Advances in Probabilistic Deep Learning and Their Applications

Erik Alexander Daxberger

Department of Engineering
University of Cambridge

This thesis is submitted for the degree of
Doctor of Philosophy

Fitzwilliam College

August 2023
Declaration

This thesis is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the Preface and specified in the text. I further state that no substantial part of my thesis has already been submitted, or, is being concurrently submitted for any such degree, diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. It does not exceed the prescribed word limit for the relevant Degree Committee.

Erik Alexander Daxberger
August 2023
Advances in Probabilistic Deep Learning and Their Applications

Erik Alexander Daxberger

Deep learning and probabilistic modeling are two machine learning paradigms with complementary benefits. Probabilistic deep learning aims to unify the two, with the potential to offer compelling theoretical properties and practical functional benefits across a variety of problems. This thesis provides contributions to the methodology and application of probabilistic deep learning. In particular, we develop new methods to address four different application domains.

The first application is out-of-distribution detection. Neural networks tend to make unreliable predictions when the data distribution changes after training. To address this, we propose a new probabilistic deep learning method based on a Bayesian variational autoencoder, where a full distribution is inferred over the model parameters, rather than just a point estimate. We then use information-theoretic measures to detect out-of-distribution inputs with this model.

The second application is data-efficient optimization. Many science and engineering problems require optimizing a costly black-box function over a high-dimensional, structured space. To tackle this, we develop a new probabilistic deep learning method that efficiently optimizes the function in the low-dimensional, continuous latent space of a variational autoencoder. We propose to periodically retrain the model to keep the latent manifold useful for optimization.

The third application is neural network calibration. Neural networks tend to be poorly calibrated on inputs not seen during training. To avoid overconfidence, models must be able to quantify their uncertainty. To this end, we develop a new probabilistic deep learning method that performs Bayesian inference over just a subset of a neural network’s parameters. We propose a way to choose such subnetworks to faithfully preserve the model’s predictive uncertainty.

The fourth application is continual deep learning. Neural networks often catastrophically forget previously learned tasks when trained on new tasks. To enable models to learn across task sequences, we introduce a new probabilistic deep learning method that unifies two popular continual learning approaches: Bayesian weight regularization and experience replay. Our method explicitly aims to approximate the model obtained from batch-training on all tasks jointly.

Overall, the goals of this thesis are twofold. Firstly, we aim to develop new methods at the intersection of probabilistic modeling and deep learning that combine their respective
advantages. Secondly, we aim to demonstrate the practical potential of those probabilistic deep learning methods by applying them to advance the diverse application areas mentioned before.
Acknowledgements

First and foremost, I would like to thank my supervisor, José Miguel Hernández-Lobato, for giving me the opportunity to pursue this PhD. Throughout the PhD, Miguel gave me the freedom to pursue the research of my interest, while providing guidance whenever I needed it.

I also have the pleasure of thanking my supervisors during several very enjoyable and formative research visits, collaborations and internships. Bernhard Schölkopf provided me with the unique opportunity to spend the last bits of my PhD at the MPI-IS in Tübingen, Germany. It was also a great pleasure to work with Emtiyaz Khan at the RIKEN Center for AIP in Tokyo, Japan (although COVID prevented me from visiting the lab in person, unfortunately). Furthermore, I am grateful to Matthias Seeger for his supervision during my first taste of industry at Amazon in Berlin, Germany. Finally, I would like to thank Marcin Eichner and Michael Emmersberger for their mentorship during my time at Apple in Zurich, Switzerland.

I have been privileged to work with and learn from amazing collaborators during my PhD. I would like to thank Austin Tripp, Eric Nalisnick, Javier Antorán, James Allingham, Alexander Immer, Runa Eschenhagen, Agustinus Kristiadi, Matthias Bauer, Philipp Hennig, David Janz, Riccardo Barbano, Siddharth Swaroop, Kazuki Osawa, Richard E. Turner, and Rio Yokota.

Furthermore, I want to express my gratitude to everyone at the Computational and Biological Learning Lab at the Engineering Department in Cambridge as well as at the Empirical Inference Department at the MPI-IS in Tübingen for providing the best academic and social environments I could have asked for (and for helping me keep my sanity throughout the COVID lockdowns). In the same vein, I want to thank the great community at Fitzwilliam College, who provided me a home away from home during my time in Cambridge, as well as all the team members at RIKEN AIP, Amazon Berlin and Apple Zurich, who made my PhD experience truly special.

I would also like to thank the EPSRC, Qualcomm, and the Cambridge Trust for generously funding my PhD, Adrian Weller for providing helpful advice as my graduate advisor, as well as David Krueger and Günter Klambauer for their efforts in examining this thesis.
Pursuing this PhD would not have been possible without the unconditional and continued support of my friends and family, and in particular of my parents and grandparents, which I am immensely thankful for. Lastly, I am especially grateful to Anna and our little daughter Lotte for their love and patience throughout the past year(s).
# Table of Contents

1 Introduction
   1.1 List of Publications ........................................... 3
   1.2 Overview of Thesis and Main Contributions ......................... 4

2 Probabilistic Deep Learning and its Applications 9
   2.1 Neural Networks and Deep Learning ................................. 9
   2.2 Probabilistic Modeling and Inference ................................ 10
      2.2.1 Probabilistic Modeling .................................... 11
      2.2.2 Maximum Likelihood Inference ............................. 13
      2.2.3 Maximum A-Posteriori Inference ........................... 14
      2.2.4 Bayesian Inference ...................................... 15
   2.3 Approximate Bayesian Inference ................................ 16
      2.3.1 Sampling-Based Approximate Inference ...................... 16
      2.3.2 Optimization-Based Approximate Inference ................. 18
      2.3.3 The Laplace Approximation ................................ 19
   2.4 Bayesian Neural Networks ..................................... 20
      2.4.1 Overview of Bayesian Deep Learning Methods ............... 21
      2.4.2 Scalable Laplace Approximations ............................ 22
   2.5 Variational Autoencoders ..................................... 36
      2.5.1 Bayesian Inference Over Global vs. Local Latent Variables 36
      2.5.2 Variational Inference in Latent Variable Models ............ 37
      2.5.3 VAEs as Deep Latent Variable Models ...................... 40
   2.6 Applications of Probabilistic Deep Learning .................... 46
      2.6.1 Out-of-Distribution Detection ............................... 46
      2.6.2 Data-Efficient Optimization ................................ 48
      2.6.3 Neural Network Calibration .................................. 50
      2.6.4 Continual Deep Learning .................................... 51
   2.7 Discussion ...................................................... 52
3 Out-of-Distribution Detection via Bayesian Variational Autoencoders

3.1 Intuition on Information-Theoretic OoD Detection

3.2 The Bayesian Variational Autoencoder (BVAE)

3.2.1 Model Formulation

3.2.2 Inference Over the Latent Variables $\mathbf{z}$

3.2.3 Inference Over the Model Parameters $\mathbf{\theta}$

3.2.4 Inference Over the Variational Parameters $\mathbf{\phi}$

3.2.5 Likelihood Estimation

3.2.6 Pseudocode of BVAE Training Procedure

3.2.7 Related Work on Fully Bayesian Deep Generative Models

3.3 Information-Theoretic Out-of-Distribution Detection

3.3.1 Out-of-Distribution Detection in Input Space

3.3.2 Out-of-Distribution Detection in Latent Space

3.3.3 Numerically Stable Implementation of the ESS Score

3.4 Related Work

3.4.1 Supervised/Discriminative OoD Detection Methods

3.4.2 Unsupervised/Generative OoD Detection Methods

3.5 Empirical Evaluation

3.5.1 Out-of-Distribution Detection in Input Space

3.5.2 Out-of-Distribution Detection in Latent Space

3.6 Discussion

4 Data-Efficient Optimization via Weighted Retraining of Variational Autoencoders

4.1 Failure Modes of Latent Space Optimization

4.2 Latent Space Optimization With Weighted Retraining

4.2.1 Training a Generative Model With a Weighted Training Objective

4.2.2 Periodic Retraining to Update the Latent Space

4.2.3 Weighted Retraining Combined

4.3 Related Work

4.4 Empirical Evaluation

4.4.1 Effect of Weighted Training

4.4.2 Effect of Weighted Retraining Parameters on Optimization

4.4.3 Comparison with Other Methods

4.5 Discussion
# Table of Contents

5 Neural Network Calibration via Subnetwork Laplace Approximations 95
  5.1 Subnetwork Posterior Approximation ........................................... 96
  5.2 Linearized Laplace Subnetwork Inference ......................................... 97
  5.3 Subnetwork Selection ................................................................. 99
  5.4 Related Work ................................................................................. 101
  5.5 Empirical Evaluation ...................................................................... 102
    5.5.1 How Does Subnetwork Inference Preserve Predictive Uncertainty? . 103
    5.5.2 Subnetwork Inference in Large Models vs Full Inference in Small Models ................................................................. 104
    5.5.3 Image Classification Under Distribution Shift ............................. 105
  5.6 Discussion ...................................................................................... 108

6 Continual Deep Learning via Gradient Reconstruction of the Past 111
  6.1 Continual Learning Methods ............................................................ 112
  6.2 A Principle of Adaptation: Gradient Reconstruction of the Past ......... 115
  6.3 A New Improved K-Prior .................................................................. 116
    6.3.1 The Error in the K-Prior When Using a Limited Memory ............. 117
    6.3.2 Reducing $e_{\text{mem}}^t$ Using an EWC-Style Regularizer ............. 117
    6.3.3 Reducing $e_{\text{NN}}^t$ Using Experience Replay .......................... 118
    6.3.4 K-Prior With EWC-Style Regularizer and Experience Replay ....... 119
  6.4 Related Work ................................................................................... 120
  6.5 Empirical Evaluation ...................................................................... 121
    6.5.1 Experimental Setup .................................................................... 122
    6.5.2 Results on Split-CIFAR .............................................................. 123
    6.5.3 Results on Split-TinyImageNet .................................................. 124
    6.5.4 Results on ImageNet-1000 ........................................................ 125
  6.6 Discussion ...................................................................................... 126

7 Conclusions and Future Directions 127
  7.1 Summary of Contributions ............................................................... 127
    7.1.1 Contributions to Out-of-Distribution Detection .......................... 127
    7.1.2 Contributions to Data-Efficient Optimization ............................. 128
    7.1.3 Contributions to Neural Network Calibration .............................. 128
    7.1.4 Contributions to Continual Deep Learning ................................. 129
  7.2 Future Research Directions .............................................................. 129
    7.2.1 Future Research on Out-of-Distribution Detection ...................... 129
    7.2.2 Future Research on Data-Efficient Optimization ......................... 130
7.2.3 Future Research on Neural Network Calibration ............... 131
7.2.4 Future Research on Continual Deep Learning ............... 131

References ........................................................................ 133

Appendix A  The Laplace Approximation .......................... 153
   A.1 Derivation of the Laplace Approximation ...................... 153

Appendix B  Out-of-Distribution Detection ...................... 155
   B.1 Precision-Recall and ROC Curves for FashionMNIST (Held-Out Classes) . 155

Appendix C  Data-Efficient Optimization ...................... 159
   C.1 Details on the Weighting Function ............................. 159
      C.1.1 More Information on Rank-Based Weighting .............. 159
      C.1.2 Mini-Batching for Weighted Training ..................... 162
      C.1.3 Implementation of Weighted Training ..................... 163
      C.1.4 Implementation of Rank Weighting ...................... 163
      C.1.5 Rank-Weighted Distributions of Objective Function Values of 2D
            Shape and Arithmetic Expression Datasets .................. 164
   C.2 Further Experimental Results .................................. 164
      C.2.1 Optimization Performance With More Weighted Retraining Parameters 164
      C.2.2 Top10 and Top50 Optimization Results ..................... 165
      C.2.3 Comparison of Chemical Design Results With Previous Papers .......... 167
      C.2.4 Pictures of the Best Molecules Found by Weighted Retraining .......... 167
   C.3 Details on Experimental Setup .................................. 168
      C.3.1 Retraining Parameters ...................................... 168
      C.3.2 Bayesian Optimization ...................................... 168
      C.3.3 Evaluation Metrics .......................................... 169
      C.3.4 2D Shape Task Details ...................................... 169
      C.3.5 Arithmetic Expression Fitting Task ......................... 169
      C.3.6 Chemical Design Task ...................................... 170
      C.3.7 Other Reproducibility Details ............................. 171

Appendix D  Neural Network Calibration ..................... 175
   D.1 Derivations of the Wasserstein Pruning Objective .............. 175
      D.1.1 Derivation of the Initial Wasserstein Pruning Objective .......... 175
      D.1.2 Derivation of the Simplified Wasserstein Pruning Objective .......... 176
   D.2 Updating the Prior Precision for Uncertainty Estimation With Subnetworks 176
D.3  Additional Image Classification Results .......................... 177
  D.3.1  Comparing the Parameter Efficiency of Subnetwork Linearized Laplace
         With Deep Ensembles ........................................ 177
  D.3.2  Scalability of Subnetwork Linearised Laplace in the Number of
         Weights ....................................................... 177
  D.3.3  Out-of-Distribution Rejection ................................ 179
  D.3.4  Additional Rotation and Corruption Results ................ 181
D.4  Experimental Setup ................................................. 186
  D.4.1  Toy Experiments .............................................. 186
  D.4.2  UCI Experiments .............................................. 186
  D.4.3  Image Experiments ............................................ 187
  D.4.4  Datasets ...................................................... 189

Appendix E  Continual Deep Learning ............................... 191
  E.1  Derivation of Error for K-priors With Limited Memory Eq. (6.9) ... 191
  E.2  Experiment Details ............................................... 192
Chapter 1

Introduction

Over the last decade, deep neural networks (DNNs) have become the de-facto standard model for tackling complex real-world problems, achieving incredible successes across various application domains including vision, speech, and language (LeCun et al., 2015). One of the main advantages of DNNs and thus reason for their success is their ability to effectively extract hierarchical, semantically meaningful representations from raw, high-dimensional data inputs such as images, audio, and text (Goodfellow et al., 2016). These representations, also referred to as embeddings or features, can then for example be used by standard machine learning techniques such as logistic regression to solve relevant downstream applications.

However, despite these tremendous successes, DNNs still suffer from critical shortcomings, and there are still applications in which they are not the method of choice. One major limitation of modern DNNs is that they are unable to identify when new a data example is coming from a data distribution different to that encountered during training (i.e., a so called out-of-distribution input). Even worse, when being exposed to such novel data points, DNNs tend to be poorly calibrated and overconfident in their predictions (Guo et al., 2017; Nguyen et al., 2015). This is particularly concerning for safety-critical applications such as healthcare or autonomous driving (Amodei et al., 2016). Another main drawback of DNNs is that they are unable to continuously learn across sequences of tasks, but catastrophically forget past information when encountering new data (Sutton, 1986). This makes model training and deployment costly, both in the financial and environmental sense, as resources are wasted by repeatedly re-training DNNs from scratch whenever the dataset is updated (García Martín, 2017). Finally, there exist problems which could benefit from the unique properties of DNNs, but where the application of DNNs is not yet much explored due to some of their limitations, including the ones mentioned earlier. One major example are optimization problems over high-dimensional, structured inputs (e.g., occurring in science...
and engineering), where DNNs could be used to map the inputs into an easier-to-optimize embedding space (Gómez-Bombarelli et al., 2018).

One promising approach to address all the aforementioned issues in a principled way is given by the framework of probabilistic modeling, where the goal is to infer probability distributions over unknown quantities of interest (Ghahramani, 2015). As an example, one could consider a probabilistic treatment of the model parameters of a DNN. This is in stark contrast to how DNNs are typically trained, namely by obtaining only a single setting of the model parameters, which is much less expressive. Probabilistic methods have been successfully applied to address the problems mentioned above, namely out-of-distribution detection, model calibration, continual learning, and sample-efficient optimization (Bishop, 2006). However, the application of probabilistic inference techniques has mostly been limited to simple machine learning models trained on low-dimensional data, thus limiting its practical potential.

In this thesis, we develop new methods that unify the complimentary benefits of deep learning and probabilistic modeling. In particular, while deep learning enables us to extract representations of high-dimensional inputs such as images, probabilistic modeling allows us to flexibly reason about distributions over model parameters. We demonstrate how such probabilistic deep learning methods help us address the aforementioned applications.

Firstly, we focus on out-of-distribution detection. We propose a variational autoencoder (Kingma and Welling, 2014) where a full posterior distribution (instead of just a point estimate) is inferred over the model parameters using stochastic gradient Markov chain Monte Carlo (Chen et al., 2014). We demonstrate how information-theoretic measures (MacKay, 1992c) can then detect out-of-distribution points both in input space and in the model’s latent space.

Secondly, we turn to data-efficient optimization. We build upon methods that do Bayesian optimization (Brochu et al., 2010) in the low-dimensional, continuous latent manifold of a variational autoencoder (Gómez-Bombarelli et al., 2018). In contrast to prior work, we periodically retrain the model on the newly-queried data points, weighted by their objective function value. This actively steers the model to maintain a manifold that is useful for the optimization process.

Thirdly, we consider neural network calibration. We develop a Bayesian neural network (Gal, 2016) method where inference is performed only over a small, carefully-selected subset of the neural network weights, while the other weights are kept as point estimates. This enables us to use expressive, otherwise intractable, posterior approximations over such subnetworks, such as full-covariance Gaussian posteriors via the Laplace approximation (MacKay, 1992a).
Lastly, we address *continual deep learning*. We present a new method that mitigates catastrophic forgetting in neural networks by combining the complementary benefits of experience replay (*Robins, 1995*) and quadratic Bayesian weight-regularization (*Kirkpatrick et al., 2017*). To this end, we leverage Knowledge-adaptation priors (*Khan and Swaroop, 2021*), a general principle of adaptation that relies on a faithful reconstruction of the gradients of the past data.

1.1 List of Publications

The research in this thesis appears in several publications I co-authored throughout the course of my PhD studies, listed below. The work on *out-of-distribution detection* via Bayesian variational autoencoders was published in *Daxberger and Hernández-Lobato (2019)*; the work on *data-efficient optimization* via weighted retraining of variational autoencoders was published in *Tripp et al. (2020)*; the work on *neural network calibration* via scalable Laplace approximations was published in *Daxberger et al. (2021c)*; the work on *continual deep learning* via gradient reconstruction of the past was published in *Daxberger et al. (2023)*; and the review of *scalable Laplace approximations for deep learning* was published in *Daxberger et al. (2021b)*, with the corresponding *laplace* library being open-sourced in *Daxberger et al. (2021a)*. Finally, the list below includes two publications that I contributed to during my PhD studies, even though the research does not appear in this thesis, namely *Antorán et al. (2022)*; *Eschenhagen et al. (2021)*. Asterisks (*) denote that the respective authors contributed equally.

**Peer-Reviewed Conference Proceedings**


**Peer-Reviewed Workshop Proceedings**


**Open Source Software**


1.2 Overview of Thesis and Main Contributions

We now provide an overview of this thesis, highlighting our main contributions in each chapter.

In Chapter 2, we review the necessary background material that our contributions build upon. We first introduce the basics of neural networks and deep learning (Section 2.1). We then cover the fundamentals of probabilistic modeling and inference (Section 2.2), going from maximum likelihood inference over maximum a-posterior inference to Bayesian inference. We then describe three major approaches to approximate Bayesian inference (Section 2.3): sampling-based approximate inference, optimization-based approximate inference, and the Laplace approximation. Based on that, we introduce the two main probabilistic deep learning
models considered in this thesis: Bayesian neural networks (Section 2.4) and variational autoencoders (Section 2.5). Finally, we motivate and describe the different applications of probabilistic deep learning that we consider in this thesis: out-of-distribution detection, data-efficient optimization, neural network calibration, and continual deep learning (Section 2.6).

Notably, in Section 2.4.2, we provide an in-depth coverage of scalable Laplace approximations for Bayesian neural networks. The Laplace approximation (LA) is a classic, and arguably the simplest family of approximations for the intractable posteriors of deep neural networks. Yet, despite its simplicity, the LA is not as popular as other Bayesian deep learning methods. This may be due to assumptions that the LA is expensive due to the involved Hessian computation, or that it is difficult to implement. We show that these are misconceptions by (i) reviewing the range of variants of the LA including versions with minimal cost overhead, and (ii) introducing laplace, an easy-to-use software library for PyTorch offering user-friendly access to all major flavors of the LA. The work in this section was developed with my co-authors Agustinus Kristiadi, Runa Eschenhagen, and Alexander Immer, with Matthias Bauer and Philipp Hennig supervising throughout. The material in this section is published in Daxberger et al. (2021b), and the corresponding laplace PyTorch library is open-sourced in Daxberger et al. (2021a).

In Chapter 3, we develop a new probabilistic deep learning approach for out-of-distribution detection. Despite their successes, deep neural networks may make unreliable predictions when faced with test data drawn from a distribution different to that of the training data, constituting a major problem for AI safety. While this has recently motivated the development of methods to detect such out-of-distribution (OoD) inputs, a robust solution is still lacking. We propose a new probabilistic, unsupervised approach to this problem based on a Bayesian variational autoencoder model, which estimates a full posterior distribution over the decoder parameters using stochastic gradient Markov chain Monte Carlo, instead of fitting a point estimate. We describe how information-theoretic measures based on this posterior can then be used to detect OoD inputs both in input space and in the model’s latent space. We empirically demonstrate the effectiveness of our proposed approach. The research in this chapter was supervised by José Miguel Hernández-Lobato and is published in Daxberger and Hernández-Lobato (2019).

In Chapter 4, we develop a new probabilistic deep learning approach for data-efficient optimization. Many important problems in science and engineering, such as drug design, involve optimizing an expensive black-box objective function over a complex, high-dimensional, and structured input space. Although machine learning techniques have shown promise in solving such problems, existing approaches substantially lack sample efficiency. We introduce an improved method for efficient black-box optimization, which performs the optimization in the
Introduction

low-dimensional, continuous latent manifold learned by a deep generative model. In contrast to previous methods, we actively steer the model to maintain a latent manifold that is highly useful for efficiently optimizing the objective. We achieve this by periodically retraining the model on the data points queried along the optimization trajectory, as well as weighting those points according to their objective function value. This weighted retraining can be easily implemented on top of existing methods, and is empirically shown to significantly improve their efficiency and performance on synthetic and real-world optimization problems. The work in this chapter was developed with my co-author Austin Tripp, with José Miguel Hernández-Lobato supervising throughout. The material in this chapter is published in Tripp et al. (2020).

In Chapter 5, we develop a new probabilistic deep learning approach for neural network calibration. Despite their successes, modern neural networks still suffer from poor calibration and overconfidence, especially when the data distribution shifts between training and testing. The Bayesian paradigm has the potential to tackle this issue by equipping neural networks with robust uncertainty estimates. Alas, scaling Bayesian inference to large parameter spaces often requires restrictive approximations. We show that it suffices to perform inference over a small subset of model parameters to obtain accurate predictive posteriors. The other parameters are kept as point estimates. This subnetwork inference framework enables us to use expressive, otherwise intractable, posterior approximations over such subsets. In particular, we implement subnetwork linearized Laplace as a simple, scalable Bayesian deep learning method: We first obtain a MAP estimate of all parameters and then infer a full-covariance Gaussian posterior over a subnetwork using the linearized Laplace approximation. We propose a subnetwork selection strategy that aims to maximally preserve the model’s predictive uncertainty. Empirically, our approach compares favorably to ensembles and less expressive posterior approximations over full neural networks. The work in this chapter was developed with my co-authors Eric Nalisnick, James Urquhart Allingham, and Javier Antorán, with José Miguel Hernández-Lobato supervising throughout. The material in this chapter is published in Daxberger et al. (2021c).

In Chapter 6, we develop a new probabilistic deep learning approach for continual deep learning. In addition to poor calibration, another critical issue of modern neural networks is that they tend to catastrophically forget previously learned tasks when sequentially trained on new tasks. It is a major challenge to enable neural networks to learn continually in an online fashion. Regularization and experience replay are two popular continual-learning strategies with complementary strengths: while regularization requires less memory, replay can more accurately mimic batch training. But can we combine them to get better methods? Despite the simplicity of the question, little is known or done to optimally combine these approaches.
We present such a method by using a recently proposed principle of adaptation that relies on a faithful reconstruction of the gradients of the past data. Using this principle, we design a prior which combines two types of replay methods with a quadratic weight-regularizer and achieves better gradient reconstructions. The combination improves performance on standard benchmarks such as Split-CIFAR, SplitTinyImageNet, and ImageNet-1000, achieving >80% of the batch performance by simply utilizing a memory of <10% of the past data. Our work shows that a good combination of replay and regularizer-based methods can be very effective in reducing forgetting, and can sometimes even completely eliminate it. The work in this chapter was developed with my co-authors Siddharth Swaroop, Kazuki Osawa, Rio Yokota, Richard E. Turner, and José Miguel Hernández-Lobato, with Mohammad Emtiyaz Khan supervising throughout. The material in this chapter is published in Daxberger et al. (2023).

Finally, in Chapter 7, we conclude by briefly summarizing the contributions made in this thesis, and then outlining interesting research directions we are keen on pursuing in the future.
Chapter 2

Probabilistic Deep Learning and its Applications

This chapter provides a review of the background on probabilistic deep learning, serving as the basis for our work. We begin by introducing neural networks and deep learning as the main approach for function approximation used in this thesis (Section 2.1). We then describe the framework of probabilistic modeling, which we use for formalizing inference in our models (Section 2.2). We proceed to survey several approximate Bayesian inference approaches that allow us to perform probabilistic inference in cases where exact inference is intractable (Section 2.3). We then continue with reviewing the two main probabilistic deep learning models considered in this work: Bayesian neural networks, in which probabilistic inference is performed over the parameters of a discriminative neural network model (Section 2.4), and variational autoencoders, in which probabilistic inference is performed over the latent variables of a generative neural network model (Section 2.5). Finally, we introduce the applications of probabilistic deep learning considered in this thesis: out-of-distribution detection, data-efficient optimization, neural network calibration, and continual deep learning (Section 2.6).

2.1 Neural Networks and Deep Learning

We denote our $L$-layer neural network (NN) function as $f_\theta : \mathcal{X} \rightarrow \mathcal{Y}$, mapping $I$-dimensional input variables $x \in \mathcal{X} \subseteq \mathbb{R}^I$ to $O$-dimensional (observed) output variables $y \in \mathcal{Y} \subseteq \mathbb{R}^O$. The $D$-dimensional parameter vector of the neural network is denoted by $\theta \in \mathbb{R}^D$; this vector is obtained by concatenating and flattening all learnable parameters of all $L$ layers of the NN.
(i.e., $L-1$ hidden layers and a final output layer). We use the terms "deep learning" and "deep neural network" when the NN has many hidden layers (Goodfellow et al., 2016).

The NN function $f_{\theta}$ is generally defined as the composition of its $L$ layers, where each layer is in turn defined by a simpler function followed by an element-wise non-linearity or activation function $\varphi$, e.g., the rectified linear unit (ReLU), $\varphi(a) = \max(0, a)$. There exist different families of NN functions $f_{\theta}$ with different choices of layers and non-linearities; the most popular so-called NN architectures include multi-layer perceptrons (MLPs), convolutional neural networks (CNNs), recurrent neural networks (RNNs) and Transformers. For the sake exposition, we here only describe MLPs, since they correspond to the simplest NN architecture, and are also common building blocks of other architectures; for details on other NN architectures (and deep learning more generally), refer to Goodfellow et al. (2016). In an MLP, each layer $l$ is affine, parameterized by a weight matrix $W^{(l)}$ and bias vector $b^{(l)}$, i.e.,

$$f_{\theta}(x) := W^{(L)}h^{(L)}(x) + b^{(L)},$$

where

$$h^{(l+1)}(x) := \varphi \left( W^{(l)}h^{(l)}(x) + b^{(l)} \right) \quad \text{for } 0 \leq l \leq L - 1,$$

and

$$h^{(0)}(x) := x.$$  

$x$ is often a high-dimensional sensory input (e.g., an image, a sentence, etc.), and $f_{\theta}$ is often viewed to extract (or encode) an embedding (or feature vector) $h^{(L)}(x)$ from $x$ that lives on some lower-dimensional embedding manifold.

In the next section, we will describe the probabilistic interpretation of the functional relationship between inputs $x$ and outputs $y$ as defined by the parametric function $f_{\theta}$. Viewing $f_{\theta}$ as a probabilistic model will provide us with different ways to learn its model parameters $\theta$ via probabilistic inference.

### 2.2 Probabilistic Modeling and Inference

In this section, we first introduce the framework of probabilistic modeling that underpins most of this thesis (Section 2.2.1). We then describe three main approaches to inference in probabilistic models. We start with maximum likelihood (Section 2.2.2) and maximum a-posteriori inference (Section 2.2.3), in which we obtain point estimates of the model parameters. Finally, we introduce Bayesian inference, which allows us to estimate entire probability distributions over the model parameters, and which will be the main focus of this thesis (Section 2.2.4). Parts of the material in this section follow the excellent exposition in Bishop (2006).
2.2.1 Probabilistic Modeling

Consider a discriminative, supervised learning task where given a dataset $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^{N}$ of $N$ input-output pairs $(x_n, y_n)$, we aim to recover the underlying functional relationship between inputs $x_n$ and outputs $y_n$ via a neural network model $f_\theta$ parameterized by $\theta$.

To address this task, this thesis predominantly leverages the probabilistic perspective of machine learning, which is fundamentally based on probabilistic modeling. This means that the data generating process is captured by a probabilistic model, and that we work with probability distributions over observed and latent quantities. The learning task is then formalized as probabilistic inference: a procedure for deducing (or estimating) unobserved (or unknown) properties, i.e., the model parameters $\theta$, given observed (or known) information, i.e., the dataset $\mathcal{D}$.

Formally, we make the assumption that the dataset $\mathcal{D}$ has been generated by first sampling inputs $x_n$ from some underlying data distribution $p(x)$, i.e., $x_n \sim p(x), 1 \leq n \leq N$, and then sampling the corresponding outputs $y_n$ from some conditional distribution $p(y|x)$, i.e., $y_n \sim p(y|x_n), 1 \leq n \leq N$. Put together, we assume that the data points are independently and identically distributed as

$$x_n, y_n \overset{i.i.d.}{\sim} p(x, y) = p(x)p(y|x), \quad 1 \leq n \leq N. \quad (2.4)$$

Unless otherwise stated, we make this assumption throughout the remainder of this thesis.

Our goal is to use $\mathcal{D}$ to learn the parameters of a model $p(y|x, \theta)$ of the conditional distribution $p(y|x)$, which then allows us to make predictions about the output $y_s$ corresponding to a previously unseen input $x_s \sim p(x)$. Formally, we aim to compute the (posterior) predictive distribution over output $y_s$ given input $x_s$ and data $\mathcal{D}$,

$$p(y_s|x_s, \mathcal{D}) = \int_{\theta} p(y_s|x_s, \mathcal{D}) d\theta = \int_{\theta} p(y_s|x_s, \theta)p(\theta|\mathcal{D}) d\theta. \quad (2.5)$$

We can see that Eq. (2.5) involves two quantities of interest:

1. Firstly, the conditional distribution $p(y|x, \theta)$ over outputs $y$ given inputs $x$ and model parameters $\theta$, which serves as our model of the conditional distribution $p(y|x)$. The function $p(y|x, \theta)$ is often referred to as modeling the likelihood of the model parameters $\theta$ under the data $(x, y)$. Following Eq. (2.4), we assume that the data points in $\mathcal{D}$ are independently and identically distributed according to the likelihood, such that the likelihood of $\theta$ under the entire dataset $\mathcal{D}$ can be written as

$$p(\mathcal{D}|\theta) = \prod_{n=1}^{N} p(y_n|x_n, \theta), \quad y_n \overset{i.i.d.}{\sim} p(y_n|x_n, \theta), \quad 1 \leq n \leq N. \quad (2.6)$$
The definition of the likelihood depends on the particular type of output set \( \mathcal{Y} \). There are two main output types which induce different target problems and thus likelihoods:

**Classification.** \( \mathcal{Y} \) is a set of \( C \) class labels (e.g., \( \mathcal{Y} = \{1, 2, \ldots, C\} \)), and the outputs are typically generated as

\[
y = \sigma(f_\theta(x)), \quad (2.7)
\]

where

\[
\sigma(a)_i = \frac{e^{a_i}}{\sum_{j=1}^{C} e^{a_j}}, \quad 1 \leq i \leq C;
\]

is the softmax function (for binary classification, i.e., \( C = 2 \), \( \sigma \) reduces to the sigmoid function). This transforms the model outputs \( f_\theta(x) \) into a probability vector, yielding a categorical likelihood,

\[
p(y|x, \theta) = p(y|f_\theta(x)) = \text{Cat}(y; \sigma(f_\theta(x))). \quad (2.9)
\]

**Regression.** \( \mathcal{Y} \) is a set of continuous values (e.g., \( \mathcal{Y} = \mathbb{R} \)), and the outputs are, for example, generated as

\[
y = f_\theta(x) + \epsilon, \quad (2.10)
\]

where

\[
\epsilon \sim \mathcal{N}(0, \sigma_n^2) \quad (2.11)
\]

is a random noise variable following a zero-mean Gaussian distribution with noise variance \( \sigma_n^2 \). This generative process thus induces a Gaussian likelihood,

\[
p(y|x, \theta) = p(y|f_\theta(x)) = \mathcal{N}(y; f_\theta(x), \sigma_n^2). \quad (2.12)
\]

2. Secondly, the posterior distribution \( p(\theta | D) \) over model parameters \( \theta \) given data \( D \), which models our *a-posteriori belief* about \( \theta \) after having observed \( D \). We thus treat \( \theta \) as a random variable instead of a fixed vector. This posterior can be computed using Bayes’ rule as

\[
p(\theta | D) = \frac{p(D | \theta)p(\theta)}{p(D)} \quad (2.13)
\]

The posterior thus depends on three quantities. First, \( p(D | \theta) \) is the likelihood from Eq. (2.6). Second, \( p(\theta) \) is a *prior distribution* over \( \theta \), which formalizes our *a-priori* belief about what values the model parameters \( \theta \) might take before having observed any data. For example, we could encode the *a-priori* assumption that \( \theta \) has small \( \ell_2 \) norm by placing a zero-mean Gaussian prior over \( \theta \). Third, the normalizing constant
of the posterior,

$$p(D) = \int_\theta p(D, \theta) d\theta = \int_\theta p(D | \theta)p(\theta) d\theta, \quad (2.14)$$

is the marginal likelihood (also referred to as evidence), which is typically intractable.

Thus, to compute the predictive distribution in Eq. (2.5), we first need to specify a likelihood \( p(y|x, \theta) \) as well as a prior \( p(\theta) \), observe a dataset \( D \), and then infer a posterior \( p(\theta | D) \) over the unknown model parameters \( \theta \). We now describe three inference paradigms, which make different assumptions about the complexity of the posterior \( p(\theta | D) \): 1) maximum likelihood inference, 2) maximum a-posteriori inference, and 3) Bayesian inference.

### 2.2.2 Maximum Likelihood Inference

One common way to infer the parameters \( \theta \) is via maximum likelihood (ML) inference, which simply maximizes the likelihood \( p(D | \theta) \) of the model \( \theta \) given the observations \( D \) (i.e., ignoring the a-priori assumptions about \( \theta \)),

$$\theta_{\text{ML}} = \arg \max_\theta p(D | \theta) \quad (2.15)$$

$$= \arg \max_\theta \prod_{n=1}^N p(y_n | x_n, \theta). \quad (2.16)$$

Since dealing with products is often tedious, and might furthermore produce numerical problems (e.g., as taking a product of scalars smaller than 1 may rapidly approach 0 when the number of scalars is large, resulting in numerical underflow), it is more convenient to instead work in the log-space. As the logarithm function is monotonic, maximizing a function is equivalent to maximizing the log of that function, thus resulting in the maximum log-likelihood estimate

$$\theta_{\text{ML}} = \arg \max_\theta \log p(D | \theta) \quad (2.17)$$

$$= \arg \max_\theta \log \prod_{n=1}^N p(y_n | x_n, \theta) \quad (2.18)$$

$$= \arg \max_\theta \sum_{n=1}^N \log p(y_n | x_n, \theta). \quad (2.19)$$
We thus only obtain a single point estimate of $\theta$, i.e., we implicitly assume that the posterior is defined by a Dirac delta function centered around the ML estimate $\theta_{\text{ML}}$, $p(\theta | D) = \delta_{\theta_{\text{ML}}} (\theta)$. The predictive distribution in Eq. (2.5) is thus tractable and reduces to $p(y_* | x_*, \theta_{\text{ML}})$.

### 2.2.3 Maximum A-Posteriori Inference

If we want to take into account our a-priori belief as encoded by the prior $p(\theta)$, we instead use **maximum a-posteriori (MAP) inference**$^1$, which maximizes the posterior distribution $p(\theta | D)$ over $\theta$.

$$\theta_{\text{MAP}} = \arg \max_{\theta} p(\theta | D)$$  \hspace{1cm} (2.20)

$$= \arg \max_{\theta} \log p(\theta | D)$$  \hspace{1cm} (2.21)

$$= \arg \max_{\theta} \log p(D | \theta) p(\theta)$$  \hspace{1cm} (2.22)

$$= \arg \max_{\theta} \log \prod_{n=1}^{N} p(y_n | x_n, \theta) p(\theta)$$  \hspace{1cm} (2.23)

$$= \arg \max_{\theta} \left[ \sum_{n=1}^{N} \log p(y_n | x_n, \theta) \right] + \log p(\theta) .$$  \hspace{1cm} (2.24)

Again, here we only obtain a single setting of $\theta$, resulting in the predictive distribution $p(y_* | x_*, \theta_{\text{MAP}})$ evaluated at the MAP estimate $\theta_{\text{MAP}}$.

We wish to clarify some terminology: in (supervised) machine learning / deep learning, the ML and MAP objectives in Eqs. (2.19) and (2.24) are typically formulated as **(regularized) empirical risk minimization**. That is, instead of the likelihood-/posterior-maximization problems in Eqs. (2.19) and (2.24), the objectives can be equivalently defined to minimize the (regularized) empirical risk $\mathcal{L}(D; \theta)$, which decomposes into a sum over empirical loss terms $\ell(x_n, y_n; \theta)$ and – in the case of MAP inference – a regularizer $r(\theta)$,

$$\theta_{\text{MAP}} = \arg \min_{\theta} \mathcal{L}(D; \theta)$$  \hspace{1cm} (2.25)

$$= \arg \min_{\theta} \left[ \sum_{n=1}^{N} \ell(x_n, y_n; \theta) + r(\theta) \right] .$$  \hspace{1cm} (2.26)

---

$^1$In fact, ML inference can be interpreted as implicitly placing a uniform or flat prior $p(\theta)$ over $\theta$, i.e., assuming that every possible value of $\theta$ is equally likely.
We thus have the following mapping between the quantities: the (regularized) empirical risk corresponds to the unnormalized negative log-posterior, i.e.,

$$L(D; \theta) = -\log p(D \mid \theta)p(\theta),$$  \hspace{1cm} (2.27)

the empirical loss terms correspond to the negative log-likelihoods, i.e.,

$$\ell(x_n, y_n; \theta) = -\log p(y_n \mid x_n, \theta),$$  \hspace{1cm} (2.28)

and the regularization term corresponds to the negative log-prior, i.e.,

$$r(\theta) = -\log p(\theta).$$  \hspace{1cm} (2.29)

Thus, imposing a prior over \( \theta \) has the effect of regularization (i.e., it can help the model to avoid overfitting to the training data \( D \)); this effect is generally stronger the fewer data is available. For example, the widely used \( \ell_2 \) weight regularizer \( r(\theta) = \frac{1}{2} \gamma^{-2} \| \theta \|^2 \) (a.k.a. weight decay) corresponds to a zero-mean Gaussian prior with variance \( \gamma^2 \), i.e., \( p(\theta) = \mathcal{N}(\theta; 0, \gamma^2 I) \).

While \( \theta_{ML} \) and \( \theta_{MAP} \) can be computed analytically for simple models (e.g., linear models), it needs to be estimated numerically in general. For example, for deep neural network models, \( \theta_{ML} / \theta_{MAP} \) are typically computed/optimized using a variant of gradient descent, using the backpropagation algorithm to efficiently compute the required loss gradients w.r.t. \( \theta \),

$$\nabla_\theta L(D; \theta) = \nabla_\theta - \log p(D \mid \theta)p(\theta).$$  \hspace{1cm} (2.30)

### 2.2.4 Bayesian Inference

Given a finite dataset \( D \), it is generally not possible to confidently determine the “best” set of parameters, as there might be many settings of \( \theta \) that explain the data equally well (especially for high-capacity models such as NNs). As a result, the specific parameter settings \( \theta_{ML} \) and \( \theta_{MAP} \) which are found to be “optimal” under the ML and MAP inference frameworks, respectively, might not sufficiently well capture our knowledge about \( \theta \) as provided by \( D \).

Thus, in stark contrast to ML and MAP inference, which only obtain a single point estimate of \( \theta \), the idea of Bayesian inference is to take into account a whole range of different solutions for the parameters \( \theta \) (along with their plausibility), instead of settling for one specific configuration. Performing full Bayesian inference involves computing the entire posterior distribution \( p(\theta \mid D) \) over \( \theta \) given \( D \). To compute the predictive distribution, we then take into account all possible settings of \( \theta \) by averaging over this posterior. More
generally, we are often interested in computing integrals of some function $F(\theta)$ over the posterior,

$$\int_{\theta} F(\theta)p(\theta|D)d\theta.$$  \hspace{1cm} (2.31)

Unfortunately, due to the involved integrals, both computing the posterior $p(\theta|D)$ via Eq. (2.13) and quantities of the form of Eq. (2.31), such as the predictive distribution in Eq. (2.5), are generally intractable for most interesting problems (in particular those involving neural networks). This motivates the use of approximate Bayesian inference methods, to be described next.

### 2.3 Approximate Bayesian Inference

In this section, we survey three main approaches to approximate Bayesian inference. We first briefly describe the main idea behind sampling-based approximate inference and introduce a state-of-the-art method called stochastic gradient Hamiltonian Monte Carlo (Section 2.3.1). We then elaborate on optimization-based approximate inference, leading up to the popular variational inference framework (Section 2.3.2). Finally, we survey the Laplace approximation, which is a classic approximate inference method that has recently re-gained in popularity (Section 2.3.3). Parts of the material on sampling- and optimization-based approximate inference follow the excellent exposition in Li (2018a).

#### 2.3.1 Sampling-Based Approximate Inference

Rather than trying to find tractable forms of the integral in Eq. (2.31), one can often use numerical integration techniques based on discretisation and Monte Carlo. A Monte Carlo approach would make use of the property that the integral is computed against a probability distribution, resulting in an estimate that first samples $\theta_k \sim p(\theta_k|D)$ from the posterior and then approximates the integral as

$$\int_{\theta} F(\theta)p(\theta|D)d\theta \simeq \frac{1}{K} \sum_{k=1}^{K} F(\theta_k).$$  \hspace{1cm} (2.32)

Unfortunately, this simple Monte Carlo approach assumes that it is easy to draw samples from the posterior, which is not true in most cases. In order to (approximately) draw samples from the posterior, one might leverage advanced sampling schemes such as importance sampling, rejection sampling and, in particular, Markov chain Monte Carlo (MCMC) (Gelman et al., 2013). Unfortunately, these methods require large numbers of samples for estimating high-
2.3 Approximate Bayesian Inference

dimensional distributions (if the random variables exhibit high correlation), and the MCMC simulation time might become prohibitively long.

Hamiltonian Monte Carlo

Hamiltonian Monte Carlo (HMC) (Betancourt, 2017; Duane et al., 1987; Neal et al., 2011) is a method for generating samples \( \theta \sim p(\theta | D) \) in a Metropolis-Hastings framework to explore the state space more efficiently than traditional MCMC methods. In particular, consider the posterior distribution

\[
p(\theta | D) \propto \exp(-U(\theta, D))
\]

(2.33)

with potential energy function

\[
U(\theta, D) = -\log p(D, \theta) = -\log(p(D|\theta)p(\theta)) = -\sum_{x \in D} \log p(x|\theta) - \log p(\theta)
\]

(2.34)

induced by the prior \( p(\theta) \) and marginal log-likelihood \( \log p(x|\theta) \). HMC generates samples \( \theta \sim p(\theta | D) \) by simulating Hamiltonian dynamics, which involves evaluating the gradient \( \nabla_{\theta} U(\theta) \) of \( U \).

Stochastic Gradient HMC

The problem of HMC is that direct computation of the gradient \( \nabla_{\theta} U(\theta) \) of \( U \) in Eq. (2.34) requires examination of the entire dataset \( D \) (due to the summation of the log-likelihood over all \( x \in D \)), which might become prohibitively costly for large datasets. To tackle this issue, Chen et al. (2014) recently proposed a scalable HMC variant called stochastic gradient Hamiltonian Monte Carlo (SGHMC), which is a state-of-the-art method for generating samples \( \theta \sim p(\theta | D) \) of parameters \( \theta \) of a deep neural network. SGHMC considers a noisy, unbiased estimate of the gradient computed from a minibatch \( M \) of points sampled uniformly at random from \( D \) (i.e., akin to minibatch-based optimization algorithms such as variants of stochastic gradient descent), i.e.,

\[
\nabla_{\theta} U(\theta, D) \simeq \nabla_{\theta} U(\theta, M) = -\frac{|D|}{|M|} \sum_{x \in M} \nabla_{\theta} \log p(x|\theta) - \nabla_{\theta} \log p(\theta).
\]

(2.35)

While our focus lies on SGHMC due its strong empirical performance, there exists a wide variety of stochastic gradient MCMC (SG-MCMC) methods which extend gradient-based MCMC methods to work with stochastic gradients via data mini-batching to enable scalability to large datasets (Ahn et al., 2012; Ding et al., 2014; Garriga-Alonso and Fortuin, 2021; Ma et al., 2015; Welling and Teh, 2011; Zhang et al., 2020).
2.3.2 Optimization-Based Approximate Inference

The idea behind optimization-based approximate inference is to find another distribution \( q(\theta) \) such that the integral \( \int_\theta F(\theta)q(\theta)d\theta \) is easier to compute than and at the same time closely approximates the exact integral in Eq. (2.31). The distribution \( q \in Q \) is assumed to come from a set of approximating distributions \( Q \) (also called the approximating family) which typically have a simple form in order to allow for analytic integration. After having specified such a class of distributions \( Q \), the task then reduces to finding the distribution \( q \in Q \) such that the integral w.r.t. \( q \) optimally approximates the exact integral. Optimization-based approximate inference thus converts a hard integration problem into an easier optimization problem, where the complexity of the family \( Q \) determines the complexity of this optimization. A main challenge thus is to choose \( Q \) to be flexible enough to capture the posterior, yet simple enough for efficient optimization. Since directly minimising the integral approximation error \( err(E_q[F],E_p[F]) \) is generally intractable (as it requires access to the exact integral \( E_p[F] \)), most methods are indirect in that they find the distribution \( q \in Q \) that is closest to the true posterior \( p(\theta \mid D) \), as quantified by some distance/divergence/discrepancy measure \( D \), i.e.,

\[
q_*(\theta) = \arg\min_{q \in Q} D(q(\theta) \| p(\theta \mid D)).
\]  

(2.36)

After having obtained the optimal \( q_*(\theta) \approx p(\theta \mid D) \), we can then make predictions by approximating the predictive distribution in Eq. (2.5) as

\[
p(y_\ast \mid x_\ast, D) = \int_\theta p(y_\ast \mid x_\ast, \theta)p(\theta \mid D)d\theta \approx \int_\theta p(y_\ast \mid x_\ast, \theta)q_*(\theta)d\theta,
\]  

(2.37)

i.e., we use the optimized approximating density as a proxy for the exact posterior.

The most common choice of divergence measure \( D \) to quantify the proximity between two densities is the information-theoretic Kullback-Leibler (KL) divergence (Kullback, 1959; Kullback and Leibler, 1951), i.e., \( D(q\|p) = D_{KL}(q\|p) \) with

\[
D_{KL}(q\|p) = \int_\theta q(\theta) \log \frac{q(\theta)}{p(\theta)}d\theta.
\]  

(2.38)

The KL divergence is asymmetric and non-negative, i.e., \( D_{KL}(q\|p) \neq D_{KL}(p\|q) \) in general and \( D_{KL}(p\|q) \geq 0 \). Furthermore, it is easy to show that the KL divergence is minimized when \( q(\cdot) = p(\cdot) \), implying \( D_{KL}(p\|q) = 0 \).

