A TUTORIAL ON VARIATIONAL METHODS FOR MACHINE LEARNING

Lucas Pinheiro Cinelli

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Elétrica.

Orientadores: Eduardo Antônio Barros da Silva
Sergio Lima Netto

Rio de Janeiro
Maio de 2019

A TUTORIAL ON VARIATIONAL METHODS FOR MACHINE LEARNING

Lucas Pinheiro Cinelli

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Examinada por:

Prof. Eduardo Antônio Barros da Silva, Ph.D.

Prof. Sergio Lima Netto, Ph.D.

Eng. Leonardo de Oliveira Nunes, D.Sc.

Prof. Luis Alfredo Vidal da Carvalho, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
MAIO DE 2019

Cinelli, Lucas Pinheiro

A Tutorial on Variational Methods for Machine Learning/Lucas Pinheiro Cinelli. – Rio de Janeiro: UFRJ/COPPE, 2019.

XV, 120 p.: il.; 29, 7cm.

Orientadores: Eduardo Antônio Barros da Silva
Sergio Lima Netto

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2019.

Referências Bibliográficas: p. 97 – 111.

1. Generative Models. 2. Variational Autoencoder. 3. Bayesian Neural Networks. 4. Uncertainty. 5. Approximate Inference. 6. Variational Inference. 7. Expectation Propagation. I. Silva, Eduardo Antônio Barros da *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

A minha familia, amigos e namorada

Acknowledgments

I would link to thank everyone that took part in my devolopment during these years.

Specially, my family: my parents who raised and taught me, and even nowadays devote their lives to my siblings and me; my brother and sister who I always admired; my nephew, nieces, uncles, aunts, and cousins, because family is something we never forget.

My friends: those who were with my in high-school and think I am still doing my final undergraduate project; those I made during my undergraduate years and have not given up trying to reach and meet me though I never show up; and those who put up with me on a daily basis and help me in all sort of problems, either it be programming, theoretical, or mundane stuff.

The professors and employees from the Signal, Mutimedia and Telecommunications laboratory, who maintain the ecosystem we live in and give us room to develop ourselves. I could not go without praising the value and patience of both my advisers, with whom I not always agree, but I know try their best to pull me back to reality. I admit it is hard work.

The examining committee, which accepted reading this dissertation and helping me improve it in spite of its subject being a bit orthogonal to what their are used to.

My girlfriend, that helped me immeasurably even without having a single clue about what lies in these pages.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

TUTORIAL EM MÉTODOS VARIACIONAIS PARA APRENDIZADO DE MÁQUINA

Lucas Pinheiro Cinelli

Maio/2019

Orientadores: Eduardo Antônio Barros da Silva
Sergio Lima Netto

Programa: Engenharia Elétrica

Apresenta-se, nesta dissertação, um tutorial sobre modelos generativos profundos, especificamente autocodificadores variacionais (*Variational Autoeconders* – VAE) e redes neurais Bayesianas profundas. Apesar de disparatos à primeira vista, ambos assuntos são próximamente interligados pela visão probabilística do aprendizado de máquina.

Dessa forma, descrevem-se métodos de inferência aproximada visto que técnicas exatas são adequedas tão somente em condições muito restritas em que o tempo de cálculo é viável. Extensões modernas de tais métodos, capazes de escalar adequadamente ao tamanho dos modelos, que contêm milhões de parâmetros, e às atuais bases de dados com milhões de exemplos, são expostas e suas aplicações em redes neurais profundas demonstradas ao longo dos capítulos junto de implementações práticas que exemplificam o uso de ditos modelos.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

A TUTORIAL ON VARIATIONAL METHODS FOR MACHINE LEARNING

Lucas Pinheiro Cinelli

May/2019

Advisors: Eduardo Antônio Barros da Silva
Sergio Lima Netto

Department: Electrical Engineering

In this work, we present a tutorial on deep generative models, specifically Variational Autoencoders (VAE) and deep Bayesian Neural Networks (BNN). In spite of being apparently distinct, both themes are intimately connected through the probabilistic view of machine leaning.

Therefore, we describe approximate inference methods since exact approaches are only adequate in very limited conditions in which the computation time remains feasible. We discuss modern extensions to such methods, capable of adequately scaling to models with millions of parameters and equally large data sets. Furthermore, we illustrate their relevance in each chapter through applications using (deep) neural networks.

Contents

List of Figures

List of Tables

List of Abbreviations

ADF	Assumed Density Filtering, p. 22
ADVI	Automatic Differentiation Variational Inference, p. 31
AEVB	Autoencoding Variational Bayes, p. 80
BBB	Bayes by Backprop, p. 35
BBVI	Black Box Variational Inference, p. 29
BO	Bayesian Optimisation, p. 67
CAVI	Coordinate Ascent VI, p. 20
CDF	Cumulative Distribution Function, p. 5
CVAE	Conditional Variational Autoencoder, p. 82
CV	Cross-Validation, p. 68
DNN	Deep Neural Network, p. 2
ELBO	Evidence Lower Bound, p. 18
EM	Expectation Maximisation, p. 21
EP	Expectation Propagation, p. 17
GGN	Generalised Gauss-Newton, p. 37
GPU	Graphical Processing Unit, p. 15
IS	Importance Sampling, p. 77
KL	Kullback-Leibler, p. 18
MAP	Maximum a Posteriori, p. 10
MBML	Model-Based Machine Learning, p. 7

MCMC	Markov Chain Monte Carlo, p. 1
MFVI	Mean-Field Variational Inference, p. 20
ML	Machine Learning, p. 1, 7
NN	Neural Network, p. 3
PBP	Probabilistic Backpropagation, p. 41
PDF	Probability Density Function, p. 4
PGM	Probabilistic Graphical Model, p. 14
RMSE	Root Mean Squared Error, p. 66
ReLU	Rectified Linear Unit, p. 42
SVI	Stochastic VI, p. 28
UMAP	Uniform Manifold Approximation and Projection, p. 84
VAE	Variational Autoencoder, p. 73
VI	Variational Inference, p. 17

Chapter 1

Introduction

Over the last two decades, Bayesian methods have largely fallen out of favour in the Machine Learning (ML) community. The culprit for such unpopularity are their complicated theory, which makes it hard for practitioners to access and comprehend them, and, specially, their heavy computational burden. Conversely, classical techniques relying on bagging, boosting and point estimates offer a cheap alternative to measure uncertainty. Consequently, Bayesian methods remained confined mostly to (Bayesian) statisticians and a handful of other researchers either working in related areas or disposing of small amounts of data.

For instance, Markov Chain Monte Carlo (MCMC) methods are powerful Bayesian tools. In a modelling problem they are able to converge to the true distribution of the model if given enough time. However, this frequently means more time than one is willing to wait, and though many modern algorithms alleviate this issue [1], the state of affairs remains roughly the same. MCMC is asymptotically exact and computationally expensive. This effect worsens with the dimensionality of the problem. Conventional Bayesian methods do not scale well to large amounts of data nor to high dimensions, situations which are becoming increasingly common in the Age of Big Data [2].

One may think that the abundant amount of data should make up for the lack of uncertainty and its estimation because in the limit of infinite samples the Bayesian estimation converges to the maximum likelihood point. The pinnacle of the disconnection from the probabilistic view is standard Deep Learning. It basically consists on very large parametric models trained on, ideally but not often, large amounts of data to fit an unknown function. Modern hardware and computational libraries render the computation possible through parallel computing. Thanks to this new representation learning technique, we have been achieving outstanding results in the last 8 years, breaking plateaus in many areas of research, e.g., speech [3] and vision [4]. As a consequence, the deep learning domain became a trending area, attracting a lot of newcomers, media attention and industry investments.

All this positive feedback reinforces the behaviour of overlooking probabilistic modelling and reasoning. After all, it seems to be working. However, reliable confidence estimates are essential to many domains such as healthcare and financial markets, and standard deep learning cannot attend their demand. Recently, researchers found that many ML models, including Deep Neural Networks (DNN) with great test set performance are deceived by adversarial examples [5], which are images apparently normal to humans but that are consistently misclassified with great confidence. Moreover, the authors describe a method to create adversarial examples [5]. On the other hand, methods that estimate uncertainty are capable of detecting adversarial examples and, more generally, examples outside the domain in which they were trained.

The current approach to ML requires large quantities of data and when not available, the models are likely to overfit and have poor generalisation. Contrarily, Bayesian methods perform well in data-poor regimes and are robust, though not immune, to overfitting. As we discuss in Section 2.1.1, there is an important fundamental difference between a large and a *statistically* large dataset. A mere 28×28 binary image has 784 dimensions and $2^{784} \approx 10^{236}$ different arrangements, which is far more than the estimated number of atoms in the observable universe ($\sim 10^{80}$) [6]. Even in a simple case as this, *statistically* large means having a virtually infinite number of examples. Naturally, we frequently assume there is an underlying low-dimensional structure that explains the observations. In Section 2.1.2, we formalise this thought and in Section 4.3 we review an algorithm that incorporates this assumption.

Probabilistic models further lend themselves to semi-supervised and unsupervised learning, allowing us to leverage the performance from unlabelled samples, and do active learning, in which the samples the system is most uncertain about are put forward for the operator to label, maximising the information gain. In general, the Bayesian framework offers a principled approach to constructing probabilistic models, reasoning under uncertainty, making predictions, detecting surprising events and simulating new data. It naturally provides mathematical tools for model fitting, comparison and prediction, but more than that, it constitutes a systematic way of approaching a problem.

Since Bayesian methods can be prohibitively expensive, we focus on approximate algorithms that on a sensible amount of time achieve reasonable performance. Technically, MCMC is one such class of algorithm, but it is based on sampling and has slow convergence rate. Here, we discuss variational methods, which instead rely on deterministic approximations. They are much faster than sampling approaches, which makes them well suited to large data sets and to quickly explore many models [7]. The toll for its speed is inferior performance, making it adequate to scenarios

where we dispose of a lot of data to compensate for that weakness and where it would be otherwise impossible to employ MCMC.

Over the last 8 years, research on variational methods for Bayesian ML started to reemerge [8] and slowly gain momentum. Since 2014, there has been an exponential growth in interest for this field [9–11], fuelled among others by the discovery of critical failure modes for conventional Deep Learning. Nowadays, the field has workshop tracks in major ML conferences and lots of papers accepted to the main tracks, venues geared towards statistics, Artificial Intelligence and uncertainty increasing in importance, visibility and number of submissions.

1.1 Objectives and Contributions

While Bayesian ML and approximate inference are rather large topics spawning entire books, the intent of this dissertation is to be a self-contained introduction to modern variational methods for Bayesian Neural Networks (NN). Even within this realm, research is (fortunately) sprouting at a rate difficult to follow and many algorithms are also being reinterpreted through Bayesian lenses. We focus on practical BNN algorithms that are either (relatively) easy to understand or fast to train as well as on one specific usage of a variational technique for generative modelling.

The target audience are those already familiar with ML and modern NN. Although basic knowledge of calculus, linear algebra and probability theory are a must to comprehend the concepts and derivations herein, they should also be enough. We explicitly avoid matrix calculus since the material may be challenging by itself and adding this difficulty does not really aid in the understanding and may actually intimidate. Furthermore, we do not assume the reader to be familiar with statistical inference and thus explain the necessary information throughout the text when needed.

Most introductory texts cover either modern NNs or general Bayesian methods for ML, with little work dedicated to both simultaneously to this date. Information is scattered around in research blog posts and introductions of published papers, with the sole in-depth work being Neal's excellent Ph.D. thesis [12] from 1996, which does not cover modern variational approximations. The current scenario makes the leap from NNs to BNNs hard from a theoretical point of view: the reader needs either to learn Bayesian methods first or to decide what matters and which algorithms to learn; the former being cumbersome and the latter troublesome in a self-study scenario.

The present dissertation has the mission of filling this gap and helping others cross from one to the other with not only a working knowledge, but also an understanding of the theoretical underpinnings of the Bayesian approach. Given the

importance of source codes in learning a computationally-oriented subject, we make ours available[1].

1.2 Organization of this Dissertation

Prior to applying variational approximations to NNs, we need to introduce and explain what actually is approximate inference and what are some of the core variational techniques. Chapter 2 accomplishes this role and presents the reader with the main concepts of the probabilistic view and model-based approach to ML. Chapter 2 is paramount to the rest of the dissertation and should be read with care, otherwise the other chapters become just a bundle of senseless statements and formulas.

Disposing of the necessary tools learnt in Chapter 2, the reader enters Chapter 3 well-equipped to follow the detailed derivation of the BNN algorithms therein. It consists of 4 different methods to frame and attack the problem [13–16], each with its own strength whether it be performance-wise or mere conceptual simplicity. The chapter is long and requires patience to get through its many equations. A sensible advice is to keep around pen and paper to help with the formulas.

Next, we focus on a different aspect and use approximate inference to establish a (deep) generative model [10]. Chapter 4 builds it from the ground up starting with a simple modelling problem, presents some extensions and points out important flaws of the method.

Finally, Chapter 5 starts by summarizing the core content of each chapter, which gives the reader a chance to review the learned material. Afterwards, we point an exciting and challenging future research axis that has so far been little explored and set ambitious interdisciplinary long-term goals.

1.3 On the notation

The following mathematical elements attend the notation:

- scalar: a and σ
- vector: $\mathbf{a}$ and $\boldsymbol{\sigma}$
- matrix: $\mathbf{A}$ and $\boldsymbol{\Sigma}$
- set (of scalars, vector or matrices): $\mathcal{A}$ and Σ

We denote Probability Density Functions (PDF) and probability distributions with lower-case notation p and use them interchangeably. Although technically

[1] `https://github.com/lpcinelli/probabilistic-nn`

incorrect and being an abuse of language, we decide to simplify notation and make it clear from the context whether the random variable is continuous or discrete. Nevertheless, we already advert to the almost non-existence of discrete random variables throughout the text, especially on Chapters 3 and 4, whose algorithms rely on continuous functions and variables, the sole discrete distribution in there being the Bernoulli distribution. Addtionally, we always denote the Cumulative Distribution Function (CDF) in upper-case, such as $F(X) = P(X \leqslant x)$.

We write parametric family of distributions p as $p(\cdot\,;\,\Xi)$ with Ξ the set of parameters that specify the member of the family. In the case of a Gaussian random variable z, the pdf would be $p(z\,;\,\mu,\sigma^2) = \mathcal{N}(z\,;\,\mu,\sigma^2)$, where the parameters are the mean μ and variance σ^2. If the parameters are random variables, we can write the conditional distribution as $p(\cdot\,|\,\Xi)$, and since we deal with Bayesian analysis these two notations get pretty similar although different.

Whenever possible, the set of variational parameters will write Ψ and the set of model parameters Θ, and if both refer to the same entity we opt for Θ. Similarly, hidden units or more generally latent variables are $\mathcal{Z}$.

Also, derivatives w.r.t. to a set is a shorthand for compactly representing the derivative w.r.t. each element of the set. For example let f be a function parametrised by $\Theta = \{\theta_1, \theta_2\}$, we have according to this notation

$$\frac{\partial f(\Theta)}{\partial \Theta} = \begin{cases} \frac{\partial f(\theta_1,\theta_2)}{\partial \theta_1} \\ \frac{\partial f(\theta_1,\theta_2)}{\partial \theta_2} \end{cases}.$$

Although this example coincides with the gradient when Θ is a vector and its elements scalars, it is not true when Θ is a set of matrices for example. One could argue that sets of vectors might be arranged into matrices and those into multidimensional tensors, thus the above notation being unnecessary. However, our aim is to simplify the notation and exempt the reader from additional technical difficulties.

Chapter 2

Probabilistic Thinking

This chapter introduces what the reader needs to go through the rest of the dissertation with flying colours.

We discuss what is the model-based approach to machine learning and its main enabling techniques: (approximate) Bayesian inference, graphical models, and, more recently, probabilistic programming. We focus on distributional approximation methods for approximate inference, and more specifically on Variational Bayes, which is the algorithm most of the models on later sections rely on.

By the end of this Section, the reader should:

- Understand the various advantages of the model-based approach.
- Discern the benefits and issues of Bayesian inference.
- Be capable of deriving Variational Bayes and Expectation Propagation.
- Understand the mean-field approximation.
- Comprehend the relations between the variational methods.
- Know the modern landscape of stochastic and black-box inference methods.

2.1 Model-Based Machine Learning

Initially, we pose ourselves a question about the very first word in this section's title – *model* – and then what it means for a Machine Learning (ML) method to be model-based.

A model can assume different forms and complexities. Physicists have different models for understanding the universe: astronomers focus on General Relativity and the interaction between celestial bodies, while particle physicists represent it according to quantum mechanics; infants draw stick figures of their families, houses and alike; neuroscientists study the drosophila ("small fruit flies") as a model for understanding the brain; drivers imagine what will change and how in order to decide what to do next.

Although all these examples seem distinct and may serve diverse purposes, they all are approximate representations of their equivalent real-world object/process. Thus, a model is a description of the world (at a given level) and as such encodes our beliefs and assumptions about it.

All models have parameters, which may be unknown *a priori* and must be learned from the available data so that we are able to discover its latent causes or predict possible outcomes. If our model does not match the observed data, we are capable of refuting the proposition and search for one that can explain it.

As one might imagine, Model-Based Machine Learning (MBML) aims at providing a *specific* solution for each application. It explicitly encodes the set of assumptions for a given application directly in the model, which we describe following mathematical tools, the language of science. Consequently, we are able to create a wide range of highly tailored models under a single development framework.

This clear distinction of what is the model, decouples the model per se from the learning (inference) algorithm. This segregation gives transparency to its functionality since the model structure becomes apparent [17] and allows the application of the same inference method to different models and vice versa, generating a large number of possible combinations. The unified framework facilitates rapid prototyping and comparison, and allows the derivation of many traditional machine learning techniques as special cases of certain model-inference configurations (examples in Section 2.1.3).

Statistical inference refers to the general procedure by which we deduce any desired probability distribution (possibly marginal or conditional) of our model or parts of it given the observed data. The ML literature usually disassociates the terms learning and inference, with the former referring to model parameter estimation and the latter to reasoning about unknowns, i.e., the model output, given the already estimated parameters. However, in statistics there is no such difference and both

mean estimation. In the present text, they are used interchangeably though we tend to say inference more often due to this term being readily associated with probability distributions.

One might question why do we want to infer probability distributions or even what are the advantages over something simpler such as point estimates, after all they are single values and already give us answers. The problem in considering only the most likely solution comes from losing information of the underlying variability of the model. Let us consider a trivial example to illustrate the issue:

Example 2.1.1. *An ambulance must take a dying person to the nearest hospital and there are two possible routes $\mathcal{A}$ and $\mathcal{B}$. $\mathcal{A}$ takes **about** 15 minutes, while $\mathcal{B}$, 17. Which one should the driver choose? Now, the driver further considers that $\mathcal{A}$ consists of regular urban streets with semaphores and possible traffic jams and that his predicted travel time **may vary** up to 8 minutes, whereas $\mathcal{B}$ is a express lane for medical emergencies and the estimated time **varies** by no more than 1 minute. Would the choice be the same?*

The highlighted keywords above give us a sense of the intrinsic variability in our problem and disregarding the knowledge they provide may be misleading. In the above example, it is clear that the average time is not sufficient information and that the uncertainty is critical for making a conscious decision. Probability theory provides a principled framework for modelling uncertainty. As seen in our example, probabilistic models allow us to reason and perform decision-making, anticipate the future and plan accordingly, detect unexpected events, among others; all by learning probability distributions of the data. Not only can we understand almost all ML through probabilistic lenses, but also connect it through this perspective to every other computational science [18, p. 2].

2.1.1 Bayesian Inference (Learning)

As promised, we start by going over the basic precepts of the Bayesian paradigm and parameter estimation. Bayesian probability inherits its name from the Bayes rule:

$$p(\mathcal{Z} \mid \mathcal{X}) = \frac{p(\mathcal{X} \mid \mathcal{Z})p(\mathcal{Z})}{p(\mathcal{X})} \ . \tag{2.1}$$

Although such equation is valid for whatever events $\mathcal{Z}$ and $\mathcal{X}$ may represent, we consider a common case of ours where $\mathcal{Z}$ is the set of unknown random variables of our model and $\mathcal{X}$ the observed data corresponding to the real-world process.

In the above expression each term has a clear interpretation:

- $p(\mathcal{Z})$ is the *prior* – it encodes into the model any prior belief or domain knowledge we might possess. In case one does not know anything about the problem at hand then one might use non-informative priors;

- $p(\mathcal{X} \mid \mathcal{Z})$ is the *likelihood* – it is the density function of the probability with respect to the parameters and not to the possible events. Intuitively, it measures how likely is the observed data given the model.

- $p(\mathcal{Z} \mid \mathcal{X})$ is the *posterior* – its name reflects the fact that our knowledge about the parameters has been updated after accounting for our (new) data, thus it is the probability of $\mathcal{Z}$ conditioned on such evidence;

- $p(\mathcal{X})$ is the *evidence* – as hinted above, the term refers to the observed data and this probability (density) works as a normalising factor equal to $\int_{\mathcal{Z}\in\Omega} p(\mathcal{X} \mid \mathcal{Z})p(\mathcal{Z})$ so that the posterior corresponds to a proper probability distribution.

The above formula carries the main concepts of the Bayesian methodology. Besides the obvious use of the formula itself to update the model, there is the prior probability distribution and the treatment of parameters not as fixed but as random variables instead. Thus all unknown quantities are treated equally and indeed there is no distinction among them. The Bayesian view is an interpretation of probability itself and its meaning. While frequentists[1] see it as the relative frequency of an event, Bayesians see it as quantification of a belief.

We can understand the role of the likelihood term $p(\mathcal{D} \mid \mathcal{W})$ more practically by means of an example. Let $f(\cdot; w)$ be a regression model parameterised by w that predicts a scalar value $\hat{y}$, such that $\hat{y} = f(x)$. Our probabilistic model assumes a given level of noise and we thus place an observation noise model on top of the output, such that the observed output is corrupted by a known process $g(\cdot)$, say an additive Gaussian noise with variance σ^2. The log-likelihood then has the form

$$\log p(y \mid x, w) = \log \mathcal{N}(y; f(x; w), \sigma^2) \tag{2.2}$$

$$= -\frac{1}{2}\log\left(2\pi\sigma^2\right) - \frac{1}{2\sigma^2}\left(y - f(x; w)\right)^2. \tag{2.3}$$

What we wish is that the observation y is as close as possible to the predicted output $f(x)$, such that our model agrees with the data. Note that the prediction and the noise model could in principle be anything.

We now present the trinity of estimation methods that we extract from Equation (2.1) (but that do not necessarily follow the Bayesian framework) along with

[1] Frequentism is the classical approach to probability, for which the probability of an event is only meaningful in the limiting case of infinite measurements.

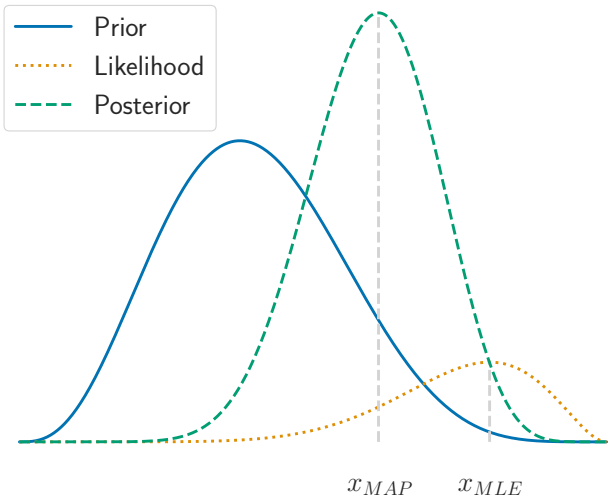


Figure 2.1: Prior, likelihood and posterior of a model. The likelihood does not sum to 1. The maximisers for the likelihood and the psoterior are different. As more data is gathered, the likelihood becomes expressive and the posterior shifts towards it.

Figure 2.1.

Maximum Likelihood Estimation

As guessed by the name, Maximum Likelihood estimation relies on the maximization of the likelihood function:

$$\begin{aligned} \mathcal{Z}_{MLE} &= \operatorname*{argmax}_{\mathcal{Z}} p(\mathcal{X} \mid \mathcal{Z}) \\ &= \operatorname*{argmax}_{\mathcal{Z}} \prod_i p(x_i \mid \mathcal{Z}) \\ &= \operatorname*{argmax}_{\mathcal{Z}} \sum_i \log p(x_i \mid \mathcal{Z}) \ . \end{aligned} \tag{2.4}$$

We assume the observed data is independent and identically distributed (iid) given $\mathcal{Z}$ so $p(\mathcal{X} \mid \mathcal{Z})$ factorises. The last equality holds as the log is a monotonically increasing function and therefore the optimisation problem is equivalent.

This method obtains a point estimate for the parameters (corresponding to the maximum) and boasts parameter transformation invariance and ease of calculation. On the other hand, it loses the variability information we previously discussed and is prone to overfitting, though asymptotically optimal.

Maximum a Posteriori Estimation

Again, by simple deduction one may guess that Maximum a Posteriori (MAP) estimation performs the maximisation of the a posteriori function $p(\mathcal{Z} \mid \mathcal{X})$ defined in

Equation (2.1). From an optimisation perspective with respect to the parameters, the evidence is fixed so we can ignore it:

$$\begin{aligned}
\mathcal{Z}_{MAP} &= \operatorname*{argmax}_{\mathcal{Z}} p(\mathcal{Z} \mid \mathcal{X}) \\
&= \operatorname*{argmax}_{\mathcal{Z}} \frac{p(\mathcal{X} \mid \mathcal{Z}) p(\mathcal{Z})}{p(\mathcal{X})} \\
&= \operatorname*{argmax}_{\mathcal{Z}} \left(p(\mathcal{X} \mid \mathcal{Z}) p(\mathcal{Z}) \right) \\
&= \operatorname*{argmax}_{\mathcal{Z}} \left(\left[\prod_i p(x_i \mid \mathcal{Z}) \right] p(\mathcal{Z}) \right) \\
&= \operatorname*{argmax}_{\mathcal{Z}} \log \left(\left[\prod_i p(x_i \mid \mathcal{Z}) \right] p(\mathcal{Z}) \right) \\
&= \operatorname*{argmax}_{\mathcal{Z}} \left(\sum_i \log p(x_i \mid \mathcal{Z}) + \log p(\mathcal{Z}) \right) \ . \qquad (2.5)
\end{aligned}$$

The MAP forcefully regards $\mathcal{Z}$ as a random variable since it also takes into consideration the prior, which is a proper probability density of $\mathcal{Z}$. The above formula is a common utility function in ML algorithms, e.g., in neural networks the second term is known as the regulariser and if we take the prior as a standard Gaussian distribution the term reduces to ℓ_2 regularisation.

Even though this approach has a more Bayesian feeling to it, the MAP estimation still is a point estimate and we may wish instead the whole probability distribution. Furthermore, its results are not invariant to parameter transformations, which is undesired because we would like to always arrive at the same point, the equivalent solution.

Bayesian Estimation

A Bayesian treatment consists of computing the full posterior distribution $p(\mathcal{Z} \mid \mathcal{X})$ instead of only point estimates, therefore the importance of the normalising constant, the evidence $\int_{\mathcal{Z}} p(x \mid \mathcal{Z}) p(\mathcal{Z})$. Those are the two important quantities we reason about. They allow us to produce point or interval estimates of the latent variables and construct predictive densities for new data. For example, at test time we compute the posterior predictive distribution over the new datum x':

$$p(x' \mid \mathcal{X}) = \int_{\mathcal{Z}} p(x' \mid \mathcal{Z}) p(\mathcal{Z} \mid \mathcal{X}) \ . \qquad (2.6)$$

Intuitively, we compute the probability of x' for each setting of the random variable $\mathcal{Z}$ taking into account the probability of $\mathcal{Z}$ itself as given by the learned posterior $p(\mathcal{Z} \mid \mathcal{X})$.

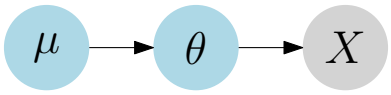


Figure 2.2: Hierarchical Bayes model of 3 stages. X is an observed random variable, θ is an unknown parameter governing its generation process, and μ a hyperparameter that determines the distribution of the random variable θ.

Bayesian models may be further decomposed into a sequence of conditional distributions spanning multiple levels following an hierarchical structure. Bayesian hierarchical models then have multiple levels of hyper-parameters which set the prior distribution of downstream stages and hyperpriors that define the corresponding distributions. In the example of Figure 2.2, μ is a hyperparamter since θ is already a parameter that influences the distribution of the observations X. If we define a distribution for μ, we are defining a hyperprior, and if we want to take this distribution into consideration in the inference process, by marginalising μ, we are performing a fully Bayesian approach. On the other hand, if we define a point value for μ by maximising its likelihood determined by the observations, we are doing empirical Bayes.

As one might already imagine, in Bayesian analysis integration is the central operation. However, this frequently leads to intractable solutions, either due to the high dimensionality that renders computation unfeasible in a viable time, or to the nonexistence of a closed-form analytical solution. In Section 2.2 we go through some approximate methods that deal with this issue and in Chapter 3 we discuss algorithms for performing Bayesian regression in deep neural networks (a class of parametric models).

We previously mentioned that the Maximum Likelihood estimator is asymptotically optimal. Also note that under a statistically large data set the probability distributions in a Bayesian model may become narrow and similar to the results of point-based methods. One may then question what is the advantage of being Bayesian if the results end up being similar. The catch here are the words "asymptotically" and "statistically large". Bayesian modelling really shines when data is limited and traditional methods are prone to overfitting, the uncertainty in the parameters is significant in these cases [17]. The recent revolution in ML relies on very large data sets. However, these data sets still are statistically small, e.g., although the ImageNet data set [19] has more than 14 million images, there are virtually infinite possible configurations for all the object classes appearances in an image. Therefore, the scientific community has been actively researching Bayesian inference methods that scale well to large data sets.

2.1.2 Latent Variable Models

Given observed data $\mathcal{X}$, how should we model the distribution $p(\mathcal{X})$ so that it reflects the true real-world population? This distribution may be arbitrarily complex and to readily assume the data points $\mathcal{X}_i$ to be iid seems rather naive. After all, they cannot be completely independent, there must be an underlying reason for them to exist the way they do, even if unknown or *latent*. We represent this hidden cause by the variable $\mathcal{Z}$, thus obtaining the joint distribution $p(\mathcal{X}, \mathcal{Z})$. Naturally, by marginalising over $\mathcal{Z}$ we obtain

$$p(\mathcal{X}) = \int p(\mathcal{X}, \mathcal{Z}) d\mathcal{Z} = \int p(\mathcal{X} \mid \mathcal{Z}) p(\mathcal{Z}) d\mathcal{Z}. \tag{2.7}$$

Instead of assuming independence, we resort to the infinite exchangeability property. A finite sequence of random variables $(\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_n)$ is said to be exchangeable if any permutation of its elements have the same probability distribution [20]. Consequently, the order of the sequence is not relevant to determine the joint distribution nor any marginal. In particular, all marginal distributions are equal. An infinite sequence of random variables is infinitely exchangeable if every finite subsequence is exchangeable. Note that this is more general than the iid property.

De Finetti's representation theorem [20] states that if the sequence of random variables is infinitely exchangeable, then there exists $p(\mathcal{Z})$ for which the joint distribution can be written as

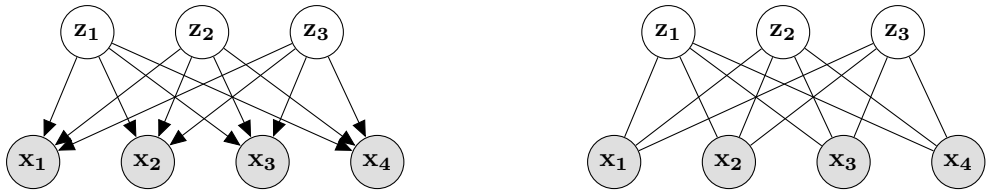
$$p(\mathcal{X}_1, \mathcal{X}_2, \ldots) = \int \prod_i p(\mathcal{X}_i \mid \mathcal{Z}) p(\mathcal{Z}) d\mathcal{Z}. \tag{2.8}$$

We can thus see the joint distribution of an infinitely exchangeable sequence of random variables as representing a process in which a random parameter is drawn from some (prior) distribution and then all observables in question are iid conditioned on that parameter.

The representation theorem shows how statistical models emerge in a Bayesian context: under the hypothesis of exchangeability of $\{\mathcal{X}\}_{i=1}^{\infty}$, a much weaker assumption than iid, **there exists** a parameter such that, given its value, the observables are conditionally iid. The theorem is a powerful motivation for Bayesian parametric models even though it does not say anything about $p(\mathcal{Z})$.

In practical problems, even if we deal with unordered data, the size is always finite. Therefore, the infinite exchangeability assumption may be impractical or wrong, but the result still holds approximately true for large n [21].

Note that we use latent variables and unknown model parameters interchangeably. For Bayesians, there is no fundamental difference between model parameters and latent variables, they are all random variables whose values we wish to infer. For



(a) Bayesian network (b) Markov random field

Figure 2.3: Examples of probabilistic graphical models.

example, if our observables $\mathcal{X}_i$ are Bernoulli random variables, then $\mathcal{Z}$ corresponds to the probability of success $p \in (0, 1)$.

2.1.3 Probabilistic Graphical Models

Full joint distributions are generally intractable so we resort to structured models [17], which associate probability distributions over a few variables to form larger, more complex models. This approach provides considerable computational simplifications. One very flexible paradigm and also dominant one over the last two decades is Probabilistic Graphical Models (PGM) [22].

PGMs use a diagrammatic representation for compactly encoding a complex distribution over a high-dimensional space [22]. Their structure captures our assumptions about the family of distributions our model may assume. Random variables become nodes, which can be shaded if observed or empty otherwise, and edges denote their relations (Figure 2.3). Plates symbolize that the subgraph they enclose repeats the number of times designed by their subscript (Figure 2.4).

For directed acyclic graphs (Figure 2.3a), each vertex $i \in \mathcal{V}$ together with its parent set $\mathit{\Pi}_i$ define a local probability distribution $p_i(x_i \mid \mathit{\Pi}_i)$, whose collection describes the joint probability of the model

$$p(x_1, x_2, \ldots, x_{|\mathcal{V}|}) = \prod_{i \in \mathcal{V}} p_i(x_i \mid \mathit{\Pi}_i) \ . \tag{2.9}$$

For undirected graphs (Figure 2.3b), since there is no ordering, we cannot decompose them into conditional probability distributions. Alternatively, the joint distribution factorizes according to fully connected subsets of vertices, to which we attribute potential functions [22].

As mentioned earlier, many traditional ML algorithms can be derived as special cases of the graphical model framework combined with the appropriate inference algorithms. For example, principal component analysis, factor analysis, logistic regression, Gaussian mixtures, and similar models can all be represented by simple graphical structures [17]. Moreover, they can be effortlessly combined. All this happens organically within the model-based ML framework. Furthermore, PGMs

(a) Principal component analysis (b) Generalized linear regression

Figure 2.4: Probabilistic graphical models of traditional machine learning algorithms

can be easily customised to a specific application or modified if the requirements of the application change. The computation of a single factor in a graphical model amounts to the computation of the corresponding node(s) in the graph. Thus, an algorithm can calculate the factor as the product between the local conditional distribution of that node and the incoming messages of adjacent vertices, that is, on whom they depends. Hence, computing the full joint probability reduces to performing local operations involving messages between neighbouring nodes. Such class of algorithms are called message-passing algorithms [23].

2.1.4 Probabilistic Programming

Probabilistic programming is a tool for statistical modelling. It borrows lessons from computer science and common programming languages so as to construct languages that allow the denotation and evaluation of inference problems [24]. Thus, it frees the developer from complex low-level details of probabilistic inference, allowing him/her to concentrate on issues more specific to the problem at hand, such as the model and the choice of inference method. Similarly to high-level programming languages which abstract away architecture-specific implementation details, it boosts performance and productivity.

One of the cornerstones for the deep learning success was – besides the availability of computing power put forth by modern Graphical Processing Units (GPU) and the obvious abundance of data in modern days – the development of specialised libraries that automate differentiation and relieve the user from the need for manually deriving the gradients for optimisation, as well as facilitate model specification. This genre of software led to the widespread use of deep learning. Nowadays, there is no need to actually understand the basics of neural networks or even calculus of derivatives to try and run a model, which does not mean any constructed model will be useful or meaningful. Probabilistic programming aims to achieve the same for probabilistic ML [24]. Consequently, it allows rapid prototyping and testing of ideas, allowing the field to flourish and pushing industry adoption.

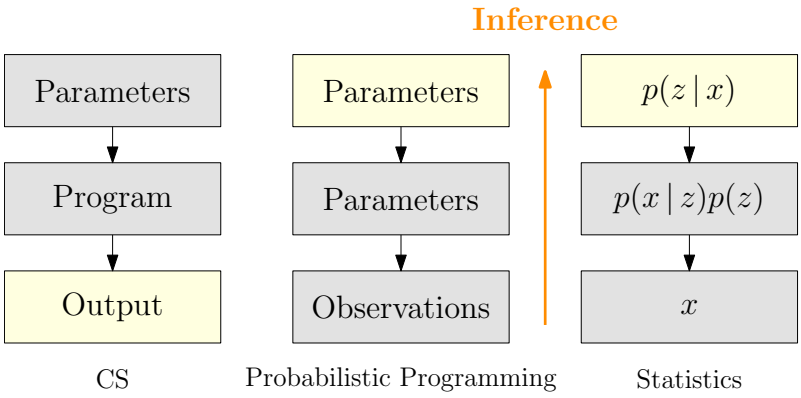


Figure 2.5: An intuitive view of probabilistic programming and how it differs from the common computer science paradigm. Shaded boxes indicate the information is available. Instead of inputing the required parameters to run the program and obtain the desired output, probabilistic programming tries to recover from the observations generated by the program which were the parameters. This process is similar to inference in statistics.

Modern probabilistic programming languages provide a more powerful framework than PGM does. Computer programs accept recursion and control flow statements which are otherwise difficult to represent [25]. There is a myriad of different languages, each with its own set os specific features: some are explicitly restrictive, others specialise in a certain types of inference techniques, or yet are general-purpose. A non-extensive list includes BUGS (from 1995) [26], WebPPL [27], Infer.NET [28], Stan [29], PyMC3 [30], Pyro [31], and Edward [32].

2.2 Approximate Inference

As briefly eluded in the previous section, for many models of practical interest it is frequently unfeasible to compute the posterior distribution or expectations over it. In the continuous case, there may not be a closed-form analytical solution and the integrand may be too complex for numerical integration. The same happens in the discrete case where summing over all possible configurations, though possible in principle, may be nonviable since the total number of combinations grows exponentially with dimension.

In such cases we have two options: either to successively simplify the model until exact inference is possible or to perform approximate inference in the original model. On this matter John Tukey stated [33, p. 13]: "Far better an approximate answer to the right question, which is often vague, than an exact answer to the wrong question, which can always be made precise".

There are two broad classes of approximation schemes: deterministic and stochastic. The latter relies on Monte-Carlo sampling to approximate expectations

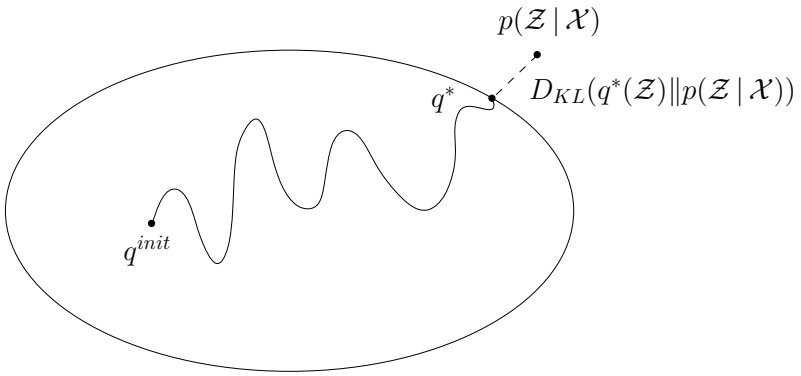


Figure 2.6: Illustration of optimisation process of VI given a family of distributions q parameterised by ν that does not contain the true posterior distribution $p(\mathcal{Z} \mid \mathcal{X})$.

over a given distribution. Given infinite computational resources, they converge to the exact result, but, in practice, sampling methods can be computationally expensive. On the other hand, deterministic methods consist of analytical approximations to the posterior and, as a consequence, cannot generate exact results. Hence, both methods are complementary.

In this text we discuss variational methods, which are deterministic. We start by its most prominent representative, Variational Inference (VI). Later, we present an alternative variational framework known as Expectation Propagation (EP).

2.2.1 Variational Inference

Essentially, VI constructs a deterministic analytical approximation to the posterior probability. Thus, it is suited to large data sets and to quickly test many models [7]. VI, Variational Bayes or Variational Bayesian Inference refer to exactly the same algorithm, and as other Bayesian methods seek to describe all available information about the variables through their probability distribution. Figure 2.6 depicts how VI works, it is the process of finding the best possible distribution q among the specified family distribution.

While standard calculus concentrates on computing derivative of functions, variational calculus focuses on derivative of functionals, which are mappings that take functions as input and output values. Several problems can be cast as functional optimisation and variational methods do exactly that for inference.

