, which indicates that the benchmark may be too easy.

PE and cSGMCMC are in the same ballpark but cSGMCMC wins out slightly in terms of RMSE, which may be due to poor optimization of PE. The number of hyperparameters is the main differentiator of the methods. PE is simpler and requires no additional parameter, bar the number of NNs in the ensemble. cSGMCMC requires more tuning which can be costly, but may also lead to significantly improved

Predictive RMSE

Dataset	**P-Dropout**	**PE**	**cSGMCMC**	**dBLR**	**dBLRE**
Boston Housing	3.17 ± 0.18 (1)	3.18 ± 0.23 (1)	3.07 ± 0.19 (1)	3.03 ± 0.18 (1)	3.27 ± 0.25 (1)
Concrete Strength	5.75 ± 0.13 (3)	5.30 ± 0.12 (2)	4.99 ± 0.10 (1)	5.57 ± 0.13 (3)	5.29 ± 0.11 (2)
Kin8nm	0.09 ± 0.00 (3)	0.08 ± 0.00 (2)	0.08 ± 0.00 (2)	0.08 ± 0.00 (2)	0.07 ± 0.00 (1)
Power Plant	4.29 ± 0.04 (3)	4.03 ± 0.03 (2)	3.95 ± 0.03 (1)	4.04 ± 0.02 (2)	4.03 ± 0.03 (2)
Protein Structure	4.69 ± 0.02 (4)	4.46 ± 0.02 (3)	4.40 ± 0.02 (1)	4.43 ± 0.00 (2)	4.41 ± 0.02 (1)
Year Prediction MSD	$9.12 \pm$ NA (5)	$8.93 \pm$ NA (3)	$8.84 \pm$ NA (1)	$8.90 \pm$ NA (2)	$8.96 \pm$ NA (4)
Sum of ranks	19	13	**7**	12	11

Predictive NLL

Dataset	**P-Dropout**	**PE**	**cSGMCMC**	**dBLR**	**dBLRE**
Boston Housing	2.40 ± 0.05 (1)	2.40 ± 0.05 (1)	2.47 ± 0.06 (1)	2.47 ± 0.05 (1)	2.37 ± 0.05 (1)
Concrete Strength	3.05 ± 0.03 (2)	2.97 ± 0.02 (1)	2.96 ± 0.03 (1)	3.04 ± 0.03 (2)	2.93 ± 0.03 (1)
Kin8nm	-1.10 ± 0.01 (4)	-1.22 ± 0.00 (3)	-1.26 ± 0.01 (1)	-1.20 ± 0.00 (2)	-1.25 ± 0.00 (1)
Power Plant	2.84 ± 0.01 (3)	2.79 ± 0.00 (2)	2.77 ± 0.01 (1)	2.80 ± 0.01 (2)	2.78 ± 0.00 (1)
Protein Structure	2.82 ± 0.02 (3)	2.78 ± 0.01 (2)	2.83 ± 0.03 (3)	2.80 ± 0.02 (2)	2.77 ± 0.01 (1)
Year Prediction MSD	$3.40 \pm$ NA (2)	$3.39 \pm$ NA (2)	$3.70 \pm$ NA (4)	$3.58 \pm$ NA (3)	$3.38 \pm$ NA (1)
Sum of ranks	15	11	11	12	**6**

Table 4.1: Comparison of MC-dropout, Random ensemble, cSGMCMC ensemble, Deep BLR, and Deep BLR ensemble on a set of standard datasets. We report the mean and standard error of predictive RMSE and NLL across 20 random training-test splits (5 for Protein Structure, 1 for Year Prediction MSD). We rank the methods for each data set (up to a standard error) and sum the ranks to quantify each method's overall performance.

	Predictive RMSE		Predictive NLL	
Dataset	**D-Dropout**	**P-Dropout**	**D-Dropout**	**P-Dropout**
Boston Housing	$\mathbf{2.97 \pm 0.85}$	$\mathbf{3.17 \pm 0.18}$	$\mathbf{2.46 \pm 0.25}$	$\mathbf{2.40 \pm 0.05}$
Concrete Strength	$\mathbf{5.23 \pm 0.53}$	$\mathbf{5.75 \pm 0.13}$	$\mathbf{3.04 \pm 0.09}$	$\mathbf{3.05 \pm 0.03}$
Kin8nm	0.10 ± 0.00	$\mathbf{0.09 \pm 0.00}$	-0.95 ± 0.03	$\mathbf{-1.10 \pm 0.01}$
Power Plant	$\mathbf{4.02 \pm 0.18}$	4.29 ± 0.04	$\mathbf{2.80 \pm 0.05}$	$\mathbf{2.84 \pm 0.01}$
Protein Structure	$\mathbf{4.36 \pm 0.04}$	$4.69 \pm .02$	2.89 ± 0.01	$\mathbf{2.82 \pm 0.02}$
Year Prediction MSD	$\mathbf{8.85 \pm NA}$	$9.12 \pm$ NA	$3.59 \pm$ NA	$\mathbf{3.40 \pm NA}$

Table 4.2: Comparison of MC-dropout with deterministic (minimizing MSE) and probabilistic (minimizing NLL) NNs performed as in 4.1. Results for D-Dropout obtained from Gal and Ghahramani (2016).

performance. We believe that a more efficient heuristic for the initial learning rate can be constructed, which would make the tuning process easier.

We note slightly better performance for PE than seen in Lakshminarayanan et al. (2017) and believe that this can be attributed to our batch size of 32 as opposed to the 100 originally used.

At the bottom of the ranking lie P-Dropout and Deep BLR. Both have similar performance in terms of NLL, but P-Dropout performs poorly in terms of RMSE. As above, this may be improved with better optimization. In the original MC-dropout paper, (Gal and Ghahramani, 2016) use a deterministic NN which doesn't take aleatoric uncertainty into account. Table 4.2 compares the predictive performance of the different approaches. We see that the difference is mostly negligible, although D-Dropout tends to have lower RMSE and P-Dropout lower NLL which matches the corresponding optimization criteria. Since we are primarily interested in accurate uncertainty estimation, we believe P-Dropout suits our purposes better, hence our choice.