However, minimizing \( D_{KL}(q(\theta)\|p(\theta \mid D)) \) directly is still intractable, as it involves computing the posterior \( p(\theta \mid D) \) and thus the intractable marginal likelihood \( p(D) \). A very popular approximate inference method that overcomes this issue is called variational...
2.3 Approximate Bayesian Inference

(a) MAP Estimation
(b) Laplace Approximation
(c) Prediction

Fig. 2.1 Probabilistic predictions with the Laplace approximation in three steps. (a) We find a MAP estimate (yellow star) via standard training (background contours = log-posterior landscape on the two-dimensional PCA subspace of the SGD trajectory (Izmailov et al., 2019)). (b) We locally approximate the posterior landscape by fitting a Gaussian centered at the MAP estimate (yellow contours), with covariance matrix equal to the negative inverse Hessian of the loss at the MAP—this is the Laplace approximation (LA). (c) We apply the LA to a regression problem to make predictions with predictive uncertainty estimates—here, the black curve is the predictive mean, and the shading covers the 95% confidence interval.

inference (VI) (Beal, 2003; Ghahramani et al., 2000; Jordan et al., 1999), which we will introduce when covering variational autoencoders in Section 2.5. For an in-depth treatment on VI, we refer to the excellent review papers by Blei et al. (2017) and Zhang et al. (2018).

2.3.3 The Laplace Approximation

In addition to the previous methods, there is the Laplace approximation, which is a classic technique for approximate Bayesian inference. First proposed in the context of neural networks by MacKay (1992a), the LA dates back to the 18th century (Laplace, 1774). Its main idea is to locally approximate the posterior with a Gaussian distribution centered at a local optimum of the model parameters, with covariance matrix corresponding to the local curvature. Figure 2.1 provides an intuition of the LA—we first fit a point estimate of the model parameters and then estimate a Gaussian distribution around that.

More precisely, the Laplace approximation uses a second-order expansion of $\mathcal{L}$ around $\theta_{\text{MAP}}$ to construct a Gaussian approximation to $p(\theta | D)$. I.e., we consider

$$\mathcal{L}(D; \theta) \approx \mathcal{L}(D; \theta_{\text{MAP}}) + \frac{1}{2}(\theta - \theta_{\text{MAP}})^T (\nabla^2_{\theta} \mathcal{L}(D; \theta)|_{\theta_{\text{MAP}}}) (\theta - \theta_{\text{MAP}}),$$

where the first-order term vanishes at $\theta_{\text{MAP}}$. Then we can identify the Laplace approximation as

$$p(\theta | D) \approx N(\theta; \theta_{\text{MAP}}, \Sigma) \quad \text{with} \quad \Sigma := (\nabla^2_{\theta} \mathcal{L}(D; \theta)|_{\theta_{\text{MAP}}})^{-1}. \quad (2.40)$$

See Chapter A for a derivation of the Laplace approximation. Thus, to obtain the approximate posterior, we first need to find the $\text{argmax} \ \theta_{\text{MAP}}$ of the log-posterior function, i.e. do
“standard” machine learning with regularized empirical risk minimization (see Fig. 2.1(a)). The only additional step is to compute the inverse of the Hessian matrix at $\theta_{\text{MAP}}$ (see Fig. 2.1(b)). The LA can therefore be constructed post-hoc to a pre-trained model, even one downloaded off-the-shelf.

Generally, any prior with twice differentiable log-density can be used. Due to the popularity of the weight decay regularizer, we assume that the prior is a zero-mean Gaussian $p(\theta) = \mathcal{N}(\theta; 0, \gamma^2 I)$ unless stated otherwise.\footnote{One can also consider a per-layer or even per-parameter weight decay, which corresponds to a more general, but still comparably simple Gaussian prior. In particular, the Hessian of this prior is still diagonal and constant.} The Hessian $\nabla^2_\theta \mathcal{L}(D; \theta)|_{\theta_{\text{MAP}}}$ then depends both on the (simple) log-prior / regularizer and the (complicated) log-likelihood / empirical risk:

$$\nabla^2_\theta \mathcal{L}(D; \theta)|_{\theta_{\text{MAP}}} = -\gamma^{-2} I - \sum_{n=1}^N \nabla^2_\theta \log p(y_n | f_\theta(x_n))|_{\theta_{\text{MAP}}}.$$ \hfill (2.41)

LAs are widely used to approximate the posterior distribution in logistic regression (Spiegelhalter and Lauritzen, 1990), Gaussian process classification (Rasmussen and Williams, 2005; Williams and Barber, 1998), and also in neural networks, both shallow (MacKay, 1992b) and deep (Ritter et al., 2018b).

The main challenge in applying the LA to neural networks is that a naive implementation of the Hessian is infeasible because the second term in Eq. (2.41) scales as $\mathcal{O}(D^2)$, i.e. quadratically with the number of neural network parameters $D$, which can be in the millions or even billions in modern deep learning (He et al., 2016; Shoeybi et al., 2019). Section 2.4.2 will outline some recent attempts in scaling the LA to large neural networks.

### 2.4 Bayesian Neural Networks

We now turn to the first probabilistic deep learning model considered in this thesis, Bayesian neural networks (BNNs). A BNN is obtained when using Bayesian inference to infer a posterior distribution $p(\theta | D)$ over the model parameters $\theta$ of a neural network. Unfortunately, due to the non-linearity of NNs, it is intractable to infer the exact posterior. Furthermore, due to the high dimensionality of $\theta$ it is even computationally challenging to faithfully approximate the posterior. Therefore, Bayesian deep learning methods typically make strong approximations to obtain an estimate of the posterior. We now first provide a brief overview of Bayesian deep learning methods, which includes approaches that aim to scale sampling-based, optimization-based, and more heuristic approximate Bayesian inference methods to neural networks (Section 2.4.1). We then provide an in-depth treatment of scalable Laplace...
approximations in particular, given their recent rise in popularity for approximate Bayesian inference in neural networks (Section 2.4.2).

2.4.1 Overview of Bayesian Deep Learning Methods

There have been significant efforts to characterize the posterior distribution over NN parameters \( p(\theta | D) \). While it is beyond the scope of this thesis to provide an exhaustive survey of existing Bayesian deep learning methods, we here provide an overview over some of the main methods in this research field. Some further relevant methods are reviewed later in Section 5.4.

To this day, Hamiltonian Monte Carlo (Neal, 1995) remains the gold standard for approximate inference in BNNs (Izmailov et al., 2021; Wenzel et al., 2020). Although asymptotically unbiased, sampling based approaches are difficult to scale to large datasets (Betancourt, 2015). As such, further approximations such as stochastic gradients (see Section 2.3.1) or distillation (Korattikara et al., 2015; Wang et al., 2018b) are often needed.

To avoid the issues with sampling-based methods, approaches which find the best surrogate posterior among an approximating family (most often Gaussians) have gained popularity. The first of these was the Laplace approximation, introduced by MacKay (1992d), who also proposed approximating the predictive posterior with that of the linearized model (Immer et al., 2021b; Khan et al., 2019). The LA has recently been shown to perform strongly (Foong et al., 2019a, b) and can be applied post-hoc to pre-trained models, making it particularly attractive for practical applications involving large pre-trained models. Due to their recent rise in popularity in the context of Bayesian deep learning, we will devote an entire section to review scalable Laplace approximations (see Section 2.4.2).

The popularisation of larger NN models has made surrogate distributions that capture correlations between weights computationally intractable. Thus, many modern methods make use of the mean-field assumption, i.e., assume complete independence between the individual neural network parameters, resulting in a full factorization of the posterior,

\[
p(\theta | D) \approx \prod_{d=1}^{D} q(\theta_d),
\]

where \( \theta_d \) is the \( d^{th} \) parameter in \( \theta \) (Blundell et al., 2015; Gal, 2016; Graves, 2011; Hernández-Lobato and Adams, 2015; Khan et al., 2018; Mishkin et al., 2018; Osawa et al., 2019).

However, it has been shown that assuming parameter independence comes at the cost of limited expressivity and suffers from severe pathologies (Foong et al., 2019a, b), resulting in empirical under-performance (Antorán et al., 2020; Ovadia et al., 2019). In contrast,
Farquhar et al. (2020) argue that in deeper networks the mean-field assumption should not be restrictive. There also exist methods that go beyond the mean-field assumption. For example, SWAG (Maddox et al., 2019b) uses the first and second empirical moment of SGD iterates to form a diagonal plus low-rank Gaussian approximation.

Most of these methods are fundamentally local approximations that cover only a single mode of the posterior. In contrast, recent works have demonstrated the benefit of capturing the multi-modality of the posterior distribution via ensembles/mixtures (Eschenhagen et al., 2021; Filos et al., 2019; Fort et al., 2019; Krueger et al., 2017; Lakshminarayanan et al., 2017; Wilson and Izmailov, 2020). Despite that, Wilson and Izmailov (2020) indicate that using a single mode might not be as limiting in practice as one might think, conjecturing that this is due to the complex, nonlinear connection between the parameter space and the function (output) space of NNs. Finally, for a review on priors in Bayesian deep learning, see Fortuin (2022).

### 2.4.2 Scalable Laplace Approximations

Despite compelling motivation for using BNNs, they have not gained much traction in practice. Common criticisms include that BNNs are difficult to implement, finicky to tune, expensive to train, and hard to scale to modern models and datasets.

For instance, popular variational Bayesian methods (Blundell et al., 2015; Graves, 2011; Hinton and van Camp, 1993, etc.) require considerable changes to the training procedure and model architecture. Also, their optimization process is slower and typically more unstable unless carefully tuned (Osawa et al., 2019). Other methods, such as deep ensembles (Lakshminarayanan et al., 2017), Monte Carlo dropout (Gal and Ghahramani, 2016), and SWAG (Maddox et al., 2019b) promise to bring uncertainty quantification to standard NNs in simple manners. But these methods either require a significant cost increase compared to a single network, have limited empirical performance, or an unsatisfying Bayesian interpretation.

As outlined in Section 2.3.3, naively applying the Laplace approximation (LA) to modern neural networks is intractable due to the required Hessian computation. In recent years, several works have addressed scalability, as well as other factors that affect approximation quality and predictive performance of the LA. These advances make the LA a simple and cost-efficient, yet competitive approximation method for inference in Bayesian deep learning. See Fig. 2.2 for an overview.

Two key advantages of the LA are that the local maximum is readily available from standard MAP training of NNs, and that curvature estimates can be easily and efficiently obtained thanks to recent advances in second-order optimization, both in terms of more efficient approximations to the Hessian (Botev et al., 2017; Heskes, 2000; Martens and
2.4 Bayesian Neural Networks

Fig. 2.2 Four key components to scale and apply the LA to a neural network \( f_\theta \) (with randomly-initialized or pre-trained weights \( \theta \)), with corresponding laplace code. ① We first choose which part of the model we want to perform inference over with the LA. ② We then select how to approximate the Hessian. ③ We can then perform model selection using the evidence: (a) If we started with an untrained model \( f_\theta \), we can jointly train the model and use the evidence to tune hyperparameters online. (b) If we started with a pre-trained model, we can use the evidence to tune the hyperparameters post-hoc. Here, shades represent the loss landscape, while contours represent LA log-posteriors—faded contours represent intermediate iterates during hyperparameter tuning to obtain the final log-posterior (thick yellow contours). ④ Finally, to make predictions for a new input \( x_* \), we have several options for computing/approximating the predictive distribution \( p(y_* | f_\theta(x_*)) \).

Grosse, 2015) and easy-to-use software libraries (Dangel et al., 2020; Osawa, 2021). Together, they make the LA practical and readily applicable to many already-trained NNs—the LA essentially enables practitioners to turn their high-performing point-estimate NNs into BNNs easily and quickly, without loss of predictive performance.

Yet, despite recent progress in scaling and improving the LA for deep learning (Daxberger et al., 2021c; Immer et al., 2021b; Khan et al., 2019; Kristiadi et al., 2020; Lee et al., 2020; Ritter et al., 2018a,b), it is far less widespread than other methods. This is likely due to misconceptions, like that the LA is hard to implement due to the Hessian computation, that it
must necessarily perform worse than the competitors due to its local nature, or quite simply that it is old and too simple. Here, we show that these are indeed misconceptions. Moreover, we argue that the LA deserves a wider adoption in both practical and research-oriented deep learning.

To this end, we now first survey recent advances and present four key components of scalable and practical Laplace approximations in deep learning. We then introduce laplace, an easy-to-use PyTorch-based library for “turning a NN into a BNN” via the LA. laplace implements a wide range of different LA variants.

Four Components of Scalable Laplace Approximations for Deep Neural Networks

1. Inference Over all Weights or Only the Last Layer

In most cases, it is possible to treat all weights probabilistically when using appropriate approximations of the Hessian, as we discuss below in 2. Another simple way to scale the LA to large NNs (without Hessian approximations) is to apply the LA to only the last linear layer of an $L$-layer NN, while fixing the feature extractor defined by the first $L - 1$ layers at its MAP estimate (Kristiadi et al., 2020; Snoek et al., 2015). More concretely, let $f_{\theta} : \mathbb{R}^I \rightarrow \mathbb{R}^O$ be an $L$-layer NN, and assume that the first $L - 1$ layers of $f_{\theta}$ is a feature map. Given MAP-trained parameters $\theta_{\text{MAP}}$, we define a Laplace-approximated posterior over the weight matrix $W^{(L)}$ of the $L$th layer,

$$p(W^{(L)} | D) \approx \mathcal{N}(W^{(L)}; W^{(L)}_{\text{MAP}}, \Sigma^{(L)})$$

and we leave the rest of the parameters with their MAP-estimated values. Since this matrix is small relative to the entire network, this last-layer LA is cost-effective, yet compelling both theoretically and in practice (Kristiadi et al., 2020). In Chapter 5, we will develop subnetwork Laplace inference, a new method that generalizes Eq. (2.43) to consider arbitrary subsets of weights.

2. Hessian Approximations and Their Factorizations

One advance in second-order optimization that the LA can benefit from are positive semi-definite approximations to the (potentially indefinite) Hessian of the log-likelihoods of NNs in the second term of Eq. (2.41) (Martens, 2020). One such choice is the Fisher information...
matrix (Amari, 1998), abbreviated as the Fisher and defined by

\[ F := \sum_{n=1}^{N} \mathbb{E}_{p(y|x_n)} s(x_n, y) s(x_n, y)^\top, \]  

(2.44)

where \( s(x, y) \) denotes the gradient of the log-likelihood at \( \theta_{\text{MAP}} \) for a given datum \((x, y)\), i.e.,

\[ s(x, y) := \nabla_{\theta} p(y | f_\theta(x)) |_{\theta_{\text{MAP}}}. \]

We shall refer to this matrix as the full Fisher. One can also use the generalized Gauss-Newton matrix (GGN) matrix (Schraudolph, 2002)

\[ G := \sum_{n=1}^{N} J(x_n) \left( \nabla_f^2 \log p(y_n | f) |_{f=f_\theta_{\text{MAP}}(x_n)} \right) J(x_n)^\top, \]

(2.45)

where

\[ J(x_n) := \nabla_{\theta} f_\theta(x_n) |_{\theta_{\text{MAP}}} \]  

(2.46)

is the NN’s Jacobian matrix. As the Fisher and GGN are equivalent for common log-likelihoods (Martens, 2020), we will henceforth refer to them interchangeably. In deep LAs, they have emerged as the default choice (Immer et al., 2021b; Kristiadi et al., 2020; Lee et al., 2020; Ritter et al., 2018a,b, etc.).

As \( F \) and \( G \) are still quadratically large (i.e., as large as the exact Hessian of the network), their computation is often infeasible. Thus, here, we review several factorization schemes that makes the computation (and storage) of the Fisher more efficient.

**Diagonal** The most lightweight is a diagonal factorization which ignores off-diagonal elements (Denker and LeCun, 1990; LeCun et al., 1990). Although MacKay recommended to not use the diagonal factorization of the Hessian (MacKay, 1992d), a recent work has indicated this factorization is usable for sufficiently deep NNs (Farquhar et al., 2020). In this factorization, we simply assume that the negative-log-posterior’s Hessian \( \Lambda \) is simply a diagonal matrix with diagonal elements equal the diagonal of the Fisher, i.e.

\[ \Lambda \approx -\text{diag}(F) \top I - \lambda I. \]

(2.47)

\(^3\)If, instead of taking expectation in equation 2.44, we use the training label \( y_n \), we call the matrix the empirical Fisher, which is distinct from the Fisher (Kunstner et al., 2019; Martens, 2020).
Since we can write

\[ \text{diag}(F) = \sum_{n=1}^{N} \mathbb{E}_{p(y|ϕ_{MAP}(x_n))}[s(x_n, y) \odot s(x_n, y)], \tag{2.48} \]

where the operator \( \odot \) denotes the Hadamard product, this factorization is efficient: Not only does it require only a vector of length \( D \) to represent \( F \) but also it incurs only a \( \mathcal{O}(D) \) cost when inverting \( \Lambda \)—down from \( \mathcal{O}(D^3) \).

**KFAC** More expressive alternatives are block-diagonal factorizations such as *Kronecker-factored approximate curvature (KFAC)* (Botev et al., 2017; Eschenhagen et al., 2023; Heskes, 2000; Martens and Grosse, 2015), which factorizes each within-layer Fisher\(^4\) as a Kronecker product of two smaller matrices. KFAC has been successfully applied to the LA (Ritter et al., 2018a,b) and can be improved by low-rank approximations of the KFAC factors (Lee et al., 2020) by leveraging their eigendecompositions (George et al., 2018).

The KFAC factorization can be seen as a midpoint between the two extremes: diagonal factorization, which might be too restrictive, and the full Fisher, which is computationally infeasible. The key idea is to model the correlation between weights in the same layer but assume that any pair of weights from two different layers are independent—this is a more sophisticated assumption compared to the diagonal factorization since there, it is assumed that all weights are independent of each other. For any layer \( l = 1, \ldots, L \), denoting \( N_l \) as the number of hidden units at the \( l \)-th layer, let \( W^{(l)} \in \mathbb{R}^{N_l \times N_{l-1}} \) be the weight matrix of the \( l \)-th layer of the network, \( a^{(l)} \) the \( l \)-th hidden vector, and \( g^{(l)} \in \mathbb{R}^{N_l} \) the log-likelihood gradient w.r.t. \( a^{(l)} \). For each \( l = 1, \ldots, L \), we can then write the outer product inside expectation in Eq. (2.44) as

\[ s(x_n, y)s(x_n, y)^\top = a^{(l-1)}a^{(l-1)^\top} \odot g^{(l)}g^{(l)^\top}. \tag{2.49} \]

Furthermore, assuming that \( a^{(l-1)} \) is independent of \( g^{(l)} \), we obtain the approximation of the \( l \)-th diagonal block of \( F \), which we denote by \( F^{(l)} \),

\[ F^{(l)} \approx \mathbb{E}[a^{(l-1)}a^{(l-1)^\top}] \odot \mathbb{E}[g^{(l)}g^{(l)^\top}] =: A^{(l-1)} \odot G^{(l)}, \tag{2.50} \]

where we represent both the sum and the expectation in Eq. (2.44) as \( \mathbb{E} \) for brevity.

From the previous expression we can see that the space complexity for storing \( F^{(l)} \) is reduced to \( \mathcal{O}(N_l^2 + N_{l-1}^2) \), down from \( \mathcal{O}(N_l^2N_{l-1}^2) \). Considering all \( L \) layers of the network, we obtain the layer-wise Kronecker factors \( \{A^{(l)}\}_{l=0}^{L-1} \) and \( \{G^{(l)}\}_{l=1}^{L} \) of the log-likelihood’s Hessian. This corresponds to the block-diagonal approximation of the full Hessian.

\(^4\)I.e., the elements in \( F \) or \( G \) corresponding to the weight matrix \( W^{(l)} \in \mathbf{θ} \) of the \( l \)-th layer of the network.
2.4 Bayesian Neural Networks

One can then readily use these Kronecker factors in a Laplace approximation. For each layer \(l\), we obtain the \(l\)th diagonal block of \(\Lambda\)—denoted \(\Lambda^{(l)}\)—by

\[
\Lambda^{(l)} \approx \left( A^{(l-1)} + \sqrt{\lambda} I \right) \otimes \left( G^{(l)} + \sqrt{\lambda} I \right)
\]

\[
=: V^{(l)} \otimes U^{(l)}.
\]

Note that we take the square root of the prior precision to avoid “double-counting” the effect of the prior. Nonetheless, this can still be a crude approximation (Immer et al., 2021b; Martens and Grosse, 2015). This particular Laplace approximation has been studied by Ritter et al. (2018a,b) and can be seen as approximating the posterior of each \(W^{(l)}\) with the matrix-variate Gaussian distribution (Gupta and Nagar, 1999): \(p(W^{(l)} | D) \approx \mathcal{MN}(W^{(l)}, W_{\text{MAP}}^{(l)}, U^{(l)-1}, V^{(l)-1})\). Hence, sampling can be done easily in a layer-wise manner:

\[
W^{(l)} \sim p(W^{(l)} | D) \iff W^{(l)} = W_{\text{MAP}}^{(l)} + U^{(l)-1} E V^{(l)-\frac{1}{2}}
\]

where \(E \sim \mathcal{MN}(0, I_{N_l}, I_{N_{l-1}})\),

where we have denoted by \(I_b\) the identity \(b \times b\) matrix, for \(b \in \mathbb{N}\). Note that the above matrix inversions and square-root are in general much cheaper than those involving the entire \(\Lambda\). Sampling \(E\) is not a problem either since \(\mathcal{MN}(0, I_{N_l}, I_{N_{l-1}})\) is equivalent to the standard \((N_l N_{l-1})\)-variate Normal distribution. As an alternative, Immer et al. (2021b) suggest to incorporate the prior exactly using an eigendecomposition of the individual Kronecker factors, which can improve performance.

**Low-rank block-diagonal** We can improve KFAC’s efficiency by considering its low-rank factorization (Lee et al., 2020). The key idea is to eigendecompose the Kronecker factors in Eq. (2.50) and keep only the eigenvectors corresponding to the first \(k\) largest eigenvalues. This can be done employing the eigenvalue-corrected KFAC (George et al., 2018). That is, for each layer \(l = 1, \ldots, L\):

\[
F^{(l)} \approx \left( U_A^{(l-1)} S_A^{(l-1)} U_A^{(l-1)\top} \right) \otimes \left( U_G^{(l)} S_G^{(l)} U_G^{(l)\top} \right)
\]

\[
= \left( U_A^{(l-1)} \otimes U_G^{(l)} \right) \left( S_A^{(l-1)} \otimes S_G^{(l)} \right) \left( U_A^{(l-1)} \otimes U_G^{(l)} \right)^\top.
\]

Under this decomposition, one can the easily obtain the optimal rank-\(k\) approximation of \(F^{(l)}\), denoted by \(F_k^{(l)}\), by selecting the top-\(k\) eigenvalues. However, the diagonal of this rank-\(k\) matrix can deviate too far from the exact diagonal elements of \(F^{(l)}\). Hence, one can make
the diagonal of this low rank matrix exact replacing \( \text{diag}(F^k) \) with \( \text{diag}(F^{(l)}) \), and obtain the following rank-\( k \)-plus-diagonal approximation of \( F^{(l)} \):

\[
F^{(l)} \approx F^k + \text{diag}(F^{(l)}) - \text{diag}(F^k).
\]

This factorization can be seen as a combination of the previous two approximations: For each diagonal block of \( F \), we use the exact diagonal elements of \( F \) and approximate the off-diagonal elements with a rank-\( k \) matrix arising from KFAC. Both the space and computational complexities are lower than those of KFAC since here we work exclusively with truncated and diagonal matrices.

**Low-rank** Instead of only approximating each block by a low-rank structure, the entire Hessian or GGN can also be approximated by a low-rank structure \((\text{Maddox et al., 2020; Sharma et al., 2021})\). Eigendecomposition of \( F \) is a convenient way to obtain a low-rank approximation. The eigendecomposition of \( F \) is given by \( Q L Q^T \) where the columns of \( Q \in \mathbb{R}^{D \times D} \) are eigenvectors of \( F \) and \( L = \text{diag}(l) \) is a \( D \)-dimensional diagonal matrix of eigenvalues. Assuming the eigenvalues in \( l \) are arranged in a descending order, the optimal \( k \)-rank approximation in Frobenius or spectral norm is given by truncation \((\text{Eckart and Young, 1936})\): let \( \hat{Q} \in \mathbb{R}^{D \times k} \) be the matrix of the first \( k \) eigenvectors corresponding to the largest \( k \) eigenvalues \( \hat{l} \in \mathbb{R}^k \). That is, we truncate all eigenvectors and eigenvalues after the \( k \) largest eigenvalues. The low-rank approximation is then given by

\[
F \approx \hat{Q} \text{diag}(\hat{l}) \hat{Q}^T.
\]

The rank \( k \) can be chosen based on the eigenvalues so as to retain as much information of the Hessian (approximation) as possible. Further, sampling and computation of the log-determinant can be carried out efficiently.

**Functional** When considering network linearization for the predictive distribution, we can directly infer the Gaussian distribution on the outputs, of which there are typically few, instead of inferring a distribution on the parameters, of which there are many \((\text{Immer et al., 2021b; Khan et al., 2019})\).

3 Hyperparameter Tuning

As with all approximate inference methods, the performance of the LA depends on the hyper-parameters of the prior and likelihood. For instance, it is typically beneficial to tune the
2.4 Bayesian Neural Networks

prior variance $\gamma^2$ used for inference\(^5\) (Immer et al., 2021a,b; Kristiadi et al., 2020; Ritter et al., 2018b). In this section, we thus focus on tuning the prior variance/precision hyperparameter for simplicity. The same principle can be used for tuning other hyperparameters, such as the observation noise variance $\sigma_n^2$ in the case of regression (cf. Eq. (2.12)).

**Post-hoc** Here, we assume that the steps of the Laplace approximation—MAP training and forming the Gaussian approximation—as two independent steps. As such, we are free to choose different prior variance $\gamma^2$ in the latter part, irrespective to the weight decay hyperparameter used in the former. Here, we review several ways to optimize $\gamma^2$ post-hoc. Ritter et al. (2018b) propose to use cross-validation, i.e. to tune $\gamma^2$ by maximizing the log-posterior-predictive over a validation set $D_{\text{val}} := \{(x_n, y_n)_{n=1}^{N_{\text{val}}}$.

That is, we solve the following one-parameter optimization problem:

$$\gamma_n^2 = \arg\max_{\gamma^2} \sum_{n=1}^{N_{\text{val}}} \log p(y_n | x_n, D).$$

(2.51)

However, Kristiadi et al. (2020) found that the previous objective tends to make the Laplace approximation overconfident to outliers. Hence, they proposed to add an auxiliary term that depends on an out-of-distribution dataset $D_{\text{out}} := \{(x_n^{(\text{out})})_{n=1}^{N_{\text{out}}}$ to equation 2.51, as follows

$$\gamma_n^2 = \arg\max_{\gamma^2} \sum_{n=1}^{N_{\text{val}}} \log p(y_n | x_n, D) + \lambda \sum_{n=1}^{N_{\text{out}}} H \left[ p(y_n | x_n^{(\text{out})}, D) \right],$$

(2.52)

where $H$ is the entropy functional and $\lambda \in (0, 1]$ is a trade-off hyperparameter. Intuitively, we choose $\gamma^2$ that balances the calibration on the true dataset and the low-confidence on outliers. Moreover, other losses could be constructed to tune the prior precision for optimal performance w.r.t. some desired quantity.

A more principled alternative for hyperparameter tuning with the LA that requires no validation data is marginal likelihood maximization, a.k.a. empirical Bayes or the evidence framework (Bernardo and Smith, 2009; MacKay, 1992b). Immer et al. (2021a) showed that marginal likelihood maximization with LA can work in deep learning; see further details below in Online. Note that such an approach is not necessarily feasible for other approximate inference methods because most do not provide an estimate of the marginal likelihood.

Finally, other recent approaches for hyperparameter tuning for the LA include Bayesian optimization (Hunt et al., 2020) or the addition of dedicated, trainable hidden units for the sole purpose of uncertainty tuning (Kristiadi et al., 2021).

---

\(^5\)Equivalently, one can also tune the inverse of the prior variance, namely the prior *precision* $\gamma^{-2}$. 

Algorithm 1 Online Laplace (adapted from Immer et al. (2021a, Algorithm 1))

1: Input: NN $f_\theta$; training set $D$; learning rate $\alpha_0$ and number of epochs $T_0$ for MAP estimation; learning rate $\alpha_1$ and number of epochs $T_1$ for hyperparameter tuning; marginal likelihood maximization frequency $F$.

2: Initialize $\theta_0$

3: for $t = 1, \ldots, T_0$ do

4: $g_t \leftarrow \nabla_{\theta} L(D; \theta)|_{\theta_{t-1}}$

5: $\theta_t \leftarrow \theta_{t-1} - \alpha_0 g_t$

6: if $t \mod F = 0$ then

7: $p(\theta | D) \approx \mathcal{N}(\theta; \theta_t, (\nabla^2 L(D; \theta)|_{\theta_t})^{-1})$ Perform a Laplace approximation

8: for $i = 1, \ldots, T_1$ do

9: Hyperparameter optimization

10: $h_i \leftarrow -\nabla_{\gamma^2} \log p(D | \gamma^2)|_{\gamma^2_{t-1}}$ The marginal likelihood follows from equation A.3

11: $\gamma^2_I \leftarrow \gamma^2_{t-1} + \alpha_1 h_i$

12: end for

13: end if

14: end for

15: Output: $\theta_{T_0}, \nabla^2 L(D; \theta)|_{\theta_{T_0}}$

**Online** Contrary to the post-hoc tuning above, here we perform a Laplace approximation and tune the prior variance simultaneously as we perform a MAP training (Immer et al., 2021a). The key is to form a Laplace-approximated posterior every $B$ epochs of a gradient descent, and use this posterior to approximate the marginal likelihood (cf. Eq. (A.3)). By maximizing this marginal likelihood, we can find the best hyperparameters. Thus, once the MAP training has finished, we automatically obtain a prior variance that is already suitable for the Laplace approximation. Note that, this way, only a single MAP training needs to be done. This is in contrast to the classic, offline evidence framework (MacKay, 1992b) where the marginal likelihood maximization is performed only when the MAP estimation is done, and these steps need to be iteratively done until convergence. As a final note, similar to the post-hoc marginal likelihood above, this online Laplace does not require a validation set and has an additional benefit of improving the network’s generalization performance (Immer et al., 2021a). See Algorithm 1 for an overview.

**Approximate Predictive Distribution**

Here, we denote $x_* \in \mathbb{R}^I$ to be a test point, and $f_* = \theta$ be the network output at this point. We will review different way to approximate the posterior predictive distribution $p(y_* | x_*, D)$
in Eq. (2.5) given the Laplace-approximated Gaussian posterior \( p(\theta \mid D) \approx q(\theta \mid D) = \mathcal{N}(\theta; \theta_{MAP}, \Sigma) \) in Eq. (2.40).

**Monte Carlo integration** The simplest but most general and unbiased approximation is the Monte Carlo (MC) integration, which can be performed by sampling an approximate posterior \( q(\theta \mid D) \) repeatedly:

\[
p(y_* \mid x_*, D) \approx \frac{1}{S} \sum_{s=1}^{S} p(y_* \mid f_{\theta_s}(x_*)), \quad \text{where } \theta_s \sim q(\theta \mid D).
\]

While the error of this approximation decays like \( 1/\sqrt{S} \) and thus requires many samples to be accurate, for practical BNNs, it is standard to use 10 or 20 samples of \( q(\theta \mid D) \) (Blundell et al., 2015; Kristiadi et al., 2020; Ritter et al., 2018b, etc.). Note that this approximation can be used regardless the form of the likelihood \( p(y_* \mid f_{\theta}(x)) \). In particular, it can be used to directly obtain the predictive distribution in both the regression and classification alike. However, for LAs with GGN and Fisher Hessian approximations, Monte Carlo integration can perform poorly (Foong et al., 2019b; Immer et al., 2021b).

**Linearization** Immer et al. (2021b) attribute the poor performance of MC integration to the inconsistency between Hessian approximation and the predictive distribution. More precisely, when using a LA with the common Fisher / GGN Hessian approximation, the underlying model is implicitly linearized. Put differently, the Gaussian distribution in Eq. (2.40) with a Fisher / GGN precision matrix (cf. Eqs. (2.44) and (2.45)) corresponds to the true posterior when the NN is approximated with a first-order Taylor expansion around \( \theta_{MAP} \) (Immer et al., 2021b; Khan et al., 2019).\(^6\) The (locally) linearized neural network is

\[
f_{\theta}(x_*) \approx f_{\theta_{MAP}}^{lin}(x_*) = f_{\theta_{MAP}}(x_*) + J(x_*)^\top(\theta - \theta_{MAP}),
\]

where \( J(x_*) \) is the Jacobian matrix of the network output. This turns the underlying probabilistic model from a BNN into a generalized linear model (GLM), where the Jacobian \( J(x_*) \) acts as a basis function expansion. To make predictions, Immer et al. (2021b) thus propose to use the GLM \( f_{\theta_{MAP}}^{lin} \) instead of the BNN \( f_{\theta} \), which has been found to improve performance (Foong et al., 2019b; Lawrence, 2001). Note that for the last-layer LA, the Hessian coincides with the GGN and the linearized predictive is exact.

This way, under a Gaussian approximate posterior \( q(\theta \mid D) = \mathcal{N}(\theta; \theta_{MAP}, \Sigma) \), the marginal distribution over neural network outputs \( f_* := f(x_*) \) (i.e., where \( \theta \) in \( f_{\theta}(x_*) \) is integrated

\(^6\)This equivalence assumes a Gaussian likelihood, but the same principle applies to other likelihoods as well.
out) is again Gaussian, given by\(^7\)

\[
p(f_* | f_\theta(x_\star), x_\star, D) = \int_\theta \delta(f_* - f_\theta(x_\star)) q(\theta | D) d\theta
\]

\[
\approx \mathcal{N}(f_*; f_{\text{MAP}}(x_\star)J(x_\star)^\top \Sigma J(x_\star)).
\]

This distribution over function outputs \(f_*\) is typically significantly lower-dimensional (number of outputs \(O\) instead of number of parameters \(D\)). This approximation has been extensively used for small networks (MacKay, 1992b), but it has since gone out of favor in deep learning due to its cost—the Jacobian \(J(x_\star)\) needs to be computed per input point. Nevertheless, this approximation is still useful in theoretical works due to its analytical nature (Eschenhagen et al., 2021; Kristiadi et al., 2020, 2021). Moreover, in problems where it can be efficiently used in practice, it offers a better approximation than the MC-integral (Foong et al., 2019b; Immer et al., 2021b). Due to the linearization in the network parameters, it is further possible to obtain a functional prior in the form of a Gaussian process (Immer et al., 2021b; Khan et al., 2019). This allows to perform function-space inference as opposed to weight-space inference which is amenable to different Hessian approximations than those pointed out above in (2), and is, for example, useful for continual learning (Pan et al., 2020). Recently, there have been many works studying and improving the linearized LA (Antoran et al., 2023; Bergamin et al., 2023; Immer et al., 2023; Kristiadi et al., 2022a,b; Sharma et al., 2023).

Given the distribution over outputs \(f_*\) in Eq. (2.54), the predictive distribution can again be obtained by integration against the likelihood (cf. Eq. (2.5)):

\[
p(y_\star | x_\star, D) = \int_{f_*} p(y_\star | f_*) p(f_* | x_\star, D) df_*.
\]

(2.56)

In the case of regression with a Gaussian likelihood with variance \(\sigma^2\), the solution can then even be obtained analytically (since the integral in Eq. (2.56) is just a convolution of two Gaussian r.v.s.):

\[
p(y_* | x_\star, D) \approx \int_{f_*} \mathcal{N}(y_*; f_*, \sigma^2 I) \mathcal{N}(f_*; f_{\text{MAP}}(x_\star), J(x_\star)^\top \Sigma J(x_\star)) df_*
\]

\[
= \mathcal{N}(y_*; f_{\text{MAP}}(x_\star), \Sigma(x_\star) + \sigma^2 I),
\]

(2.57)

(2.58)

where \(\Sigma(x_\star) = J(x_\star)^\top \Sigma J(x_\star)\). In the case of classification, the likelihood \(p(y_* | f_*)\) is non-Gaussian, so we cannot analytically obtain \(p(y_* | x_\star, D)\). Thus, in this case we are interested

\(^7\)See Bishop (2006, Sec. 4.5.2).
in approximating the intractable integral

\[ p(y_*|x_*,D) = \int_{f_*} p(y_*|f_*) N(f_*;\mu(x_*),\Sigma(x_*)) df_* \quad (2.59) \]

with \( \mu(x_*) = f_{\text{MAP}}(x_*) \),

\[ \Sigma(x_*) = J(x_*)^\top J(x_*) \quad (2.60) \]

where \( p(y_*|f_*) \) is constructed via an inverse-link function; i.e., for binary classification, \( p(y_*|f_*) = \sigma(f_*) \) where \( \sigma \) is the logistic-sigmoid function, and in the general multi-class classification case, \( p(y_*|f_*) = \text{softmax}(f_*) \). We now review the most common approaches for approximating the integral in Eq. (2.59). Again, the simplest approximation is Monte Carlo integration. Alternatively, there exist a few closed-form approximations. The delta method (Ahmed and Xing, 2007; Braun and McAuliffe, 2010; Wu et al., 2019) uses a Taylor-expansion of the softmax function around \( \mu(x_*) \) up to the second order, which results in the analytic expression

\[ p(y_*|x_*,D) \approx \text{softmax}(\mu(x_*)) + 1/2 \text{tr}(B\Sigma(x_*)), \quad (2.62) \]

where \( B \) is the Hessian matrix of the softmax at \( \mu(x_*) \). Another method is the probit approximation. In the binary case, it approximates the logistic function \( \sigma \) with the probit function \( \Phi \)—the standard Normal c.d.f.—which makes the integral solvable analytically (MacKay, 1992a,b; Spiegelhalter and Lauritzen, 1990). We then obtain the closed-form approximation

\[ p(y_*|x_*,D) \approx \int_{f_*} \Phi(f_*) N(f_*;\mu(x_*),\sigma_*^2) df_* \quad (2.63) \]

\[ = \sigma \left( \frac{\mu(x_*)}{\sqrt{1 + \frac{\sigma_*^2}{2} \sigma_*^2}} \right). \quad (2.64) \]

For the multi-class case, Gibbs (1998) generalized the probit approximation as

\[ p(y_*|x_*,D) \approx \exp(\tau_i) \left[ \sum_{j=1}^{O} \exp(\tau_j) \right]_{i=1,...,O} \quad (2.65) \]

where \( \tau_i = \frac{\mu(x_*)_i}{\sqrt{1 + \frac{\sigma_*^2}{2} \Sigma(x_*)_{ii}}} \),

\[ \mu(x_*)_i = \mu(x_*)_i \quad (2.66) \]
where \([x_i]_{i=1,...,O} \in \mathbb{R}^O\) denotes a \(O\)-dimensional vector with elements \(x_j\). This approximation ignores the correlation between logits since it only depends on the diagonal of \(\Sigma(x)\). Nevertheless, it yields good results even in deep learning (Lu et al., 2020), and is an invaluable tool for theoretical work (Eschenhagen et al., 2021). Finally, first proposed for non-BNN applications (Hennig et al., 2012; MacKay, 1998), the *Laplace bridge* performs a Laplace approximation to the Dirichlet distribution by first writing it as a distribution over \(\mathbb{R}^O\) with the help of the softmax function. This way, the Laplace approximation can be reasonably applied to approximate the Dirichlet, which can be thought as mapping the Dirichlet \(\text{Dir}(\alpha)\) to a Gaussian \(\mathcal{N}(\mu(x), \Sigma(x))\). The *Laplace bridge* is the pseudo-inverse of this map, mapping \((\mu(x), \Sigma(x))\) to \(\alpha\) where for each \(i = 1, \ldots, O\), the \(i\)th component \(\alpha\) is given by the simple closed-form expression

\[
\alpha_i = \frac{1}{\Sigma(x)_{ii}} \left( 1 - \frac{2}{O} \exp(\mu(x)_i) \sum_{j=1}^{O} \exp(-\mu(x)_j) \right).
\]

Just like the probit approximation, the Laplace bridge ignores the correlation between logits. But, unlike all the previous approximations, it yields a *full distribution* over the solutions of the softmax-Gaussian integral in Eq. (2.59). So, the Laplace bridge is a richer yet comparably simple approximation to the integral and is useful for many applications in deep BNNs (Hobbhahn et al., 2022).

**laplace**: A Toolkit for Deep Laplace Approximations

Implementing the LA is non-trivial, as it requires efficient computation and storage of the Hessian. While this is not fundamentally difficult, there exists no complete, easy-to-use, and standardized implementation of various LA flavors—instead, it is common for deep learning researchers to repeatedly re-implement the LA and Hessian computation with varying efficiency (Kristiadi, 2020; Lee and Humt, 2020; Maddox et al., 2019a, etc.). An efficient implementation typically requires hundreds of lines of code, making it hard to quickly prototype with the LA. To address this, we introduce *laplace*: a simple, easy-to-use, extensible library for scalable LAs of deep NNs in PyTorch (Paszke et al., 2019). *laplace* enables *all* sensible combinations of the four components discussed in Section 2.4.2—see Fig. 2.2 for details. Listings 2.1 and 2.2 show code examples.

The core of *laplace* consists of efficient implementations of the LA’s key quantities: (i) posterior (i.e., Hessian computation and storage), (ii) marginal likelihood, and (iii) posterior predictive. For (i), to take advantage of advances in automatic differentiation, we outsource the Hessian computation to state-of-the-art, optimized second-order optimization libraries:
from laplace import Laplace

# Load pre-trained model
model = load_map_model()

# Define and fit LA variant with custom settings
la = Laplace(model, 'classification',
              subset_of_weights='all',
              hessian_structure='diag')
la.fit(train_loader)
la.optimize_prior_precision(method='CV',
                            val_loader=val_loader)
# Make prediction with custom predictive approx.
pred = la(x, pred_type='glm', link_approx='probit')

Listing 2.1 Fit diagonal LA over all weights of a pre-trained classification model, do post-hoc tuning of the prior precision hyperparameter using cross-validation, and make a prediction for input \( x \) with the probit approximation.

Listing 2.2 Fit KFAC LA over the last layer of a pre- or un-trained regression model and differentiate its marginal likelihood w.r.t. some hyperparameters for post-hoc hyperparameter tuning or online empirical Bayes (see Immer et al. (2021a)).

BackPACK (Dangel et al., 2020) and ASDL (Osawa, 2021). Moreover, we design \texttt{laplace} in a modular manner that makes it easy to add new backends and approximations in the future. For (ii), we follow Immer et al. (2021a) in our implementation of the LA’s marginal likelihood—it is thus both efficient and differentiable and allows the user to implement both online and post-hoc marginal likelihood tuning, cf. Listing 2.2. Note that \texttt{laplace} also supports standard cross-validation for hyperparameter tuning (Kristiadi et al., 2020; Ritter et al., 2018b), as shown in Listing 2.1. Finally, for (iii), \texttt{laplace} supports all approximations to the posterior predictive distribution discussed in Section 2.4.2—it thus provides the user with flexibility in making predictions, depending on the computational budget.

**Default behavior** To abstract away from a large number of options available (Section 2.4.2), we provide the following default choices based on our extensive experiments (detailed in Daxberger et al. (2021b)); they should be applicable and perform decently in the majority of use cases: we assume a pre-trained network and treat only the last-layer weights probabilistically (last-layer LA), use the KFAC factorization of the GGN and tune the hyperparameters post-hoc using empirical Bayes. To make predictions, we use the closed-form Gaussian predictive distribution for regression and the (extended) probit approximation for classification. Of course, the user can pick custom choices (Listings 2.1 and 2.2).

**Limitations** Because \texttt{laplace} employs external libraries (BackPACK (Dangel et al., 2020) and ASDL (Osawa, 2021)) as backends, it inherits the available choices of Hessian
factorizations from these libraries. For instance, the LA variant proposed by Lee et al. (2020) can currently not be implemented via laplace, because neither backend supports eigenvalue-corrected KFAC (George et al., 2018) (yet).

2.5 Variational Autoencoders

In addition to Bayesian neural networks, we consider a second probabilistic deep learning model in this thesis, variational autoencoders (VAEs). As the name suggests, VAEs are based on variational inference (VI), which is a popular optimization-based approximate inference technique. We first clarify the main conceptual distinction between BNNs, which are supervised, discriminative models that require inference over the global model parameters, and VAEs, which are unsupervised, generative models that require inference over local latent variables (Section 2.5.1). We then introduce latent variable models and describe how to use variational inference to infer their latent variables (Section 2.5.2). Finally, we describe how VAEs can be derived as deep latent variables models by combining latent variable models with deep learning (Section 2.5.3). Parts of this section follow the excellent reviews of VAEs and VI by Doersch (2016) and Blei et al. (2017), respectively.

2.5.1 Bayesian Inference Over Global vs. Local Latent Variables

Recall that in the Bayesian paradigm, we are concerned with inferring unknown (a.k.a. latent) quantities given observations (and a model). We previously focused on the setting where the unknown quantity is the parameter vector $\theta$ of some parametric model and we observe a dataset $D = \{(x_n, y_n)\}_{n=1}^N$ of $N$ input-output pairs $(x_n, y_n)$ drawn i.i.d. from the joint data distribution $p(x, y) = p(x)p(y|x)$, see Eq. (2.4). The goal was to perform supervised learning using the labelled dataset $D$ to obtain a discriminative model $p(y|x,D)$ that can predict the outputs $y_s$ for new inputs $x_s$, i.e., $y_s \sim p(y|x_s,D)$. In contrast, we will now consider a different setting, where we only observe a dataset $D = \{x_n\}_{n=1}^N$ of $N$ inputs $x_n$ (without any corresponding outputs $y_n$) drawn i.i.d. from the data distribution $p(x)$. Our goal now is to perform unsupervised learning using the unlabelled dataset $D$ to obtain a generative model $p(x|D)$ that can produce new inputs $x_s$, i.e., $x_s \sim p(x|D)$. As we will see in the next section, this requires us to infer the latent variables $z_n$ corresponding to the inputs $x_n$, $1 \leq n \leq N$. Therefore, the main difference to the previous inference setting is that the latent variables $z_n$ are local, i.e., tied to individual inputs $x_n$, while the model parameters $\theta$ are global, i.e., shared amongst all input-output pairs $(x_n, y_n)$. We now introduce latent variable models in more detail and describe how VI is used to infer the local latent variables $z_n$. 
2.5 Variational Autoencoders

2.5.2 Variational Inference in Latent Variable Models

Consider a joint density of latent variables $z \in \mathbb{Z} \subseteq \mathbb{R}^Z$ and observed variables $x \in \mathcal{X} \subseteq \mathbb{R}^l$,

$$p(z, x) = p(z)p(x | z), \quad (2.67)$$

where the latent variables help govern the distribution of the data in some way. In particular, Eq. (2.67) encodes the assumption that an input $x$ is generated by first drawing a latent variable $z$ from a prior density $p(z)$, and then obtaining an input $x$ by feeding $z$ through the likelihood $p(x | z)$. Bayesian inference in this latent variable model then amounts to computing the posterior distribution $p(z | x)$ over the latent variables $z$ given inputs $x$. Following Bayes rule, the posterior can be written as

$$p(z | x) = \frac{p(z, x)}{p(x)} = \frac{p(x | z)p(z)}{p(x)}, \quad (2.68)$$

where the evidence/marginal likelihood $p(x)$ in the denominator marginalizes out the latents from the joint density, i.e.,

$$p(x) = \int_z p(z, x)dz = \int_z p(z)p(x | z)dz. \quad (2.69)$$

As described earlier, the integral in Eq. (2.69) is typically intractable. This motivates the use of variational inference, which tries to find the member $q \in Q$ of the variational family $\mathcal{Q}$ that minimizes the KL divergence to the posterior, i.e.,

$$q_*(z) = \arg \min_{q \in \mathcal{Q}} D_{KL}(q(z) || p(z | x)). \quad (2.70)$$

The family $\mathcal{Q}$ is typically parameterized by free "variational parameters".
Unfortunately, Eq. (2.70) is still intractable as it also requires computing the evidence \( p(x) \) in Eq. (2.69). To see this, we can rewrite the KL divergence as

\[
D_{\text{KL}}(q(z) \| p(z \| x)) = \int_z q(z) \log \frac{q(z)}{p(z \| x)} \, dz \\
= \mathbb{E}_q(z) \left[ \log \frac{q(z)}{p(z \| x)} \right] \\
= \mathbb{E}_q(z) [\log q(z) - \log p(z \| x)] \\
= \mathbb{E}_q(z) [\log q(z)] - \mathbb{E}_q(z) [\log p(z \| x)] \\
= \mathbb{E}_q(z) [\log q(z)] - \mathbb{E}_q(z) \left[ \log \frac{p(z, x)}{p(x)} \right] \\
= \mathbb{E}_q(z) [\log q(z)] - \mathbb{E}_q(z) [\log p(z, x)] + \log p(x),
\]

(2.71)

since \( \mathbb{E}_q(z) [\log p(x)] = \log p(x) \), revealing the dependence on \( p(x) \).