Let us suppose a model with joint distribution $p(\mathcal{X}, \mathcal{Z})$ over the set $\mathcal{X}$ of observed variables and the set $\mathcal{Z}$ of latent variables. As usual in a Bayesian setting, we wish to compute its posterior distribution $p(\mathcal{Z} \mid \mathcal{X})$, which we shall suppose intractable. Then, we consider a family of approximate, tractable densities $\mathcal{P}$ over the latent variables and try to find the member $q^*(\mathcal{Z})$ that is the "closest" to the exact posterior

in the Kullback-Leibler (KL) divergence sense:

$$q^*(\mathcal{Z}) = \underset{q(\mathcal{Z})\in\mathcal{P}}{\operatorname{argmin}} D_{KL}\left(q(\mathcal{Z})\|p(\mathcal{Z}\,|\,\mathcal{X})\right), \tag{2.10}$$

where

$$D_{KL}\left(q\|p\right) = \int q(\epsilon)\log\frac{q(\epsilon)}{p(\epsilon)}d\epsilon. \tag{2.11}$$

Directly minimising the KL is not possible because we would need the log of the true posterior $\log p(\mathcal{Z}\,|\,\mathcal{X})$, and hence the log evidence $\log p(\mathcal{X})$, which we assumed intractable. Aiming to get rid of this term, we perform some algebraic manipulations and arrive at

$$\begin{aligned}
D_{KL}(q(\mathcal{Z})\|p(\mathcal{Z}\,|\,\mathcal{X})) &= \int q(\mathcal{Z})\log\left(\frac{q(\mathcal{Z})}{p(\mathcal{Z}\,|\,\mathcal{X})}\right)d\mathcal{Z} \\
&= -\int q(\mathcal{Z})\log\left(\frac{p(\mathcal{X},\mathcal{Z})}{p(\mathcal{X})q(\mathcal{Z})}\right)d\mathcal{Z} \\
&= -\left(\int q(\mathcal{Z})\log\left(\frac{p(\mathcal{X},\mathcal{Z})}{q(\mathcal{Z})}\right)d\mathcal{Z} - \int q(\mathcal{Z})\log p(\mathcal{X})d\mathcal{Z}\right) \\
&= -\int q(\mathcal{Z})\log\left(\frac{p(\mathcal{X},\mathcal{Z})}{q(\mathcal{Z})}\right)d\mathcal{Z} + \log p(\mathcal{X})\int q(\mathcal{Z})d\mathcal{Z} \\
&= -\,\mathbb{E}_q\left[\log\left(\frac{p(\mathcal{X},\mathcal{Z})}{q(\mathcal{Z})}\right)\right] + \log p(\mathcal{X}).
\end{aligned} \tag{2.12}$$

Reorganizing the last equation we obtain

$$\log p(\mathcal{X}) = \mathbb{E}_q\left[\log\left(\frac{p(\mathcal{X},\mathcal{Z})}{q(\mathcal{Z})}\right)\right] + D_{KL}(q(\mathcal{Z})\|p(\mathcal{Z}\,|\,\mathcal{X})). \tag{2.13}$$

Observing that $D_{KL}(q\|p) \geqslant 0$, it follows that the first term of the right hand-side of (2.13) is a lower bound on $\log p(\mathcal{X})$, named the Evidence Lower Bound (ELBO). This remark leads to a very important result: maximising the ELBO is equivalent to minimising $D_{KL}(q\|p)$, what is very convenient because the right-hand side of (2.13) does not contain the log evidence. In the divergence above, the term $\log p(\mathcal{X},\mathcal{Z})$ decomposes into the log-likelihood $\log p(\mathcal{X}\,|\,\mathcal{Z})$ and the log prior $\log p(\mathcal{Z})$, which we are able to handle.

Alternatively, we could have obtained this bound straightforwardly by applying Jensen's inequality for concave functions $\mathbb{E}\left[f(x)\right] \leqslant f(\mathbb{E}\left[x\right])$ to the log marginal probability of $\mathcal{X}$, that is

$$\log p(\mathcal{X}) = \log\int p(\mathcal{X},\mathcal{Z})d\mathcal{Z}$$

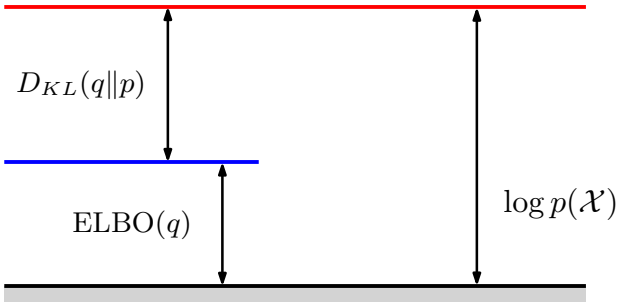


Figure 2.7: The decomposition of the marginal log-probability $p(\mathcal{X})$ into the ELBO and the $D_{KL}(q\|p)$ terms.

$$\begin{aligned} &= \log \int p(\mathcal{X}, \mathcal{Z}) \frac{q(\mathcal{Z})}{q(\mathcal{Z})} d\mathcal{Z} \\ &= \log \mathbb{E}_q \left[\frac{p(\mathcal{X}, \mathcal{Z})}{q(\mathcal{Z})} \right] \\ &\geqslant \mathbb{E}_q \left[\log \left(\frac{p(\mathcal{X}, \mathcal{Z})}{q(\mathcal{Z})} \right) \right]. \end{aligned} \tag{2.14}$$

By comparison with (2.13), the difference between the left- and right-hand sides of Equation (2.14) is exactly the KL divergence term, as shown in Figure 2.7.

We can rearrange the ELBO into the more interpretable form

$$\begin{aligned} \text{ELBO}(q) &= \mathbb{E}_q \left[\log p(\mathcal{X}, \mathcal{Z}) \right] - \mathbb{E}_q \left[\log q(\mathcal{Z}) \right] \\ &= \mathbb{E}_q \left[\mathbb{E}_q \left[\log p(\mathcal{X} \mid \mathcal{Z}) \right] + \log p(\mathcal{Z}) \right] - \mathbb{E}_q \left[\log q(\mathcal{Z}) \right] \\ &= \mathbb{E}_q \left[\log p(\mathcal{X} \mid \mathcal{Z}) \right] - D_{KL} \left(q(\mathcal{Z}) \| p(\mathcal{Z}) \right). \end{aligned} \tag{2.15}$$

The first term is the expected likelihood under the distribution $q(\mathcal{Z})$ and the second is the (negative) divergence between the $q(\mathcal{Z})$ and the prior $p(\mathcal{Z})$. While the former drives the model towards best explaining the data, the latter acts as a regulariser pushing it towards the prior $p(\mathcal{Z})$.

The ELBO is also closely related to the variational free energy $\tilde{F}$ of statistical physics, namely

$$\begin{aligned} \text{ELBO}(q) &= \mathbb{E}_q \left[\log p(\mathcal{X}, \mathcal{Z}) \right] - \mathbb{E}_q \left[\log q(\mathcal{Z}) \right] \\ &= \mathbb{E}_q \left[\log p(\mathcal{X}, \mathcal{Z}) \right] + \mathcal{H}[q] \\ &= -\widetilde{F} \end{aligned} \tag{2.16}$$

with $-\mathbb{E}_q \left[\log p(\mathcal{X}, \mathcal{Z}) \right]$ being the average of the energy function under the distribution $q(\mathcal{Z})$ and $\mathcal{H}[q]$ the entropy of $q(\mathcal{Z})$ [34, ch. 33]. Indeed, the use of the variational free energy framework in statistical learning for approximating the posterior distribution leads to the VI methodology.

Analysing Equation (2.16) we note that the solution for the first term alone is the MAP estimate of q, which maximises the log joint probability $\log p(\mathcal{X}, \mathcal{Z})$, however the negative-entropy term favours disperse distributions. The solution is then a compromise between these two terms.

Until now, we have imposed no restriction to the class of approximating distributions $\mathcal{D}$. Indeed, it could include the true posterior itself, and in cases where the posterior is tractable and belongs to $\mathcal{D}$, the method does converge to it. The class $\mathcal{D}$ should be as flexible as possible so as to better approximate the true posterior, the only restriction being its tractability. The richer the family of distributions, the closer $q^*(\mathcal{Z})$ will be to the $p(\mathcal{Z} \mid \mathcal{X})$ in Figure 2.6.

There are two main ways to constrain the class of distributions. First, by specifying a parametric form for the distribution: $q(\mathcal{Z}; \Psi)$, with the set of variational parameters being Ψ. The second, by assuming that q factorises over disjoint sets $\mathcal{S}_i$ of $\mathcal{Z}$

$$q(\mathcal{Z}) = \prod_{i=1}^{M} q_i(\mathcal{Z}_{\mathcal{S}_i}), \tag{2.17}$$

just as described by its graphical representation (Section 2.1.3).

When every latent dimension is independent of all others, this factorised form of Equation (2.17) is reffered to as Mean-Field Variational Inference (MFVI).

To find the optimal factors $q_i^*(\mathcal{Z}_{\mathcal{S}_i})$ we could solve the Lagrangian composed by the ELBO and the constraints that each factor sums to 1. However, we do not dispose of the required tools from the calculus of variations. Instead, we take a more laborious route by substituting (2.17) back into (2.16) and working out the math (available in Appendix B.1) to get

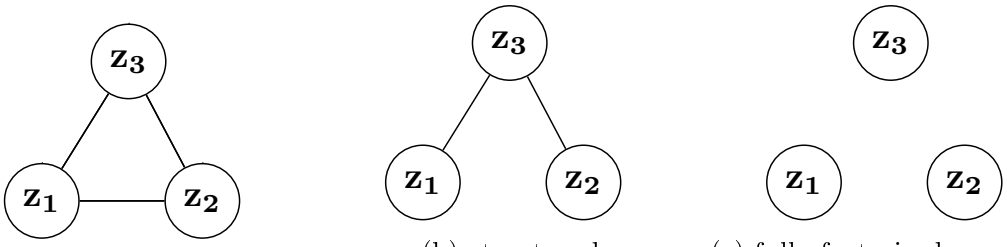
$$\log q_j^*(\mathcal{Z}_{\mathcal{S}_j}) = \mathbb{E}_{-j}\left[\log p(\mathcal{X}, \mathcal{Z})\right] + \text{const} \tag{2.18}$$

$$q_j^*(\mathcal{Z}_{\mathcal{S}_j}) \propto \exp\{\mathbb{E}_{-j}\left[\log p(\mathcal{X}, \mathcal{Z})\right]\}, \tag{2.19}$$

where $\mathbb{E}_{-j}\left[\cdot\right]$ indicates expectation over all sets $\mathcal{S}_i$ of $\mathcal{Z}$, $\mathcal{Z}_{\mathcal{S}_i}$, except $\mathcal{S}_j$.

The mutual dependence between the equations for the optimal factors indicates an iterative approach to optimise the objective function by replacing each factor by a revised estimate while keeping the others fixed (2.19). This algorithm, known as Coordinate Ascent VI (CAVI), raises the ELBO to a local optimum. An alternative approach to optimisation is gradient-based, at each iteration it calculates and follows the gradient of the objective.

So far, we consider all parameters, either local or global, either hidden units or hyper-parameters, to be whitin $\mathcal{Z}$. It is also possible to have (hyper)parameters Θ on which we perform point estimation, i.e. $p(\mathcal{Z} \mid \mathcal{X}; \Theta)$, though this would not represent



(a) True posterior (b) structured approximation (c) fully-factorised approximation

Figure 2.8: Graphical representations of the different levels of approximation to the posterior distribution as undirected graphs. On the left, the nodes in the true posterior are all dependent. At the center, z_1 and z_2 are conditionally independent, the approximation still preserves their dependency on z_3. On the right, all the nodes are marginally independent. Each approximation renders the distribution less expressive.

a fully Bayesian approach. In this case, we alternate between two distinct steps. One, approximating the posterior at each iteration by computing the *expectation* over all $\mathcal{Z}_{\mathcal{S}_i}$ as in (2.19). The other, performing the *maximisation* of the ELBO w.r.t Θ under the refined distribution $q^{\text{new}}(\mathcal{Z}) = \prod_i q_i^*(\mathcal{Z}_{\mathcal{S}_i})$. This is the Variational Expectation Maximisation (EM) algorithm. Thus VI can be understood as a fully Bayesian extension of Variational EM, in which instead of computing a point mass for the posterior over the parameters Θ (MAP estimation of Section 2.1.1), it computes the entire distribution over Θ and $\mathcal{Z}$.

The mean-field approximation to the posterior relates to how we simplify the corresponding factor graph of our model. It is very flexible, being able to capture any marginal density of the latent variables, but incapable of modelling correlation between them due to the independence assumption (Figure 2.8). This assumption is a double-edged sword, helping with scalable optimisation while limiting expressibility [7] and underestimating marginal variance. Hence the need for other families of approximations such as the already mentioned structured mean-field [35], richer covariance models [36, 37], normalising flow [38], etc.

Despite the widespread adoption of the VI framework, it still has some major issues. As presented here, it remains restricted to the conditionally conjugate exponential family for which we can compute the analytical form of the ELBO, otherwise we cannot write down a formula to optimise. Section 2.2.5 briefly presents methods that try to address this problem. Furthermore, even though minimising the KL divergence $D_{KL}\left(q(\mathcal{Z}) \| p(\mathcal{Z} \mid \mathcal{X})\right)$ and maximising the ELBO are equivalent optimisation problems, the KL is bounded below by zero no matter what the distribution p and q may be, while the ELBO has no bound whatsoever. Therefore, the KL automatically informs us how good is the approximation and how close it is to the true posterior. On the other hand, the ELBO has no absolute scale to compare with

so we have no clue how far it is from the true distribution, though it asymptotically converges to a given value that we can use for model selection.

2.2.2 Assumed Density Filtering

Assumed Density Filtering (ADF) has been independently proposed in the statistics, artificial intelligence, and control domains [39]. Its central idea relies on the model's joint probability decomposing into a product of independent factors. Instead of trying to approximate all factors at once, we sequence through each factor, including it into the current approximation. Suppose a joint distribution and its posterior

$$p(\mathcal{X}, \mathcal{Z}) = \prod_{i=1}^{N} f_i(\mathcal{Z}) \tag{2.20}$$

$$p(\mathcal{Z} \mid \mathcal{X}) = \frac{1}{p(\mathcal{X})} \prod_{i=1}^{N} f_i(\mathcal{Z}), \tag{2.21}$$

where the dependency of the factors $f_i(\cdot)$ on $\mathcal{X}$ is made implicit. Also consider an approximation to the posterior (2.21) of the form

$$q(\mathcal{Z}) = p_0(\mathcal{Z}) \prod_{i=1}^{N} \frac{1}{\widetilde{K}_i} \widetilde{f}_i(\mathcal{Z}), \tag{2.22}$$

where $p_0(\mathcal{Z})$ is the prior, $\widetilde{f}_i(\mathcal{Z})$ are the compatibility functions, and $\widetilde{K}_i$ their normalising constants. The prior of choice should have a nice form to work with since ADF consists of projecting the distributions back to the family of the prior after each update.

Let us further call

$$p_i(\mathcal{Z} \mid \mathcal{X}) = \frac{1}{p(\mathcal{X})} \prod_{k=1}^{i} f_k(\mathcal{Z}) \propto \prod_{k=1}^{i} f_k(\mathcal{Z}), \quad \forall\, 1 \leq i \leq N. \tag{2.23}$$

Basically, at iteration $1 \leq i \leq N$, that is, until all factors are seen, we have to find the best $q^{(i)}(\mathcal{Z})$ such that

$$q^{(i)}(\mathcal{Z}) = q^{(i-1)}(\mathcal{Z}) \frac{1}{\widetilde{K}_i} \widetilde{f}_i(\mathcal{Z}) \approx q^{(i-1)}(\mathcal{Z}) f_i(\mathcal{Z}) \propto p_i(\mathcal{Z} \mid \mathcal{X}). \tag{2.24}$$

At each step, the approximation gets "slightly" warped by the true factor $f_i(\mathcal{Z})$ so we project it back to the desired approximating family. From this, we immediately note that the product of all $\widetilde{K}_i$ gives an estimate of the model evidence $p(\mathcal{Z})$ and that the final approximating posterior has the same form as the prior.

This projection consists in minimising the KL divergence between the two dis-

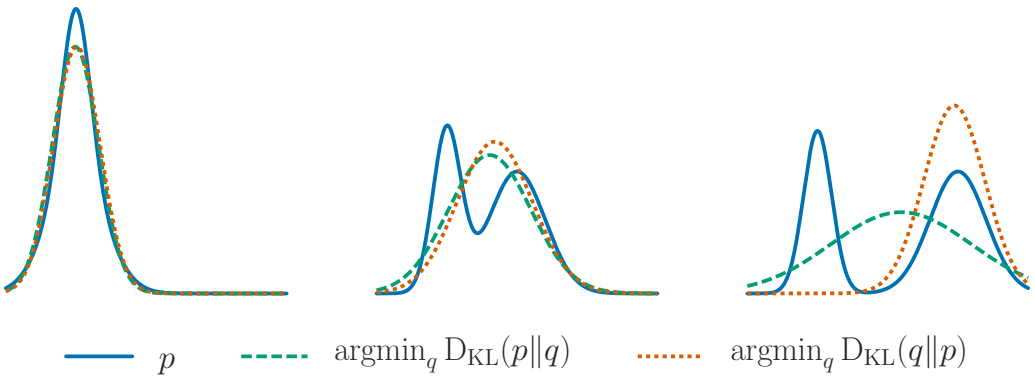


Figure 2.9: Comparison of the two alternatives forms of the KL divergence in different scenarios. The blue curve is a mixture of two Gaussians, while in the leftmost graph their mean intersect resulting in a single mode, for the two other cases the distribution becomes bimodal. The green dashed curve corresponds to the distribution q that best approximates p in the forward KL sense, whereas the red dotted curve is the best approximation according to the reverse KL. As the modes of p get farther apart, $D_{KL}(q\|p)$ seeks the most probable mode while $D_{KL}(p\|q)$ strives for the global average.

tributions. However, differently from Section 2.2.1, we now employ the *forward* KL divergence $D_{KL}(p\|q)$ for measuring the quality of the approximation (note the ordering of the arguments). This minor tweak is actually the reason why ADF (and EP in Section 2.2.3) behave so differently from VI. Since KL is a divergence and not a distance, the symmetry property does not hold. Hence exchanging their order leads to a distinct functional with distinct properties and issues.

The reverse KL divergence $D_{KL}(q\|p)$ defined in (2.11) severely penalises the approximating distribution q for placing mass in regions where p has low probability

$$D_{KL}(q\|p) = \mathbb{E}_q\left[\log q(x)\right] - \mathbb{E}_q\left[\log p(x)\right] dx. \tag{2.25}$$

The term $\log p(x) \to -\infty$ for such regions. Conversely, by exchanging p and q in Equations (2.11, 2.25) one gets

$$D_{KL}(p\|q) = \mathbb{E}_p\left[\log p(x)\right] - \mathbb{E}_p\left[\log q(x)\right] dx. \tag{2.26}$$

The forward KL has the opposite behaviour, that is, it favours spreading the mass of q over the support of p. Even low probability regions of p must have mass attributed to in q to avoid obtaining samples from $p(x)$ that have $\log q(x) \to -\infty$. Figure 2.9 neatly illustrates this property for both KL forms.

In order to be efficiently calculated, the posterior distribution must be simple to handle. So we further constrain the factors $\widetilde{f}_i$, and hence the posterior, to belong to

the exponential family

$$\widetilde{f}_i(\mathcal{Z}) = h(\mathcal{Z})g(\boldsymbol{\eta})\exp\left(\boldsymbol{\eta}^T\mathbf{u}(\mathcal{Z})\right), \tag{2.27}$$

where $h(\mathcal{Z}) \geq 0$ is the *carrier* functon, $\boldsymbol{\eta}$ the *natural parameters* of the distribution, $g(\boldsymbol{\eta})$ the *partition function*, and $\mathbf{u}(\mathcal{Z})$ the *sufficient statistics* which contain all the necessary information the samples can provide about the unknown parameters of the distribution.

From this restriction, it stems that q, being a product of all $\widetilde{f}_i$ sites, also belongs to the exponential family and the forward KL divergence reduces to

$$\begin{aligned} D_{KL}(p\|q) &= \int p(\mathcal{Z})\log p(\mathcal{Z})d\mathcal{Z} - \int p(\mathcal{Z})\log q(\mathcal{Z})d\mathcal{Z} \\ &= \int p(\mathcal{Z})\log p(\mathcal{Z})d\mathcal{Z} - \int p(\mathcal{Z})\log\left(h(\mathcal{Z})g(\boldsymbol{\eta})\exp\left(\boldsymbol{\eta}^T\mathbf{u}(\mathcal{Z})\right)d\mathcal{Z}\right) \\ &= \int p(\mathcal{Z})\log p(\mathcal{Z})d\mathcal{Z} - \left(\mathbb{E}_p\left[h(\mathcal{Z})\right] + \log g(\boldsymbol{\eta}) + \boldsymbol{\eta}^T\,\mathbb{E}_p\left[u(\mathcal{Z})\right]\right). \end{aligned} \tag{2.28}$$

In this scenario, we are interested in finding the natural parameters $\boldsymbol{\eta}$ that specify, among the imposed member of the exponential family, the element that minimises the KL. Thus we set

$$\begin{aligned} \nabla_{\boldsymbol{\eta}}\int p(\mathcal{Z})\log p(\mathcal{Z})d\mathcal{Z} - \left(\mathbb{E}_p\left[h(\mathcal{Z})\right] + \log g(\boldsymbol{\eta}) + \boldsymbol{\eta}^T\,\mathbb{E}_p\left[\mathbf{u}(\mathcal{Z})\right]\right) &= 0 \\ -\nabla_{\boldsymbol{\eta}}\log g(\boldsymbol{\eta}) - \mathbb{E}_p\left[\mathbf{u}(\mathcal{Z})\right] &= 0 \\ \nabla_{\boldsymbol{\eta}}\log g(\boldsymbol{\eta}) &= -\,\mathbb{E}_p\left[\mathbf{u}(\mathcal{Z})\right]. \end{aligned} \tag{2.29}$$

From the fact that any normalised distribution must sum to one, we arrive at a general result of the exponential family by deriving w.r.t. $\boldsymbol{\eta}$, such that

$$\begin{aligned} &\nabla_{\boldsymbol{\eta}}\int h(\mathcal{Z})g(\boldsymbol{\eta})\exp\left(\boldsymbol{\eta}^T\mathbf{u}(\mathcal{Z})\right)d\mathcal{Z} = \nabla_{\boldsymbol{\eta}}1 \\ &\nabla_{\boldsymbol{\eta}}g(\boldsymbol{\eta})\int h(\mathcal{Z})\exp\left(\boldsymbol{\eta}^T\mathbf{u}(\mathcal{Z})\right)d\mathcal{Z} + \int \mathbf{u}(\mathcal{Z})h(\mathcal{Z})g(\boldsymbol{\eta})\exp\left(\boldsymbol{\eta}^T\mathbf{u}(\mathcal{Z})\right)d\mathcal{Z} = 0 \\ &\nabla_{\boldsymbol{\eta}}g(\boldsymbol{\eta})\frac{1}{g(\boldsymbol{\eta})} + \mathbb{E}_q\left[\mathbf{u}(\mathcal{Z})\right] = 0 \\ &\nabla_{\boldsymbol{\eta}}\log g(\boldsymbol{\eta}) = -\,\mathbb{E}_q\left[\mathbf{u}(\mathcal{Z})\right]. \end{aligned} \tag{2.30}$$

Substituting (2.29) in (2.30) we arrive at

$$\mathbb{E}_q\left[\mathbf{u}(\mathcal{Z})\right] = \mathbb{E}_p\left[\mathbf{u}(\mathcal{Z})\right], \tag{2.31}$$

which means that when approximating an arbitrary distribution with an exponential

distribution, we should match their moments.

Thus in the ADF methods, it all comes down to matching the moments of the new approximation with the moments of the previous approximation tilted by the newly included true factor. If, for example, we consider a Gaussian posterior approximation $q(\mathcal{Z}) = \mathcal{N}(\mathcal{Z}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$, then we should select $\boldsymbol{\mu}_i$, $\boldsymbol{\Sigma}_i$, and K_i for the distribution $q^{(i)}$ such that

$$\boldsymbol{\mu}_i = \mathbb{E}_{q^{(i-1)}\widetilde{f}_i}[\mathcal{Z}], \tag{2.32}$$

$$\boldsymbol{\Sigma}_i = \mathrm{Cov}_{q^{(i-1)}\widetilde{f}_i}[\mathcal{Z}], \tag{2.33}$$

$$\int q^{(i)}(\mathcal{Z})d\mathcal{Z} = \frac{1}{\widetilde{K}_i}\int q^{(i-1)}(\mathcal{Z})\widetilde{f}_i(\mathcal{Z})d\mathcal{Z} = 1. \tag{2.34}$$

Even though this sequential approach is better than independently approximating each factor, it depends on the ordering of the factors. If the first factors lead to a bad approximation, the ADF produces a poor final estimate of the posterior. We would mitigate this issue at the expense of losing the online characteristic of the method by revising the initial approximations after taking the later factors into account, effectively cycling through them all,

We note that the assumption of factorisable distributions is still pretty general. Indeed, we frequently assume the observed data is iid distributed given the parameters, consequently inducing the factorisation of (2.21) over the likelihood term. In addition, if considering a graphical model, the distribution can be factored according to its structure, so the factors in (2.21) could represent sets of nodes of the graph.

Examples of ADF are projecting a Gaussian mixture posterior onto a single Gaussian [39] and projecting a Student's t-distribution onto a Gaussian (Section 3.4).

2.2.3 Expectation Propagation

As mentioned in the previous section, one of ADF's weakness is its sensitivity to the order in which the factors are considered. In a batch setting, where all factors are available at all times, it is unreasonable not to refine the previously approximated compatibility functions after observing all true factors. However, directly cycling through them n times would amount to including each factor n times as if there were n times more factors, e.g., n times more observed data. In doing so, we would continuously accumulate evidence, making the likelihood get sharper and sharper, until the prior would become irrelevant and the posterior would eventually collapse to a single point and we certainly do not want to artificially induce that.

The posterior would eventually collapse to a single point and we certainly do not want that.

EP reinterprets ADF as first approximating each observation term f_i with some

$\widetilde{f}_i$ and them combining these approximations analytically to obtain the posterior, instead of approximating the posterior itself to include f_i.

First, we initialise the compatibility functions $\widetilde{f}_i$, generally $\widetilde{f}_i = 1$, and compute the posterior as their product. Then, iteratively until convergence, we choose a compatibility function $\widetilde{f}_i$, and remove it from the current approximation to the posterior, computing the unnormalised *cavity distribution*

$$q_{-i}(\mathcal{Z}) = q_i(\mathcal{Z})/\widetilde{f}_i(\mathcal{Z}), \tag{2.35}$$

the *tilted distribution* q^{tilt} and its normalisation constant K_i

$$q^{tilt} = \frac{q_{-i}(\mathcal{Z})f_i(\mathcal{Z})}{K_i}, \tag{2.36}$$

$$K_i = \int q_{-i}(\mathcal{Z})f_i(\mathcal{Z})d\mathcal{Z}, \tag{2.37}$$

such that

$$q^{\text{new}}(\mathcal{Z}) = \underset{q}{\operatorname{argmin}}\, D_{KL}\left(q^{tilt}(\mathcal{Z})\|q(\mathcal{Z})\right). \tag{2.38}$$

Since the compatibility functions belong to the exponential family, dividing and multiplying them amounts to subtracting and summing their natural parameters $\boldsymbol{\eta}$ so these operations are computationally efficient. Likewise, the KL minimisation is done by matching the moments of the new distribution $q^{\text{new}}(\mathcal{Z})$ with the tilted distribution according to (2.31), as in Section 2.2.2.

From this point, evaluation of the new updated factor is also simple

$$\widetilde{f}_i(\mathcal{Z}) = K_i\frac{q^{\text{new}}(\mathcal{Z})}{q^{\setminus i}(\mathcal{Z})}. \tag{2.39}$$

Broadly speaking, each iteration consists of refining the approximation of the $\widetilde{f}_i(\mathcal{Z})$ by substituting its contribution by that of true factor $f_i(\mathcal{Z})$, and finding the q^{new} that minimises the KL. Even though EP approximates one factor at a time and the resulting q is a valid probability distribution, partial products thereof not necessarily represent a valid distribution.

Naturally, the enhancement provisioned by EP has costs. Besides being unsuited to online learning, differently from ADF, it has to keep each compatibility function stored in memory because it is needed to update the approximation. So, memory consumption grows linearly with the number of factors of the distribution. This may be inadequate if data sets are too large as it would be impractical or even impossible to maintain all factor in memory during optimisation.

While each step in VI is guaranteed not to hamper the cost function, namely

to decrease the ELBO, the described EP algorithm has no convergence guarantees and iterations may indeed increase the associated energy function instead of decreasing it [40, p. 510]. Nonetheless, stable EP fixed points are local minima of the optimisation problem [39].

Power EP

Not every distribution can be factored into simple terms and, hence, integrating such factors is not a simple task either. Consequently, EP fails to be computationally efficient. Power EP [41] addresses this shortcoming by cleverly raising the factors f_i to a power which cancels complicating exponents present in the true factors, thus making them easier to compute. The algorithm is essentially the same, except we perform it on the "fractional factors"

$$f_i'(\mathcal{Z}) = f_i(\mathcal{Z})^{1/n_i}, \tag{2.40}$$

$$\widetilde{f}'_i(\mathcal{Z}) = \widetilde{f}_i(\mathcal{Z})^{1/n_i}. \tag{2.41}$$

For $n_i \geqslant 1$ this amounts to artificially splitting the factors in $1/n_i$ distinct but equal terms during the optimisation, making the contribution of each term to the posterior weaker.

If we replace the divergence that the EP minimises, $D_{KL}\left(p\|q\right)$, by

$$D_\alpha(p\|q) = \frac{4}{1-\alpha^2}\left(1 - \int p(x)^{(1+\alpha)/2} q(x)^{(1-\alpha)/2}\right) dx, \tag{2.42}$$

where $-\infty < \alpha < \infty$ is a continuous parameter, the resulting algorithm we obtain will have the same fixed points as the Power EP. Therefore, we can think of Power EP as minimising the α-divergence D_α with α corresponding to a particular choice of $1/n_i$, namely $\alpha = 2(1/n_i) - 1$.

The forward and reverse KL divergences are members of the α-family in (2.42) for which $\alpha \to 1$ and $\alpha \to -1$, respectively. Values $\alpha \leqslant -1$ induce a zero-forcing behaviour, setting $q(x) = 0$ for any values of x for which $p(x) = 0$. Conversely, $\alpha \geqslant 1$ is zero-avoiding, imposing $q \geqslant 0$ for regions where $p \geqslant 0$, and typically q stretches to cover all p.

Actually, one way to understand many message-passing algorithms, including those we discussed, is simply as the same variational framework but with the minimisation of different energy functions corresponding to distinct choices for the value of α in (2.42) [42].

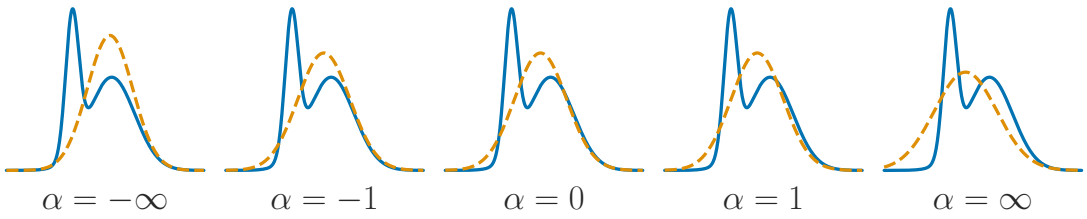


Figure 2.10: The α-divergence family. For $\alpha \to -1$ it becomes the reverse KL $D_{KL}(q\|p)$, while for $\alpha \to 1$ it is the forward KL $D_{KL}(p\|q)$.

2.2.4 Stochastic Variational Inference

As the name suggests, Stochastic Variational Inference (SVI) relies on stochastic approximation [43], that is, it computes noisy estimates of the gradient (ordinarily called as mini-batch statistics) to optimize the objective function. This approach is ubiquitous on modern ML since it is much faster than assessing a massive data set, which is commonplace nowadays. Thus, we shall stumble upon this idea on later chapters.

The major requirements for the approximation to be valid are that the gradient estimator should be unbiased and that the step size sequence $\{\alpha_i \,|\, i \in \mathcal{N}\}$ (also known to Deep Learning practitioners as learning rate) that nudges the parameters towards the optimal should be annealed so that

$$\sum_{i=0}^{\infty} \alpha_i = \infty \qquad , \qquad \sum_{i=0}^{\infty} \alpha_i^2 < \infty. \tag{2.43}$$

Intuitively, the first condition relates to the exploration capacity so the algorithm may find good solutions no matter where it is initialized, whereas the second guarantees its energy is bounded so that it can converge to the solution.

SVI is a stochastic optimisation algorithm for MFVI. Instead of computing the expectation step in (2.19) for all N data points (at every iteration), we do it for a uniformly sampled (with replacement) subset of size n, possibly a single data point. From these new variational parameters, we compute the maximisation step (or the expectation of the global variational parameters) as though we observed the data points N/n times, and update the estimate as the weighted average of the previous estimate and the subset optimal according to (2.43).

Theoretically, this process should go on forever with increasingly smaller step sizes according to the constraints stated above, but in practice it ends when it reaches a stopping criteria, which indicates the ELBO has converged.

In the original paper, the authors propose noisy estimates of the **natural** gradient of the variational objective [44]. Essentially, the natural gradient is the inverse of the Fisher information matrix multiplied by the standard gradient. For more

information see Appendix A.2. There is actually much more to this work, but a lengthy discussion is not our focus.

2.2.5 Further Practical Extensions

In this section, we briefly mention some modern extensions of the approximate inference algorithms we have seen so far. While the first two address computability and tractability issues, the last aims at usability, making VI more accessible.

Black Box Variational Inference

SVI, Section 2.2.4, computes the distribution updates in closed form, which requires model-specific knowledge and implementation, as well as the existence of closed-form analytical formulae for the (natural) gradient of the ELBO and its actual computation. Instead, Black Box Variational Inference (BBVI) [11] uses the score function estimator of Section A.1 to compute the (regular) gradient of the ELBO. This leads to

$$\nabla_\phi L(\phi) = \mathbb{E}_q \left[(\nabla_\phi \log q(z|\phi))(\log p(x,z) - \log q(z|\phi)) \right], \tag{2.44}$$

that we approximate by Monte Carlo integration and employ to perform stochastic optimisation.

The sole assumption of the gradient estimator in (2.44) about the model is the feasibility of computing the log of the joint $p(x, z_s)$. The sampling method and the gradient of the log both rely on the variational distribution q. Thus, we can derive them only once for each approximating family q and reuse them for different models $p(x, z_s)$. Hence the name black box: we just specify the generative model $p(x, z_s)$ and are already able to perform VI on it. Actually $p(x, z_s)$ needs not even to be normalised since the log of the normalisation constant does not contribute to the gradient in (2.44).

Nevertheless, as noted in Appendix A.1, the variance of the estimator may be too high, forcing the step sizes to be too small for the algorithm to be practical. Therefore, the authors in [11] further consider variance reduction methods that preserve the black box character of BBVI. Though the framework applies for general variational families, they consider fully factorised posterior distributions when deriving the variance reduction methods.

Gradient estimators

In inference problems, as well as in other domains, we frequently encounter the computation of $\nabla_\phi \mathbb{E}_{q(z;\phi)}[f(z;\theta)]$. Generally, we cannot compute this gradient directly and need obtain appropriate practical estimators.

If $q(z;\phi)$ is a continuous function of ϕ, we can use the *score-function estimator* [45]

$$\nabla_\phi \mathbb{E}_{q(z;\phi)}[f(z;\theta)] = \mathbb{E}_{q(z;\phi)}[f(z;\theta)\nabla_\phi \log q(z;\phi)]\,. \tag{2.45}$$

Alternatively, if we can express the random variable $z \sim q(z,\phi)$ as an invertible deterministic transformation $g(\cdot;\phi)$ of a base random variable $\epsilon \sim p(\epsilon)$, we may use the *pathwise derivative estimator* [46]

$$\nabla_\phi \mathbb{E}_{q(z;\phi)}[f(z;\theta)] = \mathbb{E}_{p(\epsilon)}[\nabla_\phi f(g(\epsilon;\phi);\theta)]\,. \tag{2.46}$$

While both estimators yield unbiased estimates, the score function estimator generally has higher variance.

Black Box α Minimisation

Black Box α Minimisation [47] (BB-α) optimises an approximation of the power EP energy function [42, 48]. Instead of considering i different local compatibility functions $\widetilde{f}_i$, it ties them together so that $\widetilde{f}_i = \widetilde{f}$. It is as if we were using an average factor approximation to approximate the average effect of the original f_i [47]. Restricting these factors to belong to the exponential family, this simplification amounts to tying their natural parameters. As a consequence, BB-α no longer needs to store an approximating site per likelihood factor, which leads to significant memory savings in large data sets. This results in fixed points that differs from power EP, though they are equal in the limit of infinite data.

BB-α dispenses with the need for double-loop algorithms to directly minimise the energy and employs gradient-descent methods for this matter. This is in contrast with the iterative update scheme of Section 2.2.3. As other modern methods designed for large scale learning, it employs stochastic optimisation to avoid cycling through the whole data set. Besides, it estimates the expectation over the approximating distribution q present in the energy function by Monte Carlo sampling.

Differently from BBVI [11], seen in Section 2.2.5, this method uses the reparameterisation trick to estimate the gradient of said expectation. Thus, it requires in addition to the likelihood function, its gradients. Still, they can be readily obtained with automatic differentiation if the likelihood is analytically defined and

differentiable.

As observed in Section 2.2.3, the parameter α in (2.42) controls the divergence function. Hence, the method is able to interpolate between VI ($\alpha \to -1$) and an algorithm similar to the expectation propagation (EP) ($\alpha \to 1$). Interestingly, the authors [47] claim to usually obtain the best results setting $\alpha = 0$, halfway through VI and EP. This value corresponds to the Hellinger distance, the sole member of the α-family that is symmetric.

Automatic Differentiation Variational Inference

Automatic Differentiation Variational Inference (ADVI) offers a recipe for automating the computations involved in variational inference [49]. The user then only provides the desired probabilistic model and the data set, the framework occupies itself of all the remaining blocks of the pipeline. There is no need to derive the objective function nor its derivatives for each specific combination of approximating family and model.

First, it applies a transformation $T : \mathcal{Z} \mapsto \Xi$ that maps the support of the latent variables $\mathcal{Z}$ to all real coordinate space, such that the model's joint distribution $p(\mathcal{X}, \mathcal{Z})$ becomes $p(\mathcal{X}, \Xi)$. It then approximates $p(\mathcal{X}, \Xi)$ with a Gaussian distribution, though other variational approximating families are possible. Even the simple Gaussian case induces non-Gaussian distributions in the original latent space $\mathcal{Z} = T^{-1}(\Xi)$. As usual, the ELBO (2.14) involves an intractable expectation and ADVI resorts to the reparameterisation trick (Appendix A.1) to convert the variational distribution to a deterministic function of the standard Gaussian $\mathcal{N}(0, 1)$, thus allowing automatic differentiation. Finally, it estimates the expectation over the latent space by Monte Carlo integration, producing noisy unbiased gradients of the ELBO and performing stochastic optimisation [43]. Since ADVI employs the pathwise gradient estimator, it works only for differentiable models. Besides having continuous latent variables, the derivative of the log joint probability $\nabla_z \log p(\mathcal{X}, \mathcal{Z})$ must exist. On the other hand, BBVI [11] computes the derivative of the variational approximation q and is, thus, more general, though it can suffer from high variance. Although the performance of the resulting ADVI model may not be as good as if the whole process were manually implemented, it works well for a large class of practical models on modern data sets [49]. Therefore, it allows rapid prototyping of new ideas and corrections of complex models.

Chapter 3

(Scalable) Bayesian Neural Networks

This chapter presents the ideas, derivations, advantages, and issues of 4 different algorithms for BNNs:

- Bayes By Backpropagation [13];
- Probabilistic Backpropagation [14];
- MC-Dropout [15];
- Variation Adam (Vadam) [16].

Each method approaches the problem in a considerably different manner. Still, they all share one trait in common: they all consider unstructured approximations to the posterior. With exception of MC-Dropout [15], which assumes dependency among groups of weights, but in a rather non-well-defined manner as we shall verify, all others rely on mean-field approximations.

By the end of this chapter, the reader should:

- Know the attributes a BNN should possess.
- Learn metrics to assess such characteristics.
- Discern the benefits and issues of each method.
- Understand the differences among them.
- Be capable of choosing the one that best suits oneself's needs.
- Know where to search if in need of structured BNNs.

3.1 Why BNNs?

Recently, Bayesian Neural Networks (BNNs) have been object of renewed interest within the research community. As one may imagine by now, BNNs are essentially standard deterministic NNs enhanced with Bayesian methods. Instead of learning the optimal weights $\mathcal{W}^*$, they infer the posterior weight distribution $p(\mathcal{W} \mid \mathcal{D})$ given the data set $\mathcal{D}$, whose maximum correspond to the $\mathcal{W}^*$ point in weight-space, i.e., Figure 3.1.

First introduced in [50], BNNs saw a great advance during the following years (the 1990's) [12, 51–53]. However, due to its computational complexity, they ended up relegated for a decade. Standard NNs had not had success for a long time, only picking up momentum in 2006 [54] and effectively gaining attention in 2012 after a deep learning method [4] won an important image classification competition [55] by a large margin. It certainly would not go differently for BNNs, which faced an even more difficult scenario. Over the last few years, new practical approaches to BNNs [8, 56] allied to the security flag raised by adversarial attacks [57] and the cry for uncertainty measures quintessential for some practical applications sparked interest in Bayesian methods for deep learning.

The main reason for the late acceptance of BNNs (which is still to come) is that its computational complexity impedes scalability. Modern models and data sets have, respectively, millions of parameters and instances, so nothing but very simplistic algorithms can handle well such large-scale regime. A clear example is the use of backpropagation and first-order optimisation methods (though that does not mean they are not ingenious). Consequently, latest works in this field focus on scalable and (most of the time) practical approaches that can meet the current demand and are still comprehensible, or at least usable, by practitioners.

For those still not convinced about the benefits of being Bayesian, we quickly review the state of affairs for modern deep learning.

