Universidad Zaragoza
1542

Master's Thesis

Analysis of linearization methods for Bayesian Deep Learning

Author

Carlos Plou Izquierdo

Supervisors

Rubén Martínez Cantín

Ana Cristina Murillo Arnal

ESCUELA DE INGENIERÍA Y ARQUITECTURA
2023

Abstract

Deep learning has revolutionized the field of machine learning, outperforming traditional methods in a wide range of areas such as computer vision, robotics or natural language processing. However, deep learning models are often treated as a black box, providing high accuracy but with little insight into how the model arrived at its predictions. This lack of transparency can be a major drawback in certain applications where the ability to understand and trust the model's predictions is crucial. For instance, in some tasks such as medical diagnosis or autonomous driving, it is as important to achieve high accuracy as it is to have an accurate uncertainty estimation. In these scenarios, a well-calibrated uncertainty estimate can provide valuable information for decision-making and can prevent risky situations.

Bayesian deep learning and specifically Bayesian neural networks try to address the challenge of estimating uncertainty assuming that their parameters and predictions follow a specific distribution instead of deterministic values. This enables the use of Bayesian inference, a statistical technique based on the Bayes rule, that allows for updating our distributions of interest with the arrival of new observations. However, performing Bayesian inference in deep learning algorithms is not easy due to the complexity, dimensionality of the models and the large amounts of data they are trained on. This has led to the development of approximate Bayesian inference methods such as Ensembles and Monte Carlo dropout, which provide approximate solutions to the problem of uncertainty estimation in deep learning. Even so, these methods do not fully solve the overconfidence problem in deep learning models. This has motivated to reconsider in the last years a statistical technique called Laplace approximation which was introduced in 1992 by David Mackay.

The focus of this Master's thesis is to study the opportunities of Laplace approximation in Bayesian deep learning. We conduct a thorough investigation and analysis of this powerful technique that allows us to tackle the overconfidence problem. Additionally, we evaluate its performance on benchmark datasets by comparing it with the most commonly used uncertainty estimation methods such as deep ensembles and Monte Carlo dropout. We demonstrate the effectiveness of Laplace approximation in providing accurate and reliable uncertainty estimates for deep learning models. Specifically, our experiments show that Laplace approximation outperforms the other techniques in estimating uncertainty in out-of-distribution regions.

As an example of real application, this work explores the use of Laplace approximation in the field of reinforcement learning (RL). While Laplace approximation has been widely used in other fields, its potential benefits in RL have yet to be fully explored. Exploration is a crucial problem in RL that involves exploring the environment to maximize an agent's knowledge and, consequently, its ability to solve different tasks. Nevertheless, efficient exploration in high-dimensional environments remains an unsolved challenge. A promising approach is to use the degree of novelty as a reward signal. To address this, we propose integrating Laplace approximation into a model-based RL algorithm that performs active exploration by utilizing estimated uncertainty as novelty measure. Our experiments show that this approach outperforms the baselines.

Acknowledgements

I am truly grateful to Ana and Rubén, my advisors, for their invaluable support and guidance throughout this work. Their dedication and nearness have made it a pleasure to work alongside them, and I am looking forward to continue learning from them.

I would like to extend my thanks to my classmate, Fernando Peña. Since the beginning of the master's program, Fernando has been a constant source of support, helping me to fill the gaps that I had in my Informatics background. Thanks Fernando, you have been my best teacher. Besides, I would also like to recognize the guidance and advice provided by Javier García during a significant part of this work.

Lastly, I would like to mention my colleagues in the lab who have welcomed me and made me feel like a part of the team from day one. They achieve to make the daily routine enjoyable and fulfilling. In particular, I would like to highlight the support provided by León, who has been there since the beginning, always ready to help with any challenges I encountered.

Thanks to all of you,

Carlos Plou Izquierdo.

Contents

Chapter 1

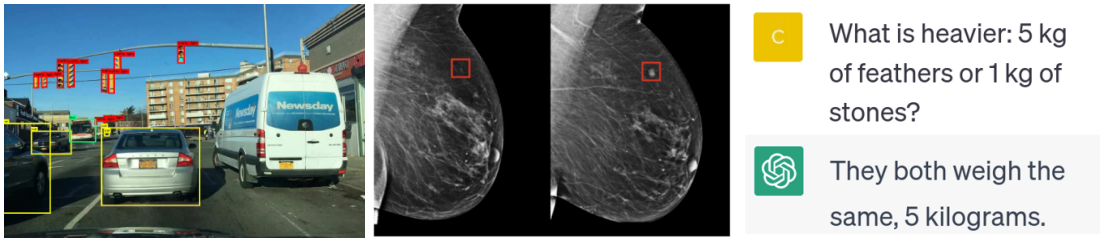
Introduction

1.1 Motivation

Bayesian deep learning-. Deep learning has revolutionized the field of artificial intelligence in recent years. The ability of deep neural networks to learn complex patterns and features from large amounts of data has led to incredible achievements in a variety of applications, including computer vision, robotics or natural language processing. Even so, deep learning models are not infallible yet, and their reliability is crucial in applications such as medical diagnosis and autonomous driving, where the consequences of model's mistakes can be fatal (Figure 1.1). Therefore, measuring the uncertainty of deep learning models is essential to ensure their safety and reliability [1]. In autonomous driving, for example, deep learning models are used to make critical decisions such as when to brake or turn. Incorporating uncertainty in the predictions could help the car to make more cautious decisions when the model is uncertain, which can reduce the risk of accidents [2, 3]. Other scenario where measuring uncertainty could be crucial is language models. Despite their impressive performance, they often provide unreliable or false responses. Therefore, quantifying how sure the model is about its predictions could be extremely helpful to users.

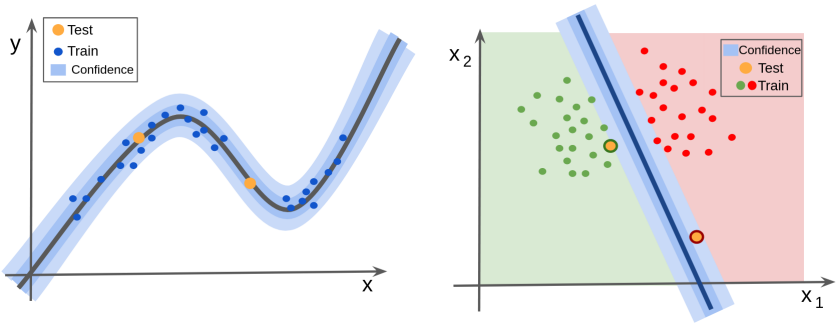
One approach for incorporating uncertainty in deep learning models is Bayesian deep learning [4, 5]. Bayesian deep learning is a subfield of machine learning that combines the principles of Bayesian statistics and deep learning. Recently, different techniques of Bayesian deep learning have appeared to deal with the uncertainty estimation problem from different perspectives [6, 7, 8]. Consequently, many works have explored these techniques in different topics such as computer vision, reinforcement learning or Graphics [9, 10, 11]. In addition to providing uncertainty estimates, Bayesian deep learning can also improve the generalization and robustness of the models. By incorporating prior information into the model, it can prevent overfitting -i.e when a model fits accurately on the training data, but performs poorly on unseen data- and make the model more robust to changes in the data distribution [12].

Laplace approximation-. Most of the Bayesian techniques that estimate model's uncertainty are overconfident. The overconfidence problem refers to the scenario where the model's estimated uncertainty is lower than it should be for new data that is out-of-distribution -i.e quite different from the data which the model was trained on- (Figure 1.2). To address this issue, the statistical technique known as Laplace approximation, originally introduced by David Mackay in 1992 [13], has gained increasing attention in recent years [14, 15, 16]. Laplace approximation is a powerful method that approximates the posterior distribution of model's parameters through a Gaussian distribution, allowing for inference and avoiding overconfidence.



(a) Object detector in AV [2]. (b) Breast cancer DL detection. (c) Language model invent.

Figure 1.1: Deep learning applications that require high reliability.

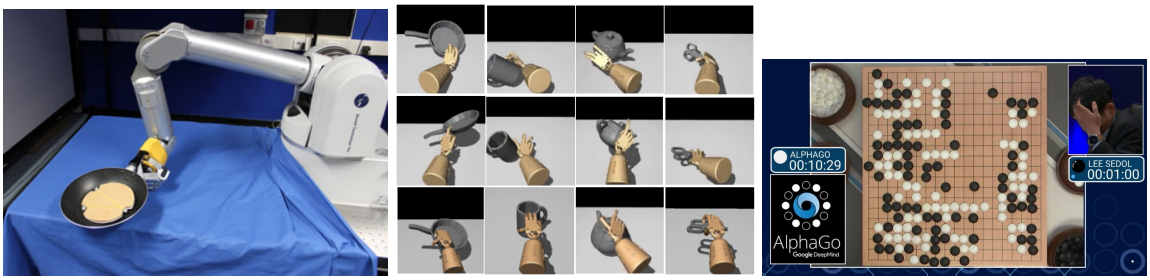


(a) Regression problem. (b) Classification problem.

Figure 1.2: Illustration of the overconfidence problem. In both problems there are two test points in which the uncertainty estimated is the same. Is it correct?.

Recently, it has been released Laplace-Redux library that allows to easily leverage Laplace approximation [17] in popular deep learning models integrated with Pytorch. Many works have shown that Laplace approximation is better calibrated and more effective in various fields as image classification and active learning [18, 19, 20]. However, to the best of our knowledge, Laplace approximation has not yet been applied in the rapidly growing field of reinforcement learning, which has a broad range of potential applications. As result, in this work, we decided to explore its capability in this field.

Active exploration in RL-. Reinforcement learning (RL) is a subfield of machine learning that deals with agents learning how to take actions in an environment to maximize a reward signal (Figure 1.3). One of the key challenges in this field is exploration, which involves exploring the environment to learn about it and improve the agent's performance at solving different tasks. In model-based RL, where a model is trained to describe the dynamics, exploration is even more crucial so as to collect diverse data for training the model [21, 22].
However, in high-dimensional environments, efficient exploration becomes an unsolved challenge in model-based RL. First approaches were reactive exploration methods which were based on the agent encountering something "novel" by chance and then deciding to investigate it further [23, 24]. Existing formulations of measuring novelty include visitation count [25, 26], prediction error [27, 28] or disagreement among models [29]. However, exploration can be more effective if it is active, with the agent proactively seeking out novelty based on its own estimate of which action sequences will lead to interesting outcomes. Specifically, our work is inspired by the framework developed in [30] which actively measures novelty as disagreement among the next state predictions of a set of models.



(a) Pancake flipping by robot. (b) Learning to grasp objects. (c) RL wins playing Go.

Figure 1.3: Three examples of applications of reinforcement learning.

This formulation suggest that active exploration problem could be optimally formulated by taking the model uncertainty estimated by Laplace approximation as novelty measure.

1.2 Goal and tasks

The main objective of this master's thesis is to conduct a comprehensive study and analysis of the Laplace approximation, comparing it to other uncertainty estimation techniques that are commonly used in deep learning. Furthermore, we aim to demonstrate the versatility of Laplace approximation by showing its applicability in complex deep learning problems, beyond some benchmarks. Specifically, we will apply Laplace approximation to the active exploration problem in reinforcement learning. In order to achieve the planned goals, the following tasks were executed throughout the project (its distribution along the months may be observed in Table 1.1):

1. A thorough study of state-of-the-art methods for uncertainty estimation in deep learning algorithms. This task aimed to provide a clear understanding of the current state of the art in the field and to identify areas where Laplace approximation can be particularly useful.

2. Unifying theoretical concepts and exploring the boundaries of the Laplace approximation. In this task, the theoretical foundations of the Laplace approximation were solidified, its strengths and limitations were identified, and recent solutions that have been proposed to cope with its limitations were studied.

3. Testing the approximation with benchmark datasets and analyzing the results. This task aimed to evaluate the performance of the Laplace approximation on several tasks with different levels of complexity, and to compare its performance to other baselines such as Ensemble or MC dropout. Besides, we explore different variations of Laplace approximation that attempt to overcome its limitations, in order to determine which approach is the best suited for more complex problems.

4. Integrating and evaluating the Laplace approximation in a reinforcement learning problem. The purpose of this task is to demonstrate the applicability of the Laplace approximation in the context of a more complex problem, as model-based active exploration. In this topic, the uncertainty of the model is used as reward to actively explore the states space and, then, build the agent to solve different tasks.

We assess the performance of this approach in different environments and compare it with models that estimate uncertainty using alternative Bayesian techniques, as well as models with different underlying architectures.

5. Documentation and intermediate presentations. Write a comprehensive report detailing the work done, the methodology followed, and the results obtained. This report also includes a thorough review of the literature relevant to the project, as well as a discussion of the implications, limitations and potential future steps of the research. Complementary, give several intermediate presentations to my advisors and colleagues throughout the project, allowing for feedback and discussion to ensure the quality and rigor of the research conducted.

Task	Oct	Nov	Dec	Jan	Feb	Apr
Study						
Exploring boundaries						
Benchmarks						
RL Application						
Documentation						

Table 1.1: Gantt diagram representing the distribution of the tasks done along the months.

1.3 Contribution

The contribution of this master's thesis may be grouped in the next points:

- We provide a **comprehensive evaluation of main Bayesian uncertainty estimation techniques including MC dropout, deep ensembles and different approaches of Laplace approximation**, in two regression datasets. One of these datasets is created by us to provide a better understanding of uncertainty. Additionally, we have developed a Python notebook that enables users to explore and experiment with all these configurations and intuitively visualize the main differences among them `https://github.com/cplou99/BayesianDL`.

- We **integrate Laplace approximation into the active exploration problem in reinforcement learning**. To the best of our knowledge, this is the first time Laplace approximation has been applied to this problem. Our experiments demonstrate that the performance of our proposed method outperforms other models that estimate uncertainty using alternative Bayesian techniques in certain environments. These results show the potential of Laplace approximation in addressing the challenging problem of efficient exploration in high-dimensional spaces.

1.4 Context and tools

This work was developed in the Robotics, Perception and Real-time group at the University of Zaragoza, in the Institute of Engineering and Research of Aragon (i3A). In fact, this project serves as the starting point for my PhD thesis and will be further developed in future research.

The main programming language used for this project was `Python`, using the `PyCharm` integrated development environment. The experiments were run using popular machine learning libraries such as `numpy` (base version 1.23.4) and `PyTorch` (version 1.12.1). Additionally, the project delved into the novel `Laplace` library (version 0.1a2) which is specifically designed to exploit the power of the Laplace approximation [17]. To address the reinforcement learning problem, the python library `gym` (version 0.13.1) [31] was used and the environments were implemented using the physics engine `mujoco-py` (version 2.1.2.14) [32]. Specifically, the code setup of the RL problem was inspired by [30].

The hardware used to run the experiments was a local machine with a NVIDIA GeForce RTX 3090 GPU. `PyTorch` library leverages this high-performance GPU allowing for efficient and fast execution of the experiments carried out in this master's thesis. Furthermore, in order to optimize these resources and identify the main bottlenecks in terms of computational resources, we use `SCALENE` library [33] to profile the implemented code.

The project documentation was written with the widely-used and well-established typesetting system LaTeX and it was edited using `Overleaf`.

1.5 Project structure

The master's thesis documentation is divided into the following chapters:

- Chapter 1 of the project, the introduction, provides an overview of the context, tools, objectives, contribution and tasks of the project.
- Chapter 2 of the project is dedicated to Bayesian inference. It begins with a recap of the background concepts such as Bayes theorem or supervised learning. Afterwards, it focuses on Bayesian deep learning, motivating its use and detailing the main Bayesian inference techniques in deep learning, such as deep ensembles and MC dropout. The section concludes by introducing the Laplace approximation and motivating its use as a powerful method for approximating the posterior distribution of the model's parameters.
- Chapter 3 focuses on the Laplace approximation. It covers the theoretical proofs and provides results from benchmark datasets. The chapter also compares the performance of the Laplace approximation to other baselines, providing a thorough analysis of the method's strengths and limitations. Besides, it analyzes the performance of different variations of Laplace approximation that attempt to overcome these limitations.
- Chapter 4 covers the active exploration RL problem. The chapter presents the problem and explains the role of the uncertainty measured by Laplace approximation. Results from different environments are shown, comparing its performance with respect to estimating uncertainty using alternative Bayesian techniques, as well as the state-of-the-art methods in this problem. Overall, we provide a comprehensive analysis of the method's effectiveness in the context of reinforcement learning.
- Chapter 5 presents the conclusions drawn from the obtained results, as well as a discussion on the limitations of the work and possible future steps to be taken.

Chapter 2

Bayesian Inference

2.1 Background

2.1.1 Probability basics

Probability is a mathematical concept that is used to describe and analyze random events. There are two main interpretations of probability: the frequentist interpretation and the Bayesian interpretation.

The **frequentist** interpretation of probability is based on the idea of long-term frequency of occurrence. In this interpretation, the probability of an event is defined as the limit of the relative frequency of the event occurring in a sequence of independent trials, as the number of trials approaches infinity. For example, if we say that the probability of a coin landing heads is 0.5, we mean that if you flip a coin 1000 times, it should land heads about 500 times.

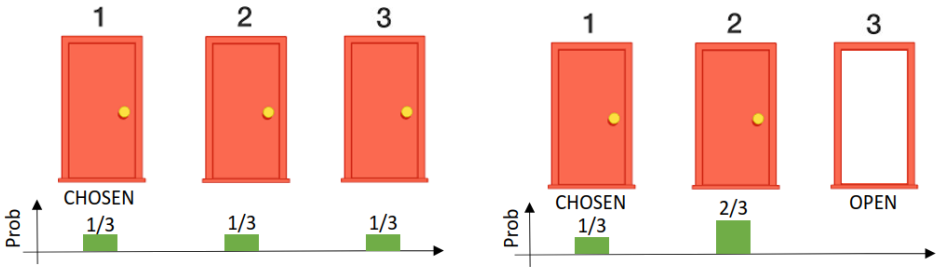
The **Bayesian** interpretation of probability is based on the idea of subjective degree of belief. In this interpretation, probability is not a measure of frequency, but rather a measure of our **uncertainty** or ignorance about something. In the coin scenario, the statement would mean that we believe the coin is equally likely to land heads or tails on the next toss. From here on, we will refer to probability from this perspective.