As we cannot compute the KL divergence, we instead optimize the evidence lower bound (ELBO) (or variational lower bound)

\[
\text{ELBO}(q) = \mathbb{E}_q(z) [\log p(z, x)] - \mathbb{E}_q(z) [\log q(z)] \\
= -D_{\text{KL}}(q(z) \| p(z \| x)) + \log p(x). \tag{2.72}
\]

(2.73)

As the ELBO in Eq. (2.73) is equivalent to the negative KL divergence in Eq. (2.71) plus \( \log p(x) \), which is a constant with respect to \( q(z) \), maximizing the ELBO is equivalent to minimizing the KL divergence. To see why the evidence lower bound is a lower bound to the (log) evidence (i.e., for any \( q(z) \)), we observe that Eq. (2.71) and Eq. (2.72) are related as

\[
\log p(x) = D_{\text{KL}}(q(z) \| p(z \| x)) + \text{ELBO}(q) \geq \text{ELBO}(q) \tag{2.74}
\]

where the inequality is due to the non-negativity of the KL divergence.\(^8\) The tightness of this bound is quantified by the KL divergence \( D_{\text{KL}}(q(z) \| p(z \| x)) \): The better \( q(z) \) approximates the true posterior \( p(z \| x) \) in terms of KL divergence, the smaller the gap between the ELBO and the log marginal likelihood \( \log p(x) \).

\(^8\)Note that in the original literature on variational inference (Jordan et al., 1999), the ELBO was derived as

\[
\log p(x) = \log \int_z p(z, x) \, dz = \log \int_z q(z) \frac{p(z, x)}{q(z)} \, dz = \log \mathbb{E}_{q(z)} \left[ \frac{p(z, x)}{q(z)} \right] \\
\geq \mathbb{E}_{q(z)} \left[ \log \frac{p(z, x)}{q(z)} \right] = \mathbb{E}_{q(z)} [\log p(z, x)] - \mathbb{E}_{q(z)} [\log q(z)] = \text{ELBO}(q) \tag{2.75}
\]

(2.75)

\[
\geq \mathbb{E}_{q(z)} \left[ \log \frac{p(z, x)}{q(z)} \right] = \mathbb{E}_{q(z)} [\log p(z, x)] - \mathbb{E}_{q(z)} [\log q(z)] = \text{ELBO}(q) \tag{2.76}
\]

(2.76)

which uses Jensen’s inequality due to the concavity of the logarithm.
Importantly, Eq. (2.73) reveals that maximizing the ELBO will simultaneously optimize the two things we care about:

1. It will approximately maximize the marginal likelihood \( p(x) \), which will improve our generative model. Thus, the ELBO can be used as a model selection criterion.

2. It will minimize the KL divergence between the approximation \( q(z) \) and the true posterior \( p(z|x) \), which will improve \( q(z) \).

To obtain further intuition and insights about the optimal variational distribution \( q_* \), we can rewrite the ELBO as

\[
\text{ELBO}(q) = \mathbb{E}_{q(z)}[\log p(z, x)] - \mathbb{E}_{q(z)}[\log q(z)]
\]

\[
= \mathbb{E}_{q(z)}[\log p(x | z)p(z)] - \mathbb{E}_{q(z)}[\log q(z)]
\]

\[
= \mathbb{E}_{q(z)}[\log p(x | z)] + \mathbb{E}_{q(z)}[\log p(z)] - \mathbb{E}_{q(z)}[\log q(z)]
\]

\[
= \mathbb{E}_{q(z)}[\log p(x | z)] + \mathbb{E}_{q(z)} \left[ \log \frac{p(z)}{q(z)} \right]
\]

\[
= \mathbb{E}_{q(z)}[\log p(x | z)] - D_{KL}(q(z) || p(z)), \tag{2.77}
\]

i.e., a sum of the expected log-likelihood of the data and the KL divergence between the prior \( p(z) \) and \( q(z) \). The ELBO in Eq. (2.77) thus encourages \( q(z) \) to place its mass on values of the latent variables \( z \) as follows:

1. The first term in Eq. (2.77) is an expected likelihood, which encourages densities that place their mass on configurations of \( z \) that explain the observed data. This is because \( \mathbb{E}_{q(z)}[\log p(x | z)] = \int q(z) \log p(x | z)dz \) is large when the integrand \( q(z) \log p(x | z) \) is large for all \( z \), which means that \( q(z) \) should be large for \( z \) for which also \( \log p(x | z) \) is large, i.e., \( z \) that have a high probability of having generated the data \( x \).

2. The second term in Eq. (2.77) is the negative KL divergence between the variational density and the prior, which encourages \( q(z) \) to be close to the prior \( p(z) \). The term serves as a regularizer, as it discourages variational densities to produce (almost) deterministic latent variables by having very small standard deviations. While this would make optimization easier, it would not yield robust latent variables \( z \), as small changes in the value of \( z \) might then result in a significantly change in the likelihood value \( \log p(x | z) \).

Another view is obtained by rewriting the ELBO as

\[
\text{ELBO}(q) = \mathbb{E}_{q(z)}[\log p(x | z)] + \mathbb{E}_{q(z)}[\log p(z)] - \mathbb{E}_{q(z)}[\log q(z)], \tag{2.78}
\]
where the first two terms encourage \( q(z) \) to stay close to the MAP estimate (i.e., the product of likelihood and prior), while the third term encourages \( q(z) \) to remain diffuse by having maximal entropy \( \mathbb{H}[q(z)] = \mathbb{E}_{q(z)}[-\log q(z)] \).

Overall, there are (at least) three perspectives to interpretations of the variational objective defined by the ELBO:

1. **The statistical estimation perspective.** The ELBO mirrors the balance between modelling accuracy and regularization penalty.

2. **The Bayesian perspective.** The ELBO mirrors the balance between the likelihood and the prior.

3. **The information-theoretical perspective.** There is a strong connection between the ELBO and the *minimum description length* (MDL) principle, as described in Doersch (2016).

### 2.5.3 VAEs as Deep Latent Variable Models

Consider a latent-variable model \( p_\theta(x, z) \) with marginal likelihood

\[
p_\theta(x) = \int_z p_\theta(x, z)dz = \int_z p_\theta(x | z)p(z)dz,
\]

(2.79)

where the likelihood \( p_\theta(x | z) \) is parameterized by a vector \( \theta \). As we are interested in modeling complicated distributions \( p(x) \), we focus on the special case of *deep* latent variable models (DLVMs) where \( \theta \) corresponds to the parameters of a deep neural network. DLVMs are particularly expressive and allow for very complex marginal distributions \( p_\theta(x) \) containing almost arbitrary dependencies, even when the conditional distribution \( p_\theta(x | z) \) is simple (e.g., Gaussian).

In a nutshell, a *variational autoencoder* (VAE) (Kingma and Welling, 2014; Rezende et al., 2014) is a DLVM which approximates the intractable posterior \( p_\theta(z | x) \) over latent variables \( z \) using *amortized variational inference* via another deep neural network \( q_\varphi(z | x) \) parameterized by \( \varphi \), and is trained by exploiting a reparameterization of the ELBO which yields a gradient estimator that can be straightforwardly optimized using standard stochastic gradient methods, thus enabling scalability to large datasets. We will now break down the individual components of a VAE, following Doersch (2016); Kingma (2017); Kingma and Welling (2014); Rezende et al. (2014).
2.5 Variational Autoencoders

**Decoder a.k.a. Generative Network**

A simple but common example DLVM defines the generative model \( p_\theta(x, z) = p(z)p_\theta(x | z) \) via a spherical Gaussian latent space and a factorized Gaussian observation model, i.e.,

\[
p(z) = \mathcal{N}(z; 0, I),
\]

\[
p_\theta(x | z) = \prod_i p_\theta(x_i | z) = \prod_i \mathcal{N}(x_i; \mu_i, \sigma_i^2) = \mathcal{N}(x; \mu, \text{diag}(\sigma^2)),
\]

where \((\mu, \log \sigma) = \text{DecoderNeuralNet}_\theta(z)\).

Doersch (2016) provides the following interpretation: The definition of \( p_\theta(x | z) \) formalizes the intuition that some \( z \sim p(z) \) needs to result in samples that are merely *like* \( x \). In general, and particularly early in training, our model will not produce outputs that are identical to any particular \( x \). By having a Gaussian distribution, we can use gradient descent (or any other optimization technique) to increase \( p(x) \) by making \( p_\theta(x | z) \) approach \( x \) for some \( z \), i.e., gradually making the training data more likely under the generative model. Note that this would not be possible if \( p_\theta(x | z) \) was a Dirac delta function, i.e., if we used \( x = f_\theta(z) \) for some deterministic function \( f_\theta(z) \). Note that, for instance, if \( x \) is binary, then the output distribution \( p_\theta(x | z) \) might be a Bernoulli distribution parameterized by \( \theta \). The important property is simply that \( p_\theta(x | z) \) can be computed, and is continuous in \( \theta \). Finally, the likelihood in a VAE is typically called the *decoder* or *generative network*.

**Encoder a.k.a. Inference Network**

Since inference of the posterior distribution \( p_\theta(z | x) \) of the DLVM is intractable, we use variational inference to approximate the posterior via an *inference model* \( q_\phi(z | x) \) (also called the *recognition model*, or *encoder*) with variational parameters \( \phi \). The goal thus becomes to optimize the variational parameters \( \phi \) such that

\[
q_\phi(z | x) \approx p_\theta(z | x).
\]

Note that our variational distribution \( q_\phi(z | x) \) here is a conditional distribution on \( x \), meaning that we allow the latent variables to depend on the observed variables \( x \). Importantly, this allows us to use a functional representation for \( q_\phi(z | x) \) where the variational parameters \( \phi \) are shared across datapoints \( x \). This results in so-called *amortized variational inference* (Gershman and Goodman, 2014), where we amortize the cost of inference across datapoints, which is realized by globally sharing (amortizing) the variational parameters \( \phi \) across datapoints. Note that this is in contrast to traditional variational inference where the variational
parameters are not shared, but instead separately optimized per datapoint. Amortized inference thus avoids a per-datapoint optimization loop, which allows scalability to large datasets.

Apart from efficiency, another central advantage of amortized inference is that the model learns how to perform inference, such that we can perform inference even for new unseen datapoints \( x \) based on the inference model trained on some training dataset. This is not possible with traditional approaches, where we would have to recompute the inference.

Note, however, that this gain in efficiency comes at a cost in flexibility, as tying parameters across datapoints limits the capacity / representational power of our variational family \( Q \). Non-amortized inference is strictly more expressive in that it allows us to independently optimize the parameters corresponding to each datapoint. In contrast, sharing the variational parameters \( \varphi \) across datapoints effectively introduces constraints for the optimization problem.

In a VAE, we parameterize the inference model \( q_\varphi(z|x) \) using a deep neural network (resulting in an inference network or recognition network), such that the variational parameters \( \varphi \) now include the weights and biases of the network. A popular choice is a simple factorized Gaussian encoder (i.e., a mean-field approximation with Gaussian potentials), i.e.,

\[
q_\varphi(z|x) = \prod_i q_\varphi(z_i|x) = \prod_i N(z_i; \mu_i, \sigma_i^2) = N(z; \mu, \text{diag}(\sigma^2)),
\]

where \((\mu, \log \sigma) = \text{EncoderNeuralNet}_\varphi(x)\).

Note that the resulting VAE involves both a mean-field assumption and an amortized parametrization. This might seem counter-intuitive at first, but while mean-field describes our assumption about the factorization structure of the variational posterior, amortization describes our assumption about how the variational parameters are parameterized (e.g., via a neural network). Thus, while the marginal distribution \( q_\varphi(z|x) \) remains a factorized Gaussian, amortization introduces implicit dependencies between the dimensions of \( z \), yielding an expressiveness similar to that of the generative network of a DLVM described earlier.

The VAE ELBO

It remains to be discussed how to learn the parameters \( \theta \) and \( \varphi \) of the generative and inference networks, respectively, given some dataset \( D = \{x_n\}_{n=1}^N \) of \( N \) inputs drawn i.i.d. from some underlying data distribution \( p(x) \). As a VAE implements variational inference,
the optimization objective is the ELBO,

\[
\mathcal{L}_{\theta, \phi}(D) = \sum_{x \in D} \mathcal{L}_{\theta, \phi}(x),
\]

(2.86)

where \( \mathcal{L}_{\theta, \phi}(x) = \mathbb{E}_{q(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x) \| p(z)), \quad x \in D. \)

(2.87)

Since maximizing the ELBO approximately maximizes \( \log p(D|\theta) \), this training procedure can be viewed as approximate maximum likelihood inference. In practice, \( \mathcal{L}_{\theta, \phi}(x) \) in Eq. (2.87) is maximized by mini-batch stochastic gradient-based optimization using low-variance, unbiased, stochastic Monte Carlo estimators of \( \nabla \mathcal{L}_{\theta, \phi} \) obtained via the reparametrization trick, to be detailed further below.

Observe that in contrast to the ordinary VI ELBO in Eq. (2.72), the VAE ELBO in Eq. (2.87) includes the amortized posterior \( q_\phi(z|x) \), inducing an interpretation as an autoencoder (hence the name variational autoencoder). In particular, we first encode an input \( x \) to produce a latent vector \( z \) via the encoder \( q_\phi(z|x) \), and \( z \) is then fed into the decoder \( p_\theta(x|z) \) to (approximately) recover the original input \( x \). The first term in Eq. (2.87) can then be interpreted as the reconstruction loss of input \( x \) when fed through the autoencoder, thus encouraging the decoder to learn to reconstruct the data. In particular, given some input \( x \), we want the (log) probability of that particular \( x \) to be as large as possible when first encoding \( x \) into \( z \), i.e., \( z \sim q_\phi(z|x) \), and then decoding \( z \) back into some \( \hat{x} \), i.e., \( \hat{x} \sim p_\theta(x|z) \). When using a Gaussian decoder, this term results in the squared error \( \|x - \hat{x}\|_2^2 \) between the original \( x \) and the reconstructed \( \hat{x} \). Since the encoding is stochastic, we want the expected reconstruction error to be minimal under all potential encodings \( z \sim q_\phi(z|x) \), hence the expectation of the decoding distribution w.r.t. the encoding distribution.

### Stochastic Gradient-Based Optimization

Importantly, the ELBO in Eq. (2.87) allows us in principle to jointly optimize w.r.t. both \( \phi \) and \( \theta \) using stochastic gradient descent (SGD). However, the individual-datapoint ELBO and its gradient \( \nabla_{\theta, \phi} \mathcal{L}_{\theta, \phi}(x) \) are in general intractable. Fortunately, we can derive unbiased estimators \( \nabla_{\theta, \phi} \mathcal{L}_{\theta, \phi}(x) \) of the ELBO which can be used instead to perform mini-batch SGD. In particular, given a mini-batch \( \mathcal{M} \subset D \) of the data, an unbiased stochastic estimate \( \nabla_{\theta, \phi} \mathcal{L}_{\theta, \phi}(\mathcal{M}) \) of the gradient of the ELBO can be obtained as

\[
\nabla_{\theta, \phi} \mathcal{L}_{\theta, \phi}(D) \approx \nabla_{\theta, \phi} \mathcal{L}_{\theta, \phi}(\mathcal{M}) = \frac{|D|}{|\mathcal{M}|} \sum_{x \in \mathcal{M}} \nabla_{\theta, \phi} \mathcal{L}_{\theta, \phi}(x).
\]

(2.88)
Unbiased gradients $\nabla_{\theta} \mathcal{L}_{\theta, \phi}(x)$ w.r.t. the generative model parameters $\theta$ can simply be obtained as

$$
\nabla_{\theta} \mathcal{L}_{\theta, \phi}(x) = \nabla_{\theta} \mathbb{E}_{q_{\phi}(z|x)}[\log p_{\theta}(x, z) - \log q_{\phi}(z|\theta)]
$$

$$
= \mathbb{E}_{q_{\phi}(z|x)}[\nabla_{\theta}(\log p_{\theta}(x, z) - \log q_{\phi}(z|\theta))]
$$

$$
\simeq \nabla_{\theta}(\log p_{\theta}(x, z) - \log q_{\phi}(z|\theta))
$$

$$
= \nabla_{\theta} \log p_{\theta}(x, z) \tag{2.89}
$$

since $\nabla_{\theta} \log q_{\phi}(z|x) = 0$, where we use a Monte Carlo estimator with a single sample $z \sim q_{\phi}(z|x)$.

Unfortunately, it is more difficult to obtain unbiased gradients w.r.t. the variational parameters $\phi$, as the expectation in the ELBO is w.r.t. $q_{\phi}(z|x)$, which depends on $\phi$, such that in general,

$$
\nabla_{\phi} \mathcal{L}_{\theta, \phi}(x) = \nabla_{\phi} \mathbb{E}_{q_{\phi}(z|x)}[\log p_{\theta}(x, z) - \log q_{\phi}(z|x)] \tag{2.90}
$$

$$
\neq \mathbb{E}_{q_{\phi}(z|x)}[\nabla_{\phi}(\log p_{\theta}(x, z) - \log q_{\phi}(z|x))]. \tag{2.91}
$$

The Reparameterization Trick

For continuous latent variables $z$, the gradient $\nabla_{\phi} \mathcal{L}_{\theta, \phi}(x)$ of the ELBO w.r.t. $\phi$ can be straightforwardly computed through a change of variables that is now widely known as the reparameterization trick (Kingma and Welling, 2014; Rezende et al., 2014).

First, observe that in many cases, we can express the random variable $z \sim q_{\phi}(z|x)$ as some deterministic, differentiable and invertible transformation $g$ of another random variable $\epsilon$ given $z$ and $\phi$, i.e.,

$$
z = g(\epsilon, \phi, x), \tag{2.92}
$$

where the distribution of $\epsilon$ is independent of $x$ or $\phi$. Using this change of variables, we thus 'externalize' the randomness in $z$ into $\epsilon$, such that we can now reformulate expectations over $z$ in terms of $\epsilon$, i.e.,

$$
\mathbb{E}_{q_{\phi}(z|x)}[f(z)] = \mathbb{E}_{p(\epsilon)}[f(g(\epsilon, \phi, x))], \quad z = g(\epsilon, \phi, x). \tag{2.93}
$$

As in this reformulation, the expectation is w.r.t. a distribution $p(\epsilon)$ independent of the parameters $\phi$, we can now commute the expectation and gradient operators to yield the
Monte Carlo estimator

\[ \nabla_{\phi} \mathbb{E}_{q_\phi(z|x)}[f(z)] = \nabla_{\phi} \mathbb{E}_{p(\epsilon)}[f(z)] = \mathbb{E}_{p(\epsilon)}[\nabla_{\phi} f(z)] \simeq \nabla_{\phi} f(z), \tag{2.94} \]

where \( z = g(\epsilon, \phi, x) \) with random noise sample \( \epsilon \sim p(\epsilon) \).

Using the reparameterization trick, we can now form a Monte Carlo estimator of \( \nabla_{\phi} \mathcal{L}_{\theta, \phi}(x) \) as

\[
\nabla_{\phi} \mathcal{L}_{\theta, \phi}(x) = \nabla_{\phi} \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x, z) - \log q_\phi(z|x)]
= \nabla_{\phi} \mathbb{E}_{p(\epsilon)}[\log p_\theta(x, z) - \log q_\phi(z|x)]
= \mathbb{E}_{p(\epsilon)}[\nabla_{\phi}(\log p_\theta(x, z) - \log q_\phi(z|x))]
\simeq \nabla_{\phi}(\log p_\theta(x, z) - \log q_\phi(z|x))
= -\nabla_{\phi} \log q_\phi(z|x) \tag{2.95}
\]

since \( \nabla_{\phi} \log p_\theta(x, z) = 0 \), and where \( z = g(\epsilon, \phi, x) \) with random noise sample \( \epsilon \sim p(\epsilon) \).

As an example, for the factorized Gaussian encoder as defined in Eq. (2.84), we obtain the reparametrization

\[ z = \mu + \sigma \odot \epsilon, \tag{2.96} \]

\[ (\mu, \log \sigma) = \text{EncoderNeuralNet}(x), \tag{2.97} \]

\[ \epsilon \sim \mathcal{N}(0, I), \tag{2.98} \]

where \( \odot \) denotes the element-wise product.

**Estimation of the Log-likelihood**

After having trained a VAE by learning the parameters \( \phi \) and \( \theta \), we are often interested in estimating the probability \( p_\theta(x) \) of some given input \( x \) under the generative model (e.g., to evaluate if our model was successfully trained to represent the data distribution well).

At first sight, one might attempt to straightforwardly approximate the log-likelihood \( p_\theta(x) \) using Monte Carlo by first sampling latent vectors \( z_k \) from the prior \( p(z) \), and then computing

\[ p_\theta(x) = \int p(z)p_\theta(x|z)dz \simeq \frac{1}{K} \sum_{k=1}^{K} p_\theta(x|z_k). \tag{2.99} \]

Unfortunately, in high-dimensional spaces, \( p_\theta(x|z) \) will be almost zero for most \( z \), such that we will need a very large number of samples \( K \) to obtain an accurate estimate of \( p_\theta(x) \) (Doersch, 2016).
The key idea is to observe that in a VAE, we can obtain latent vectors \( z \) that are likely to have produced \( x \) by sampling from the encoding distribution \( q(\theta, \varphi)(z|x) \). Rezende et al. (2014) thus propose to use importance sampling w.r.t. the variational posterior \( q(\theta, \varphi)(z|x) \) to estimate the likelihood as

\[
p_{\theta, \varphi}(x) = \mathbb{E}_{q(\varphi)(z|x)} \left[ \frac{p(x|z)p(z)}{q(\varphi)(z|x)} \right] \approx \frac{1}{K} \sum_{k=1}^{K} \frac{p_{\theta}(x|z_k)p(z_k)}{q(\varphi)(z_k|x)}, \quad z_k \sim q(\varphi)(z|x). \tag{2.100}
\]

Note that the likelihood \( p_{\theta, \varphi}(x) \) in Eq. (2.100) is parameterized by both \( \theta \) and \( \varphi \), to make explicit the dependence on the parameters \( \varphi \) of the proposal distribution \( q(\varphi)(z|x) \).

Finally, the importance-weighted autoencoder (IWAE) (Burda et al., 2015) uses the likelihood estimate in Eq. (2.100) as the training objective, which is shown to provide a tighter lower bound to the marginal likelihood than the ELBO, for \( K > 1 \).

## 2.6 Applications of Probabilistic Deep Learning

Now that we have introduced the fundamentals of probabilistic deep learning, this section motivates and describes the various application domains of probabilistic deep learning considered in this thesis, and outlines our contributions to address each problem. First, we consider out-of-distribution detection, which aims to identify whether a new data point was drawn from a data distribution different to the training data distribution (Section 2.6.1). Second, we describe data-efficient optimization, where the goal is to find the optimum of a costly objective function with as few function evaluations as possible (Section 2.6.2). Third, we introduce the task of neural network calibration, where we wish to mitigate overconfident model predictions on previously unseen data classes (Section 2.6.3). Lastly, we cover continual deep learning, where we seek to enable models to sequentially learn on streams of data without forgetting past information (Section 2.6.4).

### 2.6.1 Out-of-Distribution Detection

**Motivation & Problem Statement**

**Outlier Detection in Input Space** While deep neural networks (DNNs) have successfully tackled complex real-world problems in various domains including vision, speech, and language (LeCun et al., 2015), they still face significant limitations that make them unfit for safety-critical applications (Amodei et al., 2016). One well-known shortcoming of DNNs is when faced with test data points coming from a different distribution than the data the network saw during training, the DNN will not only output incorrect predictions, but it will do...
so with high confidence (Nguyen et al., 2015). The lack of robustness of DNNs to such out-of-distribution (OoD) inputs (or outliers/anomalies) was recently addressed by various methods to detect OoD inputs in the context of prediction tasks (typically classification) (Hendrycks and Dietterich, 2019a; Hendrycks and Gimpel, 2017; Liang et al., 2018). When we are only given input data, one simple and seemingly sensible approach to detect a potential OoD input $x_*$ is to train a likelihood-based deep generative model (DGM; e.g., a VAE, auto-regressive DGM, or flow-based DGM) by (approximately) maximizing the probability $p(D|\theta)$ of the training data $D$ under the model parameters $\theta$, and to then estimate the density $p(x_*|\theta)$ of $x_*$ under the generative model $\theta$ (Bishop, 1994). If $p(x_*|\theta)$ is large, then $x_*$ is likely in-distribution, and OoD otherwise. However, recent works have shown that this likelihood-based approach does not work in general, as DGMs sometimes assign higher density to OoD data than to in-distribution data (Nalisnick et al., 2019). While some papers developed more effective scores that correct the likelihood (Choi and Jang, 2018; Nalisnick et al., 2019; Ren et al., 2019), we argue and show that OoD detection methods fundamentally based on the unreliable likelihood estimates by DGMs are not robust.

**Outlier Detection in Latent Space**  In a distinct line of research, recent works have tackled the challenge of optimizing a costly-to-evaluate black-box function $f: \mathcal{X} \rightarrow \mathbb{R}, f(x) = y$ over a high-dimensional, richly structured input domain $\mathcal{X}$ (e.g., graphs, images). Given data $D = \{(x_n, y_n)\}_{n=1}^N$, these methods jointly train a VAE on inputs $x$ and a predictive model $g: \mathcal{Z} \rightarrow \mathbb{R}, g(z) = y$ mapping from latent codes $z$ to targets $y$, to then perform the optimization w.r.t. $y$ in the low-dimensional, continuous latent space $\mathcal{Z}$ instead of in input space $\mathcal{X}$ (Gómez-Bombarelli et al., 2018). While these methods have achieved strong results in domains including automatic chemical design and automatic machine learning (Gómez-Bombarelli et al., 2018; Lu et al., 2018; Luo et al., 2018; Tripp et al., 2020), their practical effectiveness is limited by their ability to handle the following trade-off: They need to find inputs $x$ that both have a high target value $y$ and are sufficiently novel (i.e., not too close to training inputs $D$), and at the same time ensure that the optimization w.r.t. $y$ does not progress into regions of the latent space $\mathcal{Z}$ too far away from the training data, which might yield latent points $z$ that decode to semantically meaningless or syntactically invalid inputs $x$ (Kusner et al., 2017). The required ability to quantify the novelty of latents $z$ (i.e., the semantic/syntactic distance to $D$) directly corresponds to the ability to effectively detect outliers in latent space $\mathcal{Z}$. 
Our Contributions

In Chapter 3, we propose a novel unsupervised, probabilistic method to simultaneously tackle the challenge of detecting outliers $x_*$ in input space $\mathcal{X}$ as well as outliers $z_*$ in latent space $\mathcal{Z}$. To this end, we take an information-theoretic perspective on OoD detection, and propose to use the (expected) informativeness of an input $x_*$ / latent $z_*$ as a proxy for whether $x_*$ / $z_*$ is OoD or not. To quantify this informativeness, we leverage probabilistic inference methods to maintain a posterior distribution over the parameters of a DGM, in particular of a variational autoencoder (VAE) (Kingma and Welling, 2014; Rezende et al., 2014). This results in a Bayesian VAE (BVAE) model, where instead of fitting a point estimate of the decoder parameters via maximum likelihood, we estimate their posterior using samples generated via stochastic gradient Markov chain Monte Carlo (MCMC). The informativeness of an unobserved $x_*$ / $z_*$ is then quantified by measuring the (expected) change in the posterior over model parameters after having observed $x_*$ / $z_*$, revealing an intriguing connection to information-theoretic active learning (MacKay, 1992c).

2.6.2 Data-Efficient Optimization

Motivation & Problem Statement

Model-Based Sample-Efficient Black-Box Optimization Many important problems in science and engineering can be formulated as optimizing an objective function $f: \mathcal{X} \mapsto \mathbb{R}$ over an input space $\mathcal{X}$. Solving such problems becomes particularly challenging when 1) the input space $\mathcal{X}$ is high-dimensional (i.e. 100+ effective dimensions) and/or structured (i.e., discrete spaces, or non-Euclidean spaces such as graphs, sequences, and sets) and 2) the objective function $f(x)$ is black-box (i.e., no known analytic form or derivative information available) and is expensive to evaluate (e.g., in terms of time or energy cost). Unfortunately, many real-world problems of practical interest have these characteristics. A notable example is drug design, which has a graph-structured input space, and is evaluated using expensive wet-lab experiments or time-consuming simulations. The sample-efficient optimization problem seeks to optimize $f$ over $\mathcal{X}$, evaluating $f$ as few times as possible, producing in the process a sequence of evaluated points $D_M = \{x_n, f(x_n)\}_{n=1}^M$ with $M$ function evaluations. Recently, machine learning has shown promising results in many problems that can be framed as optimization, such as conditional image (Nguyen et al., 2017; Van den Oord et al., 2016) and text (Otter et al., 2020) generation, molecular and materials design (Elton et al., 2019; Sanchez-Lengeling and Aspuru-Guzik, 2018), and neural architecture search (Elsken et al., 2019). Many existing methods employ model-based optimization, which seeks to approximate $f$ by constructing an objective model $h_\mathcal{X}: \mathcal{X} \mapsto \mathbb{R}$. $h_\mathcal{X}$ is then optimized as a
surrogate for $f$. Although this is a widely used optimization technique, it can be difficult to apply in high-dimensional spaces (which may have many local minima) or in structured spaces (where any kind of discreteness precludes the use of gradient-based optimizers). Thus, despite these successes, using machine learning on structured input spaces and with limited data is still an open research area, making the use of machine learning infeasible for many practical applications.

**Latent Space Optimization (LSO)** One promising approach which tackles both of the aforementioned challenges is a two-stage procedure that has emerged over the past few years, which we will refer to as latent space optimization (LSO) (Gómez-Bombarelli et al., 2018; Kusner et al., 2017; Lu et al., 2018; Luo et al., 2018; Nguyen et al., 2016). In the first stage, a (deep) generative model $g: \mathcal{Z} \rightarrow \mathcal{X}$ is trained to map tensors in a low-dimensional continuous space $\mathcal{Z}$ onto the data manifold in input space $\mathcal{X}$, effectively constructing a low-dimensional and continuous analog of the optimization problem. In the second stage, the objective function $f$ is optimized using model-based optimization over this learned latent space $\mathcal{Z}$. To this end, a latent objective model $h: \mathcal{Z} \rightarrow \mathbb{R}$ is constructed to approximate $f$ at the output of $g$, i.e. such that $f(g(z)) \approx h(z)$, $\forall z \in \mathcal{Z}$. If $\mathcal{Z}$ is chosen to be a low-dimensional, continuous space such as $\mathbb{R}^Z$, the aforementioned difficulties with model-based optimization can be avoided, effectively turning a discrete optimization problem into a continuous one. To realize LSO, $g$ can be chosen to be a state-of-the-art deep generative model (DGM), such as a variational autoencoder (VAE) (Kingma and Welling, 2014; Rezende et al., 2014) or a generative adversarial network (GAN) (Goodfellow et al., 2014), which have been shown to be capable of learning vector representations of many types of high-dimensional, structured data (Bowman et al., 2016; De Cao and Kipf, 2018; Simonovsky and Komodakis, 2018; Wang et al., 2018a). Furthermore, $h$ can be chosen to be a flexible probabilistic model such as a Gaussian process (Rasmussen and Williams, 2005), allowing sample-efficient Bayesian optimization to be performed (Brochu et al., 2010; Shahriari et al., 2015). $h$ can be trained by using an approximate inverse to $g$, $q: \mathcal{X} \rightarrow \mathcal{Z}$, to find a corresponding latent point $z_n$ for each data point $x_n$. Despite many successful applications in a variety of fields including chemical design (Dai et al., 2018; Gómez-Bombarelli et al., 2018; Jin et al., 2018a; Kusner et al., 2017) and automatic machine learning (Lu et al., 2018; Luo et al., 2018), LSO is primarily applied in a post-hoc manner using a pre-trained, general purpose generative model rather than one trained specifically for the explicit purpose of downstream optimization. Put differently, the training of the generative model is effectively decoupled from the optimization task.
Our Contributions

In Chapter 4, we identify and examine two types of decoupling in latent space optimization. We argue that they make optimization unnecessarily difficult and fundamentally prevent LSO from finding solutions that lie far from the training data. Motivated by this, we propose weighting of the data distribution and periodic retraining of the generative model to effectively resolve this decoupling. We argue that these two modifications are highly complementary, fundamentally transforming LSO from a local optimizer into an efficient global optimizer capable of recursive self-improvement.

2.6.3 Neural Network Calibration

Motivation & Problem Statement

A critical shortcoming of deep neural networks (NNs) is that they tend to be poorly calibrated and overconfident in their predictions, especially when there is a shift between the train and test data distributions (Guo et al., 2017; Nguyen et al., 2015). To reliably inform decision making, NNs need to robustly quantify the uncertainty in their predictions (Bhatt et al., 2021). This is especially important for safety-critical applications such as healthcare or autonomous driving (Amodei et al., 2016; Filos et al., 2019; Fridman et al., 2019).

Bayesian modeling presents a principled way to capture uncertainty via the posterior distribution over model parameters. In particular, the posterior $p(\theta | D)$ captures model uncertainty, i.e., uncertainty about the choice of weights $\theta$ arising due to multiple plausible explanations of the training data (Gal, 2016). Uncertainty in the weights is then translated into uncertainty in the predictions via the posterior predictive $p(y_* | x_*, D)$. Unfortunately, due to their non-linearities, exact posterior inference is intractable in NNs. Despite recent successes in the field of Bayesian deep learning (see Section 2.4.1), existing methods invoke unrealistic assumptions to scale to NNs with large numbers of weights. This severely limits the expressiveness of the inferred posterior and thus deteriorates the quality of the induced uncertainty estimates (Foong et al., 2019a; Fort et al., 2019; Ovadia et al., 2019).

Our Contributions

Perhaps the unrealistic inference approximations of previous works can be avoided. Due to the heavy overparameterization of NNs, their accuracy is well-preserved by a small subnetwork (Cheng et al., 2017). Moreover, doing inference over a low-dimensional subspace of the weights can result in accurate uncertainty quantification (Izmailov et al., 2019). This prompts the following question: Can a full NN’s model uncertainty be well-preserved by a small
In Chapter 5, we demonstrate that the posterior predictive distribution of a full network can be well represented by that of a subnetwork. In particular, we develop a new scalable variant of the Laplace approximation that avoids the intractable computation of the full $D \times D$ posterior covariance matrix $\Sigma$ in Eq. (2.40) by performing inference over only a small subset of the model parameters $\theta$.

### 2.6.4 Continual Deep Learning

**Motivation & Problem Statement**

Continual learning (Parisi et al., 2019) aims for accurate incremental training over a large number of individual tasks/examples. This can potentially reduce the frequency of retraining in deep learning, making algorithms easier to use and deploy, while also reducing their environmental impact (Diethe et al., 2019; Paleyes et al., 2022). The main challenge in continual learning is to remember past knowledge and reuse it to continue to adapt to new data. This can be difficult because the future is unknown and can interfere with past knowledge (Kirkpatrick et al., 2017; Mermillod et al., 2013; Sutton, 1986). Performance, therefore, heavily depends on the strategies used to represent and reuse past knowledge.

Two popular strategies of knowledge reuse are based on weight-regularization and experience replay and have complementary strengths. For example, the well-known Elastic-Weight Consolidation (EWC) (Kirkpatrick et al., 2017), which regularizes the new weight-vector to keep it close to the old one, is compact and requires storing only two vectors, one containing the weights and the other their importance (often the empirical Fisher). A variety of other such regularizers have been proposed (Li and Hoiem, 2017; Nguyen et al., 2018; Schwarz et al., 2018; Zenke et al., 2017), and are usually motivated as treating parameters in a Bayesian fashion. This is very different from experience replay (Robins, 1995; Shin et al., 2017), where past examples are simply added during future training. Memory cost here can be substantial, but it can boost accuracy if the memory represents the past well. Clearly, combining the two approaches can strike a good balance between performance and memory size.

At present, little has been done to find principled ways to combine weight-regularization and experience replay. Some works have used knowledge distillation (Buzzega et al., 2020; Rebuffi et al., 2017) or functional regularization (Pan et al., 2020; Titsias et al., 2020), where predictions evaluated at the examples in memory are regularized. Such approaches are promising, but it is not clear why the specific choices of regularizers and memory work well, and whether there are better choices that lead to further improvements.
Our Contributions

In Chapter 6, we provide a principled approach to combine and improve weight-regularization and experience replay. Our approach is based on a recently proposed principle of adaptation (Khan and Swaroop, 2021), where a prior called the Knowledge-adaptation prior (K-prior) is used to reconstruct the gradients of the past training objective. In the continual learning setting, this requires an accurate reconstruction of the gradients over all the past tasks, and can be used as a guideline to design better priors. Khan and Swaroop (2021) considered only one-task adaptation, and used a simple quadratic regularizer without any experience replay. We extend their method to multiple tasks and use it to combine different regularization and replay methods. Using the principle of gradient reconstruction of the past, we design a prior that combines a weight-regularizer, a functional regularizer, and experience replay. Each piece contributes in a new way to the reduction of a different type of error, and the combination overall gives lower gradient-reconstruction error than each individual component. This leads to consistent improvements on standard benchmarks for multi-task image classification in task-incremental continual learning, such as Split-CIFAR, Split-TinyImageNet, and ImageNet-1000, across various memory budgets from small to large sizes. This approach is principled and can yield better strategies than the current heuristics used in the literature.

2.7 Discussion

This chapter reviewed the background on probabilistic deep learning that underpins the remainder of this thesis. We first introduced neural networks and deep learning, which most models in this thesis will be built upon. We then described how probabilistic modeling and inference can be used to learn model parameters in a principled way. As exact inference is intractable for most interesting models (in particular, neural network models), we surveyed several approaches for approximate Bayesian inference.

We continued by outlining how the concepts introduced in this chapter culminate in two major probabilistic deep learning models: Bayesian neural networks, which require us to infer global latent variables (i.e., the model parameters) in a supervised fashion, and variational autoencoders, which require us to infer local latent variables in an unsupervised fashion. Apart from BNNs and VAEs, there exist other families of probabilistic deep learning models, the discussion of which is beyond the scope of this thesis, e.g., normalizing flows (Papamakarios et al., 2021) and neural processes (Garnelo et al., 2018a,b).

Finally, we introduced the range of real-world application domains that this thesis will address using probabilistic deep learning methods: out-of-distribution detection,
efficient optimization, neural network calibration, and continual deep learning. Note that there exist many further applications of probabilistic deep learning, the discussion of which is beyond the scope of this thesis, e.g., active learning (Gal et al., 2017; MacKay, 1992c), model selection (Immer et al., 2021a; MacKay, 1992b), meta learning (Gordon, 2021), compression (Havasi, 2021), and reinforcement learning (Osband et al., 2016).

In the next chapter, we will develop a method to tackle the first application area, out-of-distribution (OoD) detection, where we leverage a model that combines a VAE with a BNN to perform probabilistic inference over both the local latent variables and the global model parameters.
Chapter 3

Out-of-Distribution Detection via Bayesian Variational Autoencoders

In this chapter, we introduce a new probabilistic deep learning method for tackling the out-of-distribution (OoD) detection problem, both in input space and in latent space (cf. Section 2.6.1). In particular, our contributions in this chapter are structured as follows:

(a) We first provide intuition for our proposed approach that aims to address OoD detection in an information-theoretic way by capturing epistemic uncertainty in a VAE (Section 3.1).

(b) We then formally introduce the proposed Bayesian VAE model in which a posterior distribution is not only inferred over the local latent variables, but additionally also over the global model parameters (Section 3.2).

(c) We proceed to describe how this Bayesian VAE model can be used for information-theoretic OoD detection both in input space and the model’s latent space (Section 3.3).

(d) We review related work on OoD detection using likelihood-based deep generative models (Section 3.4).

(e) We present an empirical study demonstrating the efficacy of our approach on common benchmarks (Section 3.5).

The material in this chapter was previously published in ‘Bayesian Variational Auto-Encoders for Unsupervised Out-of-Distribution Detection’ (Daxberger and Hernández-Lobato, 2019). The research was supervised by José Miguel Hernández-Lobato. I was involved closely with all aspects of the paper, including the theoretical results, the experiments and the writing of the paper.
3.1 Intuition on Information-Theoretic OoD Detection

We take the following information-theoretic perspective on OoD detection. Consider an active learning scenario, where we have computed an estimate of our model parameters $\theta$ based on some observations $D$, and want to add an unobserved input $x_*$ to the training set $D$ to improve our estimate of $\theta$ as much as possible. To this end, in information-theoretic active learning (MacKay, 1992c), it is observed that every potential input $x_*$ contains a certain amount of information about the values of the model parameters. In other words, every data point $x_*$ helps us to some extent in updating our model parameters to find the optimal ones which capture the true underlying data-generating process. Now, we argue that inputs which are uninformative about the model parameters are likely similar to the data points already in the training set $D$, i.e., they are likely in-distribution inputs. In contrast, inputs which are very informative about the model parameters are likely different from everything we have seen so far in the training data $D$, i.e., they are likely OoD inputs. We thus propose to use the informativeness of a datum $x_*$ as a proxy for whether $x_*$ is OoD or not. To quantify this informativeness, information-theoretic active learning approaches leverage probabilistic inference techniques to maintain a posterior distribution $p(\theta | D)$ over model parameters $\theta$ given data $D$ (i.e., instead of fitting a point estimate of $\theta$ via ML or MAP inference). Given this posterior $p(\theta | D)$, the informativeness of an unobserved datum $x_*$ is then quantified by measuring the (expected) change in the posterior after having observed $x_*$, i.e., the change required to update $p(\theta | D)$ to the posterior $p(\theta | D \cup \{x_*\})$.

We follow this idea and propose to address OoD detection using a Bayesian VAE (BVAE) model, where instead of fitting a point estimate of the model parameters via maximum likelihood, we estimate their posterior distribution using samples generated via stochastic gradient MCMC.

3.2 The Bayesian Variational Autoencoder (BVAE)

In this section, we introduce our proposed Bayesian VAE model. We first define the model formulation (Section 3.2.1), before describing how to perform Bayesian inference over the latent variables $z$ (Section 3.2.2), the model parameters $\theta$ (Section 3.2.3), and the variational parameters $\phi$ (Section 3.2.4). We then explain how to estimate the likelihood of the BVAE (Section 3.2.5), and provide pseudocode for training the BVAE (Section 3.2.6). Finally, we review related work on fully Bayesian deep generative models (Section 3.2.7).
3.2 The Bayesian Variational Autoencoder (BVAE)

3.2.1 Model Formulation

In contrast to an ordinary VAE, where we fit the generative model parameters $\boldsymbol{\theta}$ via (approximate) maximum likelihood inference, i.e., $\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} \mathcal{L}(D)_{\theta, \varphi}$, to obtain the likelihood $p(x \mid z, \boldsymbol{\theta}_{\text{ML}})$ (see Section 2.5), we place a prior $p(\boldsymbol{\theta})$ over $\boldsymbol{\theta}$ and estimate its full posterior distribution $p(\boldsymbol{\theta} \mid D) \propto p(D \mid \boldsymbol{\theta})p(\boldsymbol{\theta})$, to induce the generator likelihood

$$p(x \mid z, D) = \int_{\theta} p(x \mid z, \theta)p(\theta \mid D)d\theta = \mathbb{E}_{\theta \mid D}[p(x \mid z, \theta)]. \quad (3.1)$$

Note that most literature on VAEs includes the parameters $\boldsymbol{\theta}$ as a subscript, e.g., $p_{\theta}(x, z)$ (as also introduced in Section 2.5), assuming a fixed point estimate $\boldsymbol{\theta}_{\text{ML}}$ of $\boldsymbol{\theta}$, i.e., $p_{\theta_{\text{ML}}}(x, z) = p(x, z \mid \theta = \boldsymbol{\theta}_{\text{ML}})$. We instead here use the notation $p(x, z \mid \theta)$ to make explicit that we assume $\theta$ to be a random variable governed by some distribution. This induces the marginal likelihood

$$p(x \mid D) = \mathbb{E}_{p(z)}[p(x \mid z, D)] \overset{\text{Eq. } (3.1)}{=} \int_{z} \int_{\theta} p(x \mid z, \theta)p(\theta \mid D)p(z)d\theta dz = \mathbb{E}_{p(\theta \mid D)}[p(x \mid \theta)], \quad (3.2)$$

which marginalizes over both the local latent variables $z$ and the global latent variables $\theta$. The generative process defined in Eq. (3.2) first draws a latent vector $z \sim p(z)$ from its prior and a decoder parameterization $\theta \sim p(\theta \mid D)$ from its posterior, and then generates $x \sim p(x \mid z, \theta)$ by passing both through the likelihood. Training this model now involves performing Bayesian inference over both $z$ and $\theta$, resulting in the posterior $p(z \mid x, D)$ over $z$ and the posterior $p(\theta \mid D) \propto p(D \mid \theta)p(\theta)$ over $\theta$. Since both posteriors are intractable for the model we consider, we will now discuss how to approximate inference over $z$ and $\theta$, respectively.

3.2.2 Inference Over the Latent Variables $z$

To estimate the posterior $p(z \mid x, D)$ over $z$, we follow an ordinary VAE and resort to amortized variational inference via a recognition network $q(z \mid x, \varphi)$ with variational parameters $\varphi$, using the ELBO $\mathcal{L}_{\theta, \varphi}(x) \leq \log p(x \mid \theta)$ in Eq. (2.87). We learn the encoder parameters $\varphi$ via approximate maximum likelihood inference, using a low-variance, unbiased Monte Carlo estimator of the stochastic gradient $\nabla_{\varphi}\mathcal{L}_{\theta, \varphi}(x)$, i.e.,

$$\nabla_{\varphi}\mathcal{L}_{\theta, \varphi}(x) \simeq \frac{1}{L} \sum_{l=1}^{L} \nabla_{\varphi} \log [p(x \mid z_l, \theta)] - \nabla_{\varphi} D_{\text{KL}}(q(z \mid x, \varphi) \| p(z)), \quad z_l \sim q(z \mid x, \varphi), \quad (3.3)$$

where the posterior samples $z_l \sim q(z \mid x, \varphi)$ are generated using the reparametrization trick.
3.2.3 Inference Over the Model Parameters $\theta$

To generate posterior samples $\theta \sim p(\theta | D)$ of decoder parameters (e.g., as required to approximate the gradient $\nabla_{\theta} L_{\phi}(x)$ in Eq. (3.3)), we propose to use SGHMC (see Section 2.3.1). However, the gradient of the energy function $\nabla_{\theta} U(\theta, M)$ in Eq. (2.35) used for simulating the Hamiltonian dynamics requires us to evaluate the log-likelihood $\log p(x | \theta)$, which is intractable in a VAE and thus also in our model. To alleviate this issue, we approximate the log-likelihood appearing in $\nabla_{\theta} U(\theta, M)$ by the ELBO $L_{\theta, \phi}(x)$ of an ordinary VAE in Eq. (2.87).

Given a set $\Theta = \{\theta_{m}\}_{m=1}^{M}$ of posterior samples $\theta_{m} \sim p(\theta | D)$, we can more intuitively think of working with a finite mixture/ensemble of decoders/generative models,

$$p(x | z, D) \overset{\text{Eq. (3.1)}}{=} \mathbb{E}_{p(\theta | D)}[p(x | z, \theta)] \approx \frac{1}{M} \sum_{\theta \in \Theta} p(x | z, \theta), \quad (3.4)$$

whose components are defined by the models parameterized by the samples in $\Theta$.

3.2.4 Inference Over the Variational Parameters $\varphi$

Recall that the goal of amortized variational inference is to learn how to do posterior inference, by finding the optimal parameters $\varphi_{\text{ML}} = \arg \max_{\varphi} \mathcal{L}(D)_{\theta, \varphi}$ (cf. Eq. (2.87)) of an inference network $i : \mathcal{X} \rightarrow \Psi : i_{\varphi}(x) = \psi$ mapping inputs $x$ to parameters $\psi$ of the variational posterior $q_{\psi}(z) = q(z | x, \varphi)$ over $z$. However, one fundamental shortcoming of fitting an inference network via maximum likelihood inference is that the model $q(z | x, \varphi_{\text{ML}})$ will not generalize to OoD inputs, but instead produce confidently wrong posterior inferences for OoD inputs. To alleviate this issue, we refrain from fitting a point estimate $\varphi_{\text{ML}}$, and instead consider a Bayesian treatment of the variational parameters $\varphi$ (i.e., in addition to the Bayesian treatment of the decoder parameters $\theta$ discussed earlier). While this might seem strange conceptually, it allows us to quantify our epistemic uncertainty in the amortized inference of $z$, intuitively increasing the flexibility of the recognition network. We thus also place a prior $p(\varphi)$ over $\varphi$ and infer the posterior $p(\varphi | D) \propto p(D | \varphi) p(\varphi)$, yielding the amortized posterior

$$q(z | x, D) = \int_{\varphi} q(z | x, \varphi) p(\varphi | D) d\varphi = \mathbb{E}_{p(\varphi | D)}[q(z | x, \varphi)] \approx \frac{1}{M} \sum_{j=1}^{M} q(z | x, \varphi_{j}), \quad (3.5)$$

where $\varphi_{j} \sim p(\varphi | D)$. We also use SGHMC to generate the posterior samples $\varphi_{j} \sim p(\varphi | D)$, again approximating the log-likelihood $\log p(x | \varphi, D)$ in $\nabla_{\varphi} U(\varphi, M)$ (cf. Eq. (2.35)) by the ELBO $L_{\theta, \varphi}(x)$ in Eq. (2.87).
Given a set $\Phi = \{\varphi_j\}_{j=1}^M$ of posterior samples $\varphi_j \sim p(\varphi | D)$, we can again more intuitively think of Eq. (3.5) as defining a finite mixture/ensemble of inference networks,

$$q(z | x, D) \equiv \mathbb{E}_{p(\varphi | D)}[q(z | x, \varphi)] \simeq \frac{1}{M} \sum_{\varphi \in \Phi} q(z | x, \varphi),$$  

(3.6)

whose components are defined by the models parameterized by the samples in $\Phi$.