Even though backpropagation and maximum likelihood optimisation allow fit-

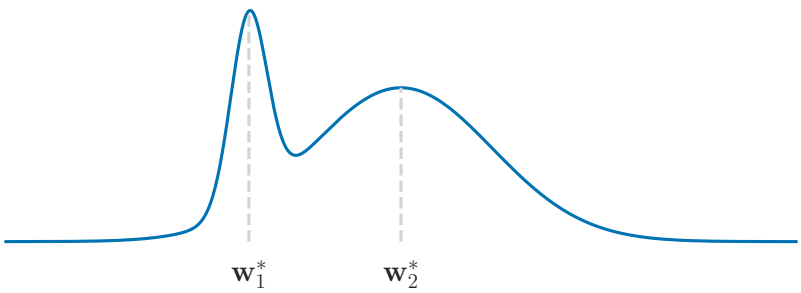


Figure 3.1: Maximiser $\mathcal{W}_1^*$ of posterior distribution and runner-up $\mathcal{W}_2^*$ corresponding to the maximum of another mode

ting large nonlinear models on massive amounts of data and find success on several tasks, they are sensitive to overfitting, specially if we try such models on not so large data sets. Employing common regularization techniques (as ℓ_1 and ℓ_2) is equivalent to maximum a posteriori optimisation (with Laplace and Gaussian priors, respectively), and in spite of alleviating overfitting, it is far from solving the problem. What is more, it makes the solution parameterisation dependent, that is, different parameterisations may lead to different optimal points. Then, one questions which parameterisation leads to the best possible solution and how sensitive it is. Even when resorting to invariant methods, we still have no measure of confidence and though bootstrapping alleviates the issue, it does not resolve the underlying problem. The Bayesian framework solves all this at once by allowing models to represent not only single point estimates but complete distributions over all possible parameters' values. It offers a unified framework for model building, inference, prediction, and decision-making. Moreover, it provides a straightforward way to score models (through the model evidence) and select among them.

There is no free lunch, and as already hinted above, BNNs have challenging inference. They rely on conditioning and marginalisation, so the main operation is integration. Thus, high-dimensional and/or complex models impose a real barrier to its deployment. We discuss approaches that mitigate this issue by employing distributional approximations (Section 2.2) to render computations amenable. Particularly, we focus on those that do not explicitly impose structure on weights, and instead assume them independent (mean-field approximation, Section 2.2.1).

For ease of notation, we shall use w as the random variable instead of z. This change of notation is not only to keep similarity to the literature in BNNs, but also to remind our readers that the distributions are over the model's weights (the parameters) and not hidden units.

3.2 Do They *Really* Know How Much They Do Not Know?

Bayesian and, more generally, probabilistic models output some measure of uncertainty in which we trust to make decisions. Can we really believe in these models? Do they reflect, approximately at least, the reality? As an example of uselessness, just imagine the case where the model predicts constant or random level of uncertainty. Having a reliable measure of confidence intuitively means the model is correct about 80% of the time if it assigns an 80% probability of being correct or, in a regression setting, the true value falling 80% of the time in an 80% credible interval around the prediction. This property is what we call calibration [58].

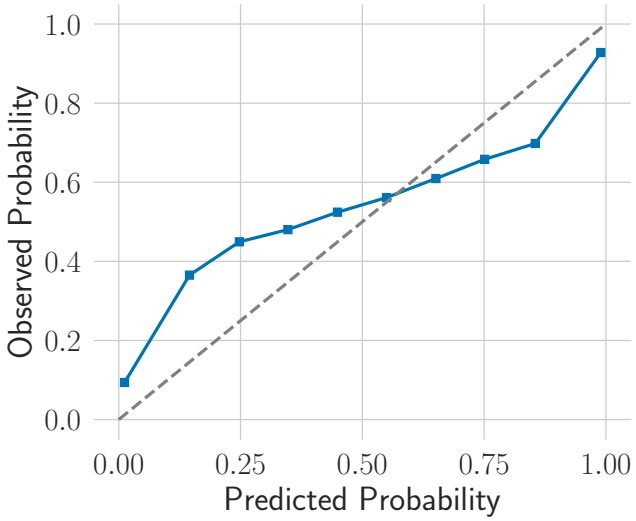


Figure 3.2: Example of calibration plot. The gray dashed line is the identity $y = x$ and the blue curve the calibration curve of an imaginary model. Ideally, we want the blue and gray line to be superposed, indicating a perfectly well calibrated model.

A common diagnostic tool for calibration is the reliability (or calibration) plot, shown in Figure 3.2. Ideally, the empirical and the predictive cumulative distribution functions should match, so plotting one against the other should give a graph as close as possible to the identity $y = x$. Namely, for each credible interval corresponding to a probability threshold p_i, we plot the observed number of times (empirical frequency) the prediction falls within the interval. We can measure the calibration error numerically by computing the (mean) squared error between the predicted and empirical frequencies for m different confidence intervals.

Of course calibration is not enough, forecasts also need to be sharp [58]. Intuitively, credible intervals should be as tight and probabilities as binary as possible in regression and classification, respectively. A model that always predicts the mean value and adjusts its confidence accordingly is calibrated by definition, but not useful. There are various ways to measure spread, variance being one of them.

3.3 Bayes by Backprop

The first method we review has a quite long history preceding it. Bayes by Backprop [13], or BBB for short, continues the work of [8] on practical VI for NNs, who in turn extends on [53]. Likewise, we separate them and build the section one method at a time.

3.3.1 Minimising the Description Length

In [53], Hinton *et al.* derive from an Information Theory perspective, using the Minimum Description Length principle, a diagonal Gaussian variational approximation to the posterior weight distribution, i.e., no correlation among weights, so they can be thought of as being independent univariate Gaussians. This was the first time VI was proposed for neural networks. The method requires deriving analytical solutions to the integrals over the variational posteriors, which are not only difficult but generally unavailable for complex systems (in [53], the authors use a single hidden-layer network with linear outputs).

3.3.2 Practical VI

Fast-forward 18 years to 2011 and Graves [8] drops the analytical solutions in favor of approximations with numerical integration. The essence of the approach is choosing a variational posterior $q(\mathcal{W}; \Psi)$ from which probable samples can be drawn efficiently so it is amenable to numerical methods, namely Monte Carlo integration. Again focusing on a diagonal Gaussian posterior, each network weight w_i requires separate mean μ_i and variance σ_i^2, so that $\Psi_i = \{\mu_i, \sigma_i^2\}$ and the set of all parameters is $\Psi = \{\boldsymbol{\mu}, \boldsymbol{\sigma}^2\}$. We express the approximating variational posterior distribution by

$$q(\mathcal{W}; \Psi) = \prod_i q(w_i; \Psi_i) = \prod_i \mathcal{N}(w_i; \mu_i, \sigma_i^2). \tag{3.1}$$

Recall that the variational objective to minimise is the negative ELBO, already defined in (2.15). We rewrite it here

$$\begin{aligned} \mathcal{L}(q) &= -\text{ELBO}(q) \\ &= -\mathbb{E}_{q(\mathcal{W};\Psi)}\left[\log p(\mathcal{D} \mid \mathcal{W})\right] + D_{KL}\left(q(\mathcal{W}; \Psi) \| p(\mathcal{W})\right) \\ &= \mathcal{L}_{data} + \mathcal{L}_{prior}, \end{aligned} \tag{3.2}$$

and make explicit the presence of two cost functions of different nature. The first, $\mathcal{L}_{data}$, which we refer to as the likelihood cost, is data-dependent and quantifies the amount of error the model commits. The second, $\mathcal{L}_{prior}$, is prior-dependent and we call it the complexity cost. While the former drives the model towards best explaining the data, the latter acts as a regulariser pushing towards the prior $p(\mathcal{W})$, as already explained in Section 2.2.1. For practicality, we also call $\mathcal{L}_N = -\log p(\mathcal{D} \mid \mathcal{W})$.

The diagonal Gaussian posterior (3.1) entails a non-closed form for $\mathcal{L}_{data}$ and its derivatives wrt μ_i and σ_i. Graves [8] resorts to MC integration, i.e., drawing different weights $\mathcal{W}_t$ from the posterior $q(\mathcal{W}; \Psi)$, performing the desired computation for

each sample and averaging the results. For $\nabla_{\sigma_i}\mathcal{L}_{data}$, one further needs second-order derivatives wrt the weights, $\nabla^2_{w_i}\mathcal{L}_{data}$, and following [8] we resort to yet another approximation, as we show next.

In order to compute the derivatives, Graves [8] uses the fact that the expectations are over the Gaussian distribution [59] and employs the identities (proven in Appendix B.2)

$$\frac{\partial \mathbb{E}_q\left[f(\mathcal{W})\right]}{\partial \mu_i} = \mathbb{E}_q\left[\frac{\partial f(\mathcal{W})}{\partial w_i}\right] \tag{3.3}$$

$$\frac{\partial \mathbb{E}_q\left[f(\mathcal{W})\right]}{\partial \sigma_i^2} = \frac{1}{2}\,\mathbb{E}_q\left[\frac{\partial^2 f(\mathcal{W})}{\partial w_i^2}\right]. \tag{3.4}$$

where f in this case is $\mathcal{L}_N$ and its expected value $\mathcal{L}_{data}$.

These identities are useful because they allow for unbiased gradient estimates and have low variance when doing Monte Carlo, but requires second order derivatives. The mean-field assumption saves us from computing the full Hessian $\nabla^2_{\mathcal{W}}\mathcal{L}_{data}$, but its diagonal is still necessary. Still, it is costly and we seek to approximate it.

Besides the MC sampling to estimate the expectations, we use the Generalised Gauss-Newton (GGN) approximation to the Hessian $\nabla^2_{\mathcal{W}} f(\mathcal{W}) \approx \nabla_{\mathcal{W}} f(\mathcal{W}) \nabla_{\mathcal{W}} f(\mathcal{W})^T$ (further details in Appendix A.3) and end up with

$$\frac{\partial \mathbb{E}_q\left[f(\mathcal{W})\right]}{\partial \sigma_i^2} \approx \frac{1}{2}\,\mathbb{E}_q\left[\left(\frac{\partial f(\mathcal{W})}{\partial w_i}\right)^2\right]. \tag{3.5}$$

This approximation spares us from second order derivatives, but introduces bias into the estimation of the variance gradient, that is, its expected value no longer is the true gradient.

Finally, Graves [8] considers a Gaussian prior $\mathcal{N}(\mathcal{W}; \mu_p\mathbf{1}, \sigma_p^2\mathbf{I})$ so that the Kullback-Leibler divergence in $\mathcal{L}_{prior}$ has the closed-form solution

$$\mathcal{L}_{prior} = \sum_i^W \log\frac{\sigma_p}{\sigma_i} + \frac{1}{2\sigma_p^2}\left[(\mu_i - \mu_p)^2 + \sigma_i^2 - \sigma_p^2\right], \tag{3.6}$$

whose derivatives wrt σ_i and μ_i are trivial to calculate.

Putting everything together we have

$$\frac{\partial \mathcal{L}}{\partial \mu_i} \approx \frac{\mu_i - \mu_p}{\sigma_p^2} + \sum_{\mathcal{X}\in\mathcal{D}} \frac{1}{T}\sum_{k=1}^{T} \frac{\partial \log p(\mathcal{X} \mid \mathcal{W}^{(k)})}{\partial w_i} \tag{3.7}$$

$$\frac{\partial \mathcal{L}}{\partial \sigma_i^2} \approx \frac{1}{2}\left(\frac{1}{\sigma_p^2} - \frac{1}{\sigma_i^2}\right) + \sum_{\mathcal{X}\in\mathcal{D}} \frac{1}{T}\sum_{k=1}^{T} \left[\frac{\partial \log p(\mathcal{X} \mid \mathcal{W}^{(k)})}{\partial w_i}\right]^2, \tag{3.8}$$

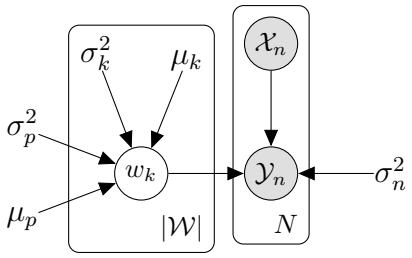


Figure 3.3: PGM representation of the model underlying the practical VI method. The observed output $\mathcal{Y}_n$ is a noisy observation of the model output for the input $\mathcal{X}_n$ with the variance noise determined by the fixed parameter σ_n^2. The constant values $\{\mu_p, \sigma_p^2\}$ govern the Gaussian prior distributions over the weights, while $\{\mu_k, \sigma_k^2\}$ their posteriors.

where $\{w_i\}_{i=0}^T$ are the MC samples, $\mathcal{X}$ are the data points, i.e., input, target pairs, and we optimise the objective (3.2) with a gradient descent method $\Psi_{m+1} = \Psi_m - k\frac{\partial \mathcal{L}}{\partial \Psi_m}$. The Probabilistic Graphical Model (PGM) of the BNN underlying the method outlined is depicted in Figure 3.3.

Observing (3.8) we note that this parameterisation may cause σ_i to assume negative values, what is nonsense given that variances can only be non-negative. Thus, external constraints must be imposed to the optimisation.

When using mini-batch optimisation such that $\mathcal{D} = \{\mathcal{D}_j \,|\, 1 \leq j \leq M\}$, it is important to scale the complexity cost $\mathcal{L}_{prior}$ in the objective accordingly. The equation (3.6) accounts for the whole data set, so naively computing the loss $\mathcal{L}_j$ in (3.2) M times, will lead to accounting M times for the complexity loss $\mathcal{L}_{prior}$ instead of one. The $\mathcal{L}_{j_{prior}}$ terms should then be weighted so that $\mathcal{L}_{prior} = B_j \mathcal{L}_{j_{prior}}$. Although, uniformly distributed weights $B_j = 1/M$ seems a natural choice, there are different ways of distributing them as long as $\sum_{j=1}^M B_j = 1$. In [13], the authors propose $B_j = 2^{M-j}/(2^M - 1)$. During the first iterations, the complexity cost dominates, and at later mini-batches, after more data is seen, the data likelihood cost $\mathcal{L}_{j_{data}}$ progressively gains more importance.

We summarize the resulting algorithm for optimising a BNN in Algorithm 1. The case we illustrate is for a diagonal Gaussian variational posterior with parameters $\Psi = \{\boldsymbol{\mu}, \boldsymbol{\sigma}^2\}$ and centred Gaussian prior with diagonal covariance matrix $\sigma_p^2\mathbf{I}$, trained with a mini-batch of size 1 and uniformly distributed weighting of the complexity term $\mathcal{L}_{prior}$ across the mini-batches.

3.3.3 Bayes by Backprop

The main contribution from [13] certainly is not the complexity cost weighting scheme for the mini-batches. Actually, it is an alternative reparameterisation that gives **unbiased** gradient estimators, and is not restricted to Gaussian distributions

Algorithm 1: Practical VI

while not converged **do**
  $\mathbf{w} \leftarrow \boldsymbol{\mu} + \sigma \odot \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  Randomly sample a data example $\mathcal{X}_i$
  $\mathbf{g} \leftarrow -\nabla \log p(\mathcal{X}_i|\mathbf{w})$
  $\Delta\boldsymbol{\mu} \leftarrow (\boldsymbol{\mu} - \mu_p \mathbf{1})/(N\sigma_p^2) + \mathbf{g}$
  $\Delta\boldsymbol{\sigma}^2 \leftarrow (\boldsymbol{\sigma}^2 - \sigma_p^2 \mathbf{1})/\left(N\sigma_p^2 \boldsymbol{\sigma}^2\right) + (\mathbf{g} \odot \mathbf{g})$
  $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} - k\Delta\boldsymbol{\mu}$
  $\boldsymbol{\sigma^2} \leftarrow \boldsymbol{\sigma^2} - k\Delta\boldsymbol{\sigma^2}$
end while

nor needs to employ the GGN approximation to avoid second-order derivatives, which was the reason behind the introduction of bias. Instead of using the identities (3.3,3.4), it relies on the reparameterisation trick (Section A.1) and considers a the variational posterior $q(w; \Psi)$ and a cost function $h(w; \Psi)$ dependent on the parameters Ψ (the KL term in the loss also directly on Ψ):

$$\nabla_{\Psi} \mathbb{E}_{q(w;\Psi)} \left[h(w; \Psi)\right] = \mathbb{E}_{p(\epsilon)} \left[\frac{\partial h(w; \Psi)}{\partial w} \frac{\partial w}{\partial \Psi} + \frac{\partial h(w; \Psi)}{\partial \Psi}\right], \tag{3.9}$$

where, as before, $w = g(\epsilon; \Psi)$, with $g(\cdot; \Psi)$ an smooth invertible deterministic transformation, and ϵ a base random variable. We recall once more that in our case $f = \log p(\mathcal{D} \,|\, \mathcal{W})$, $\Psi = \{\boldsymbol{\mu}, \boldsymbol{\sigma}^2\}$ and $q(w; \Psi)$ is the product of independent univariate Gaussians (3.1), though this approach would still work for non-Gaussian distributions that can be recast as a transformation $g(\epsilon; \cdot)$ of base distribution $p(\epsilon)$.

A convenient choice is $g(\boldsymbol{\epsilon}; \Psi) = \boldsymbol{\mu} + \boldsymbol{\Sigma}\boldsymbol{\epsilon}$ where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, which boils down to $\boldsymbol{\mu} + \boldsymbol{\sigma} \odot \epsilon$ for the uncorrelated Gaussian case, i.e., diagonal covariance matrix $\boldsymbol{\Sigma}$. Also, instead of imposing explicit constraints to avoid the σ_i assuming negative values, the authors [13] suggest the simple pointwise parameterisation $\sigma_i = \log(1 + \exp \rho_i)$, known as *softplus* transform. Applying (3.9) to our choice of $g(\epsilon; \Psi)$ gives

$$\frac{\partial \mathcal{L}}{\partial \mu_i} = \frac{\partial f(\mathcal{W}, \Psi)}{\partial w_i} + \frac{\partial f(\mathcal{W}, \Psi)}{\partial \mu_i} \tag{3.10}$$

$$\frac{\partial \mathcal{L}}{\partial \rho_i} = \frac{\partial f(\mathcal{W}, \Psi)}{\partial w_i} \frac{\epsilon}{1 + \exp(-\rho_i)} + \frac{\partial f(\mathcal{W}, \Psi)}{\partial \rho_i} \tag{3.11}$$

This apparently simple modification allows the direct computation of the gradients wrt $\mathcal{W}$ and Ψ in the computational graph just as any other deterministic node, illustrated in Figure 3.4. Automatic differentiation tools available in common frameworks handle this transparently, it constructs the graph from the definition of the forward pass of the model and the backward pass is automatically configured just as with deterministic networks. The only implementation difference being the

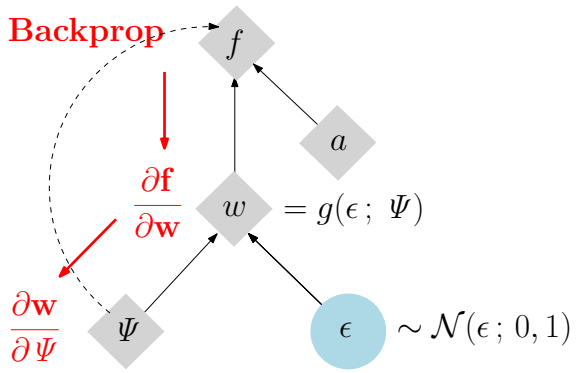


Figure 3.4: Computational graph after the reparameterisation trick. The blue round node is a random node, while the gray rhombus nodes are deterministic. Black arrows represent the forward pass of the model and the red ones (part of) the backpropagation path. The black dashed line indicates the path for the computation of the the KL divergence, that takes the distribution parameters Ψ as input. Note that thanks to the reparameterisation trick the node w is no longer random and so we can compute its gradient as usual.

reparameterisation $w = g(\epsilon; \Psi)$ in the network definition and specifying $\Psi = \{\boldsymbol{\mu}, \boldsymbol{\rho}\}$ as the learnable parameters.

Additionally, the authors [13] propose computing the KL numerically with the samples drawn from the posterior, instead of analytically. The advantage of doing as such is allowing many more combinations of prior and variational posterior families. Even though we now have one more approximation in the system, more expressive priors can be used, i.e., non-Gaussian, which potentially leads to better results. In light of this change, we drop (3.6) so (3.2) actually writes

$$\mathcal{L} \approx \sum_{i=1}^{T} -\log p(\mathcal{D} \,|\, \mathcal{W}^{(i)}) + \log q(\mathcal{W}^{(i)}; \Psi) - \log p(\mathcal{W}^{(i)}), \tag{3.12}$$

where $\mathcal{W}^{(i)}$ denotes the i-th out of T Monte Carlo samples drawn from the variational posterior $q(\mathcal{W}; \Psi)$.

The final graphical model for BBB is similar to Figure 3.3, the difference residing in the set of constant parameters that determine the prior distribution. While in Figure 3.3 it is $\{\mu_p, \sigma^2 p\}$, which defines univariate Gaussian priors, BBB handles other non-conjugate priors, such as a mixture of Gaussians. We summarize the resulting algorithm for optimising a BNN in Algorithm 2. The case we illustrate is for a diagonal Gaussian variational posterior with parameters $\Psi = \{\boldsymbol{\mu}, \boldsymbol{\rho}\}$, trained with a mini-batch of size 1 with non-uniformly distributed weighting of the complexity term $\mathcal{L}_{prior}$ across the mini-batches.

Even though the gradient estimators are now unbiased, the MC predictive **log**-likelihood estimator still is biased because a non-linear function, i.e., the log, warps

Algorithm 2: Bayes by Backprop

while not converged **do**
  $\mathbf{w} \leftarrow \boldsymbol{\mu} + \log(1 + \exp(\boldsymbol{\rho})) \odot \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  Randomly sample a data example $\mathcal{X}_i$
  $i \leftarrow (i + 1)$ mod N
  $\pi_i \leftarrow 2^{N-i}/2^{N-1}$
  for $\mathbf{s} \in \{\mathbf{w}, \boldsymbol{\mu}, \boldsymbol{\rho}\}$ **do**
    $\mathbf{g_s} \leftarrow -\nabla_{\mathbf{s}} \log p(\mathcal{X}_i|\mathbf{w}) + \pi_i \left(\nabla_{\mathbf{s}} \log q(\mathbf{w}; \Psi) - \nabla_{\mathbf{s}} \log p(\mathbf{w})\right)$
  end for
  $\Delta\boldsymbol{\mu} \leftarrow \mathbf{g_w} + \mathbf{g}_{\boldsymbol{\mu}}$
  $\Delta\boldsymbol{\rho} \leftarrow \mathbf{g_w} \odot \boldsymbol{\epsilon}/(1 + \exp(-\boldsymbol{\rho})) + \mathbf{g}_{\boldsymbol{\rho}}$
  $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} - k\Delta\boldsymbol{\mu}$
  $\boldsymbol{\rho} \leftarrow \boldsymbol{\rho} - k\Delta\boldsymbol{\rho}$
end while

the expected value. This will in general be true for all MC estimators and can be lessened by increasing the number of samples.

3.4 Probabilistic Backprop

Probabilistic Backpropagation (PBP) [14] solves the same problem as BBB but in a rather very different manner. While the algorithm of the previous section relies on optimising the ELBO for the VI equation, PBP employs Assumend Density Filtering (ADF) and Expectation Propagation (EP), discussed in Sections 2.2.2 and 2.2.3, respectively. The result is a parameter-free (not even learning rate) fully Bayesian method that has forward and backward phases as in common backpropagation, but instead of performing gradient descent in the parameter space, it incorporates infomation about the new data points into the posterior approximation at each iteration. Although, another EP-based method had been proposed before [9], it focused on binary weights and its continuous extension performed poorly, not estimating the posterior variance.

In the year following its publication, other researchers [60] developed a variant for binary and multi-class classification problems. In [36], the authors adopt the PBP framework to propose an online algorithm that models the correlations within the weights of the network with a matrix variate Gaussian distribution. However, here we shall focus solely on its original formulation for regression tasks since this already is enough work. PBP does not use the usual reverse mode automatic differentiation and requires non-trivial custom implementations, which is its major drawback and the reason why it has not seen widespread adoption. We start this section anticipating the reader that this is the most technically difficult section in the dissertation.

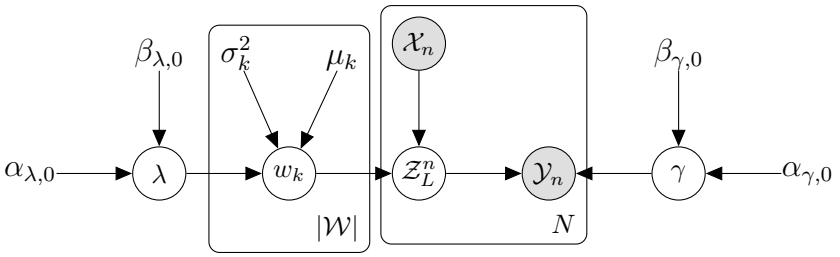


Figure 3.5: PGM representation of the PBP model. The observed output $\mathcal{Y}_n$ is a noisy observation of the model output $\mathcal{Z}_L^n$ for the input $\mathcal{X}_n$. The hyper-parameter λ govern the precision of the Gaussian prior distributions over the weights, while γ the precision noise of Gaussian observation model.

Similar to the previous method, PBP assumes independence among the network weights and the existence of additive Gaussian noise $\mathcal{N}(\epsilon \,|\, 0, \gamma^{-1})$ with precision γ corrupting the observations. Although, specifying the network architecture is not necessary for the other methods in this chapter, since they correctly function with any directed acyclic graph with no or almost none adaptations, the one at hand specialises in fully-connected layers with Rectified Linear Units (ReLU) [61], that is, $\max(0, x)$, as activation function. While modifying the model to conform to a different non-linearity is possible, it requires painstaking mathematical derivations as we glance upon in this section.

The graphical model for PBP is illustrated in Figure 3.5 and its full posterior distribution over the parameters is given by

$$\begin{aligned} p(\mathcal{W}, \gamma, \lambda) &= \frac{p(\mathcal{Y} \,|\, \mathcal{W}, \mathcal{X}, \gamma) p(\mathcal{W} \,|\, \lambda) p(\lambda) p(\gamma)}{p(\mathcal{Y} \,|\, \mathcal{X})} \\ &\propto p(\mathcal{Y} \,|\, \mathcal{W}, \mathcal{X}, \gamma) p(\mathcal{W} \,|\, \lambda) p(\lambda) p(\gamma), \end{aligned} \tag{3.13}$$

where $p(\mathcal{Y} \,|\, \mathcal{X})$ is the model evidence, $p(\mathcal{Y} \,|\, \mathcal{W}, \mathcal{X}, \gamma)$ the observation model defining the likelihood factors, $p(\mathcal{W} \,|\, \lambda)$ the prior distribution over the weights composed of univariate Gaussians with precision λ, that is,

$$p(\mathcal{W} \,|\, \lambda) = \prod_{w \in \mathcal{W}} \mathcal{N}(w \,|\, 0, \lambda^{-1}), \tag{3.14}$$

and $p(\lambda)$ and $p(\gamma)$ are hyper-prior distributions over the precision hyper-parameters of the likelihood and weight prior respectively. We specify Gamma distributions $\mathrm{Ga}(z \,|\, \alpha, \beta)$, given by

$$p(z \,|\, \alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} z^{\alpha - 1} \exp(-\beta z), \tag{3.15}$$

for both hyper-priors. The Gamma is the conjugate prior for the Gaussian distri-

bution with known mean and unknown precision parameter. Conjugate priors are frequently used because they lead to closed-form posteriors, avoiding the possible need to do numerical integration. Thus, it is a matter of mathematical convenience. We now demonstrate this fact for this particular case:

$$\begin{aligned}
p(\mathcal{W} \,|\, \lambda, \alpha_{\lambda,0}, \beta_{\lambda,0}) &\propto p(\lambda \,|\, \alpha_{\lambda,0}, \beta_{\lambda,0}) \prod_{w \in \mathcal{W}} \mathcal{N}(w \,|\, 0, \lambda^{-1}) \\
&\propto \left[\frac{\beta_{\lambda,0}^{\alpha_{\lambda,0}}}{\Gamma(\alpha_{\lambda,0})} \lambda^{\alpha_{\lambda,0}-1} \exp\left(-\beta_{\lambda,0}\lambda\right) \right] \left(\frac{\lambda}{2\pi} \right)^{\frac{|\mathcal{W}|}{2}} \exp\left(-\frac{\lambda}{2} \sum_{w \in \mathcal{W}} w^2 \right) \\
&\propto \lambda^{\left(\alpha_{\lambda,0} + \frac{|\mathcal{W}|}{2}\right)-1} \exp\left[-\lambda \left(\beta_{\lambda,0} + \frac{1}{2} \sum_{w \in \mathcal{W}} w^2 \right) \right] \\
&= \mathrm{Ga}\left(\lambda \,\middle|\, \alpha_{\lambda,0} + \frac{|\mathcal{W}|}{2}, \beta_{\lambda,0} + \frac{1}{2} \sum_{w \in \mathcal{W}} w^2 \right), \qquad (3.16)
\end{aligned}$$

where in the last equality we conclude it is a Gamma by noticing the same functional form of (3.15) in the formula for the posterior.

From (3.16) we further gain insight on the influence of the parameters $\alpha_{\lambda,0}$ and $\beta_{\lambda,0}$ on the posterior. It is as if we had already collected $2\alpha_{\lambda,0}$ observations with sample variance $\beta_{\lambda,0}/\alpha_{\lambda,0}$. We thus choose these parameters such that they impose a weak prior, not affecting the posterior distribution. Exactly the same reasoning is valid for the hyper-prior on γ.

Another distribution that we will shortly resort to arises from the following posterior probability over a single weight variable

$$\begin{aligned}
p(w \,|\, \beta_{\lambda,0}, \alpha_{\lambda,0}) &= \int_0^\infty p(w \,|\, \lambda) p(\lambda \,|\, \beta_{\lambda,0}, \alpha_{\lambda,0}) d\lambda \\
&= \int_0^\infty \left[\frac{\beta_{\lambda,0}^{\alpha_{\lambda,0}}}{\Gamma(\alpha_{\lambda,0})} \lambda^{\alpha_{\lambda,0}-1} \exp\left(-\lambda\beta_{\lambda,0}\right) \right] \left[\left(\frac{\lambda}{2\pi} \right)^{\frac{1}{2}} \exp\left(-\lambda \frac{w^2}{2} \right) \right] d\lambda \\
&= (2\pi)^{-\frac{1}{2}} \frac{\beta_{\lambda,0}^{\alpha_{\lambda,0}}}{\Gamma(\alpha_{\lambda,0})} \int_0^\infty \lambda^{(\alpha_{\lambda,0}+\frac{1}{2})-1} \exp\left[-\lambda \left(\beta_{\lambda,0} + \frac{w^2}{2} \right) \right] d\lambda \\
&= (2\pi)^{-\frac{1}{2}} \frac{\Gamma\left(\alpha_{\lambda,0} + \frac{1}{2}\right)}{\Gamma(\alpha_{\lambda,0})} \beta_{\lambda,0}^{\alpha_{\lambda,0}} \left(\beta_{\lambda,0} + \frac{w^2}{2} \right)^{-\left(\alpha_{\lambda,0}+\frac{1}{2}\right)} \\
&\qquad \times \int_0^\infty \mathrm{Ga}\left(\lambda \,\middle|\, \alpha_{\lambda,0} + \frac{1}{2}, \beta_{\lambda,0} + \frac{w^2}{2} \right) d\lambda \\
&= (2\pi)^{-\frac{1}{2}} \frac{\Gamma\left(\alpha_{\lambda,0} + 1/2\right)}{\Gamma(\alpha_{\lambda,0})} \beta_{\lambda,0}^{\alpha_{\lambda,0}} \left(\beta_{\lambda,0} + \frac{w^2}{2} \right)^{-\left(\alpha_{\lambda,0}+\frac{1}{2}\right)} \\
&= \frac{\Gamma\left(\alpha_{\lambda,0} + 1/2\right)}{\Gamma(\alpha_{\lambda,0})} (2\pi\beta_\lambda)^{-\frac{1}{2}} \left(1 + \frac{w^2}{2\beta_{\lambda,0}} \right)^{-\left(\alpha_{\lambda,0}+\frac{1}{2}\right)}, \qquad (3.17)
\end{aligned}$$

and if we compare it to the location-scale family for the Student's t-distribution

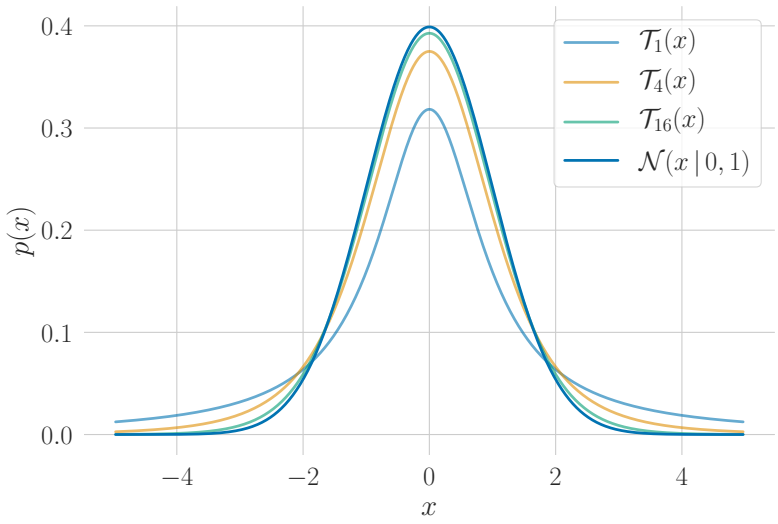


Figure 3.6: PDF of the standard Student's t-distribution $\mathcal{T}_\nu(x)$ for different values of the parameter ν. The higher the value of ν, the closer (in the KL divergence sense) the distribution becomes to the Gaussian distribution. The standard family member has the location μ and inverse scale λ parameters of (3.18) equal to 0 and 1, respectively.

(parameterised in terms of the inverse scaling parameter λ)

$$\mathcal{T}_\nu(x \mid \mu, \lambda) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\Gamma\left(\frac{\nu}{2}\right)} \left(\frac{\lambda}{\pi\nu}\right)^{\frac{1}{2}} \left(1 + \frac{\lambda(x-\mu)^2}{\nu}\right)^{-\frac{\nu+1}{2}}, \tag{3.18}$$

whose mean and variance are $\mathbb{E}[X] = \mu$, and $\mathrm{Var}(X) = \frac{1}{\lambda}\frac{\nu}{\nu-2}$, respectively. We conclude that $p(w \mid \beta_{\lambda,0}, \alpha_{\lambda,0})$ is a Student's t-distribution with $\mathcal{T}_{2\alpha_{\lambda,0}}(w \mid 0, \frac{\alpha_{\lambda,0}}{\beta_{\lambda,0}})$. whose standard PDF is shown in Figure 3.6.

PBP uses EP and ADF (Sections 2.2.3 and 2.2.2 respectively) to update the parameters $\mathcal{W}$, α_γ, β_γ, α_λ and β_λ of the approximating distribution

$$q(\mathcal{W}, \lambda, \gamma) = \left[\prod_{i=1}^{|\mathcal{W}|} \mathcal{N}(w_i \mid \mu_i, \sigma_i^2)\right] \mathrm{Ga}(\lambda \mid \alpha_\lambda, \beta_\lambda)\, \mathrm{Ga}(\gamma \mid \alpha_\gamma, \beta_\gamma), \tag{3.19}$$

by cycling through the factors in (3.13) and including them one at a time. Thus, the total number of factors is the number of data points plus the (hyper-)priors, i.e., $|\mathcal{W}|$ for the weights and 2 for the precisions.

Since EP requires storing the approximate factors to compute the cavity distributions, it does not scale well with data. Its memory consumption grows linearly with the data set size. Thus, instead of performing EP updates for the likelihood factors, PBP repeatedly employs ADF multiple times, that is, instead of going through each data point only once, it incorporates the same factors N times. Although, computationally more efficient, this approach has the risk of underestimating the parameter

posterior variance. We are artificially observing more data, which in the limit of infinite data leads to the collapse of the posterior distribution onto the MLE as we assume the data points are (conditionally) iid. However, this is clearly not the case when repeating the observations. Thus, PBP should not run for many epochs. The authors [14] advise fewer than 100 and in our case study (Section **??**) we run it for 40 epochs. Nevertheless, PBP is specifically designed for large data sets so this restriction do not matter much in practice. Yet, it is important to keep it in mind.

The models we analyse here and those employed in the original work have rather small sizes according to the current standards, i.e., 1 hidden layer with 50 units, so running EP updates is still feasible. Indeed, it is what the authors [14] propose. In modern networks, which commonly contain hundreds of thousands of parameters, EP once again becomes a problem and ADF is the way to go.

The ADF update consists in including the true factor $f_i(\mathcal{W}, \lambda, \gamma)$ into the current approximation $q(\mathcal{W}, \lambda, \gamma)$, such that the updated approximation is $K^{-1} f(\mathcal{W}, \lambda, \gamma) \times q(\mathcal{W}, \lambda, \gamma)$, where K^{-1} is a normalisation constant that assures it remains a proper probability distribution. This step usually causes the distribution to shift and no longer belong to the desired functional form. Then, to maintain the approximation manageable, we project it back to same the distribution class we had before the inclusion of the true factor, namely we minimise the KL divergence $D_{KL}\left(K^{-1} f(\mathcal{W}, \lambda, \gamma) q(\mathcal{W}, \lambda, \gamma) \| q_{new}(\mathcal{W}, \lambda, \gamma)\right)$ w.r.t. $\mathcal{W}, \lambda, \gamma$, the parameters of the new distribution q_{new}. As already shown in (2.31), this is equivalent to matching the moments of both distributions, and each update consists of an iterative deterministic procedure so there is no learning rate to modulate the step size as the other methods we discuss.

At the beginning, we have no information, so unless we have prior domain knowledge we initialise the parameters so that q is effectively uniform. This amounts to setting $\alpha_\lambda = \alpha_\gamma = 0$, $\beta_\lambda = \beta_\gamma = 1$, and $\mu = 0$, $\sigma^2 = \infty$ for every weight w.

The remainder of the section is split into different subsections explaining how each type of factor is included into the model.

3.4.1 Incorporating the hyper-priors

The first factors to incorporate into the approximation are the priors over γ and λ. As shown in (3.16), the product of the prior precision Gamma and the Normal distribution results in a distribution with the same functional form as Gamma. This is exactly the case for (3.13).

$$\begin{aligned} q_{new}(\mathcal{W}, \lambda, \gamma) &\propto \left[\lambda^{\alpha_\lambda - 1} \exp\left(-\lambda \beta_\lambda\right)\right] \left[\lambda^{\alpha_{\lambda,0} - 1} \exp\left(-\lambda \beta_{\lambda,0}\right)\right] \\ &\propto \lambda^{(\alpha_\lambda + \alpha_{\lambda,0} - 1) - 1} \exp\left(-\lambda \left(\beta_{\lambda,0} + \beta_\lambda\right)\right). \end{aligned} \tag{3.20}$$

Thus, including the Gamma prior factors into q amounts to incrementing the values of the parameters γ and λ by

$$\alpha_{\gamma,\text{new}} = \alpha_\gamma + \alpha_{\gamma,0} - 1 = \alpha_{\gamma,0} \ ,\beta_{\gamma,\text{new}} \qquad = \beta_\gamma + \beta_{\gamma,0} = \beta_{\gamma,0} \ , \tag{3.21}$$

and since there is no approximation, and, hence no loss of information, they need only to be included once.

3.4.2 Incorporating the priors on the weights

Next, we incorporate the priors over the weights $w \in \mathcal{W}$. The unnormalized shifted distribution after the inclusion of one such factor is $q(\mathcal{W}, \gamma, \lambda)\mathcal{N}(w_j \,|\, 0, \lambda^{-1})$, and the normalisation constant is

$$\begin{aligned} K &= \int q(\mathcal{W}, \gamma, \lambda)\mathcal{N}(w_j \,|\, 0, \lambda^{-1}) d\mathcal{W} d\gamma d\lambda \qquad (3.22) \\ &= \int \left[\prod_{i=1}^{|\mathcal{W}|} \mathcal{N}(w_i \,|\, \mu_i, \sigma_i^2)\right] \text{Ga}\left(\lambda \,|\, \alpha_\lambda, \beta_\lambda\right) \text{Ga}\left(\gamma \,|\, \alpha_\gamma, \beta_\gamma\right) \mathcal{N}(w_j \,|\, 0, \lambda^{-1}) d\mathcal{W} d\gamma d\lambda \\ &= \int \mathcal{N}(w_j \,|\, \mu_j, \sigma_j^2) \left[\int \mathcal{N}(w_j \,|\, 0, \lambda^{-1}) \text{Ga}\left(\lambda \,|\, \alpha_\lambda, \beta_\lambda\right) d\lambda\right] dw_j \\ &= \int \mathcal{N}(w_j \,|\, \mu_j, \sigma_j^2) \mathcal{T}_{2\alpha_\lambda}(w_j \,|\, 0, \beta_\lambda/\alpha_\lambda) dw_j, \qquad (3.23) \end{aligned}$$

where we have used the result demonstrated in (3.17), that the integral of the product of a Gamma and a Gaussian distributions is the t-Student's distribution defined in (3.18). We continue the computation of K by approximating $\mathcal{T}_{2\alpha_\lambda}(w_j \,|\, 0, \beta_\lambda/\alpha_\lambda)$ with a Gaussian with same mean and variance, what as we saw in Figure 3.6 is within reason for high enough ν. Thus, continuing the calculation of K:

$$\begin{aligned} K &\approx \int \mathcal{N}(w_j \,|\, \mu_j, \sigma_j^2) \mathcal{N}(w_j \,|\, 0, \beta_\lambda/(\alpha_\lambda - 1)) dw_j \\ &= \int \mathcal{N}\left(\mu_j \,\middle|\, 0, \sigma_j^2 + \frac{\beta_\lambda}{\alpha_\lambda - 1}\right) \mathcal{N}\left(w_j \,\middle|\, \frac{\lambda(\alpha_\lambda - 1)}{\beta\lambda + \alpha - 1}\frac{\mu}{\sigma^2}, \frac{\lambda(\alpha_\lambda - 1)}{\beta\lambda + \alpha - 1}\right) dw_j \\ &= \mathcal{N}\left(\mu_j \,\middle|\, 0, \sigma_j^2 + \frac{\beta_\lambda}{\alpha_\lambda - 1}\right) \int \mathcal{N}\left(w_j \,\middle|\, \frac{\lambda(\alpha_\lambda - 1)}{\beta\lambda + \alpha - 1}\frac{\mu}{\sigma^2}, \frac{\lambda(\alpha_\lambda - 1)}{\beta\lambda + \alpha - 1}\right) dw_j \\ &= \mathcal{N}\left(\mu_j \,\middle|\, 0, \sigma_j^2 + \frac{\beta_\lambda}{\alpha_\lambda - 1}\right), \qquad (3.24) \end{aligned}$$

where we resorted to the fact that the product of two Gaussians is also a Gaussian and is given by

$$\mathcal{N}(w \,|\, \mu_1, \sigma_1^2)\mathcal{N}(w \,|\, \mu_2, \sigma_2^2) = \mathcal{N}(\mu_1 \,|\, \mu_2, \sigma_1^2 + \sigma_2^2)\mathcal{N}(w \,|\, \mu, \sigma^2) \tag{3.25}$$

with $\sigma^2 = \left(\sigma_1^{-2} + \sigma_2^{-2}\right)^{-1}$ and $\mu = \sigma^2\left(\mu_1\sigma_1^2 + \mu_2\sigma_2^2\right)$.