4.2.4 Conclusions and further work

The combination of PE and Deep BLR leads to better predictive performance in terms of NLL than the methods it was compared with. Furthermore, while Deep BLR doesn't outperform any of the ensemble methods by itself, it delivers competitive performance with no modifications to the underlying NN. Tweaking the architecture may lead to significant improvement in performance. MC-dropout doesn't perform very well and combined with the theoretical weakness previously discussed, this evaluation leads us to prefer the other methods. Finally, both PE and cSGMCMC ensembles do well in this evaluation, with simplicity and flexibility as their respective strengths and weaknesses.

In the interest of following prior work, we kept the network architecture constant for all methods. It is clear that this may lead to a somewhat unfair comparison since the methods may benefit from individual tuning. However, individual tuning of all parameters is prohibitively expensive, and any restriction may introduce bias. For these reasons, we leave a more thorough comparison to future work.

Evaluating the methods on more difficult problems would also be interesting. It is not clear whether the small differences in performance are due to the difficulty of the datasets or actual small difference in predictive performance of the methods.

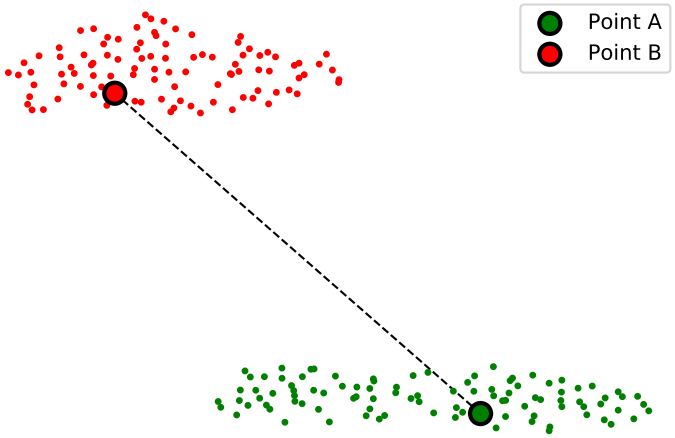


Figure 4.6: Illustration of the experimental setup used in Section 4.3. The figure depicts the dataset (with reduced dimension) with the two test points and the line segment highlighted. Dimensionality reduction was done using the nonlinear dimensionality reduction algorithm t-SNE (van der Maaten and Hinton, 2008).

4.3 Out-of-distribution inputs

A reliable uncertainty-aware NN should produce higher uncertainty estimates for out-of-distribution inputs, i.e., on samples that are far away from the training set. In this section, we present a short evaluation of this capability on a simulated dataset.

4.3.1 Simulated data and experimental setup

The data used for this experiment is generated with `make_classification` from scikit-learn (Pedregosa et al., 2011). We generate 200 samples with 10 features (of which 5 are informative, 3 redundant, and 2 noise) that essentially lie in two separate clusters. While originally intended for classification, we do regression on the class labels. Input variables and target values are normalized to have mean 0 and standard deviation 1.

The idea is as follows: pick two random points, one in each cluster, and study how the uncertainty varies as we move along the line segment between them. That is, for two points $\mathbf{x}_A, \mathbf{x}_B$, evaluate the estimated uncertainty for all points

$$\mathbf{x} \in \{(1-\alpha)\mathbf{x}_A + \alpha\mathbf{x}_B \mid \alpha \in [0,1]\}.$$

Note that these $\mathbf{x}$ are *not* in the training set. Figure 4.6 illustrates the setup.

4.3.2 Methods and hyperparameters

We use the same methods and general hyperparameters as in Section 4.2, i.e. all NNs have one hidden layer with 50 ReLU. The NNs are trained to convergence using Adam (Kingma and Ba, 2015) with learning rate 0.001 and batch size 32. All ensembles except cSGMCMC use 5 NNs. cSGMCMC uses 20 NNs obtained from 5 cycles with $\alpha = 0.5$ and $\beta = 0.9$, as in the previous section. The remaining hyperparameters were tuned manually on another set of points $\mathbf{x}_A$ and $\mathbf{x}_B$ that were replaced for the final evaluation. The following hyperparameters were chosen:

- **Deep BLR.** Prior variance $g = 1$.
- **MC-dropout.** Dropout probability $p = 0.1$.
- **cSGMCMC.** Initial stepsize $\alpha_0 = 0.5$ and prior standard deviation $\sigma_0 = 1$.

We note that both Deep BLR and MC-dropout work well for a wide range of hyperparameters while cSGMCMC is much more sensitive.

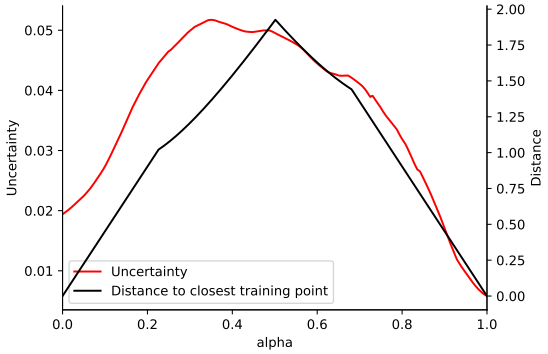
4.3.3 Results

Figure 4.7 contains the results. The predictive uncertainty is obtained for points along the line segment between $\mathbf{x}_A$ and $\mathbf{x}_B$, i.e., $\alpha = 0$ and $\alpha = 1$ gives the uncertainty for $\mathbf{x}_A$ and $\mathbf{x}_B$ respectively. The uncertainty is represented by the standard deviation of the predictive distributions. For each point, we also plot the distance to the closest point in the training set.