### Inference Over $\Theta$ Revisited

When doing inference over both $\varphi$ and $\Theta$, we need to slightly adapt the SGHMC sampling procedure for the decoder parameters $\Theta \sim p(\Theta | D)$ described in Section 3.2.3. In particular, instead of using the ELBO $L_{Q, \phi}(x)$ of an ordinary VAE in Eq. (2.87) (which is based on the recognition network $q(z | x, \varphi)$ and thus depends on both $\Theta$ and $\varphi$), we now need to use the ELBO

$$L_{\Theta}(x) = \mathbb{E}_{q(z | x, D)}[\log p(x | z, \Theta)] - D_{KL}(q(z | x, D) || p(z)) \leq \log p(x | \Theta, D), \quad (3.7)$$

which depends on $\Theta$ only, as it is based on the encoder mixture $q(z | x, D)$ in Eq. (3.5) and thus integrates the variational parameters $\varphi$ over their posterior $p(\varphi | D)$. We then obtain an approximation to the stochastic gradient $\nabla_\Theta U(\Theta, M)$ in Eq. (2.35) by using a Monte Carlo approximation of the gradient $\nabla_\Theta L_{\Theta}(x)$ of the ELBO in Eq. (3.7), i.e.,

$$\nabla_\Theta L_{\Theta}(x) = \mathbb{E}_{q(z | x, D)}[\nabla_\Theta \log p(x | z, \Theta)] \simeq \frac{1}{L} \sum_{l=1}^L \nabla_\Theta \log p(x | z_l, \Theta), \quad z_l \sim q(z | x, D). \quad (3.8)$$

The $z_l \sim q(z | x, D)$ are sampled from the encoder mixture in Eq. (3.5) by first sampling the index of the mixture component uniformly at random, i.e., $j \sim \mathcal{U}[1, M]$ (due to the mixture weights all being $1/M$), and then sampling the latent vector $z_l$ from the $j$-th mixture component, i.e., $z_l \sim q(z | x, \varphi_j)$.

### 3.2.5 Likelihood Estimation

We now discuss how to estimate the likelihood $p(x | \Theta)$ of the BVAE given an input $x$. We consider two approaches based on importance sampling, with different proposal distributions for sampling the latent codes $z_k$ depending on the assumed relationship between the samples $\Theta$ of decoder parameters and the samples $\Phi$ of encoder parameters:

1. Firstly, one may treat the samples in $\Theta$ and $\Phi$ as being coupled as $(\Theta_m, \varphi_m) \in \{(\Theta_m, \varphi_m)\}_{m=1}^M$, where each pair $(\Theta_m, \varphi_m)$ effectively defines a separate VAE. This is
motivated by the fact that in practice, we for simplicity do indeed take samples of full VAEs, by simultaneously sampling from \( p(\theta | D) \) and \( p(\varphi | D) \), as outlined in Algorithm 2 in Section 3.2.6. Since such a coupled sample \((\theta_m, \varphi_m)\) defines a VAE, the likelihood can then simply be computed as in a regular VAE, i.e., \( p(x | \theta_m) = p(x | \theta_m, \varphi_m) \) as defined in Eq. (2.100), using the proposal distribution \( q(z | x, \varphi_m) \).

2. Secondly, one may decouple the samples in \( \Theta \) and \( \Phi \) and treat them as being independent of each other. In particular, for a given decoder sample \( \theta_m \), instead of using the corresponding sample \( \varphi_m \) as the encoder, we may view the encoder as the mixture \( q(z | x, D) \simeq \frac{1}{M} \sum_{\varphi \in \Phi} q(z_k | x, \varphi) \) in Eq. (3.6) defined by all samples in \( \Phi \). The likelihood is then estimated as \( p(x | \theta) = p(x | \theta, D) \) with

\[
p(x | \theta, D) = \mathbb{E}_{q(z|x,D)} \left[ \frac{p(x | z, \theta) p(z)}{q(z|x,D)} \right] \overset{\text{Eq. (3.6)}}{=} \frac{1}{K} \sum_{k=1}^{K} \frac{1}{M} \sum_{\varphi \in \Phi} p(x | z_k, \theta) p(z_k) q(z_k | x, \varphi), \tag{3.9}
\]

where the latents \( z_k \sim q(z | x, D) \) are now drawn from the mixture proposal distribution \( q(z | x, D) \) instead of from \( q(z | x, \varphi) \). Since \( q(z | x, D) \) marginalizes over encoder parameters \( \varphi \), the likelihood in Eq. (3.9) only depends on \( \theta \).

### 3.2.6 Pseudocode of BVAE Training Procedure

Algorithm 2 and Algorithm 3 show pseudocode for two slightly different implementations of the overall Bayesian VAE (BVAE) training procedure, which both produce sets \( \Phi = \{ \varphi_B, \varphi_{B+D}, \varphi_{B+2D}, \ldots, \varphi_T \} \) and \( \Theta = \{ \theta_B, \theta_{B+D}, \theta_{B+2D}, \ldots, \theta_T \} \) of \( M = (T - B)/D + 1 \) sequentially generated posterior samples \( \varphi_t \sim p(\varphi | D) \) and \( \theta_t \sim p(\theta | D) \) that do not fall into the Markov chain’s burn-in phase of \( B \) epochs and are \( D \) epochs apart from each other (to avoid correlation).
3.2 The Bayesian Variational Autoencoder (BVAE)

**Algorithm 2** First Variant of Bayesian VAE Training

**Input:** Dataset \( \mathcal{D} \), generative model \( p(\mathbf{x}, \mathbf{z}, \boldsymbol{\theta}) \), inference model \( q(\mathbf{z} | \mathbf{x}, \varphi) \), burn-in length \( B \), sample distance \( D \), mini-batch size \( |\mathcal{M}| \), number of training epochs \( T \)

Initialize \( \varphi_0 \), \( \boldsymbol{\theta}_0 \) and \( \Phi = \emptyset \), \( \Theta = \emptyset \)

Define number of mini-batches per epoch \( N_b = \frac{|\mathcal{D}|}{|\mathcal{M}|} \)

for \( t = 1, \ldots, T \) do
  Set \( \hat{\varphi}_0 = \varphi_{t-1} \), \( \hat{\boldsymbol{\theta}}_0 = \boldsymbol{\theta}_{t-1} \)
  for \( b = 1, \ldots, N_b \) do
    Sample minibatch \( \mathcal{M} \sim \mathcal{D} \)
    Set \( \Theta_\ast = \{ \hat{\mathbf{\theta}}_{b-1} \} \) if \( t \leq B \) else \( \Theta_\ast = \emptyset \)
    Set \( \Phi_\ast = \{ \hat{\varphi}_{b-1} \} \) if \( t \leq B \) else \( \Phi_\ast = \emptyset \)
    Compute \( \nabla_{\varphi} U(\hat{\varphi}_{b-1}, \Theta_\ast, \mathcal{M}) \), update \( \hat{\varphi}_{b-1} \rightarrow \hat{\varphi}_b \) via SGHMC
    Compute \( \nabla_{\mathbf{\theta}} U(\hat{\mathbf{\theta}}_{b-1}, \Phi_\ast, \mathcal{M}) \), update \( \hat{\mathbf{\theta}}_{b-1} \rightarrow \hat{\mathbf{\theta}}_b \) via SGHMC
  end for
  if \( t \geq B \) and \( (t - B) \mod D = 0 \) then
    Add \( \Phi = \Phi \cup \{ \varphi_t \} \) and \( \Theta = \Theta \cup \{ \mathbf{\theta}_t \} \)
  end if
end for

**Output:** Posterior samples \( \Phi \) and \( \Theta \)

**Algorithm 3** Second Variant of Bayesian VAE Training

**Input:** Dataset \( \mathcal{D} \), generative model \( p(\mathbf{x}, \mathbf{z}, \boldsymbol{\theta}) \), inference model \( q(\mathbf{z} | \mathbf{x}, \varphi) \), burn-in length \( B \), sample distance \( D \), mini-batch size \( |\mathcal{M}| \), number of training epochs \( T \)

Initialize \( \varphi_0 \), \( \boldsymbol{\theta}_0 \) and \( \Phi = \emptyset \), \( \Theta = \emptyset \)

Define number of mini-batches per epoch \( N_b = \frac{|\mathcal{D}|}{|\mathcal{M}|} \)

for \( t = 1, \ldots, T \) do
  Set \( \hat{\varphi}_0 = \varphi_{t-1} \), \( \hat{\boldsymbol{\theta}}_0 = \boldsymbol{\theta}_{t-1} \)
  for \( b = 1, \ldots, N_b \) do
    Sample minibatch \( \mathcal{M} \sim \mathcal{D} \)
    Compute \( \nabla_{\varphi} U(\hat{\varphi}_{b-1}, \hat{\mathbf{\theta}}_{b-1}, \mathcal{M}) \), update \( \hat{\varphi}_{b-1} \rightarrow \hat{\varphi}_b \) via SGHMC
    Compute \( \nabla_{\mathbf{\theta}} U(\hat{\mathbf{\theta}}_{b-1}, \hat{\mathbf{\theta}}_{b-1}, \mathcal{M}) \), update \( \hat{\mathbf{\theta}}_{b-1} \rightarrow \hat{\mathbf{\theta}}_b \) via SGHMC
  end for
  if \( t \geq B \) and \( (t - B) \mod D = 0 \) then
    Add \( \Phi = \Phi \cup \{ \varphi_t \} \) and \( \Theta = \Theta \cup \{ \mathbf{\theta}_t \} \)
  end if
end for

**Output:** Posterior samples \( \Phi \) and \( \Theta \)
The difference between the two variants in Algorithm 2 and Algorithm 3 is the way we estimate the stochastic gradient in Eq. (2.35) required for the SGHMC updates.

In particular, Algorithm 2 uses all previous samples Θ and Φ for a Monte Carlo approximation of the gradient, i.e., $∇_φ \tilde{U}(\phi_{b-1}, Θ, M)$ and $∇_θ \tilde{U}(\hat{θ}_{b-1}, Φ, M)$ are defined as

$$∇_φ U(φ, M) \approx ∇_φ \tilde{U}(\phi_{b-1}, Θ, M) = -\frac{|D|}{|M|} \sum_{x \in M} ∇_φ \mathcal{L}_{\phi_{b-1}}(x) - ∇_φ \log p(\phi_{b-1}), \tag{3.10}$$

with ELBO gradient $∇_φ \mathcal{L}_{\phi_{b-1}}(x)$ as in Eq. (3.3), and

$$∇_θ U(θ, M) \approx ∇_θ \tilde{U}(\hat{θ}_{b-1}, Φ, M) = -\frac{|D|}{|M|} \sum_{x \in M} ∇_θ \mathcal{L}_{\hat{θ}_{b-1}}(x) - ∇_θ \log p(\hat{θ}_{b-1}), \tag{3.11}$$

with ELBO gradient $∇_θ \mathcal{L}_{\hat{θ}_{b-1}}(x)$ as in Eq. (3.8).

In contrast, Algorithm 3 just uses the parameters $\hat{θ}_{b-1}$ and $\phi_{b-1}$ obtained in the previous training iteration to approximate the gradient, yielding $∇_θ \tilde{U}$ and $∇_φ \tilde{U}$ with

$$\tilde{U}(\phi_{b-1}, \hat{θ}_{b-1}, M) = -\frac{|D|}{|M|} \sum_{x \in M} \mathcal{L}_{\hat{θ}_{b-1}, \phi_{b-1}}(x) - \log p(\phi_{b-1}) - \log p(\hat{θ}_{b-1}), \tag{3.12}$$

with the standard VAE ELBO as in Eq. (2.87), i.e.,

$$\mathcal{L}_{\hat{θ}_{b-1}, \phi_{b-1}}(x) = \mathbb{E}_{q(z|x, \phi_{b-1})}[\log p(x | z, \hat{θ}_{b-1})] - D_{KL}(q(z | x, \phi_{b-1}) || p(z)). \tag{3.13}$$

In contrast to Algorithm 2, Algorithm 3 thus does not make use of the posterior samples Θ and Φ during training. As a result, Algorithm 3 is essentially identical to the training procedure of an ordinary VAE, but using the SGHMC sampler instead of a stochastic gradient-based optimizer for performing the parameter updates.

Note that both Algorithm 2 and Algorithm 3 can in practice be conveniently implemented by exploiting automatic differentiation tools commonly employed by modern deep learning frameworks.

### 3.2.7 Related Work on Fully Bayesian Deep Generative Models

We now review previous approaches that perform Bayesian inference over the global parameters of a deep generative model. While most approaches focus on a VAE model, one paper also considers a generative adversarial network (GAN) (Goodfellow et al., 2014). We note
that to the best of our knowledge, none of these methods has been designed for or applied to the important problem of out-of-distribution detection.

The original VAE paper (Kingma and Welling, 2014) already considered the possibility to perform full variational Bayes over both the latent variables $z$ as well as the generative model parameters $\theta$, and derives the corresponding ELBO and gradient estimators for that (see their Appendix F). However, neither do they describe a motivation or application for additionally performing inference over $q$, nor is this extension evaluated empirically.

In Hernández-Lobato et al. (2016), they consider an importance-weighted autoencoder (IWAE, see Section 2.5) with random neural network parameters, which is achieved by performing variational inference with factorized Gaussian approximate posteriors $q(\phi)$ and $q(\theta)$ over both the encoder parameters $\phi$ and the decoder parameters $\theta$, respectively. Their goal simply is to increase the model flexibility to improve generalization performance, which they show to hold empirically, as compared to a vanilla IWAE with deterministic parameters.

The variational continual learning (VCL) paper (Nguyen et al., 2018) estimates a full Bayesian posterior over the decoder parameters $\theta$ of a VAE via mean-field Gaussian variational inference (Blundell et al., 2015) to enable continual learning in deep generative models. To this end, they describe the full ELBO w.r.t. both the encoder parameters $\phi$ (i.e., parameterizing the approximate posterior $q_\phi(z|x) \approx p(z|x)$ over the latent variables) as well as the approximate posterior $q(\theta) \approx p(\theta|D)$ over the decoder parameters.

Saatci and Wilson (2017) proposes a Bayesian Generative Adversarial Network (GAN), where stochastic gradient MCMC is performed over the discriminator and generator parameters of a GAN. Their goal again is increased expressiveness, which they demonstrate empirically.

Finally, Ice-breaker (Gong et al., 2019) combined amortized VI over the local latent variables $z$ with stochastic gradient MCMC over the global latent variables $\theta$ (i.e., the decoder parameters), resulting in a model they refer to as a Bayesian Deep Latent Gaussian Model (BELGAM). They focus on tackling the ice-start problem (i.e., the challenge of deploying machine learning systems with no or only little training data available) via feature-wise active data acquisition.

### 3.3 Information-Theoretic Out-of-Distribution Detection

We now describe how to detect out-of-distribution inputs with the proposed BV-VAE model based on information-theoretic principles, both in input space (Section 3.3.1) and the model’s latent space (Section 3.3.2). We also describe how to implement our proposed OoD detection criterion in a numerically stable way (Section 3.3.3).
### 3.3.1 Out-of-Distribution Detection in Input Space

Training a BVAE (see Section 3.2.6 for pseudocode) yields the sets \( \Phi = \{ \phi_j \}_{j=1}^M \) and \( \Theta = \{ \theta_m \}_{m=1}^M \) of posterior samples \( \phi_j \sim p(\phi | D) \) and \( \theta_m \sim p(\theta | D) \) of encoder and decoder parameters, respectively. Given these samples, we follow ideas from information-theoretic active learning (MacKay, 1992c) and aim to detect if a certain test input \( x_* \) is OoD based on its informativeness about the model parameters \( \theta \), as measured by the change in the posterior distribution over \( \theta \) after having observed \( x_* \).

In particular, assume that given some data \( D \) and prior \( p(\theta) \), we have inferred the posterior

\[
p(\theta | D) = \frac{p(D | \theta)p(\theta)}{\int_\Theta p(D | \theta)p(\theta) d\theta} = \frac{p(D | \theta)}{p(D)}p(\theta) \tag{3.14}
\]

over \( \theta \). In a sequential Bayesian setting, the posterior \( p(\theta | D) \) then serves as the new prior, which, given a new observation \( x_* \), is updated to the posterior

\[
p(\theta | D \cup \{x_*\}) = \frac{p(x_* | \theta)p(\theta | D)}{\int_\Theta p(x_* | \theta)p(\theta | D) d\theta} = \frac{p(x_* | \theta)}{p(x_* | D)p(\theta | D)}, \tag{3.15}
\]

i.e., by multiplying \( p(\theta | D) \) by the normalized likelihood \( \frac{p(x_* | \theta)}{p(x_* | D)} \) (Bishop, 2006).

The intuition now is as follows: If \( x_* \) is very different from the previous observations in \( D \), then the updated posterior \( p(\theta | D \cup \{x_*\}) \) will likely be different from \( p(\theta | D) \), to capture the atypicality of \( x_* \). In contrast, if \( x_* \) is very similar to the observations in \( D \), then the posterior \( p(\theta | D \cup \{x_*\}) \) will also be similar to the previous one, \( p(\theta | D) \). Thus, to detect whether \( x_* \) is OoD or not, we aim to quantify the change from \( p(\theta | D) \) to \( p(\theta | D \cup \{x_*\}) \), which is captured by the factor \( \frac{p(x_* | \theta)}{p(x_* | D)} \).

Since in our case, the posterior \( p(\theta | D) \) is represented by the finite set of samples \( \Theta \), we find ourselves in a sequential Monte Carlo (or particle filtering) setting (Bishop, 2006), such that for a given parameter/particle \( \theta \) and input \( x_* \), the normalized likelihood \( \frac{p(x_* | \theta)}{p(x_* | D)} \) can be written as

\[
\frac{p(x_* | \theta)}{p(x_* | D)} \underset{\text{Eq. (3.2)}}{=} \frac{p(x_* | \theta)}{\mathbb{E}_{p(\theta | D)}[p(x_* | \theta)]} \approx \frac{p(x_* | \theta)}{\frac{1}{M} \sum_{\theta \in \Theta} p(x_* | \theta)} = M w_\theta, \tag{3.16}
\]

with

\[
w_\theta = \frac{p(x_* | \theta)}{\sum_{\theta \in \Theta} p(x_* | \theta)}, \tag{3.17}
\]

where \( w_\theta \in [0, 1] \) and \( \sum_{\theta \in \Theta} w_\theta = 1 \), and where the likelihood \( p(x_* | \theta) \) can be estimated via importance sampling as described in Section 3.2.5. The scalar \( w_\theta \) defined in Eq. (3.17) can be interpreted as the probability that \( x_* \) was generated from the particle \( \theta \), thus measuring how well \( x_* \) is explained by the model \( \theta \) (and thus how useful \( \theta \) is for describing the updated
3.3 Information-Theoretic Out-of-Distribution Detection

The posterior $p(\theta \mid D \cup \{x_*\})$, relative to the other particles. More formally, the values $[w_\theta]_{\theta \in \Theta}$ correspond to the importance weights of the samples/particles $\theta \in \Theta$ drawn from the proposal distribution $p(\theta \mid D)$ (i.e., the previous posterior) for an importance sampling-based Monte Carlo approximation of an expectation w.r.t. the target distribution $p(\theta \mid D \cup \{x_*\})$ (i.e., the updated posterior after having observed $x_*$), i.e.,

$$
\mathbb{E}_{p(\theta \mid D \cup \{x_*\})}[f(\theta)] \overset{\text{Eq. (3.15)}}{=} \mathbb{E}_{p(x_* \mid \Theta)p(\theta \mid D)}[f(\theta)] \overset{\text{Eq. (3.16)}}{\approx} \mathbb{E}_{p(\theta \mid D)}[Mw_\theta f(\theta)]
$$

(3.18)

$$
\approx \frac{1}{M} \sum_{\theta \in \Theta} Mw_\theta f(\theta) = \sum_{\theta \in \Theta} w_\theta f(\theta),
$$

(3.19)

for some function $f : \Theta \rightarrow \mathbb{R}$. To measure the change in distribution from $p(\theta \mid D)$ to $p(\theta \mid D \cup \{x_*\})$, we can then use the weights $[w_\theta]_{\theta \in \Theta}$ to compute the effective sample size (ESS)

$$
\text{ESS}_\theta(x_*) = \frac{1}{\sum_{\theta \in \Theta} w_\theta^2} \mathbb{E}_{p(\theta \mid D \cup \{x_*\})} \left(\sum_{\theta \in \Theta} p(x_* \mid \theta)\right)^2
$$

(3.20)

where $\text{ESS}_\theta(x_*) \in [1, M]$. The ESS($x_*$) is a widely used measure for the efficiency of the estimator in Eq. (3.19), and measures how many i.i.d. samples drawn from the target posterior $p(\theta \mid D \cup \{x_*\})$ are equivalent to the $M$ samples $\theta \in \Theta$ drawn from the proposal posterior $p(\theta \mid D)$ and weighted according to $w_\theta$. It intuitively quantifies the degree of agreement between the particles $\theta \in \Theta$ as to how probable the input $x_*$ is, inducing the following decision rule: If ESS($x_*$) is large, then $[w_\theta]_{\theta \in \Theta}$ is close to the uniform distribution $[1_M]_{\theta \in \Theta}$ (for which ESS($x_*$) = $M$), meaning that all particles $\theta \in \Theta$ explain $x_*$ equally well and are in agreement as to how probable $x_*$ is. Thus, $x_*$ likely is an in-distribution input. Conversely, if ESS($x_*$) is small, then $[w_\theta]_{\theta \in \Theta}$ contains a few large weights (i.e., corresponding to particles that by chance happen to explain the datum well), with all other weights being very small, where in the extreme case, $[w_\theta]_{\theta \in \Theta} = [0, \ldots, 0, 1, 0, \ldots, 0]$ (for which ESS($x_*$) = 1). This means that the particles do not agree as to how probable $x_*$ is, so that $x_*$ likely is an OoD input.

### 3.3.2 Out-of-Distribution Detection in Latent Space

Detecting out-of-distribution data in latent space involves identifying latent vectors $z_*$ which are very different from the latent vectors corresponding to the training inputs $D$, and which will thus yield unpredictable results when passed through the decoder to produce $p(x \mid z_*; \theta)$. To discriminate if a given latent vector $z_*$ is an outlier or not, we follow the same approach as in Section 3.3.1 and quantify the informativeness of $z_*$ on the model parameters $\theta$ by measuring the change in the posterior $p(\theta \mid D)$. However, quantifying this change in posterior
by e.g. computing the ESS metric in Eq. (3.20) requires evaluating the likelihood $p(x_\ast | \theta)$ of the input $x_\ast$ corresponding to the latent code $z_\ast$, which is not available to us. We thus alternatively quantify the expected change in the posterior $p(\theta | D)$ by computing the expected ESS, by averaging over the likelihood mixture $p(x | z_\ast, D)$ in Eq. (3.4), thus effectively taking into account all possible inputs $x_\ast$ corresponding to $z_\ast$, i.e.,

$$\text{ESS}_\theta(z_\ast) = \mathbb{E}_{p(x | z_\ast, D)} [\text{ESS}_\theta(x)] \overset{\text{Eq. (3.20)}}{=} \frac{1}{N} \sum_{n=1}^{N} \frac{(\sum_{\theta \in \Theta} p(x_n | z_\ast, \theta))^2}{\sum_{\theta \in \Theta} p(x_n | z_\ast, \theta)^2}, \quad (3.21)$$

where the inputs $x_n \sim p(x | z_\ast, D)$ are drawn from the decoder mixture $p(x | z_\ast, D)$ in Eq. (3.4) by first sampling the index of the mixture component uniformly at random, i.e., $m \sim \mathcal{U}[1,M]$ (due to the mixture weights all being $1/M$), and then sampling $x_n$ from the $m$-th mixture component, i.e., $x_n \sim p(x | z_\ast, \theta_m)$. In contrast to computing the ESS in Eq. (3.20) for outlier detection in input space, where we had to estimate the likelihoods via importance sampling using the (mixture of) inference network(s), computing the likelihoods $p(x_n | z_\ast, \theta)$ in Eq. (3.21) simply involves passing the given latent code $z_\ast$ through the decoders parameterized by $\theta \in \Theta$. In fact, the encoding part of the model is not required for outlier detection in latent space. If we are only interested in outlier detection in latent space, we thus do not face any of the issues arising from the recognition network making wrong inferences for OoD inputs, and as a result do not require a Bayesian treatment of the encoder parameters $\varphi$.

### 3.3.3 Numerically Stable Implementation of the ESS Score

For numerical stability, we in practice always work with log-probabilities (since raw probabilities may get arbitrarily close to and thus be rounded to zero). I.e., instead of directly computing $p(x | \theta)$ in Eq. (3.9) based on the probabilities $p(x | z, \theta)$, $p(z)$ and $q(z | x, \varphi)$, we compute $\log p(x | \theta)$ based on the respective log-probabilities $\log p(x | z, \theta)$, $\log p(z)$ and
log \( q(z \mid x, \varphi) \) as follows:

\[
\log p(x \mid \theta) \simeq \log \left( \frac{1}{K} \sum_{k=1}^{K} \frac{p(x \mid z_k, \theta)p(z_k)}{\sum_{\varphi \in \Phi} q(z_k \mid x, \varphi)} \right) = \log \left( \frac{1}{K} \sum_{k=1}^{K} \exp \left( \log \frac{p(x \mid z_k, \theta)p(z_k)}{\sum_{\varphi \in \Phi} q(z_k \mid x, \varphi)} \right) \right) = \log \text{logmeanexp}_{z_k} \left( \log \frac{p(x \mid z_k, \theta)p(z_k)}{\sum_{\varphi \in \Phi} q(z_k \mid x, \varphi)} \right) = \log \text{logmeanexp}_{z_k} \left( \log \exp(\log(p(x \mid z_k, \theta)p(z_k))) + \log(p(z_k) - \log\logmeanexp_{\varphi}(\log q(z_k \mid x, \varphi))) \right),
\]

where \( z_k \sim q(z \mid x, \mathcal{D}) \), where the last equality uses \( \log \frac{\exp(a)}{\exp(b)} = \log(\exp(a - b)) = a - b \), and where logmeanexp is a variant of the commonly used numerically stable logsumexp function (Goodfellow et al., 2016) which computes the mean instead of the sum.

We thus obtain the vector \( p = [p_{\theta \mid \theta} \mid \theta \in \theta] = [\log p(x \mid \theta)]_{\theta \in \Theta} \) of log-likelihoods, which we can then use to compute the vector \( w = [w_{\theta} \mid \theta \in \Theta] \) of weights \( w_{\theta} \) in Eq. (3.16) by passing \( p \) through the numerically stable softmax function, i.e.,

\[
w_{\theta} = \frac{p(x \mid \theta)}{\sum_{\theta \in \Theta} p(x \mid \theta)} = \frac{\exp(p_{\theta})}{\sum_{\theta \in \Theta} \exp(p_{\theta})} = \frac{\exp(p_{\theta} - p^*)}{\sum_{\theta \in \Theta} \exp(p_{\theta} - p^*)} = \text{softmax}(p - p^*)_{\theta} \quad (3.22)
\]

where \( p^* = \max(p) = \max\{\log p(x \mid \theta) \mid \theta \in \Theta\} \) is the largest log-likelihood value across all \( \theta \in \Theta \).

### 3.4 Related Work

In this section, we give an overview of previous out-of-distribution (OoD) detection approaches, both supervised ones (Section 3.4.1) as well as unsupervised ones (Section 3.4.2). Our focus lies on OoD detection using likelihood-based deep generative models, so we will mainly provide details on those approaches.
3.4.1 Supervised/Discriminative OoD Detection Methods

Most existing OoD detection approaches are task-specific and designed for the context of a certain prediction task. These approaches train a deep discriminative model in a supervised fashion using the given labels. To detect outliers w.r.t. the target task, many approaches then rely on a confidence score to decide on the reliability of the prediction, which is either produced by modifying the model and/or training procedure, or computed/extracted post-hoc from the model and/or predictions (Ahmed and Courville, 2020; An and Cho, 2015; DeVries and Taylor, 2018; Hendrycks and Dietterich, 2019a; Hendrycks and Gimpel, 2017; Liang et al., 2018; Shafaei et al., 2019; Sölch et al., 2016; Sricharan and Srivastava, 2018).

While a comprehensive review of all these individual approaches is beyond the scope of this thesis, we describe DeVries and Taylor (2018) as an example of a supervised approach. In particular, DeVries and Taylor (2018) propose to add a branch to the network that learns to predict a score $c \in [0, 1]$ representing the network’s confidence of predicting the correct output given some input. To make this work, they need to introduce several heuristics and hyperparameters (i.e., they introduce a budget parameter to trade-off the two losses, use different loss functions for different points of each batch, use data augmentation to create difficult training examples, preprocess the inputs).

Alternatively, one can use predictive uncertainty estimates (as inferred by Bayesian or non-Bayesian methods) for OoD detection (Gal, 2016; Lakshminarayanan et al., 2017; Malinin and Gales, 2018; Osawa et al., 2019; Ovadia et al., 2019).

One drawback of such task-specific approaches is that discriminatively trained models by design discard all input features which are not informative about the specific prediction task, such that information which is relevant for OoD detection might be lost.

Finally, we describe the work by McAllister et al. (2019) in more detail, as this appears to be the supervised outlier detection method most closely related to our approach, since it also partly relies on a VAE model. In particular, McAllister et al. (2019) independently train a VAE as well as a Bayesian predictive model $p(y|x,D) = \int_{\psi} p(y|x,\psi)p(\psi|D)d\psi$, and then at test time quantify the uncertainty in the predictive distribution $p(y_*|x_*,D) = \int_{x_0} \int_{x_0} p(y_*,x_0,|x_0,D)p_\theta(x_0|z)q_\phi(z|x_0)dx_0dz$ for a (possible OoD) test input $x_*$. The main idea here is that latent embeddings $z$ of the OoD input $x_*$ drawn from $q_\phi(z|x_0)$ will not contain factors with small variation under the training data distribution, such that $p_\theta(x|z)$ produces outputs $x$ resembling in-distribution data that is semantically close to $x_*$ (i.e., with task-irrelevant factors of variation filtered out). The test input $x_*$ can thus be viewed to be probabilistically projected onto a training point $x$. For example, if in the training data, all walls are white, then "wall color" will not be a factor of variation (e.g. latent dimension $z_i$) in the latent space, such that mapping an image with a different wall color into the latent space will effectively remove
the wall color information (since from an information bottleneck perspective, this information
is not required to compress training inputs), and subsequent mapping back to input space
will "impute" the white wall color the model was trained on, making the image closer to
the training data. They argue that in practice, one is often only interested in points \( x_s \) that
are outliers w.r.t. to certain task-salient features/factors of variation (i.e., certain dimensions
of the latent vector \( z \) if the latent space is disentangled) but not necessarily all of them,
which in their setting are implicitly captured by a target variable \( y \) (e.g., in a robotic control
setting, the target variable "collision probability" might be independent of the task-irrelevant
factor of variation "wall color"). The predictive model thus implicitly defines which factors
of variation are relevant for the prediction task and which are not (the latter will likely be
ignored/thrown away by the model). We note that their setting can be generalized/abstracted
to OoD detection w.r.t. subsets of the latent code (or, if the latent space is not disentangled,
certain subspaces/submanifolds of the latent manifold \( \mathcal{Z} \)). However, as a result, their model
will not be able to handle an OoD point that does not live on the training data manifold, due
to the suboptimality of the inference network \( q_\phi(z | x) \).

3.4.2 Unsupervised/Generative OoD Detection Methods

In contrast, task-agnostic OoD detection methods solely use the input data for the unsu-
pervised training of a deep generative model, which makes them more general, as they do
not require the availability of labels for some prediction task. Moreover, these approaches
capture the entire input data distribution via the generative model, which might be beneficial
for OoD detection (it was shown that despite not exploiting label information, unsupervised
approaches often work better in practice (Ren et al., 2019)). We now review unsupervised
approaches to detect if a given test input \( x_s \) is OoD or not.

Firstly, people have proposed to use the reconstruction error \( ||x_s - \hat{x}||^2_2 \) between \( x_s \) and
its reconstruction \( \hat{x} \) as produced by an ordinary, non-probabilistic autoencoder (AE) (Kramer,
1991), which is motivated by the intuition that the reconstructions produced by an AE should
be better for training inputs than for OoD test inputs (Pimentel et al., 2014; Richter and Roy,
2017).

When using a variational autoencoder, we have a wider spectrum of possible scores to
use for OoD detection. Similar to as in an ordinary AE, we can adopt the score

\[
||x_s - \hat{x}||^2_2, \quad \hat{x} = \arg \max_x p_\theta(x | \hat{z}), \quad \hat{z} = \arg \max_z q_\phi(z | x_s), \tag{3.23}
\]
i.e., the squared $\ell_2$ reconstruction error

$$
\| x_\ast - \hat{x} \|^2 \propto - \log \int_z p_\theta(x_\ast | z) \delta(z - \hat{z}) dz \approx - \log p_\theta(x_\ast)
$$

between $x_\ast$ and the most probable reconstruction $\hat{x}$. That is, using the point estimate $\hat{z}$ of the most likely latent embedding of $x_\ast$ instead of the posterior $q(z | x_\ast, \varphi)$, which essentially treats a VAE with a Gaussian likelihood as a non-probabilistic AE (Šmídl et al., 2019) (a somewhat similar score is adopted in Sun et al. (2018)). Šmídl et al. (2019) note that the reconstruction error score is suboptimal, and propose an alternative score which takes into account the geometry of the data manifold.

More generally, as a VAE is probabilistic, we can use the log-likelihood $\log p_\theta(x_\ast)$ (i.e., the probability of observing $x_\ast$ under the generative model) of an ordinary VAE as estimated via importance sampling (see Section 2.5.3). To the best of our knowledge, this natural idea of using the likelihood of a generative model for OoD detection was first proposed by Bishop (1994).

Alternatively, to avoid log-likelihood estimation, several works propose to detect OoD inputs based on (parts of) the ELBO, which is motivated by the fact that since our training objective is the maximization of the ELBO, its value should be higher for training data than for OoD data. Söllch et al. (2016) propose to use the complete VAE ELBO

$$
\mathbb{E}_{q_\varphi(z|x_\ast)}[\log p_\theta(x_\ast | z)] - \text{KL}(q_\varphi(z | x_\ast) \| p(z))
$$

(they show empirical inferiority to the log-likelihood), while An and Cho (2015); Škvára et al. (2018) propose to only use the expected log-likelihood term $\mathbb{E}_{q_\varphi(z|x_\ast)}[\log p_\theta(x_\ast | z)]$, and Alemi et al. (2018) propose to only use the KL term $\text{KL}(q_\varphi(z | x_\ast) \| p(z))$ (also called rate term) measuring the distance of the latent embedding $q_\varphi(z | x_\ast)$ of $x_\ast$ from the prior $p(z)$.

Ikeda et al. (2019) propose a method to estimate the dimensions contributing to anomalies detected by a VAE. They observe that the recognition network $q_\varphi(z | x)$ might not work properly on OoD data, and note that the estimate of the ELBO of the log-likelihood of the test data is not necessarily correct, since the VAE is trained under the assumption that all the data is drawn i.i.d. from the training distribution and therefore the result computed with the OoD is biased (Xu et al., 2018).

Perhaps closest to our approach is the work by Choi and Jang (2018), which use an ensemble (Lakshminarayanan et al., 2017) of independently trained likelihood-based generative models (i.e., with random parameter initializations and random data shuffling) to
approximate the Watanabe-Akaike Information Criterion (Watanabe, 2010),
\[
\mathbb{E}_{p(\theta | D)}[\log p(x_\ast | \theta)] - \text{Var}_{p(\theta | D)}[\log p(x_\ast | \theta)].
\] (3.26)

The WAIC provides an asymptotically correct estimate between the training and test set expectations (assuming a fixed underlying data distribution) and can be viewed as a corrected log-likelihood score which penalizes points \(x\) with a large variance in log-likelihoods. Their approach has two shortcomings as compared to ours: Firstly, while we use stochastic gradient MCMC on a single model to obtain \(M\) approximate posterior samples, they train \(M\) independent models to form an ensemble, which (a) is not a principled way of doing approximate Bayesian inference (although connections can be made under certain conditions (Mandt et al., 2017)), and (b) is \(M\) times more computationally expensive in practice. Secondly, independent of the quality of the posterior approximation, their score is not expected to work well, since it still relies on the suboptimal log-likelihood estimates (as we will demonstrate in the experiments). Finally, the theoretical justification of the WAIC assumes that all data (at training and test time) is drawn from the same underlying distribution, and it is not clear what happens if OoD data is encountered during test time. While the WAIC might still be used as a heuristic in that case, there is no theoretical justification for doing so, making the approach unprincipled.

Another unsupervised approach is the likelihood ratio method by Ren et al. (2019), which corrects the likelihood \(\log p(x_\ast | \theta)\) for confounding general population level background statistics captured by a background model \(p(x_\ast | \theta_0)\), resulting in the score
\[
\log p(x_\ast | \theta) - \log p(x_\ast | \theta_0).
\] (3.27)

The background model \(p(x_\ast | \theta_0)\) is in practice trained by perturbing the data \(D\) with noise to corrupt its semantic structure, i.e., by sampling input dimensions i.i.d. from a Bernoulli distribution with rate \(\mu \in [0.1, 0.2]\) and replacing their values by uniform noise, e.g. \(x_i \sim U\{0,\ldots,255\}\) for images.

Nalisnick et al. (2019) recently proposed to use the score
\[
\left|\log p(x_\ast | \theta) - \frac{1}{N} \sum_{x \in D} \log p(x | \theta)\right|
\] (3.28)

to account for the typicality of the input \(x_\ast\), motivated by the intuition that in-distribution data should be located in the typical set of a distribution, rather than than in areas of high probability density.
While our focus lies on unsupervised OoD detection methods using likelihood-based deep generative models such as VAEs, we note that OoD detection methods based on GANs have also been proposed (Akcay et al., 2018; Zenati et al., 2018).

Finally, we note that to perform unsupervised OoD detection, it is not necessary to train a complex deep generative model, but there exist simple baselines which perform surprisingly well in some settings. For example, a classical approach is $k$-nearest neighbors ($k$-NN) (Angiulli and Pizzuti, 2002), using, e.g., $\frac{1}{k} \sum_{i=1}^{k} \|x_s - x_{n(i)}\|_2$, i.e., the average distance in input space of $x_s$ to the $k$-nearest neighbors $\{x_{n(i)}\}_{i=1}^{k}$ in the training set $D$, as a score (Škvára et al., 2018). Another simple approach that recent papers have benchmarked against is the so-called Annulus method, which measures the distance of $x_s \in \mathbb{R}^l$ to the hypersphere with radius $\sqrt{l}$, i.e., $\|x_s\|_2 - \sqrt{l}$, which corresponds to the typical set of a high-dimensional Gaussian distribution (Choi and Jang, 2018; Nalisnick et al., 2019).

### 3.5 Empirical Evaluation

We now present experimental results demonstrating the efficacy of our approach, on OoD detection both in input space (Section 3.5.1) and in latent space (Section 3.5.2).

#### 3.5.1 Out-of-Distribution Detection in Input Space

**BVAE Details**

We implemented the proposed BVAE model in PyTorch (see Section 3.2.6 for pseudocode), using the robust, scale adapted SGHMC variant proposed by Springenberg et al. (2016)\(^1\). We follow Chen et al. (2014); Springenberg et al. (2016) and use Gaussian priors over $\theta$ and $\phi$, i.e., $p(\theta) = N(0, \lambda_{\theta}^{-1})$ and $p(\phi) = N(0, \lambda_{\phi}^{-1})$. We also place Gamma hyperpriors over the precision parameters $\lambda_{\theta}$ and $\lambda_{\phi}$, i.e., $p(\lambda_{\theta}) = \Gamma(\alpha_{\theta}, \beta_{\theta})$ and $p(\lambda_{\phi}) = \Gamma(\alpha_{\phi}, \beta_{\phi})$, with $\alpha_{\theta} = \beta_{\theta} = \alpha_{\phi} = \beta_{\phi} = 1$, and resample $\lambda_{\theta}$ and $\lambda_{\phi}$ after every training epoch (i.e., after an entire pass over $D$). We also follow the previous work and use a step size of $10^{-3}$ and momentum decay of 0.05 for SGHMC. We discard samples within a burn-in phase of $B = 1$ epoch, and store a sample (of both encoder and decoder parameters) after every $D = 1$ epoch. We only keep the 10 most recent samples to represent the posterior. In addition to the BVAE described in Section 3.2 which uses the two ELBOs $\mathcal{L}_{\theta, \phi}$ in Eq. (2.87) and $\mathcal{L}_{\theta}$ in Eq. (3.7) to train/sample from $\phi$ and $\theta$, respectively (denoted by BVAE), we also report results on the following variants: a BVAE that uses the ordinary VAE ELBO in Eq. (2.87) for both $\phi$ and

\(^1\)We use their implementation of the SGHMC sampler as a PyTorch Optimizer (https://github.com/automl/pybnn), thus serving as a drop-in replacement for an optimizer such as SGD.
Table 3.1 AUROC↑, AUPRC↑, and FPR80↓ scores (where higher ↑ or lower ↓ is better) of the baselines (top) and our methods (bottom). For the experiment on FashionMNIST with with held-out classes, we report the mean scores over all five class splits.

<table>
<thead>
<tr>
<th></th>
<th>FashionMNIST vs MNIST</th>
<th>FashionMNIST (held-out)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AUROC↑</td>
<td>AUPRC↑</td>
</tr>
<tr>
<td>LL</td>
<td>0.557</td>
<td>0.564</td>
</tr>
<tr>
<td>WAIC</td>
<td>0.541</td>
<td>0.548</td>
</tr>
<tr>
<td>LLR</td>
<td>0.617</td>
<td>0.613</td>
</tr>
<tr>
<td>TT</td>
<td>0.482</td>
<td>0.502</td>
</tr>
<tr>
<td>BVAE-ESS</td>
<td>0.842</td>
<td>0.830</td>
</tr>
<tr>
<td>BVAE-ESSθ</td>
<td>0.871</td>
<td>0.855</td>
</tr>
<tr>
<td>BVAE-ESSφ</td>
<td>0.897</td>
<td>0.893</td>
</tr>
<tr>
<td>BVAE-ESSθφ</td>
<td><strong>0.921</strong></td>
<td><strong>0.907</strong></td>
</tr>
<tr>
<td>BVAEθ-ESS</td>
<td>0.904</td>
<td>0.891</td>
</tr>
</tbody>
</table>

Fig. 3.1 Histograms of LL (left) and BVAE-ESSφ (right) on FashionMNIST (in-distribution) and MNIST (out-of-distribution). The ESS score separates the data more clearly than the LL score.

\( \theta \) (BVAE; this is the second BVAE training variant described in Section 3.2.6) and a VAE where only \( \theta \) is integrated out and \( \phi \) is estimated via maximum likelihood (BVAE_\theta). We furthermore consider both alternative ways of estimating the log-likelihoods for computing the ESS (cf. Section 3.3.1), i.e., the case where log-likelihoods are computed using samples \( \Phi \) of encoders and \( \Theta \) of decoders that are treated as independent (ESS), and the case where the log-likelihoods are computed using coupled samples \( \{ (\phi_j, \theta_j) \}_{j=1}^M \) (ESS_\phi).

Experimental Setup

We use three benchmarks:
Out-of-Distribution Detection via Bayesian Variational Autoencoders

Fig. 3.2 (Left) Precision-recall curves (treating OoD data as the upper class) and (right) ROC curves of all methods on the FashionMNIST vs. MNIST benchmark.

1. FashionMNIST (in-distribution) vs. MNIST (OoD) \(\text{\cite{Akcay18, Hendrycks2019, Nalisnick19, Ren19, Zenati18}}\),

2. SVHN (in-distribution) vs. CIFAR10 (OoD)\(^2\) \(\text{\cite{Hendrycks2019, Nalisnick19}}\), and

3. eight classes of FashionMNIST (in-distribution) vs. the remaining two classes (OoD), using five different splits \(\{(0,1), (2,3), (4,5), (6,7), (8,9)\}\) of held-out classes \(\text{\cite{Ahmed20}}\).

We compare against the log-likelihood (LL) as well as three state-of-the-art methods for unsupervised OoD detection (all described in Section 3.4):

1. The generative ensemble based method by Choi and Jang (2018) composed of five independently trained models (WAIC),

2. the likelihood ratio method by Ren et al. (2019) (LLR), using Bernoulli rates \(\mu = 0.2\) for the FashionMNIST vs. MNIST experiment \(\text{\cite{Ren19}}\), and \(\mu = 0.15\) for the other experiment, and

3. the test for typicality by Nalisnick et al. (2019) (TT).

\(^2\)Interestingly, following the implementation details provided in Nalisnick et al. (2019), we were not quite able to reproduce their results. In particular, our experiments yielded better calibrated log-likelihood scores for the benchmark CIFAR10 (in-distribution) vs. SVHN (OoD) than theirs (i.e., our VAE model assigned lower likelihood to most of the SVHN OoD data than to the CIFAR10 in-distribution data, as desired); this phenomenon requires further investigation. For this reason, we instead chose the opposite benchmark, i.e., SVHN (in-distribution) vs. CIFAR10 (OoD), for which the likelihood scores largely overlap, as shown in Fig. 3.3 (left).
Table 3.2 AUROC↑, AUPRC↑, and FPR80↓ scores (where higher ↑ or lower ↓ is better) of the baselines (top) and our methods (bottom) on the SVHN vs. CIFAR10 benchmark.

<table>
<thead>
<tr>
<th></th>
<th>AUROC↑</th>
<th>AUPRC↑</th>
<th>FPR80↓</th>
</tr>
</thead>
<tbody>
<tr>
<td>LL</td>
<td>0.574</td>
<td>0.575</td>
<td>0.634</td>
</tr>
<tr>
<td>WAIC</td>
<td>0.293</td>
<td>0.380</td>
<td>0.912</td>
</tr>
<tr>
<td>LLR</td>
<td>0.570</td>
<td>0.570</td>
<td>0.638</td>
</tr>
<tr>
<td>TT</td>
<td>0.395</td>
<td>0.428</td>
<td>0.859</td>
</tr>
<tr>
<td>BVAE-ESS</td>
<td>0.667</td>
<td>0.656</td>
<td>0.564</td>
</tr>
<tr>
<td>BVAE-ESSθ</td>
<td>0.669</td>
<td>0.647</td>
<td>0.562</td>
</tr>
<tr>
<td>BVAE-ESSθ</td>
<td>0.828</td>
<td>0.817</td>
<td>0.281</td>
</tr>
<tr>
<td>BVAE-ESSθ</td>
<td>0.814</td>
<td>0.799</td>
<td>0.310</td>
</tr>
<tr>
<td>BVAEθ-ESS</td>
<td>0.807</td>
<td>0.793</td>
<td>0.331</td>
</tr>
</tbody>
</table>

Fig. 3.3 Histograms of LL (left) and BVAE-ESS (right) on SVHN (in-distribution) and CIFAR10 (out-of-distribution). The ESS score separates the data more clearly than the LL score.

All methods use VAEs for the log-likelihood estimation.[^3] They are evaluated by randomly selecting 5,000 in-distribution and OoD inputs from held-out test sets and computing the following threshold independent metrics ([Alemi et al., 2018; Hendrycks and Dietterich, 2019a; Hendrycks and Gimpel, 2017; Liang et al., 2018; Ren et al., 2019]):

1. The area under the ROC curve (AUROC↑),
2. the area under the precision-recall curve (AUPRC↑), and
3. the false-positive rate at 80% true-positive rate (FPR80↓).

[^3]: The architecture of the deconvolutional VAE we use and the training protocol follows previous work ([Choi and Jang, 2018; Nalisnick et al., 2019; Ren et al., 2019]). We use the Adam optimizer ([Kingma and Ba, 2015]) with learning rate $10^{-3}$ for all maximum likelihood fits of parameters.
Fig. 3.4 (Left) Precision-recall curves (treating OoD data as the upper class) and (right) ROC curves of all methods on the SVHN vs. CIFAR10 benchmark.