Update equations for α_λ and β_λ

Updating the posterior approximation means matching its moments with those of the shifted distribution $s = K^{-1}q(\mathcal{W}, \gamma, \lambda)\mathcal{N}(w_j \,|\, 0, \lambda^{-1})$. However, the sufficient statistics for λ does not have closed form so we revise its parameters β_λ and α_λ by matching only its first and second moments, which still produces good results [39].

Let us now derive those update formulas. We start by noting that K in (3.24) is a function of μ_j, σ_j^2, β_λ, and α_λ, and make the dependency in the two latter terms explicit by writing $K(\beta_\lambda, \alpha_\lambda)$. Additionally, for brevity we denote the integrand in (3.22) as $f(\lambda)\mathrm{Ga}(\lambda \,|\, \alpha_\lambda, \beta_\lambda)$ and compute

$$\begin{aligned}
\mathbb{E}_q\left[\lambda\right] &= \frac{1}{K(\beta_\lambda, \alpha_\lambda)} \int \lambda f(\lambda)\mathrm{Ga}(\lambda \,|\, \alpha_\lambda, \beta_\lambda) d\lambda \\
&= \frac{1}{K(\beta_\lambda, \alpha_\lambda)} \int \frac{\alpha_\lambda}{\beta_\lambda} f(\lambda)\mathrm{Ga}(\lambda \,|\, \alpha_\lambda + 1, \beta_\lambda) d\lambda \\
&= \frac{1}{K(\beta_\lambda, \alpha_\lambda)} \left[\frac{\alpha_\lambda}{\beta_\lambda} K(\alpha_\lambda + 1, \beta_\lambda)\right] \\
&= \frac{K(\alpha_\lambda + 1, \beta_\lambda)\alpha_\lambda}{K(\alpha_\lambda, \beta_\lambda)\beta_\lambda}.
\end{aligned} \tag{3.26}$$

Similarly, we obtain for the second moment

$$\mathbb{E}_q\left[\lambda^2\right] = \frac{K(\alpha_\lambda + 2, \beta_\lambda)\alpha_\lambda\left(\alpha_\lambda + 1\right)}{K(\alpha_\lambda, \beta_\lambda)\beta_\lambda^2}. \tag{3.27}$$

Recalling the mean and variance formulas for the Gamma distribution (3.15), we equate them to the above expressions to obtain

$$\frac{\alpha_{\lambda,\mathrm{new}}}{\beta_{\lambda,\mathrm{new}}} = \frac{K(\alpha_\lambda + 1, \beta_\lambda)\alpha_\lambda}{K(\beta_\lambda, \alpha_\lambda)\beta_\lambda} \tag{3.28}$$

$$\frac{\alpha_{\lambda,\mathrm{new}}}{\beta_{\lambda,\mathrm{new}}^2} = \frac{K(\alpha_\lambda + 2, \beta_\lambda)\alpha_\lambda(\alpha_\lambda + 1)}{K(\beta_\lambda, \alpha_\lambda)\beta^2} - \left[\frac{K(\alpha_\lambda + 1, \beta_\lambda)\alpha_\lambda}{K(\beta_\lambda, \alpha_\lambda)\beta_\lambda}\right]^2. \tag{3.29}$$

Solving the above equations for $\alpha_{\lambda,\mathrm{new}}$ and $\beta_{\lambda,\mathrm{new}}$, and abbreviating the normalizing coefficients $K_0 \coloneqq K(\alpha_\lambda, \beta_\lambda)$, $K_1 \coloneqq K(\alpha_\lambda + 1, \beta_\lambda)$, and $K_2 \coloneqq K(\alpha_\lambda + 2, \beta_\lambda)$, we finally get

$$\alpha_{\lambda,\mathrm{new}} = \left[K_0 K_2 K_1^{-2}(\alpha_\lambda + 1)\alpha_\lambda^{-1} - 1\right]^{-1} \tag{3.30}$$

$$\beta_{\lambda,\mathrm{new}} = \left[K_2 K_1^{-1}(\alpha_\lambda + 1)\beta_\lambda^{-1} - K_1 K_0^{-1}\alpha_\lambda\beta_\lambda^{-1}\right]^{-1}, \tag{3.31}$$

which are the update equations for the Gamma distribution over the precision pa-

rameter λ.

Update equations for the μ and σ^2

It remains to establish how the mean and variance parameters of a given random weight change when we include its prior distribution into the posterior. The derivation in this section closely follows [62]

We first note that the shifted distribution can be conveniently written as $s := K^{-1} f(w_i)\mathcal{N}(w_i \,|\, \mu_i, \sigma_i^2)$, where $f(w_i)$ comprises of all factors in $q(\mathcal{W}, \gamma, \lambda)\mathcal{N}(w_i \,|\, 0, \gamma^{-1})$ except the $\mathcal{N}(w_i \,|\, \mu_i, \sigma_i^2)$, which we make explicit.

For μ_i, we start from the easily verifiable identity

$$\nabla_{\mu_i} \mathcal{N}(w_i \,|\, \mu_i, \sigma^2) = \sigma_i^{-2}(w_i - \mu_i)\mathcal{N}(w_i \,|\, \mu_i, \sigma^2), \tag{3.32}$$

which we rearrange to

$$w_i \mathcal{N}(w_i \,|\, \mu_i, \sigma^2) = \mu_i \mathcal{N}(w_i \,|\, \mu_i, \sigma^2) + \sigma^2 \nabla_\mu \mathcal{N}(w_i \,|\, \mu_i, \sigma^2).$$

Multiplying on both sides by $K^{-1} f(w_i)$ and integrating over w_i leads to

$$\begin{aligned}
\int w_i K^{-1} f(w_i)\mathcal{N}(w_i \,|\, \mu_i, \sigma^2) dw_i &= \int \mu K^{-1} f(w_i)\mathcal{N}(w_i \,|\, \mu_i, \sigma^2) dw_i \\
&\quad + \int \sigma^2 K^{-1} f(w_i) \nabla_\mu \mathcal{N}(w_i \,|\, \mu_i, \sigma^2) dw_i \\
\mathbb{E}_s[w_i] &= \mu + \sigma^2 K^{-1} \left[\nabla_\mu \int f(w_i)\mathcal{N}(w_i \,|\, \mu_i, \sigma^2) dw_i \right] \\
&= \mu + \sigma^2 K^{-1} \nabla_\mu K \\
&= \mu + \sigma^2 \nabla_\mu \log K.
\end{aligned} \tag{3.33}$$

Since the first moment for the to-be-updated distribution $\mathcal{N}(w_i \,|\, \mu_i, \sigma^2)$ is μ_i, the update formula is

$$\mu_{i,\text{new}} = \mu + \sigma^2 \nabla_\mu \log K. \tag{3.34}$$

Through a similar identity for the derivative w.r.t σ_i^2:

$$\nabla_{\sigma_i^2} \mathcal{N}(w_i \,|\, \mu_i, \sigma^2) = \frac{\sigma_i^{-2}}{2} \left(-1 + \sigma_i^{-2}(w_i - \mu_i)^2\right) \mathcal{N}(w_i \,|\, \mu_i, \sigma^2), \tag{3.35}$$

and following exactly the same procedure as before for μ_i, we arrive at $\mathbb{E}_s[w_i^2] = \sigma_i^2 + 2\left(\sigma_i^2\right)^2 \nabla_{\sigma_i^2} \log K$. Then, the variance of the shifted distribution is

$$\text{Var}(w_i) = \mathbb{E}_s\left[w_i^2\right] - \left(\mathbb{E}_s[w_i]\right)^2 = \sigma_i^2 - \left(\sigma_i^2\right)^2 \left[\left(\nabla_\mu \log K\right)^2 - 2\nabla_{\sigma_i^2} \log K \right]. \tag{3.36}$$

From this, we establish the update for the variance of the Normally distributed weight as

$$\sigma^2_{i,\text{new}} = \sigma_i^2 - \left(\sigma_i^2\right)^2 \left[\left(\nabla_\mu \log K\right)^2 - 2\nabla_{\sigma_i^2} \log K\right]. \tag{3.37}$$

Although we derived rules for performing ADF, that is, only including the individual true factors of the model, without ever removing the approximating factors to be update, adapting them to EP is simple. The two key differences are:

1. Keep track of the parameters for each individual approximating factor;
2. Before the update, remove from the posterior the approximating factor corresponding to the true factor that will be incorporated (cavity distribution), effectively this means subtracting their contributions from the parameters of the posterior.

It remains only to determine which are the approximating factors for the prior factors $\mathcal{N}(w_i \,|\, 0, \lambda^{-1})$. The authors [14] take them to be $\mathcal{N}(w_i \,|\, \tilde{\mu}_i, \tilde{\sigma}_i^2)\text{Ga}(\lambda \,|\, \tilde{\alpha}_\lambda, \tilde{\beta}_\lambda)$

3.4.3 Incorporating the likelihood factors

In order to incorporate the information coming from a data point, we pass it forward through the network. Assuming the model to be a fully-connected multi-layer network, at each layer following the input, PBP approximates the distribution of the resulting activations with a Gaussian distribution with same mean and variance, such that the input to the next layer is again Gaussian. At the last layer, we obtain the distribution of the output $\mathcal{Y}_i$ given $\mathcal{X}_i$, to which we further apply the observation model, i.e., additive Gaussian noise with precision γ, which gives us $p(\mathcal{Y}_i \,|\, \mathcal{X}_i, \mathcal{W}, \gamma) = \mathcal{N}(\mathcal{Y}_i \,|\, f(\mathcal{X}_i, \mathcal{W}), \gamma^{-1})$. The likelihood factor is then included into the posterior approximation as usual: we shift the posterior by multiplying it by such factor, compute the first and second moments of the resulting distribution, and update the parameters to obtain these moments.

Note that in the derivation of the update formulas (3.31, 3.30, 3.34, 3.37) we have not assumed any specific format for the factors being included. The same equations can be used once again for the likelihood factors, the sole change being what the normalising constant K is. Consequently, in what follows we unveil the expression for K in this case.

The normalising factor

We consider a network with L layers and V_l units on each layer l, taking in vector-shaped inputs $\mathbf{x}_i$. Thus, the output $\mathbf{z}_l$ of each layer can be arranged into a vector, and

the weights between two consecutive layers into a weight matrix $\mathbf{W}_l$ with dimensions $V_l \times (V_{l-1} + 1)$, where the $+1$ stems from the inclusion of a bias term. The pre-activation of a layer l is given by $\mathbf{a}_l = \mathbf{W}\mathbf{z}_{l-1}/\sqrt{V_{l-1}+1}$, and for all except the last layer, this gets transformed according to the non-linear mapping $\max(a, 0)$, known as ReLU [61].

We make the simplifying assumption that the output z_L of the network at the last layer L is distributed as a Gaussian and proceed to compute the normalising constant K of the associated shifted distribution as

$$\begin{aligned}
K &= \int q(\mathcal{W}, \gamma, \lambda)\mathcal{N}(\mathbf{y}_i \,|\, f(\mathcal{X}_i, \mathcal{W}), \gamma^{-1})d\mathcal{W}d\gamma d\lambda \\
&\approx \int q(\mathcal{W}, \gamma, \lambda)\mathcal{N}(\mathbf{y}_i \,|\, \mathbf{z}_L, \gamma^{-1})\mathcal{N}(\mathbf{z}_L \,|\, \mu_{\mathbf{z}_L}, \sigma^2_{\mathbf{z}_L})d\mathcal{W}d\mathbf{z}_L d\gamma d\lambda \\
&= \int \mathrm{Ga}\,(\gamma \,|\, \alpha_\gamma, \beta_\gamma)\,\mathcal{N}(\mathbf{y}_i \,|\, \mathbf{z}_L, \gamma^{-1})\mathcal{N}(\mathbf{z}_L \,|\, \mu_{\mathbf{z}_L}, \sigma^2_{\mathbf{z}_L})d\mathbf{z}_L d\gamma \\
&= \int \mathcal{T}_{2\alpha_\gamma}(\mathbf{y}_i \,|\,, \mathbf{z}_L, \beta_\gamma/\alpha_\gamma)\mathcal{N}(\mathbf{z}_L \,|\, \mu_{\mathbf{z}_L}, \sigma^2_{\mathbf{z}_L})d\mathbf{z}_L d\gamma \\
&\approx \int \mathcal{N}(\mathbf{y}_i \,|\,, \mathbf{z}_L, \beta_\gamma/(\alpha_\gamma - 1)\mathcal{N}(\mathbf{z}_L \,|\, \mu_{\mathbf{z}_L}, \sigma^2_{\mathbf{z}_L})d\mathbf{z}_L d\gamma \\
&= \int \mathcal{N}(\mathbf{y}_i \,|\,, \mathbf{z}_L, \beta_\gamma/(\alpha_\gamma - 1)\mathcal{N}(\mathbf{z}_L \,|\, \mu_{\mathbf{z}_L}, \sigma^2_{\mathbf{z}_L})d\mathbf{z}_L d\gamma \\
&= \mathcal{N}(\mathbf{y}_i \,|\, \mu_{\mathbf{z}_L}, \beta_\gamma/(\alpha_\gamma - 1) + \sigma^2_{\mathbf{z}_L}),
\end{aligned} \tag{3.38}$$

where we have followed the same steps and performed the same approximations as in the derivation of (3.24).

Computing the mean $\mu_{\mathbf{z}_L}$ and variance $\sigma^2_{\mathbf{z}_L}$ amounts to propagating the input through the network until the last layer. If we assume that the layer $l-1$ has output $\mathbf{z}_L$ with a diagonal covariance Gaussian distribution with mean and variance $\mu_{\mathbf{z}_{l-1}}$ and $\sigma^2_{\mathbf{z}_{l-1}}$, respectively, we can compute the mean and variance of the pre-activation $\mathbf{z}_l$ at the following layer according to

$$\mu_{\mathbf{a}_l} = \mathbb{E}\left[\mathbf{W}_l\mathbf{z}_{l-1}/\sqrt{V_{l-1}+1}\right] = \mathbf{M}_l\mathbf{z}_{l-1}/\sqrt{V_{l-1}+1} \tag{3.39}$$

$$\begin{aligned}
\sigma^2_{\mathbf{a}_l} &= \mathrm{Var}\left(\mathbf{W}_l\mathbf{z}_{l-1}/\sqrt{V_{l-1}+1}\right) \\
&= \frac{1}{V_{l-1}+1}\left[\left(\mathbb{E}\left[\mathbf{W}_l\right]\right)^2 \mathrm{Var}\left(\mathbf{z}_{l-1}\right) + \mathrm{Var}\left(\mathbf{W}_l\right)\left(\mathbb{E}\left[\mathbf{z}_{l-1}\right]\right)^2 + \mathrm{Var}\left(\mathbf{W}_l\right)\mathrm{Var}\left(\mathbf{z}_{l-1}\right)\right] \\
&= \frac{1}{V_{l-1}+1}\left[\left(\mathbf{M}_l \odot \mathbf{M}_l\right)\sigma^2_{\mathbf{z}_{l-1}} + \mathbf{V}_l\left(\mu_{\mathbf{z}_{l-1}} \odot \mu_{\mathbf{z}_{l-1}}\right) + \mathbf{V}_l\sigma^2_{\mathbf{z}_{l-1}}\right],
\end{aligned} \tag{3.40}$$

where $\mathbf{M}_l$ and $\mathbf{V}_l$ are the mean and variance matrices for the weights in $\mathbf{W}_l$, whose values are determined by the corresponding Gaussian factors of the model.

If the number V_{l-1} of inputs to the layer l is large enough and we further assume the entries of $\mathbf{a}_l$ are independent, we can invoke the Central Limit Theorem and claim

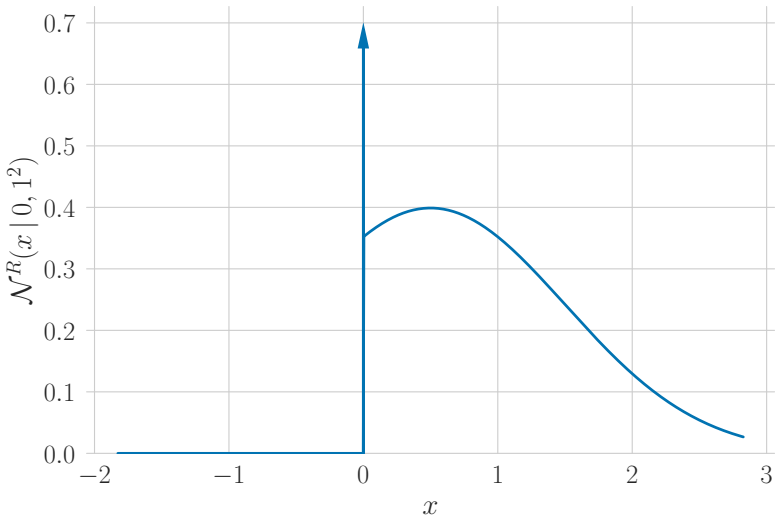


Figure 3.7: PDF of the rectified Gaussian distribution $\mathcal{N}^R(x; 0.5, 1^2)$.

the pre-activation $\mathbf{a}_l$ is Normally distributed with the above mean and variance [9].

We are now to consider the effect of the non-linear activation function on $\mathbf{a}_l$. The $\max(0, a_{i,l})$ causes all probability density spread over $\mathbb{R}^-$ to concentrate at zero as Figure 3.7 indicates. The resulting distribution is called rectified Gaussian and has its PDF given by

$$\mathcal{N}^R\left(a_{i,l}; \mu_{i,l}, \sigma^2_{i,l}\right) = \Phi\left(-\frac{\mu_{i,l}}{\sigma_{i,l}}\right)\delta(a_{i,l}) + \frac{1}{\sqrt{2\pi\sigma^2_{i,l}}} e^{-\frac{(a_{i,l}-\mu_{i,l})^2}{2\sigma^2_{i,l}}} \mathrm{U}(a_{i,l}), \tag{3.41}$$

where μ, σ^2 are the mean and variance of the Gaussian prior to rectification, $\Phi(\cdot)$ is the CDF of the standard Gaussian at the specified point, $\delta(\cdot)$ is Dirac's impulse function, and $\mathrm{U}(\cdot)$ is the unit step function. Its mean and variance are

$$\mu_{z_{i,l}} = \Phi\left(\frac{\mu_{a_{i,l}}}{\sigma_{a_{i,l}}}\right)\mu_{a_{i,l}} + \sigma_{a_{i,l}}\phi\left(-\frac{\mu_{a_{i,l}}}{\sigma_{a_{i,l}}}\right) \tag{3.42}$$

$$\sigma^2_{a_{i,l}} = m\left(\mu_{a_{i,l}} + \sigma_{a_{i,l}}\kappa\right)\Phi\left(-\frac{\mu_{a_{i,l}}}{\sigma_{a_{i,l}}}\right) + \Phi\left(\frac{\mu_{a_{i,l}}}{\sigma_{a_{i,l}}}\right)\sigma^2_{a_{i,l}}\left(1 - \kappa\left(\kappa + \frac{\mu_{a_{i,l}}}{\sigma_{a_{i,l}}}\right)\right) \tag{3.43}$$

where $\kappa = \phi\left(-\frac{\mu_{a_{i,l}}}{\sigma_{a_{i,l}}}\right) \Big/ \Phi\left(\frac{\mu_{a_{i,l}}}{\sigma_{a_{i,l}}}\right)$, and $\phi(\cdot)$ is the PDF of the standard Gaussian at the specified position.

The output distribution of the corresponding layer is then a Gaussian with entries determined by the above formulas plus an extra element we append for the bias term, which has mean $\mathbf{1}$ and variance $\mathbf{0}$. Then, finding the values μ_{z_L} and $\sigma^2_{z_L}$ consists in iteratively computing the equations (3.39, 3.40, 3.42, 3.43) from the first until the last layer for each data point $(\mathbf{x}_i, \mathbf{y}_i)$.

We summarize PBP's steps in Algorithm 3, in which we perform ADF updates

only and condense the forward pass responsible for computing the output distribution $\mathcal{N}(\mathcal{Y}_i \,|\, f(\mathcal{X}_i, \mathcal{W}), \gamma^{-1})$ into a single step.

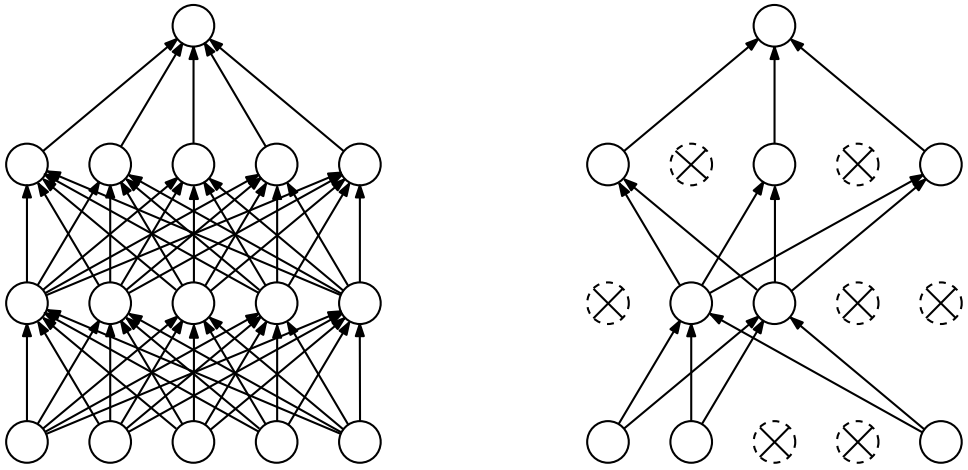
Algorithm 3: Probabilistic Backprop

Initialise parameters $\alpha_\lambda, \alpha_\gamma, \beta_\lambda, \beta_\gamma, \{\mu_j, \sigma_j^2\}_{j=0}^{|\mathcal{W}|}$
for $s \in \{\lambda, \gamma\}$ **do**
  $\alpha_s \leftarrow \alpha_s + \alpha_{s,0} - 1$
  $\beta_s \leftarrow \beta_s + \beta_{s,0}$
end for
while not converged **do**
  for $j = 1$ **to** $|\mathcal{W}|$ **do**
    for $s = 0$ **to** 2 **do**
      $K_s \leftarrow \mathcal{N}\left(\mu_j \,\middle|\, 0, \sigma_j^2 + \beta_\lambda/(\alpha_\lambda - 1 + s)\right)$
    end for
    $\alpha_\lambda \leftarrow \left[K_0 K_2 K_1^{-2}(\alpha_\lambda + 1)\alpha_\lambda^{-1} - 1\right]^{-1}$
    $\beta_\lambda \leftarrow \left[K_2 K_1^{-1}(\alpha_\lambda + 1)\beta_\lambda^{-1} - K_1 K_0^{-1}\alpha_\lambda\beta_\lambda^{-1}\right]^{-1}$
    $\mu_j \leftarrow \mu_j + \sigma^2 \nabla_\mu \log K_0$
    $\sigma_j^2 \leftarrow \sigma_j^2 - \left(\sigma_j^2\right)^2 \left[\left(\nabla_{\mu_j} \log K_0\right)^2 - 2\nabla_{\sigma_j^2} \log K_0\right]$
  end for
  for $j = 1$ **to** N **do**
    $\mu_{\mathbf{z}_L}, \sigma^2_{\mathbf{z}_L} \leftarrow f(\mathcal{X}_j, \mathcal{W})$
    for $s = 0$ **to** 2 **do**
      $K_s \leftarrow \mathcal{N}(\mathbf{y}_i \,|\, \mu_{\mathbf{z}_L}, \sigma^2_{\mathbf{z}_L}) + \beta_\gamma/(\alpha_\gamma - 1 + s)$
    end for
    $\alpha_\gamma \leftarrow \left[K_0 K_2 K_1^{-2}(\alpha_\gamma + 1)\alpha_\gamma^{-1} - 1\right]^{-1}$
    $\beta_\gamma \leftarrow \left[K_2 K_1^{-1}(\alpha_\gamma + 1)\beta_\gamma^{-1} - K_1 K_0^{-1}\alpha_\gamma\beta_\gamma^{-1}\right]^{-1}$
    $\mu_j \leftarrow \mu_j + \sigma^2 \nabla_\mu \log K_0$
    $\sigma_j^2 \leftarrow \sigma_j^2 - \left(\sigma_j^2\right)^2 \left[\left(\nabla_{\mu_j} \log K_0\right)^2 - 2\nabla_{\sigma_j^2} \log K_0\right]$
  end for
end while

It is important to note that the authors [14] assume the inputs are normalised, i.e., zero mean and unit variance. Hence, we need to normalise the data points before feeding them to the model and then to denormalise its outputs.

3.5 MC Dropout

The Monte Carlo Dropout [15], usually referred to as MC Droupout, stems from reinterpreting Dropout [63] as doing approximate Bayesian inference. Consequently, it suffices to use Droupout both during training and testing to obtain the advantages of Bayesian inference and model uncertainty measures.



(a) Standard Neural Network (b) Neural Network after dropout

Figure 3.8: Effect of dropout on the network.

3.5.1 Dropout

First, we review Dropout [63]. Succinctly, it is a stochastic regularisation technique to avoid overfitting the data. The basic idea is to corrupt the model's units with random multiplicative noise while training. Mathematically, it amounts to multiplying the input $\mathbf{h}_l$ of layer l pointwise by a realisation of a random vector $\boldsymbol{\epsilon}$, such that $\hat{\mathbf{h}}_\mathbf{l} = \mathbf{h}_l \odot \boldsymbol{\epsilon}$.

In the case of Bernoulli noise, the units are randomly *dropped out* with probability $1 - p$, i.e. their values are set to zero, each iteration, as illustrated in Figure 3.8. Dropping units causes different subnetworks with considerable less parameters to be used at each iteration (12 instead of 55 in Figure 3.8). When testing all units are kept as if an ensemble with all subnetworks was being used for evaluation.

Other works propose other types of noise. For example, in [63, 64], the authors study corrupting the activations with multiplicative Gaussian noise, and in [65], independently injecting noise on each weight, instead of on the input. The latter technique is called DropConnect.

3.5.2 A Bayesian View

Optimising a model with dropout and an approximate Bayesian inference model leads to similar objective functions with similar stochastic gradient update steps. So similar that under some conditions they are indeed equivalent [15]. Although we shall only consider here the Bernoulli Dropout, a similar development is possible for other types of noise.

Let ut first review the cost function of a standard deterministic neural network

$f(\cdot;\Theta)$ with deterministic parameters Θ:

$$\mathcal{L} = \mathcal{L}_{data}\left(\mathcal{D}, f(\cdot;\Theta)\right) + \mathcal{L}_{reg}(\Theta), \tag{3.44}$$

where the first term is data-dependent and measures the model's prediction error, and the second is a regularisation term to help against overfitting. Considering a regression task with data points $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i) \,|\, 1 \leqslant i \leqslant N\}$, a model with parameters $\Theta = \{\mathbf{M}_l \,|\, 1 \leqslant l \leqslant L\}$, and $\mathcal{L}_{reg}$ as the usual ℓ_2-norm with strength factors $\lambda_{\mathbf{M}}$, (3.44) becomes

$$\mathcal{L} = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{D}} \frac{1}{2} \left(\mathbf{y} - f(\mathbf{x};\Theta)\right)^2 + \sum_{\mathbf{M}\in\Theta} \lambda_{\mathbf{M}} \|\mathbf{M}\|_2^2 \,. \tag{3.45}$$

If we reinterpret Dropout as instead of corrupting the layers' inputs, corrupting the corresponding weights, we get for an arbitrary intermediate layer l with activation function $g_l(\cdot)$, the expression

$$\begin{aligned} \mathbf{h}_l &= g_l\left(\mathbf{M}_l \hat{\mathbf{h}}_{l-1}\right) && (3.46)\\ &= g_l\left(\mathbf{M}_l \left(\boldsymbol{\epsilon}_l \odot \mathbf{h}_{l-1}\right)\right) \\ &= g_l\left(\mathbf{M}_l \left(\mathrm{diag}\left(\boldsymbol{\epsilon}_l\right) \mathbf{h}_{l-1}\right)\right) \\ &= g_l\left(\left(\mathbf{M}_l \mathrm{diag}\left(\boldsymbol{\epsilon}_l\right)\right) \mathbf{h}_{l-1}\right) \\ &= g_l\left(\mathbf{W}_l \mathbf{h}_{l-1}\right), && (3.47) \end{aligned}$$

where $\mathbf{M}_l$ is the (deterministic) weight matrix, $\mathbf{h}_{l-1}$ the input, $\boldsymbol{\epsilon}_l$ the random noise, and $\mathbf{W}_l = \mathbf{M}_l \mathrm{diag}\left(\boldsymbol{\epsilon}_l\right)$.

We have demonstrated that multiplying the input is equivalent to multiplying the columns of the upcoming weight matrix. Hence, dropout can be understood as an operation that samples from a distribution over the weights just as BNNs do. Figure 3.9 shows the resulting weight matrix after being transformed by realisations of different types of noise.

If we now rewrite (3.45) taking this into consideration and make the sampling explicit, we get

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} \left(\mathbf{y}_i - f^{(i)}(\mathbf{x}_i;\Theta)\right)^2 + \sum_{l=1}^{L} \lambda_l \|\mathbf{M}_l\|_2^2, \tag{3.48}$$

where the notation $f^{(i)}(\cdot;\Theta)$ indicates a sample of the random parameters drawn for the data point $(\mathbf{x}_i, \mathbf{y}_i)$. Since Θ now defines distribution parameters, we replace it by Ψ in order to keep compliance with our notation of variational parameters and ease the comparison with other methods.

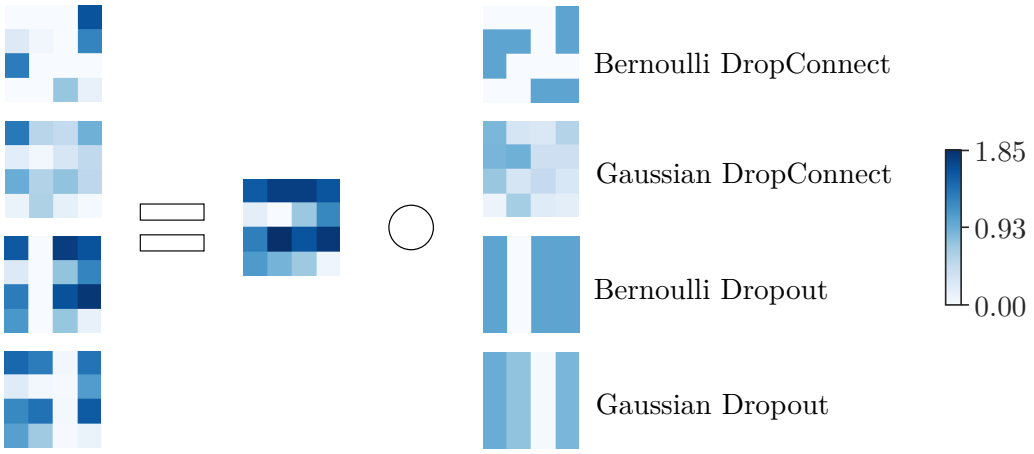


Figure 3.9: An illustration of the resulting sampled network weights using the different base variational distributions.

Substituting the first term of the above equation according to (2.3), we obtain

$$\begin{aligned}\mathcal{L} &= -\frac{1}{N}\sum_{i=1}^{N}\sigma_n^2 \log p(y_i \,|\, \mathbf{x}_i, \mathcal{W}^{(i)}) + \sum_{l=1}^{L}\lambda_l \|\mathbf{M}_l\|_2^2 - \frac{\sigma_n^2}{2}\log\left(2\pi\sigma_n^2\right) \\ &= -\frac{1}{N}\sum_{i=1}^{N}\sigma_n^2 \log p(y_i \,|\, \mathbf{x}_i; \mathcal{W}^{(i)}) + \sum_{l=1}^{L}\lambda_l \|\mathbf{M}_l\|_2^2 + \text{const}\ , \end{aligned} \tag{3.49}$$

where σ_n is the observation noise, $\mathcal{W}$ is the set of random weights and $\mathcal{W}^{(i)}$ is one sample from the distribution. The term that only depends on σ_n is considered a constant since this hyper-parameter is set by cross-validation and not gradient-descent optimisation.

Equation (3.49) is pretty similar to a one-sample MC estimator of the VI cost function $\hat{\mathcal{L}}_{VI}$ defined in (2.15), and (3.2), which after approximating with MC integration writes

$$\begin{aligned}\hat{\mathcal{L}}_{VI} &= -\frac{1}{T}\sum_{k=1}^{T}\log p(\mathcal{D} \,|\, \mathcal{W}^{(k)}) + D_{KL}\left(q(\mathcal{W}^{(k)}; \Psi) \| p(\mathcal{W}^{(k)})\right) \\ &= -\log p(\mathcal{D} \,|\, \mathcal{W}^{(1)}) + D_{KL}\left(q(\mathcal{W}^{(1)}; \Psi) \| p(\mathcal{W}^{(1)})\right) \\ &= -\sum_{i=1}^{N}\log p(y_i \,|\, \mathbf{x}_i, \mathcal{W}^{(1)}) + D_{KL}\left(q(\mathcal{W}^{(1)}; \Psi) \| p(\mathcal{W}^{(1)})\right). \end{aligned} \tag{3.50}$$

Taking the derivative of both (3.49) and (3.50) w.r.t. their parameters, we note that they possess the same objective (up to a constant scale factor), as long we assure that

$$\frac{\partial}{\partial \Psi} D_{KL}\left(q(\mathcal{W}; \Psi) \| p(\mathcal{W})\right) = \frac{N}{\sigma_n^2}\frac{\partial}{\partial \Psi}\sum_{l=1}^{L}\lambda_l \|\mathbf{M}_l\|_2^2. \tag{3.51}$$

This condition is now the sole thing impeding us from using dropout (or any other similar noise injection technique) as an approximate Bayesian model. For (3.51) to hold, we have to choose the hyper-parameters σ_n and $\Lambda = \{\lambda_l | 1 \leqslant l \leqslant L\}$ such that it induces a sensible prior $p(\mathcal{W})$ for the underlying variational distribution $q(\mathcal{W}; \Psi)$.

In Bernoulli Dropout, the input of each layer is independently sampled from a Bernoulli distribution. The equivalent distribution over the model weights factorises over the layers and the columns of the weight matrices as

$$\begin{aligned} q(\mathcal{W}; \Psi) &= \prod_{l,j} q(\mathcal{W}_{lj}; \Psi) \\ &= \prod_{l,j} \left[(1 - p_l)\delta(\mathbf{w}_{j,l} - \mathbf{0}) + p_l \delta(\mathbf{w}_{j,l} - \mathbf{m}_{j,l}) \right], \end{aligned} \tag{3.52}$$

where $\delta(\cdot)$ is the Dirac function, $1-p_l$ is the dropout probability for the l-th layer, and $\mathbf{w}_{j,l}$ and $\mathbf{m}_{j,l}$ are the j-th column of the random and deterministic weight matrices $\mathbf{W}_l$ and $\mathbf{M}_l$, respectively.

Specifying the prior $p(\mathcal{W})$ to have the same factorisation as $q(\mathcal{W}; \Psi)$, the KL divergence decomposes as the sum of separate KL terms, one for each factor, as

$$D_{KL}\left(q(\mathcal{W}, \Psi) \| p(\mathcal{W})\right) = \sum_{k,j} D_{KL}\left(q(w_{j,l}; \Psi_{j,l} \| p(w_{j,l})\right). \tag{3.53}$$

If we further define these factors to be centred Gaussian distributions over the weight vectors, that is, $w_{ij,l} \sim \mathcal{N}(0, \gamma_l^{-1})$, and assume some conditions are attended [66, Appendix], one of which being that the number of units per layer is large enough, we can write the KL between p and q as

$$\sum_{k,j} D_{KL}\left(q(w_{j,l}; \Psi_{j,l}) \| p(w_{j,l})\right) \approx \frac{p_l}{2} \mathbf{m}_{j,l}^T \mathbf{m}_{j,l} + \text{const}\ , \tag{3.54}$$

where the constant term represents all terms that do not include $\mathbf{m}_{j,l}$. Deriving it w.r.t to $\mathbf{m}_{j,l}$ we get

$$\begin{aligned} \frac{\partial}{\partial \mathbf{m}_{j,l}} D_{KL}\left(q(\mathcal{W}, \Psi) \| p(\mathcal{W})\right) &= \frac{\partial}{\partial \mathbf{m}_{j,l}} D_{KL}\left(q(w_{j,l}; \Psi_{j,l} \| p(w_{j,l})\right) \\ &\approx \frac{p_l \gamma_l}{2} \frac{\partial}{\partial \mathbf{m}_{j,l}} \mathbf{m}_{j,l}^T \mathbf{m}_{j,l} \\ &= \frac{p_l \gamma_l}{2} \frac{\partial}{\partial \mathbf{m}_{j,l}} \sum_{l=1}^{L} \|\mathbf{M}_l\|_2^2 \\ &= \frac{N}{\sigma_n^2} \frac{\partial}{\partial \mathbf{m}_{j,l}} \sum_{l=1}^{L} \lambda_l \|\mathbf{M}_l\|_2^2, \end{aligned} \tag{3.55}$$

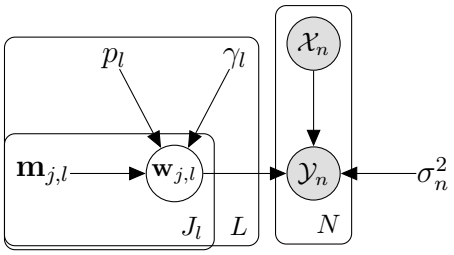


Figure 3.10: PGM representation of the MC Dropout model. The observed output $\mathcal{Y}_n$ is a noisy observation of the model output of the model output for the input $\mathcal{X}_n$ with the variance noise determined by the fixed parameter σ_n^2. The weight vectors $\mathbf{m}_l, j$ get selected by Bernoulli random variables with success rate p_l and $\mathbf{w}_l, j$ have centred Gaussian priors with fixed precision γ_l, whose value is readily determined by the choice of the two previous hyper-parameters together with the regularisation strength λ_l.

where $\lambda_l = (p_l \sigma_n^2)/(2N\gamma_l)$.

Let us stop here and digest this result. No specific assumption about the neural network architecture was assumed other than having a Dropout layer before each weight layer. This is the only restriction to obtain approximate Bayesian inference with the model in Figure 3.10, the other being readily attended: to every choice of dropout probability $1-p_l$, observation noise σ_n^2 (or, equivalently, noise precision τ_n) and regularisation strength λ_l corresponds a prior precision γ_l (or, according to [15] a prior length-scale l_l), whether or not its value is reasonable for the problem at hand. For other network architectures such as convolutional [66] and recurrent [67], few additional considerations are required to achieve a similar result. If the employed model does not have Dropout in-between every layer, as is usually the case in pre-trained models with only the last fully-connected layers of the classifier possessing Dropout, we can think of them as having a deterministic feature extractor part and a subsequent approximate Bayesian classifier. Although not as powerful, this is still a nice interpretation if we want to do inference and are bound to a given model.

Also, the posterior approximation for Bernoulli Dropout factorises over different layers and over connections going out of the same unit, but not over the connections arriving at the same unit. As the same Bernoulli random variable acts on the same weight matrix column, naturally they are not independent. The other methods of this chapter use mean-field approximation to the posterior, completely missing any codependency among the weights. In this sense, MC Dropout is less restrictive.

The predictive distribution computed as (2.6) is multi-modal and its variance can in principle assume any positive real value. Despite the approximating posterior essentially being composed of discrete Bernoulli random variables, the successive sum thereof throughout the layers allow the construction of arbitrary functions and thus arbitrary variances.

However, the author [66] warns that to get well-calibrated uncertainty estimates the dropout probability must be optimised as well. Since this is a variational parameter, it cannot be directly chosen by observing the ELBO objective [66, Appendix]. The recommendation is then to set it by maximising the log-likelihood over a validation set.

In the derivation of the KL-condition that leads to (3.54), the author relaxes the discrete Bernoulli distribution to a mixture of two Gaussians with small enough variances. The smaller the variances σ, the larger the KL divergence becomes, and hence the looser the ELBO becomes. When using the ELBO to compare models with different weight parameters, this does not represent a problem because the model remains the same and so does the lower bound. Hence, we can use the ELBO for model selection. For hyper-parameter optimisation, on the other hand, the model does change and thus its evidence. Therefore, using the ELBO actually implies comparing bounds to different things. As the lower bounds for our models are not tight, a higher ELBO do not necessarily imply a higher marginal probability.

Observing the behaviour of the dropout approximation for a Bayesian linear regression problem where the analtyical solution is known, the author concludes that the variational posterior induced by dropout collapse to the MAP solution in the limit of infinite data, specifically, the optimal value $p^* \to 1$ [66].