Bayesian inference

Let $\mathbf{X}$ be a random variable that takes values in the state space $\mathcal{X}$, we denote its probability mass/density function as $p(x)$, $x \in \mathcal{X}$. **Bayesian inference** refers to the process of updating a distribution over unknown values of some quantity of interest $p(\mathbf{X} = x)$, given relevant observed data $\mathbf{Y} = y$. It is based on **Bayes theorem**,

$$p(\mathbf{X} = x | \mathbf{Y} = y) = \frac{p(\mathbf{Y} = y \mid \mathbf{X} = x) p(\mathbf{X} = x)}{p(\mathbf{Y} = y)}. \tag{2.1}$$

Each term of (2.1) plays a meaningful role in Bayesian inference. First, $p(\mathbf{X} = x)$ receives the name of **prior distribution**. It represents the prior knowledge about $\mathbf{X}$ before taking any observation. Similarly, $p(\mathbf{Y} = y)$ represents the probability of observed data, also known as **evidence**. In high dimensional continuous problems its computation is often infeasible. The conditional probability of having observed such data assuming that our variable of interest take the unknown values $p(\mathbf{Y} = y \mid \mathbf{X} = x)$ is known as **likelihood** function. As result, we get the **posterior distribution** $p(\mathbf{X} = x | \mathbf{Y} = y)$ which represents the updated knowledge about $\mathbf{X}$ after observations.



(a) Prior distribution. (b) Posterior distribution.

Figure 2.1: Overview of Monty Hall problem when you choose door 1 and door 3 is opened.

Example 1. The Monty Hall problem is a probability puzzle that was first introduced on the American television show "Let's Make a Deal" (Figure 2.1). It is named after the show's original host, Monty Hall. Here's how the puzzle works: You are a contestant on a game show and are presented with three doors. Behind one of the doors is a prize, while behind the other two are goats. You are asked to choose a door, and you do so. The game show host, knowing what is behind each door, then opens one of the other two doors, revealing a goat. The host then asks you if you would like to switch your choice to the other remaining door. The question is, should you switch your choice?

Our variable of interest is the door x_i which hides the prize $\mathbf{X} = \{x_1, x_2, x_3\}$. Theoretically, its prior distribution is equally spread between the three doors. Besides, the observation you will receive is the door y_j that they will open to you after you choose one of them $\mathbf{Y} = \{y_1, y_2, y_3\}$. So, let us suppose that you choose door number 1 (the other two scenarios would be equivalent), then the likelihood of the different feasible observations (y_2, y_3) are:

$$\Pr(y_2 \mid x_1) = \Pr(y_3 \mid x_1) = \frac{1}{2},\ \Pr(y_3 \mid x_2) = \Pr(y_2 \mid x_3) = 1,\ \Pr(y_2 \mid x_2) = \Pr(y_3 \mid x_3) = 0.$$

Now, let us assume $\mathbf{Y} = y_3$ (again the other scenario is equivalent). Then, the evidence is

$$\Pr(\mathbf{Y} = y_3) = \Pr(y_3 \mid x_1)\Pr(x_1) + \Pr(y_3 \mid x_2)\Pr(x_2) + \Pr(y_3 \mid x_3)\Pr(x_3) = \frac{1}{2}\cdot\frac{1}{3} + 1\cdot\frac{1}{3} + 0\cdot\frac{1}{3} = \frac{1}{2}.$$

Therefore, the posterior distribution is:

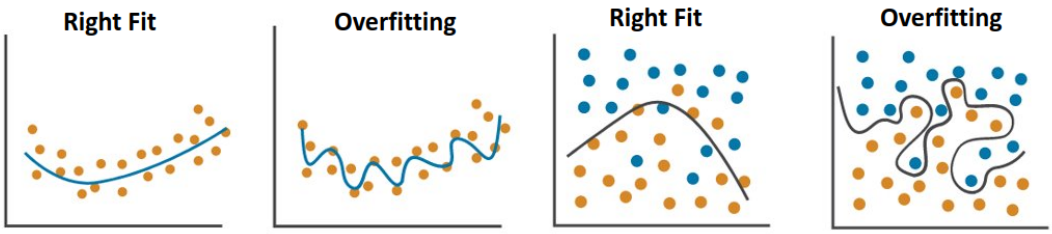
$$\Pr(x_1 \mid y_3) = \frac{\Pr(y_3 \mid x_1)\Pr(x_1)}{\Pr(\mathbf{Y} = y_3)} = \frac{1}{3},\ \Pr(x_2 \mid y_3) = \frac{\Pr(y_3 \mid x_2)\Pr(x_2)}{\Pr(\mathbf{Y} = y_3)} = \frac{2}{3},\ \Pr(x_3 \mid y_3) = 0.$$

So, it is twice more likely to win if you switch your initial choice!

Uncertainty

In Bayesian inference, uncertainty is a fundamental concept that describes the lack of definite knowledge about the true values of our variable of interest or just the parameters of our model. There are two types of uncertainty that are typically considered in Bayesian inference: **epistemic uncertainty** and **aleatoric uncertainty**.

Epistemic or model uncertainty refers to the uncertainty that arises from a lack of knowledge or information about the system being modeled. This type of uncertainty can be reduced as more data is collected or more information is acquired about the system.



(a) Regression dataset. (b) Classification dataset.

Figure 2.2: Overfitting example in a regression and a classification problem.

For example, if we are trying to estimate the true mean of a population, our estimate will be more uncertain if we only have a small sample of data compared to if we have a larger sample.

Aleatoric or data uncertainty, on the other hand, refers to the uncertainty that arises from random variations or noise in the system. This type of uncertainty cannot be reduced by collecting more data or acquiring more information. For example, if we are trying to predict the weather, there will always be some uncertainty in our predictions due to random fluctuations in the atmosphere.

In Bayesian inference, we typically try to model both uncertainties.

2.1.2 Supervised Learning

Supervised learning is a field of machine learning whose algorithms are trained on a labeled dataset, which contains D inputs or features $\mathbf{X} = \{\mathbf{x}_n\}_{n=1}^N \in \mathbb{R}^{N\times D}$ and the corresponding correct output or target $\mathbf{y} = \{y_n\}_{n=1}^N$. Their goal is to learn a mapping from inputs to outputs, so that it is able to make predictions on new data.

We will refer to the dataset as $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N = (\mathbf{X}, \mathbf{y})$. Usually, we divide data in three disjoint sets that play a specific role. The **training set** is used to train the model. The model learns the relationship between the features (inputs) and the outputs minimizing a **loss function**. The **validation set** is used to evaluate the model while it is being trained. This is used to tune the hyperparameters of the model. The **test set** is used to evaluate the performance of the final, trained model by way of some metrics. This gives an estimate of the performance of the model on unseen data. It is important to keep the test set separate and not use it for training or validation in order to avoid **overfitting**, which occurs when a model fits accurately on the training data, but performs poorly on unseen data (Figure 2.2).

There are two main problems within supervised learning: regression and classification. A detailed description of these problems can be seen in Appendix A.1. There, we define several key concepts of supervised learning that will be used in this work from here on.

2.2 Bayesian Deep Learning

Deep learning is a subset of machine learning that uses neural networks with multiple layers to learn representations of data. These algorithms are most commonly trained to find the maximum likelihood estimation (MLE) of their parameters, that is, they pursue a point estimation of their parameters, ignoring any uncertainty about them. This often leads to overconfident predictions, especially in areas that are weakly covered by training data. This problem is addressed by **Bayesian deep learning** which refers to the application of Bayesian inference to deep learning.

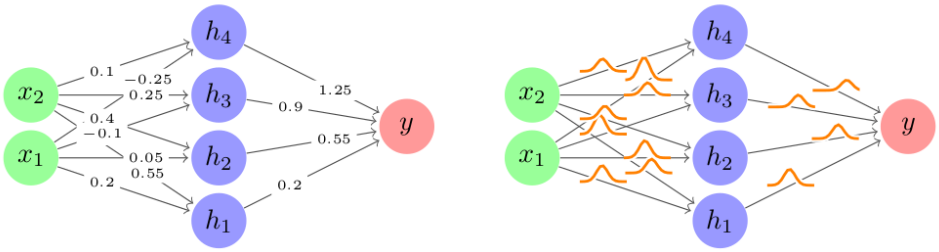


Figure 2.3: Comparison of a MLP with point estimate (left) vs distribution (right) for each weight.

2.2.1 Bayesian neural networks

Bayesian neural networks (BNN) are a popular type of neural network due to their ability to quantify the uncertainty in their predictive weights and output. In contrast to other neural networks, BNN train the model weights as a distribution rather than searching for an optimal value (Figure 2.3). This makes them more robust and allows them to generalize better with less overconfidence.

BNN impose a prior distribution $p(\boldsymbol{\theta})$ on the parameters with the aim of computing their posterior given the data, $p(\boldsymbol{\theta}|\mathcal{D})$; a typical choice is to assume a Gaussian prior $p(\boldsymbol{\theta}) \sim \mathcal{N}(\mathbf{0}, \gamma^{-1}\mathbf{I}_P)$, where γ^2 denotes the prior precision. Consequently, Bayes theorem allows us to compute the posterior distribution.

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{D})} = \frac{\mathcal{L}(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{\int p(\mathcal{D}|\boldsymbol{\theta}')p(\boldsymbol{\theta}')d\boldsymbol{\theta}'}, \tag{2.2}$$

This is never the final step since our variables of interest are not the parameters of the neural network, but the output. Then, once we compute the posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$, we could translate this uncertainty into the predictive variable. Thus, for new inputs $\mathbf{x}_*$, the predictive distribution could be computed by way of,

$$p(\mathbf{y}_*|\mathbf{x}_*, \mathcal{D}) = \mathbb{E}_{p(\boldsymbol{\theta}|\mathcal{D})}\left[p(\mathbf{y}_*|f(\mathbf{x}_*;\boldsymbol{\theta}))\right] = \int p(\boldsymbol{\theta}|\mathcal{D})p(\mathbf{y}_*|f(\mathbf{x}_*;\boldsymbol{\theta}))d\boldsymbol{\theta}. \tag{2.3}$$

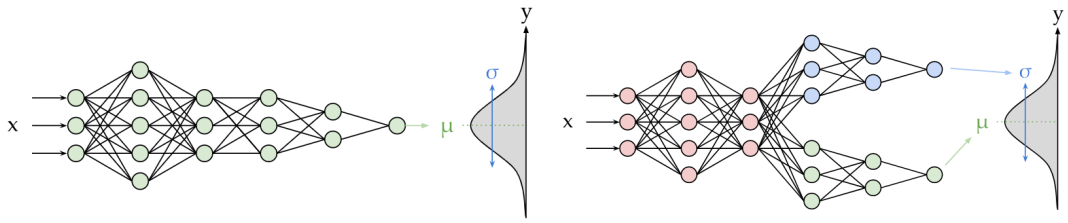
Unfortunately, both computations are only feasible in linear functions and, as we know, current neural networks are complex nonlinear functions. Additionally, we should face two other drawbacks: first, the order of parameters in current neural networks usually scale to millions and, second, the size of datataset used to train them is stunningly high. We therefore have to approximate both computations by way of some inference techniques.

2.2.2 Approximate Bayesian inference

In this section, we take an overview about the main techniques that put the ideas described in section 2.2.1 into practice.

Two “heads” Bayesian neural network

This strategy arises to capture the aleatoric uncertainty in the predicted output. As we discuss in Appendix (A.1), one of the first and strongest assumptions in regression problems is that the model is homoscedastic. This limitation may be addressed adding a "head" at the end of the network to predict the variance. Hence, the predicted output variance is input-dependent (heteroscedastic), rather than a constant.



(a) Simple Bayesian neural network. (b) Two "heads" Bayesian neural network.

Figure 2.4: From `https://brendanhasz.github.io/2019/07/23/Bayesian-density-net.html`. Comparison scheme of homoscedastic and heteroscedastic (two "heads") Bayesian MLP.

This kind of neural networks receive the name of **two "heads" networks** (one head predicts the mean and the other the variance) and are trained minimizing the NLL (Figure 2.4).

Maximum A Posteriori (MAP)

This strategy suggests to introduce the prior knowledge over the parameters into the neural network training. In this way, we could train them to find the **maximum a posteriori** (MAP) estimation of its parameters $\boldsymbol{\theta}_{MAP}$ instead of $\boldsymbol{\theta}_{MLE}$,

$$\boldsymbol{\theta}_{MAP} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\mathcal{D}) = \arg\max_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D}|\boldsymbol{\theta}) p(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \left[-\ell(\mathcal{D}|\boldsymbol{\theta}) - \log p(\boldsymbol{\theta})\right]. \tag{2.4}$$

However, this is still a point estimation of the parameters. In other words, it does not address the problem of approximating both distributions (2.2), (2.3). Other techniques, try to capture model uncertainty in the output introducing certain randomness along the neural network training or, even, prediction.

MC Dropout

Dropout is a regularization technique that randomly sets a portion of the input neurons of a neural network layer to zero during training [34]. Monte Carlo (MC) dropout extrapolates this idea to test time [35]. Hence, it performs several forward passes randomly dropping out different hidden units during each one. As result, it generates multiple predictions for a given input, which can be used to estimate the model's uncertainty -i.e variance among forward passes- and to improve the final predictions -i.e average among forward passes-,

$$p(\mathbf{y}_*|\mathbf{x}_*, \mathcal{D}) = \frac{1}{M} \sum_{m=1}^{M} p(\mathbf{y}_*|\mathbf{x}_*, \boldsymbol{\theta}^m),$$

where $\boldsymbol{\theta}^m$ is a version of $\boldsymbol{\theta}_{MAP}$ where some connections are dropped out with probability p. This technique has been shown to be effective in a wide range of contexts, reducing overfitting and improving the uncertainty estimation [36, 9, 37]. However, it requires multiple forward passes through the network with different dropout masks during inference, which can be computationally expensive, especially for large models.

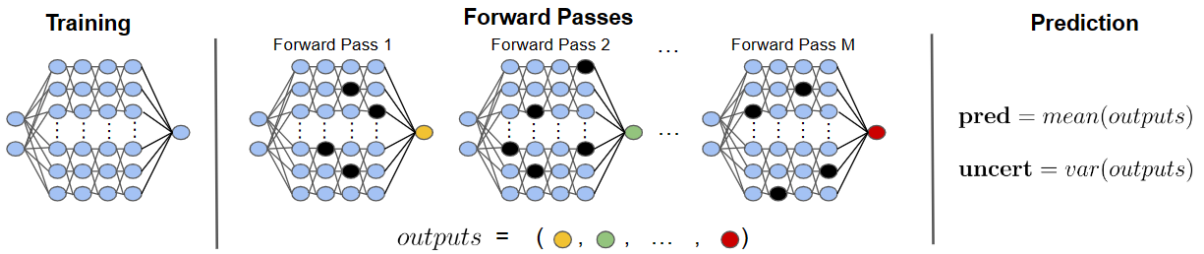


Figure 2.5: An scheme of MC dropout technique in a Multilayer Perceptron with M forward passes where black hidden units are those that have been dropped out at test time.

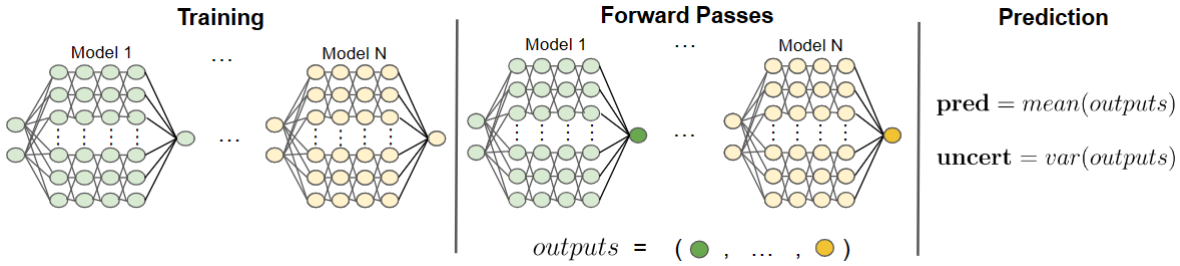


Figure 2.6: Illustration of a Ensemble of N Multilayer Perceptrons.

Deep Ensembles

The name of deep ensembles comes from the idea of training N models with different architectures, hyperparameters, or initial weights [38]. As result, you may combine their predictions to produce a more accurate final prediction -i.e average among model predictions-,

$$p(\mathbf{y}_*|\mathbf{x}_*, \mathcal{D}) = \frac{1}{N}\sum_{n=1}^{N} p(\mathbf{y}_*|\mathbf{x}_*, \boldsymbol{\theta}^n),$$

and uncertainty estimation -i.e variance among model predictions-, where $\boldsymbol{\theta}^n$ are the trained weights of model $1 \leq n \leq N$. This technique has been effectively employed across diverse topics [39, 30]. Comparing it with respect to MC dropout, it is observed that deep ensembles technique involves training and maintaining multiple models, which can be quite resource-intensive. An overview of this technique may be observed in Figure 2.6.

Overall, although both strategies tackle the uncertainty estimation problem, they have some computational time and load constraints. Furthermore, they use some heuristic techniques which, despite of being inspired by Bayesian methods, are not entirely Bayesian-based [40]. In contrast, the Laplace approximation provides a more direct connection to Bayesian principles. By approximating the posterior distribution of the model parameters, it offers a theoretically grounded approach to uncertainty estimation.

Chapter 3

Laplace Approximation

In this section, we show that Laplace approximation presents several advantages over other Bayesian inference techniques such as MC dropout or deep ensembles, particularly in terms of robust theoretical foundations and out-of-distribution generalization. Although Laplace approximation is a quite old technique (1992), it has gained renewed popularity in recent years in the context of Bayesian deep learning [17, 16]. Intense research efforts have been accomplished to overcome its limitations and explore new ways to apply the method in modern deep learning problems [15, 41, 14].

First, in section 3.1, we prove that Laplace approximation is derived from Bayesian principles. This offers a more reliable way to quantify uncertainty in deep learning models. The strong theoretical basis of the Laplace approximation ensures consistency and interpretability in various contexts. Second, in real-world applications, it is essential for a deep learning model to provide accurate and well-calibrated uncertainty estimates when faces with out-of-distribution data. Most techniques, including MC dropout and deep ensembles, tend to produce overconfident predictions for out-of-distribution samples. In contrast, the Laplace approximation is capable of reflecting the true uncertainty on unfamiliar data and reduces the risk of overconfident predictions [18, 42]. This will be shown in section 3.2 by way of some benchmark datasets.