Results

Table 3.1 shows that our approaches generally perform better on the FashionMNIST vs. MNIST and FashionMNIST with held-out classes benchmarks. Coupling the encoder and decoder samples appears beneficial, which might be due to the fact that we actually take the samples in couples after every epoch. Being Bayesian about the encoder parameters $\phi$ also seems to help. Fig. 3.2 show the precision-recall and ROC curves used to compute the metrics, for the FashionMNIST vs. MNIST experiment (see Chapter B for the curves for the FashionMNIST with held-out classes experiments). Fig. 3.1 shows histograms of the log-likelihoods (left) and of the BVAE-ESS scores (right) on FashionMNIST in-distribution (blue) vs. MNIST OoD (orange) test data. While the log-likelihoods strongly overlap, the ESS scores more clearly separate in-distribution data (closer to $\text{ESS} = 10$ on the r.h.s.) from OoD data (closer to $\text{ESS} = 1$ on the l.h.s.).

For the SVHN vs. CIFAR10 benchmark, looking at Table 3.2 and Fig. 3.4, we arrive at the same conclusion as with the other two benchmarks in that our approaches appear to significantly outperform the baselines we compare against. While the variants $\text{BVAE-ESS}$ and $\text{BVAE-ESS}^\theta$ are not as strong as the other BVAE-ESS variants on this benchmark (as compared to the benchmarks considered in Section 3.3.1), they still outperform the other baselines. The histograms in Fig. 3.3 again show that the BVAE-ESS scores (right) more clearly separate in-distribution data from OoD data than the log-likelihood scores (left).
3.5 Empirical Evaluation

3.5.2 Out-of-Distribution Detection in Latent Space

While input space OoD detection is a well-studied problem, OoD detection in latent space was only recently identified as an important open problem (Alperstein et al., 2019; Gómez-Bombarelli et al., 2018; Griffiths and Hernández-Lobato, 2020; Mahmood and Hernández-Lobato, 2019). Thus, there is a lack of suitable experimental benchmarks, making a quantitative evaluation challenging. A major issue in designing toy benchmarks based on commonly-used datasets such as MNIST or FashionMNIST (i.e., as typically used for assessing input space OoD detection) is that it is not clear how to obtain ground truth labels for which latent points are OoD and which are not, as we require OoD labels for all possible latent vectors \( z \in \mathbb{R}^{Z} \), not just for those corresponding to inputs \( x \) from some training or test set. As a first step to facilitate a systematic empirical evaluation of latent space OoD detection techniques, we propose the following experimental protocol.

We use the BVAE variant where only \( \theta \) is integrated out and \( \varphi \) is estimated via maximum likelihood (denoted by BVAE\( _{\theta} \) in Section 3.5.1, using the same prior and hyperprior as described there). We train this BVAE model on some dataset (we will use FashionMNIST here), and then sample \( N = 10,000 \) latent test vectors \( z_* \) from the Gaussian \( \mathcal{N}(0, b \cdot I_Z) \) where \( b \in \mathbb{R}^+ \), following Mahmood and Hernández-Lobato (2019) (we use \( b = 10,000 \)). Since there does not exist a ground truth label for whether a given latent point \( z_* \) is OoD or not, we instead compute some OoD proxy score (to be detailed below) for each of the \( N \) latent test vectors and then simply define the \( N/2 \) latents with the lowest scores to be in-distribution, and the \( N/2 \) latents with the highest scores to be OoD.

For this OoD score, we propose to train an ensemble of \( J \) convolutional neural network classifiers with parameters \( W = \{w_j\}_{j=1}^J \) on FashionMNIST (Lakshminarayanan et al., 2017).
We then approximate the mutual information score\(^4\) proposed by Houlsby et al. (2011), i.e.,

$$I(\mathbf{w}, y | \mathbf{x}_*, D) = H(p(y | \mathbf{x}_*, D)) - \mathbb{E}_{p(\mathbf{w} | D)}[H(p(y | \mathbf{x}_*, \mathbf{w}))],$$

where

$$\mathbb{E}_{p(\mathbf{w} | D)}[H(p(y | \mathbf{x}_*, \mathbf{w}))] \simeq \frac{1}{J} \sum_{\mathbf{w} \in \mathcal{W}} H(p(y | \mathbf{x}_*, \mathbf{w}))$$

is the average entropy of the predictive class distribution of the classifier with parameters \(\mathbf{w}\), and

$$H(p(y | \mathbf{x}_*, D)) \simeq H\left(\frac{1}{J} \sum_{\mathbf{w} \in \mathcal{W}} p(y | \mathbf{x}_*, \mathbf{w})\right)$$

is the entropy of the mixture \(\frac{1}{J} \sum_{\mathbf{w} \in \mathcal{W}} p(y | \mathbf{x}_*, \mathbf{w})\) of categorical distributions \(p(y | \mathbf{x}_*, \mathbf{w})\) (which is again categorical with averaged probits). Note that this score requires a test input \(\mathbf{x}_*\). Since in our setting, we only know the latent code \(\mathbf{z}_*\) corresponding to \(\mathbf{x}_*\), we instead use the expected mutual information under the mixture decoding distribution \(p(\mathbf{x} | \mathbf{z}_*, D)\) in Eq. (3.4) defined by the decoder ensemble \(\Theta\), i.e.,

$$\mathbb{E}_{p(\mathbf{x} | \mathbf{z}_*, D)}[I(\mathbf{w}, y | \mathbf{x}, D)] \simeq \frac{1}{N} \sum_{n=1}^{N} I(\mathbf{w}, y | \mathbf{x}_n, D), \quad \mathbf{x}_n \sim p(\mathbf{x} | \mathbf{z}_*, D).$$

In practice, we use an ensemble of \(J = 5\) classifiers and \(N = 32\) input samples to compute the score in Eq. (3.32).

We compare the expected ESS (see Section 3.3.2, with \(N = 32\)) against two baselines:

1. The distance of \(\mathbf{z}_* \in \mathbb{R}^2\) to the spherical annulus of radius \(\sqrt{Z - 1}\), since that is where most probability mass lies under our prior \(\mathcal{N}(\mathbf{0}, I_Z)\) (Annulus)\(^5\) (Alperstein et al., 2019), and

2. the log-probability of \(\mathbf{z}_*\) under the training data distribution in latent space \(q(\mathbf{z}) = \frac{1}{N} \sum_{\mathbf{x} \in D} q(\mathbf{z} | \mathbf{x}, \varphi)\), i.e., a uniform mixture of \(N\) Gaussians in our case \((qz)\(^6\) (Mahmood and Hernández-Lobato, 2019).

Fig. 3.5 shows that our proposed method significantly outperforms the two baselines on this benchmark task, which do not perform much better than random.

\(^4\)This score is closely related to the disagreement score \(\sum_{\mathbf{w} \in \mathcal{W}} D_{KL}(p(y | \mathbf{x}_*, \mathbf{w}) \| p(y | \mathbf{x}_*, D))\) proposed by Lakshminarayanan et al. (2017) for ensemble-based OoD detection, which could be used alternatively.

\(^5\)The Annulus score is in practice equivalent to using the log-probability of \(\mathbf{z}_*\) under the prior \(p(\mathbf{z})\).

\(^6\)For efficiency, we only consider the 100 nearest neighbors (found by a 100-NN model) of a latent test point \(\mathbf{z}_*\) for computing this log-probability (Mahmood and Hernández-Lobato, 2019).
3.6 Discussion

In this chapter, we introduced a new probabilistic deep learning method for tackling the out-of-distribution (OoD) detection problem, both in input space as well as in latent space. Our method uses a Bayesian VAE model, where inference is not just performed over the latent variables, but also over the model parameters. The resulting posterior distribution over model parameters then allows us to define information-theoretic measures to detect OoD data, which we have shown to be effective in practice. In the next chapter, we will turn to the problem of data-efficient optimization, developing a new method that is also based on a VAE. In particular, we will propose an approach that optimizes the objective function in the low-dimensional, continuous latent space of a VAE, instead of in the high-dimensional, structured input space. This is conceptually related to the latent space OoD detection approach considered in this chapter.
Chapter 4

Data-Efficient Optimization via Weighted Retraining of Variational Autoencoders

In this chapter, we develop a new probabilistic deep learning method for tackling data-efficient optimization problems (cf. Section 2.6.2). In particular, our contributions in this chapter are structured as follows:

(a) We identify and describe two critical failure modes of previous methods based on latent-space-optimization (LSO) which severely limit their efficiency and performance, and thus practical applicability (Section 4.1).

(b) We propose to combine dataset weighting with periodic retraining of the generative model used within LSO as an effective way to directly address the issues identified (Section 4.2).

(c) We review relevant prior work on LSO, weighted retraining, Bayesian optimization, reinforcement learning and conditional generative models (Section 4.3).

(d) We empirically demonstrate that weighted retraining significantly benefits LSO across a variety of application domains and generative models, achieving substantial improvements over state-of-the-art methods on a widely-used chemical design benchmark (Section 4.4).

The material in this chapter was previously published in ‘Sample-Efficient Optimization in the Latent Space of Deep Generative Models via Weighted Retraining’ (Tripp et al., 2020). The research was conducted in collaboration with my co-first author Austin Tripp, and was supervised by José Miguel Hernández-Lobato throughout. I was involved closely with all aspects of the paper, including the theoretical results, the experiments and the writing of the paper.
Fig. 4.1 Schematic illustrating LSO with and without weighted retraining. The cartoon illustrates the input/latent space of the generative model (top). The latent manifold from Section 4.4.2’s 2D shape area maximization task is shown for comparison (bottom). Each image in the manifold shows the result of decoding a latent point on a uniform square grid in a 2D latent space; images are centered on the original grid points. Red/green regions correspond to points with low/high objective function values respectively. The yellow star is the global optimum in $X$. Coloured circles are data points; their radius represents their weight. The dashed line surrounds the region of $X$ modelled by $g$ (i.e. $g(Z)$, the image of $Z$). (a) The status of the generative model $g$ at the start of optimization. (b) The result of standard LSO with $g$ fixed, which queries the points in orange. It is only able to find points close to the training data used to learn $Z$, resulting in slow and incomplete exploration of $X$. (c) The result midway (left) and at the end (right) of LSO with our proposed approach, which weights data points according to their objective function value and retrains $g$ to incorporate newly queried data. This continually adjusts $Z$ to focus on modelling the most promising regions of $X$, speeding up the optimization and allowing for substantial extrapolation beyond the initial training data.

4.1 Failure Modes of Latent Space Optimization

To understand the shortcomings of LSO, it is necessary to first examine in detail the role of the generative model, which is usually a DGM. State-of-the-art DGMs such as VAEs and GANs are trained with a prior $p(z)$ over the latent space $Z$. This means that although the resulting function $g : Z \mapsto X$ is defined over the entire latent space $Z$, it is effectively only trained on points in regions of $Z$ with high probability under $p$. Importantly, even if $Z$ is an
unbounded space with infinite volume such as $\mathbb{R}^Z$, because $p$ has finite volume, there must exist a finite subset $Z' \subset Z$ that contains virtually all the probability mass of $p$. We call $Z'$ the feasible region of $Z$. Although in principle optimization can be performed over all of $Z$, it has been widely observed that optimizing outside of the feasible region tends to give poor results, yielding samples that are low-quality, or even invalid (e.g. invalid molecular strings, non-grammatical sentences); therefore all LSO methods known to us employ some sort of measure to restrict the optimization to near or within the feasible region (Daxberger and Hernández-Lobato, 2019; Gómez-Bombarelli et al., 2018; Griffiths and Hernández-Lobato, 2020; Kusner et al., 2017; Mahmood and Hernández-Lobato, 2019; Nguyen et al., 2016; White, 2016). This means that LSO should be treated as a bounded optimization problem, whose feasible region is determined by $p$.

Informally, the training objective of $g$ encourages points sampled from within the feasible region to match the data distribution that $g$ was trained on, effectively “filling” the feasible region with points similar to the dataset, such that a point’s relative volume is roughly proportional to its frequency in the training data. For many optimization problems, most of the training data for the DGM is low-scoring (i.e., highly sub-optimal objective function values), thereby causing most of the feasible region to contain low-scoring points. Not only does this make the optimization problem more difficult to solve (like finding the proverbial “needle in a haystack”), but actually leaves insufficient space in the feasible region for a large number of novel, high-scoring points that lie outside the training distribution to be modelled by the DGM. Therefore, even a perfect optimization algorithm with unlimited evaluations of the objective function might be unable to find a novel point that is substantially better than the best point in the original dataset, simply because such a point may not exist in the feasible region.

This pathological behaviour is conceptually illustrated in Fig. 4.1b, where LSO is unable to find or even approach the global optimum that lies far from the training data. We propose that LSO’s performance is severely limited by two concrete problems in its setup. The first problem is that the generative model’s training objective (to learn a latent space that captures the data distribution as closely as possible), does not necessarily match the true objective (to learn a latent space that is amenable to efficient optimization of the objective function). Put in terms of the cartoon in Fig. 4.1b, the feasible region that is learned, which uniformly and evenly surrounds the data points, is not the feasible region that would be useful for optimization, which would model more of the green region at the expense of the red region. This is also seen in the 2D shape area maximization task in Fig. 4.1b, where the latent manifold contains only low-area shapes that the model was trained on, and nothing close to the all-black global optimum. The second problem is that information on new points acquired
during the iterative optimization procedure is not propagated to the generative model, where it could potentially help to refine and expand the coverage of the feasible region, uncovering new promising regions that an optimization algorithm can exploit. In terms of Fig. 4.1b, the new data is not used to shift the feasible region toward the green region, despite the optimization process indicating that this is a very promising region of $X$ for optimization. Luckily, we believe that neither of these two problems is inherent to LSO, and now pose a framework that directly addresses them.

4.2 Latent Space Optimization With Weighted Retraining

In this section, we start by independently introducing the two components underpinning our proposed approach. First, we describe how generative models can be trained with an objective that uses data weighting to put different emphasis on different data points (Section 4.2.1). Second, we introduce how periodic retraining of a generative model allows us to continually update its latent space to adapt to new data (Section 4.2.2). Finally, we describe how these two components can be combined to yield our proposed weighted retraining method (Section 4.2.3).

4.2.1 Training a Generative Model With a Weighted Training Objective

While it is unclear in general how to design a generative model that is maximally amenable to LSO, the argument presented in Section 4.1 suggests that it would at least be beneficial to dedicate a higher fraction of the feasible region to modelling high-scoring points. One obvious but inadequate method of achieving this is to simply discard all low-scoring points from the dataset used to train the DGM, e.g. by keeping only the top 10% of the data set (in terms of score). While this strategy could be feasible if data is plentiful, when data is scarce this option may not be viable because state-of-the-art neural networks need a large amount of training data to avoid overfitting. This issue can be resolved by not viewing inclusion in the dataset as a binary choice, but instead as a continuum that can be realized by weighting the data points unevenly. If the generative model is trained on a distribution that systematically places more probability mass on high-scoring points and less mass on low-scoring points, the distribution-matching term in the DGM’s training objective will incentivize a larger fraction of the feasible region’s volume to be used to model high-scoring points, while simultaneously using all known data points to learn useful representations and avoid overfitting.

A simple way to achieve this weighting is to assign an explicit weight $w_n$ to each data point, such that $\sum_n w_n = 1$. As the training objective of common DGMs involves the
expected value of a loss function $\mathcal{L}$ with respect to the data distribution,\(^1\) weighted training can be implemented by simply replacing the empirical mean over the training data with a weighted empirical mean: i.e. $\sum_{x_n \in \mathcal{D}} w_n \mathcal{L}(x_n)$ instead of $\sum_{x_n \in \mathcal{D}} \frac{1}{N} \mathcal{L}(x_n)$. In practice, mini-batch stochastic gradient descent is used to optimize this objective to avoid summing over all data points. Unbiased mini-batches can be constructed by sampling each data point $x_n$ with probability $w_n$ with replacement to construct each batch (see Section C.1.2 for more details).

We offer no universal rules for setting weights, except that all weights $w_n$ should be restricted to strictly positive values, because a negative weight would incentivize the model to perform poorly, and a weight of zero is equivalent to discarding a point. This aside, there are many reasonable ways to choose the weights such that high-scoring points are weighted more, and low-scoring points are weighted less. In this work, we decide to use a rank-based weight function,

$$w(x; \mathcal{D}, k) \propto \frac{1}{kN + \text{rank}_{f, \mathcal{D}}(x)}, \quad \text{rank}_{f, \mathcal{D}}(x) = |\{x_n : f(x_n) > f(x), \quad x_n \in \mathcal{D}\}|, \quad (4.1)$$

which assigns a weight roughly proportional to the reciprocal (zero-based) rank of each data point. We chose Eq. (4.1) because it yields weights which are always positive, resilient to outliers, and has stable behaviour over a range of dataset sizes (this is explained further in Section C.1.1). Furthermore, as shown in Fig. 4.2, it admits a single tunable hyperparameter $k$ which continuously controls the degree of weighting, where $k = \infty$ corresponds to uniform weighting, i.e. $w_n = \frac{1}{N}, \forall n$, while $k = 0$ places all mass on only the single point with the highest objective function value.

\(^1\)For a VAE, $\mathcal{L}$ is the per-datapoint ELBO (Kingma and Welling, 2014), while for a GAN, $\mathcal{L}$ is the discriminator score (Goodfellow et al., 2014).

![Fig. 4.2](image-url)  
**Fig. 4.2** Histogram of objective function values for the ZINC dataset (see Section 4.4) with uniform weighting (in blue) as well as rank weighting from Eq. (4.1) for different $k$ values (in orange). Large $k$ approaches uniform weighting, while small $k$ places most weight on high-scoring points.
4.2.2 Periodic Retraining to Update the Latent Space

To allow the latent manifold to adapt to new information, we propose a conceptually simple solution: periodically retraining the generative model during the optimization procedure. In practice, this could be done by training a new model from scratch, or by fine-tuning the previously trained model on the novel data. However, as is often pointed out in the active learning literature, the effect of adding a few additional points to a large dataset is rather negligible, and thus it is unlikely that the generative model will change significantly if retrained on this augmented dataset (Sener and Savarese, 2018). While one could also retrain on only the new data, this might lead to the well-known phenomenon of catastrophic forgetting (McCloskey and Cohen, 1989).

A key observation we make is that the data weighting outlined in Section 4.2.1 actually resolves this problem. Specifically, if the new points queried are high-scoring, then a suitable weighting scheme (such as Eq. (4.1)) will assign a large weight to them, while simultaneously decreasing the weights of many of the original data points, meaning that a small number of new points can have a disproportionate impact on the training distribution. If the generative model is then retrained using this distribution, it can be expected to change significantly to incorporate these new points into the latent space in order to minimize the weighted loss. By contrast, if the new points queried are low-scoring, then the distribution will change negligibly, and the generative model will not significantly update, thereby avoiding adding new low-scoring points into the feasible region.

4.2.3 Weighted Retraining Combined

When put together, data weighting and periodic retraining complement each other elegantly, transforming the generative model from a passive decoding function into an active participant in the optimization process, whose role is to ensure that the latent manifold is constantly occupied by the most updated and relevant points for optimization. Their combined effect is visualized conceptually in Fig. 4.1c. In the first iteration, weighted training creates a latent space with more high scoring points, causing the feasible region to extend farther into the green region at the expense of the red region. This allows a better orange point to be chosen relative to Fig. 4.1b. In the second iteration in Fig. 4.1c, weighted training with the orange point incorporates even more high-scoring points into the latent space, allowing an even better point to be found. Qualitatively similar results can be seen in the 2D shape area maximization task, where weighted retraining introduces points with very high areas into the latent space compared to Fig. 4.1b (details for this experiment are given in Section 4.4).
4.3 Related Work

In the remainder of this paper, we refer to the combination of these techniques as weighted retraining for brevity; see Algorithm 4 for pseudocode. We highlight that this algorithm is straightforward to implement in most models, with brief examples given in Section C.1.3. Computationally, the overhead of the weighting is minimal, and the cost of the retraining can be reduced by fine-tuning an existing model on the weighted dataset instead of retraining it from scratch. Although this may still be prohibitively expensive for some applications, we stress that in many scenarios the cost of training a model is insignificant compared to even a single evaluation of the objective function (e.g., performing wet-lab experiments for drug design), making weighted retraining a sensible choice.

4.3 Related Work

While a large body of work is applicable to the general optimization problem considered in this chapter (cf. Section 2.6.2) — both with and without machine learning — below we focus only on the most relevant machine learning literature.

Latent Space Optimization Early formulations of LSO were motivated by scaling Gaussian processes (GPs) to high dimensional problems with simple linear manifolds, using either random projections (Wang et al., 2013) or a learned transformation matrix (Garnett et al., 2014). LSO using DGMs was first applied to chemical design (Gómez-Bombarelli et al.,...
2018), and further built upon subsequently (Dai et al., 2018; Daxberger and Hernández-Lobato, 2019; Eissman et al., 2018; Griffiths and Hernández-Lobato, 2020; Jin et al., 2018a; Kajino, 2019; Kusner et al., 2017; Mahmood and Hernández-Lobato, 2019). It has also been applied to other fields, e.g. automatic machine learning (Lu et al., 2018; Luo et al., 2018; Zhang et al., 2019), conditional image generation (Nguyen et al., 2017, 2016), and model explainability (Antorán et al., 2021). If the surrogate model is a GP, the DGM can be viewed as an “extended kernel”, making LSO conceptually related to deep kernel learning (Huang et al., 2015; Wilson et al., 2016).

**Weighted Retraining** A few previous machine learning methods can be viewed as implementing a version of weighted retraining. The cross-entropy (CE) method iteratively retrains a generative model using a weighted training set, such that high-scoring points receive higher weights (De Boer et al., 2005; Rubinstein, 1999, 1997). Indeed, particular instantiations of the CE method such as reward-weighted regression (Peters and Schaal, 2007), feedback GAN (Gupta and Zou, 2019), and design/conditioning by adaptive sampling (DbAS/CbAS) (Brookes et al., 2019; Brookes and Listgarten, 2018) have been applied to similar problem settings as our work. However, our proposed method of weighted retraining has two main differences from CE. Firstly, *standard CE produces only binary weights* (De Boer et al., 2005), which amounts to simply adding or removing points from the training set. This is sub-optimal for reasons discussed in Sections 4.2.1 and 4.2.2, and consequently, *we consider a strictly more general form of weighting*. Secondly, *CE has no intrinsic optimization component*. High-performing points are found only by repeatedly sampling from the generative model and evaluating $f$. By contrast, our method *explicitly selects high-performing points* using Bayesian optimization. The necessity of repeated sampling in CE makes it only suitable in cases where evaluating $f$ is cheap, which is *not* what we are considering. Moreover, works such as Segler et al. (2018) perform optimization by fine-tuning a generative model on a smaller dataset of high-scoring samples. This can also be viewed as a special case of weighted retraining with binary weights, where the weights are implicitly defined by the number of fine-tuning epochs.

**Bayesian Optimization (BO)** is a technique that maintains a probabilistic model of the objective function, and chooses new points to evaluate based on the modelled distribution of the objective value at unobserved points. BO is widely viewed as the go-to framework for sample-efficient black-box optimization (Brochu et al., 2010; Snoek et al., 2012). However,

---

2 Although methods such as DbAS (Brookes and Listgarten, 2018) generalize these weights to lie in $[0, 1]$, this is determined by the noise of the oracle and therefore will still produce binary weights when $f$ is deterministic, as considered in this paper.
most practical BO models exist for continuous, low-dimensional spaces (Shahriari et al., 2015). Recent works have tried to develop models to extend BO to either structured (Baptista and Poloczek, 2018; Daxberger et al., 2020; Kim et al., 2019; Oh et al., 2019) or high-dimensional (Hoang et al., 2018; Kandasamy et al., 2015; Mutny and Krause, 2018) input spaces. To our knowledge, only BO methods with a significant amount of domain-specific knowledge infused into their design are able to handle input spaces that are both high-dimensional and structured. A noteworthy example is ChemBO which uses both a customized molecular kernel and a synthesis graph to perform BO on molecules (Korovina et al., 2020), which we compare against in Section 4.4. In contrast, our method can be applied to any problem without domain knowledge, and has comparable performance to ChemBO. Finally, in an interesting parallel to our work, Blanchard and Sapsis (2021) use a weighted acquisition function to increase the sample efficiency of BO.

**Reinforcement Learning (RL)** frames optimization problems as Markov decision processes for which an agent learns an optimal policy (Sutton et al., 1998). It has recently been applied to various optimization problems in structured input spaces (Li, 2018b), notably in chemical design (Guimaraes et al., 2017; Olivercrona et al., 2017; Popova et al., 2018; Simm et al., 2020; You et al., 2018; Zhou et al., 2019). While RL is undoubtedly effective at optimization, it is generally extremely sample inefficient, and consequently its biggest successes are in virtual environments where function evaluations are inexpensive (Li, 2018b).

**Conditional Generative Models** Finally, one interesting direction is the development of conditional generative models, which directly produce novel points conditioned on a specific property value (Mirza and Osindero, 2014; Sohn et al., 2015). Although many variants of these algorithms have been applied to real-world problems such as chemical design (Brookes and Listgarten, 2018; Jin et al., 2018b; Kang and Cho, 2018; Li et al., 2018; Lim et al., 2018), the sample efficiency of this paradigm is currently unclear.

### 4.4 Empirical Evaluation

This section aims to empirically answer three main questions:

1. How does weighted training affect the latent space of DGMs? (Section 4.4.1)
2. How do the parameters of weighted retraining influence optimization? (Section 4.4.2)
3. Does weighted retraining compare favourably to existing methods? (Section 4.4.3)
To answer these questions, we perform experiments using three optimization tasks chosen to represent three different data and model types. The tasks are described in more detail below. Because there is no obvious single metric to evaluate sample-efficient optimization, we choose to plot the $K$th best novel evaluated point as a function of the number of objective function evaluations, which we denote as the TopK score (details in Section C.3.3). All plots show the average performance and standard deviation across runs with 5 different random seeds unless otherwise stated. This evaluation method is common practice in Bayesian optimization (Shahriari et al., 2015). It contrasts with previous works which typically report only final scores, and take the maximum across seeds rather than the average (Dai et al., 2018; Gómez-Bombarelli et al., 2018; Jin et al., 2018a; Kusner et al., 2017).

### 2D Shape Area Maximization Toy Task

As a simple toy task that can be easily visualized in 2D, we optimize for the shape with the largest total area in the space of $64 \times 64$ binary images (i.e., the largest number of pixels with value 1). **Data:** A dataset of $\approx 10,000$ squares of different sizes and positions on a $64 \times 64$ background, with a maximum area of 400 (see Fig. C.9 in Section C.3 for examples). **Model:** A convolutional VAE with $Z = \mathbb{R}^2$, as a standard neural network architecture for image modelling. **Latent Optimizer:** We enumerate a grid in latent space over $[-3, +3]^2$, to emulate a perfect optimizer for illustration purposes (this is only feasible since $Z$ is low-dimensional).

### Arithmetic Expression Fitting Task

We follow Kusner et al. (2017) and optimize in the space of single-variable arithmetic expressions generated by a formal grammar. Examples of such expressions are $\sin(2), v/(3+1)$ and $v/2 * \exp(v)/\sin(2*v)$, which are all considered to be functions of some variable $v$. Following Kusner et al. (2017), the objective is to find an expression with minimal mean squared error to the target expression $x^* = 1/3 * v * \sin(v*v)$, computed over 1,000 values of $v$ evenly-spaced between $-10$ and

---

Fig. 4.3 Objective value distribution for the training set and samples from the DGM’s prior for all three tasks for different $k$ values, before and after weighted retraining (see Section 4.4.2).
4.4 Empirical Evaluation

+10. **Data:** 50,000 univariate arithmetic expressions generated by the formal grammar from Kusner et al. (2017). **Model:** A grammar VAE (Kusner et al., 2017), chosen because of its ability to produce only valid grammatical expressions. **Latent Optimizer:** Bayesian optimization with the expected improvement acquisition function (Jones et al., 1998) and a sparse Gaussian process model with 500 inducing points (Titsias, 2009), following Kusner et al. (2017).

**Chemical Design Task**  We follow Gómez-Bombarelli et al. (2018) and optimize the drug properties of molecules. In particular, we consider the standardized task originally proposed in Gómez-Bombarelli et al. (2018) of synthesizing a molecule with maximal penalized *water-octanol partition coefficient* (logP), starting from the molecules in the ZINC250k molecule dataset (Irwin et al., 2012) (see Section C.3.6 for more details). This task has been studied in a long series of papers performing optimization in chemical space, allowing the effect of weighted retraining to be quantitatively compared to other optimization approaches (Dai et al., 2018; Jin et al., 2018a; Kusner et al., 2017; You et al., 2018; Zhou et al., 2019).

**Data:** The ZINC250k molecule dataset (Irwin et al., 2012), using the same train/test split as Jin et al. (2018a). **Model:** A junction tree VAE (Jin et al., 2018a), chosen because it is a state-of-the-art VAE for producing valid chemical structures. For direct comparability to previous results, we use the pre-trained model provided in the code repository of Jin et al. (2018a) as the unweighted model, and create weighted models by fine-tuning the pre-trained model for 1 epoch over the full weighted dataset. **Latent Optimizer:** Same as for the arithmetic expression task.

More details on the experimental setup are given in Section C.3. All experimental data and code to reproduce the experiments can be found at [https://github.com/cambridge-mlg/weighted-retraining](https://github.com/cambridge-mlg/weighted-retraining).

### 4.4.1 Effect of Weighted Training

In this section, we seek to validate some of the conjectures made in Sections 4.1 and 4.2, namely that 1) the latent space of a DGM trained on uniformly weighted data contains many poor-performing points, and 2) that weighted training fixes this by introducing more high-performing points into the latent space. To test this, we train a VAE for each task using rank weighting with a variety of $k$ values (noting that $k = \infty$ corresponds to uniform weighting), initializing the weights using a pre-trained VAE to ensure that the different runs are comparable. We evaluate $f$ on samples from the DGM’s prior for each task, and plot the resulting distributions in Fig. 4.3 in gray with the label *before retraining*. Although the distribution of scores for $k = \infty$ does not exactly match the training distribution for any
Fig. 4.4 Top1 optimization performance of weighted retraining for all tasks, for different $k$ values (i.e. $k \in \{10^{-3}, \infty\}$) and retraining frequencies (i.e. $r_{\text{low}} = 5$ for the 2D shape area task, and $r_{\text{low}} = 50$ for the other two tasks). Shaded area corresponds to standard deviation.

example, it tends to have a similar range, showing that much of the latent space is dedicated to modelling low-scoring points. Weighted training robustly causes the distribution to skew towards higher values at the expense of lower values, which is exactly the intended effect. The upshot is that the result on all 3 tasks broadly supports our conjectures.

4.4.2 Effect of Weighted Retraining Parameters on Optimization

When using rank-weighting from Eq. (4.1) with parameter $k$ and picking a fixed period for model retraining $r$, LSO with weighted retraining can be completely characterized by $k$ and $r$. The baseline of uniform weighting and no retraining is represented by $k = r = \infty$, with decreasing values of $k$ and $r$ representing more skewed weighting and more frequent retraining, respectively. For each task, we choose a value $r_{\text{low}}$ based on our computational retraining budget, then perform LSO for each value of $k \in \{k_{\text{low}}, \infty\}$ and $r \in \{r_{\text{low}}, \infty\}$. For computational efficiency retraining is done via fine-tuning. Further experimental details are given in Section C.3.

The results are shown in Fig. 4.4. Firstly, comparing the case of $k = \infty, r = \infty$ with $k = \infty, r = r_{\text{low}}$ and $k = k_{\text{low}}, r = \infty$ suggests that both weighting and retraining help individually, as hypothesized in Section 4.2. Secondly, in all cases, weighted retraining with $k = k_{\text{low}}, r = r_{\text{low}}$ performs better than all other methods, suggesting that they have a synergistic effect when combined. Note that the performance often increases suddenly after retraining, suggesting that the retraining does indeed incorporate new information into the latent space, as conjectured. Lastly, the objective function values of prior samples from the models after weighted retraining with $r = r_{\text{low}}$ is shown in Fig. 4.3 in blue. In all cases, the distribution becomes more skewed towards positive values, with the difference being more pronounced for lower $k$ values. This suggests that weighted retraining is able to significantly
modify the latent space, even past the initial retraining. See Section C.2 for results with a larger set of $k$ and $r$ values, and TopK plots for other values of $K$.

4.4.3 Comparison with Other Methods

Finally, we compare our proposed method of LSO with weighted retraining with other methods on the same tasks. The first class of methods are based on the cross-entropy method as discussed in Section 4.3, namely design by adaptive sampling (DbAS) (Brookes and Listgarten, 2018), the cross-entropy method with probability of improvement (CEM-PI) (Rubinstein, 1999), the feedback VAE (FBVAE) (Gupta and Zou, 2019) and reward-weighted regression (RWR) (Peters and Schaal, 2007). These methods are noteworthy because they can be viewed as a particular case of weighted retraining, where the weights are binary (except for DbAS) and the latent "optimizer" simply consists of sampling from the DGM’s prior. The hyperparameters of these methods are the sequence of quantiles, and the retraining frequency. We optimize these hyperparameters using a grid search, as detailed in Section C.3.

Figure 4.5 shows the performance of these methods on the best hyperparameter setting found, as a function of the number of samples drawn (with a budget of 5,000 samples in total). We plot the average and standard deviation across 3 random seeds, as we found the variances to be relatively low. We observe that all other forms of weighted retraining perform significantly worse than our own, failing to achieve the performance of our approach, even with an evaluation budget that is an order of magnitude larger than ours (i.e. 5,000 vs 500). We attribute this both to their binary weighting scheme and their lack of a sample-efficient latent optimizer.
Secondly, we compare against other methods in the literature that have attempted the same chemical design task. To our knowledge, the best previously reported score obtained using a machine learning method is 11.84 and was obtained with \( \approx 5,000 \) samples (Zhou et al., 2019). By contrast, our best score is 27.84 and was achieved with only 500 samples. Expanding the scope to include more domain-specific optimization methods, we acknowledge that ChemBO achieved an impressive score of 18.39 in only 100 samples (Korovina et al., 2020), which is better than our method’s performance with only 100 samples. Table C.1 gives a more detailed comparison with other work.

4.5 Discussion

In this chapter, we introduced a new probabilistic deep learning method for tackling data-efficient optimization problems. In particular, we proposed a method for efficient black-box optimization over high-dimensional, structured input spaces, combining latent space optimization with weighted retraining. That is, the optimization is performed in the latent space of a VAE, and the VAE is periodically retrained on a weighted dataset adapt the latent space to new data. We showed that while being conceptually simple and easy to implement on top of previous methods, weighted retraining significantly boosts their efficiency and performance on challenging real-world optimization problems. In the next chapter, we will turn to the problem of neural network calibration, where the goal is to avoid overconfidence in the predictions of neural network models. To this end, we will move away from using VAEs, but will instead focus on the second class of probabilistic deep learning models considered in this thesis, namely Bayesian neural networks (BNNs). In particular, we will propose a new approximate inference method for BNNs that performs inference only over a subset of the model parameters, allowing us to scale to large neural network models.
Chapter 5

Neural Network Calibration via Subnetwork Laplace Approximations

In this chapter, we develop a new probabilistic deep learning method for fixing poor neural network calibration (cf. Section 2.6.3). In particular, our contributions in this chapter are structured as follows:

(a) We propose subnetwork inference, a general framework for scalable Bayesian deep learning in which inference is performed over only a small subset of the NN parameters, while all other parameters are kept deterministic (Section 5.1). This allows us to use expressive posterior approximations that are typically intractable in large NNs.

(b) We present a concrete instantiation of this framework that first fits a MAP estimate of the full NN, and then uses the linearized Laplace approximation to infer a full-covariance Gaussian posterior over a subnetwork, illustrated in Fig. 5.1 (Section 5.2).

(c) We derive a subnetwork selection strategy based on the Wasserstein distance between the approximate posterior for the full network and the approximate posterior for the subnetwork (Section 5.3). For scalability, we employ a diagonal approximation during subnetwork selection. Selecting a small subnetwork then allows us to infer parameter covariances. Empirically, we find that making approximations during subnetwork selection is much less harmful to the posterior predictive than making them during inference.

(d) We review prior work that is related to our proposed subnetwork inference approach (Section 5.4).

(e) We empirically evaluate our method on a range of benchmarks for uncertainty calibration and robustness to distribution shift (Section 5.5). Our experiments demonstrate
that expressive subnetwork inference can outperform popular Bayesian deep learning methods that do less expressive inference over the full NN as well as deep ensembles.

The material in this chapter was previously published in ‘Bayesian Deep Learning via Subnetwork Inference’ (Daxberger et al., 2021c). The research was conducted in collaboration with my co-authors Eric Nalisnick, James Urquhart Allingham, and Javier Antorán, and was supervised by José Miguel Hernández-Lobato throughout. I was involved closely with all aspects of the paper, including the theoretical results, the experiments and the writing of the paper.

### 5.1 Subnetwork Posterior Approximation

In this work, we question the widespread implicit assumption that an expressive posterior approximation must include all $D$ of the model parameters. Instead, we try to perform inference only over a small subset of $S \ll D$ of the parameters. The following arguments motivate this approach:

1. **Overparameterization**: Maddox et al. (2020) have shown that, in the neighborhood of local optima, there are many directions that leave the NN’s predictions unchanged. Moreover, NNs can be heavily pruned without sacrificing test-set accuracy (Frankle and Carbin, 2019). This suggests that the majority of a NN’s predictive power can be isolated to a small subnetwork.
2. **Inference over submodels**: Previous work\(^1\) has provided evidence that inference can be effective even when not performed on the full parameter space. Examples include Izmailov et al. (2019) and Snoek et al. (2015) who perform inference over low-dimensional projections of the parameters, and only the last layer of a NN, respectively.

We therefore combine these two ideas and make the following two-step approximation of the posterior in Eq. (2.13):

\[
p(\theta | D) \approx p(\theta_S | D) \prod_r \delta_{\hat{\theta}_r}(\theta_r)
\]

\[
\approx q(\theta_S) \prod_r \delta_{\hat{\theta}_r}(\theta_r) = q_S(\theta).
\]

The first approximation Eq. (5.1) decomposes the full NN posterior \(p(\theta | D)\) into a posterior \(p(\theta_S | D)\) over the subnetwork \(\theta_S \in \mathbb{R}^S\) and Dirac delta functions \(\delta_{\hat{\theta}_r}(\theta_r)\) over the \(D - S\) remaining parameters \(\theta_r\) to keep them at fixed values \(\hat{\theta}_r \in \mathbb{R}\). Since posterior inference over the subnetwork is still intractable, Eq. (5.2) further approximates \(p(\theta_S | D)\) by \(q(\theta_S)\). However, importantly, if the subnetwork is much smaller than the full network, we can afford to make \(q(\theta_S)\) more expressive than would otherwise be possible. We hypothesize that being able to capture rich dependencies across the parameters within the subnetwork will provide better results than crude approximations applied to the full set of parameters.

**Relationship to Parameter Pruning Methods.** Note that the posterior approximation in Eq. (5.2) can be viewed as pruning the variances of the parameters \(\{\theta_r\}_r\) to zero. This is in contrast to parameter pruning methods (Cheng et al., 2017) that set the parameters themselves to zero. I.e., parameter pruning methods can be viewed as removing parameters to preserve the predictive mean (i.e. to retain accuracy close to the full model). In contrast, subnetwork inference can be viewed as removing just the variances of certain parameters—while keeping their means—to preserve the predictive uncertainty (e.g. to retain calibration close to the full model). Thus, they are complementary approaches. Importantly, by not pruning parameters, subnetwork inference retains the full predictive power of the full NN to retain its predictive accuracy.

### 5.2 Linearized Laplace Subnetwork Inference

In this work we satisfy Eq. (5.2) by approximating the posterior distribution over the parameters with the linearized Laplace approximation (see Section 2.4.2).

\(^1\)See Section 5.4 for a more thorough discussion of related work.
We outline the following procedure for scaling the linearized Laplace approximation to large neural network models within the framework of subnetwork inference.

**Step #1: Point Estimation, Fig. 5.1 (a).** Train a neural network to obtain a point estimate of the parameters, denoted \( \tilde{\theta} \) (if not specified otherwise, this is equivalent to the MAP estimate, i.e., \( \tilde{\theta} = \theta_{\text{MAP}} \)). This can be done using stochastic gradient-based optimization methods (Goodfellow et al., 2016). Alternatively, we could make use of a pre-trained model.

**Step #2: Subnetwork Selection, Fig. 5.1 (b).** Identify a small subnetwork \( \theta_S \in \mathbb{R}^S, S \ll D \). Ideally, we would like to find the subnetwork which produces a predictive posterior ‘closest’ to the full-network’s predictive distribution. Regrettably, reasoning in the space of functions directly is challenging (Burt et al., 2020). Instead, in Section 5.3, we describe a strategy that minimizes the Wasserstein distance between the sub- and full-network’s parameter posteriors.

**Step #3: Bayesian Inference, Fig. 5.1 (c).** Use the GGN-Laplace approximation to infer a full-covariance Gaussian posterior over the subnetwork’s parameters \( \theta_S \in \mathbb{R}^S \),
\[
p(\theta_S | D) \approx q(\theta_S) = \mathcal{N}(\theta_S; \tilde{\theta}_S, G_S^{-1}),
\]
where \( G_S \in \mathbb{R}^{S \times S} \) is the GGN w.r.t. the parameters \( \theta_S \),
\[
G_S := \sum_{n=1}^{N} J_S(x_n) \left( \nabla^2 \log p(y_n | f) |_{f=f_{\theta_S}(x_n)} \right) J_S(x_n)^T + \lambda_S I,
\]
where
\[
J_S(x_n) := \nabla_{\theta_S} f_{\theta_S}(x_n) |_{\tilde{\theta}_S}
\]
is the subnetwork’s Jacobian matrix.

In order to best preserve the magnitude of the predictive variance, we update our prior precision to be \( \lambda_S = \lambda \cdot S/D \) (see Section D.2 for more details). All parameters not belonging to the chosen subnetwork are fixed at their MAP values. Note that this whole procedure (i.e. Steps #1-#3) is a perfectly valid mixed inference strategy: We perform full Laplace inference over the selected subnetwork and MAP inference over all remaining parameters. The resulting approximate posterior Eq. (5.2) is
\[
q_S(\theta) \overset{Eq. (5.3)}{=} \mathcal{N}(\theta_S; \tilde{\theta}_S, G_S^{-1}) \prod_{r} \delta_{\tilde{\theta}_r}(\theta_r).
\]
Given a sufficiently small subnetwork \( \theta_S \), it is feasible to store and invert \( G_S \). In particular, naively storing and inverting the full GGN \( G \) scales as \( \mathcal{O}(D^2) \) and \( \mathcal{O}(D^3) \), respectively. Using the subnetwork GGN \( G_S \) instead reduces this burden to \( \mathcal{O}(S^2) \) and \( \mathcal{O}(S^3) \), respectively.

In our experiments, \( S \ll D \) with our subnetworks representing less than 1% of the total parameters. Note that quadratic/cubic scaling in \( S \) is unavoidable if we are to capture parameter correlations.

**Step #4: Prediction, Fig. 5.1 (d).** Perform a local linearization of the NN (see Section 2.4.2) while fixing \( \theta_r \) to \( \hat{\theta}_r \),

\[
f^{\text{lin}}(x) = f_{\hat{\theta}}(x) + J_S(x)(\theta_S - \hat{\theta}_S). \tag{5.7}
\]

Following Eq. (2.58) and Eq. (2.65), the corresponding predictive distributions are

\[
p(y_s | x_s, D) = \mathcal{N}(y_s; f_{\hat{\theta}}(x_s), \Sigma_S(x_s) + \sigma^2 I) \tag{5.8}
\]

for regression and

\[
p(y_s | x_s, D) \approx \text{softmax} \left( \frac{f_{\hat{\theta}}(x_s)}{\sqrt{1 + \frac{\sigma^2}{S} \text{diag}(\Sigma_S(x_s))}} \right) \tag{5.9}
\]

for classification, where \( \Sigma(x_s) \) in Eq. (2.58) and Eq. (2.65) is substituted with \( \Sigma_S(x_s) = J_S(x_s)^\top G^{-1}_S J_S(x_s) \).

### 5.3 Subnetwork Selection

Ideally, we would like to choose a subnetwork such that the induced predictive posterior distribution is as close as possible to the predictive posterior provided by inference over the full network in Eq. (2.56). This discrepancy between stochastic processes is often quantified through the functional Kullback-Leibler (KL) divergence (Burt et al., 2020; Sun et al., 2019):

\[
\sup_{n \in \mathbb{N}, X \in \mathbb{X}^n} D_{KL}(p_S(y_s | X_s, D) || p(y_s | X_s, D)), \tag{5.10}
\]

where \( p_S \) denotes the subnetwork predictive posterior and \( \mathbb{X}^n \) denotes a finite measurement set of \( n \) elements. Regrettably, reasoning directly in function space is a difficult task (Antóran et al., 2020; Burt et al., 2020; Nalisnick et al., 2021; Nalisnick and Smyth, 2018; Pearce et al., 2019; Sun et al., 2019). Instead we focus our attention on parameter space.
In parameter space, our aim is to minimize the discrepancy between the exact posterior over the full network in Eq. (2.13) and the subnetwork approximate posterior in Eq. (5.2). This provides two challenges. Firstly, computing the exact posterior distribution remains intractable. Secondly, common discrepancies, like the KL divergence or the Hellinger distance, are not well defined for the Dirac delta distributions found in Eq. (5.2).

To solve the first issue, we again resort to local linearization, introduced in Section 2.4.2. The true posterior for the linearized model is Gaussian or approximately Gaussian:

$$p(\theta | D) \simeq \mathcal{N}(\theta; \hat{\theta}, G^{-1}).$$  

(5.11)

We solve the second issue by choosing the squared 2-Wasserstein distance, which is well defined for distributions with disjoint support. For the case of a full covariance Gaussian in Eq. (5.11) and a product of a full covariance Gaussian with Dirac deltas in Eq. (5.6), this metric takes the following form:

$$W_2^2(p(\theta | D), q_S(\theta))^2 = \text{Tr} \left( G^{-1} + G_{S+1}^{-1} - 2 \left( G_{S+1}^{-1/2} G^{-1} G_{S+1}^{-1/2} \right)^{1/2} \right),$$

(5.12)

where the covariance matrix $G_{S+1}^{-1}$ is equal to $G_S^{-1}$ padded with zeros at the positions corresponding to $\theta_r$, matching the shape of $G^{-1}$. See Section D.1 for details.

Finding the subset $\theta_S \in \mathbb{R}^S$ of size $S$ that minimizes Eq. (5.12) would be combinatorially difficult, as the contribution of each parameter depends on every other parameter. To address this issue, we make an independence assumption among parameters, resulting in the simplified objective

$$W_2^2(p(\theta | D), q_S(\theta))^2 \approx \sum_{d=1}^{D} \sigma_d^2 (1 - m_d),$$

(5.13)

where $\sigma_d^2$ is the marginal variance of the $d$th parameter, and $m_d = 1$ if $\theta_d \in \theta_S$ and 0 otherwise (see Section D.1). The objective Eq. (5.13) is trivially minimized by a subnetwork containing the $S$ parameters with highest variances. This is related to common magnitude-based parameter pruning methods (Cheng et al., 2017). The main difference is that our selection strategy involves parameter variances rather than magnitudes as we target predictive uncertainty rather than accuracy.

In practice, even computing the marginal variances (i.e. the diagonal of $G^{-1}$) is intractable, as it requires storing and inverting the GGN $G$. However, we can approximate posterior

---

2When not making predictions with the linearized model, the Gaussian posterior would represent a crude approximation.
marginal variances with the diagonal Laplace approximation \(\text{diag}(G^{-1}) \approx \text{diag}(G)^{-1}\) (Denker and LeCun, 1990; Kirkpatrick et al., 2017), diagonal SWAG (Maddox et al., 2019b), or even mean-field variational inference (Blundell et al., 2015; Osawa et al., 2019). In this work we rely on the former two, as the the latter involves larger overhead.

It may seem that we have resorted to the poorly performing diagonal assumptions that we sought to avoid in the first place (Ashukha et al., 2020; Foong et al., 2019a; Ovadia et al., 2019). However, there is a key difference. We make the diagonal assumption during subnetwork selection rather than inference; we do full covariance inference over \(\theta_S\). In Section 5.5, we provide evidence that making a diagonal assumption during subnetwork selection is reasonable by showing that 1) it is substantially less harmful to predictive performance than making the same assumption during inference, and 2) it outperforms random subnetwork selection.

5.4 Related Work

We note that Farquhar et al. (2020) argue that in deeper networks the mean-field assumption should not be restrictive. Our empirical results seem to contradict this proposition. We find that scaling up approximations that do consider weight correlations, e.g., Louizos and Welling (2016); MacKay (1992d); Maddox et al. (2019b); Ritter et al. (2018b), by lowering the dimensionality of the weight space outperforms diagonal approximations. We conclude that more research is warranted in this area.

Neural Network Linearization In the limit of infinite width, NNs converge to Gaussian process (GP) behaviour (Garriga-Alonso et al., 2018; Matthews, 2017; Neal, 1995). Recently, these results have been extended to finite width BNNs when the surrogate posterior is Gaussian (Khan et al., 2019). We draw upon these results to formulate a subnetwork selection strategy for BNNs.