We summarize the resulting procedure in Algorithm 4, we illustrate the case for the Bernoulli dropout trained with a mini-batch of size 1. However, as pointed out at the beginning of this section, other stochastic regularisers can be recast as performing approximate Bayesian inference by following a similar derivation. For example, for DropConnect [65] the only difference is in using separate random variables for each weight instead of each column, what might have been guessed by observing Figure 3.9. If we consider an independent additive Gaussian noise for each weight parameter as the regulariser, we recover the algorithm of Section 3.3.2. Even though being a stochastic regulariser, interpreting the multiplicative Gaussian Dropout [63] as doing Bayesian inference with a log-uniform prior [64] has several issues as pointed out in [68].

Algorithm 4: MC Dropout

while not converged **do**
  Randomly sample a data example $\{\mathbf{x}_i, \mathbf{y}_i\}$
  for $l = 1$ **to** L **do**
    $\mathbf{W}_l \leftarrow \mathbf{M}_l \text{diag}(\boldsymbol{\epsilon}_l)$, where $\boldsymbol{\epsilon}_l \sim \text{Bern}(p_l)$
  end for
  $\mathbf{g} \leftarrow \frac{1}{2}\nabla \left(\mathbf{y}_i - f(\mathcal{X}_i; \{\mathbf{W}_l\}_{l=0}^L)\right)^2 + \sum_{l=1}^L \lambda_l \nabla \|\mathbf{W}_l \text{diag}(\boldsymbol{\epsilon}_l)\|_2^2$
  $\mathbf{m}_{j,l} \leftarrow \mathbf{m}_{j,l} - k\mathbf{g}$
end while

3.6 Fast Natural Gradient

The parameter space is in general Riemannian and not Euclidean, so learning methods should take the structure of the space into account [69]. Natural gradient methods do that by warping the gradient according to the information geometry encoded into the Fisher information matrix (see Appendix A.2). As a consequence, they are invariant (up to first order) to changes in the parameterisation of the problem, what is in stark contrast to standard gradient descent, whose efficiency and convergence rate are sensitive to the parameterisation used.

Current frameworks focus on MLE and adapting them for VI requires modifications in the code, increasing development time, memory requirements, and computation costs. For example, the algorithms of Sections 3.3 and 3.4 have twice the number of parameters of a deterministic model with the same architecture, besides the additional implementation effort. Adaptive optimisers further enlarge the costs since each parameter has its own scaling variable that regulates the learning rate.

The authors [16] build upon previous work [70] on natural gradient for Gaussian MFVI and propose a series of progressively more practical but less accurate optimisers. It is a lengthy read to grasp all the details, but certainly worth the effort. Here, we review and rederive the core algorithm of [16], named Vadam.

3.6.1 Vadam

From all reviewed methods, Vadam [16] is the more recent and practical one. Similar to the Adam optimiser [71] by construction, it is a natural gradient method (see Appendix A.2) with momentum designed specifically for MFVI. Starting from a parameter update equation proposal, the authors [16] embed several approximations defining different algorithms until reaching the method they name Vadam, for Variational Adam.

Gradient optimisers with momentum establish the update step as a linear combination between the steepest descent direction and the last displacement [72], such as

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \bar{\alpha}_t \nabla_{\mathbf{w}} f(\mathbf{w}_t) + \bar{\gamma}_t (\mathbf{w}_t - \mathbf{w}_{t-1}), \tag{3.56}$$

where $\{\bar{\alpha}_t\}$ and $\{\bar{\alpha}_t\}$ form a sequence of scalars that determines the contribution of each term and must obey the convergence conditions discussed in Section 2.2.4.

The latter term in (3.56) keeps the algorithm's movement along previous search directions, and is thus named momentum. Reasoning about its dynamics since the first iteration, each step can be understood as an exponentially decaying average of past gradients, hence the tendency to accumulate contributions in directions of

persistent descent, while directions that oscillate tend to cancel out, or at least remain small [72].

Instead of (3.56), the authors [16] propose

$$\boldsymbol{\eta}_{t+1} = \boldsymbol{\eta}_t - \bar{\alpha}_t \widetilde{\nabla}_{\boldsymbol{\eta}} f(\boldsymbol{\eta}_t) + \bar{\gamma}_t(\boldsymbol{\eta}_t - \boldsymbol{\eta}_{t-1}), \tag{3.57}$$

where $\widetilde{\nabla}$ is the natural gradient and the optimisation is on the natural parameter $\boldsymbol{\eta}$ of an exponential family member. For this family the natural gradient assumes a simple and efficient form, requiring less memory and computations than gradient-based methods. Besides, it improves the convergence rate by exploiting the information geometry of posterior approximations.

Constraining the variational approximation to the exponential family allows the use of the relation [73]

$$\widetilde{\nabla}_{\boldsymbol{\eta}} f(\boldsymbol{\eta}) \coloneqq \mathcal{I}^{-1}(\boldsymbol{\eta}) \nabla_{\boldsymbol{\eta}} f(\boldsymbol{\eta}) = \nabla_{\mathbf{m}} f(\mathbf{m}), \tag{3.58}$$

which states that the natural gradient w.r.t. the natural parameter is equal to the gradient w.r.t. the mean parameter $\mathbf{m}$ when $f(\cdot)$ is parametrised according to $\mathbf{m} = \mathbb{E}[\mathbf{u}(\mathbf{w})]$. This identity frees us from computing the Fisher matrix and its inverse, that is why it is so useful and a whole host of other practical natural gradient algorithms resort to it [44, 70, 74].

By writing (3.57) as a constrained minimisation problem as done in Appendix A.2 and using (3.58), we arrive at

$$\boldsymbol{\eta}_{t+1} = \boldsymbol{\eta}_t - \frac{\beta_t}{1-\alpha_t} \nabla_{\mathbf{m}} \mathcal{L}(\mathbf{m}_t) + \frac{\alpha_t}{1-\alpha_t}(\boldsymbol{\eta}_t + \boldsymbol{\eta}_{t-1}), \tag{3.59}$$

where α and β are related to the Lagrange multipliers of the KL restrictions.

The mean parameterisation for a univariate Gaussian[1] is

$$^{(1)}m = \mathbb{E}[x] = \mu, \tag{3.60}$$

$$^{(2)}m = \mathbb{E}\left[x^2\right] = \mu^2 + \sigma^2, \tag{3.61}$$

and the natural parameterisation is

$$^{(1)}\eta = \mu/\sigma^2, \tag{3.62}$$

$$^{(2)}\eta = -1/(2\sigma^2). \tag{3.63}$$

By using the chain rule we can express the gradient w.r.t. the mean parameters

[1] In the original paper, the authors consider a multivariate Gaussian with covariance structure Σ, simplifying to the mean-field case only at the end. However, here we already start with the independence assumption to avoid unnecessarily complicated equations.

(3.61) in terms of the μ and σ^2 such as

$$\frac{\partial f}{\partial\, {}^{(1)}m} = \frac{\partial f}{\partial \mu}\frac{\partial \mu}{\partial\, {}^{(1)}m} + \frac{\partial f}{\partial \sigma^2}\frac{\partial \sigma^2}{\partial\, {}^{(1)}m} = \frac{\partial f}{\partial \mu} - 2\mu\frac{\partial f}{\partial \sigma^2}, \tag{3.64}$$

$$\frac{\partial f}{\partial\, {}^{(2)}m} = \frac{\partial f}{\partial \mu}\frac{\partial \mu}{\partial\, {}^{(2)}m} + \frac{\partial f}{\partial \sigma^2}\frac{\partial \sigma^2}{\partial\, {}^{(2)}m} = \frac{\partial f}{\partial \sigma^2}. \tag{3.65}$$

Substituting these back into (3.59) and finding the corresponding update equations in terms of μ and σ^2, whose relation to the natural parameters is given by (3.63), is now a mere algebraic exercise that leads to

$$\mu_{t+1} = \mu_t - \frac{\beta_t}{1-\alpha_t}\sigma^2_{t+1}\nabla_\mu \mathcal{L}_t + \frac{\alpha_t}{1-\alpha_t}\sigma^2_{t+1}\sigma^{-2}_{t-1}(\mu_t - \mu_{t-1}), \tag{3.66}$$

$$\sigma^{-2}_{t+1} = \frac{1}{1-\alpha_t}\sigma_t^{-2} - \frac{\alpha_t}{1-\alpha_t}\sigma^{-2}_{t-1} + \frac{2\beta_t}{1-\alpha_t}\nabla_{\sigma^2}\mathcal{L}_t. \tag{3.67}$$

These pair of update equations are the natural momentum extension of [70]. We immediately note that the learning rate of μ gets scaled by the variance. Additionally, as σ^2 may assume negative values just like the methods in Section 3.3, one needs external constraints to sidestep this issue.

No specific knowledge of the cost function $\mathcal{L}$ has been absorbed into the algorithm so far. If we take into consideration the negative ELBO defined in (3.2), and specify univariate Gaussian priors $p(w) = \mathcal{N}(w; 0, \sigma_p^2)$ for the weights, recalling the derivatives of the KL term already calculated in (3.6)-(3.8), we get

$$\nabla_\mu \mathcal{L} = \nabla_\mu \,\mathbb{E}_q\left[N f(w)\right] + \frac{\mu - \mu_p}{\sigma_p^2}, \tag{3.68}$$

$$\nabla_{\sigma^2} \mathcal{L} = \nabla_{\sigma^2} \,\mathbb{E}_q\left[N f(w)\right] + \frac{1}{2}\left(\frac{1}{\sigma_p^2} - \frac{1}{\sigma^2}\right), \tag{3.69}$$

where we rewrite the data loss term as $\mathcal{L}_{data} = \mathbb{E}_q\left[N f(w)\right]$ with the negative log-likelihood $f(w) = -\frac{1}{N}\sum_{i=1}^{N}\log(x_i \,|\, w)$. Using the identities (3.3) and (3.4) to change the order of the gradient and expectation operators, we get

$$\nabla_\mu \mathcal{L} = N\,\mathbb{E}_q\left[\nabla_w f(w)\right] + \frac{\mu}{\sigma_p^2}, \tag{3.70}$$

$$\nabla_{\sigma^2} \mathcal{L} = \frac{N}{2}\,\mathbb{E}_q\left[\nabla_w^2 f(w)\right] + \frac{1}{2}\left(\frac{1}{\sigma_p^2} - \frac{1}{\sigma^2}\right), \tag{3.71}$$

from where we can appreciate the advantage of having defined $f(\cdot)$: the magnitude of its gradients does not depend on the data set size.

Now, after determining the prior and posterior distributions over the weights, we have completely defined the underlying Vadam model, shown in Figure 3.11. It has the same structure as the one used in Section 3.3 (Figure 3.3), the difference

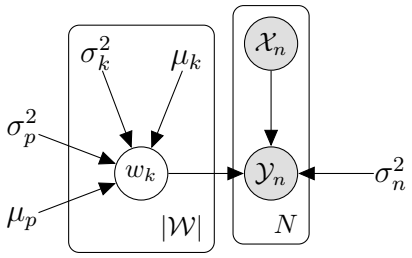


Figure 3.11: PGM representation of the Vadam model. It is the same as the one in Section 3.3. The observed output $\mathcal{Y}_n$ is a noisy observation of the model output for the input $\mathcal{X}_n$ with the variance noise determined by the fixed parameter σ_n^2. The constant values $\{\mu_p, \sigma_p^2\}$ govern the Gaussian prior distributions over the weights, while $\{\mu_k, \sigma_k^2\}$ their posteriors.

between both methods being the approximations included in Vadam to make it more computationally efficient.

If we use a one-sample MC estimator for the expectations (3.70) and (3.71), the gradient formulas become

$$\nabla_\mu \mathcal{L} = N \nabla_w f(w_t) + \frac{\mu}{\sigma_p^2}, \tag{3.72}$$

$$\nabla_{\sigma^2} \mathcal{L} = \frac{N}{2} \nabla_w^2 f(w_t) + \frac{1}{2} \left(\frac{1}{\sigma_p^2} - \frac{1}{\sigma^2} \right), \tag{3.73}$$

and $w_t \sim \mathcal{N}(\mu_t, \sigma_t^2)$ is a realisation of the random variable defined by the parameters from the t-th iteration. Then, if we plug these into (3.66), (3.67) and further relax σ_{t-1}^2 into σ_t^2 in the update equation for μ, we get

$$\mu_{t+1} = \mu_t - \frac{\beta_t}{1-\alpha_t} \sigma_{t+1}^2 N \left(\nabla_w f(w_t) + \frac{\mu_t}{N \sigma_p^2} \right) + \frac{\alpha_t}{1-\alpha_t} \sigma_{t+1}^2 \sigma_t^{-2} \left(\mu_t - \mu_{t+1} \right), \tag{3.74}$$

$$\sigma_{t+1}^{-2} = \sigma_t^{-2} + \frac{\beta_t}{1-\alpha_t} N \nabla_w^2 f(w_t) - \frac{\beta_t}{1-\alpha_t} (\sigma_t^{-2} - \sigma_p^{-2}). \tag{3.75}$$

Now, let us define the scaled prior precision $\widetilde{\lambda} \coloneqq \sigma_p^{-2}/N$ and the new parameter $s_t \coloneqq (\sigma_t^{-2} - \sigma_p^{-2})/N$, as well as replace the gradients with their stochastic versions $\widehat{\nabla}_w$ and $\widehat{\nabla}_w^2$. By doing so the above equations start to resemble the Adam update in Algorithm 5:

$$\begin{aligned} \mu_{t+1} = \mu_t &- \frac{\beta_t}{1-\alpha_t} \left[\frac{1}{s_t + \widetilde{\lambda}} \right] \left(\widehat{\nabla}_w f(w_t) + \mu_t \widetilde{\lambda} \right) \\ &+ \frac{\alpha_t}{1-\alpha_t} \left[\frac{s_t + \widetilde{\lambda}}{s_{t+1} + \widetilde{\lambda}} \right] (\mu_t - \mu_{t+1}) \end{aligned} \tag{3.76}$$

$$s_{t+1} = \left(1 - \frac{\beta_t}{1-\alpha_t} \right) s_t + \frac{\beta_t}{1-\alpha_t} \widehat{\nabla}_w^2 f(w_t). \tag{3.77}$$

Note that the equation for s_t became a simple exponential moving average. However, second derivatives are computationally expensive, besides being able to lead to negatives values for the parameter s. Resorting to the GGN approximation gives

$$s_{t+1} = \left(1 - \frac{\beta_t}{1-\alpha_t}\right) s_t + \frac{\beta_t}{1-\alpha_t} \left(\frac{1}{M} \sum_{i=1}^{M} \nabla_w f(w_t; x_i)^2\right), \tag{3.78}$$

where we make explicit the average over the mini-batch of data points.

Modern frameworks are not optimised to operate separately on each element of a batch after computing its derivatives, thus the above equation cannot be perfomed efficiently. Alternatively, we may incorporate yet another approximation, namely

$$\widetilde{\nabla}_w^2 f(w_t) \approx \frac{1}{M} \sum_{i=1}^{M} \nabla_w f(w_t; x_i)^2 \approx \left(\frac{1}{M} \sum_{i=1}^{M} \nabla_w f(w_t; x_i)\right)^2. \tag{3.79}$$

This last approximation, referred to as the gradient magnitude approximation and employed by several usual optimisers [71, 75, 76], causes s to act as diagonal rescaling that simply assures equal progress along each axis of μ rather than closely approximating the curvature [72] (disregarding the momentum term of the update equation that counter-balances this effect by favouring historically good directions).

The gradient magnitude approximation biases the estimation even more than GGN, its expectation is in-between that of GGN and the squared gradient of the full-batch. The larger the mini-batch, the larger the bias gets: if the whole data set is used to compute this approximation then all second-order information is lost, while if computed if a single data point it is equal to the GGN. Hence, there is a compromise between biasing estimations but converging quickly versus being "exact" (GGN-wise) but slow.

At last, we apply square root on the scaling vector s in the μ update formula so that the method gets more similar to Adam. Although this modification does not change the algorithm's fixed point solutions, it alters the dynamics [16]. At the end, we arrive at

$$\mu_{t+1} = \mu_t - \bar{\alpha}_t \left[\frac{1}{\sqrt{s_t} + \widetilde{\lambda}}\right] \left(\widehat{\nabla}_w f(w_t) + \mu_t \widetilde{\lambda}\right) + \bar{\gamma}_t \left[\frac{\sqrt{s_t} + \widetilde{\lambda}}{\sqrt{s_{t+1}} + \widetilde{\lambda}}\right] (\mu_t - \mu_{t+1}), \tag{3.80}$$

$$s_{t+1} = (1 - \bar{\alpha}_t)\, s_t + \bar{\alpha}_t \widehat{\nabla}_w^2 f(w_t), \tag{3.81}$$

where we have defined the step sizes $\bar{\alpha}_t \coloneqq \beta_t/(1-\alpha_t)$ and $\bar{\gamma}_t \coloneqq \alpha_t/(1-\alpha_t)$.

Unwinding these final update equations and using different step sizes γ_1 and γ_2 for μ and s instead of $\bar{\alpha}_t$ and $(1-\bar{\alpha}_t)$, respectively, we get the Algorithm 6, just like Adam 5. Remember that the scale factor s actually relates to σ^2 by $\sigma_t^{-2} = N s_t + \sigma_p^{-2}$

and each weight sample w_t is drawn for the distribution $\mathcal{N}(\mu_t, \sigma_t^2)$.

Algorithm 5: Adam

while not converged **do**
  $\mathbf{w} \leftarrow \boldsymbol{\mu}$
  Randomly sample a data example $\mathcal{X}_i$
  $\mathbf{g} \leftarrow -\nabla \log p(\mathcal{X}_i|\mathbf{w})$
  $\mathbf{m} \leftarrow \gamma_1 \mathbf{m} + (1-\gamma_1)\,\mathbf{g}$
  $\mathbf{s} \leftarrow \gamma_2 \mathbf{s} + (1-\gamma_2)\,(\mathbf{g} \odot \mathbf{g})$
  $\hat{\mathbf{m}} \leftarrow \mathbf{m}/(1-\gamma_1^t), \quad \hat{\mathbf{s}} \leftarrow \mathbf{s}/(1-\gamma_2^t)$
  $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} - \alpha\ \hat{\mathbf{m}}/(\sqrt{\hat{\mathbf{s}}} + \delta)$
  $t \leftarrow t + 1$
end while

Algorithm 6: Vadam

while not converged **do**
  $\mathbf{w} \leftarrow \boldsymbol{\mu} + \sigma \odot \epsilon$ where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, $\boldsymbol{\sigma} \leftarrow 1/\sqrt{N\mathbf{s} + \sigma_p^{-2}}$
  Randomly sample a data example $\mathcal{X}_i$
  $\mathbf{g} \leftarrow -\nabla \log p(\mathcal{X}_i|\mathbf{w})$
  $\mathbf{m} \leftarrow \gamma_1 \mathbf{m} + (1-\gamma_1)\,(\mathbf{g} + \sigma_p^{-2}\mu/N)$
  $\mathbf{s} \leftarrow \gamma_2 \mathbf{s} + (1-\gamma_2)\,(\mathbf{g} \odot \mathbf{g})$
  $\hat{\mathbf{m}} \leftarrow \mathbf{m}/(1-\gamma_1^t), \quad \hat{\mathbf{s}} \leftarrow \mathbf{s}/(1-\gamma_2^t)$
  $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} - \alpha\ \hat{\mathbf{m}}/(\sqrt{\hat{\mathbf{s}}} + \sigma_p^{-2}/N)$
  $t \leftarrow t + 1$
end while

Figure 3.12: Comparison between Adam [71] and Vadam [16]. The two algorithms are almost identical, but Adam performs MLE/MAP while Vadam performs VI.

Throughout the development of the Vadam algorithm, we have considered that the algorithm would already be running. Consequently, the exponential moving average would actually encode information about the geometry of the space. During the initial iterations, however, this estimation would be biased towards the starting point [71]. In order to reduce this effect, the authors [16] introduce a bias-correcting factor that decays exponentially as the optimisation runs.

The final method is indeed very similar to Adam [71], but has the advantage of providing uncertainty estimates due to the implicit posterior inference it performs. Apart from being fast, Vadam offers a plug-and-play manner of performing VI. Differently from the previous methods of this chapter, the user has only to define the model as if it were deterministic and optimise it with Vadam. There is no silver bullet and the price for such easiness and speed is inferior posterior estimates.

3.7 Comparing the Methods

In the remainder of this section, we shall compare the performance of the 4 algorithms we studied in-depth during this chapter.

3.7.1 UCI Data Sets

We benchmark the four studied algorithms in 8 different data sets of the UCI Machine Learning Repository [77] for the regression task, a procedure which has already become a staple in the related literature [14–16, 36, 37, 47, 78]. Below we give a brief description of each.

Boston

This data set consists of 14 attributes collected by the U.S Census Service concerning housing in the area of Boston Mass. The target variable is the median value of owner-occupied homes, while the other 13 variables measure property and neighbourhood characteristics such as average number of rooms per dwelling, nitric oxides concentration, per capita crime rate, distances to employment centres, pupil-teacher ratio, among others.

Concrete

The aim of this data set is to predict the compressive strength of concrete given its age and 7 ingredients, such as cement, water, fine and coarse aggregate, among other component concentrations. The output variable is a highly nonlinear function of the attributes.

Energy

This data set is composed of a collection of simulated buildings with different shapes and characteristics summarized by 8 attributes, among them: glazing, wall, roof and surface areas, as well as other properties such as orientation. The task is to predict the heating load requirements of the buildings as a function of the parameters.

Kin8nm

This data set consists of the angular positions of the joints of an 8-link all-revolute robotic arm, which is known to be highly non-linear. Data was synthetically generated from a simulation of its forward kinematics. The aim is to predict the distance of the end-effector from a given target.

Naval

This data set comprises of 16 measurements of a numerical simulator of a naval vessel with a gas turbine propulsion plant at steady state. From features such as ship speed, fuel flow, torques from turbine and propellers, temperatures and pressures coming in and out of the compressor, which indirectly represent the state of the system subject to performance decay, one should be able to predict the compressor decay state coefficient.

Powerplant

The aim of this data set is to predict the net hourly electrical energy output of a power plant. The features collected from a real plant over 6 years, when the plant was set to work with full load, consist of hourly averages of temperature, ambient pressure, relative humidity, and exhaust vacuum.

Wine

This data set disposes of 11 physicochemical characteristics of different brands of the red variant of the Portuguese "Vinho Verde" wine. From these features, one should predict the quality of the wine, a score between 0 and 10.

Yacht

In this data set, one should predict the hydrodynamic performance of sailing yachts, represented by their residuary resistance, from hull geometry coefficients, totalling 6 features.

3.7.2 Experimental setup

We evaluate training times, as well as the predictive (Gaussian) log-likelihood (2.3) and the Root Mean Squared Error (RMSE) defined as

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{N}}. \tag{3.82}$$

While the latter exclusively measures the prediction accuracy, thus assessing how close to the target value the predictions are in absolute terms, the former takes into account the prediction variance and thus incorporate into the evaluation the prediction uncertainty. Intuitively, the lower the variance, the more reliable the prediction should be and, hence, the higher the penalty for being wrong; but still we want predictions to be reliable so large variances also receive higher penalties.

Otherwise, constantly predicting uncertain values would amount to good scores, even though the model would not be of much use.

We do not directly measure structural uncertainty, that is, the uncertainty stemming from the model, which could be corrected with an infinite amount of data. However, highly uncertain weights, i.e., weights with large variances, lead to very different outputs for the same inputs each time we draw a different set of weight values, consequently those outputs frequently fall far from the true value even if their mean is correct. This causes the estimated predictive log-likelihood, that should be ideally high, to be low. Hence, this metric give us a sense of the model uncertainty, though indirectly.

We follow the setup proposed in [15]: for each data set we run the models on 20 random train-test splits after doing hyper-parameter search with 30 iterations of Bayesian Optimisation (BO) [79] on each split.

Bayesian Optimisation (BO)

BO is an approach to optimizing objective functions that take a long time to evaluate, in our case that would be running the model for E epochs to verify whether the chosen hyper-parameter configuration is the best so far. BO builds a surrogate for the objective and quantifies the uncertainty in that surrogate through Gaussian Process regression [80]. At each iteration we observe the objective at a new point, that is, at a new (hyper-)parameter configuration, use this information to update the Bayesian posterior probability distribution that describes the potential values of the objective at each point, and sample a new point whose values maximise a given acquisition function, i.e., th expected improvement.

Thus, BO keeps track of all previous configurations tested and the score they attain to decide what point in the hyper-parameter space to investigate next. By relying on that additional information, instead of, for example, local gradients, this technique can find good solutions for complex non-convex functions with considerably fewer iterations. On the other hand, the decision where to evaluate next, makes each iteration computationally expensive to run, imposing an overhead. Still, if the model evaluation is costly, as generally is the case nowadays for machine learning algorithms, and in our case means training the models for E epochs, this hyper-parameter tuning approach is justifiable.

For each method, we use the same 20 splits[2] to avoid fluctuations in the results due to the reduced size of the data sets and the effect different splits may have.

[2]Available at: `https://github.com/yaringal/DropoutUncertaintyExps/tree/master/UCI_Datasets`

For each split, we set the optimal hyper-parameter configurations of the precision parameter λ (or equivalently the prior variance σ_p^2), the observation noise precision γ, and, in the MC Dropout case, the dropout probability p by running 30 iterations of BO on the training set for 40 epochs. In order to reduce the noise stemming from sampling, the performance of the hyper-parameter configuration of one BO iteration is averaged over a 5-fold Cross-Validation, thus for each setting we train and evaluate the model 5 times. After finding the best configuration for a given split, we fit the model to the whole training set (of that specific train-test split). All this procedure follows from [15].

We observe that this structure escalates quickly, as for **each** data set and model we have:

$$20 \text{ splits} \times \left(30 \text{ BO iters} \times \left(5 \times \frac{4}{5}\right) \text{ CV iters } + 1 \text{ full run}\right) \times 40 \text{ epochs} = 96800, \tag{3.83}$$

plus the time BO takes to decide on the next point to test. Thus, to be able to analyse them in a viable amount of time, we perform ablation studies with a single hidden-layer network with 50 units.

The source codes for PBP[3] and Vadam[4] were borrowed from the authors' repositories. With exception of PBP which is implemented in Theano 1.0 [81], all remaining algorithms and supporting code is in PyTorch 1.0 [82].

3.7.3 Training Configuration

We maintain a training configuration similar to [16]. We use a mini-batch of size 32 on the 4 smaller data sets (Boston, Concrete, Energy and Yacht), and 128 on the other 4. During training, we employ respectively for BBB, Vadam, and MC Dropout, 20, 10, and 10 MC samples for the 4 smaller data sets, and 10, 5, and 10 for the 4 larger ones. All algorithms use 100 MC samples for evaluation. Again, PBP is the exception because it uses a mini-batch of size 1 in the original implementation [14], which we use, and has no MC approximation of the weights' posterior since it propagates entire distributions through the layers.

BBB, MC Dropout and Vadam use gradient descent optimisers. Both BBB and MC Dropout use the Adam optimiser [71], while Vadam is itsef a (variational) optimiser and the experiment consists of using it in lieu of Adam. Following [16], we configure the respective optimisers of all three methods to use a learning rate $k = 0.01$, and moving-average parameters $\gamma_1 = 0.99$ and $\gamma_2 = 0.9$ (instead of the usual $\gamma_1 = 0.9$ and $\gamma_2 = 0.999$) to encourage convergence within 40 epochs. The

[3] `https://github.com/HIPS/Probabilistic-Backpropagation`

[4] `https://github.com/emtiyaz/vadam`

Table 3.1: Average amount of time in seconds each algorithm takes to complete a whole training cycle, that is, from finding the optimal hyper-parameters to finding the final posterior approximation to the weights. In parenthesis, we ease the comparison by writing the time ratio to Vadam.

			Absolute Avg. Running Time (s)			
Dataset	**Size**	**Dim**	**BBB**	**MC Drop.**	**PBP**	**Vadam**
Boston	506	13	5945 (2.45)	1934 (0.80)	21 (0.008)	2431 (1.0)
Concrete	1030	8	11611 (2.53)	4453 (0.97)	34 (0.007)	4597 (1.0)
Energy	768	8	7944 (2.31)	2598 (0.76)	52 (0.01)	3439 (1.0)
Kin8nm	8192	8	13279 (2.20)	7819 (1.29)	231 (0.03)	6043 (1.0)
Naval	11934	16	24085 (3.18)	7914 (1.04)	334 (0.04)	7582 (1.0)
Powerplant	9568	4	15487 (2.62)	5498 (0.93)	220 (0.03)	5917 (1.0)
Wine	1599	11	4126 (3.28)	1945 (1.54)	51 (0.04)	1259 (1.0)
Yacht	308	6	2804 (1.91)	1112 (0.76)	41 (0.02)	1468 (1.0)

initial precision for the the posterior approximation is set to 10 (attention, this is not the prior precision) for BBB and Vadam.

3.7.4 Analysis

PBP automatically sets all its hyper-parameters by the Bayesian framework thanks to the hyper-priors, thus dispensing with the BO. In this case, the number in (3.83) reduces to 20, that is, 1 run per random split. Table 3.1 shows the required (wall-clock) time each algorithm takes to complete the full training schedule (including BO).

PBP outspeeds all others by being at least 25 times faster. This difference results from the absence of hyper-parameter tuning, which exempts the method from running the equivalent of $30 \times 4 = 120$ times to find a good hyper-parameter configuration prior to finally fitting to the full training set. On top of that, there is the overhead imposed by the Bayesian Optimisation inference. Instead of requiring computer time, PBP requires human time to workout all its derivations and approximations. However, if we were not take the hyper-parameter tuning into consideration, PBP's advantage would fade away and it would actually be the slowest method on average. There are actually different factors contributing to this:

- PBP's current implementation uses a mini-batch size of 1, and increasing it to 32, the same size as the others, makes the method once again the fastest [83], though not by that large of a margin as before;
- PBP uses the framework Theano, which is no longer officially supported, while the other 3 methods build upon Pytorch [82], a more recent and rapidly grow-

Table 3.2: The average RMSE (low values are better) over the 20 random resampled splits of the UCI regression data sets. The ± value reported is the standard error and not the standard deviation

	Avg. Test RMSE			
Dataset	**BBB**	**MC Drop.**	**PBP**	**Vadam**
Boston	3.630 ± 0.2262	3.700 ± 0.1806	2.965 ± 0.1704	3.852 ± 0.2689
Concrete	6.183 ± 0.1052	10.031 ± 0.1847	5.683 ± 0.1098	6.846 ± 0.0712
Energy	2.747 ± 0.0609	1.697 ± 0.0663	1.817 ± 0.0525	1.722 ± 0.1364
Kin8nm	0.096 ± 0.0005	0.163 ± 0.0001	0.092 ± 0.0000	0.106 ± 0.0012
Naval	0.004 ± 0.0002	0.011 ± 0.0001	0.006 ± 0.0000	0.002 ± 0.0001
Powerplant	4.258 ± 0.0352	7.041 ± 0.0380	4.132 ± 0.0327	4.290 ± 0.0303
Wine	0.660 ± 0.0084	0.654 ± 0.0080	0.635 ± 0.0078	0.653 ± 0.0085
Yacht	2.342 ± 0.0927	1.612 ± 0.0883	1.071 ± 0.0512	1.432 ± 0.1025

Table 3.3: The average log-likelihood (high values are better) over 20 the random resampled splits of the UCI regression data sets. The ± value reported is the standard error and not the standard deviation

	Avg. Test Log-Likelihood			
Dataset	**BBB**	**MC Drop.**	**PBP**	**Vadam**
Boston	-2.751 ± 0.057	-2.747 ± 0.039	-2.558 ± 0.084	-2.840 ± 0.074
Concrete	-3.242 ± 0.017	-3.729 ± 0.017	-3.164 ± 0.022	-3.408 ± 0.012
Energy	-2.460 ± 0.023	-1.968 ± 0.040	-2.049 ± 0.022	-2.220 ± 0.078
Kin8nm	0.939 ± 0.007	0.379 ± 0.011	0.957 ± 0.000	0.737 ± 0.006
Naval	4.210 ± 0.048	3.050 ± 0.020	3.667 ± 0.005	5.131 ± 0.065
Powerplant	-2.866 ± 0.009	-3.372 ± 0.006	-2.839 ± 0.008	-2.878 ± 0.008
Wine	-1.053 ± 0.044	-0.998 ± 0.013	-0.968 ± 0.014	-1.042 ± 0.029
Yacht	-2.482 ± 0.015	-1.952 ± 0.033	-1.645 ± 0.016	-1.738 ± 0.042

ing framework powered by Facebook Artificial Intelligence Research, hence the mathematical operations are expected to be better optimised.

BBB is by far the slowest, taking on average 2.5 times what Vadam takes to train. The latter does not have additional parameters for the variances of weights; instead it directly computes them from the intermediate variable used to normalise the directions of the parameter space, something the optimser Adam already does. However, we need to remember that this difference is also caused in part by using twice the number of MC samples; equating them would narrow down this gap to ≈ 1.4. Still, even with such advantage during training, BBB's test performance is comparable or even slightly worse on the test set on both RMSE and log-likelihood, according to Table 3.2 and Table 3.3, respectively.

We choose not to compare the best scoring method out of the 4 using a paired t-test as in [16] for the the samples, i.e., the performances on each random split, are

not independent. Indeed, the realisations consist of the 20 different random draws of 10% of the (same) data set, which, violates the assumptions of this statistical test.

All in all, PBP is the best performing method out of the 4, followed by Vadam. In addition, another strength PBP possesses inherited from EP is being naturally well-suited to data-parallelisation across machines, and if using only ADF updates, online learning.

MC Dropout does not achieve good results, and the values we find are remarkably different from the ones in [15]. Fixing the length-scale to 0.01 and only optimising the dropout probability p and the noise precision λ per the original paper [15] does not change the outlines of the results. The difference lies in the following: the author [15] claims training the model for $10\times$ more iterations (400 epochs in total) after finding the optimal hyper-parameter values because dropout takes longer to converge. Nevertheless, further experiments are needed so that we can confirm this is the reason behind the discrepancy and under the same conditions what the results of the other methods would be.

On a final note, we leave a general recommendation for those needing to develop a custom solution for a certain task: use Vadam [16], it is fast, out-of-the-box and has reasonable performance. It still needs hyper-parameter tuning, but at this point, almost everything does. If the problem calls for better predictive accuracy or uncertainty estimation, resort to PBP or other method not covered here, a few of which are mentioned in Section 3.8.

3.8 Closing Remarks

In this chapter we have discussed BNNs, along with motivations for recurring to the computationally heavier Bayesian approach instead of contenting ourselves with traditional point estimates. Bayesian models offer a large number of advantages such as robustness to overfitting, principled model comparison and uncertainty estimation not only in their outputs, but also in all of their parameters.

Additionally, we reviewed and experimentally compared 4 key variational algorithms throughout the chapter. Namely, Bayes by Backprop [13], Probabilistic Backprop [14], MC Dropout [15], and Vadam [16].

3.8.1 Practical Comparison of Studied Algorithms

To make future reference easier or even for those not interested in a lengthy analysis, we build what we call the *Practitioner's Table* shown in Table 3.4. There is no number nor formula, only plain simple adjectives to characterise what we believe to

be the the 3 most important aspects of a BNN algorithm:

- The implementation effort to build a custom solution;
- The quality of the model predictions, both accuracy and uncertainty;
- The time it takes to train the model.

By carefully reading the previous sections, the reader will have no difficulty to comprehend the content of Table 3.4.

3.8.2 Alternative directions

Even though we have only seen algorithms that do not (explicitly) model the correlation structure amount the weights, this also is an active research subject with many interesting works such as

- Matrix variate Gaussian prior [36] and posterior approximation [37];
- Structured covariance with noisy natural gradient [78];
- Low-rank covariance approximation with natural gradient [84].

Although the above methods relies either on VI, ADF or EP, by focusing on modelling the structure between the parameters, they achieve better posteriors approximations and uncertainty estimations.

There is a whole other sort of methods that relies on Markov Chain MC approximations to the posterior predictive density, which is not the focus of our discussion. Still, we name a few so that the interested reader knows where to start:

- Hamiltonian MC [85];
- Stochastic gradient Langevin dynamics [56, 86];
- Posterior distribution distillation [87].

Table 3.4: Practitioner's Table: a rough comparison between the variational methods studied for BNNs. The MC Dropout quality is poor according to the results of our simulation in which all algorithms ran in approximately similar conditions, but in the original work [15], the authors obtain considerably better results by training longer.

Method	Effort	Quality	Train
BPB	Medium	OK	Slow
PBP	Very hard	Good	Very fast
MC Dropout	Very easy	Poor*	OK
Vadam	None	OK	OK+

Chapter 4

(Deep) Generative Algorithms

This chapter will use the tools of MBML and VI with a very different idea in mind. Here, we are interested in modelling the process that causes the observed data. This empowers us to simulate new data, create world models, grasp underlying generative factors, learn with little to no supervision.

What to expect in the following sections:

- What a generative model is and what its benefits are;
- How to evaluate a generative model;
- Detailed explanation of the Variational Autoencoder (VAE);
- Developments that enhance the VAE's performance in different aspects;
- Central problems of this breed of models;
- Examples and demonstrations of the discussed VAE models.

After the dense chapter on BNNs, the reader will find this one a bit simpler, as the required tools were actually already introduced. By the end of this chapter, one should:

- Be able to characterise generative models;
- Understand when they are useful;
- Know the challenges on assessing their quality;
- Possess breadth knowledge on VAE's core idea;
- Comprehend the ideas of its extensions and what effectively changes;
- Be motivated to seek more information on oneself's own.

4.1 Motivations

Generative models are statistical models of data that (try to) capture the entire probability distribution from the observables, that is, to estimate $p(\mathcal{X})$ from $\mathcal{D} = \{\mathcal{X}\}_n$. It is a complete description of the probabilistic model that generates the observed data. In possession of the full model we can extrapolate to unseen examples, generate new samples, infer relations and dependencies, perform prediction, and do all else probability theory allows.

Differently from discriminative models which estimate the distribution on $\mathcal{Y} \mid \mathcal{X}$ and hence need the supervisory signal coming from the target $\mathcal{Y}$, e.g., image labels, generative models need not to be supervised. Thus this kind of model constitutes a good basis for developing unsupervised and semi-supervised algorithms, at the same time being capable of handling target variables by modelling the joint distribution $p(\mathcal{Y}, \mathcal{X})$ and performing discriminative tasks such as classification. The models in Chapter 3 although probabilistic, do not estimate the joint $p(\mathcal{Y}, \mathcal{X})$, but instead the conditional $p(\mathcal{Y} \mid \mathcal{X})$, where $\mathcal{Y}$ in our examples is a real-valued scalar variable.

Knowledge of the complete probabilistic model means we can simulate how the world evolves [88, 89], anticipate and plan for the future [90, 91], reason and make decisions [91, 92], understand elements and their factors of variation [92, 93], among other high-level abstract tasks. Exciting applications involve mainly image, such as super-resolution [94], compression [95], denoising [96], and audio [97]. Other examples outside the multimedia domain and arguably even more compelling are in chemistry for efficient exploration of new compounds [98], and biology for prediction of the effects of mutations in proteins and RNAs [99].

A classical example of a generative model is Naive Bayes that constructs the joint probability for classification. Here, the focus are on modern methods that use approximate inference and, though, many exist with different combinations of modelling assumptions, architectures and inference algorithms, we discuss VAEs. The interested reader will find at the end of the chapter a two-paragraph presentation on alternative modern methods and indications of good reads.

4.2 Evaluating Generative Networks

There is no universal metric for measuring the performance of a generative model, thus assessing its quality is not straightforward and is often misleading if not done with this care in mind. Of course when the KL divergence is zero, the model matches the true posterior perfectly and the samples are indistinguishable from those drawn from the true posterior.

While models trained for the same criteria do relate well across the different axis

of properties, and comparison among them being direct, i.e., training criteria itself as metric, this does not hold true for different objective functions. Then the question arises on how to rate the models, or, even more fundamentally, on what exactly to search for in a model: likely samples, high quality samples, compact representations, useful representation, etc.

A model may have a high log-likelihood but have an average mode-matching behaviour, causing it to assign probability mass to low-density regions. Then, generated samples are likely to be very distinct from samples from the true distribution in spite of the approximation's good log-likelihood. On the other hand, for a model with moment-matching behaviour the contrary may be true: realistic samples but a poor log-likelihood if, for example, the true model is highly multi-modal, which would mean that the approximation fits well only one such mode and misses out on a great deal of information. In this case, sample diversity would also be low. The higher the dimension of space, the less correlated the metrics become [100].

Average log-likelihood has become the de facto standard measure of quality for generative models, but depending on the type of the model under consideration, it may even not be possible to compute it (at least in a viable amount of time). Furthermore, as we explained above, average log-likelihood and sample quality measure different things, not being possible to deduce one from the other. Finally, there is still the issue of diversity that connects the other two, that is, if samples seem to come from the true distribution and, at the same time, have great diversity, then probably overall the two densities match well, what implies good average log-likelihood for the model. The diversity of the distribution relates to its spread or entropy, and allows us to gauge overfitting.

In models for which a direct measure of the log-likelihood is not possible, it is common to resort to Parzen window estimates, which consists on drawing samples from the model and constructing another model, a tractable one, through non-parametric methods to emulate the intractable one [100]. Nevertheless, the authors [100] strongly recommend against Parzen window. They experimentally demonstrate its estimates not to correlate well with likelihood nor sample quality.

Model sample inspection is naturally the go-to metric when dealing with image synthesis, besides it allows us to peek into the model in an intuitive manner, possibly giving us insights about what is happening. Still, only in the specific case of image or other sensory samples for which humans evolved to excel at is this approach effective, for other types of data we would need in-depth analysis. Thus, samples represent an interesting diagnostic tool but should not be used as a proxy for other tasks [100].

In addition, perceptual quality metrics do not take into account generalisation capacity. If the model simply outputs training examples without ever producing a

new one, samples would seem real, and actually would indeed be. The usual Euclidean distance nearest neighbours algorithm to counter this flaw and find similar images in the training set is also problematic. It does not correlate well with perception: perceptually similar images can have large distances, e.g., a one-pixel shift of a texture-rich image. On top of that, overfit models do not necessarily reproduce the images from the data set [100].