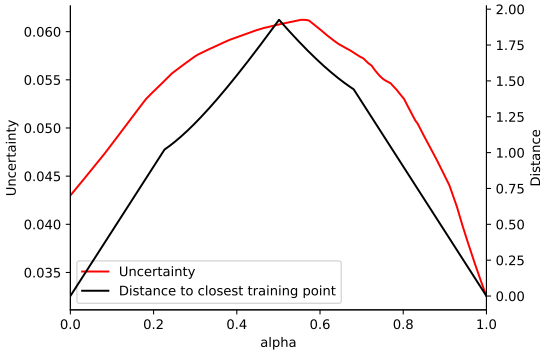
The first thing to note is that all methods estimate higher uncertainty for one of the training points, which shouldn't be surprising. Furthermore, all methods estimate increased uncertainty between the two points, which is good. PE and Deep BLR ensemble have very similar uncertainty estimates, whereas cSGMCMC and Deep BLR differ slightly. The only method that really stands out is MC-dropout, where we see the uncertainty increasing very slowly, particularly from $\alpha = 1$, and then change drastically. This means that MC-dropout would estimate fairly low uncertainty even for $\alpha = 0.6$ which is far from any data it has seen.

4.3.4 Conclusions

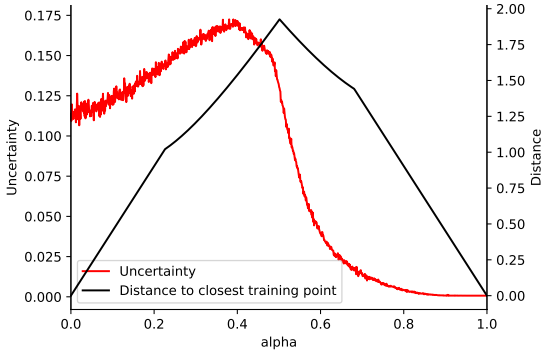
We have seen that all methods estimate higher uncertainty for out-of-distribution inputs which is promising. However, MC-dropout doesn't behave like the other methods and provides questionable uncertainty estimates. This lends further credence to our previous conclusion that MC-dropout is the worst method out of the ones evaluated.



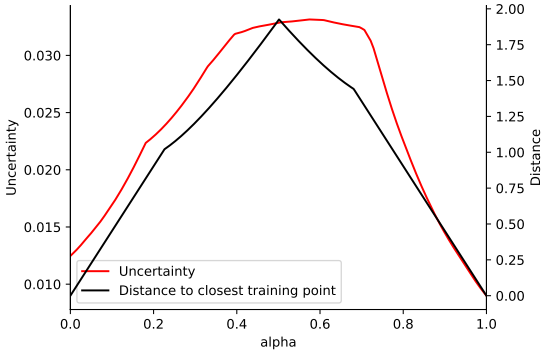
(a) Probabilistic ensemble.



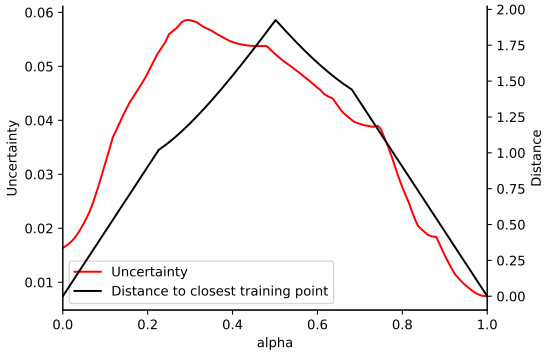
(b) cSGMCMC ensemble.



(c) MC-dropout.



(d) Deep BLR.



(e) Deep BLR ensemble.

Figure 4.7: Uncertainty estimates along the line segment between two points in the training set, as described in Section 4.3, for different methods for obtaining uncertainty-aware NNs. The uncertainty is represented by the standard deviation of the predictive distributions. For each point, the distance to the closest point in the training set is plotted.

5

Evaluation in Reinforcement Learning

While uncertainty-aware neural networks can be useful in almost any deep learning application, we are primarily interested in using them as uncertainty-aware models in model-based reinforcement learning. In this chapter, Probabilistic ensembles, Deep Bayesian linear regression and Deep BLR ensembles are compared based on the downstream performance when used as components in a RL algorithm.

The algorithm uses uncertainty-aware models for particle-based planning, where probabilistic ensembles were originally used. While evaluating also the other methods for obtaining uncertainty would be interesting, the experiments conducted require significant computation and so we picked the candidate we found most interesting.

5.1 Deep BLR in PETS

We use the RL algorithm Probabilistic Ensembles with Trajectory Sampling (PETS, Chua et al. (2018)). It is a planning-based algorithm that, for each interaction, generates action sequences that are evaluated based on average performance across all models in the probabilistic ensemble. The first action of the best action sequence is then executed in the real environment, and planning recommences. The uncertainty-aware model is updated after each episode. Chua et al. (2018) show that it learns faster than state-of-the-art model-free algorithms on some continuous toy environments.

The section is organized as follows: the algorithm is first described in more detail, followed by a description of the experiments, and ends with the results and a brief discussion.

5.1.1 The algorithm

The particular instantiation of the algorithm that is used is PETS with TS1 using the cross-entropy method (CEM, (Botev et al., 2013)) for optimization. We chose TS1 as it is the main variant presented in the paper, but also because it allows us to sample directly from the posterior predictive distribution which is convenient.

Algorithm 1 PETS with TS1 (Chua et al., 2018):

Initialize data $\mathcal{D}$ with a random controller for one trial.
for Trial $k = 1$ to K **do**
  Train uncertainty-aware dynamics model to obtain $p(\mathbf{s}' \mid \mathbf{s}, \mathbf{a}, \mathcal{D})$.
  for Time $t = 0$ to TaskHorizon **do**
    for Actions sampled $\mathbf{a}_{t:t+T} \sim \text{CEM}(\cdot)$, 1 to NSamples **do**
      Propagate P state particles $\mathbf{s}_\tau^p$ with $\mathbf{a}_{t:t+T}$ using $p(\mathbf{s}' \mid \mathbf{s}, \mathbf{a}, \mathcal{D})$.
      Evaluate the cost of $\mathbf{a}_{t:t+T}$ as $-\frac{1}{P}\sum_{p=1}^{P}\sum_{\tau=t}^{t+T} r(s_\tau^p, \mathbf{a}_\tau)$.
      Update CEM$(\cdot)$ distribution using the computed cost.
    Execute first action $\mathbf{a}_t^*$ from the optimal action sequence $\mathbf{a}_{t:t+T}^*$.
    Record outcome: $\mathcal{D} \leftarrow \mathcal{D} \cup \{\mathbf{s}_t, \mathbf{a}_t^*, \mathbf{s}_{t+1}\}$.