3.1 Theoretical foundations

In this section, we present the theoretical basis of Laplace's method which was first proposed in this context by MacKay [**?**]. The idea behind Laplace's method is to approximate the posterior $p(\boldsymbol{\theta}|\mathcal{D})$ by a Gaussian $q(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Since the posterior only depends on $\boldsymbol{\theta}$, it may be expressed as,

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{\mathcal{L}(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{D})} = \frac{1}{Z}g(\boldsymbol{\theta}) \approx q(\boldsymbol{\theta}),$$

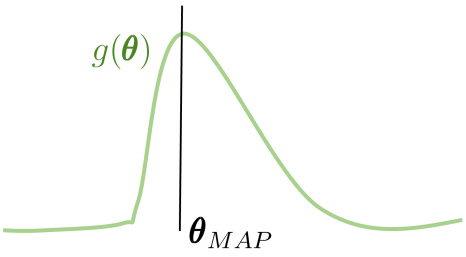
where $Z = p(\mathcal{D})$ is the constant that normalizes $g(\boldsymbol{\theta}) = \mathcal{L}(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})$, that is, $Z = \int g(\boldsymbol{\theta})d\boldsymbol{\theta}$. By definition of $\boldsymbol{\theta}_{MAP}$ (2.4), it is followed that $g(\boldsymbol{\theta})$ has a peak at $\boldsymbol{\theta}_{MAP}$ (Figure 3.1a).

In this way, it starts considering a truncated second-order Taylor expansion of $\log g(\boldsymbol{\theta})$ around its peak $\boldsymbol{\theta}_{MAP}$ (Figure 3.1b). Since the gradient is zero around its peak, we get

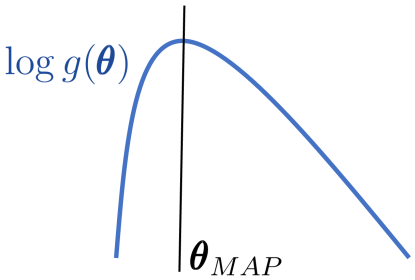
$$\log g(\boldsymbol{\theta}) \approx \log g(\boldsymbol{\theta}_{MAP}) - \frac{1}{2}\mathbf{H}(\boldsymbol{\theta} - \boldsymbol{\theta}_{MAP})^2, \tag{3.1}$$

where $\mathbf{H}$ denotes the hessian of $\log g(\boldsymbol{\theta})$ at $\boldsymbol{\theta}_{MAP}$ (Figure 3.1c),

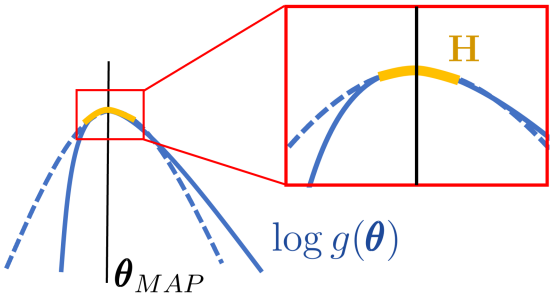
$$\mathbf{H} = -\nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}} \log g(\boldsymbol{\theta})\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_{MAP}}. \tag{3.2}$$



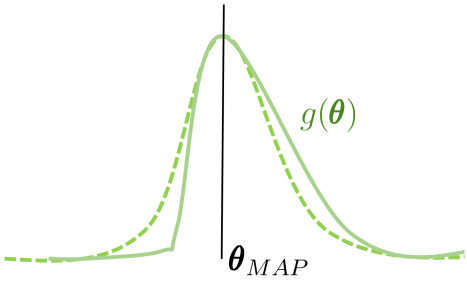
(a) $\boldsymbol{\theta}_{MAP}$ is the maximum of $g(\boldsymbol{\theta})$.



(b) Taking logarithm, we get $\log g(\boldsymbol{\theta})$.



(c) Approach $\log g(\boldsymbol{\theta})$ by Taylor.



(d) Taking exponential, we approach $g(\boldsymbol{\theta})$.

Figure 3.1: Scheme of Laplace's method in the scenario in which $\boldsymbol{\theta}$ is 1-dimensional. We represent with dashed lines the approximations done by the method of the continuous lines.

Taking the exponential from (3.1), we get (Figure 3.1d),

$$g(\boldsymbol{\theta}) \approx g(\boldsymbol{\theta}_{MAP}) \exp\left\{-\frac{\mathbf{H}}{2}(\boldsymbol{\theta}-\boldsymbol{\theta}_{MAP})^2\right\}.$$

This equation, in the absence of a pair of terms, resembles the pdf of a Gaussian. Consequently, we approximate Z by the normalizing constant of this Gaussian,

$$Z = \sqrt{\frac{(2\pi)^P}{|\mathbf{H}|}} g(\boldsymbol{\theta}_{MAP}).$$

As result, we get an analytical approximation of the posterior $p(\boldsymbol{\theta}|\mathcal{D})$,

$$q(\boldsymbol{\theta}) \sim \mathcal{N}(\boldsymbol{\theta}_{MAP}, \mathbf{H}^{-1}). \tag{3.3}$$

3.1.1 Hyperparameter tuning

In regression scenario (see Appendix (A.3)), the hessian would be equal to

$$\mathbf{H} = -\nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}} \left[\log \mathcal{L}(\mathcal{D}|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta})\right]\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_{MAP}} = \frac{1}{2\sigma^2}\nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}} \sum_{n=1}^{N} (\mathbf{y}_n - f(\mathbf{x}_n, \boldsymbol{\theta}))^2 \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_{MAP}} + \gamma^2 \mathbf{I}_P,$$

which means that both sigma noise σ and prior precision γ^2 play a significant role in the covariance of the posterior distribution (3.3). Therefore, it is typically beneficial to tune both of them. The most substantiated alternative is to maximize (gradient ascent) the marginal log-likelihood which was approximated by Mackay as [43],

$$\log p(\mathcal{D}) = \log \mathcal{L}(\mathcal{D} \mid \boldsymbol{\theta}_{MAP}) - \frac{1}{2}\log\left(\det\left(\frac{\mathbf{H}}{2\pi}\right)\right).$$

Meanwhile, in classification scenario (A.6), the procedure would be similar but, the unique hyperparameter to tune is the prior precision γ^2.

3.1.2 Approximations

Unfortunately, the main bottleneck of this technique comes from the hessian (3.2). The reason is twofold: first and foremost, in terms of computation since you have to compute the second derivatives of $\log g(\boldsymbol{\theta})$ with respect to all the parameters of your model. Second, in terms of storage, this matrix has as many rows/columns as parameters your model and, in the case of deep learning algorithms, it may scale to the order of millions. Depending on the nature of the problem, different solutions arise:

Hessian approximations

Intense research efforts have been accomplished to deal with the computation of the hessian. Let us detail some of them (Figure 3.2),

- The default choice is to approximate the hessian by the **generalized Gauss-Newton** (GGN) matrix [44],

$$\mathbf{H} \approx \sum_{n=1}^{N} J(\mathbf{x}_n) \left(\nabla^2_{ff} \log p(y_n \mid f) \big|_{f=f(\mathbf{x}_n, \boldsymbol{\theta}_{MAP})} \right) J(\mathbf{x}_n)^T,$$

  where $J(\mathbf{x}_n) = \nabla_{\boldsymbol{\theta}} f(\mathbf{x}_n, \boldsymbol{\theta}) \big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_{MAP}}$ is the NN's jacobian matrix. However, GGN matrix is still quadratically large and further approximations are required.

- As explained in [14], **Kronecker factored** Laplace yields in block-diagonal factorization, which defines the hessian $\mathbf{H}_l$ of each layer l as a Kronecker product of two smaller matrixes,

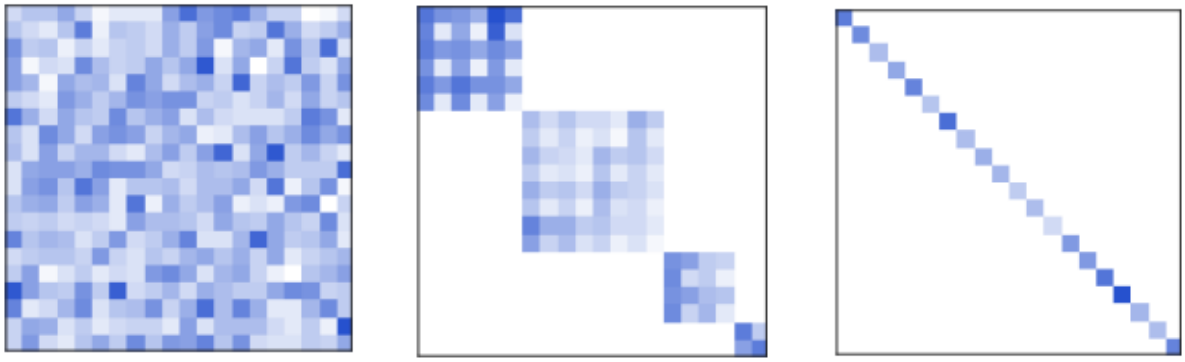
$$\mathbf{H}_l = \frac{\partial^2 E}{\partial \mathbf{W}_l \partial \mathbf{W}_l} = \mathcal{Q}_l \otimes \mathcal{R}_l,$$

  where $\mathcal{Q}_l$ and $\mathcal{R}_l$ are the covariance of the pertinent activation and the pre-activation hessian, respectively.

- However, previous approximations may not be enough for large NNs. In such cases, it is typically implemented the **diagonal factorization** which ignores off-diagonal elements.

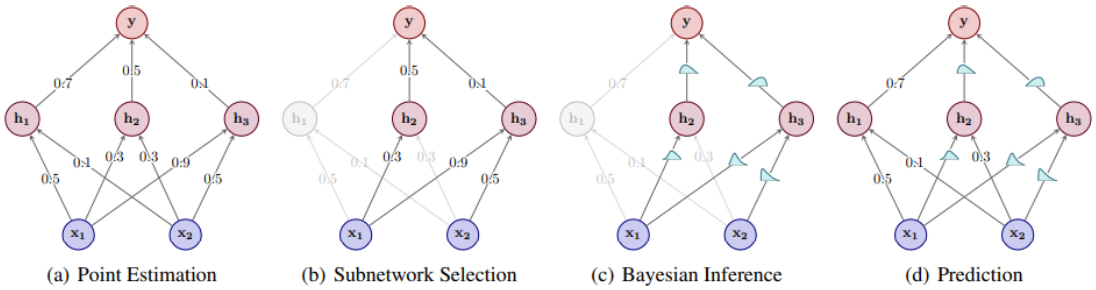
Subnetwork inference

As discussed in [41], a simple way to scale Laplace's method is to only consider probabilistically a subset of the parameters. In other words, apply Laplace's method to this subset, while the remaining parameters would take their MAP estimate (Figure 3.3). In practice, the most common choice is to just apply Laplace to the last layer of the NN, capturing all the model uncertainty in its parameters and avoiding taking further approximations in the hessian. Other common choice is to just consider as Bayesian weights the k weights with greatest absolute values. This idea relies on the fact that the nearer the weights are from 0, the less significant they are in the predictions.



(a) GGN Matrix. (b) Kronecker Factorization. (c) Diagonal approximation.

Figure 3.2: Hessian approximations to address memory and computational cost challenges.



(a) Point Estimation (b) Subnetwork Selection (c) Bayesian Inference (d) Prediction

Figure 3.3: Schematic illustration of the approach proposed by [41]. (a) Train a neural network to obtain a point estimate of the weights. (b) Choice a small subset of the weights. (c) Estimate a posterior distribution over the selected subnetwork via Bayesian inference techniques. (d) Make predictions using the full network with a mix of Bayesian and deterministic weights.

3.1.3 Predictive distribution

Assuming the posterior approximation obtained from Laplace's method (3.3), the uncertainty captured in the model parameters may be translated into our predictions,

$$\mathbf{y}_* = f(\mathbf{x}_*;\boldsymbol{\theta}) + \varepsilon, \;\; \boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{\theta}_{MAP}, \mathbf{H}^{-1}). \tag{3.4}$$

Theoretically, we could apply (2.3) to get an analytical distribution but, in general, it is intractable. Alternatively, there exists the following approximations.

Monte Carlo

The simplest but the unique approximation which is valid for both regression and classification problems is Monte Carlo integration. Monte Carlo takes K samples from (3.3) so as to average the outputs,

$$\mathbf{y}_* \approx \frac{1}{K}\sum_{k=1}^{K} f(\mathbf{x}_*;\boldsymbol{\theta}_k); \quad \boldsymbol{\theta}_k \sim p(\boldsymbol{\theta}|\mathcal{D}).$$

The model uncertainty could be measured as the variance among the K samples.

Linearization

For Gaussian likelihoods, e.g regression problems, if we want to capture all the model uncertainty in a predictive distribution, f must be linear with respect to $\boldsymbol{\theta}$. We may approximate the network function by its truncated first-order Taylor expansion centered at $\boldsymbol{\theta}_{MAP}$,

$$f(\mathbf{x}_*;\boldsymbol{\theta}) \approx f(\mathbf{x}_*;\boldsymbol{\theta}_{MAP}) + J(\mathbf{x}_*)(\boldsymbol{\theta} - \boldsymbol{\theta}_{MAP}). \tag{3.5}$$

Consequently, it is followed from (3.5) that,

$$\mathbb{E}[f(\mathbf{x}_*;\boldsymbol{\theta})] = f(\mathbf{x}_*;\boldsymbol{\theta}_{MAP}), \quad \mathbb{V}ar[f(\mathbf{x}_*;\boldsymbol{\theta})] = J(\mathbf{x}_*)^T\mathbf{H}^{-1}J(\mathbf{x}_*),$$

which, under the Gaussian assumption, allow us to get an analytical predictive distribution,

$$p(f(\mathbf{x}_*;\boldsymbol{\theta})|\mathbf{x}_*,\mathcal{D}) \approx \mathcal{N}\left(f(\mathbf{x}_*;\boldsymbol{\theta}_{MAP}), J(\mathbf{x}_*)^T\mathbf{H}^{-1}J(\mathbf{x}_*)\right). \tag{3.6}$$

Linearization is the most mathematically rigorous technique for translating model uncertainty, estimated using the Laplace approximation, into predictive distribution. To sum up, let us review its main steps,

1. Train a neural network to obtain a point estimate of the weights $\boldsymbol{\theta}_{MAP}$.
2. Estimate the hessian $\mathbf{H}$ of the weights leveraging the explained approximations (section 3.1.2).
3. In inference, for each new input $\mathbf{x}_*$, compute the jacobians $J(\mathbf{x}_*)$ by way of backpropagation. In fact, if model's output dimension is n, the matrix covariance of (3.6) should be $n \times n$ and, then, we will have to perform a backpropagation parting from each output dimension.
4. Finally, equation (3.6) is applied to obtain the predictive distribution.

One of the main challenges of using linearization in the Laplace approximation is step 3, since it may significantly slow down the inference, making it impractical for real-time applications. As a result, researchers have been working to develop more efficient methods to overcome this limitation [15, 16].

Probit/Bridge

In classification problems, there exists two main alternatives to Monte Carlo. First, Mackay [**?**] described the probit approximation. This technique approaches the logistic function with the probit function (the inverse of the standard normal cumulative distribution function) enabling us to compute (2.3) analytically. More recently, it appeared the bridge approximation which estimates (2.3) via a Dirichlet distribution [45].

3.2 Benchmark Datasets

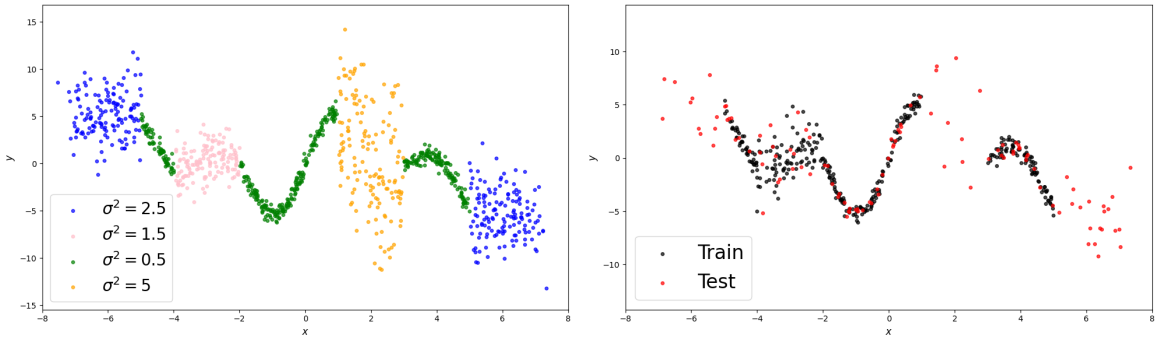
In this section, we will compare the performance of the Laplace approximation method with three other commonly used techniques - MC dropout, deep ensembles, and default MAP estimation - in two regression datasets. Our objective is to demonstrate the strengths of Laplace's method in out-of-distribution predictions in terms of accuracy, likelihood and calibration. Complementary, we have developed a Python notebook that enables users to explore and experiment with all these configurations and intuitively visualize the main differences among them `https://github.com/cplou99/BayesianDL`.

Laplace Library-. To implement the Laplace method, we will use Laplace-Redux library [17]. This Python library was recently developed by Alex Immer in 2021 and can be easily used with popular deep learning models integrated with PyTorch. It allows the use of all the hessian approximations and prediction techniques explained in sections 3.1.2 and 3.1.3, as well as the hyperparameter estimation by way of the marginal likelihood (section 3.1.1).
Architecture details-. Due to the simplicity of these datasets, we will employ a multilayer perceptron (MLP) of 3 layers and 25 hidden size units. It is followed that our neural network has 726 weights. To estimate $\boldsymbol{\theta}_{MAP}$, we consider as loss (2.4) which will be minimized by `Adam` optimizer with a learning rate of 0.01. Since the MLP is homoscedastic by default, we can add another head that predicts the variance (section 2.2.2) in order to deal with heteroscedasticity and thus, estimate aleatoric uncertainty in an input-dependent way. As result, the model outputs $(\hat{y}, \hat{\sigma}_a^2)$ for each input. Refering to this model as `base model`, we implement the next Bayesian techniques in order to estimate the epistemic uncertainty $\hat{\sigma}_e^2$:

- **Deep ensembles**: we build the deep ensembles from 10 `base models` with random weights initialization. After training them independently, we estimate the final predictions, the aleatoric uncertainty and the epistemic uncertainty as $Avg(\hat{y})$, $Avg(\hat{\sigma}_a^2)$ and $\mathbb{V}ar(\hat{y})$ among model's predictions, respectively.

- **MC dropout**: we add a dropout layer in the one-to-last layer with $p = 0.25$ and perform 10 forward passes during inference. We estimate the final predictions, the aleatoric uncertainty and the epistemic uncertainty as $Avg(\hat{y})$, $Avg(\hat{\sigma}_a^2)$ and $\mathbb{V}ar(\hat{y})$ among forward passes, respectively.

- **Laplace**: we consider different strategies to estimate both weights and predictive distributions. In this way, we will test next approaches: first, we consider as Bayesian weights either the full network or just the last layer or just the 200 parameters with highest absolute values. The posterior distribution of the selected weights will be estimated by GGN approximation. Afterwards, to estimate the predictive distribution we will use the linearization technique since the neural network size is limited. The hyperparameters, including the sigma noise σ^2 and prior precision γ^2, will be estimated using the marginal likelihood.