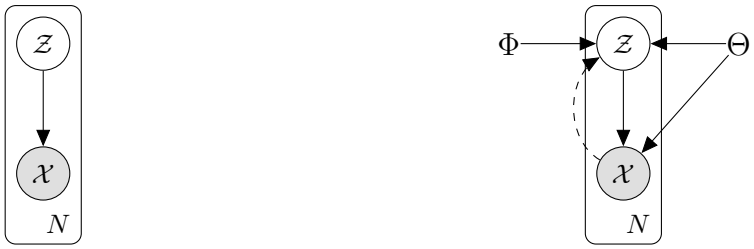
A different approach for assessing the model's performance is to measure it directly on a surrogate task, what is specially useful if the aim is to learn good feature representations. Although indirect, measuring the effect of the model in the intended downstream application provides exactly what one searches, all other metrics being secondary. For example, one could use a linear classifier on the learned representations to test if they form well defined clusters, which would be indicated by a high classification accuracy.

The community has been devoting recent efforts to propose principled scores that embrace (some of) these aspects, and allow to objectively compare between competing algorithms. For images, the Inception Score [101] and the Frechet Inception Distance [102] are two popular metrics, they use pretrained classification models to compare between a hold-out test set and a set of generated samples. The former measures sharpness and diversity of samples through the distribution over the classifier's prediction, whereas the latter measures similarities in the feature representation space. Both empirically correlate well with the perceived quality of samples. However, using the Inception Score of a model pretrained on a dataset different from the one on which the generative model is evaluated can be misleading when ranking models [103]. Specifically, the commonly used Imagenet-pretrained Inception classifier is only valid for evaluating generative models also trained on ImageNet.

In summary, there is no universal metric and sample quality, classification accuracy, and log-likelihood are largely independent properties in high-dimensional spaces. Thus, proper assessment of the model's performance depends on the application: different applications require different metrics.

4.3 Variational Autoencoders

We begin this section by posing a modelling problem and steadily progressing towards constructing the VAE. We start from latent variable models, whose value we already discussed in Section 2.1.2. Thus we have $p(\mathcal{D}) = \int p(\mathcal{D}, \mathcal{Z}) d\mathcal{Z}$, where $\mathcal{D}$ is the set of observations $\{\mathcal{X}\}_{n=1}^{N}$ we possess, $\mathcal{Z}$ is the unknown latent variables which we assume are the hidden causes for the observed $\mathcal{D}$, and $p(\mathcal{D}, \mathcal{Z})$ is the generative model of Figure 4.1a.



(a) The latent variable model (b) The parametrised model

Figure 4.1: Graphical representations of the generative model $p(\mathcal{D}, \mathcal{Z})$. Figure 4.1a is the initial model we formulate and Figure 4.1b the parametrised model with the dashed line representing the posterior approximation $q(\mathcal{Z} \mid \mathcal{D}; \Psi)$.

We can actually rewrite the integral over the joint distribution according to the chain rule for conditional probability, what gives us $p(\mathcal{D} \mid \mathcal{Z})p(\mathcal{Z})$, where $p(\mathcal{Z})$ is the prior distribution over the latent space, and $p(\mathcal{D} \mid \mathcal{Z})$ the likelihood function. For all but very simple models, the integral $\int p(\mathcal{D} \mid \mathcal{Z})p(\mathcal{Z})$ is intractable and cannot be analytically calculated, thus we further approximate it by MC sampling, such as

$$p(\mathcal{D}) = \int p(\mathcal{D}, \mathcal{Z}) d\mathcal{Z} = \int p(\mathcal{D} \mid \mathcal{Z}) p(\mathcal{Z}) d\mathcal{Z} \approx \frac{1}{T} \sum_{i=1}^{T} p(\mathcal{D} \mid \mathcal{Z}^{(i)}), \tag{4.1}$$

with $\mathcal{Z} \sim p(\mathcal{Z})$. However, the latent space may be high dimensional and finding samples of $\mathcal{Z}$ for which $p(\mathcal{D} \mid \mathcal{Z})$ is large is challenging: we need millions of draws to obtain reasonable estimates for $p(\mathcal{D})$. Then, how to choose $p(\mathcal{Z})$ such that we obtain plausible values of $\mathcal{Z}$, for which $p(\mathcal{D} \mid \mathcal{Z})$ is high, with high probability?

Once more, we rewrite the problem as

$$\begin{aligned} p(\mathcal{D}) &= \int p(\mathcal{D} \mid \mathcal{Z}) p(\mathcal{Z}) d\mathcal{Z} \\ &= \int p(\mathcal{D} \mid \mathcal{Z}) p(\mathcal{Z}) \frac{q(\mathcal{Z} \mid \mathcal{D})}{q(\mathcal{Z} \mid \mathcal{D})} d\mathcal{Z} \\ &= \mathbb{E}_q \left[\frac{p(\mathcal{D} \mid \mathcal{Z}) p(\mathcal{Z})}{q(\mathcal{Z} \mid \mathcal{D})} \right] \\ &\approx \frac{1}{T} \sum_{i=1}^{T} \frac{p(\mathcal{D} \mid \mathcal{Z}^{(i)}) p(\mathcal{Z}^{(i)})}{q(\mathcal{Z}^{(i)} \mid \mathcal{D})}, \end{aligned} \tag{4.2}$$

with $\mathcal{Z} \sim q(\mathcal{Z} \mid \mathcal{D})$. Under this new perspective, the sampling process occurs according to the new distribution $q(\mathcal{Z} \mid \mathcal{D})$ and to obtain the same result as before we need to properly weight the values of $p(\mathcal{D} \mid \mathcal{Z})$ by $p(\mathcal{Z})/q(\mathcal{Z} \mid \mathcal{D})$.

This alternative approach corresponds to MC with Importance Sampling (IS) [20]. This technique is generally applied to reduce the variance of the estimator or when it is difficult to simulate from the original density, the latter being

the present case. Clearly, the optimal proposal distribution $q^*(\mathcal{Z} \mid \mathcal{D})$ is

$$q^*(\mathcal{Z} \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \mathcal{Z})p(\mathcal{Z})}{p(\mathcal{D})} = p(\mathcal{Z} \mid \mathcal{D}), \tag{4.3}$$

for which we obtain for the single-sample estimator $\hat{p}(\mathcal{D})$ the true distribution we seek, that is,

$$\hat{p}(\mathcal{D}) = \frac{p(\mathcal{D} \mid \mathcal{Z}^{(1)})p(\mathcal{Z}^{(1)})}{q(\mathcal{Z}^{(1)} \mid \mathcal{D})} = \frac{p(\mathcal{D} \mid \mathcal{Z}^{(1)})p(\mathcal{Z}^{(1)})}{\frac{p(\mathcal{D} \mid \mathcal{Z}^{(1)})p(\mathcal{Z}^{(1)})}{p(\mathcal{D})}} = p(\mathcal{D}). \tag{4.4}$$

Yet, the inability to compute $p(\mathcal{D}) = \int p(\mathcal{D} \mid \mathcal{Z})p(\mathcal{Z})d\mathcal{Z}$ was the very reason that motivated us to search for other solutions. Let us parameterise the distributions as $p(\mathcal{D} \mid \mathcal{Z}; \Theta)$ and $q(\mathcal{Z} \mid \mathcal{D}; \Psi)$, and jointly optimise for Θ and Ψ. The corresponding Probabilistic Graphical Model (PGM) is in Figure 4.1b. We immediately note that the ideal value for Ψ is such that $q(\mathcal{Z} \mid \mathcal{D}; \Psi) = p(\mathcal{Z} \mid \mathcal{D}; \Theta)$, or at least as close to it as possible. Thus we are learning both the posterior distribution and the likelihood function of the data. While the first allows inferring latent distributions relating to the observables, the second enables the generation of new samples when paired with the prior, what effectively means sampling from the joint distribution.

Note that we have arrived at a familiar framework: we wish to maximise the mode evidence $p(\mathcal{D}; \Theta)$ and for this we do

$$\underset{\Theta, \Psi}{\operatorname{argmax}}\, p(\mathcal{D}; \Theta) = \underset{\Theta, \Psi}{\operatorname{argmax}} \log p(\mathcal{D}; \Theta) = \underset{\Theta, \Psi}{\operatorname{argmax}} \log \mathbb{E}_q \left[\frac{p(\mathcal{D} \mid \mathcal{Z}; \Theta)p(\mathcal{Z})}{q(\mathcal{Z} \mid \mathcal{D}; \Psi)} \right]. \tag{4.5}$$

Applying Jensen's inequality exactly as we did in Section 4.3 leads us once again to the ELBO objective (2.14, 2.15) as we verify in

$$\log \mathbb{E}_q \left[\frac{p(\mathcal{D} \mid \mathcal{Z}; \Theta)p(\mathcal{Z})}{q(\mathcal{Z} \mid \mathcal{D}; \Psi)} \right] \geqslant \mathbb{E}_q \left[\log \frac{p(\mathcal{D} \mid \mathcal{Z}; \Theta)p(\mathcal{Z})}{q(\mathcal{Z} \mid \mathcal{D}; \Psi)} \right] = \text{ELBO}(\Theta, \Psi). \tag{4.6}$$

Even though the final utility function borrows the same form as that of the methods in Chapter 3, the model we describe now is fundamentally different in what it accomplishes. It performs MLE/MAP estimation of the model parameters Θ, and VI only on Ψ. In Chapter 3, we performed VI on all model parameters, which were global parameters, since they were the same for all data points, and there were no local latent variables. Although possible to do VI on both Ψ and Θ, a Full Variational Bayes approach, here we focus on Ψ only, as is customary in the related literature. A full treatment can be found in the appendix of [10].

Note that the target density $p(\mathcal{D} \mid \mathcal{Z}; \Theta)$ changes over the course of training, not being static as the cases of Chapter 3. Hence, $q(\mathcal{Z} \mid \mathcal{D}; \Psi)$ must track this evolution

so that the approximation remains "close" to the true (modelled) distribution. We obtain the posterior approximation through a *recognition model* whose outputs are the parameters Φ that determine the member of the specified parametric family, e.g., mean and variance for the Gaussian case. Each new data point x' goes through a function $f(x'; \Psi) \mapsto \Phi$. Instead of solving a different optimisation problem for each observable to find the corresponding latent posterior distribution, we solve only one: obtaining the parameters that define the mapping $f(\cdot; \Psi)$. This approach of sharing the variational parameters across all data points gets the name of *amortised inference* and is common in settings with large data sets because it effectively amortises the inference cost.

From the complete model developed so far, shown in Figure 4.2, we observe that the distribution over the latent space $\mathcal{Z}$ is in-between the recognition model and the likelihood, creating an information bottleneck if $dim(\mathcal{Z}) < dim(\mathcal{D})$. Generally, this is the case since we assume the data lives in a lower-dimensional manifold than the space in which it is defined. Therefore, we may interpret the present class of models as encoding $\mathcal{D}$ to a lower-dimensional space $\mathcal{Z}$, thus throwing away unnecessary information and preserving what is meaningful, which actually helps the decoder to reconstruct the original input. Hence, $p(\mathcal{D} \mid \mathcal{Z}; \Theta)$ can be understood as a probabilistic encoder and $q(\mathcal{Z} \mid \mathcal{D}; \Psi)$ as a probabilistic decoder. Indeed, if we write the ELBO in its most usual form, we have

$$\begin{aligned} \text{ELBO}(\Theta, \Psi) &= \mathbb{E}_q \left[\log p(\mathcal{D} \mid \mathcal{Z}; \Theta)\right] - D_{KL}\left(q(\mathcal{Z} \mid \mathcal{D}; \Psi) \| p(\mathcal{Z})\right) \\ &= \sum_{n=1}^{N} \mathbb{E}_q \left[\log p(\mathcal{X}_n \mid \mathcal{Z}_n; \Theta)\right] - D_{KL}\left(q(\mathcal{Z}_n \mid \mathcal{X}_n; \Psi) \| p(\mathcal{Z})\right). \end{aligned} \tag{4.7}$$

We note that the first term aims at maximising the reconstruction error, e.g., assuming Gaussian additive noise this would become the squared distance between $\mathcal{D}$ and $\hat{\mathcal{D}}$. The second term, on the other hand, works as a regularisation imposing structure to the latent space. Without the KL term, the latent distribution would degenerate to a point estimate, namely, the MLE solution that maximises the log-likelihood. This would entail a conventional autoencoder that deterministically maps a data point $\mathcal{D}_i$ to $\mathcal{Z}_i$ and deterministically reconstructs it. Consequently, nearby latent points would not necessarily represent similar data points, just imagine for example a lookup table. Thus, the latent space would not have any special structure or meaning as we desire. The autoencoder with the KL regularisation term in the latent space receives the name of *Variational Autoencoder*.

The fact of the latent distribution being sandwiched between the encoder and decoder raises difficulty when trying to use gradient descent to optimise the model. This is only natural since the objective function we optimise (4.7) has the same

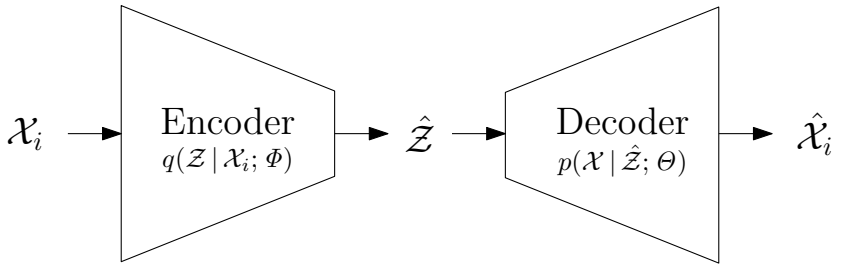


Figure 4.2: Schematic of the VAE model. The data point $\mathcal{D}_i$ gets mapped to a distribution over the latent variable $\mathcal{Z}$, from which we sample a value $\hat{\mathcal{Z}}$ (one-sample MC estimator) and obtain the distribution $p(\mathcal{D} \mid \mathcal{Z}_i; \theta)$ whose most probably values should be the $\mathcal{D}_i$ that generated the latent sample $\hat{\mathcal{Z}}$.

form as in previous chapters, and as before we resort to the pathwise gradient estimator (Appendix A.1). The famous alternative name reparameterisation trick comes exactly from the paper that introduced the VAE [10]. However, the authors first stated it in the more general form

$$\widehat{\text{ELBO}}_1(\Theta, \Psi) = \sum_{n=1}^{N} \sum_{i=1}^{T} \log p(\mathcal{X}_n, \mathcal{Z}_n^{(i)}; \Theta) - \log q(\mathcal{Z}_n^{(i)} \mid \mathcal{X}_n; \Psi), \tag{4.8}$$

where $\mathcal{Z}_n^{(i)} = g(\epsilon^{(i)}, \mathcal{X}_n; \Psi)$ is a deterministic transformation and $\epsilon^{(i)}$ a base random variable $\epsilon^{(i)} \sim p(\epsilon)$.

The above estimator is equivalent to (3.12) put forth in [13] for BNNs. Choosing families of distributions for $p(\mathcal{Z})$ and for $q(\mathcal{Z} \mid \mathcal{D}; \Psi)$ for which there exists a closed-form analytical formula for the KL in (4.7), there is no need to calculate it numerically, and we rewrite the above estimator as

$$\widehat{\text{ELBO}}_2(\Theta, \Psi) = \sum_{n=1}^{N} \sum_{i=1}^{T} \log p(\mathcal{X}_n \mid \mathcal{Z}_n^{(i)}; \Theta) - D_{KL}\left(q(\mathcal{Z}_n^{(i)} \mid \mathcal{X}_n; \Psi) \| p(\mathcal{Z})\right), \tag{4.9}$$

which is the form we have used throughout Chapter 3. Figure 3.4 still is a valid depiction of the reparameterisation trick, the sole difference being that here the random variables are $\mathcal{Z}$ instead of $\mathcal{W}$.

The main technical contribution of [10] is introducing for the first time (2013) in the Deep Learnig community this reparameterisation trick to achieve a low-variance gradient estimator, the vastly used VAE model is simply an use-case example of this estimator the authors offer midway through the paper [10]. For the generic formulation of Figure 4.1a optimised with (4.8), the authors name the algorithm Autoencoding Variational Bayes (AEVB). Here we consider the most common instantiation of VAE: $q(\mathcal{Z} \mid \mathcal{X}; \Psi)$ and $p(\mathcal{X} \mid \mathcal{Z}; \Theta)$ both implemented with feedforward NNs. Still, we could implement the same general model with other blocks, such as autoregressive models.

Suppose the a priori $p(\mathcal{Z})$ to be a centred diagonal Gaussian distribution, $\mathcal{N}(\mathbf{0}, \mathbf{I})$, and the likelihood function $p(\mathcal{X} \mid \mathcal{Z}; \Theta)$ a distribution with parameters determined by an NN, Gaussian for real-valued $\mathcal{X}$ and Bernoulli for binary. In addition, we approximate the posteriori with an approximately Gaussian distribution with diagonal covariance matrix, even though there is no specific reason to believe the true density has this form since it is an infinite mixture of Gaussians and can be arbitrarily complex. These choices for the distributions are not at all due to restrictions in the algorithm, but rather motivated by their simplicity. These settings lead us to an objective function similar to (3.6) for the KL term, where $\sigma_p = 1$ and $\mu_p = 0$. The deterministic transformation $g(\epsilon; \mathcal{X}_n, \Psi)$ is then

$$g(\epsilon; \mathcal{X}, \Psi) = \mu(f(\mathcal{X}; \Psi)) + \sigma(f(\mathcal{X}; \Psi)) \odot \epsilon, \tag{4.10}$$

with $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $f(\cdot; \Psi)$ the recognition model. Although we employ the same transformation all the time, it does not mean it is the only possible one, it just happens that location-scale transformations of the standard distribution of a given family are simple and practical. Another viable option, for example, is the to specify g as the inverse CDF of the desired distribution and $\epsilon \sim U(\mathbf{0}, \mathbf{1})$.

Experimentally, the authors [10] verified that when using mini-batch optimisation with size M, one sample from the approximate posterior $\mathcal{Z}^{(1)} \sim q(\mathcal{Z} \mid \mathcal{X}; \Theta)$ is enough so as long M is large enough, e.g., 100. Nevertheless, it has become common for practitioners to use $L = 1$ even when the mini-batch size is not large enough because of the computational gains, what may cause longer training time, i.e., more iterations.

Algorithm 7: VAE (or more generally, AEVB algorithm)

1: **while** not converged **do**
2: Randomly sample a data example $\mathcal{X}_i$
3: Randomly sample ϵ from the base distribution $p(\epsilon)$
4: Compute the gradients of the ELBO estimator w.r.t. Θ and Ψ
5: Update the parameters Θ and Ψ using the gradients
6: **end while**

4.3.1 Conditional VAE

The vanilla VAE model does not allow us to constrain the generated sample to have a particular characteristic: one should relentlessly draw samples until obtaining the desired feature; which restricts its usefulness in practical applications. For example, an ordinary task would be automatically colouring a person's hair in a photograph prior to dyeing it. The question then arises on how to capacitate the model to create

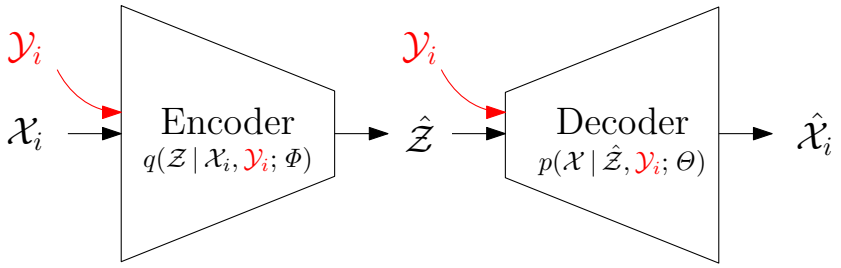


Figure 4.3: Schematic illustration of the CVAE model. Note that this is basically the same as the VAE, the sole difference being the inclusion of additional conditioning information $\mathcal{Y}$ to the input $\mathbf{x}$ and the sampled latent variable $\mathbf{z}$. The former so that the recognition model can infer the distribution corresponding to that condition, and the latter so that the generation network also knows to which condition that distribution refers.

targeted samples rather than completely random ones.

What we really wish is to *condition* the model output on some kind of information $\mathcal{Y}$, hence the name Conditional VAE (CVAE) [104]. Thus, the aim becomes to maximise $\log p(\mathcal{X}_i \mid \mathcal{Y}_i)$ for each observed variable $\mathcal{X}_i$, which following the same derivation as in (4.6) and (4.7) leads to

$$\begin{aligned} \log p(\mathcal{X}_i \mid \mathcal{Y}_i; \Theta) &\geqslant \mathbb{E}_{q(\mathcal{Z} \mid \mathcal{X}_i, \mathcal{Y}_i; \Psi)} \left[p(\mathcal{X}_i, \mathcal{Z} \mid \mathcal{Y}_i; \Theta) - q(\mathcal{Z} \mid \mathcal{X}_i, \mathcal{Y}_i; \Psi) \right] \\ &= \mathbb{E}_{q(\mathcal{Z} \mid \mathcal{X}_i, \mathcal{Y}_i; \Psi)} \left[\log p(\mathcal{X}_i \mid \mathcal{Z}, \mathcal{Y}_i; \Theta) \right] \\ &\quad - D_{KL} \left(q(\mathcal{Z} \mid \mathcal{X}_i, \mathcal{Y}_i; \Psi) \| p(\mathcal{Z} \mid \mathcal{X}_i, \mathcal{Y}_i; \Theta) \right). \end{aligned} \tag{4.11}$$

Recalling that for the VAE, $q(\mathcal{Z}_i \mid \mathcal{X}_i; \Psi)$ is a model, i.e., a NN, with input $\mathcal{X}_i$ and output $\mathcal{Z}_i$, it becomes evident that for its conditional counterpart $q(\mathcal{Z}_i \mid \mathcal{X}_i, \mathcal{Y}_i; \Psi)$ we must just add $\mathcal{Y}_i$ as input. Similar reasoning applies to the generator model. Thus by just concatenating the condition $\mathcal{Y}_i$, e.g., the data label, to both the input and latent space, while still maintaining all the other aspects of VAE unchanged, we obtain a CVAE [104].

From an implementation perspective, we can encode category information as a one-hot representation, indicating to the model which class is that input (or latent code) about. Intuitively, the prior gets split into different regions, each corresponding to a specific label, which gives us the ability to choose among them. In addition, by separating the samples into different classes, the data points within the same category become more similar, enhancing the modelling capacity and sample quality of CVAEs.

4.4 VAE Issues

VAEs are a great tool for generative modelling but they are not without shortcomings. In what follows we succinctly present the main known problems so far.

4.4.1 Inexpressive Posterior

The independent Gaussian assumption for the posterior limits the expressiveness of the model, which coupled with the mean-squared error from the log-likelihood term in the objective function (4.7) causes the generated samples to be blurry. More complex distributions help the model attain better marginal log-likelihood and simulated samples. Normalising flow transformations have been researched as a way of improving variational inference [38, 105].

4.4.2 The Posterior Collapse

Excessively powerful generators models p often lead to a state commonly known as "posterior collapse", where the model is said to ignore the latent variables. In effect, the distributions under optimisation degenerate to $q(\mathcal{Z} \mid \mathcal{X}; \Psi) = p(\mathcal{Z} \mid \mathcal{X}, \Theta) = p(\mathcal{Z})\ \forall \mathcal{X}$, what means that the posterior carries no information on the sample $\mathcal{X}$ and, in particular, no meaningful structure [106]. Apparently, the cause for this problem is related to the learning dynamics at the beginning of training, which may encourage the model to ignore the latent encoding. Thus a simple yet effective modification to the training procedure is proposed in [106].

4.4.3 Continuous Distributions

VAEs rely on the pathwise gradient estimator to enable the computation of gradients through random nodes in computational graphs and the usage of automatic differentiation tools. However, this estimator assumes the distribution to be continuous and cannot be used for the discrete case, which restraints our modelling capacity. Categorical distributions for the latent space for example are outside VAEs can do. Nevertheless, there are works on how to reparameterise discrete random variables by relaxing them into continuous distributions [107, 108], and use the pathwise estimator to obtain low-variance biased gradient estimates of the objective function. Still, the cost of these methods is the introduction of a new temperature parameter that should be anealled during training.

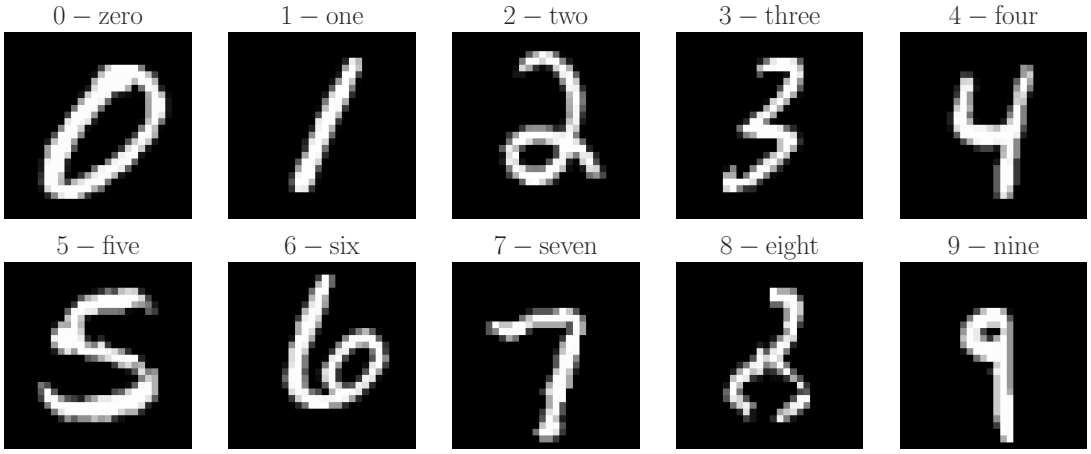


Figure 4.4: Mosaic of the 10 different classes of the MNIST dataset

4.5 Experiments

Although VAEs can generate any kind of data in principle, we use it on images to illustrate our points and keep the discussion attractive. This type of data has great appeal, is intuitive, and has broad support on modern programming frameworks, i.e., Pytorch [82].

4.5.1 Data Sets

We evaluate the VAE and CVAE algorithms on both MNIST [109], the classical 10-class handwritten digits, and Fashion-MNIST [110], a recently proposed data set to supersede the former.

MNIST

The MNIST data set is composed of 60,000 training and 10,000 testing 28×28 grayscale images of handwritten digits. Each sample depicts a single digit out the 10 possibilities. Figure 4.4 presents a one example of each class.

Uniform Manifold Approximation and Projection (UMAP) [111] can be used as an out-of-the-box visualisation tool similar to t-SNE [112], while being faster, better scaling to high dimensions and better preserving aspects of global structure of the data. Using this dimension reduction technique, we observe in Figure 4.5 the structure of MNIST data set. It has well-defined clusters for all of its classes.

Fashion-MNIST

Fashion-MNIST possess the same general layout: 60,000 training and 10,000 test samples, 28×28 grayscale images, and 10 possible exclusive classes. However, each

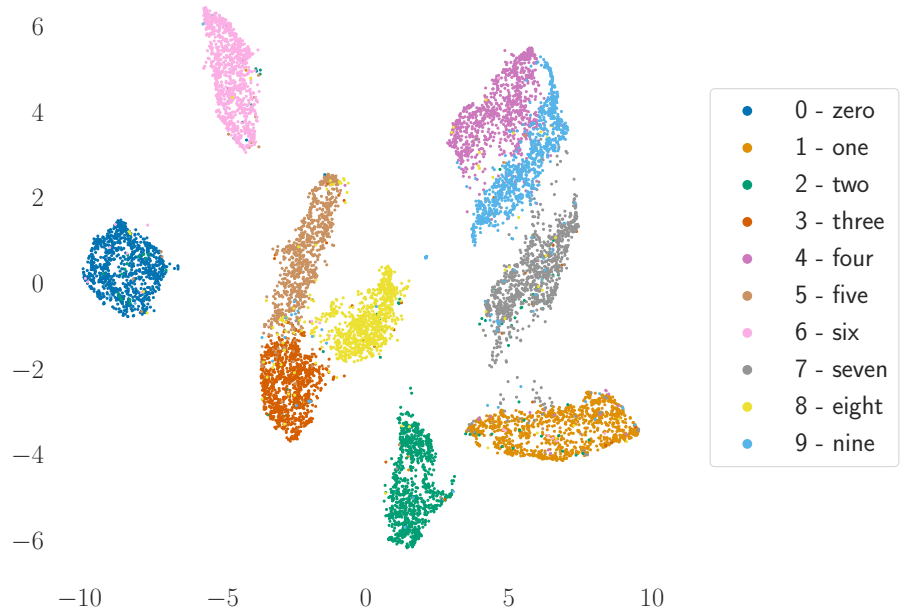


Figure 4.5: UMAP 2D-projection of the raw pixel space of MNIST

class is associated with a different piece of clothing as shown in Figure 4.6.

Standard ML algorithms obtain over 97% classification accuracy, and deep learning models over 99%, so there is no space left for researchers to evaluate if observed performance improvements are statistically relevant. Hence, MNIST cannot be used any more for benchmarking, and is no longer representative of modern computer vision tasks. Still, it has been employed in recent years mainly as a toy data set to do sanity checks and algorithm prototyping. Although easy, Fashion-MNIST is not as easy (90% and 95% accuracy for standard ML methods and deep learning, respectively) and still has margin for improvements.

Comparing the raw pixel structure of Fahion-MNIST data set, shown in in Figure 4.7 with that from MNIST in Figure 4.5, we note that the former has clusters corresponding to garments for the same body region partially overlapping.

4.5.2 Experimental setup

We train both the VAE and CVAE with varying latent space sizes d. We implement the encoder and generator of all models as fully-connected networks with ReLU activations [61], and Gaussian distributions for the latent spaces. Since the MNIST data set is simpler we use 1 hidden layer for both the encoder and the generator, whereas for the FashionMNIST we use 2. Hence the constructed models have the structure:

- MNIST: $784 \to 200 \to d \to 200 \to 784$;

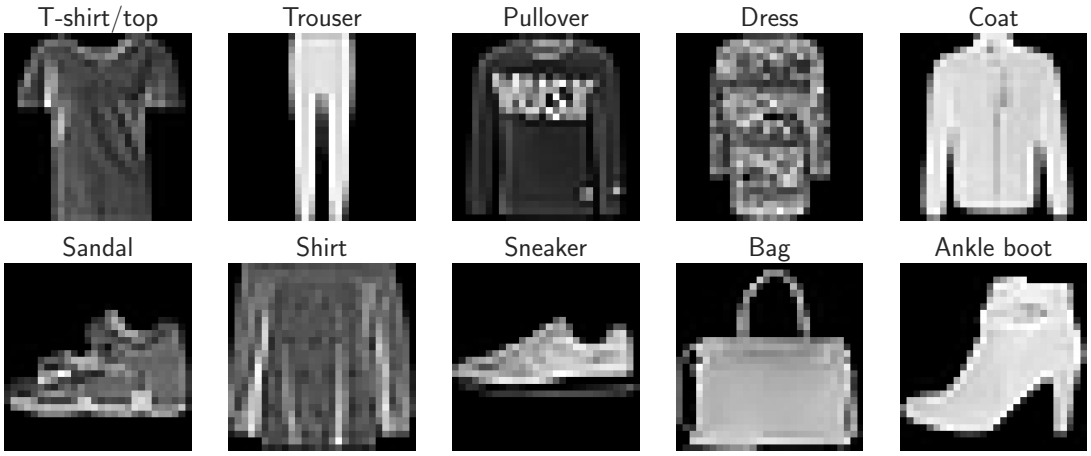


Figure 4.6: Mosaic of the 10 different classes of the Fashion-MNIST dataset

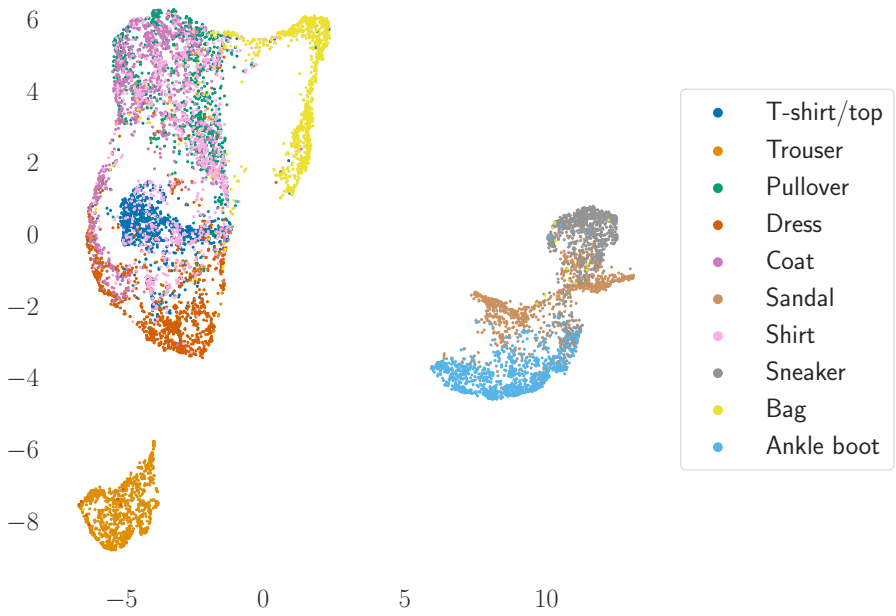


Figure 4.7: UMAP 2D-projection of the raw pixel space of Fashion-MNIST

- FashionMNIST: $784 \to 400 \to 200 \to d \to 200 \to 400 \to 784$.

Although the decoders are mirrored versions of the encoders, there is no special reason other than personal taste.

We model the likelihood function as a Bernoulli random variable and thus use binary cross entropy for the log-likelihood term of the objective. Additionally, we use mini-batches of size 128, 1 MC sample of the latent space for each input example, and train for 40 epochs with Adam [71] using a learning rate of 0.001.

4.5.3 Results

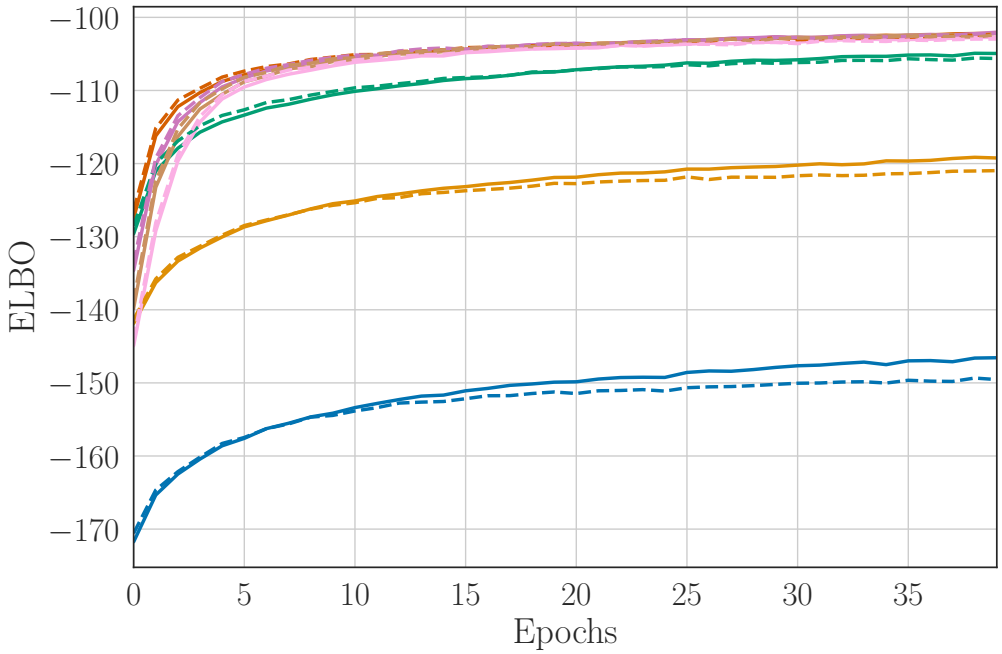
We observe from Figures 4.9 and **??** a similar behaviour across all models during training.

Increasing the latent dimension brings expressive gains when the size is too low, because we cannot properly represent the different features of the data set. Furthermore, increasing it more than necessary, that is, having an excessively large latent space has almost no negative effect on performance: carefully inspecting all 4 graphics, allows us to see that the ELBO for the 200-dimensional models are slightly lower than 20-dimensional model. Moreover, the train-test gap remains more or less constant regardless of the latent dimension size. Thus, we conclude VAEs are robust to overfitting, at least with respect to the size of the latent space.

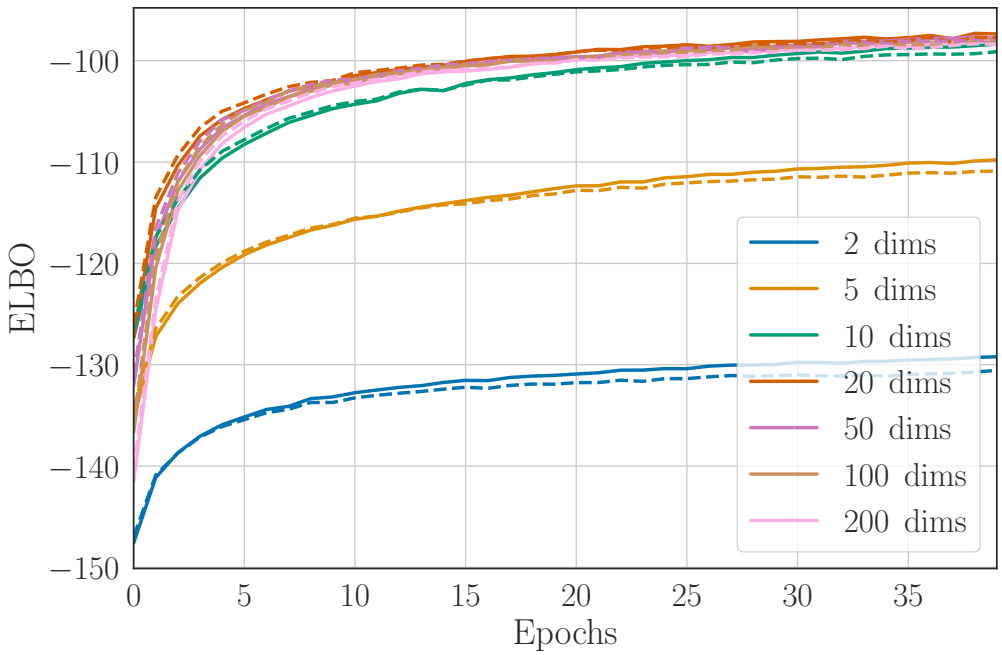
The KL divergence remains stable throughout the whole training, varying very little, as shown in Figure 4.10. The learned posterior distribution moves away from the prior within the first epoch and stays at approximately the same "distance" during the rest of the optimisation procedure. This is a consequence of the powerful regularising effect the KL term has on the model. Although we only exhibit the case of the CVAE trained on MNIST, this is a general behaviour observed in all experiments.

FashionMNIST models have remarkably worse results, indeed we can confirm it visually by observing the generated samples from Figure 4.11. While the MNIST samples are not very rich in details, this is not true for the FashionMNIST objects, thus the VAE struggles to recover such fine details. Even though CVAE does a better job at generating new images, what we can immediately observe by comparing Figures 4.11a and 4.11b, for the FashionMNIST data the sample quality of images in Figure 4.11d are only marginally better than those of the VAE model, displayed at Figure 4.11c. The quality of both models is overall poor if compared to the original samples of Figure 4.6.

This is actually a general characteristic of VAEs, originating from the objective that seeks to minimise the *average* log-likelihood of data and causes the reconstructed (and generated) samples to be blurry. On average they may be good, but

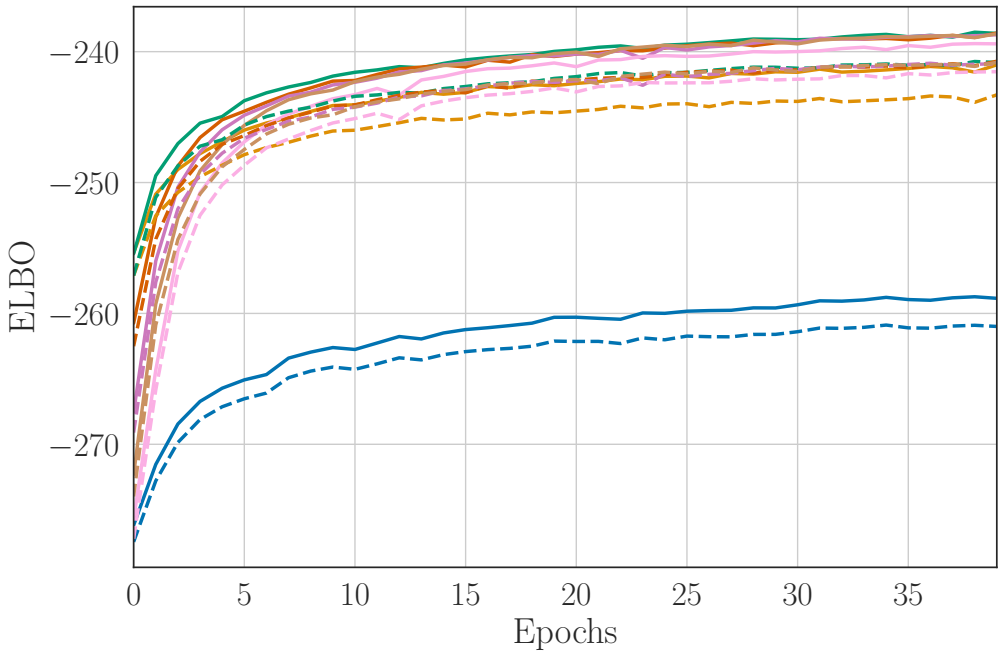


(a) VAE model

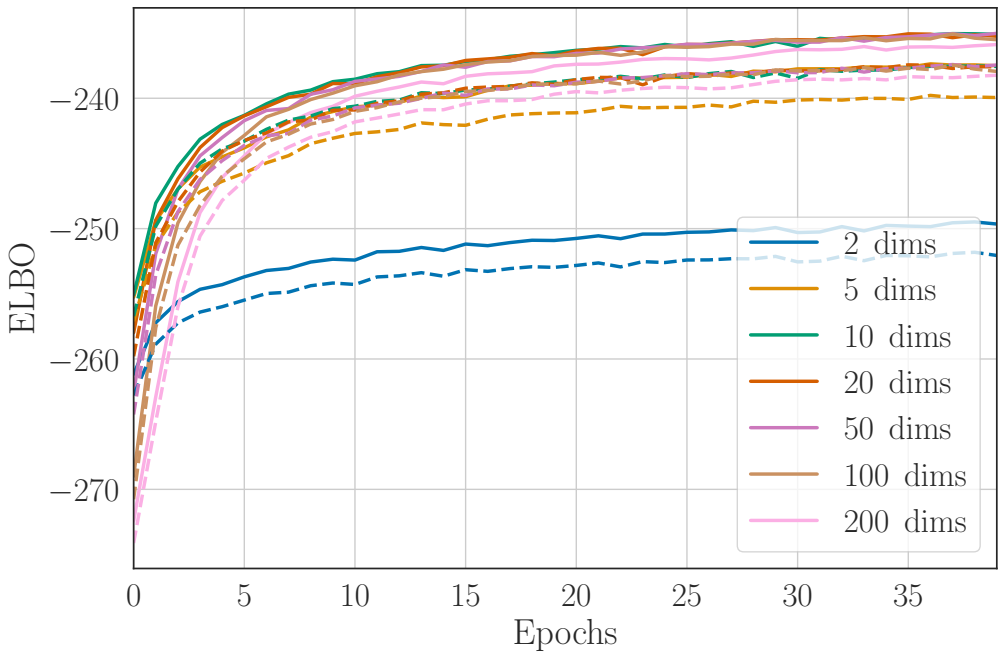


(b) CVAE model

Figure 4.8: Training and evaluation ELBO for the MNIST data set for different sizes of the latent dimension space. The continuous curves represent the training performance, while dashed ones the test.