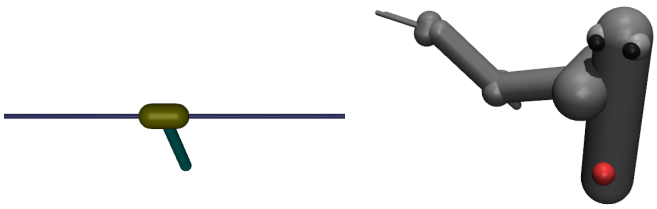
Algorithm 1 describes PETS with TS1. In the case of a Deep BLR model, $p(\mathbf{s}' \mid \mathbf{s}, \mathbf{a}, \mathcal{D})$ refers to the posterior predictive distribution. When a probabilistic ensemble is used, it refers to the uniformly weighted normal mixture density.

At each timestep in the true environment, the cross-entropy method (CEM) is used for some number of iterations to attempt to find the action sequence with the lowest mean cost (highest mean total reward) across all P particles, each following its own trajectory using the dynamics model p for some horizon. CEM optimizes in an evolutionary fashion by iteratively sampling action sequences from a truncated normal distribution, computing their costs, and refitting the distribution using some number of elite (top performing) action sequences.

For a concrete example, suppose the planning horizon is 25, the action dimension is 2, the population size is 400, and the number of elites is 40. We also have a parameter $\alpha \in [0, 1)$. At each iteration i, 400 samples are drawn from a truncated normal distribution with mean and variance μ_i and σ_i^2, both $25 \cdot 2$-dimensional. The cost for each sample, representing an action sequence, is then computed as described in Algorithm 1. The samples are sorted by their cost, and the mean and variance μ_E and σ_E^2 of the 40 elite samples is computed. The mean and variance are then updated for the next iteration to be

$$\mu_{i+1} = \alpha\mu_i + (1-\alpha)\mu_E$$
$$\sigma_{i+1}^2 = \alpha\sigma_i^2 + (1-\alpha)\sigma_E^2.$$

After some iterations, e.g. 5, the optimization is interrupted and the best action executed. For the next timestep, the previous best action sequence (with the first action removed) is used as the initial mean for the CEM optimization.



(a) Cartpole. **(b)** Reacher.

Figure 5.1: The two environments used in our evaluation. In Cartpole, the goal is to swing up the pole by controlling the horizontal force applied to the cart. The observation is 4-dimensional and contains the position and velocity of the cart and pole. In Reacher, the goal is to control the robot arm to touch the ball. The action and observation spaces are 7- (the robot has 7 degrees of freedom) and 17-dimensional respectively.

5.1.2 Experimental details

We compare the downstream performance of the two methods for estimating uncertainty by running PETS on two environments using the MuJoCo (Todorov et al., 2012) simulator, Cartpole and 7-dof Reacher. Chua et al. (2018) also tried Pusher or Half-Cheetah, which we skip only due to lack of resources. Each experiment was repeated 10 times with different random seeds to combat the large variance.

We use the same hyperparameters as Chua et al. (2018), only replacing the linear output layer of the NN with BLR. The NN has three hidden fully-connected layers with 500 and 200 Swish units for Cartpole and Reacher respectively. It outputs the mean and variance with a small trick to prevent zero or infinite variance for out-of-distribution inputs, see Appendix A.2. Instead of predicting the next state, we predict the difference between the current state and the next state, as is common in the MBRL literature. It is trained for 5 epochs before each episode by minimizing the negative log-likelihood using Adam with learning rate 0.001 and batch size 32. The ensemble uses 5 NNs, as in the original paper. For both environments, CEM runs for 5 iterations with a population/elite size of 400/40 and $\alpha = 0.1$.

Three variants of uncertainty-aware models are evaluated: Probabilistic ensemble, Deep BLR, and Deep BLR ensemble. Each NN in the ensembles is trained on a bootstrap sample of the data, as in Chua et al. (2018). We also run the method with a single probabilistic NN to see if the epistemic uncertainty induced by the other methods is necessary for good performance. For Deep BLR, we used $g = 0.1$ for Cartpole and $g = 0.01$ for Reacher. We also tried $g = 1.0$ and $g = 10.0$, but found that it resulted in poor performance. Our hypothesis is that the regularization enforced by a small g leads to a simpler, more stable model. Since the delta-state is predicted, it is reasonable that the prior on the weights is tightly centered around zero.

For implementation, we started with an open-source PyTorch implementation[1] more or less directly ported from the authors' official implementation in TensorFlow. We modified this to use TS1 rather than TS∞ and added Deep BLR.

5.1.3 Results

Figure 5.2 and 5.3 display the mean return per episode on CartPole and Reacher respectively. The poor performance of 1 NN on Cartpole indicates that epistemic uncertainty is helpful for Cartpole. For the rest of the methods, we see similar performance. In particular, Deep BLR performs as well as PE with 5 NN, which indicates that Deep BLR is able to capture uncertainty that is helpful for model-based RL. Deep BLR ensemble performs slightly better than the other methods, lending evidence to the hypothesis that ensembling improves uncertainty estimation. This also proves the perhaps obvious fact that both Deep BLR and PE have room for improvement.

On Reacher, there is little differentiation between the methods, and all methods perform roughly equally well. Deep BLR ensemble seems to have a slight edge, but the performance of 5 NN is much more stable than the Deep BLR methods. Unlike Cartpole, 1 NN performs almost as well as the other methods. This might explain the similar performance between Deep BLR ensemble and 5 NN: improved uncertainty estimation doesn't help much in this environment. Further study of the effect of replacing ensembles with Deep BLR should therefore be done in environments where a clear benefit of ensembling, and thus epistemic uncertainty, can be seen.