Neural Linear Methods These represent a generalised linear model in which the basis functions are defined by the \(L-1\) first layers of a NN. That is, neural linear methods perform inference over only the last layer of a NN, while keeping all other layers fixed (Kristiadi et al., 2020; Ober and Rasmussen, 2019; Ovadia et al., 2019; Pinsler et al., 2019; Riquelme et al., 2018; Snoek et al., 2015). They can also be viewed as a special case of subnetwork inference, in which the subnetwork is simply defined to be the last NN layer.
Inference Over Subspaces  The subfield of NN pruning aims to increase the computational efficiency of NNs by identifying the smallest subset of weights which are required to make accurate predictions; see, e.g., Frankle and Carbin (2019); Wang et al. (2020). Approaches trade-off computational cost with compression efficiency, ranging from those that require multiple training runs (Frankle and Carbin, 2019) to those that prune before training (Wang et al., 2020). Our work differs in that it retains all NN weights but aims to find a small subset over which to perform probabilistic reasoning post training. More closely related work to ours is that of Izmailov et al. (2019), who propose to perform inference over a low-dimensional subspace of weights; e.g., one constructed from the principal components of the SGD trajectory. Moreover, several recent approaches use low-rank parameterizations of approximate posteriors in the context of variational inference (Dusenberry et al., 2020; Rossi et al., 2020; Swiatkowski et al., 2020). This could also be viewed as doing inference over an implicit subspace of weight space. In contrast, we propose a technique to find subsets of weights which are relevant to predictive uncertainty, i.e., we identify axis aligned subspaces. Relatedly, a recent follow-up work also studies the question of whether BNNs need to be fully stochastic (Sharma et al., 2023a).

Bayesian Sparsity  Finally, there have been recent works studying NN sparsity / pruning from a Bayesian perspective (Cui et al., 2020; Ghosh and Doshi-Velez, 2019; Gomez et al., 2019; Lee et al., 2018; Louizos et al., 2018; Molchanov et al., 2017; Polson and Ročková, 2018; van Baalen et al., 2020). Their goal is fundamentally different to ours: those methods aim to perform model selection by either explicitly or implicitly pruning unnecessary weights. Our goal is to approximate inference in large models.

5.5 Empirical Evaluation

We empirically assess the effectiveness of subnetwork inference compared to methods that do less expressive inference over the full network as well as state-of-the-art methods for uncertainty quantification in deep learning. We consider three benchmark settings: (a) small-scale toy regression (Section 5.5.1), (b) medium-scale tabular regression (Section 5.5.2), and (c) image classification with ResNet-18 (Section 5.5.3). We use the log-likelihood as the main metric for model calibration (Guo et al., 2017). Our proposed subnetwork (linearized) Laplace method is implemented within the laplace PyTorch library (Daxberger et al., 2021b). Further experimental results and setup details are presented in Section D.3 and Section D.4, respectively.
5.5 Empirical Evaluation

Fig. 5.2 Predictive distributions (mean ± std) for 1D regression. The numbers in parentheses denote the number of parameters over which inference was done (out of 2,600 in total). The blue box highlights subnetwork inference using Wasserstein (top) and random (bottom) subnetwork selection. Wasserstein subnetwork inference maintains richer predictive uncertainties at smaller parameter counts.

5.5.1 How Does Subnetwork Inference Preserve Predictive Uncertainty?

We first assess how the predictive distribution of a full-covariance Gaussian posterior over a selected subnetwork qualitatively compares to that obtained from 1) a full-covariance Gaussian over the full network (Full Cov), 2) a factorised Gaussian posterior over the full network (Diag), 3) a full-covariance Gaussian over only the (Final layer) of the network (Snoek et al., 2015), and 4) a point estimate (MAP). For subnetwork inference, we consider both Wasserstein (Wass) (as described in Section 5.3) and uniform random subnetwork selection (Rand) to obtain subnetworks that comprise of only 50%, 3% and 1% of the model parameters. For this toy example, it is tractable to compute exact posterior marginal variances to guide subnetwork selection.

Our NN consists of 2 ReLU hidden layers with 50 hidden units each. We employ a homoscedastic Gaussian likelihood function where the noise variance is optimised with maximum likelihood. We use GGN-Laplace inference over network weights (not biases) in combination with the linearized predictive distribution in Eq. (5.8). Thus, all approaches considered share their predictive mean, allowing better comparison of their uncertainty estimates. We set the full network prior precision to $\lambda = 3$ (a value which we find to work well empirically) and set $\lambda_S = \lambda \cdot S / D$.

We use a synthetic 1D regression task with two separated clusters of inputs (Antorán et al., 2020), allowing us to probe for ‘in-between’ uncertainty (Foong et al., 2019b). Results are shown in Fig. 5.2. Subnetwork inference preserves more of the uncertainty of full
network inference than diagonal Gaussian or final layer inference while doing inference over fewer weights. By capturing weight correlations, subnetwork inference retains uncertainty in between clusters of data. This is true for both random and Wasserstein subnetwork selection. However, the latter preserves more uncertainty with smaller subnetworks. Finally, the strong superiority to diagonal Laplace shows that making a diagonal assumption for subnetwork selection but then using a full-covariance Gaussian for inference (as we do) performs significantly better than making a diagonal assumption for the inferred posterior directly (cf. Section 5.3). These results suggest that expressive inference over a carefully selected subnetwork retains more predictive uncertainty than crude approximations over the full network.

![Fig. 5.3 Mean test log-likelihood values obtained on UCI datasets across all splits.](image_url)

5.5.2 Subnetwork Inference in Large Models vs Full Inference in Small Models

Secondly, we study how subnetwork inference in larger NNs compares to full network inference in smaller ones. We explore this by considering 4 fully connected NNs of increasing size. These have numbers of hidden layers $h_d = \{1, 2\}$ and hidden layer widths $w_d = \{50, 100\}$. For a dataset with input dimension $i_d$, the number of weights is given by $D = (i_d + 1)w_d + (h_d - 1)w_d^2$. Our 2 hidden layer, 100 hidden unit NNs have a weight count of the order $10^4$. Full covariance inference in these NNs borders the limit of computational tractability on commercial hardware. We first obtain a MAP estimate of each NN’s weights and our homoscedastic likelihood function’s noise variance. We then perform full network GGN-Laplace inference for each NN. We also use our proposed Wasserstein rule to prune every NN’s weight variances such that the number of variances that remain matches the size...
5.5 Empirical Evaluation

of every smaller NN under consideration. We employ the diagonal Laplace approximation to cheaply estimate posterior marginal variances for subnetwork selection. We employ the linearization in Eq. (2.58) and Eq. (5.8) to compute predictive distributions. Consequently, NNs with the same number of weights make the same mean predictions. Increasing the number of weight variances considered will thus only increase predictive uncertainty.

We employ 3 tabular datasets of increasing size (input dimensionality; num. points): wine (11; 1,439), kin8nm (8; 7,373) and protein (9; 41,157). We consider their standard train-test splits (Hernández-Lobato and Adams, 2015) and their gap variants (Foong et al., 2019b), designed to test for out-of-distribution uncertainty. Details are provided in Section D.4.4. For each split, we set aside 15% of the train data as a validation set. We use these for early stopping when finding MAP estimates and for selecting the weights’ prior precision. We keep other hyperparameters fixed across all models and datasets. Results are shown in Fig. 5.3.

We present mean test log-likelihood (LL) values, as these take into account both accuracy and uncertainty. Larger ($w_d = 100, h_d = 2$) models tend to perform best when combined with full network inference, although Wine-gap and Protein-gap are exceptions. Interestingly, these larger models are still best when we perform inference over subnetworks of the size of smaller models. We conjecture this is due to an abundance of degenerate directions (i.e., weights) in the weight posterior NN models (Maddox et al., 2020). Full network inference in small models captures information about both useful and non-useful weights. In larger models, our subnetwork selection strategy allows us to dedicate a larger proportion of our resources to modelling informative weight variances and covariances. In 3 out of 6 datasets, we find abrupt increases in LL as we increase the number of weights over which we perform inference, followed by a plateau. Such plateaus might be explained by most of the informative weight variances having already been accounted for. Considering that the cost of computing the GGN dominates that of NN training, these results suggest that, **given the same amount of compute, it is better to perform subnetwork inference in larger models than full network inference in small ones.**

5.5.3 Image Classification Under Distribution Shift

We now assess the robustness of large convolutional neural networks with subnetwork inference to distribution shift on image classification tasks compared to the following baselines: point-estimated networks (MAP), Bayesian deep learning methods that do less expressive inference over the full network: MC Dropout (Gal and Ghahramani, 2016), diagonal Laplace, VOGN (Osawa et al., 2019) (all of which assume factorisation of the weight posterior), and SWAG (Maddox et al., 2019b) (which assumes a diagonal plus low-rank posterior). We also benchmark deep ensembles (Lakshminarayanan et al., 2017). The latter is considered
Fig. 5.4 Results on the rotated MNIST (left) and the corrupted CIFAR (right) benchmarks, showing the mean ± std of the error (top) and log-likelihood (bottom) across three different seeds. Subnetwork inference retains better uncertainty calibration and robustness to distribution shift than point-estimated networks and other Bayesian deep learning approaches. See Section D.3 for ECE and Brier score results.

state-of-the-art for uncertainty quantification in deep learning (Ashukha et al., 2020; Ovadia et al., 2019). We use ensembles of 5 NNs, as suggested by (Ovadia et al., 2019), and 16 samples for MC Dropout, diagonal Laplace and SWAG. We use a Dropout probability of 0.1 and a prior precision of $\lambda = 4 \times 10^4$ for diagonal Laplace, found via grid search. We apply all approaches to ResNet-18 (He et al., 2016), which is composed of an input convolutional block, 8 residual blocks and a linear layer, for a total of 11,168,000 parameters.

For subnetwork inference, we compute the linearized predictive distribution in Eq. (5.9). We use Wasserstein subnetwork selection to retain only 0.38% of the weights, yielding a subnetwork with only 42,438 weights. This is the largest subnetwork for which we can tractably compute a full covariance matrix. Its size is $42,438^2 \times 4 \text{ Bytes} \approx 7.2 \ \text{GB}$. We use diagonal SWAG (Maddox et al., 2019b) to estimate the marginal weight variances needed for subnetwork selection. We tried diagonal Laplace but found that the selected weights where those where the Jacobian of the NN evaluated at the train points was always zero (i.e., dead ReLUs). The posterior variance of these weights is large as it matches the prior. However, these weights have little effect on the NN function. SWAG does not suffer from this problem as it disregards weights with zero training gradients. We use a prior precision of $\lambda = 500$, found via grid search.

To assess to importance of principled subnetwork selection, we also consider the baseline where we select the subnetwork uniformly at random (called Ours (Rand)). We perform the following two experiments, with results in Fig. 5.4.
5.5 Empirical Evaluation

![Graphs showing log-likelihoods and memory footprints](image)

(a) Rotated MNIST  
(b) Corrupted CIFAR10

<table>
<thead>
<tr>
<th>Subnet Size</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.2M (100%)</td>
<td>500TB</td>
</tr>
<tr>
<td>40K (0.36%)</td>
<td>6.4GB</td>
</tr>
<tr>
<td>20K (0.18%)</td>
<td>1.5GB</td>
</tr>
<tr>
<td>10K (0.09%)</td>
<td>380MB</td>
</tr>
<tr>
<td>5K (0.04%)</td>
<td>95MB</td>
</tr>
<tr>
<td>1K (0.01%)</td>
<td>4.0MB</td>
</tr>
<tr>
<td>100 (0.001%)</td>
<td>40KB</td>
</tr>
</tbody>
</table>

(c) Memory Footprints

Fig. 5.5 Log-likelihoods of our method with subnetwork sizes between 100-40K using ResNet-18 on (a) rotated MNIST and (b) corrupted CIFAR10, vs. Ensembles and Diagonal Laplace, and (c) respective covariance matrix memory footprints. For all subnetwork sizes, we use the same hyperparameters as in Section 5.5.3 (i.e., no individual tuning per size). The smaller subnetwork GGNs are simply constructed as submatrices of the original subnetwork GGN of size 40K. Performance degrades smoothly with subnetwork size, but our method retains strong calibration even with very small subnetworks (requiring only marginal extra memory).

Rotated MNIST: Following Antorán et al. (2020); Ovadia et al. (2019), we train all methods on MNIST and evaluate their predictive distributions on increasingly rotated digits. While all methods perform well on the original MNIST test set, their accuracy degrades quickly for rotations larger than 30 degrees. In terms of LL, ensembles perform best out of our baselines. Subnetwork inference obtains significantly larger LL values than almost all baselines, including ensembles. The only exception is VOGN, which achieves slightly better performance. It was also observed in Ovadia et al. (2019) that mean-field variational inference (which VOGN is an instance of) is very strong on MNIST, but its performance deteriorates on larger datasets. Subnetwork inference makes accurate predictions in-distribution while assigning higher uncertainty than the baselines to out-of-distribution points.
Corrupted CIFAR: Again following Antorán et al. (2020); Ovadia et al. (2019), we train on CIFAR10 and evaluate on data subject to 16 different corruptions with 5 levels of intensity each (Hendrycks and Dietterich, 2019b). Our approach matches a MAP estimated network in terms of predictive error as local linearization makes their predictions the same. Ensembles and SWAG are the most accurate. Even so, subnetwork inference differentiates itself by being the least overconfident, outperforming all baselines in terms of log-likelihood at all corruption levels. Here, VOGN performs rather badly; while this might appear to contrast its strong performance on the MNIST benchmark, the behaviour that mean-field VI performs well on MNIST but poorly on larger datasets was also observed in Ovadia et al. (2019).

Furthermore, on both benchmarks, we find that randomly selecting the subnetwork performs substantially worse than using our more sophisticated Wasserstein subnetwork selection strategy. This highlights the importance of the way the subnetwork is selected. Overall, these results suggest that subnetwork inference results in better uncertainty calibration and robustness to distribution shift than other popular uncertainty quantification approaches.

What about smaller subnetworks? One might wonder if a subnetwork of ~40K weights is actually necessary. In Fig. 5.5, we show that one can also retain strong calibration with significantly smaller subnetworks. Full covariance inference in a ResNet-18 would require storing ~11.2M² params (~500TB). Subnet inference reduces the cost (on top of MAP) to as little as 1K² params (~4.0MB) while remaining competitive with deep ensembles. This suggests that subnetwork inference can allow otherwise intractable inference methods to be applied to even larger NNs.

5.6 Discussion

In this chapter, we developed a new probabilistic deep learning method for mitigating overconfidence and poor calibration in neural networks. To this end, we proposed to perform Bayesian inference over only a small subset of the model parameters, while keeping all other parameters at point estimates. We combined the idea of subnetwork inference with the linearized Laplace approximation to obtain a new scalable variant of the Laplace approximation. We derived a strategy for selecting subnetworks that aims to recover the predictive uncertainty induced by the full posterior. Empirically, we found that subnetwork Laplace compared favourably against relevant baselines. In the next chapter, we will consider the continual deep learning problem, where the goal is to enable neural networks to learn sequentially on multiple tasks without forgetting previously-learned tasks. To this end, we will develop a
probabilistic technique that aims to recover the gradients of the past training objective to mitigate forgetting.
Chapter 6

Continual Deep Learning via Gradient Reconstruction of the Past

In this chapter, we develop a new probabilistic deep learning method to enable continual deep learning (cf. Section 2.6.4). In particular, our contributions in this chapter are structured as follows:

(a) We define the continual learning problem and introduce three popular approaches to tackle continual learning problems, namely weight regularization, experience replay, and functional regularization (Section 6.1).

(b) We describe the Knowledge-adaptaion prior (K-prior), a recently-proposed principle of adaptation that allows us to reconstruct the gradients of the past training objective, and outline its limitations when it comes to addressing continual learning problems (Section 6.2).

(c) We develop our proposed method which improves upon K-priors by combining it with principles from weight regularization and experience replay in a principled way, aiming to reduce the gradient reconstruction error to enable continual learning (Section 6.3).

(d) We review relevant prior work, including the different continual learning paradigms (i.e., weight-regularization-based, memory/rehearsal-based, and architecture-based methods) as well as continual learning settings (i.e., task-incremental, domain-incremental, and class-incremental learning) (Section 6.4).

(e) We present an empirical study that showcases the efficacy and scalability of our proposed method across various memory sizes on several continual learning benchmarks, compared to relevant baselines (Section 6.5).
We focus on a continual learning (CL) problem to incrementally learn from a sequence of data sets \(D_1, \ldots, D_T\), corresponding to a total of \(T\) tasks. This is different from the commonly used batch-training in deep learning, where data from all the tasks is assumed to be available at all times during training. CL is challenging as the model needs to repeatedly adapt to new tasks. This is particularly difficult when the model needs to remember past knowledge while learning new tasks.

### 6.1 Continual Learning Methods

We focus on a continual learning (CL) problem to incrementally learn from a sequence of data sets \(D_1, \ldots, D_T\), corresponding to a total of \(T\) tasks. This is different from the commonly used batch-training in deep learning, where data from all the tasks is assumed to be available at all times during training. CL is challenging as the model needs to repeatedly adapt to new tasks.

**Previous Methods**

The material in this chapter was previously published in ‘Improving Continual Learning by Accurate Gradient Reconstructions of the Past’ (Daxberger et al., 2023). The research was conducted in collaboration with my co-authors Siddharth Swaroop, Kazuki Osawa, Rio Yokota, Richard E. Turner, José Miguel Hernández-Lobato, and was supervised by Mohammad Emtiyaz Khan throughout. I was involved closely with all aspects of the paper, including the theoretical results, the experiments and the writing of the paper.

**Our Method**

The principle of gradient reconstructions, we design a new prior (right) to combine different types of regularization and memory-based methods. EWC uses a quadratic regularizer based on the old weight-vector and its importance, while experience replay uses a memory of past examples along with their labels. Function regularization often does not require the labels, but also lacks a weight regularizer. Our method combines these different types of methods, using two memory sets, \(M_1 \cup M_2 = M\). The notable differences are that our method’s importance matrix excludes the examples in the entire memory set \(M\), and that experience replay is applied only to the second memory set \(M_2\). The examples in the first memory set \(M_1\) do not require labels, and can be compressed to only keep a small set of representative inputs that may not be part of the old training sets. See Section 6.3 for details on our method and the choice of memory.

**Fig. 6.1** Using the principle of gradient reconstructions, we design a new prior (right) to combine different types of regularization and memory-based methods. EWC uses a quadratic regularizer based on the old weight-vector and its importance, while experience replay uses a memory of past examples along with their labels. Function regularization often does not require the labels, but also lacks a weight regularizer. Our method combines these different types of methods, using two memory sets, \(M_1 \cup M_2 = M\). The notable differences are that our method’s importance matrix excludes the examples in the entire memory set \(M\), and that experience replay is applied only to the second memory set \(M_2\). The examples in the first memory set \(M_1\) do not require labels, and can be compressed to only keep a small set of representative inputs that may not be part of the old training sets. See Section 6.3 for details on our method and the choice of memory.
6.1 Continual Learning Methods

tasks, while not forgetting previously-gathered knowledge. Our goal is to get the performance as close as possible to the model that is trained on data from all tasks.

Formally, consider a supervised learning problem with $\mathcal{D}_t$ containing $N$ input-output pairs $(x_n, y_n)$, with $x_n \in \mathbb{R}^D$ and $y_n \in \mathcal{Y}$, and we wish to train a model with output $f_{\theta}(x_n)$ (also denoted as $f_{\theta}^n$), and a $D$-length parameter $\theta$ in a space $\Theta \subset \mathbb{R}^D$. Then, at any given task $t$, the best possible model parameters $\theta_t^*$ can be obtained by training on all the data examples $\mathcal{D}_{1:t} = \bigcup_{n=1}^{t} \mathcal{D}_n$, for example, by solving the optimization problem shown below,

$$\theta_t^* = \arg\min_{\theta \in \Theta} \ell_t^{\text{batch}}(\theta),$$

where $\ell_t^{\text{batch}}(\theta) = \sum_{n \in \mathcal{D}_{1:t}} \ell(y_n, \sigma(f_{\theta}^n)) + \mathcal{R}(\theta).$ (6.2)

We assume the loss $\ell(y_n, \sigma(f_n))$ is defined through the log-likelihood of an exponential family distribution (e.g., cross-entropy), with $\sigma(f)$ being a transformation of the model outputs (e.g., softmax) defined using the link-function of the distribution. We denote a regularizer by $\mathcal{R}(\theta)$, and in what follows, we will use an $L_2$ regularizer $\mathcal{R}(\theta) = \frac{1}{2} \delta \|\theta\|^2$, with $\delta \geq 0$.

The main challenge in CL is to remember the useful past knowledge extracted from previous tasks $\mathcal{D}_{1:t-1}$, and reuse it during training over the new task $\mathcal{D}_t$ to get as close as possible to $\theta_t^*$. We would like a compact summary of the past knowledge, because we are not allowed to store all past data. We will now describe two strategies used in the literature, based on regularization and experience replay, and discuss the challenges in combining them.

Regularization-based approaches aim to keep the new weight-vector close to the old one, hoping that it will help to avoid forgetting and facilitate knowledge reuse. The most common is a weight-regularization technique known as Elastic-Weight Consolidation (EWC) (Kirkpatrick et al., 2017), which, at a task $t$, minimizes the following objective using the previous weight $\theta_{t-1}$,

$$\ell_t^{\text{weight}}(\theta) = \sum_{n \in \mathcal{D}_t} \ell(y_n, \sigma(f_{\theta}^n)) + \frac{\lambda}{2} (\theta - \theta_{t-1})^T F_{t-1}(\theta - \theta_{t-1}),$$

where the second term is a quadratic regularizer with a weight-importance matrix $F_{t-1}$, and $\lambda \geq 0$ is a trade-off hyperparameter. The simplest choice of $F_{t-1}$ is to use the diagonal of a generalized Gauss-Newton (GGN) matrix (Martens, 2020). The GGN is defined as

$$G(\mathcal{D}_{1:t}) := \sum_{n \in \mathcal{D}_{1:t}} [\nabla f_{\theta}^n] \sigma'(f_{\theta}^n) [\nabla f_{\theta}^n]^T,$$ (6.4)
where $\nabla f^\theta t = \nabla_{\theta} f_{\theta}(x_n)|_{\theta} = : J_t(x_n)$ denotes the Jacobian, that is, derivative of $f_{\theta}(x_n)$ with respect to $\theta$ at $\theta = \theta_t$, and $\sigma'(f^\theta_t) = \nabla_f^2 \log p(y_n|f) |_{f=f^\theta_t(x_n)}$. This choice of $F_{t-1}$ corresponds to a (diagonal) Laplace approximation over the model parameters (Huszár, 2018) (see also Section 2.4.2).

Often, the regularizer $\delta$ is also added to the GGN matrix to reduce ill-conditioning. Computation of $F_{t-1}$ can be done in an online fashion as the training proceeds over tasks (Schwarz et al., 2018). This method uses a compact representation of the past knowledge as we need to only store $\theta_{t-1}$ and the diagonal matrix $F_{t-1}$, requiring $O(P)$ memory. The method is also simple to implement within deep-learning codebases, requiring relatively little overhead. Other choices are possible for the importance matrix (Aljundi et al., 2018; Benzing, 2022; Zenke et al., 2017).

An alternative to regularization-based methods is based on experience replay (Robins, 1995; Shin et al., 2017). Such methods store a subset of past data in a memory $M_t$ and add it to the new data during training:

$$\ell^\text{er}_t(\theta) = \sum_{n \in D_t \cup M_{t-1}} \ell(y_n, \sigma(f^\theta_t)).$$

This method can be accurate when the memory represents the data well, but this often requires a large memory that grows with the number of tasks.

A popular approach to combine these two previous approaches is to use functional regularization (Pan et al., 2020; Titsias et al., 2020) where, instead of regularizing the weights, we regularize the function outputs at a few past input locations stored in a memory:

$$\ell^\text{func}_t(\theta) = \sum_{n \in D_t} \ell(y_n, \sigma(f^\theta)) + \frac{\lambda}{2} \sum_{j \in M_{t-1}} \ell(\sigma(f^\theta_{t-1}), \sigma(f^\theta_j)).$$

Some methods implement this via knowledge-distillation (Buzzega et al., 2020; Rebuffi et al., 2017). Some simply use the original loss while others use the squared loss (Benjamin et al., 2019). An advantage of functional regularization is that it does not require the labels associated with the inputs in $M_t$, which enables the use of an arbitrary input $x$ which is not restricted to be from $D_{1:t}$. For example, we can use a deep generative model to generate pseudo-inputs (Shin et al., 2017), or learn them as in sparse Gaussian processes (Titsias, 2009).

Overall, we see that each method has its own complementary strengths: Weight-regularization is compact, experience replay can be accurate, and functional regularization can use arbitrary
memory inputs. Combining these approaches can strike a good balance between performance and memory size, but at present, little has been done to find principled ways to combine them. One could simply add them together, but there are many choices one needs to make. For example, how should we choose the importance matrix, the memory set, and the specific forms of the regularizers? Our goal in this paper is to provide a principled approach to answer such questions.

### 6.2 A Principle of Adaptation: Gradient Reconstruction of the Past

We will use the recently proposed principle of adaptation by Khan and Swaroop (2021) to combine and improve the CL strategies discussed in Section 6.1. The principle suggests to reconstruct the gradient of the past objective by using a combination of weight and function-space regularizers. Specifically, at task $t$, we consider minimizing (for some $\tau > 0$)

$$
\ell^{K\text{-prior}}_t(\theta) = \sum_{n \in D_t} \ell(y_n, \sigma(f^n_\theta)) + \tau K_t(\theta; \theta_{t-1}, M_{t-1}),
$$

where $K_t(\theta; \theta_{t-1}, M_{t-1})$ is a regularizer that combines a weight-space regularizer using $\theta_t$ and a function-space regularizer over the memory $M_{t-1}$. The regularizer is called the Knowledge-adaptation prior (K-prior), and is designed with the goal to minimize the gradient reconstruction error of the past training objective. We will use $\tau = 1$ unless noted otherwise.

Specifically, at task $t$, the past training objective is $\ell^{\text{batch}}_{t-1}(\theta)$, and we want to design the prior to minimize the magnitude of the gradient error for all $\theta$:

$$
e_t(\theta) := \nabla \ell^{\text{batch}}_{t-1}(\theta) - \nabla K_t(\theta; \theta_{t-1}, M_{t-1}).
$$

The loss $\ell^{\text{batch}}_{t-1}$ depends on all the past data $D_{1:t-1}$, and our goal is to reconstruct its gradient by using the weight vector $\theta_{t-1}$ and a memory $M_{t-1}$. Khan and Swaroop (2021) showed that many existing adaptive strategies in machine learning for one-step adaptation tasks can be recovered from this principle. For example, the following K-prior with $M_{t-1} = D_{1:t-1}$ gives zero error $e_t(\theta)$ for generalized-linear models $f^n_\theta = x_n^T \theta$:

$$
K_t(\theta; \theta_{t-1}, M_{t-1}) = \sum_{n \in M_{t-1}} \ell\left(\sigma(f^n_{\theta_{t-1}}), \sigma(f^n_\theta)\right) + \frac{\delta}{2}(\theta - \theta_{t-1})^T(\theta - \theta_{t-1}),
$$
Continual Deep Learning via Gradient Reconstruction of the Past

where $\delta \geq 0$ is the $L_2$ regularization constant. The K-prior does not use the labels, just like the functional regularization discussed earlier, yet the gradient can be reconstructed by using the predictions at the inputs locations $x_n \in D_{1:t-1}$. Many other similar results are shown by Khan and Swaroop (2021) for other models, such as support vector machines, Gaussian processes, knowledge distillation, functional-regularization, and the memory-based methods used in continual learning.

While promising, there are still multiple issues with the work of Khan and Swaroop (2021), which we fix in this paper. First, they did not consider the sequential setup such as continual learning. Second, the error incurred by their K-prior, of form Eq. (6.8), is non-zero for neural networks: see Khan and Swaroop (2021, Sec. 4.2). For the continual learning problem, this can be disastrous because errors can accumulate quickly over tasks, deteriorating performance. Third, the weight-regularizer in Eq. (6.8) ignores the weight importance, which is commonly used in other works (Kirkpatrick et al., 2017; Ritter et al., 2018a; Schwarz et al., 2018) and can improve performance.

In this chapter, we will fix these issues, combining and improving regularization and memory-based methods. We will start with functional-regularization, and then add a weight regularizer and experience replay to decrease its error. This will yield a prior with lower error than each individual method on its own.

6.3 A New Improved K-Prior

Using the principle described in the previous section, we will now design a prior that combines a weight regularizer, a functional regularizer, and experience replay (Fig. 6.1). As described in the previous section, our goal is to minimize the gradient reconstruction error of the past training objective, i.e., the difference in the gradient of the loss for the full batch method and the K-priors loss with limited memory, see Eq. (6.7). The functional regularizer is based on knowledge distillation, and combined with an EWC-style quadratic regularizer which uses a specific importance vector to minimize the error in the K-prior of Eq. (6.8). An additional experience-replay term further reduces the error by storing the labels for a subset of the memory set. We will see that each piece contributes to the reduction of a different type of error, and the combination overall gives lower gradient reconstruction error than each individual component.
6.3.1 The Error in the K-Prior When Using a Limited Memory

We start by analyzing the gradient error in the K-prior of Eq. (6.8) when it is defined with a limited memory $\mathcal{M}_{t-1}$, instead of the full data $\mathcal{D}_{1:t-1}$. As shown in Section E.1, the error is given as follows,

$$
e_t(\theta) = \sum_{n \in \mathcal{D}_{1:t-1} \setminus \mathcal{M}_{t-1}} \nabla f_{\theta}^n \left[ \sigma(f_{\theta}^n) - \sigma(f_{\theta_{t-1}}^n) \right] + \sum_{n \in \mathcal{D}_{1:t-1}} \nabla f_{\theta}^n v_{\theta_{t-1}}^n + \delta_{\theta_{t-1}}, \quad (6.9)$$

where $r_{\theta_{t-1}}^n = \sigma(f_{\theta_{t-1}}^n) - y_n$ is the residual. The first error term $e_t^{\text{mem}}$ arises due to the use of limited memory $\mathcal{M}_{t-1}$, and can be reduced to zero by increasing the memory size to include all past input examples. In contrast, the second error term $e_t^{\text{NN}}$ arises due to the use of neural networks, and reduces only when the network gets better in predicting the past data $\mathcal{D}_{1:t-1}$, that is, when the residuals go to zero. We will now show that $e_t^{\text{mem}}$ can be reduced by adding an EWC-style weight regularizer, while $e_t^{\text{NN}}$ can be reduced by adding an experience replay term with a specific memory.

6.3.2 Reducing $e_t^{\text{mem}}$ Using an EWC-Style Regularizer

The error $e_t^{\text{mem}}$ can be reduced by using a first-order Taylor approximation of $\sigma(f_{\theta_{t-1}}^n)$ at $\theta_{t-1}$,

$$
\sigma(f_{\theta_{t-1}}^n) \approx \sigma(f_{\theta_{t-1}}^n) + \sigma'(f_{\theta_{t-1}}^n)(\nabla f_{\theta_{t-1}}^n)^\top(\theta - \theta_{t-1}). \quad (6.10)
$$

Plugging this into the definition of the error, we get

$$
e_t^{\text{mem}} \approx \left[ \sum_{n \in \mathcal{D}_{1:t-1} \setminus \mathcal{M}_{t-1}} \nabla f_{\theta_{t-1}}^n \sigma'(f_{\theta_{t-1}}^n)(\nabla f_{\theta_{t-1}}^n)^\top \right] (\theta - \theta_{t-1}),
$$

where we use the definition of the GGN matrix given in Eq. (6.4), and approximate $\nabla f_{\theta_{t-1}}^n \approx \nabla f_{\theta_{t}}$. The right hand side is equal to the gradient of an EWC-style regularizer,

$$
e_t^{\text{mem}} = (\theta - \theta_{t-1})^\top G(\mathcal{D}_{1:t-1} \setminus \mathcal{M}_{t-1})(\theta - \theta_{t-1}), \quad (6.11)$$

which uses the GGN over the past data but, importantly, excludes the memory $M_{t-1}$ from it. We can now define a new K-prior by adding the correction term to Eq. (6.6), giving,

$$K_{t}^{\text{mem}}(\theta; \theta_{t-1}, F_{t-1}, M_{t-1}) = \sum_{n \in M_{t-1}} \ell \left( \sigma(f^n_{\theta_{t-1}}), \sigma(f^n_{\theta}) \right) + \frac{1}{2} (\theta - \theta_{t-1})^\top F_{t-1} (\theta - \theta_{t-1}),$$

(6.12)

where we define $F_{t-1} = G(D_{1:t-1} \setminus M_{t-1}) + \delta I$. By correcting for the error term, this K-prior reduces the error introduced in the K-prior of Eq. (6.8) due to a limited memory. Similarly to Online EWC, we can use a diagonal approximation to the GGN, and update it online.

The new K-prior not only reduces the gradient error, but also is more general because other regularizers are obtained as special cases by changing the memory. For an empty memory $M_{t-1} = \emptyset$, it reduces to the EWC regularizer of Eq. (6.3): the first term in Eq. (6.12) disappears and the importance $F_{t-1}$ is the GGN defined over all the past data, plus $\delta I$. On the other hand, when the memory includes all past data, it reduces to the original K-prior in Eq. (6.8). When using limited memory, the new K-prior combines the functional and weight regularizers in a way to reduce the error in both EWC and the original K-prior.

6.3.3 Reducing $e_{t}^{\text{NN}}$ Using Experience Replay

The $e_{t}^{\text{NN}}$ term depends on the mistakes made on the past data, and can be corrected by including an additional memory, denoted by $M_{2:t-1}$, of the past data where mistakes are significant. We first note that the error is equivalent to the gradient of the following,

$$c_{t}^{\text{NN}} := \sum_{n \in M_{2:t-1}} f^n_{\theta_{t-1}} + \frac{1}{2} \delta \theta^\top \theta_{t-1},$$

(6.13)

when $M_{2:t-1}$ is set to all the past data. Therefore, by adding $K_{t}^{\text{NN}} = K_{t} + c_{t}^{\text{NN}}$ with a subset of past data $M_{2:t-1}$, we reduce the error simply to $\sum_{n \in D_{1:t-1} \setminus M_{2:t-1}} \left[ \nabla f^n_{\theta_{t-1}} \right]$. 
6.3 A New Improved K-Prior

6.3.4 K-Prior With EWC-Style Regularizer and Experience Replay

Our new improved K-prior is obtained by simply correcting the K-prior from Eq. (6.8) by adding the correction terms from Eq. (6.11) and Eq. (6.13), that is,

\[ K_{t}^{\text{new}} = K_{t} + c_{t}^{\text{NN}} + c_{t}^{\text{mem}} = \sum_{n \in \mathcal{M}_{t-1}} \ell\left( \sigma(f_{\theta_{t-1}}^{n}), \sigma(f_{\theta}^{n}) \right) + \sum_{n \in \mathcal{M}_{2:t-1}} f_{\theta}^{n} r_{\theta_{t-1}}^{n} \quad (6.14) \]

\[ + \frac{1}{2} \delta \theta^{\top} \theta_{t-1} + \frac{1}{2} (\theta - \theta_{t-1})^{\top} F_{t-1} (\theta - \theta_{t-1}). \quad (6.15) \]

We make a specific choice of the memory: we choose \( \mathcal{M}_{2:t-1} \) to be a subset of \( \mathcal{M}_{t-1} \). This, as we show now, simplifies the computation and brings experience replay into our new K-prior.

Consider the first two terms from our improved K-prior in Eq. (6.14),

\[ \sum_{n \in \mathcal{M}_{t-1}} \ell\left( \sigma(f_{\theta_{t-1}}^{n}), \sigma(f_{\theta}^{n}) \right) + \sum_{n \in \mathcal{M}_{2:t-1}} f_{\theta}^{n} r_{\theta_{t-1}}^{n}. \]

The gradient of these two terms can be simplified when the second memory is a subset of the first one,

\[
\nabla \left[ \sum_{n \in \mathcal{M}_{t-1}} \ell\left( \sigma(f_{\theta_{t-1}}^{n}), \sigma(f_{\theta}^{n}) \right) + \sum_{n \in \mathcal{M}_{2:t-1}} f_{\theta}^{n} r_{\theta_{t-1}}^{n} \right] \\
= \sum_{n \in \mathcal{M}_{t-1}} \nabla f_{\theta}^{n} \left[ \sigma(f_{\theta}) - \sigma(f_{\theta_{t-1}}^{n}) \right] + \sum_{n \in \mathcal{M}_{2:t-1}} \nabla f_{\theta}^{n} r_{\theta_{t-1}}^{n} \\
= \sum_{n \in \mathcal{M}_{t-1} \setminus \mathcal{M}_{2:t-1}} \nabla f_{\theta}^{n} \left[ \sigma(f_{\theta}) - \sigma(f_{\theta_{t-1}}^{n}) \right] + \sum_{n \in \mathcal{M}_{2:t-1}} \nabla f_{\theta}^{n} \left[ \sigma(f_{\theta}) - \sigma(f_{\theta_{t-1}}^{n}) \right] + \nabla f_{\theta}^{n} r_{\theta_{t-1}}^{n} \\
= \sum_{n \in \mathcal{M}_{t-1} \setminus \mathcal{M}_{2:t-1}} \nabla f_{\theta}^{n} \left[ \sigma(f_{\theta}) - \sigma(f_{\theta_{t-1}}^{n}) \right] + \sum_{n \in \mathcal{M}_{2:t-1}} \nabla f_{\theta}^{n} \left[ \sigma(f_{\theta}) - y_{n} \right]
\]

where in the second line we used the expression in Eq. (E.1) for the gradient of the (exponential-family) loss, and where in the last line we used the definition of the residual \( r_{\theta_{t-1}}^{n} = \sigma(f_{\theta_{t-1}}^{n}) - y_{n} \) to simplify. This gradient is equal to the gradient of a sum of a functional-regularizer term and an experience replay term,

\[ \sum_{n \in \mathcal{M}_{t-1} \setminus \mathcal{M}_{2:t-1}} \ell\left( \sigma(f_{\theta_{t-1}}^{n}), \sigma(f_{\theta}^{n}) \right) + \sum_{n \in \mathcal{M}_{2:t-1}} \ell(y_{n}, \sigma(f_{\theta}^{n})), \quad (6.16) \]
again using the expression for the loss gradient in Eq. (E.1). Therefore, we can rewrite the new K-prior as the following combination,

$$K_{\text{new}}(\theta; \theta_{t-1}, F_{t-1}, M_{t-1}) = \sum_{n \in M_{1,t-1}} \ell\left(\sigma(f_{\theta}^n), \sigma(f_{\theta}^{n'})\right) + \sum_{n \in M_{2,t-1}} \ell(y_n, \sigma(f_{\theta}^n))$$

$$+ \frac{1}{2}(\theta - \theta_{t-1})^T F_{t-1} (\theta - \theta_{t-1}) + \frac{1}{2} \delta^T \theta \theta_{t-1},$$

(6.17)

where $M_{1,t-1} = M_{t-1} \setminus M_{2,t-1}$ and $M_{2,t-1}$ are two disjoint subsets of the memory, and where we define $F_{t-1} = G(D_{1,t-1} \setminus M_{t-1}) + \delta I$. The new prior combines a weight regularizer, a functional regularizer, and experience replay. The functional regularizer is based on knowledge distillation, and combined with an EWC-style quadratic regularizer which uses an importance vector obtained over all the past data but excluding the memory $M_{t-1} = M_{1,t-1} \cup M_{2,t-1}$. Using this combination, the new prior gives lower gradient reconstruction error than each method alone.

In our experiments, we select the memories as follows. We first randomly (uniformly) sample the memory set $M_{t-1}$ from all data $D_{1,t-1}$, and then randomly split it into the two memory subsets $M_{1,t-1}$ and $M_{2,t-1}$; for simplicity, we use equally sized subsets, i.e., $|M_{1,t-1}| = |M_{2,t-1}|$. We found this simple selection strategy to work surprisingly well, and leave the study of more sophisticated approaches (as well as of settings where $|M_{1,t-1}| \neq |M_{2,t-1}|$) for future work.

### 6.4 Related Work

**Continual Learning Paradigms** Existing CL methods can be classified into three orthogonal/complimentary paradigms (which can be combined): 1) weight-regularization-based methods regularize parameter updates by penalizing changes in ‘important’ parameters for previous tasks, 2) memory/rehearsal-based methods rehearse some past data or pseudo-data, and 3) architecture-based methods change the model architecture or mask model weights (Parisi et al., 2019; Swaroop, 2022). Here, we focus on the first two CL paradigms. Weight-regularization-based approaches include Elastic Weight Consoliation (EWC) (Kirkpatrick et al., 2017), Online EWC (Schwarz et al., 2018), Synaptic Intelligence (SI) (Zenke et al., 2017), Learning without Forgetting (LwF) (Li and Hoiem, 2017), and Variational Continual Learning (VCL) (Nguyen et al., 2018). Memory/rehearsal-based approaches include (Ex-
We use memory via functional-regularization / distillation, sharing conceptual similarities to Functional Regularization for CL (FRCL) (Titsias et al., 2020), Functional Regularisation of Memorable Past (FROMP) (Pan et al., 2020), Incremental Classifier and Representation Learning (iCaRL) (Rebuffi et al., 2017), and Dark Experience Replay (DER) (Buzzega et al., 2020).

**Continual Learning Settings** There are three distinct continual learning settings, which differ in whether for a given image at test time, the corresponding task identity is known, and, if not, whether the model is explicitly required to infer the task identity (Van de Ven and Tolias, 2019): 1) **task-incremental learning**, where the task identity is known, allowing us have task-specific model components such as a multiple output heads; 2) **domain-incremental learning**, where the task identity is not known and the model is not required to infer the task identity, which is typically solved by single-headed architectures, and 3) **class-incremental learning**, where the task identity is not known and is required to be inferred by the model, which includes common real-world problems such as sequentially learning new object classes (Van de Ven and Tolias, 2019).

**Knowledge-Adaptation Priors (K-Priors)** Khan and Swaroop (2021) are a principled, foundational approach for general, quick and accurate model adaptation that unifies and generalizes many previous methods. In addition to our main technical contribution of improving K-priors by correcting for two major sources of error, we substantially expand upon Khan and Swaroop (2021) in two aspects: 1) They only consider one-step adaptation settings (i.e. with 2 tasks). Alas, we will show that K-priors incur an error that accumulates as more tasks are met. This makes it especially important to assess more realistic CL scenarios with longer task sequences, which we do in this work (up to 10 tasks). 2) their experiments are limited to small-scale models and datasets, the largest only involving a small CNN (1.2M parameters) on CIFAR-10 (10 classes, 60K data points). In contrast, we demonstrate scalability up to a ResNet-18 (11M parameters) on ImageNet (1,000 classes, 1.2M data points).

### 6.5 Empirical Evaluation

We first describe the general experimental setup used throughout our evaluation. We then present empirical results corroborating the practical efficacy of our proposed Kprior+EWC+Replay method. We focus on multi-class classification in the task-incremental learning setting. In
particular, we evaluate on three continual learning benchmarks with increasing size and thus
difficulty: 1) Split-CIFAR (medium-scale), 2) Split-TinyImageNet (medium-to-large-scale),
and 3) ImageNet-1000 (large-scale). See Section E.2 for more details on the experiments
(e.g. hyperparameters used).

6.5.1 Experimental Setup

Continual learning setup. We mostly follow previous works on CL. We consider the
common multi-head task-incremental (Van de Ven and Tolias, 2019) setting with known task
identities: each method is trained sequentially on all tasks with a separate classification head
per task, and is told which task an input belongs to both at train and test time. We report test
accuracy of the final model trained on the entire task sequence (averaged over the test sets of
all observed tasks). We also compute average forgetting (aka backward-transfer) as defined in
Lopez-Paz and Ranzato (2017), which captures the (average) difference in accuracy between
when a task is first trained and after the final task. We plot mean ± standard error over three
seeds, and assess performance across a wide range of memory sizes.

Methods. In addition to our proposed $K_{\text{prior}}+EWC+\text{Replay}$ method, we evaluate four
relevant baselines for comparison. As our method combines $K_{\text{prior}}$ with both EWC and
experience replay, we also do ablations to assess the benefit of either term alone. In summary,
we consider the following methods:

1. $K_{\text{prior}}+EWC+\text{Replay}$. Our proposed regularizer which combines the K-prior function
   regularizer over $\mathcal{M}_1$ with the EWC-style weight regularizer and the experience replay
   term over $\mathcal{M}_2$.

2. $K_{\text{prior}}$. The original K-prior regularizer over $\mathcal{M}_1$ as proposed by Khan and Swaroop
   (2021), without the EWC-style weight regularization term and without the experience
   replay term over $\mathcal{M}_2$.

3. $K_{\text{prior}}+EWC$ (ablation). A regularizer combining the K-prior function regularizer over
   $\mathcal{M}_1$ with only the EWC-style weight regularization, i.e. without the experience replay
   term over $\mathcal{M}_2$.

4. $K_{\text{prior}}+\text{Replay}$ (ablation). A regularizer combining the K-prior function regularizer
   over $\mathcal{M}_1$ with only the experience replay term over $\mathcal{M}_2$, i.e. without the EWC-style
   weight regularization.

5. Batch Joint. Joint batch training of a single multi-head model across the data of
   all tasks, i.e. the optimal CL solution which serves as an upper bound we wish to
   approach.
6.5 Empirical Evaluation

Fig. 6.2 Results on Split-CIFAR. Kprior+EWC+Replay is superior across memory sizes, closely approaching Batch Joint for memory size 33.3% (left; x-axis log-scaled). It also forgets less (relative to Batch Joint) with a growing number of tasks, for memory sizes 0.7% (middle) and 33.3% (right).


7. Online EWC (Schwarz et al., 2018), which has the same weight regularizer as Kprior+EWC+Replay, but without the function regularizer over $M_1$ or the experience replay term over $M_2$.

6.5.2 Results on Split-CIFAR

Setup. Split-CIFAR (Zenke et al., 2017) has 6 tasks with 10 classes each. The first task is CIFAR-10 (Krizhevsky and Hinton, 2009) with 50,000 training and 10,000 test data points across 10 classes. The subsequent 5 tasks are taken sequentially from CIFAR-100 (Krizhevsky and Hinton, 2009), each with 5,000 training and 1,000 test data points across 10 classes. In total, we thus have 90,000 data points. We use the same CifarNet model as Pan et al. (2020); Zenke et al. (2017): a multi-head CNN with 4 convolutional layers, followed by 2 dense layers with dropout, with $\sim 1.2$M model parameters in total. On each task, we train for 80 epochs using Adam with learning rate $10^{-3}$ and batch size 256.

Results. Fig. 6.2 shows our results on Split-CIFAR. We consider memory sizes between 100 and 5,000 per task; at 5,000, we thus store 10% of the data for task 1, and all data for tasks 2-5 (which in total thus corresponds to a third of all data). We see that Kprior performs poorly at small memory sizes and is much worse than Online EWC. While performance improves noticeably with growing memory size, Kprior remains far below Batch Joint even at memory size 33.3%. This confirms that when using a NN instead of a GLM, the theory behind Kprior (see Section 6.2) indeed ceases to hold. However, when we add the experience replay term to correct for that error (Kprior+Replay), performance is substantially boosted at large memory sizes, actually enabling us to reach Batch Joint performance for memory
Continual Deep Learning via Gradient Reconstruction of the Past

Fig. 6.3 Results on Split-TinyImageNet. $K_{prior}+EWC$ performs well across memory sizes (left; x-axis log-scaled) and compared to further strong baselines from Delange et al. (2021) (right).

size 33.3%. However, the experience replay correction term does not help at small memory sizes. In contrast, adding the EWC-style weight regularizer ($K_{prior}+EWC$) substantially improves performance at small memory sizes, empirically confirming the property that $K_{prior}+EWC$ converges to Online EWC for small memories (see Section 6.3.2). However, we also confirm that for large memories, $K_{prior}+EWC$ converges to $K_{prior}$, resulting in a performance drop.

Finally, we see that $K_{prior}+EWC+Replay$ obtains $\sim 97\%$ of the batch performance with just $\sim 10\%$ of the data (and full batch performance with $\sim 33.3\%$ of the data), but does not degrade as memory size decreases (unlike $K_{prior}+Replay$). $K_{prior}+EWC+Replay$ thus combines the complementary benefits of both error correction terms, i.e., of both EWC-style weight-regularization and experience replay, to significantly improve upon vanilla $K_{prior}$ in both the small and large memory regime. Fig. 6.2 (mid & right) further shows that our method can leverage prior knowledge more effectively than other methods, thereby suffering less from forgetting with a growing number of tasks. This confirms that the two error correction terms in $K_{prior}+EWC+Replay$ are particularly important for mitigating error accumulation across longer task sequences.