(a) VAE model



(b) CVAE model

Figure 4.9: Training and evaluation ELBO for the FashionMNIST data set for different sizes of the latent dimension space. The continuous curves represent the training performance, while dashed ones the test.

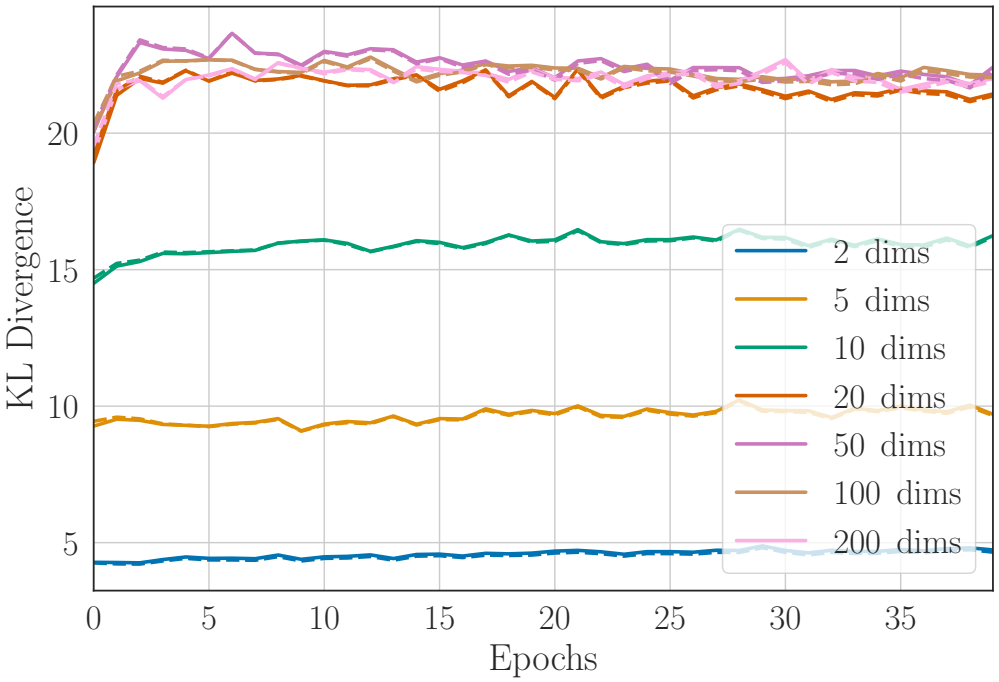
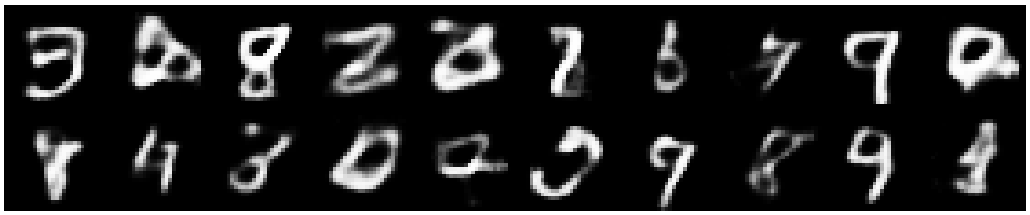


Figure 4.10: Training and evaluation KL divergence curve of CVAE models with different latent dimension sizes in the MNIST data set.

individually they are not sharp. This effect is more pronounced in FashionMNIST as clothes are more diverse and have more details than digits, which is also the cause for the greater difficulty of FashionMNIST over MNIST, and precisely what we observe in Figure 4.11. In order to achieve better log-likelihood and sample quality, we need to employ better generator and encoder models, plain fully-connected networks have pretty much been replaced nowadays by convolutional architectures in the image domain.

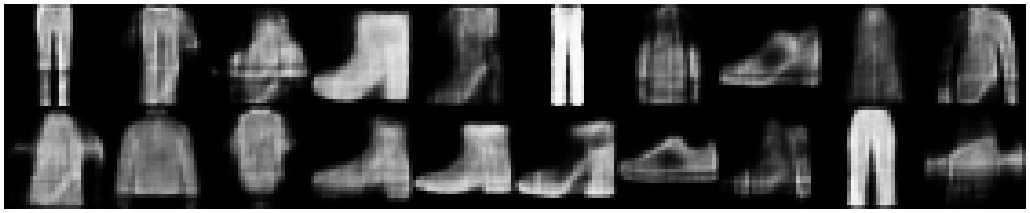
In both cases, MNIST and FashionMNIST, we note the latent space of the CVAE model to be unstructured according to Figures 4.12b and 4.13b, respectively. However, this is not exactly true because it becomes segmented after conditioning. All we see here is the latent representation of the different labels superposed, but the act of conditioning selects which distribution to observe, what gives the model liberty to use the "full" latent space to model the specified object class, that is why the log-likelihood of the CVAE is invariably higher than its VAE counterpart for the same latent size. We use quotation marks for full because when conditioning by concatenating the one-hot representation of the chosen class, we are effectively augmenting the latent space by 10 dimensions.

Interestingly, we note from Figure 4.13a that samples from the digits 4 and 9, as well as 5 and 3 are generally overlapped, which indicates the model cannot properly tell them apart. A classifier built from the the latent feature space would have a poor accuracy for samples from these classes. Similarly, and as already observed in

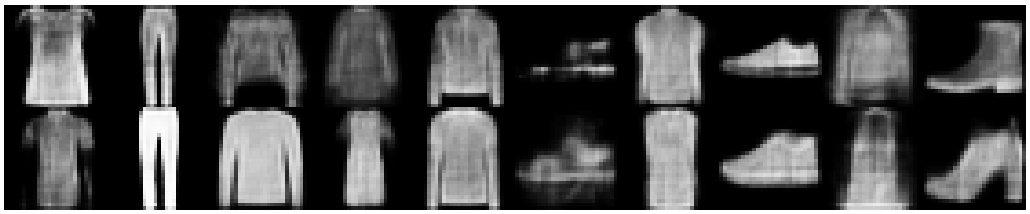


(a) VAE model - MNIST

(b) CVAE model - MNIST



(c) VAE model - FashionMNIST



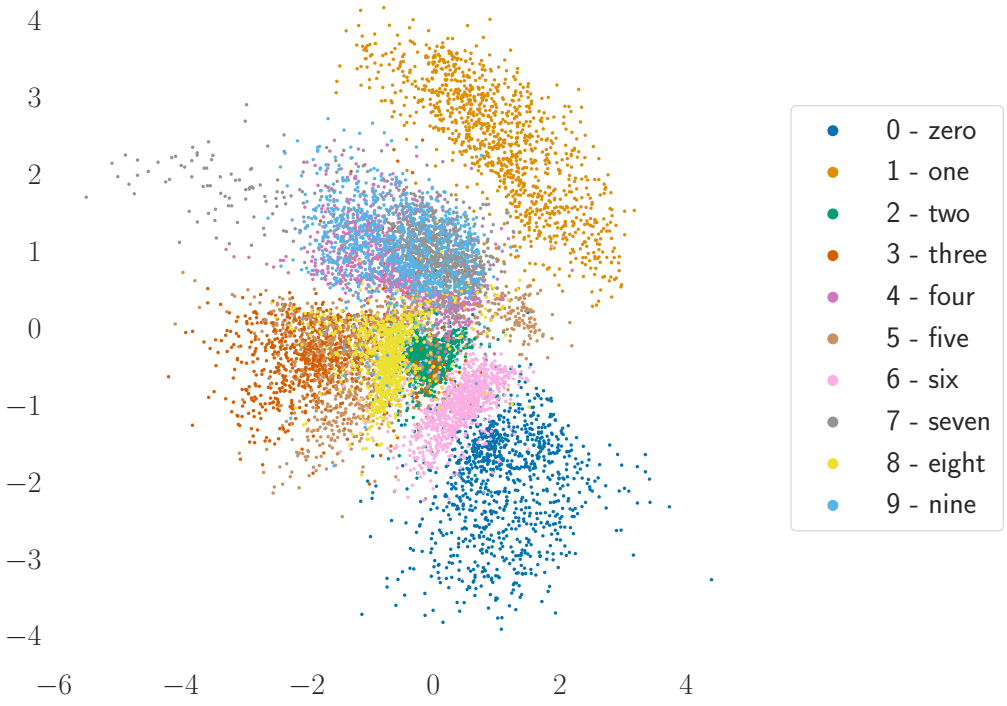
(d) CVAE model - FashionMNIST

Figure 4.11: Samples generated by the 20D-latent-space VAE and CVAE models.

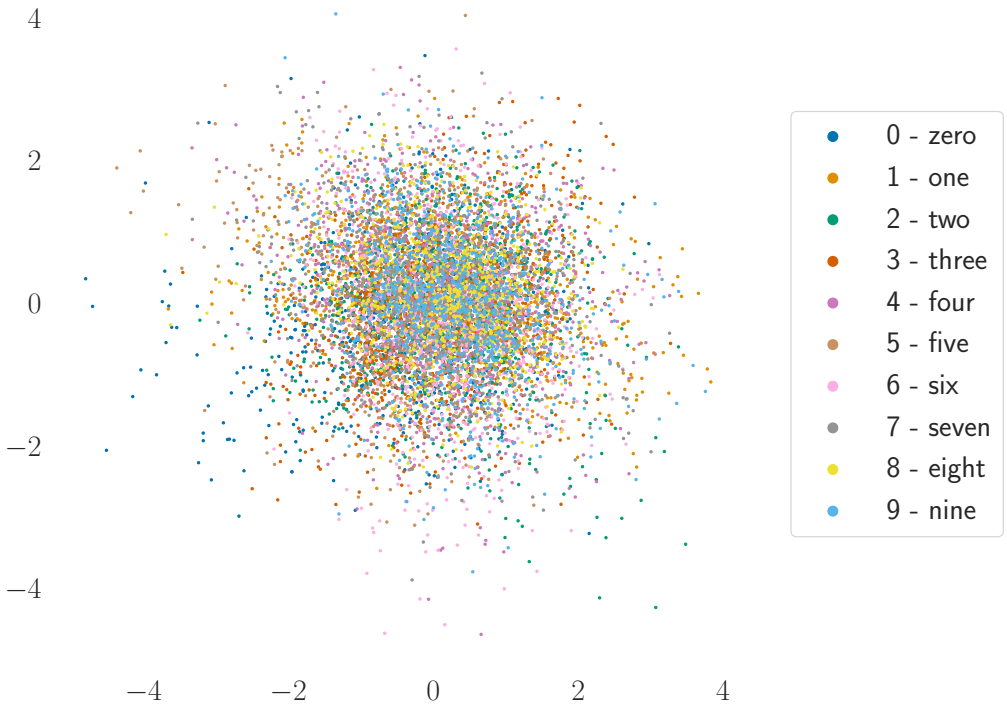
Figure ??, the pullover, coat, and shirt classes are mostly distributed on the same region of the latent space, which intuitively makes sense since they are designed for the same body part and, thus, have similar shapes. From this, we can conclude the inference network was not capable of identifying the distinctive features of those classes, in accordance with the previous discussion of our models not being powerful enough.

4.6 Closing remarks

In this chapter we discussed vanilla and conditional VAEs, and how the models arise by applying VI to the latent variable modelling. Moreover, we presented their major drawbacks and some of the research work being developed to diminish those issues. There exists several other types of generative algorithms, the most popular being Generative Adversarial Networks [113], whose training dynamic can also be interpreted as through probabilistic lenses.

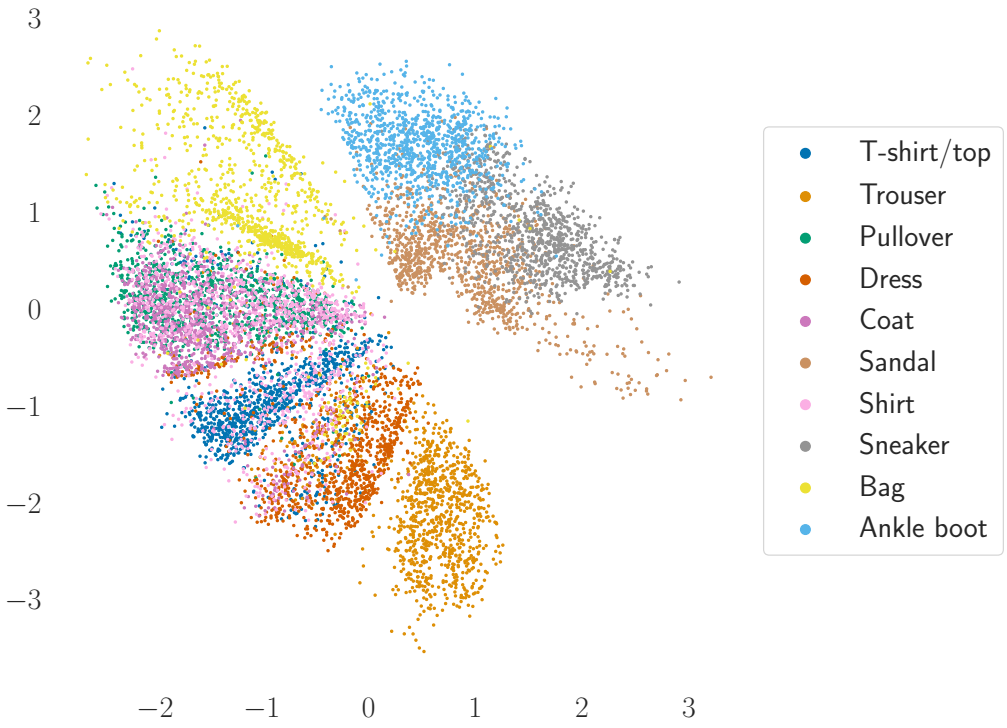


(a) VAE model

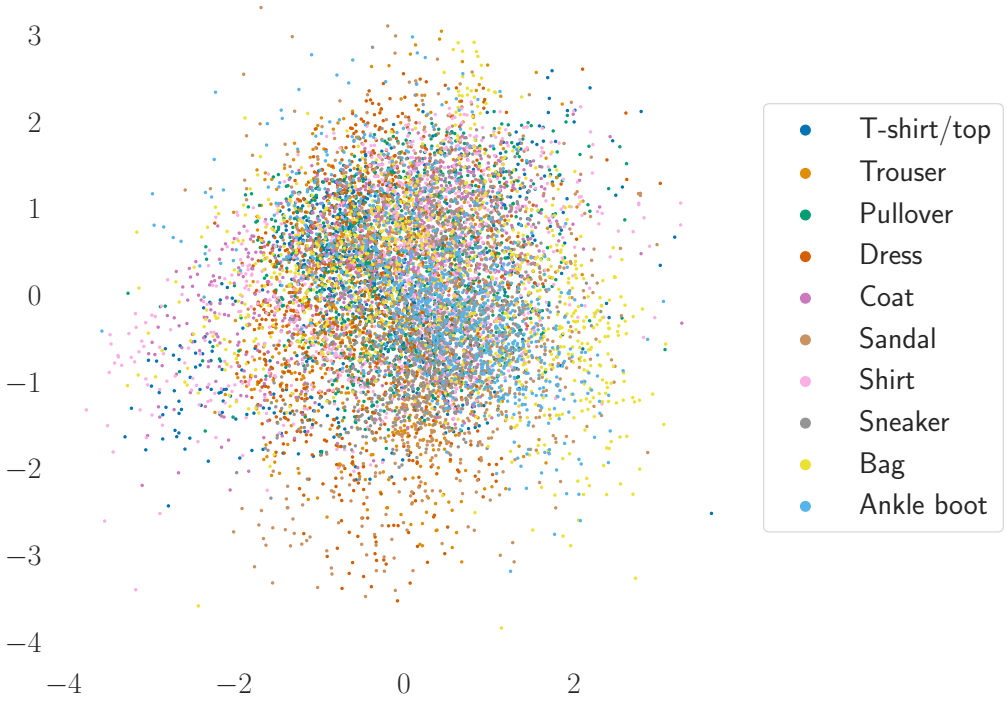


(b) CVAE model

Figure 4.12: Visualisation of the 2D latent space for MNIST data set for varying latent dimensionality.



(a) VAE model



(b) CVAE model

Figure 4.13: Training and evaluation ELBO for the FashionMNIST data set for different sizes of the latent dimension space. The continuous curves represent the training performance, while dashed ones the test.

Chapter 5

Conclusions and Future Work

Throughout the dissertation we have seen the value of the Bayesian approach to probabilistic modelling and how it seamlessly allows us see to reason under uncertainty, make predictions, and simulate new data, all through marginalisation and conditioning. Moreover, model fitting and comparison naturally arises within the framework and it constitutes a good contender better data-poor regimes, being more robust to overfitting.

However, the calculation the method entails rarely have closed-formed solution for complex models and requires computational methods to approximate it. We then discussed variational methods, a class of deterministic approximations that elicits reasonable performance on a viable amount of time, as opposed to stochastic methods that get prohibitively expensive in high dimensions. Specifically, we examined VI, ADF, and EP, and briefly mentioned other general purpose practical extensions of these three techniques.

Empowered with these new concepts and motivations, we tackled the problem of (approximate) posterior inference in BNNs. Instead of estimating a real-value scalar for each weight in the network, we sought to establish entire PDFs over the possible values of the parameters. As we saw, the additional computational cost can be considerably minimised at the expense of performance, thus constituting an important trade-off one should consider when developing applications, as well as analysing the complexity required to implementing custom solutions.

Next, instead of using approximate inference to estimate the posterior distribution over the weights as in BNNs, we used it to estimate the posterior over the latent space of our observables. This led us to a generative model resembling an autoencoder, hence its name VAE, that could do both approximate inference given new data and generate new samples through ancestral sampling. Furthermore, thanks to the regularisation effect of the variational objective, the latent space of the model has structure and allow us to interpolate and combine samples. Still, VAE has several drawbacks, which the scientific community is currently studying and working

to alleviate. Although briefly, we presented most of these problems together with references for the dedicated reader.

5.1 Future Research

5.1.1 Combining Generative Models and Uncertainty

Research on either generative models or BNN alone is virtually infinite, and may involve theoretical aspects or applications. However, presenting them both together has an underlying motive, which is to develop algorithms for practical applications using generative models that estimate the uncertainty of their predictions. More specifically, the unsupervised task of future video frame prediction. This task requires exceedingly good efficiency since dealing with videos is computationally demanding by itself, further imposing posterior inference instead of MLE severely complicates the issue. Additionally requiring the predictions to be near real time, otherwise the future will have long happened, makes the challenge yet more remarkable even with parallel computing in Graphical Processing Units made possible.

Furthermore, Bayesian prediction of future scenes has been largely unexplored to this date, with only one work published so far [114].

5.1.2 Long-term Ambitions

The above application is itself a milestone of a much far-fetched objective, regarding neuroscience and artificial intelligence. Uncertainty is a natural way of thinking under the dubious interpretation of sensory informations, and indeed behavioural studies support that humans perform nearly optimal Bayesian inference, efficiently integrating multi-sensory information while being energetically efficient. Bayesian approximate predictions of the future in videos is an instantiation of this question, one of particular interest to researchers marvelled by the visual system.

Interaction between neuroscience and artificial intelligence is a two-way road and both may benefit from leaning and observing the other. For example, we can question ourselves on how to bring stunning capacities of the brain such as few-shot learning, energy-efficiency, concept abstraction, continual learning, and causality to an artificial system. On the other hand, we can port our insights from practical issues and simulations to neuroscience and search for similarities on the brain. A growing body of research investigates whether something similar to deep learning happens inside our brains [115–117]

Bibliography

[1] HOMAN, M. D., GELMAN, A. "The No-U-turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo", *J. Mach. Learn. Res.*, v. 15, n. 1, pp. 1593–1623, jan. 2014. ISSN: 1532-4435. Disponível em: <http://dl.acm.org/citation.cfm?id=2627435.2638586>.

[2] CHEN, M., MAO, S., LIU, Y. "Big Data: A Survey", *Mobile Networks and Applications*, v. 19, n. 2, pp. 171–209, Apr 2014. ISSN: 1572-8153. doi: 10.1007/s11036-013-0489-0. Disponível em: <https://doi.org/10.1007/s11036-013-0489-0>.

[3] HINTON, G., DENG, L., YU, D., et al. "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups", *IEEE Signal Processing Magazine*, v. 29, n. 6, pp. 82–97, Nov 2012. ISSN: 1053-5888. doi: 10.1109/MSP.2012.2205597.

[4] KRIZHEVSKY, A., SUTSKEVER, I., HINTON, G. E. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pp. 1097–1105, USA, 2012. Curran Associates Inc. Disponível em: <http://dl.acm.org/citation.cfm?id=2999134.2999257>.

[5] GOODFELLOW, I. J., SHLENS, J., SZEGEDY, C. "Explaining and harnessing adversarial examples". In: *Proceedings of the 3rd International Conference on Learning Representations*, May 7–9 2015.

[6] PLANCK COLLABORATION, ADE, P. A. R., AGHANIM, N., et al. "Planck 2015 results. XIII. Cosmological parameters", *A&A*, 594:A13, set. 2016. doi: 10.1051/0004-6361/201525830.

[7] BLEI, D. M., KUCUKELBIR, A., MCAULIFFE, J. D. "Variational Inference: A Review for Statisticians", *Journal of the American Statistical Association*, v. 112, n. 518, pp. 859–877, 2017. doi: 10.1080/01621459.2017.

1285773. Disponível em: <https://doi.org/10.1080/01621459.2017.1285773>.

[8] GRAVES, A. "Practical Variational Inference for Neural Networks". In: *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, pp. 2348–2356, USA, 2011. Curran Associates Inc. ISBN: 978-1-61839-599-3. Disponível em: <http://dl.acm.org/citation.cfm?id=2986459.2986721>.

[9] SOUDRY, D., HUBARA, I., MEIR, R. "Expectation Backpropagation: Parameter-free Training of Multilayer Neural Networks with Continuous or Discrete Weights". In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, pp. 963–971, Cambridge, MA, USA, 2014. MIT Press. Disponível em: <http://dl.acm.org/citation.cfm?id=2968826.2968934>.

[10] KINGMA, D. P., WELLING, M. "Auto-encoding variational bayes". In: *Proceedings of the 2nd International Conference on Learning Representations*, 14–16 April 2014. Disponível em: <https://arxiv.org/pdf/1312.6114.pdf>.

[11] RANGANATH, R., GERRISH, S., BLEI, D. "Black Box Variational Inference". In: Kaski, S., Corander, J. (Eds.), *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, v. 33, *Proceedings of Machine Learning Research*, pp. 814–822, Reykjavik, Iceland, 22–25 April 2014. PMLR. Disponível em: <http://proceedings.mlr.press/v33/ranganath14.html>.

[12] NEAL, R. M. *Bayesian Learning for Neural Networks*. Tese de Doutorado, Toronto, 1996.

[13] BLUNDELL, C., CORNEBISE, J., KAVUKCUOGLU, K., et al. "Weight Uncertainty in Neural Networks". In: *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pp. 1613–1622. JMLR.org, 2015. Disponível em: <http://dl.acm.org/citation.cfm?id=3045118.3045290>.

[14] HERNÁNDEZ-LOBATO, J. M., ADAMS, R. P. "Probabilistic Backpropagation for Scalable Learning of Bayesian Neural Networks". In: *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pp. 1861–1869. JMLR.org, 2015. Disponível em: <http://dl.acm.org/citation.cfm?id=3045118.3045316>.

[15] GAL, Y., GHAHRAMANI, Z. "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning". In: Balcan, M. F., Weinberger, K. Q. (Eds.), *Proceedings of The 33rd International Conference on Machine Learning*, v. 48, *Proceedings of Machine Learning Research*, pp. 1050–1059, New York, New York, USA, 20–22 June 2016. PMLR. Disponível em: <http://proceedings.mlr.press/v48/gal16.html>.

[16] KHAN, M., NIELSEN, D., TANGKARATT, V., et al. "Fast and Scalable Bayesian Deep Learning by Weight-Perturbation in Adam". In: Dy, J., Krause, A. (Eds.), *Proceedings of the 35th International Conference on Machine Learning*, v. 80, *Proceedings of Machine Learning Research*, pp. 2611–2620, Stockholmsmässan, Stockholm Sweden, 10–15 July 2018. PMLR. Disponível em: <http://proceedings.mlr.press/v80/khan18a.html>.

[17] BISHOP, C. M. "Model-based machine learning", *Phil. Trans. R. Soc. A*, v. 371, n. 1984, pp. 20120222, 2013.

[18] MOHAMED, S. "Planting the Seeds of Probabilistic Thinking: Foundations, Tricks and Algorithms". Tutorial presentation, August 2018.

[19] DENG, J., DONG, W., SOCHER, R., et al. "Imagenet: A large-scale hierarchical image database". In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. Ieee, 2009.

[20] MIGON, H. S., GAMERMAN, D., LOUZADA, F. *Statistical inference: an integrated approach*. CRC press, 2014.

[21] DIACONIS, P., FREEDMAN, D. "Finite Exchangeable Sequences", *The Annals of Probability*, v. 8, n. 4, pp. 745–764, 1980. ISSN: 00911798. Disponível em: <http://www.jstor.org/stable/2242823>.

[22] KOLLER, D., FRIEDMAN, N., BACH, F. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[23] YEDIDIA, J. S. "Message-Passing Algorithms for Inference and Optimization", *Journal of Statistical Physics*, v. 145, n. 4, pp. 860–890, Nov 2011. ISSN: 1572-9613. doi: 10.1007/s10955-011-0384-7. Disponível em: <https://doi.org/10.1007/s10955-011-0384-7>.

[24] VAN DE MEENT, J.-W., PAIGE, B., YANG, H., et al. "An Introduction to Probabilistic Programming", *arXiv e-prints*, art. arXiv:1809.10756, September 2018.

[25] GHAHRAMANI, Z. "Probabilistic machine learning and artificial intelligence", *Nature*, v. 521, n. 7553, pp. 452, 2015.

[26] VIDAKOVIC, B. "Bayesian Inference Using Gibbs Sampling – BUGS Project". In: *Statistics for Bioengineering Sciences: With MATLAB and WinBUGS Support*, pp. 733–745, New York, NY, Springer New York, 2011. ISBN: 978-1-4614-0394-4. doi: 10.1007/978-1-4614-0394-4_19. Disponível em: <https://doi.org/10.1007/978-1-4614-0394-4_19>.

[27] GOODMAN, N. D., STUHLMÜLLER, A. "The Design and Implementation of Probabilistic Programming Languages". http://dippl.org, 2014. Accessed: 2019-3-23.

[28] MINKA, T., WINN, J., GUIVER, J., et al. "/Infer.NET 0.3". 2018. Microsoft Research Cambridge. http://dotnet.github.io/infer.

[29] CARPENTER, B., GELMAN, A., HOFFMAN, M., et al. "Stan: A Probabilistic Programming Language", *Journal of Statistical Software, Articles*, v. 76, n. 1, pp. 1–32, 2017. ISSN: 1548-7660. doi: 10.18637/jss.v076.i01. Disponível em: <https://www.jstatsoft.org/v076/i01>.

[30] SALVATIER J, WIECKI TV, F. C. "Probabilistic programming in Python using PyMC3", *PeerJ Computer Science*, v. 55, n. 2, 2016.

[31] BINGHAM, E., CHEN, J. P., JANKOWIAK, M., et al. "Pyro: Deep Universal Probabilistic Programming", *Journal of Machine Learning Research*, 2018.

[32] TRAN, D., KUCUKELBIR, A., DIENG, A. B., et al. "Edward: A library for probabilistic modeling, inference, and criticism", *arXiv preprint arXiv:1610.09787*, 2016.

[33] TUKEY, J. W. "The future of data analysis", *The annals of mathematical statistics*, v. 33, n. 1, pp. 1–67, 1962.

[34] MACKAY, D. J., MAC KAY, D. J. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[35] HOFFMAN, M., BLEI, D. "Stochastic Structured Variational Inference". In: Lebanon, G., Vishwanathan, S. V. N. (Eds.), *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, v. 38, *Proceedings of Machine Learning Research*, pp. 361–369, San Diego, California, USA, 09–12 May 2015. PMLR. Disponível em: <http://proceedings.mlr.press/v38/hoffman15.html>.

[36] SUN, S., CHEN, C., CARIN, L. "Learning Structured Weight Uncertainty in Bayesian Neural Networks". In: Singh, A., Zhu, J. (Eds.), *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, v. 54, *Proceedings of Machine Learning Research*, pp. 1283–1292, Fort Lauderdale, FL, USA, 20–22 Apr 2017. PMLR. Disponível em: <http://proceedings.mlr.press/v54/sun17b.html>.

[37] LOUIZOS, C., WELLING, M. "Structured and Efficient Variational Deep Learning with Matrix Gaussian Posteriors". In: Balcan, M. F., Weinberger, K. Q. (Eds.), *Proceedings of The 33rd International Conference on Machine Learning*, v. 48, *Proceedings of Machine Learning Research*, pp. 1708–1716, New York, New York, USA, 20–22 Jun 2016. PMLR. Disponível em: <http://proceedings.mlr.press/v48/louizos16.html>.

[38] REZENDE, D., MOHAMED, S. "Variational Inference with Normalizing Flows". In: Bach, F., Blei, D. (Eds.), *Proceedings of the 32nd International Conference on Machine Learning*, v. 37, *Proceedings of Machine Learning Research*, pp. 1530–1538, Lille, France, 07–09 Jul 2015. PMLR. Disponível em: <http://proceedings.mlr.press/v37/rezende15.html>.

[39] MINKA, T. P. "Expectation Propagation for Approximate Bayesian Inference". In: *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence*, UAI '01, pp. 362–369, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN: 1-55860-800-1. Disponível em: <http://dl.acm.org/citation.cfm?id=647235.720257>.

[40] BISHOP, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg, Springer-Verlag, 2006. ISBN: 0387310738.

[41] MINKA, T. *Power EP*. Relatório técnico, January 2004. Disponível em: <https://www.microsoft.com/en-us/research/publication/power-ep/>.

[42] MINKA, T., OTHERS. *Divergence measures and message passing*. Relatório técnico, Microsoft Research, 2005.

[43] ROBBINS, H., MONRO, S. "A Stochastic Approximation Method", *The Annals of Mathematical Statistics*, v. 22, n. 3, pp. 400–407, 1951. ISSN: 21682909. doi: 10.1109/TSMC.1971.4308316.

[44] HOFFMAN, M. D., BLEI, D. M., WANG, C., et al. "Stochastic Variational Inference", *Journal of Machine Learning Research*, v. 14, pp. 1303–1347.

[45] WILLIAMS, R. J. "Simple statistical gradient-following algorithms for connectionist reinforcement learning", *Machine learning*, v. 8, n. 3, pp. 229–256, May . ISSN: 1573-0565. doi: 10.1007/BF00992696. Disponível em: <https://doi.org/10.1007/BF00992696>.

[46] PRICE, R. "Natural Gradient Works Efficiently in Learning", *IRE Transactions on Information Theory*, v. 4, n. 2, pp. 69–72, June 1958. doi: 10.1109/TIT.1958.1057444.

[47] HERNANDEZ-LOBATO, J., LI, Y., ROWLAND, M., et al. "Black-Box Alpha Divergence Minimization". In: Balcan, M. F., Weinberger, K. Q. (Eds.), *Proceedings of The 33rd International Conference on Machine Learning*, v. 48, *Proceedings of Machine Learning Research*, pp. 1511–1520, New York, New York, USA, 20–22 June 2016. PMLR. Disponível em: <http://proceedings.mlr.press/v48/hernandez-lobatob16.html>.

[48] MINKA, T. "The EP Energy Function and Minimization Schemes". August 2007. Disponível em: <https://www.microsoft.com/en-us/research/publication/ep-energy-function-minimization-schemes/>.

[49] KUCUKELBIR, A., BLEI, D., GELMAN, A., et al. "Automatic Differentiation Variational Inference", *Journal of Machine Learning Research*, v. 18, pp. 1–45, January 2017. ISSN: 1532-4435.

[50] TISHBY, N., LEVIN, E., SOLLA, S. A. "Consistent inference of probabilities in layered networks: Predictions and generalization". In: *International Joint Conference on Neural Networks*, v. 2, pp. 403–409. IEEE, December 1989.

[51] MACKAY, D. J. C. "A Practical Bayesian Framework for Backpropagation Networks", *Neural Comput.*, v. 4, n. 3, pp. 448–472, May 1992. ISSN: 0899-7667. doi: 10.1162/neco.1992.4.3.448. Disponível em: <http://dx.doi.org/10.1162/neco.1992.4.3.448>.

[52] NEAL, R. M. "Bayesian Learning via Stochastic Dynamics". In: *Advances in Neural Information Processing Systems 5, [NIPS Conference]*, pp. 475–482, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc. ISBN: 1-55860-274-7. Disponível em: <http://dl.acm.org/citation.cfm?id=645753.667903>.

[53] HINTON, G. E., VAN CAMP, D. "Keeping the Neural Networks Simple by Minimizing the Description Length of the Weights". In: *Proceedings of the Sixth Annual Conference on Computational Learning Theory*, COLT '93, pp. 5–13, New York, NY, USA, 1993. ACM. ISBN: 0-89791-611-5. doi: 10.1145/168304.168306. Disponível em: <http://doi.acm.org/10.1145/168304.168306>.

[54] HINTON, G. E., OSINDERO, S., TEH, Y.-W. "A Fast Learning Algorithm for Deep Belief Nets", *Neural Computation*, v. 18, n. 7, pp. 1527–1554, jul. 2006. ISSN: 0899-7667. doi: 10.1162/neco.2006.18.7.1527. Disponível em: <http://dx.doi.org/10.1162/neco.2006.18.7.1527>.

[55] RUSSAKOVSKY, O., DENG, J., SU, H., et al. "ImageNet Large Scale Visual Recognition Challenge", *International Journal of Computer Vision (IJCV)*, v. 115, n. 3, pp. 211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[56] WELLING, M., TEH, Y. W. "Bayesian Learning via Stochastic Gradient Langevin Dynamics". In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, pp. 681–688, USA, 2011. Omnipress. ISBN: 978-1-4503-0619-5. Disponível em: <http://dl.acm.org/citation.cfm?id=3104482.3104568>.

[57] PAPERNOT, N., MCDANIEL, P., GOODFELLOW, I., et al. "Practical Black-Box Attacks Against Machine Learning". In: *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, pp. 506–519, New York, NY, USA, 2017. ACM. ISBN: 978-1-4503-4944-4. doi: 10.1145/3052973.3053009. Disponível em: <http://doi.acm.org/10.1145/3052973.3053009>.

[58] KULESHOV, V., FENNER, N., ERMON, S. "Accurate Uncertainties for Deep Learning Using Calibrated Regression". In: Dy, J., Krause, A. (Eds.), *Proceedings of the 35th International Conference on Machine Learning*, v. 80, *Proceedings of Machine Learning Research*, pp. 2796–2804, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. Disponível em: <http://proceedings.mlr.press/v80/kuleshov18a.html>.

[59] OPPER, M., ARCHAMBEAU, C. "The Variational Gaussian Approximation Revisited", *Neural Computation*, v. 21, n. 3, pp. 786–792, 2009. doi: 10.1162/neco.2008.08-07-592. Disponível em: <https://doi.org/10.1162/neco.2008.08-07-592>. PMID: 18785854.

[60] GHOSH, S., FAVE, F. D., YEDIDIA, J. "Assumed Density Filtering Methods for Learning Bayesian Neural Networks". 12–17 February

2016. Disponível em: <https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12391>.

[61] NAIR, V., HINTON, G. E. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pp. 807–814, USA, 2010. Omnipress. ISBN: 978-1-60558-907-7. Disponível em: <http://dl.acm.org/citation.cfm?id=3104322.3104425>.

[62] HERBRICH, R. "Minimising the Kullback-Leibler Divergence", August 2005. Disponível em: <https://www.microsoft.com/en-us/research/publication/minimising-the-kullback-leibler-divergence/>.

[63] SRIVASTAVA, N., HINTON, G., KRIZHEVSKY, A., et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", *J. Mach. Learn. Res.*, v. 15, n. 1, pp. 1929–1958, January 2014. ISSN: 1532-4435. Disponível em: <http://dl.acm.org/citation.cfm?id=2627435.2670313>.

[64] KINGMA, D. P., SALIMANS, T., WELLING, M. "Variational Dropout and the Local Reparameterization Trick". In: Cortes, C., Lawrence, N. D., Lee, D. D., et al. (Eds.), *Advances in Neural Information Processing Systems 28*, Curran Associates, Inc., pp. 2575–2583, 2015. Disponível em: <http://papers.nips.cc/paper/5666-variational-dropout-and-the-local-reparameterization-trick.pdf>.

[65] WAN, L., ZEILER, M., ZHANG, S., et al. "Regularization of Neural Networks using DropConnect". In: Dasgupta, S., McAllester, D. (Eds.), *Proceedings of the 30th International Conference on Machine Learning*, v. 28, *Proceedings of Machine Learning Research*, pp. 1058–1066, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. Disponível em: <http://proceedings.mlr.press/v28/wan13.html>.

[66] GAL, Y. *Uncertainty in Deep Learning*. Tese de Doutorado, University of Cambridge, 2016.

[67] GAL, Y., GHAHRAMANI, Z. "A Theoretically Grounded Application of Dropout in Recurrent Neural Networks". In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016*, pp. 1019–1027, 5–10 December 2016. Disponível em: <http://papers.nips.cc/paper/6241-a-theoretically-grounded-application-of-dropout-in-recurrent-neural-networks>.

[68] HRON, J., MATTHEWS, A., GHAHRAMANI, Z. "Variational Bayesian dropout: pitfalls and fixes". In: Dy, J., Krause, A. (Eds.), *Proceedings of the 35th International Conference on Machine Learning*, v. 80, *Proceedings of Machine Learning Research*, pp. 2019–2028, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. Disponível em: <http://proceedings.mlr.press/v80/hron18a.html>.

[69] AMARI, S.-I. "Natural Gradient Works Efficiently in Learning", *Neural Compututation*, v. 10, n. 2, pp. 251–276, February 1998. ISSN: 0899-7667. doi: 10.1162/089976698300017746. Disponível em: <http://dx.doi.org/10.1162/089976698300017746>.

[70] KHAN, M., LIN, W. "Conjugate-Computation Variational Inference : Converting Variational Inference in Non-Conjugate Models to Inferences in Conjugate Models". In: Singh, A., Zhu, J. (Eds.), *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, v. 54, *Proceedings of Machine Learning Research*, pp. 878–887, Fort Lauderdale, FL, USA, 20–22 Apr 2017. PMLR. Disponível em: <http://proceedings.mlr.press/v54/khan17a.html>.

[71] KINGMA, D. P., BA, J. "Adam: A method for stochastic optimization", *arXiv preprint arXiv:1412.6980*, 2014.

[72] BOTTOU, L., CURTIS, F. E., NOCEDAL, J. "Optimization Methods for Large-Scale Machine Learning", *SIAM Review*, v. 60, n. 2, pp. 223–311, 2018. doi: 10.1137/16M1080173. Disponível em: <https://doi.org/10.1137/16M1080173>.

[73] AMARI, S.-I. "Natural Gradient Learning and Its Dynamics in Singular Regions". In: *Information Geometry and Its Applications*, pp. 279–314, Tokyo, Springer Japan, 2016. ISBN: 978-4-431-55978-8. doi: 10.1007/978-4-431-55978-8_12. Disponível em: <https://doi.org/10.1007/978-4-431-55978-8_12>.

[74] HONKELA, A., TORNIO, M., RAIKO, T., et al. "Natural Conjugate Gradient in Variational Inference". In: Ishikawa, M., Doya, K., Miyamoto, H., et al. (Eds.), *Neural Information Processing*, pp. 305–314, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN: 978-3-540-69162-4.

[75] TIELEMAN, T., HINTON, G. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude". 2012.

[76] DUCHI, J., HAZAN, E., SINGER, Y. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization", *J. Mach. Learn. Res.*, v. 12, pp. 2121–2159, jul. 2011. ISSN: 1532-4435. Disponível em: <http://dl.acm.org/citation.cfm?id=1953048.2021068>.

[77] DUA, D., GRAFF, C. "UCI Machine Learning Repository". 2017. Disponível em: <http://archive.ics.uci.edu/ml>.