(a) Heteroscedastic Dataset. (b) Training and Test sets.

Figure 3.4: Two images of Simulated Regression Dataset in which we show its heterothedasticity and the out-of-distribution points of the test set.

Regression Metrics-. We use the negative log likelihood (NLL) metric to measure the probability that the actual values (y) follow the predicted distributions $\mathcal{N}(\hat{y}, \hat{\sigma}_a^2 + \hat{\sigma}_e^2)$. Specifically, we report the average of the NLL along the test set with lower values indicating better performance. Besides, to measure accuracy, we use the root mean squared error (RMSE) since it is quite sensitive to observations that are further from the mean. We leverage the area under the sparsification error (AUSE) curve to measure the calibration of the uncertainty estimation [46]. AUSE measures the difference between the predictive uncertainty and an oracle based on true prediction error (RMSE). Together, these metrics provide a comprehensive evaluation of the performance of both the regression model and the uncertainty estimation techniques.

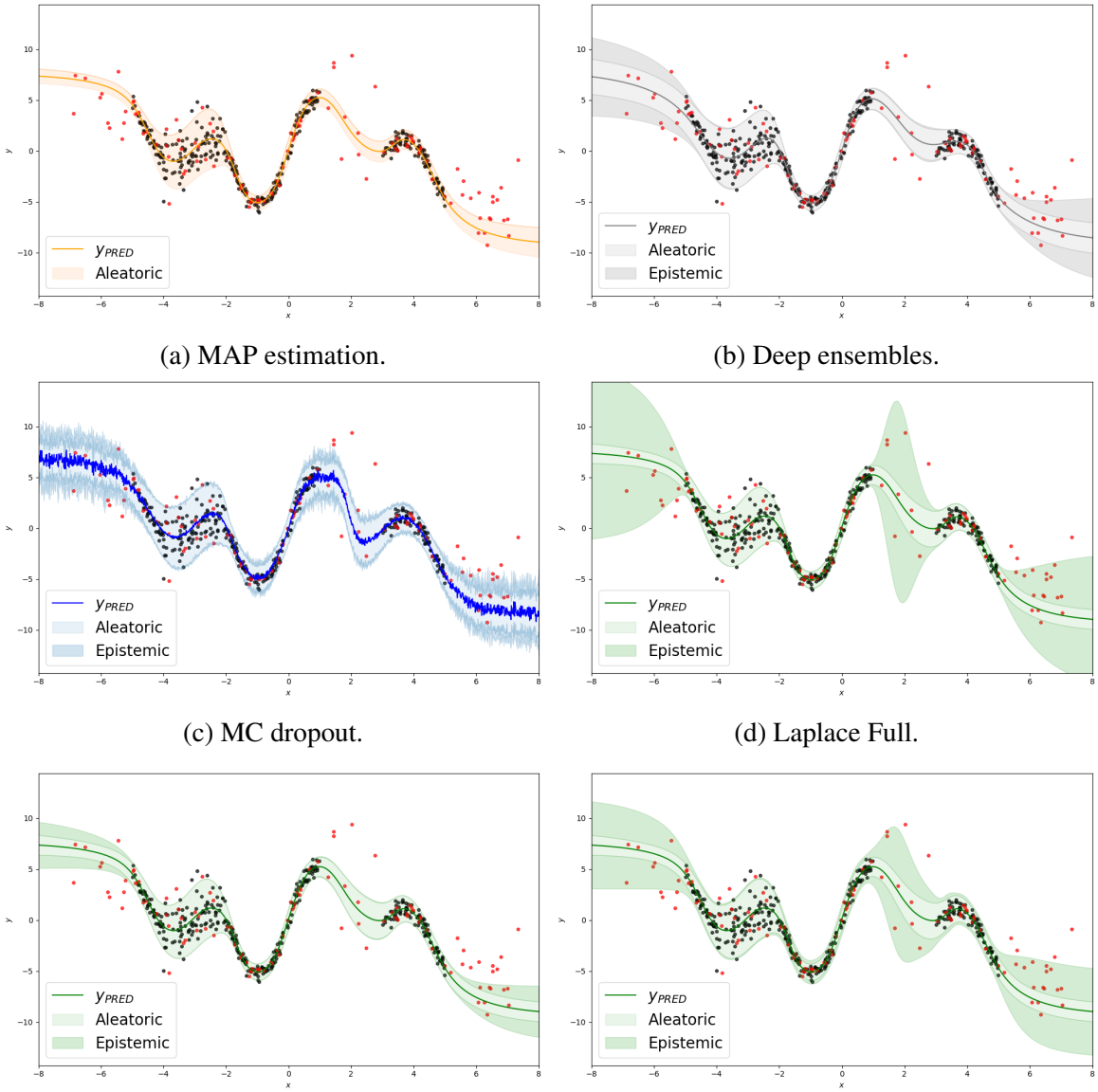
3.2.1 Simulated Dataset

Dataset-. This is a one-dimensional dataset designed to describe the meaning of both aleatoric and epistemic uncertainty. It was constructed by applying a specific function $f(x)$ to 1000 points within the interval $x \in [-7, 7]$, and adding certain noise $\varepsilon \sim \mathcal{N}(0, \sigma^2)$ to each point, resulting $y = f(x) + \varepsilon$. The variance of the noise term σ^2 varies based on the input value (heteroscedastic dataset). The resulting dataset together with the chosen σ^2 values may be visualized in Figure 3.4a.
Training/Test split-. We split the dataset into training and test set, comprising 85% and 15% of the data, respectively. Following the approach of Foong and Li [18], we define certain regions where training points will not be present. Specifically, these regions are $[-7, -5]$, $[1, 3]$, and $[5, 7]$. Any points within these intervals are removed from the training set. This approach ensured that the test set contained data both inside and outside of the aforementioned regions, representing both out-of-distribution and in-distribution data, respectively (Figure 3.4b).
Results-. We evaluate 6 different alternatives: (1) `base model` MAP estimation as baseline since it does not estimate epistemic uncertainty, (2) deep ensembles, (3) MC dropout and Laplace considering as Bayesian weights either the (4) full network (Laplace Full) or (5) the last layer (Laplace LL) or (6) the Subnetwork (Laplace Subnet) with the 200 parameters with highest absolute values. We may observe in Figure 3.4 the predictions and uncertainty estimations along the x axis for each method and in Table 3.1 their belonging metric values in the test set. If we focus on the predictions and take MAP predictions as reference, we realise that whereas Laplace's are identical to MAP's just as (3.5) establishes, deep ensembles and MC dropout predictions vary slightly since they are the average of 10 forward passes. This fact leads to a slight improvement in accuracy terms (RMSE).

It is highlighted the stochasticity of the MC dropout predictions due to Dropout layer. Regarding aleatoric uncertainty, it is noticed that it fits accurately to the noise of each train set region for all methods. Main differences come from the epistemic uncertainty. All methods detect the outside regions as the most uncertain ones, however both deep ensembles, MC dropout and Laplace LL seem to be still overconfident. This fact is even more noticeable in the inside region that does not contain training points since there, only Laplace Full and, to a lesser extent, Laplace Subnet are able to predict high uncertainty values for these test points. These observations are translated into analytic values by NLL metric. The performance of Laplace Last Layer is clearly worse than the other two Laplace's methods. This may be due to the fact that with this method only 26 weights are considered as Bayesian and it does not seems to be enough to capture the uncertainty of the model.
At AUSE values sight (their belonging plots may be observed in Appendix B.1), Laplace Full and Subnet are the ones that show higher calibration, that is, higher correlation between uncertainty and error.
Consequently, Laplace Full clearly outperforms the other techniques and Laplace Subnet arises as the main alternative if we work with models or problems that require higher memory resources.

3.2.2 Boston Dataset

Dataset-. The Boston dataset is a well-known benchmark dataset used in machine learning for regression tasks. It is available in the scikit-learn library as part of its datasets module. The dataset contains information about housing values in Boston and related factors, such as crime rate, average number of rooms per dwelling, and distance to employment centers. It consists of 506 samples, each with 13 features, making it a medium-sized dataset.
Training/Test split-. We randomly split the data into training and test sets using a $75\% - 25\%$ ratio. Additionally, we normalized the data to ensure that all features had the same scale. Specifically, we used scikit-learn's `StandardScaler` to transform the data such that each feature had a mean of 0 and standard deviation of 1. This preprocessing step is important, as it can improve the convergence and performance of machine learning algorithms.
Results-. We evaluate the same 6 models as in the previous benchmark. We may observe in Figure 3.4 the predictions, uncertainty estimations and ground truth of the test set. Besides, just as before, we find in Table 3.1 their belonging metric values.
In this scenario, the best performance in terms of accuracy is achieved by MC dropout followed by Laplace methods and, lastly, deep ensembles which means that the training of the model benefits from certain randomness. However, regarding NLL values, we realise that the epistemic uncertainty estimated by MC dropout is significantly more erroneous than deep ensembles or Laplace Full/Subnet. In fact, MC dropout seems to be high overconfident since there are many wrong predictions with low estimated uncertainty. This fact is noticed in AUSE metric (may be visualized in AUSE plots in Appendix B.2). Other techniques such as deep ensembles or Laplace Subnet are slightly better calibrated (less AUSE) and are not as overconfident as MC dropout. Just as before, the model that achieves the best performance in both NLL and AUSE metrics is Laplace Full. In fact, in some cases, it seems to be too underconfident, that is, it estimates quite high uncertainties when predictions are accurate.



(a) MAP estimation.

(b) Deep ensembles.

(c) MC dropout.

(d) Laplace Full.

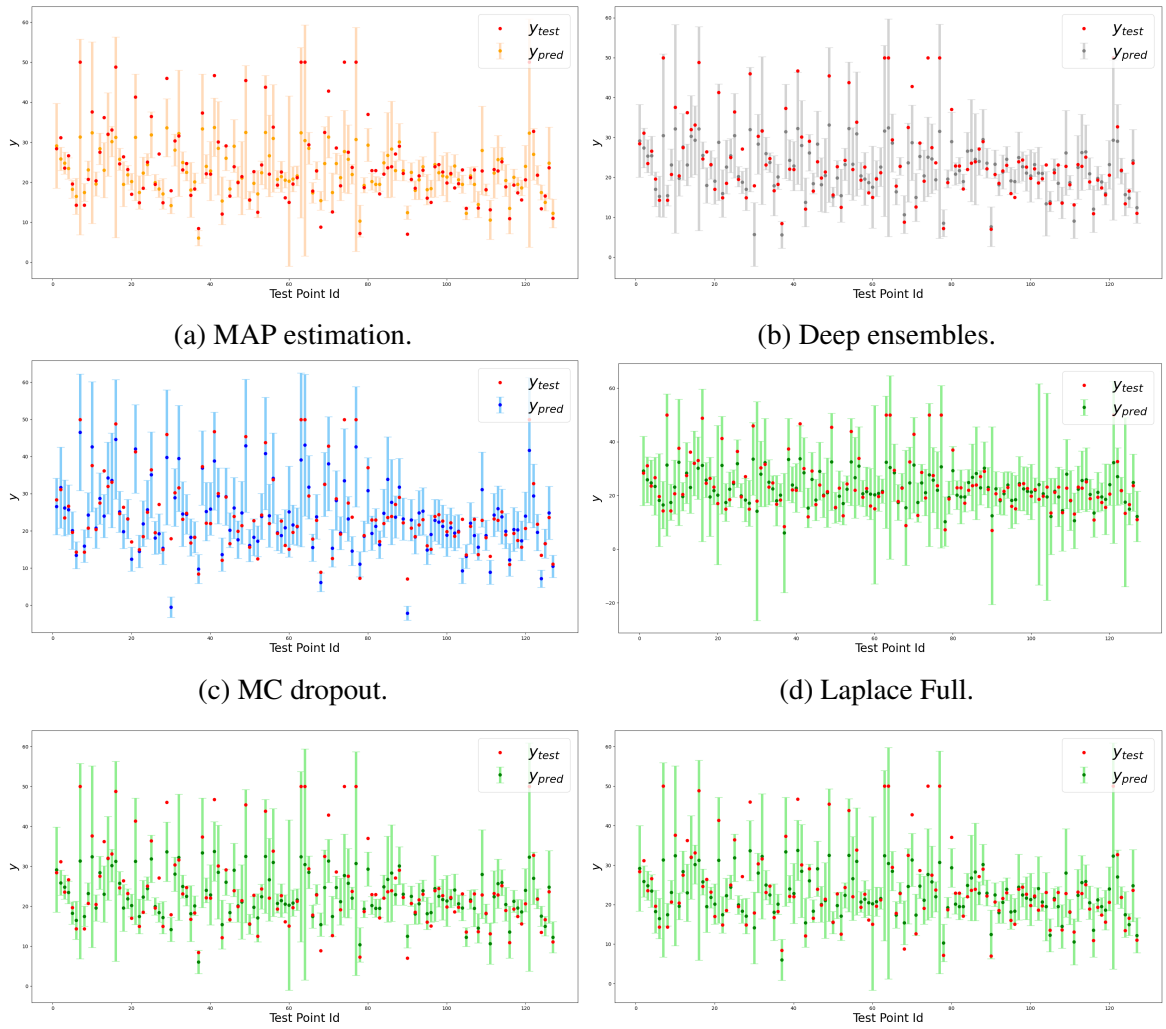
(e) Laplace Last Layer.

(f) Laplace Subnet.

Figure 3.5: Prediction and uncertainty estimation along the x axis of the different models in simulated Dataset. Both uncertainties are plotted as areas of different opacity centered on predictions (line with the same color).

Table 3.1: Metrics values for all the models in both benchmark datasets. Arrows indicate the direction of the desired relationship between each metric and performance.

	Simulated Dataset			Boston Dataset		
	RMSE ↓	NLL ↓	AUSE ↓	RMSE ↓	NLL ↓	AUSE ↓
MAP	2.15	8.54	97.61	5.76	4.85	178.32
Deep ensembles	**1.99**	2.86	39.45	6.16	2.95	107.73
MC dropout	2.14	2.45	38.02	**5.61**	3.85	241.25
Laplace Full	2.15	**1.81**	26.17	5.76	**2.78**	**92.29**
Laplace LL	2.15	3.77	38.23	5.76	3.88	141.59
Laplace Subnet	2.15	1.85	**24.42**	5.76	2.95	95.08



(a) MAP estimation.

(b) Deep ensembles.

(c) MC dropout.

(d) Laplace Full.

(e) Laplace Last Layer.

(f) Laplace Subnet.

Figure 3.6: Prediction and uncertainty estimation of the different models in Boston dataset. Both aleatoric and epistemic uncertainty are summed and plotted with lines of the same color as predictions (points in the middle of the lines). Ground truth correspond to red points.

Chapter 4

RL: Active Exploration

4.1 Reinforcement Learning

Reinforcement learning (RL) is a branch of machine learning where an **agent** learns a **policy** from trial-and-error interactions with an **environment**. The policy is in charge of making decisions, that is, it returns the **action** that it should take in each **state**. The agent's goal is to maximize a cumulative **reward** signal, which is provided by the environment. All these concepts are merged at **Markov decision process (MDP)**, a formal framework that captures the interaction between an agent and its environment. An MDP consists of a set of states, a set of actions, a set of probabilities that describe the transitions between states when an action is taken and a set of rewards associated to transitions.

Reinforcement learning can be divided into two categories: model-based and model-free.

- In a model-based RL approach, a model is trained to describe the dynamics of the system, and uses this model to plan its actions. This model takes as input a pair state-action (s, a) and outputs the resulting state s' reached by the agent. This approach is useful when the environment's dynamics are known or can be learned accurately. A key advantage of the model-based approach is that it allows the agent to plan ahead and consider long-term consequences of its actions [47, 48].

- In a model-free RL approach, the agent does not try to explicitly model the environment's dynamics. Instead, the agent directly learns a policy that maps states to actions. This approach is useful when the environment's dynamics are complex or unknown [49, 50].

Usually, in most of RL problems, the agent's policy should balance **exploration** and **exploitation**. On the one hand, it needs to explore the environment to discover new, potentially more rewarding actions and states. On the other hand, it needs to exploit the current knowledge to maximize the reward in the short term.

Efficient exploration in high-dimensional environments is an unsolved problem in reinforcement learning. High-dimensional environments are characterized by a large number of possible states and actions, making it challenging for the agent to explore all relevant areas of the state space. Current exploration methods are grouped in two branches:

- **Reactive exploration** methods rely on chance discoveries to drive exploration [23, 24]. The agent explores the environment in a more random manner, hoping to accidentally find new and interesting areas. Once the agent has found a novel area, it will typically receive a bonus or an intrinsic motivation reward to encourage further exploration in that area.

However, this approach can be inefficient because the agent has to unlearn the bonus once the novelty finishes. This can lead to over-commitment, where the agent spends too much time exploring an area that is no longer novel, instead of exploring other areas that might be more promising.

- In **Active exploration** methods, the agent seeks out new and interesting areas based on its internal estimate of what actions might lead to interesting transitions. These methods allow the agent to explore the environment in a more efficient way, reducing the likelihood of over-commitment and enabling the agent to discover new areas of the environment more quickly. That is the reason why active exploration methods can be more effective than reactive methods in high-dimensional environments.

In **pure exploration** RL problems, the agent's goal is to learn as much as possible about the environment without maximizing the external reward signal. In these problems, exploration itself is the objective, and the agent uses a novelty measure as internal reward, where the agent is incentivized to explore areas of the environment that it has not yet visited. However, developing an effective novelty measure can be challenging, particularly in high-dimensional environments. Existing formulations of measuring novelty include visitation count [25, 26], prediction error [27, 28] and diversity in the visited states [51, 52].

4.2 Model-based Active Exploration

Model-based reinforcement learning involves learning a model of the environment, including its transition dynamics or reward function, to inform the agent's decision-making process. Consequently, deep learning has been widely adopted in model-based reinforcement learning due to its ability to learn complex representations of the state space and dynamics of the environment. Besides, deep learning models can also generalize well to unseen situations, enabling the agent to make informed decisions in novel situations.

Moreover, in pure exploration problems, it is required a method to predict the consequences of actions and to measure their degree of novelty. One effective way to do this is by using Bayesian deep learning, which provides an optimal framework to make predictions and estimate the uncertainty associated to its predictions. In this setting, the novelty of a given state transition can be measured by the degree of uncertainty associated with it [29, 30]. Thus, we leverage the different Bayesian methods analyzed in previous sections as frameworks to address Model-based Active Exploration problem.