6.5.3 Results on Split-TinyImageNet

Setup. Following Delange et al. (2021), we construct Split-TinyImageNet by dividing TinyImageNet (Le and Yang, 2015) into a sequence of 10 tasks with 20 classes each (using the same random division as in Delange et al. (2021)). Each class has 500 data points split into training (80%) and validation (20%), and 50 test points (totalling to 110,000 points). We use the VGG-like BASE model from Delange et al. (2021) with 6 convolutional layers, 4 max pooling layers, and 2 dense layers, with a total of $\sim 3.5M$ parameters. On each task,
we train for 70 epochs (with early stopping and exponential learning rate decay, without
regularization) using SGD with momentum 0.9 and batch size 200. This replicates the setup
of Delange et al. (2021) to make our results directly comparable.¹

Results. Fig. 6.3 shows our results on Split-TinyImageNet. We found that the experience
replay error correction term does not help on this benchmark, so we representatively plot
just $K_{\text{prior}}$ and $K_{\text{prior}}+E\text{WC}.²$ We again see that $K_{\text{prior}}+E\text{WC}$ can substantially improve
over $K_{\text{prior}}$ (especially at small memory sizes) and Online E\text{WC}, achieving $\sim90\%$ of
the batch performance with $\sim10\%$ of the data (Fig. 6.3 left). It also compares favourably
against a diverse range of other strong CL methods across all three CL paradigms: 1) memory/rehearsal – iCaRL (Rebuffi et al., 2017), GEM (Lopez-Paz and Ranzato, 2017),
R-FM & R-PM (Delange et al., 2021), 2) weight-regularization – LwF (Li and Hoiem, 2017),
EBLL (Rannen et al., 2017), EWC (Kirkpatrick et al., 2017), SI (Zenke et al., 2017), MAS
(Aljundi et al., 2018), mode-IMM (Lee et al., 2017), and 3) architectural – PackNet (Mallya
and Lazebnik, 2018), HAT (Serra et al., 2018) (Fig. 6.3 right).³

6.5.4 Results on ImageNet-1000

Setup. We consider the ImageNet-1000 benchmark proposed by Rebuffi et al. (2017), which
randomly splits the full ImageNet dataset (Deng et al., 2009) of $\sim1.2$M data points into a
sequence of 10 tasks with 100 classes and $\sim120$K data points each. Following Rebuffi et al.
(2017), we use a ResNet-18 with $\sim11$M model parameters. For training on each task, we use
the ImageNet reference training pipeline (with 40 epoch configuration) of the FFCV library
(Leclerc et al., 2022).⁴

Results. Fig. 6.4 shows our results on ImageNet-1000. We consider memory sizes
between 200 and 10K per task, where the latter amounts to 7.5% of the entire data. The
observed trends qualitatively match those from previous experiments. In particular, $K_{\text{prior}}$
derperforms for small memory sizes, and while it improves with increasing memory, it
peaks at a 3.8% memory and then even starts declining. We hypothesize that this is again due
to accumulation of the NN error, which might become more severe with a larger memory as
more data points can contribute to the error. This is evidenced by the fact that correcting for
the NN error ($K_{\text{prior}}+\text{Replay}$) substantially boosts performance at a 7.5% memory (but it

¹The only difference lies in the hyperparameter tuning procedure: while Delange et al. (2021) use their
proposed online tuning algorithm, we resort to a standard grid search for simplicity; see Section E.2 for details.
²This is likely because almost perfect train accuracy is attained on all tasks (see, e.g., Table 14 in Delange
et al. (2021)). Thus, $e^\text{NN}_t$ in Eq. (6.9) is close to zero, such that NN error correction cannot boost performance.
³Results are from Delange et al. (2021); their total memory sizes [4500,9000] correspond to [5.6%,11.2%]
of the data.
⁴For all details of the training procedure, see https://github.com/libffcv/ffcv-imagenet/.
Fig. 6.4 Results on ImageNet-1000. Kprior+EWC+Replay performs favorably across a range of memory sizes (left; x-axis log-scaled), and suffers less from forgetting (relative to Batch Joint) with an increasing number of tasks, here exemplary shown at the largest memory size of 7.5% (right).

remains poor at small memories). In contrast, Kprior+EWC again improves accuracy only for small memories. Finally, Kprior+EWC+Replay combines the benefit of both error correction terms to perform well across all memory sizes, achieving >80% of the batch performance with a memory of <10% of the past data. It also again forgets less along the task sequence, demonstrating that it better mitigates error accumulation.

6.6 Discussion

In this chapter, we developed a new probabilistic deep learning method to address the continual deep learning problem. In particular, we proposed an approach that aims to explicitly approximate the solution obtained from batch training on all tasks jointly. To achieve this, our method combines principles from function-regularization, weight-regularization, and experience replay to faithfully reconstruct the gradients of the past training objective. Empirically, we demonstrated the efficacy of our method across different memory sizes and benchmark scales. In the next and final chapter, we will conclude this thesis by first providing a summary of our contributions and then outlining directions for future research, for all the application domains considered.
Chapter 7

Conclusions and Future Directions

This chapter concludes this thesis by first summarizing our main contributions across the different application areas of probabilistic deep learning considered (Section 7.1). We finally describe future research directions that we are interested in pursuing for each application (Section 7.2).

7.1 Summary of Contributions

In this section, we summarize the main contributions we made in this thesis to advance four different application domains using probabilistic deep learning: out-of-distribution detection (Section 7.1.1), data-efficient optimization (Section 7.1.2), neural network calibration (Section 7.1.3), and continual deep learning (Section 7.1.4).

7.1.1 Contributions to Out-of-Distribution Detection

In Chapter 3, we developed a new probabilistic deep learning approach for out-of-distribution detection. Our method is based on a new Bayesian VAE model, which not just performs probabilistic inference over the local latent variables, but additionally also over the global model parameters. In particular, we used stochastic gradient MCMC to infer a full posterior distribution over the model parameters, rather than just inferring a point estimate, as done in regular VAEs. By capturing epistemic uncertainty in the model parameters in this way, we can detect OoD inputs — both in input space and the model’s latent space — by leveraging the effective sample size as an information-theoretic measure of data point novelty, similar to how it is done in probabilistic active learning. We empirically verified the efficacy of our proposed approach. We believe that our proposed BVAE approach will contribute to the ongoing efforts of making deep learning systems more robust and safe by endowing them
with the ability to detect when a data point was drawn from a different distribution than the training data distribution.

### 7.1.2 Contributions to Data-Efficient Optimization

In Chapter 4, we developed a new probabilistic deep learning approach for data-efficient optimization. To this end, we proposed a method that performs the optimization in the low-dimensional, continuous latent space of a VAE, instead of in the high-dimensional, structured input space. In order to maintain a latent manifold that is as useful as possible for the optimization procedure, we developed a weighted retraining procedure that continually updates the VAE and its latent space to take into account newly sampled data points. In particular, this is achieved by periodically retraining the model on the data points queried along the optimization trajectory, as well as weighting those points according to their objective function value. Despite its conceptual simplicity and ease of implementation, our method was shown to compare favourably in practice, both on synthetic and real-world optimization tasks. We hope that our proposed weighted retraining approach will help push the frontier on sample-efficient black-box optimization to allow practitioners to tackle many important science and engineering problems such as drug design.

### 7.1.3 Contributions to Neural Network Calibration

In Chapter 5, we developed a new probabilistic deep learning approach for neural network calibration. Our work has three main findings: 1) modelling weight correlations in NNs is crucial to obtaining reliable predictive posteriors, 2) given these correlations, unimodal approximations of the posterior can be competitive with approximations that assign mass to multiple modes (e.g., deep ensembles), 3) inference does not need to be performed over all the weights in order to obtain reliable predictive posteriors. We use these insights to develop a framework for scaling Bayesian inference to NNs with a large number of weights. We approximate the posterior over a subset of the weights while keeping all others deterministic. Computational cost is decoupled from the total number of weights, allowing us to conveniently trade it off with the quality of approximation. This allows us to use more expressive posterior approximations, such as full-covariance Gaussian distributions. Linearized Laplace subnetwork inference can be applied post-hoc to any pre-trained model, making it particularly attractive for practical use. Our empirical analysis suggests that this method 1) is more expressive and retains more uncertainty than crude approximations over the full network, 2) allows us to employ larger NNs, which fit a broader range of functions, without sacrificing the quality of our uncertainty estimates, and 3) is competitive with state-
of-the-art uncertainty quantification methods, like deep ensembles. Our hope is that our proposed subnetwork inference framework will help make neural network models more reliable and safe by endowing them with calibrated uncertainty estimates, enabling them to 'know when they do not know' the answer.

7.1.4 Contributions to Continual Deep Learning

In Chapter 6, we developed a new probabilistic deep learning approach for continual deep learning. In particular, we proposed to address the continual learning problem in a theoretically-grounded way by explicitly approximating the optimal model obtained via batch-training on all tasks jointly. To this end, we developed $K_{\text{prior}} + \text{EWC} + \text{Replay}$, a new continual learning method which efficiently re-uses prior knowledge to reconstruct the gradients of the past training objective as faithfully as possible. To achieve this, our method combines principles from function-regularization, weight-regularization, and experience replay to reduce the gradient-reconstruction error. Empirically, we demonstrated the effectiveness and scalability of $K_{\text{prior}} + \text{EWC} + \text{Replay}$ across different memory sizes on common task-incremental continual learning benchmarks. In particular, we showed that our proposed approach can be less susceptible to catastrophic forgetting and thus achieve better performance compared to various baselines. Notably, our method can achieve $>80\%$ of the batch performance by utilizing a memory of $<10\%$ of the past data on the benchmarks considered. We believe that our proposed $K_{\text{prior}} + \text{EWC} + \text{Replay}$ can help fix the notorious catastrophic forgetting problem of neural networks by enabling them to continually adapt to new problems, eventually reducing the cost and environmental impact of deep learning.

7.2 Future Research Directions

In this section, we outline potential future research directions to further advance the four different application domains considered in this thesis using probabilistic deep learning: out-of-distribution detection (Section 7.2.1), data-efficient optimization (Section 7.2.2), neural network calibration (Section 7.2.3), and continual deep learning (Section 7.2.4).

7.2.1 Future Research on Out-of-Distribution Detection

There are several immediate avenues to extend our proposed BVVAE approach for OoD detection. We mostly considered a vanilla VAE model as originally proposed; in contrast, one could also consider more elaborate VAE architectures, e.g., using more sophisticated posteriors (Rezende and Mohamed, 2015), priors (Bauer and Mnih, 2019; Papamakarios
Conclusions and Future Directions

et al., 2017; Tomczak and Welling, 2018), decoders, etc., which are all complementary. Or, we could use an entirely different class of likelihood-based deep generative models, e.g., flow-based and auto-regressive deep generative models. We could also use alternative lower bounds to the marginal log-likelihood, such as the importance weighted lower bound used in an IWAE (Burda et al., 2015), or combinations of the VAE and IWAE lower bounds, as proposed in Rainforth et al. (2018). Another idea would be to use alternative, non-sampling-based approximate inference techniques for estimating the posterior distribution over model parameters, e.g., variational inference or the Laplace approximation. An interesting direction might also be to try to exploit label information (e.g., by using a hybrid model combining our BVAE with a predictive model, to enable semi-supervised OoD detection), and comparing how this empirically compares against task-specific (i.e., supervised) OoD detection approaches; it would also be interesting to know if it is beneficial to have uncertainty both in the predictive model as well as in the generative model. Given the conceptual similarity of our OoD detection approach to information-theoretic active learning, it seems natural to also consider using our proposed BVAE model and the ESS score for active learning\footnote{Related work includes (Gong et al., 2019), which however focus on the feature-wise active learning problem.} for efficient model training; note that this is not limited to generative models: we could also use the ESS in discriminative models trained using SG-MCMC, and it would be interesting to see how this performs compared to state-of-the-art active learning methods. Additionally, it would be beneficial to get a better understanding of the theoretical underpinnings of our proposed approach; in particular, one of the variants we developed performed Bayesian inference over the variational (i.e., encoder) parameters of the VAE, and it is not clear what the theoretical implications of this are. Finally, one could extend the empirical evaluation of our method to consider datasets other than image data (e.g., more structured data such as molecules), and to explore a more elaborate benchmark for OoD detection in latent space (given that our proposed benchmark was somewhat synthetic).

7.2.2 Future Research on Data-Efficient Optimization

There are several drawbacks to our proposed weighted retraining method for sample-efficient optimization that motivate promising directions for future work. Firstly, we often found it difficult to train a latent objective model that performed well at optimization in the latent space, which is critical for good performance. We believe that further research is necessary into other techniques to make the latent space of DGMs more amenable to optimization. Secondly, our method requires a large dataset of labelled data to train a DGM, making it unsuitable for problems with very little data, motivating adaptations that would allow...
unlabelled data to be utilized. Finally, due to being relatively computationally intensive, we were unable to characterize the long-term optimization behaviour of our algorithm. In this regime, we suspect that it may be beneficial to use a weighting schedule instead of a fixed weight, which may allow balancing exploration vs. exploitation similar to simulated annealing (Van Laarhoven and Aarts, 1987).

7.2.3 Future Research on Neural Network Calibration

Despite the encouraging results on neural network calibration achieved by our proposed subnetwork Laplace inference method, there remain several limitations of our method which would be interesting to explore in future work. Firstly, Jacobian computation in multi-output models remains challenging. With reverse mode automatic differentiation used in most deep learning frameworks, it requires as many backward passes as there are model outputs. This prevents using linearized Laplace in settings like semantic segmentation (Liu et al., 2019) or classification with large numbers of classes (Deng et al., 2009). Note that this issue applies to the linearized Laplace method and that other inference methods, without this limitation, could be used in our framework. Secondly, the choice of prior precision $\lambda$ determines the performance of the Laplace approximation to a large degree. Our proposed scheme to update $\lambda$ for subnetworks relies on having a sensible parameter setting for the full network. Since inference in the full network is often intractable, currently the best approach for choosing $\lambda$ is cross validation using the subnetwork approximation directly. Thirdly, the space requirements for the Hessian limit the maximum number of subnetwork weights. For example, storing a Hessian for 40K weights requires around 6.4GB of memory. For very large models, like modern transformers, tractable subnetworks would represent a vanishingly small proportion of the weights. While we demonstrated that strong performance does not necessarily require large subnetworks, finding better subnetwork selection strategies remains a key direction for future research. Finally, we are keen to apply recent advances to the Laplace approximation orthogonal to subnetwork inference, e.g. using the marginal likelihood to select the prior precision (Immer et al., 2021a), adding uncertainty units to the model to improve its calibration (Kristiadi et al., 2021), or considering a mixture of Laplace approximations to better capture the multi-modality of the posterior (Eschenhagen et al., 2021).

7.2.4 Future Research on Continual Deep Learning

There are several aspects in which we could extend our proposed $K_{prior}$+EWC+Replay method for continual deep learning. Firstly, our method combines two out of the three main
Conclusions and Future Directions

continual learning paradigms, namely weight-regularization based CL and memory/rehearsal-based CL. It would be interesting to explore if one could further improve the method (ideally also in a principled way) by additionally including the third major CL paradigm, namely architecture-based CL, which changes the model architecture and/or masks model weights to avoid forgetting. Secondly, the way we select the two memory sets is rather simplistic, namely uniformly at random. We would be keen investigate how more sophisticated strategies for selecting the two memory sets (e.g., based on some measure of importance or relevance of the data points) can further boost performance of our method. Thirdly, our empirical evaluation focused on CL benchmarks that consider the task-incremental setting in the context of image classification problems. In addition to exploring other data modalities (e.g., text data), it would be interesting to consider the other two main CL settings, namely domain-incremental CL and class-incremental CL, in which task identities are not known, making the problem more difficult.
References


Deng, J., Dong, W., Socher, R., Li, L., Kai Li, and Li Fei-Fei (2009). ImageNet: A large-scale hierarchical image database. In CVPR.


Ober, S. W. and Rasmussen, C. E. (2019). Benchmarking the neural linear model for regression. In *Symposium on Advances in Approximate Bayesian Inference (AABI)*.


References


References


Appendix A

The Laplace Approximation

A.1 Derivation of the Laplace Approximation

Let $p(\theta | D)$ be an intractable posterior, written as

$$p(\theta | D) := \frac{1}{\int_\theta p(D | \theta) p(\theta) d\theta} p(D | \theta) p(\theta) =: \frac{1}{Z} h(\theta) \quad (A.1)$$

Our goal is to approximate this distribution with a Gaussian arising from the Laplace approximation. The key observation is that we can rewrite the normalizing constant $Z$ as the integral $\int \exp(\log h(\theta)) d\theta$. Let $\theta_{\text{MAP}} := \arg \max_\theta \log p(\theta | D) = \arg \max_\theta \log h(\theta)$ be a (local) maximum of the posterior—the so-called maximum a posteriori (MAP) estimate. Taylor-expanding $\log h$ around $\theta_{\text{MAP}}$ up to the second order yields

$$\log h(\theta) \approx h(\theta_{\text{MAP}}) - \frac{1}{2} (\theta - \theta_{\text{MAP}})^\top \Lambda (\theta - \theta_{\text{MAP}}), \quad (A.2)$$

where $\Lambda := -\nabla^2 \log h(\theta)|_{\theta_{\text{MAP}}}$ is the negative Hessian matrix of the log-joint in Eq. (A.1), evaluated at $\theta_{\text{MAP}}$. Similar to its original formulation, here we again obtain a (multivariate) Gaussian integral, the analytic solution of which is readily available:

$$Z \approx \exp(\log h(\theta_{\text{MAP}})) \int_\theta \exp \left( -\frac{1}{2} (\theta - \theta_{\text{MAP}})^\top \Lambda (\theta - \theta_{\text{MAP}}) \right) d\theta \quad (A.3)$$

$$= h(\theta_{\text{MAP}}) \frac{(2\pi)^{\frac{d}{2}}}{(\det \Lambda)^{\frac{1}{2}}}.$$
Plugging the approximations Eq. (A.2) and Eq. (A.3) back into the expression of $p(\theta \mid D)$, we obtain

$$
p(\theta \mid D) = \frac{1}{Z} h(\theta) \approx \frac{(\det \Lambda)^{\frac{1}{2}}}{(2\pi)^{\frac{n}{2}}} \exp \left( -\frac{1}{2} (\theta - \theta_{\text{MAP}})^\top \Lambda (\theta - \theta_{\text{MAP}}) \right), \quad (A.4)
$$

which we can immediately identify as the Gaussian density $\mathcal{N}(\theta; \theta_{\text{MAP}}, \Sigma)$ with mean $\theta_{\text{MAP}}$ and covariance matrix $\Sigma := \Lambda^{-1}$. 
Appendix B

Out-of-Distribution Detection

B.1 Precision-Recall and ROC Curves for FashionMNIST (Held-Out Classes)

Fig. B.1 (Left) Precision-recall curves and (right) ROC curves of all methods on the FashionMNIST (held-out classes) benchmark with classes 0 and 1 held-out.
Fig. B.2 (Left) Precision-recall curves and (right) ROC curves of all methods on the Fashion-MNIST (held-out classes) benchmark with classes 2 and 3 held-out.

Fig. B.3 (Left) Precision-recall curves and (right) ROC curves of all methods on the Fashion-MNIST (held-out classes) benchmark with classes 4 and 5 held-out.

Fig. B.4 (Left) Precision-recall curves and (right) ROC curves of all methods on the Fashion-MNIST (held-out classes) benchmark with classes 6 and 7 held-out.
Fig. B.5 (Left) Precision-recall curves and (right) ROC curves of all methods on the Fashion-MNIST (held-out classes) benchmark with classes 8 and 9 held-out.
Appendix C

Data-Efficient Optimization

C.1 Details on the Weighting Function

C.1.1 More Information on Rank-Based Weighting

Independence from dataset size  We show that the key properties of rank-based weighting depend only on $k$, and not on the dataset size $N$, meaning that applying rank weighting with a fixed $k$ to differently sized datasets will yield similar results. In particular, we show that under mild assumptions, the fraction of weights devoted to a particular quantile of the data depends on $k$ but not $N$.

Suppose that the quantile of interest is the range $q_1–q_2$ (for example, the first quartile corresponds to the range 0–0.25). This corresponds approximately to the points with ranks $q_1N–q_2N$. We make the following assumptions:

1. $kN \gg 1$
2. $kN$ is approximately integer valued, which is realistic if $N \gg 1/k$
3. $q_1$ and $q_2$ are chosen so that $q_1N$ and $q_2N$ are integers.

Because the ranks form the sequence $0, 1, \ldots, N – 1$, under the above assumptions all weights are reciprocal integers, so the sum of the rank weights is strongly connected to the harmonic series. Recall that the partial sum of the harmonic series can be approximated by the natural logarithm:

$$\sum_{j=1}^{N} \frac{1}{j} \approx \ln N + \gamma$$  \hspace{1cm} (C.1)

Here, $\gamma$ is the Euler–Mascheroni constant. The fraction of the total weight devoted to the quantile $q_1–q_2$ can be found by summing the weights of points with rank $q_1N–q_2N$, and
dividing by the normalization constant (the sum of all weights). First, because $kN \gg 1$ implies that $(kN - 1) \approx kN$, the sum of all the weights can be expressed as:

$$
\sum_{r=0}^{N-1} w(x_r; D, k) = \frac{1}{kN + r} = \sum_{r=1}^{kN(N-1)} \frac{1}{r} - \sum_{r=1}^{kN-1} \frac{1}{r} \\
\approx (\ln((k+1)N - 1) + \gamma) - (\ln(kN - 1) + \gamma) \\
= \ln\frac{(k+1)N - 1}{kN - 1} \approx \ln\frac{(k+1)N}{kN} = \ln\left(1 + \frac{1}{k}\right)
$$

Note that this does not depend on the dataset size $N$. Second, using the same assumption, the sum of the weights in the quantile is:

$$
\sum_{r=q_1N}^{q_2N} w(x_r; D, k) = \frac{1}{kN + r} = \sum_{r=1}^{(k+q_2)N} \frac{1}{r} - \sum_{r=1}^{(k+q_1)N-1} \frac{1}{r} \\
\approx (\ln((k+q_2)N - 1) + \gamma) - (\ln((k+q_1)N - 1) + \gamma) \\
= \ln\frac{(k+q_2)N}{(k+q_1)N - 1} \approx \ln\frac{(k+q_2)N}{(k+q_1)N} = \ln\frac{(k+q_2)}{(k+q_1)}
$$

which is also independent of $N$ (note that setting $q_1 = 0$, $q_2 = 1$ into the formula yields the same expression for the sum of the weights as derived above). Therefore, the fraction of the total weight allocated to a given quantile of data is independent of $N$, being only dependent on $k$. Although the analysis that led to this result made some assumptions about certain values being integers, in practice the actual distributions of weights are extremely close to what this analysis predicts. Fig. C.1 shows the allocation of the weights to different quantiles of the datasets. For $kN > 1$, the distribution is essentially completely independent of $N$. Only when $kN < 1$ this fails to hold.

Finally, we discuss some potential questions about the rank-based weighting.

**Why do the weights need to be normalized?** If the objective is to minimize $\sum_{x_n \in D} w_n \mathcal{L}(x_n)$, for any $a > 0$, minimizing $a \sum_{x_n \in D} w_n \mathcal{L}(x_n)$ is an equivalent problem. Therefore, in principle,
C.1 Details on the Weighting Function

Fig. C.1 Cumulative distribution of rank weights (sorted highest to lowest), showing a distribution that is independent of $N$ if $kN > 1$.

the absolute scale of the weights does not matter, and so the weights do not need to be normalized, even if this precludes their interpretation as a probability distribution. However, in practice, if minimization is performed using gradient-based algorithms, then the scaling factor for the weights is also applied to the gradients, possibly requiring different hyperparameter settings (such as a different learning rate). By normalizing the weights, it is easier to identify hyperparameter settings that work robustly across different problems, thereby allowing weighted retraining to be applied with minimal tuning.

**Why not use a weight function directly based on the objective function value?** Although there is nothing inherently flawed about using such a weight function, there are some practical difficulties.

- Such a weight function would either be bounded (in which case values beyond a certain threshold would all be weighted equally), or it would be very sensitive to outliers (i.e., extremely high or low values which would directly cause the weight function to take on an extremely high or low value). This is extremely important because the weights are normalized, so one outlier would also affect the values of all other points.

- Such a weight function would not be invariant to simple transformations of the objective function. For example, if the objective function is $f$, then maximizing $f(x)$ or $f_{ab}(x) = af(x) + b$ is an equivalent problem (for $a > 0$), but would yield different weights. This would effectively introduce scale hyperparameters into the weight function, which is undesirable.
C.1.2 Mini-Batching for Weighted Training

As mentioned in the main text, one method of implementing the weighting with mini-batch stochastic gradient descent is to sample each point $x_n$ with probability proportional to its weight $w_n$ (with replacement). A second method is to sample points with uniform probability and re-weight each point’s contribution to the total loss by its weight,

$$\sum_{x_n \in D} w_n L(x_n) \approx \frac{N}{M} \sum_{m=1}^{M} w_m L(x_m).$$  \hspace{1cm} (C.2)

If done naively, these mini-batches may have extremely high variance, especially if the variance of the weights is large. In practice, we found it was sufficient to reduce the variance of the weights by simply adding multiple copies of any $x_n$ with $w_n > w_{\text{max}}$, then reducing the weight of each copy such that the sum is still $w_n$. The following is a Python code snippet implementing this variance reduction:

```python
def reduce_variance(data, weights, w_max):
    new_data = []
    new_weights = []
    for x, w in zip(data, weights):
        if w <= w_max:  # If it is less than the max weight, just add it
            new_data.append(x)
            new_weights.append(w)
        else:  # Otherwise, add multiple copies
            n_copies = int(math.ceil(w / w_max))
            new_data += [x] * n_copies
            new_weights += [w / n_copies] * n_copies
    return new_data, new_weights
```

The parameter $w_{\text{max}}$ was typically set to 5.0, which was chosen to both reduce the variance, while simultaneously not increasing the dataset size too much. Note that this was applied after the weights were normalized. This also makes it feasible to train for only a fraction of an epoch, since without variance reduction techniques there is a strong possibility that high-weight data points would be missed if the entire training epoch was not completed.
C.1 Details on the Weighting Function

C.1.3 Implementation of Weighted Training

One of the benefits of weighted retraining which we would like to highlight is its ease of implementation. Below, we give example implementations using common machine learning libraries.

PyTorch (weighted sampling)

**Standard Training**

```python
from torch.utils.data import *
dataloader = DataLoader(data)
for batch in dataloader:
    # ...
```

**Weighted Training**

```python
from torch.utils.data import *
sampler = WeightedRandomSampler(weights, len(data))
dataloader = DataLoader(data, sampler=sampler)
for batch in dataloader:
    # ...
```

PyTorch (direct application of weights)

**Standard Training**

```python
criterion = nn.MSELoss()
outputs = model(inputs)
loss = criterion(outputs, targets)
loss.backward()
```

**Weighted Training**

```python
criterion = nn.MSELoss(reduction=None)
outputs = model(inputs)
loss = criterion(outputs, targets)
loss = torch.mean(loss * weights)
loss.backward()
```

Keras

**Standard Training**

```python
model.fit(x, y)
```

**Weighted Training**

```python
model.fit(x, y, sample_weight=weights)
```

C.1.4 Implementation of Rank Weighting

We provide a simple implementation of rank-weighting:

```python
import numpy as np
def get_rank_weights(outputs, k):

    # argsort argsort to get ranks (a cool trick!)
    # assume here higher outputs are better
```
```
outputs_argsort = np.argsort(-np.asarray(outputs))
ranks = np.argsort(outputs_argsort)
return 1 / (k * len(outputs) + ranks)
```

**C.1.5  Rank-Weighted Distributions of Objective Function Values of 2D Shape and Arithmetic Expression Datasets**

Finally, to complement the rank-weighted distributions of objective function values of the ZINC dataset in Fig. 4.2, we here also show the corresponding distributions for the 2D shape and arithmetic expression datasets used in Section 3.5 (Fig. C.2 and Fig. C.3).

![Fig. C.2](similar to Fig. 4.2)

**Fig. C.2** Illustration of rank weighting (Eq. (4.1)) on the shapes dataset (see Section 3.5) (similar to Fig. 4.2).

![Fig. C.3](similar to Fig. 4.2)

**Fig. C.3** Illustration of rank weighting (Eq. (4.1)) on the arithmetic expression dataset (see Section 3.5) (similar to Fig. 4.2).

**C.2  Further Experimental Results**

**C.2.1  Optimization Performance With More Weighted Retraining Parameters**

Holding \( r \) fixed at \( r_{\text{low}} \), we vary \( k \) from \( k_{\text{low}} \) to \( \infty \) (Fig. C.4) and vice versa (Fig. C.5). In general, performance increases monotonically as \( k, r \) decrease, suggesting a continuous improvement from increasing weighting or retraining. The arithmetic expression task did not
C.2 Further Experimental Results

Fig. C.4 Top1 optimization performance of weighted retraining for different $k$ values with $r = r_{\text{low}}$.

Fig. C.5 Top1 optimization performance of weighted retraining for different $r$ values with $k = k_{\text{low}}$, $r_{\text{low}} = 5$, $r_{\text{high}} = 50$ for the 2D shape area task; $r_{\text{low}} = 50$, $r_{\text{high}} = 100$ for the others.

show this behaviour for retraining, which we attribute to the high degree of randomness in the optimization.

C.2.2 Top10 and Top50 Optimization Results

Figure C.6 and Fig. C.7 give the Top10 and Top50 scores for the experiment described in Section 4.4.2. These results are qualitatively similar to those in Fig. 4.4, suggesting that our method finds many unique high-scoring points.
Fig. C.6 Top10 optimization performance of weighted retraining for all tasks (setup identical to Fig. 4.4).

Fig. C.7 Top50 optimization performance of weighted retraining for all tasks (setup identical to Fig. 4.4).
C.2 Further Experimental Results

C.2.3 Comparison of Chemical Design Results With Previous Papers

Table C.1 compares the results attained in this paper with the results from previous papers that attempted the same task. Weighted retraining clearly beats the previous best methods, which were based on reinforcement learning, while simultaneously being more sample-efficient. Note that despite using the same pre-trained model as Jin et al. (2018a), we achieved better results by training our sparse Gaussian process on only a subset of data and clipping excessively low values in the training set, which allowed us to get significantly better results than they reported.

<table>
<thead>
<tr>
<th>Model</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>no. queries (source)</th>
</tr>
</thead>
<tbody>
<tr>
<td>JT-VAE (Jin et al., 2018a)</td>
<td>5.30</td>
<td>4.93</td>
<td>4.49</td>
<td>2500 (paper(^1))</td>
</tr>
<tr>
<td>GCPN (You et al., 2018)</td>
<td>7.98</td>
<td>7.85</td>
<td>7.80</td>
<td>(\approx 10^6) (email(^2))</td>
</tr>
<tr>
<td>MolDQN (Zhou et al., 2019)</td>
<td>11.84</td>
<td>11.84</td>
<td>11.82</td>
<td>(\geq 5000) (paper(^3))</td>
</tr>
<tr>
<td>ChemBO (Korovina et al., 2020)</td>
<td>18.39</td>
<td>-</td>
<td>-</td>
<td>100 (paper(^4))</td>
</tr>
</tbody>
</table>

| JT-VAE (our Bayesian optimization) | 5.65  | 5.63  | 5.43  | 500       |
| JT-VAE \((k = 10^{-3}, \text{no retraining})\) | 5.95  | 5.75  | 5.72  | 500       |
| JT-VAE \((k = 10^{-3}, \text{retraining})\) | 21.20 | 15.34 | 15.34 | 500       |
| JT-VAE \((k = 10^{-3}, \text{retraining, best result})\) | **27.84** | **27.59** | **27.21** | **500** |

Table C.1 Comparison of top 3 scores on chemical design task. Baseline results are copied from Zhou et al. (2019). All our results state the median of 5 runs unless otherwise stated (judged by best result), each run being 500 epochs.

C.2.4 Pictures of the Best Molecules Found by Weighted Retraining

Figure C.8 illustrates some of the best molecules found with weighted retraining. Note that all the high-scoring molecules are extremely large. It has been reported previously that larger molecules achieve higher scores, thereby diminishing the value of this particular design task for reinforcement learning algorithms (Zhou et al., 2019). However, the fact that these molecules were found with a generative model strongly highlights the ability of weighted retraining to find solutions outside of the original training distribution.

\(^1\)These were the top results across 10 seeds, with 250 queries performed per seed.

\(^2\)Obtained through email correspondence with the authors.

\(^3\)The experimental section states that the model was trained for 5000 episodes, so at least 5000 samples were needed. It is unclear if any batching was used, which would make the number of samples greater.

\(^4\)See Table 3 of Korovina et al. (2020).
C.3 Details on Experimental Setup

C.3.1 Retraining Parameters

When retraining a model with frequency $r$, the model is optionally fine-tuned initially, then repeatedly fine-tuned on queries $r$, $2r$, $3r$, $\ldots$ until the query budget is reached. All results use the rank-based weighting function defined in Eq. (4.1) unless otherwise specified. We consider a budget of $B = 500$ function evaluations, which is double the budget used in Jin et al. (2018a); Kusner et al. (2017).

C.3.2 Bayesian Optimization

For optimizing over the latent manifold, we follow previous work (Jin et al., 2018a; Kusner et al., 2017) and use Bayesian optimization with a variational sparse Gaussian process (SGP) surrogate model (Titsias, 2009) (with 500 inducing points) and the expected improvement acquisition function (Jones et al., 1998). We re-implemented the outdated and inefficient Theano-based Bayesian optimization implementation of Kusner et al. (2017) (see https://github.com/mkusner/grammarVAE), which was also used by Jin et al. (2018a),

Fig. C.8 Some of the best molecules found using weighted retraining. Numbers indicate the score of each molecule.
C.3 Details on Experimental Setup

using the popular and modern Tensorflow 2.0-based GPflow Gaussian process library (De G. Matthews et al., 2017) to benefit from GPU acceleration.

For computational efficiency, we fit the SGP only on a subset of the data, consisting of the 2000 points with the highest objective function values, and 8000 randomly chosen points. This also has the effect of ensuring that the SGP properly fits the high-performing regions of the data. Disregarding computational efficiency, we nonetheless found that fitting on this data subset remarkably improved performance of the optimization, even using the baseline model (without weighted retraining).

C.3.3 Evaluation Metrics

We report, as a function of the objective function evaluation $b = 1, \ldots, B$, the single best score obtained up until query $b$ (denoted as Top1), and the worst of the 10 and 50 best scores obtained up until evaluation query $b$ (denoted as Top10 and Top50, respectively). Since our goal is to synthesize entities with the desired properties that are both a) syntactically valid and b) novel, we discard any suggested data points which are either a) invalid or b) contained in the training data set (i.e., they are not counted towards the evaluation budget and thus not shown in any of the plots). For statistical significance, we always report the mean plus/minus one standard deviation across multiple random seeds.

C.3.4 2D Shape Task Details

Fig. C.9 shows example images from our 2D squares dataset.

The convolutional variational autoencoder (VAE) architecture may be found in our code. The decoder used an approximately mirror architecture to the encoder with transposed convolutions. Following general conventions, we use a standard normal prior $p(z) = \mathcal{N}(0, 1)$ over the latent variables $z$ and a Bernoulli likelihood $p(x|z)$ to sample binary images. Our implementation used PyTorch (Paszke et al., 2017) and PyTorch Lightning (Falcon, 2019).

C.3.5 Arithmetic Expression Fitting Task

Following Kusner et al. (2017), the dataset we use consists of randomly generated univariate arithmetic expressions from the following grammar:

$$S \rightarrow S \ '+' \ T \mid S \ '+' \ T \mid S \ '/ ' \ T \mid T$$
$$T \rightarrow '(' \ S \ ')' \mid 'sin(' \ S \ ')' \mid 'exp(' \ S \ ')')$$
$$T \rightarrow 'v' \mid '1' \mid '2' \mid '3'$$
where S and T denote non-terminals and the symbol $|$ separates the possible production rules generated from each non-terminal. Every string in the dataset was generated by applying at most 15 production rules, yielding arithmetic expressions such as \( \sin(2) \), \( \frac{v}{3+1} \) and \( \frac{v}{2} \ast \exp(v)/\sin(2\ast v) \), which are all considered to be functions of the variable \( v \).

The objective function we use is defined as \( f(x) = -\log(1 + \text{MSE}(x,x_*)) \), where \( \text{MSE}(x,x_*) \) denotes the mean squared error between \( x \) and the target expression \( x_* = 1/3 \ast v \ast \sin(v\ast v) \), computed over 1000 evenly-spaced values of \( v \) in the interval between \(-10\) and \(+10\). We apply the logarithm function following Kusner et al. (2017) to avoid extremely large MSE values resulting from exponential functions in the generated arithmetic expressions. In contrast to Kusner et al. (2017), we negate the logarithm to arrive at a maximization problem (instead of a minimization problem), to be consistent with our problem formulation and the other experiments. The global maximum of this objective function is \( f(x) = 0 \), achieved at \( x = x_* \) (and \( f(x) < 0 \) otherwise).

In contrast to the original dataset of size 100,000 used by Kusner et al. (2017), which includes the target expression and many other well-performing inputs (thus making the optimization problem easy in theory), we make the task more challenging by discarding the 50% of points with the highest scores, resulting in a dataset of size 50,000 with objective function value distribution shown in Fig. C.3.

Our implementation of the grammar VAE is based on the code from Kusner et al. (2017) provided at https://github.com/mkusner/grammarVAE, which we modified to use Tensorflow 2 (Abadi et al., 2016) and Python 3.

C.3.6 Chemical Design Task

The precise scoring function for a chemical \( x \) is defined as:

\[
\text{score}(x) = \max \left( \log P(x) - \overline{\text{SA}}(x) - \overline{\text{cycle}}(x), -4 \right)
\]

where \( \log P, \text{SA}, \) and cycle are property functions, and the \( \overline{\text{operation}} \) indicates standard normalization of the raw function output using the ZINC training set data (i.e., subtracting the mean of the training set, and dividing by the standard deviation). This is identical to the scoring function from references Dai et al. (2018); Jin et al. (2018a); Kusner et al. (2017); You et al. (2018); Zhou et al. (2019), except that we bound the score below by \(-4\) to prevent points with highly-negative scores from substantially impacting the optimization procedure. Functionally, because this is a maximization task, this makes little difference to the scoring of the outcomes, but does substantially help the optimization.
C.3 Details on Experimental Setup

Our code for the junction tree VAE is a modified version of the “fast jtnn” code from the authors of Jin et al. (2018a) (available at https://github.com/wengong-jin/icml18-jtnn). We adapted the code to be backward-compatible with their original pre-trained model, and to use PyTorch Lightning.

C.3.7 Other Reproducibility Details

**Range of hyperparameters considered** We originally considered $k$ values in the range $10^1, 10^0, \ldots, 10^{-5}$, and found that there was generally a regime where improvement was minimal, but below a certain $k$ value there was significant improvement (which is consistent with our theory). We chose $k = 10^{-3}$ as an intermediate value that consistently gave good performance across tasks. This value was chosen in advance of running our final experiments (i.e. we had preliminary but incomplete results with other $k$ values, then chose $k = 10^{-3}$, and then got our main results). The retraining frequency of 50 was chosen arbitrarily in advance of doing the experiments (specifically it was chosen because it would entail retraining 10 times in our 500 epochs of optimization). The hyperparameters for model design and learning were dictated by the papers whose models we chose, except for the convolutional neural network for the shape task, where we chose a generic architecture. For the baseline methods (i.e. DbAS, CEM-PI, FBVAE, and RWR), we identified the best hyperparameter settings using a grid search over a reasonable range. We used the following hyperparameter settings: a quantile parameter of 0.95 for DbAS, CEM-PI and FBVAE (for all benchmarks), a retrain frequency of 200 for all baselines and for all benchmarks, an exponential coefficient of $10^{-3}$ (for the shapes task) and $10^{-1}$ (for the expression and chemical design tasks) for RWR, and a noise variance of 10 (for the shapes task) and 0.1 (for the expression and chemical design tasks) for DbAS.

**Average run time for each result** All experiments were performed using a single GPU. Runtime results are given in Table C.2.

**Computing infrastructure used** All experiments were done using a single GPU (either NVIDIA P100, 2070 Ti, or 1080 Ti). In practice, a lot of the experiments were run on a high-performance computing cluster to allow multiple experiments to be run in parallel, although this was strictly for convenience: in principle, all experiments could be done on a single machine with one GPU.
<table>
<thead>
<tr>
<th>Experiment</th>
<th>GPU hours per run</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shapes (model pre-training)</td>
<td>0:20</td>
</tr>
<tr>
<td>Shapes (optim., retraining)</td>
<td>0:20</td>
</tr>
<tr>
<td>Shapes (optim., no retraining)</td>
<td>0:01</td>
</tr>
<tr>
<td>Expressions (optim., retraining)</td>
<td>3:15</td>
</tr>
<tr>
<td>Expressions (optim., no retraining)</td>
<td>1:45</td>
</tr>
<tr>
<td>Chemical Design (optim., retraining)</td>
<td>5:00</td>
</tr>
<tr>
<td>Chemical Design (optim., no retraining)</td>
<td>3:00</td>
</tr>
</tbody>
</table>

*Table C.2* Approximate runtimes of main experiments
**Fig. C.9** Sample images from our 2D squares dataset.
Appendix D

Neural Network Calibration

D.1 Derivations of the Wasserstein Pruning Objective

D.1.1 Derivation of the Initial Wasserstein Pruning Objective

We now provide a derivation of the initial Wasserstein pruning objective in Eq. (5.12). Note that, for our linearized model (described in Section 2.4.2), the true posterior $p(\theta | D)$ is either Gaussian or approximately Gaussian. Additionally, the approximate posterior $q_{S}(\theta) = q(\theta_{S}) \prod_{r} \delta(\theta_{r} - \hat{\theta}_{r})$ can be seen as a degenerate Gaussian in which rows and columns of the covariance matrix are zeroed out. Thus, we consider the squared 2-Wasserstein distance between two Gaussian distributions $\mathcal{N}(\mu_{1}, \Sigma_{1})$ and $\mathcal{N}(\mu_{2}, \Sigma_{2})$, which has the following closed-form expression (Givens et al., 1984):

$$W_{2}(\mathcal{N}(\mu_{1}, \Sigma_{1}), \mathcal{N}(\mu_{2}, \Sigma_{2}))^{2} = \|\mu_{1} - \mu_{2}\|_{2}^{2} + \text{Tr} \left( \Sigma_{1} + \Sigma_{2} - 2 \left( \Sigma_{2}^{1/2} \Sigma_{1} \Sigma_{2}^{1/2} \right)^{1/2} \right).$$

(D.1)

In this case both distributions have the same mean: $\mu_{1} = \mu_{2} = \hat{\theta}$. The true posterior’s covariance matrix is the inverse GGN matrix, i.e. $\Sigma_{1} = G^{-1}$. For the approximate posterior $\Sigma_{2} = G_{S+}^{-1}$, which is equal to $G_{S}^{-1}$ (the inverse GGN matrix of the subnetwork) padded with zeros at the positions corresponding to point estimated weights $\theta_{r}$, matching the shape of $G^{-1}$. Alternatively, but equivalently, we can define $G_{S+}^{-1} = M_{S} \odot G^{-1}$, where $\odot$ is the Hadamard product, and $M_{S}$ is a mask matrix with zeros in the rows and columns corresponding to $\theta_{r}$, i.e. the rows and columns corresponding to weights not included in the subnetwork. This

---

1This also holds for our case of a degenerate Gaussian with singular covariance matrix (Givens et al., 1984).
D.1.2 Derivation of the Simplified Wasserstein Pruning Objective

We now provide a derivation of the simplified Wasserstein pruning objective in Eq. (5.13). For $G = \text{diag}(\sigma^2_1, \ldots, \sigma^2_D)$, the Wasserstein pruning objective in Eq. (5.12) simplifies to

$$W_2(p(\theta \mid D), q_S(\theta))^2 = \text{Tr} \left( G^{-1} + G_{S+}^{-1} - 2 \left( G_{S+}^{-1/2} G^{-1} G_{S+}^{-1/2} \right)^{1/2} \right) = \text{Tr} \left( G^{-1} + G_{S+}^{-1} - 2 \left( G_{S+}^{-1/2} G^{-1} G_{S+}^{-1/2} \right)^{1/2} \right).$$

where $m_d$ is the $d^{th}$ diagonal element of $M_S$, i.e. $m_d = 1$ if $\theta_d$ is included in the subnetwork or 0 otherwise.

D.2 Updating the Prior Precision for Uncertainty Estimation With Subnetworks

As described in Section 2.4.2, the linearised Laplace method can be understood as approximating our NN with a basis function linear model, where the Jacobian of the NN evaluated at $x, J(x) \in \mathbb{R}^{O \times D}$ represents the feature expansion. When employing an isotropic Gaussian prior with precision $\lambda$, and for a given output dimension $i$, this formulation corresponds to a Gaussian process with kernel

$$k_i(x, x') = \lambda^{-1} J(x)_i J(x')_i^\top = \lambda^{-1} \sum_{d=1}^{D} J(x)_i d J(x')_i d.$$  

(D.2)
D.3 Additional Image Classification Results

For our subnetwork model, the Jacobian feature expansion is $J_S(x) \in \mathcal{R}^{O \times S}$, which is a submatrix of $J(x)$. It follows that the implied kernel will be computed in the same way as Eq. (D.2), removing $D - S$ terms from the sum. The updated prior precision $\lambda_S = \lambda \cdot S/d$ aims to maintain the magnitude of the sum, thus making the kernel corresponding to the subnetwork as similar as possible to that of the full network.

D.3 Additional Image Classification Results

In this section, we provide additional experimental results for image classification tasks.

D.3.1 Comparing the Parameter Efficiency of Subnetwork Linearized Laplace With Deep Ensembles

Despite, the promising results shown by Subnetwork Linearized Laplace in Section 5.5.3, we note that our method has a notably larger space complexity than our baselines. We therefore investigate the parameter efficiency of our method.

Our ResNet18 Model has $\sim 11.2M$ parameters. Our subnetwork’s covariance matrix contains $42,438^2$ parameters. This totals $\sim 1,830M$ parameters. This same amount of memory could be used to store around 163 ensemble elements. In Fig. D.1 we compare our subnetwork Linearized Laplace model with increasingly large ensembles on both rotated MNIST and corrupted CIFAR10. Although the performance of ensembles improves as more networks are added, it plateaus around 15 ensemble elements. This is in agreement with the findings of recent works (Antorán et al., 2020; Ashukha et al., 2020; Lobacheva et al., 2020). At large rotations and corruptions, the log likelihood obtained by Subnetwork Linearised Laplace is greater than the asymptotic value obtained by ensembles. This suggests that using a larger number of parameters in a approximate posterior covariance matrix is a more efficient use of space than saving a large number of ensemble elements. We also note that inference in a very large ensemble requires performing a forward pass for every ensemble element. On the other hand, Linearised Laplace requires performing one backward pass for every output dimension and one forward pass.

D.3.2 Scalability of Subnetwork Linearised Laplace in the Number of Weights

The aim of subnetwork inference is to scale existing posterior approximations to large networks. To further validate that this objective can be achieved, we perform subnetwork
Fig. D.1 Rotated MNIST (left) and Corrupted CIFAR10 (right) results for deep ensembles (Lakshminarayanan et al., 2017) with large numbers of ensemble members (i.e. up to 55). Horizontal axis denotes number of ensemble members, and vertical axis denotes performance in terms of log-likelihood. Straight horizontal lines correspond to the performance of our method, as a reference. Colors denote different levels of rotation (left) and corruption (right).

Inference in ResNet50. We use a similar (slightly smaller) subnetwork size than we used with ResNet18: our subnetwork contains 39,190 / 23,466,560 (0.167%) parameters. The results obtained with this model are displayed in Fig. D.2. Subnetwork inference in ResNet50 improves upon a simple MAP estimate of the weights in terms of both log-likelihood and calibration metrics.

Fig. D.2 MNIST rotation results for ResNet-50, reporting predictive error, log-likelihood (LL), expected calibration error (ECE) and brier score. We choose a subnetwork containing only 0.167% (39,190 / 23,466,560) of the parameters of the full network.
D.3 Additional Image Classification Results

D.3.3 Out-of-Distribution Rejection

In this section we provide additional results on out-of-distribution (OOD) rejection using predictive uncertainty. First, we train our models on a source dataset. We then evaluate them on the test set from our source dataset and on the test set of a target (out-of-distribution) dataset. We expect predictions for the target dataset to be more uncertain than those for the source dataset. Using predictive uncertainty as the discriminative variable we compute the area under ROC for each method under consideration and display them in Table D.1. The CIFAR-SVHN and MNIST-Fashion dataset pairs are chosen following Nalisnick et al. (2019). On the CIFAR-SVHN task, all methods perform similarly, except for ensembles, which clearly does best. On MNIST-Fashion, SWAG performs best, followed by Subnetwork Linearised Laplace and ensembles.