As touched upon in Section 3.5, Deep BLR can add a significant overhead when the NNs are small enough to be trained in parallel, and this is such a case. PETS with Deep BLR takes almost twice as long as with PE, and this can be attributed to the additional computation of the posterior predictive distribution, which essentially amounts to computing $Z^T V_N Z$. Using samples from the posterior instead may help speed this up, especially in the case of TS∞, but potentially also for TS1. Since the model is updated only at the beginning of each episode, we could obtain a large number of samples from the posterior after training and then use them throughout the episode.

5.1.4 Discussion and further work

The main conclusion we draw from this experiment is that Deep BLR can be competitive with ensemble methods when used in uncertainty-aware MBRL. Furthermore, combining Deep BLR and ensembles may yield superior performance. However, PETS with Deep BLR displayed unstable performance on one of the environments, where we saw some episodes with low return. We see potential for future work in

[1] `https://github.com/quanvuong/handful-of-trials-pytorch`

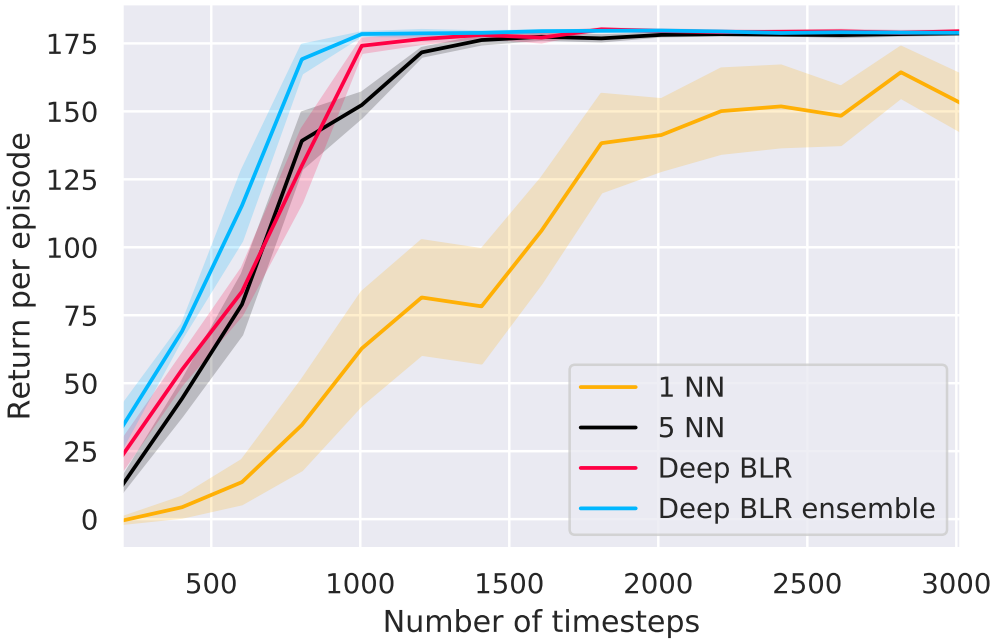


Figure 5.2: Comparison of 1 probabilistic NN, 5 probabilistic NNs, Deep BLR, and Deep BLR ensemble when used as an uncertainty-aware model for PETS on CartPole. The experiment is repeated 10 times and the mean return for each of the 15 episodes is reported, with the shaded area representing one standard error.

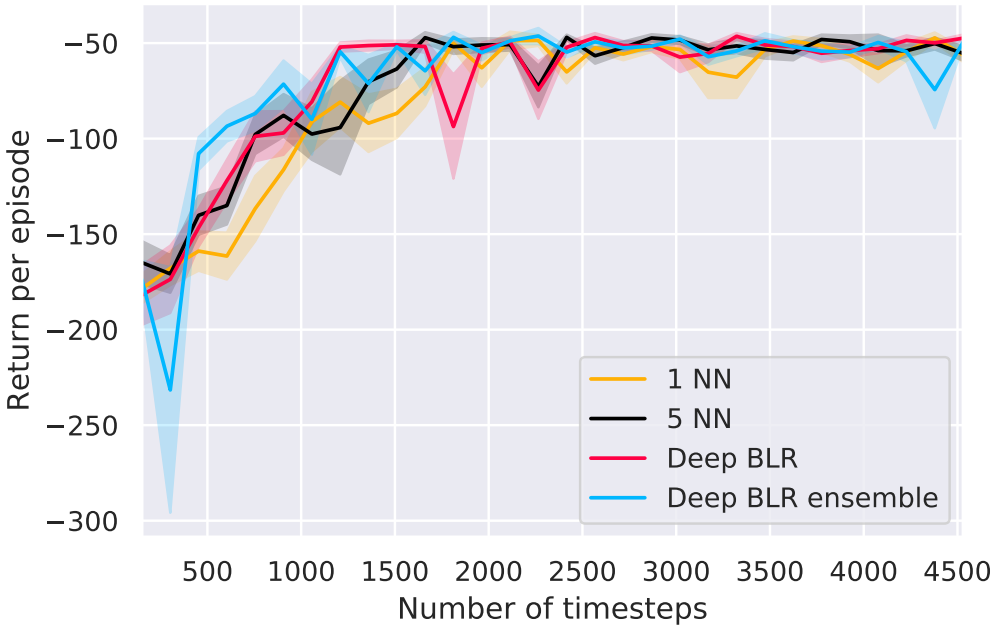


Figure 5.3: Comparison of 1 probabilistic NN, 5 probabilistic NNs, Deep BLR, and Deep BLR ensemble when used as an uncertainty-aware model for PETS on Reacher. The experiment is repeated 10 times and the mean return for each of the 30 episodes is reported, with the shaded area representing one standard error.

two main areas, evaluating on more difficult environments and tuning NN hyperparameters for Deep BLR, which we discuss in the next paragraphs.