4.2.1 Problem Formulation

The key idea behind the formulation of active exploration problem is to distinguish between the two MDPs that we are considering. First, the internal MDP used to exploration where novelty is defined as reward. Second, the external MDP that corresponds to the environment whose aim is to perform a specific task and thus, employs an external reward that describes the desired task. Consequently, the novelty of transitions are estimated before they are encountered by the agent in the environment.

Let us define the external MDP as $(\mathcal{S}, \mathcal{A}, t^*, r)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space and t^* is the unknown transition function that determines the probability $p(s'|s,a,t^*)$ of reaching state s' after taking action a from state s. Lastly, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ refers to the external reward function.

Let T be the space of all transition functions and $P(T)$ the probability distribution over transition functions that captures the current belief of how the environment works. In this way, the internal MDP may be defined as $(\mathcal{S}, \mathcal{A}, \bar{t}, u)$, where the sets $\mathcal{S}, \mathcal{A}$ are those of the external MDP, the transition function $\bar{t}$ is defined as,

$$p(s'|s,a,\bar{t}) = \mathbb{E}_{t\sim P(T)} p(s'|s,a,t),$$

and u refers to the utility $u(s,a)$ of the pair state-action (s,a). Notice that the utility in the internal MDP plays the same role as the reward of the task in the external MDP. Then, pure exploration can be defined as an iterative process, where a policy $\pi^{exp} : \mathcal{S} \times \mathcal{A} \to [0,1)$ is used in each iteration to gather information about unexplored areas of the environment.

4.2.2 Pipeline

The pipeline of the problem may be divided in three main folds: exploration, evaluation and policy creation. In this section, we carefully describe the pipeline belonging to each of these folds. Complementary, we provide their corresponding pseudocodes in Appendix A.2. In this way, we encourage to follow simultaneously the explanation with the pseudocode to achieve a better understanding.

Exploration-. First of all, the backbone of the problem is the exploration algorithm (Algorithm 1 in Appendix A.2) which is represented in Figure 4.1 and may be summarized as follows:

1. The agent starts acting randomly along n^{ex}_{warm} steps. As our agent interacts with the environment, it collects trajectories of the form $\{s_i, a_i, s'_i\}$ in the buffer Φ, where i denotes the number of exploration steps performed.

2. A model f^{ex} that simulates the dynamics and a policy π^{ex} are built (Policy pipeline) from the collected trajectories Φ. To achieve this, it uses the utility function that measures novelty as reward.

3. The agent acts along n_{pol} steps following policy π^{ex}. As our agent interacts with the environment, it continues collecting trajectories $\{s_i, a_i, s'_i\}$ in Φ. Afterwards, the transition model and, what is more, the policy π^{ex} are discarded.

4. We repeat stages 2-3 until the agent reaches n^{ex}_{steps} exploration steps, that is, until $i = n^{ex}_{steps}$. Meanwhile, each n_{eval} exploration steps, we evaluate the exploration accomplished up to that moment, entering in the evaluation pipeline with all the collected trajectories Φ.

Evaluation-. Similarly, the evaluation pipeline (Algorithm 2 in Appendix A.2) which is represented in Figure 4.2, may be explained in the next steps:

1. For each task of the environment, we repeat n_k times the following stages in order to get a more accurate estimate of the agent's performance.

2. A transition model f^{ev} that simulates the dynamics of the environment and a policy π^{ev} are built (Policy pipeline) from the collected trajectories Φ. To achieve this, it leverages the reward of the desired task.

3. The agent acts along n^{ev}_{steps} steps following policy π^{ev}. The final reward is the sum of the task reward along these steps.

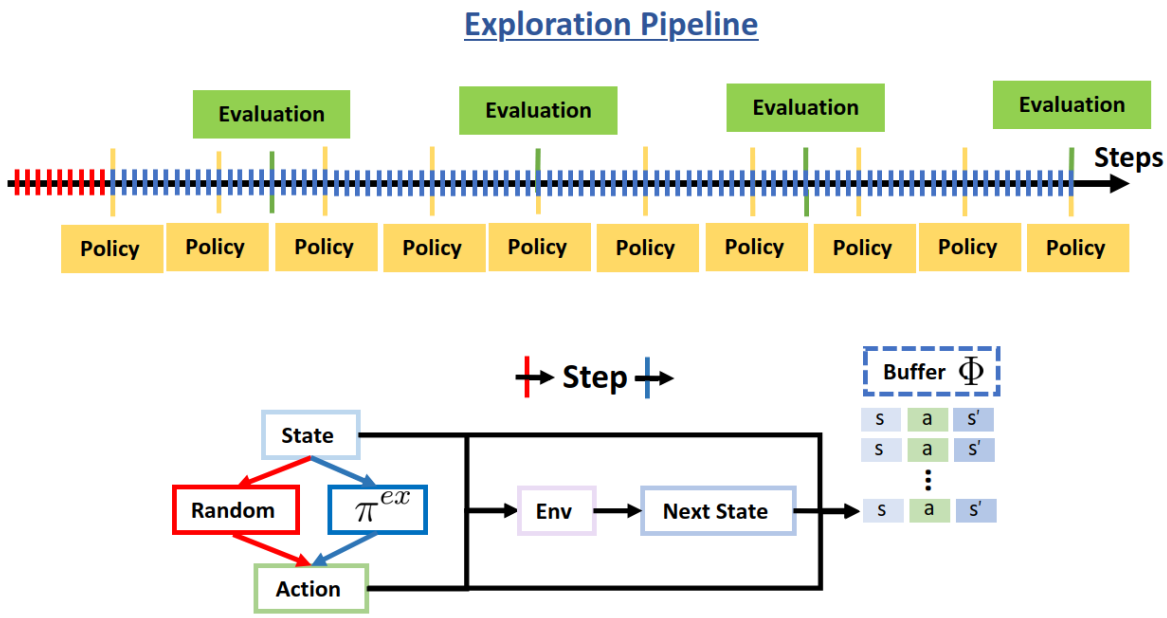


Figure 4.1: Scheme of the Exploration Pipeline (Algorithm 1) detailing its steps. Specifically, this example shows $n^{ex}_{steps} = 100$ exploration steps where the first $n^{ex}_{warm} = 10$ are warm-up (steps in red). Each step is stored in buffer Φ which is the output. Policy π^{ex} (Figure 4.3) is recomputed each $n_{pol} = 10$ steps and Evaluation (Figure 4.2) is accomplished each $n_{eval} = 25$ steps.

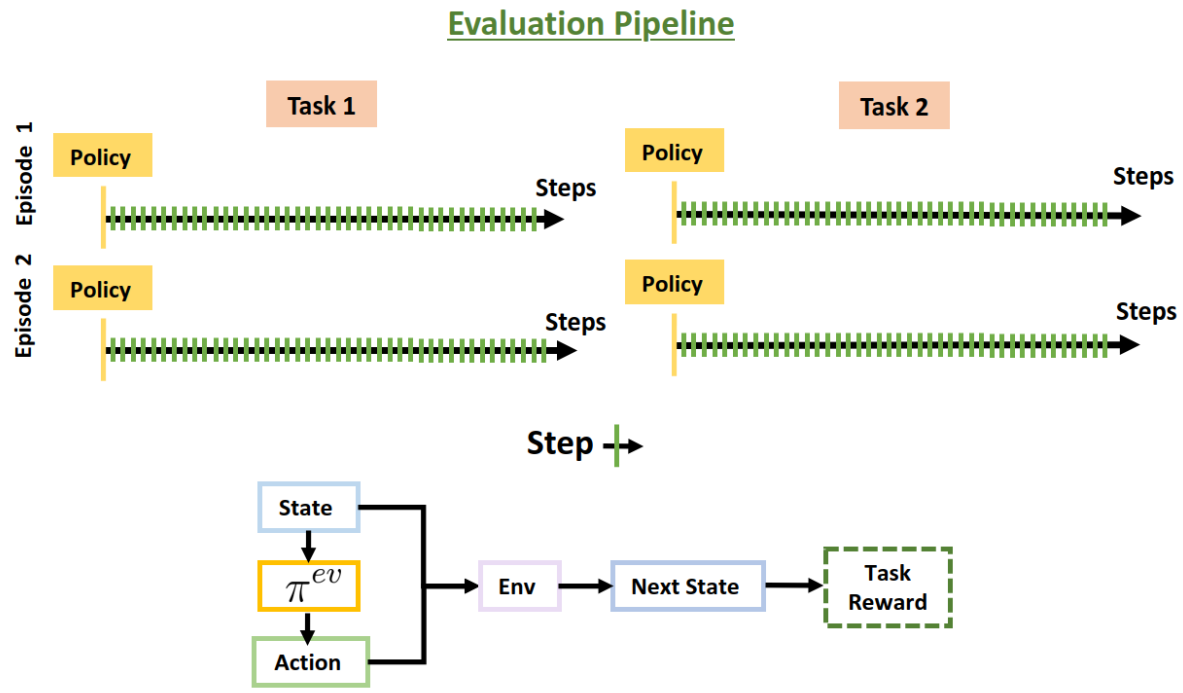


Figure 4.2: Scheme of the Evaluation Pipeline (Algorithm 2) detailing its steps. Specifically, in this example there are $n_k = 2$ episodes per task. Each episode starts computing the policy π^{ev} (Figure 4.3) and, subsequently, runs $n^{ev}_{steps} = 50$ steps. It stores the task reward achieved in each step and then, the output is its sum.

Policy-. As we have observed, the agent requires a policy to act in both exploration and evaluation pipelines. These policies are created following the same steps, but varying the model f and the reward function R. These steps (Algorithm 3 in Appendix A.2) are represented in Figure 4.3 and may be summarized as follows,

1. The collected trajectories Φ are used to train the neural network f. Specifically, the model is trained to map each pair state-action (s_i, a_i) to the resulting state s'_i.

2. Once the model is trained we can build the policy π running n^{pol}_{eps} episodes. Each episode starts from an initial state s_{init} and carries n^{pol}_{steps} steps. Besides, in each step we simulate the consequence of n_{act} different actions with the trained model f. These actions are sampled randomly in the first n^{pol}_{warm} episodes and, afterwards, are given by the policy. After each step, the policy is updated with the n_{act} tuples (s, a, s', r), where the reward r has been previously estimated according to the desired reward function R.

For learning both pure exploration and task-specific policies, we employed Soft-Actor Critic (SAC) which is the state-of-the-art for efficient reinforcement learning [53]. SAC is a deep reinforcement learning algorithm that optimizes a stochastic policy in an off-policy way -i.e, the updated policy is different from the behavior policy-. Specifically, the policy is trained to maximize a trade-off between expected return and entropy (randomness).

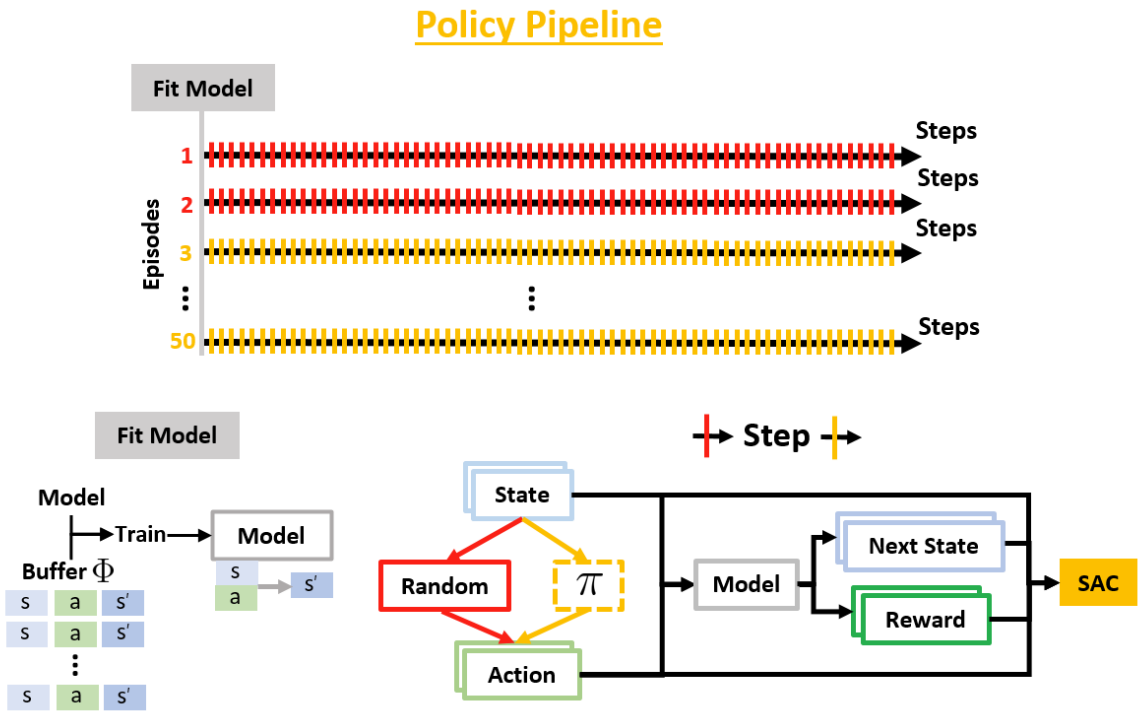


Figure 4.3: Scheme of the Policy Pipeline (Algorithm 3) together with an overview of its steps and model's training. In this scheme, after training the model, $n^{pol}_{eps} = 50$ episodes are performed (the first $n^{pol}_{warm} = 2$ episodes are warm-up, shown in red). Each episode consists of $n^{pol}_{steps} = 75$ steps. After each step, policy π (output) is updated by way of SAC.

4.3 Approaches

In this section, we detail the different approaches employed as transition models to make predictions and estimate their uncertainty (section 4.3.1) as well as the utility measures configured as rewards in the exploration pipeline (section 4.3.2).

4.3.1 Transition models

The role of the models is twofold: they are in charge of making predictions by simulating the dynamics of the environment and also, they estimate the uncertainty of their predictions in the exploration pipeline. In this way, all the models share a `base_model` as backbone. This `base_model` f is a two "head" Bayesian MLP (section 2.2.2) that receives as input a pair state-action (s,a) and outputs both the next state estimation $f_{\hat{s}'}(s,a;\boldsymbol{\theta})$ and its variance $f_{\hat{\sigma}^2_{s'}}(s,a;\boldsymbol{\theta})$ (aleatoric uncertainty). In this way, the model is trained to find the MAP $\boldsymbol{\theta}_{MAP}$ (section 2.2.2), where the likelihood is given by

$$N_{\boldsymbol{\theta}} \sim \mathcal{N}\left(f_{\hat{s}'}(s,a;\boldsymbol{\theta}), f_{\hat{\sigma}^2_{s'}}(s,a;\boldsymbol{\theta})\right), \tag{4.1}$$

and the prior distribution of the `base_model` weights is $\mathcal{N}(0,1/\gamma^2)$, where γ^2 is their prior precision. From this baseline and with the aim of capturing the epistemic uncertainty, we consider the following Bayesian approaches as transition models.

Deep ensembles-. We stack n_{ens} `base_models` and initialize their weights randomly (section 2.2.2). Hence, the input of each of these n_{ens} `base_models` is the same and their outputs should vary showing the model uncertainty. Although the training and inference of these n_{ens} `base_models` can be easily parallelized, it could be a bottleneck in terms of memory resources.

MC dropout-. We include Dropout in a layer of the `base_model` and perform n_{fp} forward passes to capture the model uncertainty among the n_{fp} predictions (section 2.2.2). This approach allows us to store and train only a model but, it still requires many forward passes at inference.

Laplace-. Trying to quantify uncertainty in a more reliable way, we may approximate first the posterior distribution of the `base_model` parameters through Laplace's approximation and, subsequently, translate this uncertainty into the predicted variable (section 3.1). Among all different alternatives to approximate the posterior (section 3.1.2) and the predictive distribution (section 3.1.3), we have chosen the next ones:

- LaplaceLLMC: we apply Laplace's method to the last layer of the `base_model`, capturing all the model uncertainty in its parameters and avoiding taking further approximations in the Hessian. To translate this uncertainty into the predicted variable, we apply Monte Carlo. Specifically, we take n_{MC} samples of the Last Layer weights from Laplace's distribution and perform a forward pass per sample. Since the remaining parameters would take their MAP estimate, it is not necessary to perform the forward pass over the full network n_{MC} times. We could make the forward pass up to the one to last layer once, and then for each sample make the last layer forward. Furthermore, the reason why we did not apply the linearization technique in this scenario is that if you backpropagate only a linear layer you get the same result (input neuron) independently the ouput neuron you part from and, consequently regarding equation (3.5), the covariance is a diagonal matrix with all its entries equal.

- LaplaceSubnetMC: we consider as Bayesian weights the n_{sub} weights with greatest absolute values, estimating their posterior distribution with Laplace's method.

This approach allows us to consider the weights that are theoretically more relevant in the `base_model` predictions. Nevertheless, if we use Monte Carlo as before, we must perform a forward pass along the full network per sample (similarly to MC dropout).

- LaplaceSubnetLinearized: we apply linearization technique instead of Monte Carlo to estimate the predictive distribution from the posterior distribution of the Subnetwork. This approach returns an analytical and well-calibrated predictive distribution, but it may significantly slow down the pipeline since it must compute the Jacobians -i.e perform $dim(\mathcal{S})$ backward propagations, one from each output dimension of $f_{s'}(s,a;\boldsymbol{\theta})$- per each input (s,a). This is the reason why we only tested this approach in the simplest (smaller state spaces) environments.

4.3.2 Utilities

In RL, the utility function is usually used to assign a value to each state and action based on how much reward the agent can expect to receive from taking that action in that state until termination. In our scenario, we employ utility as reward function of the exploration MDP and it assigns to each state-action pair (s,a), the expected novelty of future steps. In this way, we will use the next metrics to measure novelty.

Epistemic Uncertainty-. Since a model f^{ex} is used in the exploration MDP to simulate the dynamics of the environment, we could think about novelty as model's ignorance. In our Bayesian framework, model's ignorance may be measured as model/epistemic uncertainty. Particularly, it may be deduced from either the predictive samples or predictive distributions returned by the models of the previous section. If the model returns a set of predictive samples $\left\{f^{ex}_{s'}(s,a;\boldsymbol{\theta}_1),\dots,f^{ex}_{s'}(s,a;\boldsymbol{\theta}_n)\right\}$ where $n \in \left\{n_{ens}, n_{fp}, n_{MC}\right\}$, we could just compute the variance among samples,

$$u(s,a) = \mathbb{V}ar[f^{ex}_{s'}(s,a;\boldsymbol{\theta})] = \mathbb{E}\left[\left(f^{ex}_{s'}(s,a;\boldsymbol{\theta}) - \mathbb{E}[f^{ex}_{s'}(s,a;\boldsymbol{\theta})]\right)^2\right], \tag{4.2}$$

or we could compute the entropy of this variance assuming that the samples follow a Gaussian distribution $f^{ex}_{s'}(s,a;\boldsymbol{\theta}) \sim \mathcal{N}\left(\mathbb{E}[f^{ex}_{s'}(s,a;\boldsymbol{\theta})], \mathbb{V}ar[f^{ex}_{s'}(s,a;\boldsymbol{\theta})]\right)$. In mathematical terms,

$$u(s,a) = \mathbb{H}[f^{ex}_{s'}(s,a;\boldsymbol{\theta})] = \frac{1}{2}\log\left(\mathbb{V}ar[f^{ex}_{s'}(s,a;\boldsymbol{\theta})]\right) + \frac{dim(\mathcal{S})}{2}(1+\log(2\pi)). \tag{4.3}$$

Otherwise, if the model - i.e. LaplaceSubnetLinearized- directly returns a Gaussian distribution (3.6), the simplest way is to compute its entropy as before.