Table D.1 AUC-ROC scores for out-of-distribution detection, using CIFAR10 vs SVHN and MNIST vs FashionMNIST as in- (source) and out-of-distribution (target) datasets, respectively.

<table>
<thead>
<tr>
<th>Source</th>
<th>Target</th>
<th>Ours</th>
<th>Ours (Rand)</th>
<th>Dropout</th>
<th>Diag-Lap</th>
<th>Ensemble</th>
<th>MAP</th>
<th>SWAG</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR10</td>
<td>SVHN</td>
<td>0.85±0.03</td>
<td>0.86±0.02</td>
<td>0.85±0.01</td>
<td>0.85±0.02</td>
<td>0.91±0.00</td>
<td>0.86±0.02</td>
<td>0.83±0.00</td>
</tr>
<tr>
<td>MNIST</td>
<td>Fashion</td>
<td>0.92±0.05</td>
<td>0.75±0.02</td>
<td>0.82±0.12</td>
<td>0.75±0.01</td>
<td>0.90±0.09</td>
<td>0.72±0.03</td>
<td>0.97±0.01</td>
</tr>
</tbody>
</table>

We also simulate a realistic OOD rejection scenario (Filos et al., 2019) by jointly evaluating our models on an in-distribution and an OOD test set. We allow our methods to reject increasing proportions of the data based on predictive entropy before classifying the rest. All predictions on OOD samples are treated as incorrect. Following Nalisnick et al. (2019), we use CIFAR10 vs SVHN and MNIST vs FashionMNIST as in- and out-of-distribution datasets, respectively. Note that the SVHN test set is randomly sub-sampled down to a size of 10,000 to match that of CIFAR10. The results are shown in Fig. D.3. On CIFAR-SVHN all methods perform similarly, with exceptions being ensembles, which perform best and SWAG which does worse. On MNIST-Fashion SWAG performs best, followed by Subnetwork Linearised Laplace. All other methods fail to distinguish very uncertain in-distribution data from low uncertainty OOD points.
Neural Network Calibration

Fig. D.3 Rejection-classification plots.
### Additional Rotation and Corruption Results

We complement our results from Fig. 5.4 in the main text with results on additional calibration metrics: ECE and Brier Score, in Fig. D.4. Please refer to the appendix of Antorán et al. (2020) for a description of these.

For reference, we provide our results from Fig. 5.4 and Fig. D.4 in numerical format in the tables below.

#### Table D.2 MNIST – no rotation.

<table>
<thead>
<tr>
<th></th>
<th>OURS</th>
<th>OURS (RAND)</th>
<th>DROPOUT</th>
<th>DIAG-LAP</th>
<th>ENSEMBLE</th>
<th>MAP</th>
<th>SWAG</th>
<th>VOGN</th>
</tr>
</thead>
<tbody>
<tr>
<td>LL</td>
<td>−0.07±0.01</td>
<td>−0.01±0.00</td>
<td>−0.01±0.00</td>
<td>−0.04±0.03</td>
<td>−0.01±0.00</td>
<td>−0.01±0.00</td>
<td>−0.01±0.00</td>
<td>−0.14±nan</td>
</tr>
<tr>
<td>error</td>
<td>0.01±0.00</td>
<td>0.00±0.00</td>
<td>0.00±0.00</td>
<td>0.01±0.01</td>
<td>0.00±0.00</td>
<td>0.00±0.00</td>
<td>0.00±0.00</td>
<td>0.01±nan</td>
</tr>
<tr>
<td>ECE</td>
<td>0.05±0.01</td>
<td>0.00±0.00</td>
<td>0.00±0.00</td>
<td>0.00±0.00</td>
<td>0.00±0.00</td>
<td>0.00±0.00</td>
<td>0.00±0.00</td>
<td>0.10±nan</td>
</tr>
<tr>
<td>brier score</td>
<td>0.02±0.00</td>
<td>0.01±0.00</td>
<td>0.01±0.00</td>
<td>0.02±0.01</td>
<td>0.01±0.00</td>
<td>0.01±0.00</td>
<td>0.01±0.00</td>
<td>0.04±nan</td>
</tr>
</tbody>
</table>

#### Table D.3 MNIST – 15° rotation.

<table>
<thead>
<tr>
<th></th>
<th>OURS</th>
<th>OURS (RAND)</th>
<th>DROPOUT</th>
<th>DIAG-LAP</th>
<th>ENSEMBLE</th>
<th>MAP</th>
<th>SWAG</th>
<th>VOGN</th>
</tr>
</thead>
<tbody>
<tr>
<td>LL</td>
<td>−0.14±0.02</td>
<td>−0.05±0.00</td>
<td>−0.05±0.00</td>
<td>−0.11±0.08</td>
<td>−0.04±0.00</td>
<td>−0.05±0.00</td>
<td>−0.04±0.00</td>
<td>−0.19±nan</td>
</tr>
<tr>
<td>error</td>
<td>0.02±0.00</td>
<td>0.02±0.00</td>
<td>0.01±0.00</td>
<td>0.03±0.02</td>
<td>0.01±0.00</td>
<td>0.02±0.00</td>
<td>0.01±0.00</td>
<td>0.02±nan</td>
</tr>
<tr>
<td>ECE</td>
<td>0.08±0.01</td>
<td>0.00±0.00</td>
<td>0.00±0.00</td>
<td>0.01±0.01</td>
<td>0.00±0.00</td>
<td>0.00±0.00</td>
<td>0.00±0.00</td>
<td>0.12±nan</td>
</tr>
<tr>
<td>brier score</td>
<td>0.05±0.01</td>
<td>0.03±0.00</td>
<td>0.02±0.00</td>
<td>0.05±0.03</td>
<td>0.02±0.00</td>
<td>0.02±0.00</td>
<td>0.02±0.00</td>
<td>0.07±nan</td>
</tr>
</tbody>
</table>

#### Table D.4 MNIST – 30° rotation.

<table>
<thead>
<tr>
<th></th>
<th>OURS</th>
<th>OURS (RAND)</th>
<th>DROPOUT</th>
<th>DIAG-LAP</th>
<th>ENSEMBLE</th>
<th>MAP</th>
<th>SWAG</th>
<th>VOGN</th>
</tr>
</thead>
<tbody>
<tr>
<td>LL</td>
<td>−0.42±0.04</td>
<td>−0.36±0.01</td>
<td>−0.32±0.02</td>
<td>−0.44±0.06</td>
<td>−0.28±0.02</td>
<td>−0.39±0.01</td>
<td>−0.30±0.00</td>
<td>−0.51±nan</td>
</tr>
<tr>
<td>error</td>
<td>0.11±0.01</td>
<td>0.10±0.00</td>
<td>0.09±0.01</td>
<td>0.12±0.01</td>
<td>0.08±0.01</td>
<td>0.10±0.00</td>
<td>0.08±0.00</td>
<td>0.14±nan</td>
</tr>
<tr>
<td>ECE</td>
<td>0.10±0.02</td>
<td>0.04±0.01</td>
<td>0.03±0.00</td>
<td>0.06±0.01</td>
<td>0.02±0.00</td>
<td>0.05±0.00</td>
<td>0.04±0.00</td>
<td>0.13±nan</td>
</tr>
<tr>
<td>brier score</td>
<td>0.19±0.02</td>
<td>0.16±0.00</td>
<td>0.14±0.01</td>
<td>0.18±0.02</td>
<td>0.12±0.01</td>
<td>0.16±0.00</td>
<td>0.12±0.00</td>
<td>0.23±nan</td>
</tr>
</tbody>
</table>

#### Table D.5 MNIST – 45° rotation.

<table>
<thead>
<tr>
<th></th>
<th>OURS</th>
<th>OURS (RAND)</th>
<th>DROPOUT</th>
<th>DIAG-LAP</th>
<th>ENSEMBLE</th>
<th>MAP</th>
<th>SWAG</th>
<th>VOGN</th>
</tr>
</thead>
<tbody>
<tr>
<td>LL</td>
<td>−1.09±0.03</td>
<td>−1.60±0.05</td>
<td>−1.44±0.11</td>
<td>−1.68±0.20</td>
<td>−1.36±0.07</td>
<td>−1.75±0.06</td>
<td>−1.35±0.02</td>
<td>−1.15±nan</td>
</tr>
<tr>
<td>error</td>
<td>0.36±0.01</td>
<td>0.35±0.01</td>
<td>0.35±0.01</td>
<td>0.35±0.03</td>
<td>0.31±0.01</td>
<td>0.35±0.01</td>
<td>0.29±0.00</td>
<td>0.40±nan</td>
</tr>
<tr>
<td>ECE</td>
<td>0.03±0.01</td>
<td>0.22±0.01</td>
<td>0.19±0.02</td>
<td>0.22±0.02</td>
<td>0.17±0.01</td>
<td>0.23±0.01</td>
<td>0.18±0.00</td>
<td>0.01±nan</td>
</tr>
<tr>
<td>brier score</td>
<td>0.49±0.02</td>
<td>0.55±0.02</td>
<td>0.52±0.02</td>
<td>0.55±0.04</td>
<td>0.48±0.02</td>
<td>0.56±0.02</td>
<td>0.46±0.01</td>
<td>0.53±nan</td>
</tr>
</tbody>
</table>

#### Table D.6 MNIST – 60° rotation.

<table>
<thead>
<tr>
<th></th>
<th>OURS</th>
<th>OURS (RAND)</th>
<th>DROPOUT</th>
<th>DIAG-LAP</th>
<th>ENSEMBLE</th>
<th>MAP</th>
<th>SWAG</th>
<th>VOGN</th>
</tr>
</thead>
<tbody>
<tr>
<td>LL</td>
<td>−2.10±0.03</td>
<td>−3.54±0.18</td>
<td>−3.54±0.23</td>
<td>−4.11±0.66</td>
<td>−3.60±0.10</td>
<td>−4.29±0.21</td>
<td>−2.95±0.08</td>
<td>−1.92±nan</td>
</tr>
<tr>
<td>error</td>
<td>0.63±0.01</td>
<td>0.63±0.01</td>
<td>0.62±0.01</td>
<td>0.62±0.05</td>
<td>0.61±0.01</td>
<td>0.63±0.01</td>
<td>0.53±0.02</td>
<td>0.64±nan</td>
</tr>
<tr>
<td>ECE</td>
<td>0.25±0.02</td>
<td>0.40±0.02</td>
<td>0.43±0.02</td>
<td>0.47±0.06</td>
<td>0.42±0.01</td>
<td>0.48±0.02</td>
<td>0.36±0.02</td>
<td>0.17±nan</td>
</tr>
<tr>
<td>brier score</td>
<td>0.85±0.02</td>
<td>1.04±0.03</td>
<td>1.00±0.03</td>
<td>1.05±0.10</td>
<td>0.98±0.02</td>
<td>1.07±0.03</td>
<td>0.86±0.03</td>
<td>0.80±nan</td>
</tr>
</tbody>
</table>
### Table D.7 MNIST – 75° rotation.

<table>
<thead>
<tr>
<th></th>
<th>OURS</th>
<th>OURS (RAND)</th>
<th>DROPOUT</th>
<th>DIAG-LAP</th>
<th>ENSEMBLE</th>
<th>MAP</th>
<th>SWAG</th>
<th>VOGN</th>
</tr>
</thead>
<tbody>
<tr>
<td>LL error</td>
<td>-3.02 ± 0.07</td>
<td>-5.93 ± 0.28</td>
<td>-5.49 ± 0.38</td>
<td>-6.92 ± 0.32</td>
<td>-5.74 ± 0.15</td>
<td>-6.63 ± 0.33</td>
<td>-4.46 ± 0.18</td>
<td>-2.54 ± 0.01</td>
</tr>
<tr>
<td>ECE</td>
<td>0.86 ± 0.07</td>
<td>0.79 ± 0.01</td>
<td>0.79 ± 0.01</td>
<td>0.84 ± 0.00</td>
<td>0.78 ± 0.00</td>
<td>0.79 ± 0.01</td>
<td>0.72 ± 0.02</td>
<td>0.77 ± 0.01</td>
</tr>
<tr>
<td>Brier score</td>
<td>1.08 ± 0.04</td>
<td>1.34 ± 0.04</td>
<td>1.30 ± 0.02</td>
<td>1.39 ± 0.01</td>
<td>1.29 ± 0.02</td>
<td>1.37 ± 0.04</td>
<td>1.17 ± 0.04</td>
<td>0.95 ± 0.01</td>
</tr>
</tbody>
</table>

### Table D.8 MNIST – 90° rotation.

<table>
<thead>
<tr>
<th></th>
<th>OURS</th>
<th>OURS (RAND)</th>
<th>DROPOUT</th>
<th>DIAG-LAP</th>
<th>ENSEMBLE</th>
<th>MAP</th>
<th>SWAG</th>
<th>VOGN</th>
</tr>
</thead>
<tbody>
<tr>
<td>LL error</td>
<td>-3.35 ± 0.13</td>
<td>-6.46 ± 0.15</td>
<td>-6.18 ± 0.41</td>
<td>-7.32 ± 0.67</td>
<td>-6.39 ± 0.17</td>
<td>-7.18 ± 0.22</td>
<td>-5.63 ± 0.12</td>
<td>-2.91 ± 0.01</td>
</tr>
<tr>
<td>ECE</td>
<td>0.84 ± 0.02</td>
<td>0.84 ± 0.01</td>
<td>0.84 ± 0.01</td>
<td>0.85 ± 0.01</td>
<td>0.84 ± 0.01</td>
<td>0.84 ± 0.01</td>
<td>0.82 ± 0.02</td>
<td>0.81 ± 0.01</td>
</tr>
<tr>
<td>Brier score</td>
<td>1.13 ± 0.03</td>
<td>1.40 ± 0.05</td>
<td>1.37 ± 0.01</td>
<td>1.44 ± 0.04</td>
<td>1.36 ± 0.01</td>
<td>1.43 ± 0.05</td>
<td>1.34 ± 0.02</td>
<td>1.02 ± 0.01</td>
</tr>
</tbody>
</table>

### Table D.9 MNIST – 105° rotation.

<table>
<thead>
<tr>
<th></th>
<th>OURS</th>
<th>OURS (RAND)</th>
<th>DROPOUT</th>
<th>DIAG-LAP</th>
<th>ENSEMBLE</th>
<th>MAP</th>
<th>SWAG</th>
<th>VOGN</th>
</tr>
</thead>
<tbody>
<tr>
<td>LL error</td>
<td>-3.59 ± 0.05</td>
<td>-7.06 ± 0.45</td>
<td>-7.69 ± 0.99</td>
<td>-7.01 ± 0.17</td>
<td>-7.87 ± 0.53</td>
<td>-6.28 ± 0.19</td>
<td>-3.10 ± 0.01</td>
<td>-3.10 ± 0.01</td>
</tr>
<tr>
<td>ECE</td>
<td>0.85 ± 0.02</td>
<td>0.84 ± 0.01</td>
<td>0.85 ± 0.01</td>
<td>0.84 ± 0.01</td>
<td>0.84 ± 0.01</td>
<td>0.84 ± 0.01</td>
<td>0.81 ± 0.00</td>
<td>0.81 ± 0.01</td>
</tr>
<tr>
<td>Brier score</td>
<td>1.17 ± 0.05</td>
<td>1.44 ± 0.07</td>
<td>1.38 ± 0.02</td>
<td>1.44 ± 0.04</td>
<td>1.40 ± 0.01</td>
<td>1.46 ± 0.07</td>
<td>1.34 ± 0.02</td>
<td>1.07 ± 0.01</td>
</tr>
</tbody>
</table>

### Table D.10 MNIST – 120° rotation.

<table>
<thead>
<tr>
<th></th>
<th>OURS</th>
<th>OURS (RAND)</th>
<th>DROPOUT</th>
<th>DIAG-LAP</th>
<th>ENSEMBLE</th>
<th>MAP</th>
<th>SWAG</th>
<th>VOGN</th>
</tr>
</thead>
<tbody>
<tr>
<td>LL error</td>
<td>-3.43 ± 0.07</td>
<td>-6.73 ± 0.53</td>
<td>-6.62 ± 0.39</td>
<td>-7.92 ± 0.59</td>
<td>-6.73 ± 0.61</td>
<td>-7.53 ± 0.63</td>
<td>-6.49 ± 0.36</td>
<td>-3.07 ± 0.01</td>
</tr>
<tr>
<td>ECE</td>
<td>0.80 ± 0.02</td>
<td>0.79 ± 0.02</td>
<td>0.78 ± 0.01</td>
<td>0.81 ± 0.01</td>
<td>0.78 ± 0.01</td>
<td>0.79 ± 0.02</td>
<td>0.76 ± 0.02</td>
<td>0.76 ± 0.01</td>
</tr>
<tr>
<td>Brier score</td>
<td>1.10 ± 0.03</td>
<td>1.35 ± 0.07</td>
<td>1.29 ± 0.02</td>
<td>1.39 ± 0.06</td>
<td>1.30 ± 0.01</td>
<td>1.36 ± 0.07</td>
<td>1.27 ± 0.04</td>
<td>1.04 ± 0.01</td>
</tr>
</tbody>
</table>

### Table D.11 MNIST – 135° rotation.

<table>
<thead>
<tr>
<th></th>
<th>OURS</th>
<th>OURS (RAND)</th>
<th>DROPOUT</th>
<th>DIAG-LAP</th>
<th>ENSEMBLE</th>
<th>MAP</th>
<th>SWAG</th>
<th>VOGN</th>
</tr>
</thead>
<tbody>
<tr>
<td>LL error</td>
<td>-3.24 ± 0.06</td>
<td>-6.43 ± 0.38</td>
<td>-6.46 ± 0.28</td>
<td>-7.05 ± 0.88</td>
<td>-6.57 ± 0.10</td>
<td>-7.24 ± 0.48</td>
<td>-6.40 ± 0.37</td>
<td>-2.89 ± 0.01</td>
</tr>
<tr>
<td>ECE</td>
<td>0.71 ± 0.02</td>
<td>0.71 ± 0.02</td>
<td>0.70 ± 0.01</td>
<td>0.71 ± 0.01</td>
<td>0.70 ± 0.01</td>
<td>0.71 ± 0.02</td>
<td>0.70 ± 0.02</td>
<td>0.67 ± 0.01</td>
</tr>
<tr>
<td>Brier score</td>
<td>0.32 ± 0.01</td>
<td>0.55 ± 0.03</td>
<td>0.52 ± 0.01</td>
<td>0.56 ± 0.02</td>
<td>0.52 ± 0.01</td>
<td>0.56 ± 0.03</td>
<td>0.53 ± 0.02</td>
<td>0.25 ± 0.01</td>
</tr>
</tbody>
</table>

### Table D.12 MNIST – 150° rotation.

<table>
<thead>
<tr>
<th></th>
<th>OURS</th>
<th>OURS (RAND)</th>
<th>DROPOUT</th>
<th>DIAG-LAP</th>
<th>ENSEMBLE</th>
<th>MAP</th>
<th>SWAG</th>
<th>VOGN</th>
</tr>
</thead>
<tbody>
<tr>
<td>LL error</td>
<td>-3.25 ± 0.05</td>
<td>-6.56 ± 0.18</td>
<td>-6.62 ± 0.33</td>
<td>-7.04 ± 0.36</td>
<td>-6.88 ± 0.11</td>
<td>-7.41 ± 0.25</td>
<td>-6.39 ± 0.27</td>
<td>-2.69 ± 0.01</td>
</tr>
<tr>
<td>ECE</td>
<td>0.63 ± 0.02</td>
<td>0.63 ± 0.01</td>
<td>0.63 ± 0.00</td>
<td>0.65 ± 0.01</td>
<td>0.62 ± 0.01</td>
<td>0.63 ± 0.01</td>
<td>0.63 ± 0.01</td>
<td>0.60 ± 0.01</td>
</tr>
<tr>
<td>Brier score</td>
<td>0.29 ± 0.01</td>
<td>0.50 ± 0.01</td>
<td>0.48 ± 0.01</td>
<td>0.52 ± 0.01</td>
<td>0.48 ± 0.01</td>
<td>0.51 ± 0.01</td>
<td>0.49 ± 0.01</td>
<td>0.23 ± 0.01</td>
</tr>
</tbody>
</table>

Table D.7 MNIST – 75° rotation.
Fig. D.4 Full MNIST rotation and CIFAR10 corruption results, for ResNet-18, reporting predictive error, log-likelihood (LL), expected calibration error (ECE) and Brier score, respectively (from top to bottom).
### Table D.13 MNIST – 165° rotation.

<table>
<thead>
<tr>
<th></th>
<th>OURS</th>
<th>OURS (RAND)</th>
<th>DROPOUT</th>
<th>DIAG-LAP</th>
<th>ENSEMBLE</th>
<th>MAP</th>
<th>SWAG</th>
<th>VOGN</th>
</tr>
</thead>
<tbody>
<tr>
<td>LL error</td>
<td>3.42±0.12</td>
<td>7.01±0.15</td>
<td>7.08±0.39</td>
<td>7.80±0.12</td>
<td>7.51±0.11</td>
<td>7.91±0.18</td>
<td>6.63±0.24</td>
<td>2.67±0.15</td>
</tr>
<tr>
<td>error</td>
<td>0.58±0.01</td>
<td>0.58±0.01</td>
<td>0.58±0.01</td>
<td>0.58±0.00</td>
<td>0.57±0.01</td>
<td>0.58±0.01</td>
<td>0.59±0.00</td>
<td>0.56±0.00</td>
</tr>
<tr>
<td>ECE</td>
<td>0.32±0.02</td>
<td>0.49±0.01</td>
<td>0.48±0.01</td>
<td>0.49±0.01</td>
<td>0.48±0.00</td>
<td>0.51±0.01</td>
<td>0.48±0.00</td>
<td>0.25±0.00</td>
</tr>
<tr>
<td>brier score</td>
<td>0.90±0.02</td>
<td>1.05±0.01</td>
<td>1.04±0.01</td>
<td>1.05±0.01</td>
<td>1.03±0.01</td>
<td>1.07±0.02</td>
<td>1.03±0.01</td>
<td>0.82±0.00</td>
</tr>
</tbody>
</table>

### Table D.14 MNIST – 180° rotation.

<table>
<thead>
<tr>
<th></th>
<th>OURS</th>
<th>OURS (RAND)</th>
<th>DROPOUT</th>
<th>DIAG-LAP</th>
<th>ENSEMBLE</th>
<th>MAP</th>
<th>SWAG</th>
<th>VOGN</th>
</tr>
</thead>
<tbody>
<tr>
<td>LL error</td>
<td>3.32±0.13</td>
<td>6.63±0.18</td>
<td>6.87±0.32</td>
<td>7.10±0.47</td>
<td>7.16±0.16</td>
<td>7.43±0.20</td>
<td>6.61±0.22</td>
<td>2.71±0.15</td>
</tr>
<tr>
<td>error</td>
<td>0.56±0.01</td>
<td>0.56±0.01</td>
<td>0.56±0.00</td>
<td>0.55±0.01</td>
<td>0.55±0.01</td>
<td>0.56±0.00</td>
<td>0.57±0.00</td>
<td>0.55±0.00</td>
</tr>
<tr>
<td>ECE</td>
<td>0.29±0.02</td>
<td>0.46±0.00</td>
<td>0.46±0.00</td>
<td>0.46±0.00</td>
<td>0.46±0.00</td>
<td>0.46±0.00</td>
<td>0.47±0.00</td>
<td>0.25±0.00</td>
</tr>
<tr>
<td>brier score</td>
<td>0.86±0.02</td>
<td>1.00±0.01</td>
<td>0.99±0.01</td>
<td>0.99±0.01</td>
<td>0.99±0.00</td>
<td>1.01±0.02</td>
<td>1.01±0.00</td>
<td>0.82±0.00</td>
</tr>
</tbody>
</table>

### Table D.15 CIFAR10 – no corruption.

<table>
<thead>
<tr>
<th></th>
<th>OURS</th>
<th>OURS (RAND)</th>
<th>DROPOUT</th>
<th>DIAG-LAP</th>
<th>ENSEMBLE</th>
<th>MAP</th>
<th>SWAG</th>
<th>VOGN</th>
</tr>
</thead>
<tbody>
<tr>
<td>LL error</td>
<td>0.27±0.00</td>
<td>0.43±0.01</td>
<td>0.37±0.01</td>
<td>0.50±0.02</td>
<td>0.21±0.01</td>
<td>0.46±0.02</td>
<td>0.48±0.01</td>
<td>0.61±0.00</td>
</tr>
<tr>
<td>error</td>
<td>0.09±0.00</td>
<td>0.08±0.00</td>
<td>0.08±0.00</td>
<td>0.09±0.00</td>
<td>0.06±0.00</td>
<td>0.08±0.00</td>
<td>0.11±0.00</td>
<td>0.21±0.00</td>
</tr>
<tr>
<td>ECE</td>
<td>0.01±0.00</td>
<td>0.06±0.00</td>
<td>0.04±0.00</td>
<td>0.06±0.00</td>
<td>0.01±0.00</td>
<td>0.06±0.00</td>
<td>0.07±0.00</td>
<td>0.03±0.00</td>
</tr>
<tr>
<td>brier score</td>
<td>0.13±0.00</td>
<td>0.14±0.00</td>
<td>0.13±0.00</td>
<td>0.15±0.00</td>
<td>0.09±0.00</td>
<td>0.14±0.00</td>
<td>0.17±0.00</td>
<td>0.30±0.00</td>
</tr>
</tbody>
</table>

### Table D.16 CIFAR10 – level 1 corruption.

<table>
<thead>
<tr>
<th></th>
<th>OURS</th>
<th>OURS (RAND)</th>
<th>DROPOUT</th>
<th>DIAG-LAP</th>
<th>ENSEMBLE</th>
<th>MAP</th>
<th>SWAG</th>
<th>VOGN</th>
</tr>
</thead>
<tbody>
<tr>
<td>LL error</td>
<td>0.51±0.01</td>
<td>0.91±0.06</td>
<td>0.80±0.02</td>
<td>1.03±0.02</td>
<td>0.50±0.02</td>
<td>0.96±0.02</td>
<td>0.89±0.02</td>
<td>0.99±0.00</td>
</tr>
<tr>
<td>error</td>
<td>0.17±0.01</td>
<td>0.16±0.00</td>
<td>0.16±0.00</td>
<td>0.17±0.00</td>
<td>0.13±0.00</td>
<td>0.16±0.00</td>
<td>0.17±0.00</td>
<td>0.32±0.00</td>
</tr>
<tr>
<td>ECE</td>
<td>0.03±0.00</td>
<td>0.11±0.00</td>
<td>0.10±0.00</td>
<td>0.13±0.00</td>
<td>0.04±0.00</td>
<td>0.12±0.00</td>
<td>0.11±0.00</td>
<td>0.03±0.00</td>
</tr>
<tr>
<td>brier score</td>
<td>0.24±0.00</td>
<td>0.27±0.00</td>
<td>0.25±0.00</td>
<td>0.29±0.00</td>
<td>0.19±0.00</td>
<td>0.27±0.00</td>
<td>0.29±0.00</td>
<td>0.44±0.00</td>
</tr>
</tbody>
</table>

### Table D.17 CIFAR10 – level 2 corruption.

<table>
<thead>
<tr>
<th></th>
<th>OURS</th>
<th>OURS (RAND)</th>
<th>DROPOUT</th>
<th>DIAG-LAP</th>
<th>ENSEMBLE</th>
<th>MAP</th>
<th>SWAG</th>
<th>VOGN</th>
</tr>
</thead>
<tbody>
<tr>
<td>LL error</td>
<td>0.73±0.01</td>
<td>1.29±0.02</td>
<td>1.20±0.02</td>
<td>1.50±0.12</td>
<td>0.80±0.01</td>
<td>1.40±0.01</td>
<td>1.21±0.00</td>
<td>1.31±0.00</td>
</tr>
<tr>
<td>error</td>
<td>0.23±0.00</td>
<td>0.22±0.00</td>
<td>0.22±0.00</td>
<td>0.23±0.00</td>
<td>0.19±0.00</td>
<td>0.22±0.00</td>
<td>0.22±0.00</td>
<td>0.40±0.00</td>
</tr>
<tr>
<td>ECE</td>
<td>0.06±0.00</td>
<td>0.16±0.00</td>
<td>0.14±0.00</td>
<td>0.17±0.01</td>
<td>0.07±0.00</td>
<td>0.16±0.00</td>
<td>0.15±0.00</td>
<td>0.10±0.00</td>
</tr>
<tr>
<td>brier score</td>
<td>0.33±0.00</td>
<td>0.37±0.01</td>
<td>0.35±0.01</td>
<td>0.40±0.02</td>
<td>0.28±0.00</td>
<td>0.37±0.01</td>
<td>0.37±0.00</td>
<td>0.56±0.00</td>
</tr>
</tbody>
</table>

### Table D.18 CIFAR10 – level 3 corruption.

<table>
<thead>
<tr>
<th></th>
<th>OURS</th>
<th>OURS (RAND)</th>
<th>DROPOUT</th>
<th>DIAG-LAP</th>
<th>ENSEMBLE</th>
<th>MAP</th>
<th>SWAG</th>
<th>VOGN</th>
</tr>
</thead>
<tbody>
<tr>
<td>LL error</td>
<td>1.06±0.02</td>
<td>2.06±0.12</td>
<td>1.85±0.07</td>
<td>2.13±0.17</td>
<td>1.28±0.03</td>
<td>2.18±0.08</td>
<td>1.63±0.03</td>
<td>1.83±0.00</td>
</tr>
<tr>
<td>error</td>
<td>0.32±0.01</td>
<td>0.31±0.01</td>
<td>0.31±0.01</td>
<td>0.31±0.01</td>
<td>0.28±0.00</td>
<td>0.31±0.01</td>
<td>0.28±0.00</td>
<td>0.51±0.00</td>
</tr>
<tr>
<td>ECE</td>
<td>0.11±0.01</td>
<td>0.24±0.01</td>
<td>0.21±0.01</td>
<td>0.24±0.01</td>
<td>0.12±0.00</td>
<td>0.24±0.01</td>
<td>0.20±0.00</td>
<td>0.19±0.00</td>
</tr>
<tr>
<td>brier score</td>
<td>0.46±0.01</td>
<td>0.54±0.02</td>
<td>0.50±0.02</td>
<td>0.54±0.03</td>
<td>0.42±0.00</td>
<td>0.54±0.02</td>
<td>0.47±0.01</td>
<td>0.72±0.00</td>
</tr>
</tbody>
</table>
### Table D.19 CIFAR10 – level 4 corruption.

<table>
<thead>
<tr>
<th></th>
<th>Ours</th>
<th>Ours (Rand)</th>
<th>Dropout</th>
<th>Diag-Lap</th>
<th>Ensemble</th>
<th>MAP</th>
<th>SWAG</th>
<th>VOGN</th>
</tr>
</thead>
<tbody>
<tr>
<td>LL error</td>
<td>-1.25±0.03</td>
<td>-2.43±0.18</td>
<td>-2.28±0.10</td>
<td>-2.54±0.18</td>
<td>-1.56±0.05</td>
<td>-2.57±0.15</td>
<td>-1.95±0.04</td>
<td>-1.99±num</td>
</tr>
<tr>
<td>ECE</td>
<td>0.36±0.01</td>
<td>0.35±0.01</td>
<td>0.35±0.01</td>
<td>0.35±0.01</td>
<td>0.32±0.01</td>
<td>0.35±0.01</td>
<td>0.32±0.00</td>
<td>0.54±num</td>
</tr>
<tr>
<td>Brier score</td>
<td>0.51±0.02</td>
<td>0.60±0.03</td>
<td>0.57±0.01</td>
<td>0.61±0.02</td>
<td>0.47±0.01</td>
<td>0.60±0.03</td>
<td>0.53±0.00</td>
<td>0.76±num</td>
</tr>
</tbody>
</table>

### Table D.20 CIFAR10 – level 5 corruption.

<table>
<thead>
<tr>
<th></th>
<th>Ours</th>
<th>Ours (Rand)</th>
<th>Dropout</th>
<th>Diag-Lap</th>
<th>Ensemble</th>
<th>MAP</th>
<th>SWAG</th>
<th>VOGN</th>
</tr>
</thead>
<tbody>
<tr>
<td>LL error</td>
<td>-1.47±0.03</td>
<td>-2.82±0.11</td>
<td>-2.71±0.13</td>
<td>-3.20±0.13</td>
<td>-1.88±0.05</td>
<td>-3.03±0.10</td>
<td>-2.31±0.09</td>
<td>-2.00±num</td>
</tr>
<tr>
<td>ECE</td>
<td>0.41±0.00</td>
<td>0.40±0.01</td>
<td>0.40±0.01</td>
<td>0.41±0.01</td>
<td>0.37±0.01</td>
<td>0.40±0.00</td>
<td>0.36±0.01</td>
<td>0.54±num</td>
</tr>
<tr>
<td>Brier score</td>
<td>0.58±0.00</td>
<td>0.69±0.01</td>
<td>0.65±0.01</td>
<td>0.72±0.03</td>
<td>0.55±0.01</td>
<td>0.69±0.01</td>
<td>0.61±0.01</td>
<td>0.75±num</td>
</tr>
</tbody>
</table>
D.4 Experimental Setup

D.4.1 Toy Experiments

We train a single, 2 hidden layer network, with 50 hidden ReLU units per layer using MAP inference until convergence. Specifically, we use SGD with a learning rate of $1 \times 10^{-3}$, momentum of 0.9 and weight decay of $1 \times 10^{-4}$. We use a batch size of 512. The objective we optimise is the Gaussian log-likelihood of our data, where the mean is outputted by the network and the variance is a hyperparameter learnt jointly with NN parameters by SGD. This variance parameters is shared among all datapoints. Once the network is trained, we perform post-hoc inference on it using different approaches. Since all of these involve the linearized approximation, the mean prediction is the same for all methods. Only their uncertainty estimates vary.

Note that while for this toy example, we could in principle use the full covariance matrix for the purpose of subnetwork selection, we still just use its diagonal (as described in Section 5.3) for consistency. We use GGN Laplace inference over network weights (not biases) in combination with the linearized predictive distribution in Eq. (2.58). Thus, all approaches considered share their predictive mean, allowing us to better compare their uncertainty estimates.

All approaches share a single prior precision of $\lambda = 3$, scaled as $\lambda_S = \lambda \cdot s/D$. We choose this prior precision such that the full covariance approach (optimistic baseline), where $\lambda_S = \lambda$, presents reasonable results. We first tried a precision of 1 and found the full covariance approach to produce excessively large errorbars (covering the whole plot). A value of 3 produces more reasonable results.

Final layer inference is performed by computing the full Laplace covariance matrix and discarding all entries except those corresponding to the final layer of the NN. Results for random sub-network selection are obtained with a single sample from a scaled uniform distribution over weight choice.

D.4.2 UCI Experiments

In this experiment, our fully connected NNs have numbers of hidden layers $h_d = \{1, 2\}$ and hidden layer widths $w_d = \{50, 100\}$. For a dataset with input dimension $i_d$, the number of weights is given by $D = (i_d + 1)w_d + (h_d - 1)w_d^2$. Our 2 hidden layer, 100 hidden unit models have a weight count of the order $10^4$. The non-linearity used is ReLU.

We first obtain a MAP estimate of each model’s weights. Specifically, we use SGD with a learning rate of $1 \times 10^{-3}$, momentum of 0.9 and weight decay of $1 \times 10^{-4}$. We use a batch
size of 512. The objective we optimise is the Gaussian log-likelihood of our data, where the mean is outputted by the network and the variance is a hyperparameter learnt jointly with NN parameters by SGD.

For each dataset split, we set aside 15% of the train data as a validation set. We use these for early stopping training. Training runs for a maximum of 2000 epochs but early stops with a patience of 500 if validation performance does not increase. For the larger Protein dataset, these values are 500 and 125. The weight settings which provide best validation performance are kept.

We then perform full network GGN Laplace inference for each model. We also use our proposed Wasserstein rule together with the diagonal Hessian assumption to prune every network’s weight variances such that the number of variances that remain matches the size of every smaller network under consideration. The prior precision used for these steps is chosen such that the resulting predictor’s log-likelihood performance on the validation set is maximised. Specifically, we employ a grid search over the values: \( \lambda : [0.0001, 0.001, 0.1, 0.5, 1, 2.5, 10, 100, 1000] \). In all cases, we employ the linearized predictive in Eq. (2.58). Consequently, networks with the same number of weights make the same mean predictions. Increasing the number of weight variances considered will thus only increase predictive uncertainty.

### D.4.3 Image Experiments

The results shown in Section 5.5.3 and Section D.3 are obtained by training ResNet-18 (and ResNet-50) models using SGD with momentum. For each experiment repetition, we train 7 different models: The first is for: ‘MAP’, ‘Ours’, ‘Ours (Rand)’, ‘SWAG’, ‘Diag-Laplace’ and as the first element of ‘Ensemble’. We train 4 additional ‘Ensemble’ elements, 1 network with ‘Dropout’, and, finally 1 network for ‘VOGN’. The methods ‘Ours’, ‘Ours (Rand)’, ‘SWAG’, and ‘Diag-Laplace’ are applied post training.

For all methods except ‘VOGN’ we use the following training procedure. The (initial) learning rate, momentum, and weight decay are 0.1, 0.9, and \( 1 \times 10^{-4} \), respectively. For ‘MAP’ we use 4 Nvidia P100 GPUs with a total batch size of 2048. For the calculation of the Jacobian in the subnetwork selection phase we use a single P100 GPU with a batch size of 4. For the calculation of the hessian we use a single P100 GPU with a batch size of 2. We train on 1 Nvidia P100 GPU with a batch size of 256 for all other methods. Each dataset is trained for a different number of epochs, shown in Table D.21. We decay the learning rate by a factor of 10 at scheduled epochs, also shown in Table D.21. Otherwise, all methods and datasets share hyperparameters. These hyperparameter settings are the defaults provided by PyTorch for training on ImageNet. We found them to perform well across the board. We report results
obtained at the final training epoch. We do not use a separate validation set to determine the best epoch as we found ResNet-18 and ResNet-50 to not overfit with the chosen schedules.

Table D.21 Per-dataset training configuration for image experiments.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>No. Epochs</th>
<th>LR Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>90</td>
<td>40, 70</td>
</tr>
<tr>
<td>CIFAR10</td>
<td>300</td>
<td>150, 225</td>
</tr>
</tbody>
</table>

For ‘Dropout’, we add dropout to the standard ResNet-50 model (He et al., 2016) in between the 2nd and 3rd convolutions in the bottleneck blocks. This approach follows Zagoruyko and Komodakis (2016) and Ashukha et al. (2020) who add dropout in-between the two convolutions of a WideResNet-50’s basic block. Following Antorán et al. (2020), we choose a dropout probability of 0.1, as they found it to perform better than the value of 0.3 suggested by Ashukha et al. (2020). We use 16 MC samples for predictions. ‘Ensemble’ uses 5 elements for prediction. Ensemble elements differ from each other in their initialisation, which is sampled from the He initialisation distribution (He et al., 2015). We do not use adversarial training as, inline with Ashukha et al. (2020), we do not find it to improve results. For ‘VOGN’ we use the same procedure and hyper-parameters as used by Osawa et al. (2019) in their CIFAR10 experiments, with the exception that we use a learning rate of $1 \times 10^{-3}$ as we we found a value of $1 \times 10^{-4}$ not to result in convergence. We train on a single Nvidia P100 GPU with a batch size of 256. See the authors’ GitHub for more details: github.com/team-approx-bayes/dl-with-bayes/blob/master/distributed/classification/configs/cifar10/resnet18_vogn_bs256_8gpu.json.

We modify the standard ResNet-50 and ResNet-18 architectures such that the first $7 \times 7$ convolution is replaced with a $3 \times 3$ convolution. Additionally, we remove the first max-pooling layer. Following Goyal et al. (2017), we zero-initialise the last batch normalisation layer in residual blocks so that they act as identity functions at the start of training.

At test time, we tune the prior precision used for ‘Ours’, ‘Diag-Laplace’ and ‘SWAG’ approximation on a validation set for each approach individually, as in Kristiadi et al. (2020); Ritter et al. (2018b). We use a grid search from $1 \times 10^{-4}$ to $1 \times 10^{4}$ in logarithmic steps, and then a second, finer-grained grid search between the two best performing values (again with logarithmic steps).
D.4 Experimental Setup

D.4.4 Datasets

The 1d toy dataset used in Section 5.5.1 was taken from Antorán et al. (2020). We obtained it from the authors’ github repo: https://github.com/cambridge-mlg/DUN. Table D.22 summarises the datasets used in Section 5.5.2.

We employ the Wine, Kin8nm and Protein datasets, together with their gap variants, because we find our models’ performance to be most dependent on the quality of the estimated uncertainty here. On most other commonly used UCI regression datasets (Hernández-Lobato and Adams, 2015) we find increased uncertainty to hurt LL performance. In other words, the predictions made when using the MAP setting of the weights are better than those from any Bayesian ensemble.

Wine and Protein are available from the UCI dataset repository Dua and Graff (2017). Kin8nm is available from https://www.openml.org/d/189 Foong et al. (2019b). For the standard splits (Hernández-Lobato and Adams, 2015) 90% of the data is used for training and 10% for validation. For the gap splits (Foong et al., 2019b) a split is obtained per input dimension by ordering points by their values across that dimension and removing the middle 33% of the points. These are used for validation.

The datasets used for our image experiments are outlined in Table D.23.

### Table D.22 Datasets from tabular regression used in Section 5.5.2

<table>
<thead>
<tr>
<th>Dataset</th>
<th>N Train</th>
<th>N Val (15% train)</th>
<th>N Test</th>
<th>Splits</th>
<th>Output Dim</th>
<th>Output Type</th>
<th>Input Dim</th>
<th>Input Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wine</td>
<td>1223</td>
<td>216</td>
<td>160</td>
<td>20</td>
<td>1</td>
<td>Continuous</td>
<td>11</td>
<td>Continuous</td>
</tr>
<tr>
<td>Wine Gap</td>
<td>906</td>
<td>161</td>
<td>532</td>
<td>11</td>
<td>1</td>
<td>Continuous</td>
<td>11</td>
<td>Continuous</td>
</tr>
<tr>
<td>Kin8nm</td>
<td>6267</td>
<td>1106</td>
<td>819</td>
<td>20</td>
<td>1</td>
<td>Continuous</td>
<td>8</td>
<td>Continuous</td>
</tr>
<tr>
<td>Kin8nm Gap</td>
<td>4642</td>
<td>820</td>
<td>2730</td>
<td>8</td>
<td>1</td>
<td>Continuous</td>
<td>8</td>
<td>Continuous</td>
</tr>
<tr>
<td>Protein</td>
<td>34983</td>
<td>6174</td>
<td>4573</td>
<td>5</td>
<td>1</td>
<td>Continuous</td>
<td>9</td>
<td>Continuous</td>
</tr>
<tr>
<td>Protein Gap</td>
<td>25913</td>
<td>4573</td>
<td>15244</td>
<td>9</td>
<td>1</td>
<td>Continuous</td>
<td>9</td>
<td>Continuous</td>
</tr>
</tbody>
</table>

### Table D.23 Summary of image datasets. The test and train set sizes are shown in brackets, e.g. (test & train).

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
<th>Input Dim.</th>
<th>No. Classes</th>
<th>No. Splits</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST (LeCun et al., 1998)</td>
<td>70,000 (60,000 &amp; 10,000)</td>
<td>784 (28 × 28)</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>Fashion-MNIST (Xiao et al., 2017)</td>
<td>70,000 (60,000 &amp; 10,000)</td>
<td>784 (28 × 28)</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>CIFAR10 (Krizhevsky and Hinton, 2009)</td>
<td>60,000 (50,000 &amp; 10,000)</td>
<td>3072 (32 × 32 × 3)</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>SVHN (Netzer et al., 2011)</td>
<td>99,289 (73,257 &amp; 26,032)</td>
<td>3072 (32 × 32 × 3)</td>
<td>10</td>
<td>2</td>
</tr>
</tbody>
</table>
Appendix E

Continual Deep Learning

E.1 Derivation of Error for K-priors With Limited Memory

Eq. (6.9)

Recall that $f^n_\theta = f_\theta(x_n)$ is a shorthand for the model outputs. For ease-of-notation, we will also use the shorthand $\sigma^n_\theta = \sigma(f^n_\theta) = \sigma(f_\theta(x_n))$ for the model predictions. Consider the following expression for the gradient of the (exponential-family) loss (Khan and Swaroop, 2021),

$$
\nabla \ell(y_n, \sigma(f^n_\theta)) = \nabla f^n_\theta [\sigma(f^n_\theta) - y_n].
$$

(E.1)
We then have
\[
\nabla \ell_t^{\text{batch}}(\theta) - \nabla \ell_t^{\text{K-prior}}(\theta)
\]
\[
= \nabla \left( \ell_t^{\text{batch}}(\theta) - \ell_t^{\text{K-prior}}(\theta) \right)
\]
\[
= \nabla \left( \ell_t^{\text{batch}}(\theta) - \ell_{t-1}^{\text{K-prior}}(\theta) \right)
\]
\[
Eqs. (6.1) and (6.6)
\]
\[
= \nabla \left( \sum_{n \in D_t} \ell(y_n, \sigma^n_\theta) + \ell_{t-1}^{\text{batch}}(\theta) - \sum_{n \in D_t} \ell(y_n, \sigma^n_\theta) - K(\theta; \theta_{t-1}, M) \right)
\]
\[
= \nabla \left( \ell_{t-1}^{\text{batch}}(\theta) - K(\theta; \theta_{t-1}, M) \right)
\]
\[
Eqs. (6.1) and (6.8)
\]
\[
= \sum_{n \in D_{t-1}} \nabla \ell(y_n, \sigma^n_\theta) - \sum_{n \in M} \nabla \ell(\sigma^n_{\theta_{t-1}}, \sigma^n_\theta) + \delta \theta_{t-1}
\]
\[
Eq. (E.1)
\]
\[
= \sum_{n \in D_{t-1}} \nabla f^n_\theta(\sigma^n_\theta - y_n) - \sum_{n \in M} \nabla f^n_\theta(\sigma^n_{\theta_{t-1}} - \sigma^n_\theta) + \delta \theta_{t-1}
\]
\[
= \sum_{n \in D_{t-1}} \nabla f^n_\theta(\sigma^n_{\theta_{t-1}} - y_n) + \sum_{n \in D_{t-1}} \nabla f^n_\theta(\sigma^n_{\theta_{t-1}} - \sigma^n_{\theta_{t-1}}) - \sum_{n \in M} \nabla f^n_\theta(\sigma^n_{\theta_{t-1}} - \sigma^n_{\theta_{t-1}}) + \delta \theta_{t-1}
\]
\[
= \sum_{n \in D_{t-1}, M} \nabla f^n_\theta(\sigma^n_{\theta_{t-1}} - \sigma^n_{\theta_{t-1}}) + \sum_{n \in D_{t-1}} \nabla f^n_\theta(\sigma^n_{\theta_{t-1}} - y_n) + \delta \theta_{t-1}
\]
\[
= \sum_{n \in D_{t-1}, M} \nabla f^n_\theta(\sigma^n_{\theta_{t-1}} - \sigma^n_{\theta_{t-1}}) + \sum_{n \in D_{t-1}} \nabla f^n_\theta(\sigma^n_{\theta_{t-1}} - \sigma^n_{\theta_{t-1}}) + \delta \theta_{t-1}
\]
where \(r^n_{\theta_{t-1}} = \sigma^n_{\theta_{t-1}} - y_n\) is the residual of the \(n\)th input using the past model parameters \(\theta_{t-1}\).

### E.2 Experiment Details

For all methods in all experiments, we tuned hyperparameters in the common way by conducting a (exponentially-spaced) grid search and evaluating performance on a held-out validation set. In particular, we tune the following hyperparameters: a temperature parameter \(T\) to scale the logits in the experience replay term (as commonly-done in knowledge distillation, see Khan and Swaroop (2021) for a discussion), a trade-off parameter \(\tau\) in front of the K-prior-style function regularization term, and a trade-off parameter \(\lambda\) in front of the EWC-style weight regularization term.
We used the following tuned values for those hyperparameters: for Split-CIFAR, we have $T = 2.0$ for all methods, $\tau = 0.1$ for $K_{\text{prior}}$ and $K_{\text{prior}+\text{Replay}}$, $\tau = 0.25$ and $\lambda = 2.0$ for $K_{\text{prior}+\text{EWC}}$ and $K_{\text{prior}+\text{EWC}+\text{Replay}}$, and $\lambda = 10.0$ for Online EWC. For Split-TinyImageNet, we have $T = 1.0$ and $\tau = 16.0$ for all methods. We found that a fixed $\lambda$ across tasks does not work well for this benchmark, so we tuned a separate $\lambda$ per task, resulting in the sequence $[330, 85, 45, 30, 20, 15, 10, 10]$ for all methods. In contrast to our grid search procedure, Delange et al. (2021) use a dedicated iterative hyperparameter tuning strategy that trades-off plasticity vs. stability. For ImageNet, we used $T = 1.0$, $\lambda = 1.0$ and $\tau = 0.16$ for all methods.