The obvious next step is to evaluate these methods on more difficult environments, e.g., Half-Cheetah in which Kurutach et al. (2018) showed good performance for PETS. As previously mentioned, such an evaluation should be done in environments where a clear benefit from accurate uncertainty estimation can be seen. We did not do this due to lack of time and resources. The planning phase of PETS is computationally expensive, and it is unclear how well it performs on even more complex environments, e.g., Humanoid walking or more complex robotics tasks. An alternative uncertainty-aware MBRL algorithm is ME-TRPO (Kurutach et al., 2018), where replacing PE by Deep BLR could also be investigated. Since ME-TRPO has been shown to work on e.g. Humanoid, this might be a better way to study Deep BLR's capacity to model the uncertainty of more complex environments.

Another interesting avenue for research is to evaluate the relationship between the NN architecture and Deep BLR performance. In this experiment, we used the NN architecture as-is, but optimizing for Deep BLR performance may yield significantly better performance. Studying the relationship between the prior variance g and the underlying deep representation would also be interesting.

6

Conclusions and Further Research

In this final chapter, we end with conclusions from our work and discuss potential for further research.

6.1 Conclusions

In this thesis, we have evaluated five methods for obtaining uncertainty-aware NNs, first directly in terms of supervised learning, and then indirectly in terms of downstream performance when used as a component in a model-based RL algorithm.

The first part involved visual inspection of the predictive distributions on 1-D and 2-D regression problems, a qualitative comparison on standard regression datasets, and studying the uncertainty estimates for out-of-distribution inputs. In the second part, Deep BLR and Deep BLR ensembles were compared to PE as uncertainty-aware models in the model-based RL algorithm PETS (Chua et al., 2018).

We found that all methods yield visually reasonable uncertainty estimates on 1-D and 2-D problems. In particular, our variation of Deep BLR did so, despite its simplicity. In our evaluation on standard datasets, we found that MC-dropout was underwhelming, Deep BLR was competitive, and cSGMCMC and Deep BLR ensembles were the strongest contenders. PE and cSGMCMC performed similarly, but cSGMCMC allows and requires much more hyperparameter tuning. Finally, all methods, with the exception of MC-dropout, estimate consistently high uncertainty for out-of-distribution inputs. Combined with the results on standard dataset, this led us to view MC-dropout as the worst method considered.

Our evaluation of the downstream RL performance indicate that Deep BLR is competitive with PE as an uncertainty-aware model. Furthermore, Deep BLR ensemble may outperform PE in environments where the capacity to model epistemic uncertainty is important. On the other hand, Deep BLR yielded slightly less stable performance and is computationally more expensive when the full posterior predictive distribution is used.

To conclude, our main finding is that Deep BLR is a reliable method for produc-

ing uncertainty-aware NNs[1] that may outperform PE, which is the most common method used today. We want to emphasize that, in all our comparisons with other methods, Deep BLR has been applied with *no changes to the NN architecture*. It is possible, and even likely, that Deep BLR performs better with method-specific adjustment of the NN.

6.2 Further research

In Section 3.7, we surveyed some of the methods that we didn't consider in this thesis. Out of these, we find Noisy K-FAC (Zhang et al., 2018) and fvBNN (Sun et al., 2019) the most promising. Based on our results in Section 4.2, cSGMCMC (Zhang et al., 2019) may also be a promising method for uncertainty-aware models, although we didn't have time to evaluate it. We think that a comprehensive downstream comparison of these methods, Deep BLR, and PE in RL would be interesting, including both for exploration and model-based RL.

Furthermore, a major hurdle in evaluating methods for uncertainty-aware NNs is implementation. Developing a software library that implements state-of-the-art uncertainty estimation techniques would be a huge boon for the field, similar to what OpenAI Baselines[2] did for RL. Similarly, designing environments that facilitate benchmarking uncertainty estimation would be useful.

Finally, we would like to see Deep BLR evaluated as an uncertainty-aware model in a wider range of MBRL algorithms, e.g., ME-TRPO (Kurutach et al., 2018), and on more difficult environments where accurate uncertainty estimation is important. On a more general note, studying ways in which uncertainty-aware models can be used for safe RL is also interesting, as in, for example, Kahn et al. (2017).

[1] We remind the reader of the limits of Deep BLR discussed in Section 3.4.4. It can realistically only be applied to NNs with a linear output layer, which prevents it from being used with a convolutional output layer, i.e., in most applications with image outputs.

[2] `https://github.com/openai/baselines`

Bibliography

Kamyar Azizzadenesheli, Emma Brunskill, and Animashree Anandkumar. Efficient exploration through Bayesian deep Q-networks. In *2018 Information Theory and Applications Workshop, ITA 2018*. Institute of Electrical and Electronics Engineers Inc., 10 2018. ISBN 9781728101248. doi: 10.1109/ITA.2018.8503252.

Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D. Goodman. Pyro: Deep Universal Probabilistic Programming. 10 2018. URL `http://arxiv.org/abs/1810.09538`.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight Uncertainty in Neural Networks. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

Z.I. Botev, D.P. Kroese, R. Y. Rubinstein, and P. L'Ecuyer. *The cross-entropy method for optimization*, volume 31. Elsevier, 2013.

Tianqi Chen, Emily B. Fox, and Carlos Guestrin. Stochastic Gradient Hamiltonian Monte Carlo. In *Proceedings of the 31st International Conference on Machine Learning*, 2014.

Youngmin Cho and Lawrence K Saul. Kernel Methods for Deep Learning. In *Advances in Neural Information Processing Systems 22*, 2009.

Kurtland Chua, Roberto Calandra, Rowan Mcallister, and Sergey Levine. Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models. In *32nd Conference on Neural Information Processing Systems*, 2018.

Andreas C. Damianou and Neil D. Lawrence. Deep Gaussian Processes. In *Proceedings of the 16th International Conference on Artificial Intelligence and Statistics*, 2012. URL `http://arxiv.org/abs/1211.0358`.

DeepMind. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II, 2019. URL `https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/`.

Stefan Depeweg, José Miguel Hernández-Lobato, Finale Doshi-Velez, and Steffen Udluft. Learning and Policy Search in Stochastic Dynamical Systems with Bayesian Neural Networks. In *International Conference on Learning Representations (ICLR)*, 2017.