Jensen-Rényi Divergence-. Other alternative is to measure novelty with Jensen-Rényi Divergence (JRD) as in [30]. This metric captures the amount of disagreement present in a mixture of multivariate Gaussians $\left\{N_{\boldsymbol{\theta}_1},\dots,N_{\boldsymbol{\theta}_n}\right\}$, where $N_{\boldsymbol{\theta}_i} \sim \mathcal{N}(\mu_i,\Sigma_i)$ refers to the Gaussian distribution predicted by the `base_model` (4.1) with weights $\boldsymbol{\theta}_i$. Its value may be computed from,

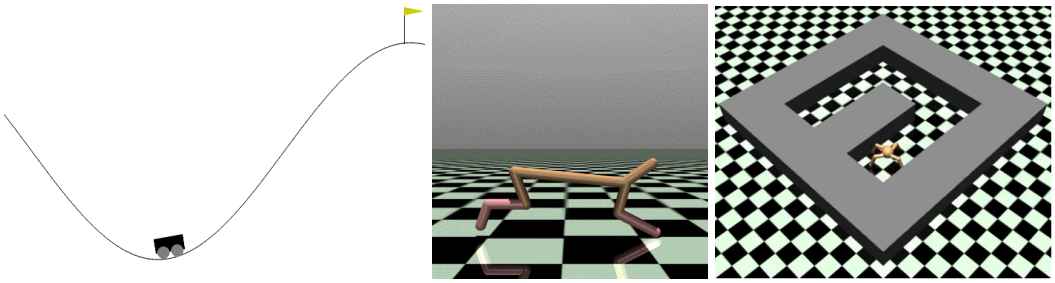
$$u(s,a) = JRD\left\{N_{\boldsymbol{\theta}_1},\dots,N_{\boldsymbol{\theta}_n}\right\} = -\log\left[\frac{1}{n^2}\sum_{i,j}^{n}\mathscr{D}(N_i,N_j)\right] - \frac{1}{n}\sum_{i=1}^{n}\frac{\log|\Sigma_i|}{2} - \frac{\log(2)dim(\mathcal{S})}{2}, \tag{4.4}$$

where

$$\mathscr{D}(N_i,N_j) = \frac{1}{|\Omega_{ij}|^{\frac{1}{2}}}\exp\left(-\frac{1}{2}\Delta_{ij}^T\Omega_{ij}^{-1}\Delta_{ij}\right),$$

and denoting $\Omega_{ij} = \Sigma_i + \Sigma_j$, $\Delta_{ij} = \mu_j - \mu_i$.

Particularly, the state-of-the-art method (MAX) that we compare with utilizes deep ensembles as a Bayesian model and leverages the Jensen-Rényi Divergence as a utility function [30].



(a) `MountainCar`. (b) `HalfCheetah`. (c) `AntMaze`.

Figure 4.4: Illustration of the tested RL environments.

Table 4.1: Basic information about the tested RL environments.

Environment	$dim(\mathcal{S})$	$dim(\mathcal{A})$	Tasks	n^{ex}_{steps}
`MountainCar`	2	1	`CaptureFlag`	2500
`HalfCheetah`	17	6	`Running` and `Flipping`	20000
`AntMaze`	27	8	`MazeCoverage`	12000

4.4 Experiments

4.4.1 Environments

In order to evaluate our approach to model-based active exploration, we conducted experiments on several standard environments using the MuJoCo physics engine [32] and the OpenAI Gym interface [31]. MuJoCo provides a fast and accurate simulation of rigid-body dynamics, which allows for realistic and challenging tasks to be formulated as MDPs. OpenAI Gym is a widely used platform for benchmarking RL algorithms, providing a standardized set of environments with well-defined reward structures and observation spaces. The environments we focused on in our experiments and their pertinent tasks were the following ones:
`MountainCar-`. In this environment a car is placed stochatically at the bottom of a valley and the agent controls it to reach a flag on top of the right hill (Figure 4.4a). The state space consists of the car's position and velocity, while the action space is a float representing the directional force applied on the car. The task is to reach the flag and, then, the reward is 100 if it reaches the flag and, otherwise, 0.
`HalfCheetah-`. It is a 2-dimensional robot consisting of 9 links and 8 joints connecting them (Figure 4.4b). The state space consists of the positional and velocity values of different body parts, while the action space contains six continuous actions representing the torques applied to the robot's joints. In this problem we considered two different tasks: (1) Running: move forward as faster as possible and (2) Flipping: perform flips.
`AntMaze-`. The Ant is a four-legged 3D robot that aims to move synchronously (Figure 4.4c). The state space consists of positional and velocity values of different body parts, whereas the action space contains the torques applied at the hinge joints (Figure 4.4c). The evaluation in this environment consists of computing the percentage of the maze covered across the exploration. As result, it will not be necessary the evaluation pipeline.

These tasks represent a wide range of complexity levels in terms of tasks difficulty and dimension of action and state spaces (Table 4.1). By testing our approach on these environments, we aimed to demonstrate its generality and effectiveness in a variety of settings.

4.4.2 Experimental details

Pipeline-. The number of exploration steps depends on the environment (Table 4.1), from which the first $n^{ex}_{warm} = 256$ steps correspond to warm-up ($n^{ex}_{warm} = 100$ for `MountainCar`). Besides, we recompute the exploration policy each $n_{pol} = 25$ steps and evaluate each $n_{eval} = 2000$ steps. First, to compute the exploration policy we first train the model and, afterwards, we run $n^{pol}_{eps} = 50$ episodes (the first $n^{pol}_{warm} = 3$ episodes as warm-up) along $n^{pol}_{steps} = 50$ steps where the consequence of $n_{act} = 128$ actions are computed simultaneously. Second, to evaluate the specific task we run $n_k = 3$ evaluation episodes in which we start computing the task policy ($n^{pol}_{eps} = 250$, $n^{pol}_{warm} = 3$, $n^{pol}_{steps} = 100$, $n_{act} = 128$) and, afterwards, we run $n^{ev}_{steps} = 100$ steps.
Transition Bayesian models-. The `base_model` consists of a MLP with 5 layers and 512 neurons per hidden layer. The input and the output sizes of the `base_model` are $dim(\mathcal{S}) + dim(\mathcal{A})$ and $2 * dim(\mathcal{S})$ respectively, where $dim(\cdot)$ refers to the dimension of the state/action space (Table 4.1). We take the *swish* activation function $h(x) = x\sigma(x)$ as non-linear transformation.

To compare the exploration performance of the models discussed in section 4.3.1, we will vary the models f^{ex} used in the exploration pipeline, while keeping the evaluation pipeline fixed to a single model f^{ev} (deep ensembles model). All the models are trained along 50 epochs, with a learning rate of 1e$-$3 and a batch size of 256. We consider Adam Optimizer to find $\boldsymbol{\theta}_{MAP}$ (2.4). The prior precision of the Bayesian weights is $\gamma^2 = 1$. Eventually, with the aim of making a fair comparison among the different models, we choose $n_{ens} = n_{fp} = n_{MC} = 32$. Specifically, Dropout is introduced in the middle hidden layer with a probability of $p = 0.25$. Furthermore, LaplaceSubnet is built with the $1k$ parameters with highest absolute values and, in general, the hyperparameters σ^2, γ^2 of all Laplace models are fitted minimizing the marginal log-likelihood along 50 epochs (section 3.1.1).

4.4.3 Results

`MountainCar-`. This environment is a simple problem in which, besides, the reward is not continuous and only takes values of 0 or 100, depending on whether the car reaches the flag or not. These two facts limit the analysis of the results. Therefore, although the results presented in Table 4.2 provide insights into the performance of each model, we do not believe they are significant enough to make robust conclusions. However, the simplicity of the problem allows us to visualize the uncertainty over each state along the exploration process (Figure 4.5). At plot sight, we may observe how the agent creates trajectories to cover the entire state space and how the model uncertainty vanishes in regions crossed by these trajectories.

`AntMaze-`. We may observe in Table 4.3 the performance of the different models. It is highlighted that some models only need 6000 exploration steps to cover the 80% of the maze, reaching the 85% at the end of the exploration (12000 steps). Nevertheless, it should be pointed out that with this metric we are evaluating the exploration accomplished in only 2 of the 27 state space dimensions. In other words, although it is a useful sign of the full exploration accomplished, it could be tricky since, perhaps, it covers the full maze but without exploring the range of velocities. In particular, we may visualize in Figure 4.6 an example of the exploration performed by LaplaceSubnetMC model across 3 runs. At Figure sight, it seems that once the ant reaches the end of the maze, it has some difficulties to come back to this region to keep on exploring. Perhaps, the model uncertainty fades too fast once it has been sparely explored and, thus, utility is not high enough to make the ant to come back.

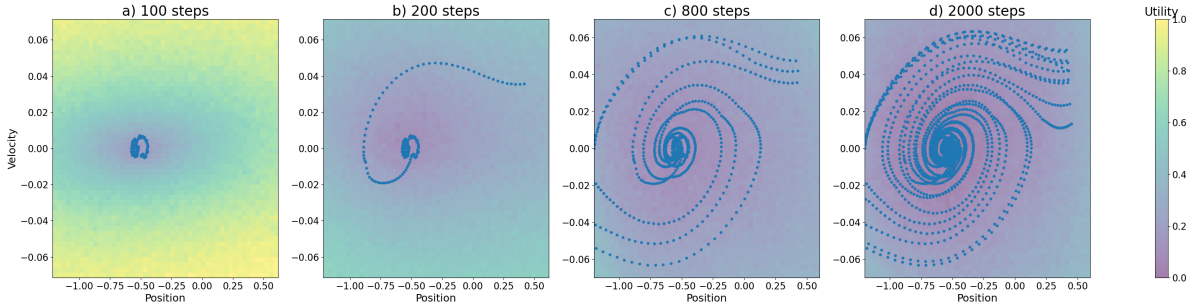
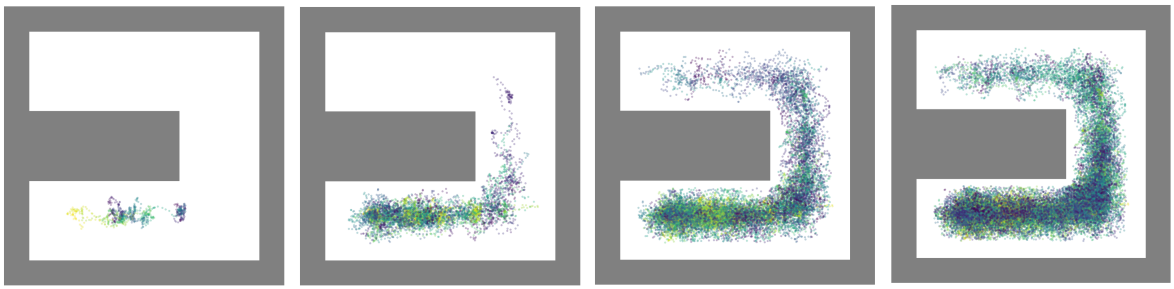


Figure 4.5: Illustration of `MountainCar` exploration by LaplaceLLMC model. The plots display the agent's state space discretized as a 2D grid with color indicating the average uncertainty of a state across all actions. The agent's trajectories are illustrated with dotted lines.



(a) 256 steps. (b) 2000 steps. (c) 6000 steps. (d) 12000 steps.

Figure 4.6: `AntMaze` exploration by LaplaceSubnetMC Rényi model across 3 runs. Chronological order of steps within an episode is encoded with the color spectrum, going from yellow (earlier) to purple (later).

`HalfCheetah-`. First, we may find in Table 4.4 a brief quantitative comparison among the different models and utilities tested by us. Some of them reach a reward higher than 330 and 600 for running and flipping task, respectively. Specifically, the best average between both tasks is achieved by LapSubnetMC model. This is the best example to motivate the significance of exploration in any RL problem. As it is observed, **after performing an accurate exploration of its state and action spaces, the agent is able to rapidly build a specific policy for each task whose performances are brilliant**. Comparing with respect to other exploration methods (Figure 4.8), note that some of our models outperform MAX (EnsRényi), which is the method in which this work is based. Even so, it is highlighted that its performance and ours are significantly better than other methods. However, these results alone do not capture real performance. To achieve this goal, we render the evaluation steps (watch videos in `https://github.com/cplou99/BayesianDL/tree/main/Applications`). These videos demonstrate exceptional task performance (Figure 4.7 shows a sequence solving Flipping task). Complementary, we compare in Appendix B.2 the models in terms of calibration.

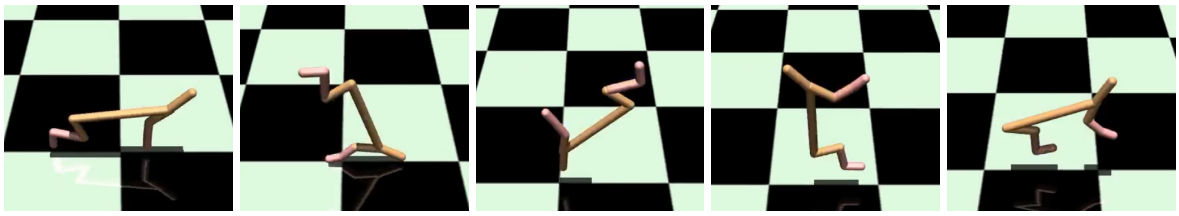


Figure 4.7: Sequence of a `HalfCheetah` flip after 20000 exploration steps performed by LapSubnetMC model. Videos may be found in `https://github.com/cplou99/BayesianDL/tree/main/Applications`.

Table 4.2: Experimental results in `MountainCar` environment. The metric is the percentage of evaluation sequences in which the car reaches the flag. Evaluation is performed each 500 exploration steps. Each result corresponds to the average over 3 runs.

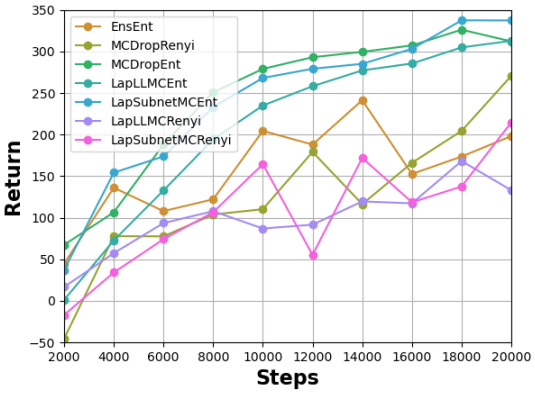
		Steps				
Bayesian Model	Utility	500	1000	1500	2000	Max
Deep ensembles	Rényi Div.	0.00	0.00	66.67	22.22	66.67
MCDropout	Rényi Div.	11.11	0.00	0.00	0.00	11.11
LaplaceLLMC	Rényi Div.	22.22	66.66	55.55	33.33	66.66
LaplaceSubnetMC	Rényi Div.	77.78	77.77	88.89	55.55	**88.89**
Deep ensembles	Entropy	33.33	66.67	66.67	55.55	66.67
MCDropout	Entropy	77.77	77.78	77.77	55.55	<u>77.77</u>
LaplaceLLMC	Entropy	55.55	66.66	22.22	55.55	66.66
LaplaceSubnetMC	Entropy	55.55	66.66	22.22	55.55	66.66
LaplaceSubnetLinear	Entropy	44.44	33.33	55.55	66.66	66.66

Table 4.3: Experimental results in `AntMaze` environment. The metric is the percentage of the maze covered with the exploration steps. Each result corresponds to the average over 3 runs. Best result in bold, second best result underlined.

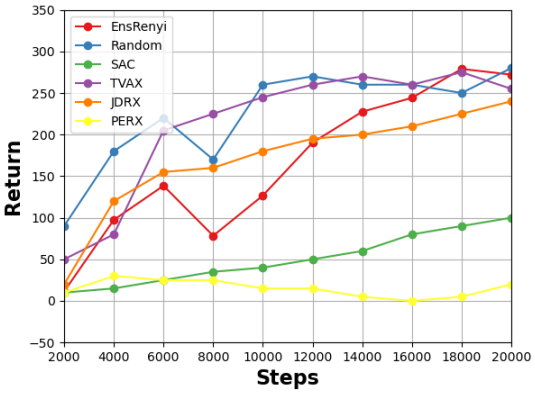
		Steps						
Bayesian Model	Utility	256	2000	4000	6000	8000	10000	12000
Deep ensembles	Rényi	10.42	40.18	64.29	78.57	81.85	83.93	**85.42**
MCDropout	Rényi	9.52	58.33	73.81	78.27	82.74	83.93	<u>85.12</u>
LaplaceLLMC	Rényi	10.12	28.27	49.11	60.72	74.70	80.06	81.55
LaplaceSubnetMC	Rényi	10.51	35.32	47.62	55.26	65.48	70.34	74.11
Deep ensembles	Entropy	9.23	33.48	41.52	46.88	48.66	50.00	52.24
MCDropout	Entropy	9.82	33.33	41.97	45.84	47.32	52.09	52.98
LaplaceLLMC	Entropy	9.22	29.17	42.26	53.57	61.90	70.24	73.51
LaplaceSubnetMC	Entropy	10.12	44.64	50.00	55.95	64.29	67.26	73.51

Table 4.4: Experimental results in `HalfCheetah` environment with Running and Flipping tasks. We compute the maximum reward of the evaluations performed each 2000 exploration steps. In other words, in this table we show the maximum of each line that appear in Figure 4.8. Specifically, these results are the average across 3 runs. Best result in bold, second best result underlined.