[78] ZHANG, G., SUN, S., DUVENAUD, D., et al. "Noisy Natural Gradient as Variational Inference". In: Dy, J., Krause, A. (Eds.), *Proceedings of the 35th International Conference on Machine Learning*, v. 80, *Proceedings of Machine Learning Research*, pp. 5852–5861, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. Disponível em: <http://proceedings.mlr.press/v80/zhang18l.html>.

[79] SNOEK, J., LAROCHELLE, H., ADAMS, R. P. "Practical Bayesian Optimization of Machine Learning Algorithms". In: *Proceedings of the 25th International Conference on Neural Information Processing Systems*, v. 2, *NIPS'12*, pp. 2951–2959, USA, 2012. Curran Associates Inc. Disponível em: <http://dl.acm.org/citation.cfm?id=2999325.2999464>.

[80] RASMUSSEN, C. E., WILLIAMS, C. K. I. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005. ISBN: 026218253X.

[81] THEANO DEVELOPMENT TEAM. "Theano: A Python framework for fast computation of mathematical expressions", *arXiv e-prints*, v. abs/1605.02688, maio 2016. Disponível em: <http://arxiv.org/abs/1605.02688>.

[82] PASZKE, A., GROSS, S., CHINTALA, S., et al. "Automatic differentiation in PyTorch". In: *NIPS-W*, 2017.

[83] BENATAN, M., DARESBURY, S.-T., PYZER-KNAPP, E. O. "Practical Considerations for Probabilistic Backpropagation". In: *Third workshop on Bayesian Deep Learning (NeurIPS 2018)*, 2018.

[84] MISHKIN, A., KUNSTNER, F., NIELSEN, D., et al. "SLANG: Fast Structured Covariance Approximations for Bayesian Deep Learning with Natural Gradient". In: Bengio, S., Wallach, H., Larochelle, H., et al. (Eds.), *Advances in Neural Information Processing Systems 31*, Curran Associates, Inc., pp. 6245–6255, 2018. Disponível em: <http://papers.nips.cc/paper/7862-slang-fast-structured-

covariance-approximations-for-bayesian-deep-learning-with-natural-gradient.pdf>.

[85] NEAL, R. M. "MCMC using Hamiltonian dynamics". In: *Handbook of Markov Chain Monte Carlo*, 1 ed., CRC Press, cap. 5, pp. 113–162, Boca Raton, Florida, USA, 7 2011.

[86] LI, C., CHEN, C., CARLSON, D., et al. "Preconditioned Stochastic Gradient Langevin Dynamics for Deep Neural Networks". In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pp. 1788–1794. AAAI Press, 2016. Disponível em: <http://dl.acm.org/citation.cfm?id=3016100.3016149>.

[87] KORATTIKARA, A., RATHOD, V., MURPHY, K., et al. "Bayesian Dark Knowledge". In: *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pp. 3438–3446, Cambridge, MA, USA, 2015. MIT Press. Disponível em: <http://dl.acm.org/citation.cfm?id=2969442.2969623>.

[88] LEE, A. X., ZHANG, R., EBERT, F., et al. "Stochastic Adversarial Video Prediction", *arXiv preprint arXiv:1804.01523*, 2018.

[89] KALCHBRENNER, N., VAN DEN OORD, A., SIMONYAN, K., et al. "Video Pixel Networks". In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pp. 1771–1779. JMLR.org, 2017. Disponível em: <http://dl.acm.org/citation.cfm?id=3305381.3305564>.

[90] HOUTHOOFT, R., CHEN, X., CHEN, X., et al. "VIME: Variational Information Maximizing Exploration". In: Lee, D. D., Sugiyama, M., Luxburg, U. V., et al. (Eds.), *Advances in Neural Information Processing Systems 29*, Curran Associates, Inc., pp. 1109–1117, 2016. Disponível em: <http://papers.nips.cc/paper/6591-vime-variational-information-maximizing-exploration.pdf>.

[91] HA, D., SCHMIDHUBER, J. "Recurrent World Models Facilitate Policy Evolution". In: Bengio, S., Wallach, H., Larochelle, H., et al. (Eds.), *Advances in Neural Information Processing Systems 31*, Curran Associates, Inc., pp. 2450–2462, 2018. Disponível em: <http://papers.nips.cc/paper/7512-recurrent-world-models-facilitate-policy-evolution.pdf>.

[92] ESLAMI, S. M. A., JIMENEZ REZENDE, D., BESSE, F., et al. "Neural scene representation and rendering", *Science*, v. 360, n. 6394, pp. 1204–1210, 2018. ISSN: 0036-8075. doi: 10.1126/science.aar6170. Disponível em: <https://science.sciencemag.org/content/360/6394/1204>.

[93] ESLAMI, S. M. A., HEESS, N., WEBER, T., et al. "Attend, Infer, Repeat: Fast Scene Understanding with Generative Models". In: Lee, D. D., Sugiyama, M., Luxburg, U. V., et al. (Eds.), *Advances in Neural Information Processing Systems 29*, Curran Associates, Inc., pp. 3225–3233, 2016. Disponível em: <http://papers.nips.cc/paper/6230-attend-infer-repeat-fast-scene-understanding-with-generative-models.pdf>.

[94] LEDIG, C., THEIS, L., HUSZAR, F., et al. "Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 22–25 July 2017. Disponível em: <http://openaccess.thecvf.com/content_cvpr_2017/papers/Ledig_Photo-Realistic_Single_Image_CVPR_2017_paper.pdf>.

[95] TSCHANNEN, M., AGUSTSSON, E., LUCIC, M. "Deep Generative Models for Distribution-Preserving Lossy Compression". In: Bengio, S., Wallach, H., Larochelle, H., et al. (Eds.), *Advances in Neural Information Processing Systems 31*, Curran Associates, Inc., pp. 5929–5940, 4–6 December 2018. Disponível em: <http://papers.nips.cc/paper/7833-deep-generative-models-for-distribution-preserving-lossy-compression.pdf>.

[96] CHEN, J., CHEN, J., CHAO, H., et al. "Image Blind Denoising With Generative Adversarial Network Based Noise Modeling". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 19–21 June 2018.

[97] VAN DEN OORD, A., DIELEMAN, S., ZEN, H., et al. "WaveNet: A Generative Model for Raw Audio". In: *9th ISCA Speech Synthesis Workshop*, pp. 125–125, September 13–15 2016.

[98] GÓMEZ-BOMBARELLI, R., WEI, J. N., DUVENAUD, D., et al. "Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules", *ACS Central Science*, v. 4, n. 2, pp. 268–276, 2018. doi: 10.1021/acscentsci.7b00572. Disponível em: <https://doi.org/10.1021/acscentsci.7b00572>.

[99] RIESSELMAN, A. J., INGRAHAM, J. B., MARKS, D. S. "Deep generative models of genetic variation capture the effects of mutations", *Nature Methods*, v. 15, pp. 816–822, October 2018. doi: 10.1038/s41592-018-0138-4. Disponível em: <https://doi.org/10.1038/s41592-018-0138-4>.

[100] THEIS, L., OORD, A. V. D., BETHGE, M. "A note on the evaluation of generative models". In: *Proceedings of the 4th International Conference on Learning Representations*, 2–4 May 2016.

[101] SALIMANS, T., GOODFELLOW, I., ZAREMBA, W., et al. "Improved Techniques for Training GANs". In: Lee, D. D., Sugiyama, M., Luxburg, U. V., et al. (Eds.), *Advances in Neural Information Processing Systems 29*, Curran Associates, Inc., pp. 2234–2242, 2016. Disponível em: <http://papers.nips.cc/paper/6125-improved-techniques-for-training-gans.pdf>.

[102] HEUSEL, M., RAMSAUER, H., UNTERTHINER, T., et al. "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium". In: Guyon, I., Luxburg, U. V., Bengio, S., et al. (Eds.), *Advances in Neural Information Processing Systems 30*, Curran Associates, Inc., pp. 6626–6637, 2017. Disponível em: <http://papers.nips.cc/paper/7240-gans-trained-by-a-two-time-scale-update-rule-converge-to-a-local-nash-equilibrium.pdf>.

[103] ROSCA, M., LAKSHMINARAYANAN, B., WARDE-FARLEY, D., et al. "Variational approaches for auto-encoding generative adversarial networks", *arXiv preprint arXiv:1706.04987*, 2017.

[104] SOHN, K., LEE, H., YAN, X. "Learning Structured Output Representation using Deep Conditional Generative Models". In: Cortes, C., Lawrence, N. D., Lee, D. D., et al. (Eds.), *Advances in Neural Information Processing Systems 28*, Curran Associates, Inc., pp. 3483–3491, 2015. Disponível em: <http://papers.nips.cc/paper/5775-learning-structured-output-representation-using-deep-conditional-generative-models.pdf>.

[105] KINGMA, D. P., SALIMANS, T., JOZEFOWICZ, R., et al. "Improved Variational Inference with Inverse Autoregressive Flow". In: Lee, D. D., Sugiyama, M., Luxburg, U. V., et al. (Eds.), *Advances in Neural Information Processing Systems 29*, Curran Associates, Inc., pp. 4743–4751, 2016. Disponível em: <http://papers.nips.cc/paper/6581-improved-variational-inference-with-inverse-autoregressive-flow.pdf>.

[106] HE, J., SPOKOYNY, D., NEUBIG, G., et al. "Lagging Inference Networks and Posterior Collapse in Variational Autoencoders". In: *Proceedings of the 8th International Conference on Learning Representations*, 6–9 May 2019. Disponível em: <https://openreview.net/forum?id=rylDfnCqF7>.

[107] MADDISON, C., MNIH, A., TEH, Y. W. "The Concrete Distribution: a Continuous Relaxation of Discrete Random Variables". In: *Proceedings of the 5th International Conference on Learning Representations*, 24–26 April 2017.

[108] JANG, E., GU, S., POOLE, B. "Categorical Reparameterization with Gumbel-Softmax". In: *Proceedings of the 5th International Conference on Learning Representations*, 24 – 26 April 2017.

[109] LECUN, Y., BOTTOU, L., BENGIO, Y., et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE*, pp. 2278–2324, 1998.

[110] XIAO, H., RASUL, K., VOLLGRAF, R. "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms", *arXiv e-prints*, art. arXiv:1708.07747, Aug 2017.

[111] MCINNES, L., HEALY, J., MELVILLE, J. "UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction", *arXiv e-prints*, art. arXiv:1802.03426, Feb 2018.

[112] MAATEN, L. V. D., HINTON, G. "Visualizing data using t-SNE", *Journal of machine learning research*, v. 9, n. Nov, pp. 2579–2605, 2008.

[113] GOODFELLOW, I., POUGET-ABADIE, J., MIRZA, M., et al. "Generative Adversarial Nets". In: Ghahramani, Z., Welling, M., Cortes, C., et al. (Eds.), *Advances in Neural Information Processing Systems 27*, Curran Associates, Inc., pp. 2672–2680, 2014. Disponível em: <http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf>.

[114] BHATTACHARYYA, A., FRITZ, M., SCHIELE, B. "Bayesian Prediction of Future Street Scenes using Synthetic Likelihoods". In: *Proceedings of the 8th International Conference on Learning Representations*, 6–9 May 2019. Disponível em: <https://openreview.net/forum?id=rkgK3oC5Fm>.

[115] MARBLESTONE, A. H., WAYNE, G., KORDING, K. P. "Toward an Integration of Deep Learning and Neuroscience", *Frontiers in Computational Neuroscience*, v. 10, pp. 94, 2016. ISSN: 1662-5188. doi: 10.3389/fncom.

2016.00094. Disponível em: <https://www.frontiersin.org/article/10.3389/fncom.2016.00094>.

[116] GUERGUIEV, J., LILLICRAP, T. P., RICHARDS, B. A. "Towards deep learning with segregated dendrites", *eLife*, v. 6, pp. e22901, dec 2017. ISSN: 2050-084X. doi: 10.7554/eLife.22901. Disponível em: <https://doi.org/10.7554/eLife.22901>.

[117] BARTUNOV, S., SANTORO, A., RICHARDS, B., et al. "Assessing the Scalability of Biologically-Motivated Deep Learning Algorithms and Architectures". In: Bengio, S., Wallach, H., Larochelle, H., et al. (Eds.), *Advances in Neural Information Processing Systems 31*, Curran Associates, Inc., pp. 9368–9378, 2018. Disponível em: <http://papers.nips.cc/paper/8148-assessing-the-scalability-of-biologically-motivated-deep-learning-algorithms-and-architectures.pdf>.

[118] PASCANU, R., BENGIO, Y. "Revisiting Natural Gradient for Deep Networks". In: *Proceedings of the 2nd International Conference on Learning Representations*, April 14 – 16 2014.

Appendix A

Knowledge Goodies

A.1 Gradient estimators

In inference problems, as well as in other domains, we frequently encounter the computation of $\nabla_\phi \mathbb{E}_{q(z;\phi)}[f(z;\theta)]$. This is the gradient w.r.t. ϕ of the expectation of the function $f(z;\theta)$ under the distribution $q(z;\phi)$, with θ and ϕ being their parameters respectively. Generally, we cannot compute this gradient directly because the expectation is intractable. Hence, we assume certain conditions so as to rewrite it and obtain appropriate practical estimators, which we approximate by Monte Carlo integration.

If $q(z;\phi)$ is known and it is a continuous function of ϕ, though not necessarily of z, we derive the *reinforce* or *score-function estimator* [45] through

$$\begin{aligned}
\nabla_\phi \mathbb{E}_{q(z;\phi)}[f(z;\theta)] &= \nabla_\phi \left[\int q(z;\phi) f(z;\theta) dz \right] \\
&= \int \nabla_\phi [q(z;\phi)] f(z;\theta) dz \\
&= \int q(z;\phi) \nabla_\phi [\log q(z;\phi)] f(z;\theta) dz \\
&= \mathbb{E}_{q(z;\phi)} [f(z;\theta) \nabla_\phi [\log q(z;\phi)]] . \qquad \text{(A.1)}
\end{aligned}$$

Note that third equality comes from the log derivative trick

$$\frac{\partial \log g(\xi)}{\partial \xi} = \frac{1}{g(\xi)} \frac{\partial g(\xi)}{\partial \xi} \qquad \text{(A.2)}$$

and that we made no assumptions about $f(z;\theta)$ and it may indeed be non-differentiable or even discrete.

If we, instead, express the random variable $z \sim q(z;\phi)$ as an invertible deterministic transformation $g(\cdot;\phi)$ of a base random variable $\epsilon \sim p(\epsilon)$, we arrive at the

pathwise derivative estimator [46]

$$\begin{aligned}\nabla_\phi \mathbb{E}_{q(z;\phi)}\left[f(z;\theta)\right] &= \nabla_\phi \left[\int p(\epsilon) f(g(\epsilon;\phi);\theta) d\epsilon\right] \\ &= \nabla_\phi \left[\mathbb{E}_{p(\epsilon)}\left[f(g(\epsilon;\phi);\theta)\right]\right] \\ &= \mathbb{E}_{p(\epsilon)}\left[\nabla_\phi \left[f(g(\epsilon;\phi);\theta)\right]\right]. \end{aligned} \tag{A.3}$$

This approach requires not only the distribution $q(z;\phi)$ to be reparameterisable, but also $f(z;\theta)$ to be known and a continuous function of θ for all values of z. It is also known as *reparameterisation trick* and recently popularised by [10].

While both estimators yields unbiased estimates, the score function estimator generally has higher variance due to the derivative of the log. This behaviour makes sense if we think that it computes the derivative only on q and does not include information about the function $f(z;\theta)$, which is the objective function.

A.2 Natural Gradient and the Fisher Information Matrix

Here we shall motivate the natural gradient and explain how it appears in our optimisation context. The derivation closely follows [118]. For the sake of brevity, we use the notation $p_{\boldsymbol{\psi}}$ instead of the usual $p(\cdot;\boldsymbol{\psi})$ to indicate a family of distributions p parametrised by $\boldsymbol{\psi}$.

The update step in the gradient descent algorithm can vary abruptly at each iteration. Neither clipping nor fixing its magnitude guarantees a clear limit on the change induced in our model. Furthermore, the gradient depends on the coordinate system, so constraining its norm amounts to different restrictions in different coordinate systems. Logically, while for some distributions small changes in the parameters have large effects on the probability represented by the model, for others it is the opposite. If we wish to move along the functional manifold with constant speed, regardless of its curvature, we must impose a constraint on the probability manifold, that is, on the output space of our model.

Suppose we want to minimise the loss function $\mathcal{L}$ (or maximise if dealing with the ELBO) w.r.t. the distribution $p_{\boldsymbol{\psi}}$ parametrised by $\boldsymbol{\psi}$. Hence, at each iteration we wish to find the step

$$\begin{aligned} &\underset{\delta\boldsymbol{\psi}}{\operatorname{argmin}}\, \mathcal{L}(\boldsymbol{\psi} + \delta\boldsymbol{\psi}) \\ &\text{s.t. } D_{KL}(p_{\boldsymbol{\psi}} \| p_{\boldsymbol{\psi}+\delta\boldsymbol{\psi}}) = \text{const.} \end{aligned} \tag{A.4}$$

The KL divergence constraint assures that the output space, which is what effectively matters, will change by a constant value. Although not a proper metric, for small enough $\delta\boldsymbol{\psi}$ we can approximate it around $\boldsymbol{\psi}$ by its Taylor expansion up to the second order

$$D_{KL}(p_{\boldsymbol{\psi}} \| p_{\boldsymbol{\psi}+\delta\boldsymbol{\psi}}) \approx D_{KL}(p_{\boldsymbol{\psi}} \| p_{\boldsymbol{\psi}}) + \delta\boldsymbol{\psi}\, \nabla_{\boldsymbol{\psi}'} D_{KL}(p_{\boldsymbol{\psi}} \| p_{\boldsymbol{\psi}'})\big|_{\boldsymbol{\psi}'=\boldsymbol{\psi}} + \frac{1}{2}\delta\boldsymbol{\psi}^T H(\boldsymbol{\psi})\delta\boldsymbol{\psi} \tag{A.5}$$

where $H(\boldsymbol{\psi})$ is the Hessian of the D_{KL} computed at $\boldsymbol{\psi}$. Since the divergence has its minimum at $\delta\boldsymbol{\psi} = 0$, both the first and second right-hand terms in (A.5) vanish and only the second-order one remains. The Hessian is given by

$$\begin{aligned}
H(\boldsymbol{\psi}) &= -\nabla^2_{\boldsymbol{\psi}}\, \mathbb{E}\left[\log p_{\boldsymbol{\psi}}\right] = \mathbb{E}\left[-\nabla^2_{\boldsymbol{\psi}} \log p_{\boldsymbol{\psi}}\right] \\
&= \mathbb{E}_{p_{\boldsymbol{\psi}}}\left[-\nabla_{\boldsymbol{\psi}}\left[\frac{1}{p_{\boldsymbol{\psi}}}\nabla_{\boldsymbol{\psi}} p_{\boldsymbol{\psi}}^T\right]\right] \\
&= \mathbb{E}_{p_{\boldsymbol{\psi}}}\left[\frac{1}{p^2_{\boldsymbol{\psi}}}\nabla_{\boldsymbol{\psi}} p_{\boldsymbol{\psi}} \nabla_{\boldsymbol{\psi}} p_{\boldsymbol{\psi}}^T\right] - \mathbb{E}_{p_{\boldsymbol{\psi}}}\left[\frac{1}{p_{\boldsymbol{\psi}}}\nabla^2_{\boldsymbol{\psi}} p_{\boldsymbol{\psi}}\right] \\
&= \mathbb{E}\left[\nabla_{\boldsymbol{\psi}} \log p_{\boldsymbol{\psi}} \nabla_{\boldsymbol{\psi}} \log p_{\boldsymbol{\psi}}^T\right] - \int \nabla^2_{\boldsymbol{\psi}} p_{\boldsymbol{\psi}} d\mathcal{Z} \\
&= \mathbb{E}\left[\nabla_{\boldsymbol{\psi}} \log p_{\boldsymbol{\psi}} \nabla_{\boldsymbol{\psi}} \log p_{\boldsymbol{\psi}}^T\right] \\
&= \mathbb{E}\left[\nabla_{\boldsymbol{\psi}} \log p_{\boldsymbol{\psi}} \nabla_{\boldsymbol{\psi}} \log p_{\boldsymbol{\psi}}^T\right] - \underbrace{\mathbb{E}\left[\nabla_{\boldsymbol{\psi}} \log p_{\boldsymbol{\psi}}\right]}_{=0} \mathbb{E}\left[\nabla_{\boldsymbol{\psi}} \log p_{\boldsymbol{\psi}}^T\right] \\
&= \mathrm{Cov}(\nabla_{\boldsymbol{\psi}} \log p_{\boldsymbol{\psi}}, \nabla_{\boldsymbol{\psi}} \log p_{\boldsymbol{\psi}}) \\
&= \mathcal{I}(\boldsymbol{\psi}),
\end{aligned} \tag{A.6}$$

where $\mathcal{I}(\boldsymbol{\psi})$ is the Fisher information matrix and the last equality comes from the definition.

The Fisher matrix measures the variance in the distribution caused by changes in the parameter space. Intuitively, it carries the amount of information that the observed data $\mathcal{X}$ has about the parameters $\boldsymbol{\psi}$. From (A.6) we can appreciate that the Fisher matrix is the negative of the expected value of the Hessian of the log likelihood, thus $\mathcal{I}(\boldsymbol{\psi})$ encodes the curvature of the manifold.

From (A.5) and (A.6), we can write the Lagrangian form of (A.4) as

$$\underset{\delta\boldsymbol{\psi}}{\operatorname{argmin}}\, \mathcal{L}(\boldsymbol{\psi} + \delta\boldsymbol{\psi}) + \lambda\left(\frac{1}{2}\delta\boldsymbol{\psi}^T \mathcal{I}(\boldsymbol{\psi})\delta\boldsymbol{\psi} - \mathrm{const}\right), \tag{A.7}$$

If we further assume the linearisation of $\mathcal{L}(\boldsymbol{\psi}+\delta\boldsymbol{\psi})$ around the vicinity of $\boldsymbol{\psi}$ is valid,

we obtain

$$\underset{\delta\boldsymbol{\psi}}{\operatorname{argmin}}\, \mathcal{L}(\boldsymbol{\psi}) + \nabla_{\boldsymbol{\psi}}\mathcal{L}(\boldsymbol{\psi})^T\delta\boldsymbol{\psi} + \lambda\left(\frac{1}{2}\delta\boldsymbol{\psi}^T\mathcal{I}(\boldsymbol{\psi})\delta\boldsymbol{\psi} - \text{const}\right). \tag{A.8}$$

Finally, setting the gradient w.r.t. $\delta\boldsymbol{\psi}$ equal to zero in order to solve the problem, we arrive at

$$\begin{aligned} 0 &= \nabla_{\boldsymbol{\psi}}\mathcal{L}(\boldsymbol{\psi}) + \lambda\mathcal{I}(\boldsymbol{\psi})\delta\boldsymbol{\psi} \\ \lambda\mathcal{I}\delta\boldsymbol{\psi} &= -\nabla_{\boldsymbol{\psi}}\mathcal{L}(\boldsymbol{\psi}) \\ \delta\boldsymbol{\psi} &= -\frac{1}{\lambda}\mathcal{I}^{-1}(\boldsymbol{\psi})\nabla_{\boldsymbol{\psi}}\mathcal{L}(\boldsymbol{\psi}) \\ \delta\boldsymbol{\psi} &= -k\widetilde{\nabla}_{\boldsymbol{\psi}}\mathcal{L}(\boldsymbol{\psi}), \end{aligned} \tag{A.9}$$

where we define $\widetilde{\nabla}_{\boldsymbol{\psi}}\mathcal{L}(\boldsymbol{\psi}) \equiv \mathcal{I}^{-1}(\boldsymbol{\psi})\nabla_{\boldsymbol{\psi}}\mathcal{L}(\boldsymbol{\psi})$ as the natural gradient and $k = 1/\lambda$ as the step size (step factor would be the most appropriate name, however it is not a popular one).

We have thus obtained an algorithm that not only is robust to reparameterisations (the Fisher matrix is invariant to one-to-one transformations), and move along the manifold with constant speed, but also follows the steepest descent direction [69].

A.3 Gauss-Newton Approximation

The Gauss-Newton method is a classical approach for non-linear least-squares that approximates the Hessian of the (vectorial) function $f(\cdot)$ with its Jacobian J_f. When the objective function is not a sum of squares, but rather an arbitrary (scalar) function $\ell(\cdot)$, i.e., the negative logarithm, we do

$$\begin{aligned} \frac{\partial^2\ell(f(\mathbf{x}))}{\partial x_j\partial x_i} &= \frac{\partial}{\partial x_j}\left(\frac{\partial\ell(f(\mathbf{x}))}{\partial x_i}\right) \\ &= \frac{\partial}{\partial x_j}\left(\sum_{k=0}^{K}\frac{\partial\ell}{\partial f_k(\mathbf{x})}\frac{\partial f_k(\mathbf{x})}{\partial x_i}\right) \\ &= \sum_{k=0}^{K}\frac{\partial}{\partial x_j}\left(\frac{\partial\ell}{\partial f_k(\mathbf{x})}\right)\frac{\partial f_k(\mathbf{x})}{\partial x_i} + \sum_{k=0}^{K}\frac{\partial\ell}{\partial f_k(\mathbf{x})}\frac{\partial^2 f_k(\mathbf{x})}{\partial x_j\partial x_i} \\ &= \sum_{k=0}^{K}\sum_{m=0}^{K}\left(\frac{\partial^2\ell}{\partial f_m(\mathbf{x})f_k(\mathbf{x})}\right)\frac{\partial f_m(\mathbf{x})}{\partial x_j}\frac{\partial f_k(\mathbf{x})}{\partial x_i} + \sum_{k=0}^{K}\frac{\partial\ell}{\partial f_k(\mathbf{x})}\frac{\partial^2 f_k(\mathbf{x})}{\partial x_j\partial x_i} \qquad \text{(A.10)} \\ &\approx \sum_{k=0}^{K}\sum_{m=0}^{K}\left(\frac{\partial^2\ell}{\partial f_m(\mathbf{x})f_k(\mathbf{x})}\right)\frac{\partial f_m(\mathbf{x})}{\partial x_j}\frac{\partial f_k(\mathbf{x})}{\partial x_i} \\ &= G_{ij} \qquad \text{(A.11)} \end{aligned}$$

The first term in (A.10) is the component of the Hessian due to variations in $f_k(\mathbf{x})$, whereas the second is due to variations in x. Around the minimum of the loss function, the second term is inexpressive and we can neglect it. This approximation is what we call the Generalised Gauss-Newton and is exact when the loss is zero, since the term $\partial\ell/\partial f_k(\mathbf{x})$ is zero, but gets increasingly worse for distant points.

Writing the GGN in matricial form gives

$$G = J_f(\mathbf{x})^T H_\ell(f(\mathbf{x})) J_f(\mathbf{x}), \tag{A.12}$$

which is always positive semi-definite. In the particular case where $\ell(f(\mathbf{x})) = -\log f(\mathbf{x})$, it becomes

$$\begin{aligned} G &= J_f(\mathbf{x})^T \frac{1}{f(\mathbf{x})^2} J_f(\mathbf{x}) \\ &= \nabla_{\mathbf{x}} f(\mathbf{x}) \nabla_{\mathbf{x}} f(\mathbf{x})^T \end{aligned} \tag{A.13}$$

The shortcoming of this approach is losing second order interaction between the different dimensions of the parameter space, which might mean a loss of curvature information [72].

Appendix B

Demonstrations of Properties and Identities

B.1 Update formula for CAVI

Without resorting to variational calculus we derive the update equations for the optimal factors of the approximating distribution $q(\mathcal{Z})$ under the VI algorithm in Section 2.2.1.

We rewrite here the formulas for the factorized approximating distribution 2.17 and the ELBO:

$$q(\mathcal{Z}) = \prod_{i=1}^{M} q_i(\mathcal{Z}_{\mathcal{S}_i}), \tag{B.1}$$

$$\text{ELBO}(q) = \mathbb{E}_q \left[\log p(\mathcal{X}, \mathcal{Z})\right] - \mathbb{E}_q \left[\log q(\mathcal{Z})\right]. \tag{B.2}$$

We substitute (B.1) into (B.2) and extract the dependence on one of the factors $q_i(\mathcal{Z}_{\mathcal{S}_i})$. Simplifying the notation from $q_i(\mathcal{Z}_{\mathcal{S}_i})$ to q_i we get

$$\begin{aligned}
\text{ELBO}(q) &= \int \left(\prod_i q_i\right) \log p(\mathcal{X}, \mathcal{Z}) d\mathcal{Z} - \int \left(\prod_k q_k\right) \log \prod_l q_l d\mathcal{Z} \\
&= \int q_j \left[\int \log p(\mathcal{X}, \mathcal{Z}) \prod_{-j} (q_i dz_i)\right] dz_j - \sum_k \int \prod_l q_l \log q_k d\mathcal{Z} \\
&= \int q_j \, \mathbb{E}_{-j} \left[\log p(\mathcal{X}, \mathcal{Z})\right] dz_j - \sum_k \int q_k \log q_k \left[\prod_{l \neq k} \int q_l dz_l\right] dz_k
\end{aligned}$$

where the symbol $\mathbb{E}_{-j}[\cdot]$ in the first term denotes expectation with respect to the q distribution over all variables $\mathcal{Z}_{\mathcal{S}_i}$ for except $i = j$.

Since each q_l in the second term is an independent factor, it is normalised and sums to 1. In addition, we define a new distribution $\tilde{p}_{-j}$ such that $\log \tilde{p}_{-j} =$

$\mathbb{E}_{-j}\left[\log p(\mathcal{X}, \mathcal{Z})\right] + \text{const}$, and the const term appears to compensate for the normalisation. Then,

$$\begin{aligned}
\text{ELBO}(q) &= \int q_j \log \widetilde{p}_{-j} dz_j - \sum_i \int q_k \log q_k dz_k + \text{const} \\
&= \int q_j \log \widetilde{p}_{-j} dz_j - \int q_j \log q_j dz_j - \sum_{k \neq j} \int q_k \log q_k dz_k + \text{const} \\
&= \int q_j \log \left(\frac{\widetilde{p}_{-j}}{q_j} \right) dz_j + \text{const} \\
&= -D_{KL}(q_j \| \tilde{p}_{-j}) + \text{const} \qquad \text{(B.3)}
\end{aligned}$$

We keep all q_{-j} fixed and maximise the ELBO w.r.t. to q_j. Since the maximum of (B.3) happens when the KL term is zero, we find the optimal q^* by setting it to be equal to $\tilde{p}_{-j}$

$$\begin{aligned}
q_j^*(\mathcal{Z}_{\mathcal{S}_j}) &= \tilde{p}_{-j}(\mathcal{X}, \mathcal{Z}_{\mathcal{S}_j}) \\
\log q_j^*(\mathcal{Z}_{\mathcal{S}_j}) &= \mathbb{E}_{-j}\left[\log p(\mathcal{X}, \mathcal{Z})\right] + \text{const} \qquad \text{(B.4)} \\
q_j^*(\mathcal{Z}_{\mathcal{S}_j}) &\propto \exp\{\mathbb{E}_{-j}\left[\log p(\mathcal{X}, \mathcal{Z})\right]\}. \qquad \text{(B.5)}
\end{aligned}$$

The equations (B.5) for the latent variable sets $\mathcal{S}_j$ are coupled and solving them requires an iterative approach to optimise the objective function by replacing each factor by a revised estimate of (B.5) in coordinate manner.

B.2 Gaussian Gradient Identities

Here we review the derivations of Bonnet's and Price's theorems, equations 3.3 and 3.4 respectively, that were presented in Section 3.3. During those proofs, two other results are useful, so we first derive them.

Given a multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu}, \mathbf{C})$, where $dim(\boldsymbol{\xi}) = d$, the gradient with respect to μ_i can be rewrite as:

$$\begin{aligned}
\nabla_{\mu_i}\mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu}, \mathbf{C}) &= \frac{\partial \left((2\pi)^{-d/2} |\mathbf{C}|^{-1/2} e^{-\frac{1}{2}(\boldsymbol{\xi}-\boldsymbol{\mu})^T \mathbf{C}^{-1}(\boldsymbol{\xi}-\boldsymbol{\mu})} \right)}{\partial \mu_i} \\
&= (2\pi)^{-d/2} |\mathbf{C}|^{-1/2} e^{-\frac{1}{2}(\boldsymbol{\xi}-\boldsymbol{\mu})^T \mathbf{C}^{-1}(\boldsymbol{\xi}-\boldsymbol{\mu})} \frac{\partial \left(-\frac{1}{2}(\boldsymbol{\xi}-\boldsymbol{\mu})^T \mathbf{C}^{-1}(\boldsymbol{\xi}-\boldsymbol{\mu}) \right)}{\partial \mu_i} \\
&= \mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu}, \mathbf{C}) \left(\sum_{k=1}^{d} (\xi_k - \mu_k) l_{ik} \right),
\end{aligned}$$

where $l_{i,i}$ is the i-th element of the i-th column of $\mathbf{C}^{-1}$.

Analogously, we have $\nabla_{\xi_i}\mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu},\mathbf{C}) = -\mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu},\mathbf{C})\left(\sum_{k=1}^{d}(\xi_k-\mu_k)l_{ik}\right)$, so

$$\nabla_{\mu_i}\mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu},\mathbf{C}) = -\nabla_{\xi_i}\mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu},\mathbf{C}). \tag{B.6}$$

The last result we need before we proceed is about the derivative of the Gaussian w.r.t. to its covariance matrix elements:

$$\begin{aligned}
\nabla_{c_{i,j}}\mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu},\mathbf{C}) &= \frac{\partial\left((2\pi)^{-d/2}|\mathbf{C}|^{-1/2}e^{-\frac{1}{2}(\boldsymbol{\xi}-\boldsymbol{\mu})^T\mathbf{C}^{-1}(\boldsymbol{\xi}-\boldsymbol{\mu})}\right)}{\partial c_{i,j}} \\
&= (2\pi)^{-d/2}|\mathbf{C}|^{-1/2}e^{-\frac{1}{2}(\boldsymbol{\xi}-\boldsymbol{\mu})^T\mathbf{C}^{-1}(\boldsymbol{\xi}-\boldsymbol{\mu})}\frac{\partial\left(-\frac{1}{2}(\boldsymbol{\xi}-\boldsymbol{\mu})^T\mathbf{C}^{-1}(\boldsymbol{\xi}-\boldsymbol{\mu})\right)}{\partial c_{i,j}} \\
&+ (2\pi)^{-d/2}e^{-\frac{1}{2}(\boldsymbol{\xi}-\boldsymbol{\mu})^T\mathbf{C}^{-1}(\boldsymbol{\xi}-\boldsymbol{\mu})}\frac{\partial|\mathbf{C}|^{-1/2}}{\partial c_{i,j}} \\
&= \mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu},\mathbf{C})\left(\frac{1}{2}(\boldsymbol{\xi}-\boldsymbol{\mu})^T\mathbf{C}^{-1}\frac{\partial\mathbf{C}}{\partial c_{i,j}}\mathbf{C}^{-1}(\boldsymbol{\xi}-\boldsymbol{\mu})\right) \\
&+ (2\pi)^{-d/2}e^{-\frac{1}{2}(\boldsymbol{\xi}-\boldsymbol{\mu})^T\mathbf{C}^{-1}(\boldsymbol{\xi}-\boldsymbol{\mu})}\left(-\frac{1}{2}|\mathbf{C}|^{-3/2}|\mathbf{C}|\mathrm{tr}\left(\mathbf{C}^{-1}\frac{\partial\mathbf{C}}{\partial c_{i,j}}\right)\right) \\
&= -\frac{1}{2}\mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu},\mathbf{C})\left(-\sum_{k_1=1}^{d}\left((\xi_{k_1}-\mu_{k_1})l_{i,k_1}\sum_{k_2=1}^{d}(\xi_{k_2}-\mu_{k_2})l_{j,k_2}\right)+l_{i,j}\right),
\end{aligned}$$

Now, taking the derivative of $\nabla_{\xi_i}\mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu},\mathbf{C})$ w.r.t. to ξ_j we have:

$$\begin{aligned}
\nabla_{\xi_i,\xi_j}\mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu},\mathbf{C}) &= -\frac{\partial\mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu},\mathbf{C})}{\partial\xi_j}\left(\sum_{k=1}^{d}(\xi_k-\mu_k)l_{i,k}\right)-\mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu},\mathbf{C})l_{i,j} \\
&= -\mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu},\mathbf{C})\left(l_{i,j}-\left(\sum_{k=1}^{d}(\xi_k-\mu_k)l_{i,k}\right)\left(\sum_{k=1}^{d}(\xi_k-\mu_k)l_{j,k}\right)\right),
\end{aligned}$$

which implies on

$$\nabla_{c_{i,j}}\mathcal{N}(\boldsymbol{\xi}|\mu,\mathbf{C}) = \frac{1}{2}\nabla_{\xi_i,\xi_j}\mathcal{N}(\boldsymbol{\xi}|\mu,\mathbf{C}). \tag{B.7}$$

Theorem B.2.1 (Bonnet's theorem)**.** *Let $f(\boldsymbol{\xi}) : R^d \mapsto R$ be a integrable and twice differentiable function. The gradient of the expectation of $f(\boldsymbol{\xi})$ under a Gaussian distribution $\mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu},\mathbf{C})$ with respect to the mean $\boldsymbol{\mu}$ can be expressed as the expectation of the gradient of $f(\boldsymbol{\xi})$, i.e.:*

$$\nabla_{\mu_i}\mathbb{E}_{\mathcal{N}(\mu,\mathbf{C})}\left[f(\boldsymbol{\xi})\right] = \mathbb{E}_{\mathcal{N}(\boldsymbol{\mu},\mathbf{C})}\left[\nabla_{\xi_i}f(\boldsymbol{\xi})\right].$$

Proof.

$$\begin{aligned}
\nabla_{\mu_i} \mathbb{E}_{\mathcal{N}(\boldsymbol{\mu},\mathbf{C})}\left[f(\boldsymbol{\xi})\right] &= \int \nabla_{\mu_i} \mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu},\mathbf{C}) f(\boldsymbol{\xi}) d\boldsymbol{\xi} \\
&= -\int \nabla_{\xi_i} \mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu},\mathbf{C}) f(\boldsymbol{\xi}) d\boldsymbol{\xi} \\
&= -\int \nabla_{\xi_i} \left(\mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu},\mathbf{C}) f(\boldsymbol{\xi})\right) d\boldsymbol{\xi} \\
&+ \int \mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu},\mathbf{C}) \nabla_{\xi_i} f(\boldsymbol{\xi}) d\boldsymbol{\xi} \\
&= -\int_{\xi_1} \cdots \int_{\xi_n} \underbrace{\int_{\xi_i} \nabla_{\xi_i} \left(\mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu},\mathbf{C}) f(\boldsymbol{\xi})\right) d\xi_i}_{=\left[\mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu},\mathbf{C}) f(\boldsymbol{\xi})\right]_{\xi_i=-\infty}^{\xi_i=+\infty}} d\xi_n \cdots d\xi_1 \\
&+ \int \mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu},\mathbf{C}) \nabla_{\xi_i} f(\boldsymbol{\xi}) d\boldsymbol{\xi} \\
&= \left[\int \mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu},\mathbf{C}) f(\boldsymbol{\xi}) d\boldsymbol{\xi}_{\neg i}\right]_{\xi_i=-\infty}^{\xi_i=+\infty} + \mathbb{E}_{\mathcal{N}(\boldsymbol{\mu},\mathbf{C})}\left[\nabla_{\xi_i} f(\boldsymbol{\xi})\right] \\
&= \mathbb{E}_{\mathcal{N}(\boldsymbol{\mu},\mathbf{C})}\left[\nabla_{\xi_i} f(\boldsymbol{\xi})\right], \qquad \text{(B.8)}
\end{aligned}$$

where we have used the identity (B.6) in moving from step 1 to 2, and the product rule for derivatives from step 2 to 3. We have rewritten the first term in moving from step 3 to 4. At the last step we eliminated the first term, which equals zero. □

Theorem B.2.2 (Price's theorem)**.** *Under the same conditions as before. The gradient of the expectation of $f(\boldsymbol{\xi})$ under a Gaussian distribution $\mathcal{N}(\boldsymbol{\xi}|\mathbf{0},\mathbf{C})$ with respect to the covariance $\mathbf{C}$ can be expressed in terms of the expectation of the Hessian of $f(\boldsymbol{\xi})$ as*

$$\nabla_{C_{i,j}} \mathbb{E}_{\mathcal{N}(\mathbf{0},C)}\left[f(\boldsymbol{\xi})\right] = \frac{1}{2} \mathbb{E}_{\mathcal{N}(\mathbf{0},C)}\left[\nabla_{\xi_i,\xi_j} f(\boldsymbol{\xi})\right]$$

Proof.

$$\begin{aligned}
\nabla_{C_{i,j}} \mathbb{E}\left[\mathcal{N}(\mathbf{0},\mathbf{C})\right]\left[f(\boldsymbol{\xi})\right] &= \int \nabla_{C_{i,j}} \mathcal{N}(\boldsymbol{\xi}|\mathbf{0},\mathbf{C}) f(\boldsymbol{\xi}) d\boldsymbol{\xi} \\
&= \frac{1}{2} \int \nabla_{\xi_i,\xi_j} \mathcal{N}(\boldsymbol{\xi}|\mathbf{0},\mathbf{C}) f(\boldsymbol{\xi}) d\boldsymbol{\xi} \\
&= \frac{1}{2} \int \mathcal{N}(\boldsymbol{\xi}|\mathbf{0},\mathbf{C}) \nabla_{\xi_i,\xi_j} f(\boldsymbol{\xi}) d\boldsymbol{\xi} \\
&= \frac{1}{2} \mathbb{E}_{\mathcal{N}(\mathbf{0},\mathbf{C})}\left[\nabla_{\xi_i,\xi_j} f(\boldsymbol{\xi})\right]. \qquad \text{(B.9)}
\end{aligned}$$

In moving from steps 1 to 2, we have used the identity (B.7). From step 2 to 3 we have used the product rule for integrals twice. □