Stefan Depeweg, José Miguel Hernández-Lobato, Finale Doshi-Velez, and Steffen Udluft. Decomposition of Uncertainty in Bayesian Deep Learning for Efficient and Risk-sensitive Learning. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

Dheeru Dua and Casey Graff. UCI Machine Learning Repository, 2017. URL `http://archive.ics.uci.edu/ml`.

Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In *Proceedings of the 33rd International Conference on Machine Learning*, 2016.

Yarin Gal, Jiri Hron, and Alex Kendall. Concrete Dropout. In *31st Conference on Neural Information Processing Systems*, 2017.

Mohammad Ghavamzadeh, Shie Mannor, Joelle Pineau, and Aviv Tamar. Bayesian Reinforcement Learning: A Survey. *Foundations and Trends in Machine Learning*, Vol. 8(No. 5-6), 2015.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

Alex Graves. Practical Variational Inference for Neural Networks. In *Advances in Neural Information Processing Systems 24*, 2011.

Fredrik K. Gustafsson, Martin Danelljan, and Thomas B. Schön. Evaluating Scalable Bayesian Deep Learning Methods for Robust Computer Vision. 6 2019. URL `http://arxiv.org/abs/1906.01620`.

David Ha and Jurgen Schmidhuber. World Models. 2018.

James Hensman, Nicoì O Fusi, and Neil D Lawrence. Gaussian Processes for Big Data. In *Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence*, 2013.

José Miguel Hernández-Lobato and Ryan P. Adams. Probabilistic Backpropagation for Scalable Learning of Bayesian Neural Networks. In *Proceedings of the 33rd International Conference on Machine Learning*.

José Miguel Hernández-Lobato, Yingzhen Li, Mark Rowland, Daniel Hernández-Lobato, Daniel Hernandez Es, Thang D Bui, and Richard E Turner. Black-Box α-Divergence Minimization. In *Proceedings of the 33rd International Conference on Machine Learning*, 2016.

Eric Jones, Travis Oliphant, and Pearu Peterson. SciPy: Open source scientific tools for Python, 2001. URL `http://www.scipy.org/`.

Gregory Kahn, Adam Villaflor, Vitchyr Pong, Pieter Abbeel, and Sergey Levine. Uncertainty-Aware Reinforcement Learning for Collision Avoidance. 2017.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, George Tucker, and Henryk

Michalewski. Model-Based Reinforcement Learning for Atari. 3 2019. URL http://arxiv.org/abs/1903.00374.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. In *International Conference on Learning Representations*, 2017.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*, 2015.

Diederik P Kingma and Max Welling. Auto-Encoding Variational Bayes. In *The 2nd International Conference on Learning Representations*, 2013.

Anoop Korattikara, Vivek Rathod, Kevin Murphy, and Max Welling. Bayesian Dark Knowledge. In *29th Conference on Neural Information Processing Systems*, 2015.

Volodymyr Kuleshov, Nathan Fenner, and Stefano Ermon. Accurate Uncertainties for Deep Learning Using Calibrated Regression. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-Ensemble Trust-Region Policy Optimization. In *International Conference on Learning Representations*, 2018.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. In *31st Conference on Neural Information Processing Systems*, 2017.

Gilwoo Lee, Brian Hou, Aditya Mandalika, Jeongseok Lee, Sanjiban Choudhury, and Siddhartha S. Srinivasa. Bayesian Policy Optimization for Model Uncertainty. 10 2018a. URL http://arxiv.org/abs/1810.01014.

Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S. Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep Neural Networks as Gaussian Processes. In *International Conference on Learning Representations,*, 2018b.

Joel Lehman, Jeff Clune, Dusan Misevic, Christoph Adami, Lee Altenberg, Julie Beaulieu, Peter J. Bentley, Samuel Bernard, Guillaume Beslon, David M. Bryson, Patryk Chrabaszcz, Nick Cheney, Antoine Cully, Stephane Doncieux, Fred C. Dyer, Kai Olav Ellefsen, Robert Feldt, Stephan Fischer, Stephanie Forrest, Antoine Frénoy, Christian Gagné, Leni Le Goff, Laura M. Grabowski, Babak Hodjat, Frank Hutter, Laurent Keller, Carole Knibbe, Peter Krcah, Richard E. Lenski, Hod Lipson, Robert MacCurdy, Carlos Maestre, Risto Miikkulainen, Sara Mitri, David E. Moriarty, Jean-Baptiste Mouret, Anh Nguyen, Charles Ofria, Marc Parizeau, David Parsons, Robert T. Pennock, William F. Punch, Thomas S. Ray, Marc Schoenauer, Eric Shulte, Karl Sims, Kenneth O. Stanley, François Taddei, Danesh Tarapore, Simon Thibault, Westley Weimer, Richard Watson, and Jason Yosinski. The Surprising Creativity of Digital Evolution: A Collection of Anecdotes from the Evolutionary Computation and Artificial Life Research Communities. 3 2018. URL http://arxiv.org/abs/1803.03453.

Felix Leibfried, Nate Kushman, and Katja Hofmann. A Deep Learning Approach for Joint Video Frame and Reward Prediction in Atari Games. In *ICML 2017 Workshop on Principled Approaches to Deep Learning*, 2017.

Christos Louizos and Max Welling. Structured and Efficient Variational Deep Learning with Matrix Gaussian Posteriors. 2016. ISSN 1938-7228. doi: 10.1080/02757540.2016.1162297.

Fangchang Ma, Guilherme Venturelli Cavalheiro, and Sertac Karaman. Self-supervised Sparse-to-Dense: Self-supervised Depth Completion from LiDAR and Monocular Camera. 7 2018. URL `http://arxiv.org/abs/1807.00275`.

Rowan Mcallister, Gregory Kahn, Jeff Clune, and Sergey Levine. Robustness to Out-of-Distribution Inputs via Task-Aware Generative Uncertainty, 2018. URL `https://arxiv.org/abs/1812.10687`.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. 2013. ISSN 0028-0836. doi: 10.1038/nature14236.