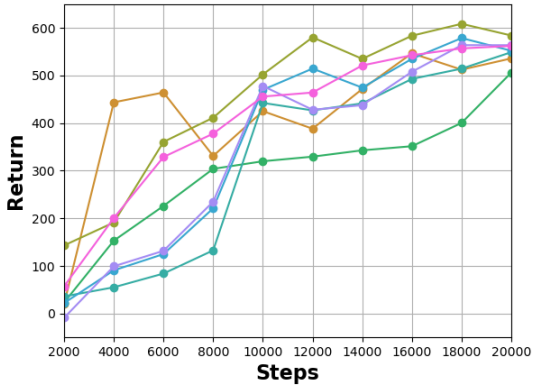
		Maximum Reward		
Model	Utility	Running	Flipping	Average
Deep ensembles	Rényi	278.9	<u>590.9</u>	<u>431.5</u>
MCDropout	Rényi	270.6	**608.3**	427.2
LapLLMC	Rényi	168.1	563.5	365.8
LapSubnetMC	Rényi	214.6	561.8	388.2
Deep ensembles	Entropy	241.3	546.3	354.3
MCDropout	Entropy	<u>326.1</u>	505.9	409.0
LapLLMC	Entropy	312.7	548.7	430.7
LapSubnetMC	Entropy	**337.4**	578.4	**457.9**



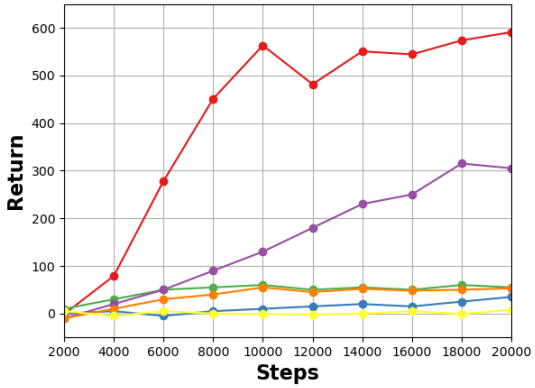
(a) Ours: Running task performance.



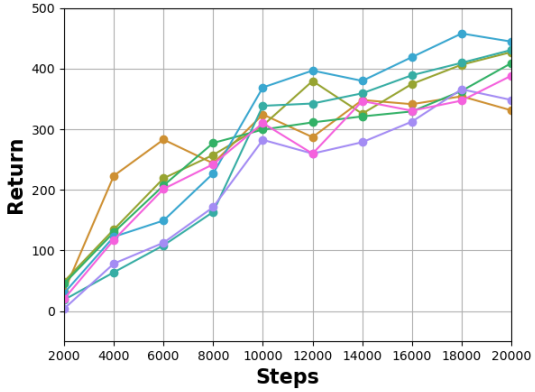
(b) SOTA: Running task performance.



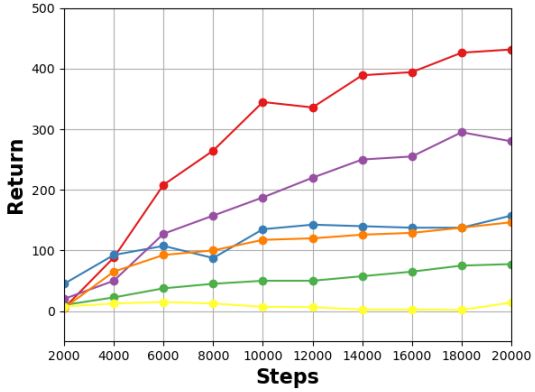
(c) Ours: Flipping task performance.



(d) SOTA: Flipping task performance.



(e) Ours: Average performance.



(f) SOTA: Average performance.

Figure 4.8: Performance comparison among the different methods. On the left side, the figures correspond to our approaches, while on the right side, the results of the state-of-the-art (SOTA) methods are showcased. The results of SOTA methods are taken from [30].

Chapter 5

Conclusions, challenges and future work

5.1 Conclusions

We have shown that Laplace Approximation provides a more reliable way to measure uncertainty in deep learning models and outperforms current baselines in out-distribution predictions in simple problems. However, in more complex problems, further approximations are required to deal with memory and time constraints which limit its power. Nevertheless, our experiments indicate that Laplace Approximation clearly reaches state-of-the-art performance levels, or even surpasses them.

Regarding the Reinforcement Learning problem studied, we have conducted experiments comparing different Bayesian models and other exploration techniques of different nature, such as SAC or Random. Our results demonstrate that the highest performance in the environments tested are achieved by the models suggested by us (Laplace and MC-Dropout).

Overall, the findings of this work demonstrate the potential of Laplace Approximation as a tool to take into account to measure uncertainty in current Deep Learning problems.

5.2 Challenges and limitations

The challenges and limitations encountered during this work were the following ones. Firstly, this was the first time that I tackled such a complex problem, which involved learning about various libraries, including Laplace, PyTorch or Gym, and then integrating them into a cohesive environment. However, a significant difficulty that arose was the compatibility issues with different library versions, particularly with Python. Additionally, understanding the pipeline code taken as reference of the RL problem was puzzling.

Furthermore, the Laplace library code was another challenge. The linearization prediction technique was carefully analysed to be optimized. In particular, Jacobians computation was identified as the main bottleneck, and potential optimization techniques for this could be explored further, specially when only a Subnetwork is considered as Bayesian.

The primary challenge in RL problems is the time it takes to run the experiments, which complicates the hyperparameter selection process and restricts the ability to perform more tests.

5.3 Future work

Based on the limitations and challenges encountered in this work, there are several future directions that can be pursued. One possible direction is to explore different ways to optimize the

Jacobians computation in the Laplace library. This computation is likely to be parallelizable. Thus, it could potentially improve the performance of the Laplace Approximation method in more complex problems, since we could use the linearization technique which arose the best results.

Regarding the studied reinforcement learning problem, next steps include to change the transition model f^{ev} in the evaluation pipeline and to obtain more metrics (calibration and time) to compare them.

Finally, it may be worthwhile to investigate the use of Laplace Approximation in other problems such as active learning or mapping.

Bibliography

[1] Moloud Abdar, Farhad Pourpanah, Sadiq Hussain, Dana Rezazadegan, Li Liu, Mohammad Ghavamzadeh, Paul Fieguth, Xiaochun Cao, Abbas Khosravi, U Rajendra Acharya, et al. A review of uncertainty quantification in deep learning: Techniques, applications and challenges. *Information Fusion*, 76:243–297, 2021.

[2] Jiwoong Choi, Dayoung Chun, Hyun Kim, and Hyuk-Jae Lee. Gaussian yolov3: An accurate and fast object detector using localization uncertainty for autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 502–511, 2019.

[3] Sebastian Brechtel, Tobias Gindele, and Rüdiger Dillmann. Probabilistic decision-making under uncertainty for autonomous driving using continuous pomdps. In *17th international IEEE conference on intelligent transportation systems (ITSC)*, pages 392–399. IEEE, 2014.

[4] Hao Wang and Dit-Yan Yeung. Towards bayesian deep learning: A framework and some existing methods. *IEEE Transactions on Knowledge and Data Engineering*, 28(12):3395–3408, 2016.

[5] Laurent Valentin Jospin, Hamid Laga, Farid Boussaid, Wray Buntine, and Mohammed Bennamoun. Hands-on bayesian neural networks—a tutorial for deep learning users. *IEEE Computational Intelligence Magazine*, 17(2):29–48, 2022.

[6] Wesley J Maddox, Pavel Izmailov, Timur Garipov, Dmitry P Vetrov, and Andrew Gordon Wilson. A simple baseline for bayesian uncertainty in deep learning. *Advances in Neural Information Processing Systems*, 32, 2019.

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

[8] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *International conference on machine learning*, pages 1613–1622. PMLR, 2015.

[9] Javier Rodríguez-Puigvert, Rubén Martínez-Cantín, and Javier Civera. Bayesian deep neural networks for supervised learning of single-view depth. *IEEE Robotics and Automation Letters*, 7(2):2565–2572, 2022.

[10] John Asmuth, Lihong Li, Michael L Littman, Ali Nouri, and David Wingate. A bayesian sampling approach to exploration in reinforcement learning. *arXiv preprint arXiv:1205.2664*, 2012.

[11] Xuran Pan, Zihang Lai, Shiji Song, and Gao Huang. Activenerf: Learning where to see with uncertainty estimation. In *European Conference on Computer Vision*, pages 230–246. Springer, 2022.

[12] Agustinus Kristiadi, Matthias Hein, and Philipp Hennig. Being bayesian, even just a bit, fixes overconfidence in relu networks. In *International conference on machine learning*, pages 5436–5446. PMLR, 2020.

[13] David JC MacKay. Bayesian interpolation. *Neural computation*, 4(3):415–447, 1992.

[14] Hippolyt Ritter, Aleksandar Botev, and David Barber. A scalable laplace approximation for neural networks. In *International Conference on Learning Representations*, 2018.

[15] Javier Antorán, David Janz, James U Allingham, Erik Daxberger, Riccardo Rb Barbano, Eric Nalisnick, and José Miguel Hernández-Lobato. Adapting the linearised laplace model evidence for modern deep learning. In *International Conference on Machine Learning*, pages 796–821. PMLR, 2022.

[16] Javier Antorán, Shreyas Padhy, Riccardo Barbano, Eric Nalisnick, David Janz, and José Miguel Hernández-Lobato. Sampling-based inference for large linear models, with application to linearised laplace. *arXiv preprint arXiv:2210.04994*, 2022.

[17] Erik Daxberger, Agustinus Kristiadi, Alexander Immer, Runa Eschenhagen, Matthias Bauer, and Philipp Hennig. Laplace redux-effortless bayesian deep learning. *Advances in Neural Information Processing Systems*, 34:20089–20103, 2021. `https://github.com/AlexImmer/Laplace`.

[18] Andrew YK Foong, Yingzhen Li, José Miguel Hernández-Lobato, and Richard E Turner. 'in-between'uncertainty in bayesian neural networks. *arXiv preprint arXiv:1906.11537*, 2019.

[19] Mohammad Emtiyaz E Khan, Alexander Immer, Ehsan Abedi, and Maciej Korzepa. Approximate inference turns deep networks into gaussian processes. *Advances in neural information processing systems*, 32, 2019.

[20] Mijung Park, Greg Horwitz, and Jonathan Pillow. Active learning of neural response functions with gaussian processes. *Advances in neural information processing systems*, 24, 2011.

[21] MA Wiering and Jürgen Schmidhuber. Efficient model-based exploration. 1998.

[22] Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

[23] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. *Advances in neural information processing systems*, 29, 2016.

[24] Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. *Advances in neural information processing systems*, 29, 2016.

[25] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

[26] Manuel Lopes, Tobias Lang, Marc Toussaint, and Pierre-Yves Oudeyer. Exploration in model-based reinforcement learning by empirically estimating learning progress. *Advances in neural information processing systems*, 25, 2012.

[27] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR, 2017.

[28] Jürgen Schmidhuber. Curious model-building control systems. In *Proc. international joint conference on neural networks*, pages 1458–1463, 1991.

[29] Deepak Pathak, Dhiraj Gandhi, and Abhinav Gupta. Self-supervised exploration via disagreement. In *International conference on machine learning*, pages 5062–5071. PMLR, 2019.

[30] Pranav Shyam, Wojciech Jaśkowski, and Faustino Gomez. Model-based active exploration. In *International conference on machine learning*, pages 5779–5788. PMLR, 2019.

[31] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[32] 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*, pages 5026–5033. IEEE, 2012.

[33] Emery D Berger, Sam Stern, and Juan Altmayer Pizzorno. Triangulating python performance issues with scalene. *arXiv preprint arXiv:2212.07597*. `https://github.com/plasma-umass/scalene`, year=2022.

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

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

[36] Alex Kendall, Vijay Badrinarayanan, and Roberto Cipolla. Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. *arXiv preprint arXiv:1511.02680*, 2015.

[37] Jishnu Mukhoti and Yarin Gal. Evaluating bayesian deep learning methods for semantic segmentation. *arXiv preprint arXiv:1811.12709*, 2018.

[38] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017.

[39] Zongyao Lyu, Nolan Gutierrez, Aditya Rajguru, and William J Beksi. Probabilistic object detection via deep ensembles. In *Computer Vision–ECCV 2020 Workshops: Glasgow, UK, August 23–28, 2020, Proceedings, Part VI 16*, pages 67–75. Springer, 2020.

[40] Loic Le Folgoc, Vasileios Baltatzis, Sujal Desai, Anand Devaraj, Sam Ellis, Octavio E Martinez Manzanera, Arjun Nair, Huaqi Qiu, Julia Schnabel, and Ben Glocker. Is mc dropout bayesian? *arXiv preprint arXiv:2110.04286*, 2021.

[41] Erik Daxberger, Eric Nalisnick, James Urquhart Allingham, Javier Antorán, and José Miguel Hernández-Lobato. Bayesian deep learning via subnetwork inference, 2022.

[42] Agustinus Kristiadi, Matthias Hein, and Philipp Hennig. Being bayesian, even just a bit, fixes overconfidence in relu networks, 2020.

[43] David JC MacKay. The evidence framework applied to classification networks. *Neural computation*, 4(5):720–736, 1992.

[44] Nicol N Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural computation*, 14(7):1723–1738, 2002.

[45] Marius Hobbhahn, Agustinus Kristiadi, and Philipp Hennig. Fast predictive uncertainty for classification with bayesian deep networks. In *Uncertainty in Artificial Intelligence*, pages 822–832. PMLR, 2022.

[46] Eddy Ilg, Ozgun Cicek, Silvio Galesso, Aaron Klein, Osama Makansi, Frank Hutter, and Thomas Brox. Uncertainty estimates and multi-hypotheses networks for optical flow. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 652–667, 2018.

[47] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 7559–7566. IEEE, 2018.

[48] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

[49] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[50] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[51] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

[52] Joel Lehman and Kenneth O Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223, 2011.

[53] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

Appendix A

Complementary theory

A.1 Supervised learning: Regression and Classification

There are two main problems within supervised learning: regression and classification.

Regression

The core of regression models is to estimate a continuous target variable $\mathbf{y} \in \mathbb{R}^N$ (e.g. the price of a stock tomorrow) by a function $f(\mathbf{X};\boldsymbol{\theta})$, where $\boldsymbol{\theta} \in \mathbb{R}^D$ is a set of parameters that must be estimated to fit data. In mathematical terms,

$$\mathbf{y} = f(\mathbf{X};\boldsymbol{\theta}) + \boldsymbol{\varepsilon}, \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2 I_D), \tag{A.1}$$

where $\boldsymbol{\varepsilon}$ is the Gaussian noise error and $\sigma^2 > 0$ its variance. As it is noticed, we are considering an homoscedastic model, that is, the variance or aleatoric uncertainty is identical for all data. Moreover, samples are independent and identically distributed (**i.i.d**).

$$\mathbb{E}[\mathbf{y}] = f(\mathbf{X};\boldsymbol{\theta}), \quad \mathbb{V}ar[\mathbf{y}] = \sigma^2 I_D. \tag{A.2}$$

Therefore, we may assume that observed data follow a Gaussian distribution whose mean and variance is determined by (A.2). Consequently, the likelihood of data $\mathcal{L}(\mathcal{D}|\boldsymbol{\theta})$ is given by,

$$\mathcal{L}(\mathcal{D}|\boldsymbol{\theta}) = p(\mathcal{D}|\boldsymbol{\theta}) = \prod_{n=1}^{N} p(\mathbf{y}_n | f(\mathbf{x}_n;\boldsymbol{\theta})) = \prod_{n=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{ -\frac{1}{2}\left(\frac{\mathbf{y}_n - f(\mathbf{x}_n;\boldsymbol{\theta})}{\sigma} \right)^2 \right\}. \tag{A.3}$$

However, we usually work with the **negative log-likelihood** $\ell(\mathcal{D}|\boldsymbol{\theta}) = -\log \mathcal{L}(\mathcal{D}|\boldsymbol{\theta})$ as loss function since it facilitates the computation of the **Maximum Likelihood Estimation (MLE)** of the parameters $\boldsymbol{\theta}_{\mathbf{MLE}}$,

$$\frac{\partial}{\partial \boldsymbol{\theta}} \ell(\mathcal{D}|\boldsymbol{\theta}) \bigg|_{\boldsymbol{\theta}_{\mathbf{MLE}}} = 0. \tag{A.4}$$

Depending on the complexity of the model, either we will be able to solve analytically (A.4) or we will require a numerical method. We may observe this difference in Examples 2 and 3. Afterwards, we could make predictions $\hat{\mathbf{y}} = f(\mathbf{X};\boldsymbol{\theta}_{\mathbf{MLE}})$ and test its performance through some known metrics as Mean Squared Error (MSE) or Negative Log-Likelihood (NLL).

Example 2. Linear Regression-. The model is determined by $f(\mathbf{X};\boldsymbol{\theta}) = \mathbf{X}\mathbf{w} + b$, where $\mathbf{w}$ and b receive the name of weights and bias, respectively. Hence, the negative log-likelihood is a convex function over $\boldsymbol{\theta}$ and (A.4) has an analytical closed form $\boldsymbol{\theta}_{\mathbf{MLE}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$.

Example 3. Neural network-. Let us consider a multilayer perceptron (MLP) with L layers defined as the composition of L functions $f(\mathbf{X};\boldsymbol{\theta}) = f_L(f_{L-1}(\cdots(f_1(\mathbf{X}))))$ (**forward pass**), where $f_j(\mathbf{X}) = \phi(\mathbf{X})\mathbf{w_j} + b_j,\ 1 \leq j \leq L$. Besides, $\phi(\cdot)$ denotes a non-linear transformation (**activation function**). In this scenario, the loss is not longer a convex function and we require an optimizer as **Stochastic Gradient Descent** (SGD) to find the global minimum of $\ell(\mathcal{D}|\boldsymbol{\theta})$. The basic idea of SGD is to update the weights and biases of the network in the direction that minimizes the loss function. This is done by computing the gradient of the loss function with respect to the weights and biases (**backpropagation**), and then taking a step in the opposite direction.

Classification

Classification is used to predict a categorical label $\mathbf{y} \in \{-1,1\}^N$ (e.g. "spam" or "not spam" for an email). The algorithm is trained to assign a new input to one of a set of predefined categories. In classification, the likelihood function takes one of the following known expressions in order to facilitate the subsequent computations:

- In binary classification, sigmoid function $\sigma(x)$ can be used to model the probability of the instance belong to the positive class. In this way, likelihood is often deduced from Bernoulli distribution,

$$\mathcal{L}(\mathcal{D}|\boldsymbol{\theta}) = \prod_{n=1}^{N} Ber(\mathbf{y}_n|\sigma(f(\mathbf{x}_n;\boldsymbol{\theta}))), \quad \sigma(f(\mathbf{x}_n;\boldsymbol{\theta})) = \frac{1}{1+\exp(-f(\mathbf{x}_n;\boldsymbol{\theta}))}. \tag{A.5}$$

- In multi-class classification, there are as many outputs as classes (m). Softmax function measures the probability of the instance belong to each class. In this case, likelihood is computed assuming a Categorical distribution,

$$\mathcal{L}(\mathcal{D}|\boldsymbol{\theta}) = \prod_{n=1}^{N} Cat(\mathbf{y}_n|Softmax(f(\mathbf{x}_n;\boldsymbol{\theta}))), \ \ Softmax(f(\mathbf{x}_n;\boldsymbol{\theta})) = \frac{\exp(f(\mathbf{x}_n;\boldsymbol{\theta}))}{\sum_{k=1}^{m}\exp(f(\mathbf{x}_n;\boldsymbol{\theta}))}. \tag{A.6}$$

Usually, both log-likelihoods receive the name of **cross-entropy loss** which present the same challenges as in regression to find $\boldsymbol{\theta}_{\mathbf{MLE}}$. With the aim of testing the performance, we could measure accuracy or NLL. In addition to performance, we could evaluate the **calibration** of the model, that is, the reliability of a model's probability estimates. Calibration metrics measure the correspondence between the predicted probabilities and the true outcomes.

A.2 Algorithms

In this section, we provide the pseudocodes of the exploration, evaluation and policy pipelines explained in Section 4.2.2.

Algorithm 1: Exploration. Pseudocode of the exploration pipeline (Section 4.2.2).

Data: env, f^{ex}, f^{ev}, Utility
Hyperparams: $n^{ex}_{steps}, n^{ex}_{warm}, n_{eval}, n_{pol}$
Result: rewards
$\Phi = \emptyset, rewards = \emptyset, s_0 =$ env.init_state;
for $i \leftarrow 0$ **to** n^{ex}_{steps} **do**
  if $i < n^{ex}_{warm}$ **then**
    $a_i \leftarrow \text{SampleAction}(\mathcal{A}, 1)$;
  else
    if $i \,\%\, n_{eval} = 0$ **then**
      $reward \leftarrow \text{Evaluation}(\text{env}, f^{ev}, \Phi)$;
      $rewards \leftarrow rewards \cup \{i, reward\}$;
    end
    if $i = n^{ex}_{warm} \,||\, i \,\%\, n_{pol} = 0$ **then**
      $\pi^{ex} \leftarrow \text{Policy}(\Phi, s_i, f^{ex}, \text{Utility})$;
    end
    $a_i \leftarrow \pi^{ex}(s_i)$;
  end
  $s'_i \leftarrow \text{env.Step}(s_i, a_i)$;
  $\Phi \leftarrow \Phi \cup \{s_i, a_i, s'_i\}$;
  $s_{i+1} \leftarrow s'_i$;
end

Algorithm 2: Evaluation. Pseudocode of the evaluation pipeline (Section 4.2.2).

Data: env, f^{ev}, Φ
Hyperparams: n_k, n^{ev}_{steps}
Result: reward
$reward \leftarrow \emptyset, s_0 \leftarrow$ env.init_state;
for *task in env.tasks* **do**
  $R_{task} \leftarrow task$.Reward;
  $r_{task} \leftarrow \emptyset$;
  for $k \leftarrow 0$ **to** n_k **do**
    $r_k \leftarrow 0$;
    $\pi^{ev} \leftarrow \text{Policy}(\Phi, s_0, f^{ev}, R_{task})$;
    for $i \leftarrow 0$ **to** n^{ev}_{steps} **do**
      $a_i \leftarrow \pi^{ev}(s_i)$;
      $s'_i \leftarrow \text{env.Step}(s_i, a_i)$;
      $r_k \leftarrow r_k + R_{task}(s_i, s'_i, a_i)$;
      $s_{i+1} \leftarrow s'_i$;
    end
    $r_{task} \leftarrow r_{task} \cup r_k$
  end
  $reward \leftarrow reward \cup r_{task}$
end

Algorithm 3: Policy. Pseudocode of the Policy creation pipeline (Section 4.2.2).

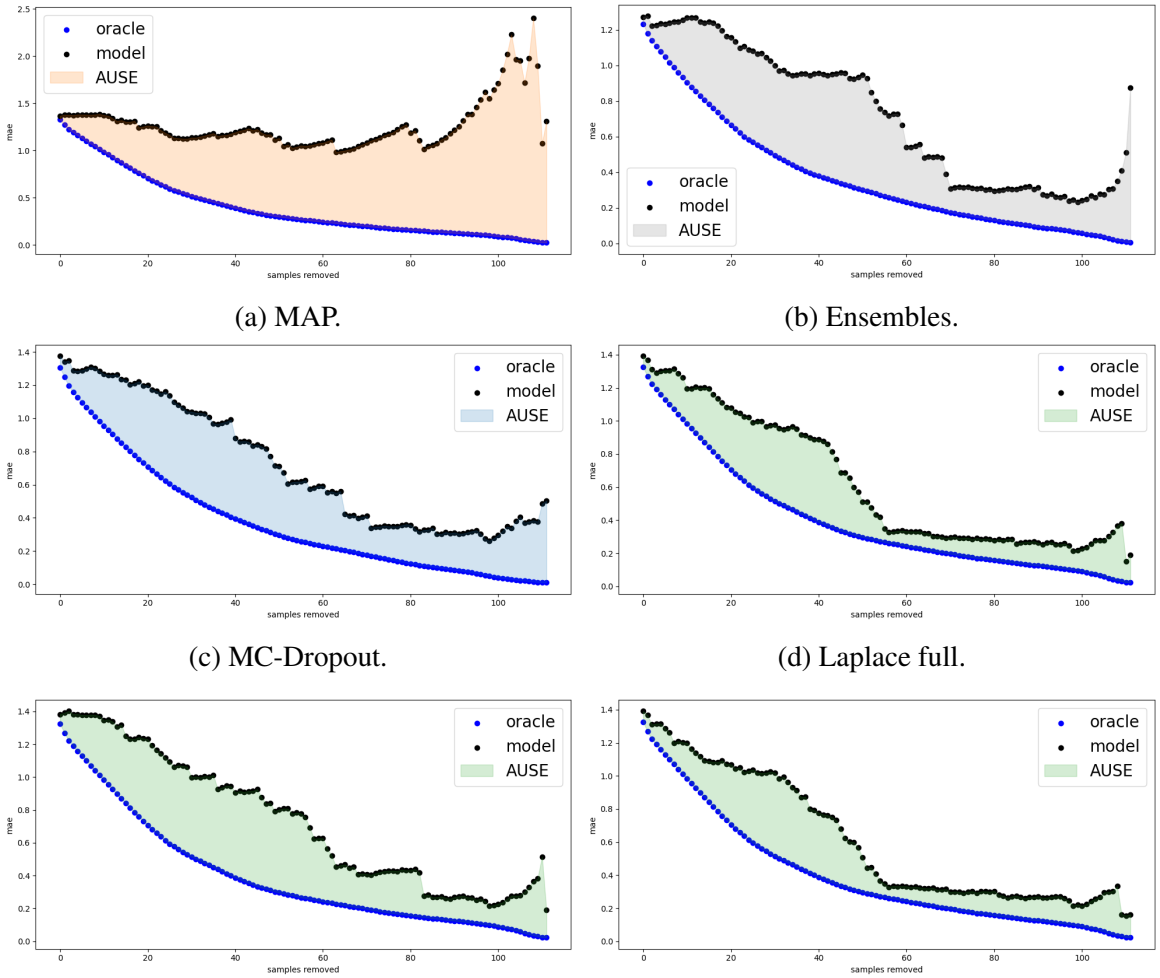
Data: Φ, s_{init}, f, R
Hyperparams: $n^{pol}_{eps}, n^{pol}_{steps}, n^{pol}_{warm}, n_{act}$
Result: π
$f \leftarrow \text{FitModel}(f, \Phi)$;
$\pi \leftarrow$ Policy.Init();
for $ep \leftarrow 0$ **to** n^{pol}_{eps} **do**
  $\mathbf{s_0} \leftarrow \text{Repeat}(s_{init}, n_{act})$;
  for $i \leftarrow 0$ **to** n^{pol}_{steps} **do**
    if $ep < n^{pol}_{warm}$ **then**
      $\mathbf{a_i} \leftarrow \text{SampleAction}(\mathcal{A}, n_{act})$;
    else
      $\mathbf{a_i} \leftarrow \pi(\mathbf{s_i})$;
    end
    $\mathbf{m_i}, \mathbf{v_i} \leftarrow f(\mathbf{s_i}, \mathbf{a_i})$;
    $\mathbf{s'_i} \leftarrow \mathcal{N}(\mathbf{m_i}, \mathbf{v_i})$.Sample();
    $\mathbf{r_i} \leftarrow R(\mathbf{s_i}, \mathbf{s'_i}, \mathbf{m_i}, \mathbf{v_i}, \mathbf{a_i})$;
    $\pi \leftarrow \text{SAC.Update}(\pi, \mathbf{s_i}, \mathbf{s'_i}, \mathbf{a_i}, \mathbf{r_i})$;
    $\mathbf{s_{i+1}} \leftarrow \mathbf{s'_i}$;
  end
end

Appendix B

Additional Results

B.1 AUSE benchmark datasets

In this section, you will find the AUSE plots of the benchmarks described in Section 3.2.



(a) MAP.

(b) Ensembles.

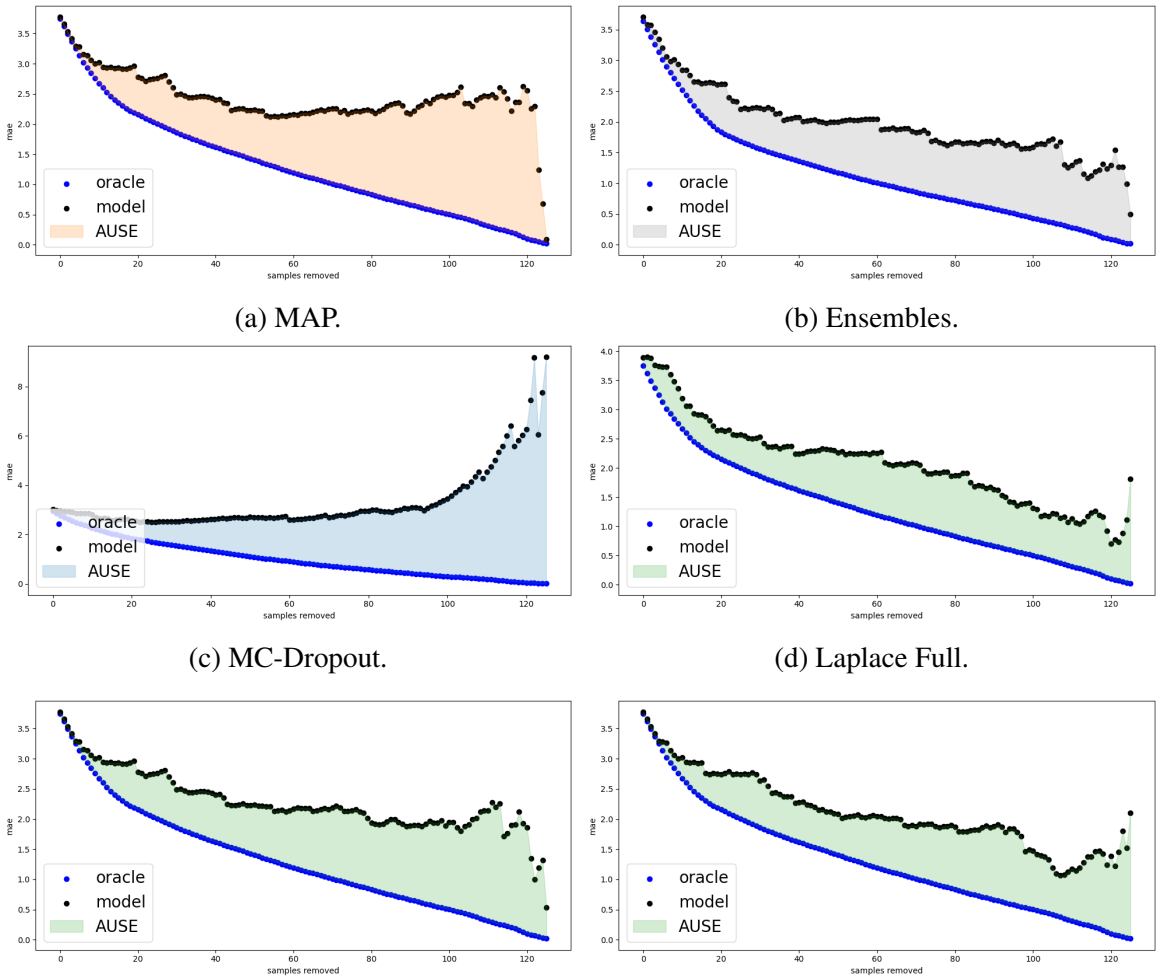
(c) MC-Dropout.

(d) Laplace full.

(e) Laplace Last Layer.

(f) Laplace Subnet.

Figure B.1: AUSE (Area Under Sparsification Error) curves of the different models in the Simulated Regression Dataset.



(a) MAP.

(b) Ensembles.

(c) MC-Dropout.

(d) Laplace Full.

(e) Laplace Last Layer.

(f) Laplace Subnetwork.

Figure B.2: AUSE (Area Under Sparsification Error) curves of the different models in Boston Dataset.

Table B.1: Calibration results in `HalfCheetah` environment. It corresponds to the AUSE values of the models trained with the final exploration buffer and performing 100 episodes (100 steps per episode) with the corresponding policy of the task. Ground truth is obtained from the simulator. Best result in bold, second best result underlined.

		Calibration (AUSE)		
Model	Utility	Running	Flipping	Average
Deep ensembles	Rényi	495.1	488.0	<u>491.0</u>
MCDropout	Rényi	751.4	723.7	737.55
LapLLMC	Rényi	830.4	**389.6**	610.0
LapSubnetMC	Rényi	624.9	540.4	582.6
Deep ensembles	Entropy	**339.3**	<u>443.5</u>	**391.4**
MCDropout	Entropy	727.3	651.1	689.2
LapLLMC	Entropy	930.4	632.3	781.3
LapSubnetMC	Entropy	<u>377.5</u>	632.3	504.9

B.2 Calibration `Half Cheetah`

With the aim of providing another metric to compare the bayesian transition models employed in `HalfCheetah` environment (Section 4.4), we study their calibration. To achieve this goal, once the exploration has finished (20k steps) we train each model with its corresponding exploration buffer. Afterwards, we leverage this trained model to build a policy (Policy pipeline) oriented to solve a task (running or flipping). Once we get this policy, we run 100 episodes, each episode starting from a random initial state, along 100 steps. In each step, we compute the model's predictions (next state and uncertainty) and, the ground truth provided by `Gym`. Therefore, we may measure the uncertainty calibration with the errors by way of AUSE metric (Table B.1). Complementary, some AUSE plots may be found in Figure B.3.

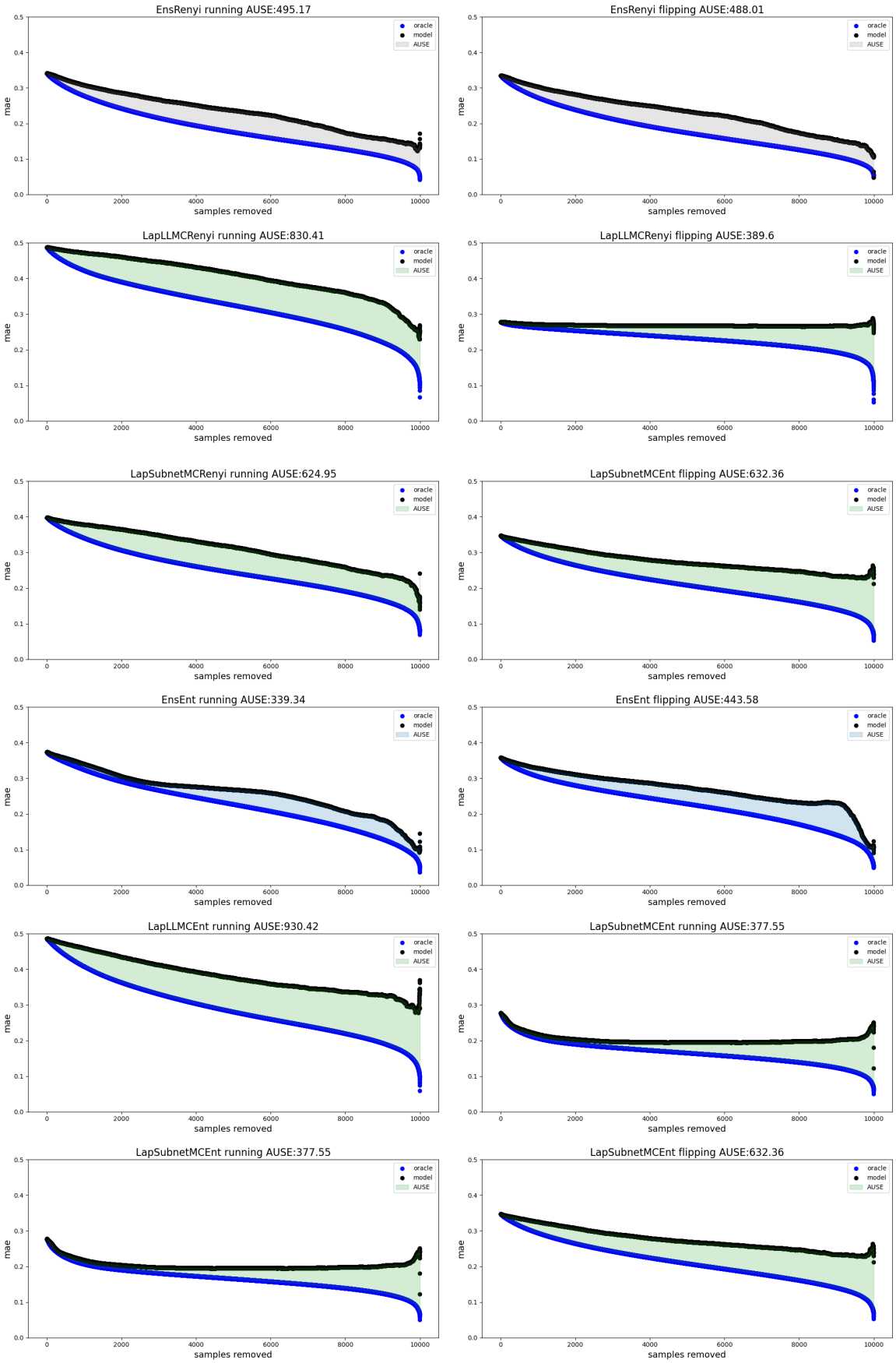


Figure B.3: AUSE curves of the different bayesian models in `HalfCheetah` environment.