Kevin Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

Radford Neal. MCMC Using Hamiltonian Dynamics. In *Handbook of Markov Chain Monte Carlo*, chapter 5. Chapman & Hall / CRC Press, 2011.

Radford M. Neal. *Bayesian Learning for Neural Networks*, volume 118 of *Lecture Notes in Statistics*. Springer New York, New York, NY, 1996. ISBN 978-0-387-94724-2. doi: 10.1007/978-1-4612-0745-0. URL `http://link.springer.com/10.1007/978-1-4612-0745-0`.

OpenAI. OpenAI Five, 2019. URL `https://openai.com/five/`.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep Exploration via Bootstrapped DQN. 2016a. ISSN 10495258. doi: 10.1145/2661829.2661935.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep Exploration via Bootstrapped DQN. In *Advances in Neural Information Processing Systems 29*, 2016b.

Ian Osband, John Aslanides, and Albin Cassirer. Randomized Prior Functions for Deep Reinforcement Learning. In *32nd Conference on Neural Information Processing Systems*, 2018.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary Devito Facebook, A I Research, Zeming Lin, Alban Desmaison, Luca Antiga, Orobix Srl, and Adam Lerer. Automatic differentiation in PyTorch. Technical report.

Tim Pearce, Mohamed Zaki, Alexandra Brintrup, and Andy Neel. Uncertainty in Neural Networks: Bayesian Ensembling. 2018.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for Activation Functions. 10 2017. URL `http://arxiv.org/abs/1710.05941`.

C. E. Rasmussen and C. K. I Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

Carlos Riquelme, George Tucker, and Jasper Snoek. Deep Bayesian Bandits Showdown: An Empirical Comparison of Bayesian Deep Networks for Thompson Sampling. In *6th International Conference on Learning Representations*, 2018.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. 2017.

Pranav Shyam, Wojciech Jaśkowski, and Faustino Gomez. Model-Based Active Exploration. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George Van Den Driessche, Thore Graepel, and Hassabis Demis. Mastering the Game of Go without Human Knowledge. *Nature*, 2017.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian Optimization of Machine Learning Algorithms. In *Advances in Neural Information Processing Systems 25*, 2012.

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Md. Mostofa Ali Patwary, Prabhat, and Ryan P. Adams. Scalable Bayesian Optimization Using Deep Neural Networks. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. ISSN 1533-7928.

Shengyang Sun, Changyou Chen, and Lawrence Carin. Learning Structured Weight Uncertainty in Bayesian Neural Networks. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

Shengyang Sun, Guodong Zhang, Jiaxin Shi, and Roger Grosse. Functional Variational Bayesian Neural Networks. In *International Conference on Learning Representations*, 2019.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

Laurens van der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE . *Journal of Machine Learning Research*, 9:2579–2605, 2008. URL `http://www.jmlr.org/papers/v9/vandermaaten08a.html`.

Max Welling and Teh Yee Whye. Bayesian Learning via Stochastic Gradient Langevin Dynamics. In *Proceedings of the 28th International Conference on Machine Learning*, 2011.

Guodong Zhang, Shengyang Sun, David Duvenaud, and Roger Grosse. Noisy Natural Gradient as Variational Inference. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

Ruqi Zhang, Chunyuan Li, Jianyi Zhang, Changyou Chen, and Andrew Gordon Wilson. Cyclical Stochastic Gradient MCMC for Bayesian Deep Learning. 2019.

A

Experimental details

A.1 Toy regression

We used the same hyperparameters for each method throughout the toy regression experiments, with the natural exception of figure 4.4 where the number of models used was varied. The following hyperparameters were used:

- **Probabilistic ensemble:** 10 NNs, 5 hidden layers with 10 ReLU neurons per layer. Optimized using Adam with a learning rate of 1e-3 and no weight decay.
- **cSGMCMC ensemble:** 20 cycles with 4 samples per cycle, resulting in 80 NNs with 5 hidden layers and 10 ReLU neurons per layer. $\alpha = 0.5$, $\beta = 0.9$, initial learning rate $\alpha_0 = 1.0$ and prior standard deviation of 2.0.
- **MC-dropout:** 2 hidden layers with 1000 ReLU neurons each. Dropout rate $p = 0.25$ and predictive distribution approximated with 100 stochastic forward passes. Optimized using Adam with a learning rate of 1e-3 and no weight decay.
- **Deep BLR:** 3 hidden layers with 50 ReLU neurons each, so regression on a 50-dimensional representation with $g = 2$. Optimized using Adam with a learning rate of 1e-4 and no weight decay. The ensemble was built using 10 such models.
- **Gaussian process:** ReLU kernel, $\sigma^2 = 0.005$, prior variance of w=10, prior variance of b=500.

A.2 Trick for bounded variance in PETS

Chua et al. (2018) observed that the variance outputted by a probabilistic NN can take arbitrary values for out-of-distribution inputs, for example collapsing to zero or exploding to infinity. It can be fixed by using the following trick.

For each output, two variables `max_logvar` and `min_logvar` are instantiated, with initial values 0.5 and -10 respectively. The output is then given by

$$\begin{aligned}\texttt{logvar} &\leftarrow \texttt{max\_logvar} - \text{softplus}(\texttt{max\_logvar} - \texttt{logvar}) \\ \texttt{logvar} &\leftarrow \texttt{min\_logvar} + \text{softplus}(\texttt{logvar} - \texttt{min\_logvar}) \\ \texttt{var} &\leftarrow \exp(\texttt{logvar})\end{aligned}$$

where $\text{softplus}(x) = \log(1 + e^x)$, a differentiable approximation to $\max(x, 0)$. This ensures that the log-variance, and thus variance, will be bounded. A small regularization penalty is added to `max_logvar` and $-$`min_logvar` so that they don't grow above or below the maximum and minimum variance of the training set.

Unfortunately, we saw this trick too late to incorporate it in our experiments prior to PETS, but believe it is a good idea to utilize it for probabilistic NNs (although we haven't analyzed it in-depth.) Nevertheless, it was used for the PETS experiments.