Lucas Pinheiro Cinelli
Matheus Araújo Marins
Eduardo Antônio Barros da Silva
Sérgio Lima Netto

# Variational Methods for Machine Learning with Applications to Deep Networks

Springer

# Variational Methods for Machine Learning with Applications to Deep Networks

Lucas Pinheiro Cinelli • Matheus Araújo Marins
Eduardo Antônio Barros da Silva
Sérgio Lima Netto

# Variational Methods for Machine Learning with Applications to Deep Networks

Springer

Lucas Pinheiro Cinelli
Program of Electrical Engineering - COPPE
Federal University of Rio de Janeiro
Rio de Janeiro, Brazil

Matheus Araújo Marins
Program of Electrical Engineering - COPPE
Federal University of Rio de Janeiro
Rio de Janeiro, Brazil

Eduardo Antônio Barros da Silva
Program of Electrical Engineering - COPPE /
Department of Electronics - Poli
Federal University of Rio de Janeiro
Rio de Janeiro, Brazil

Sérgio Lima Netto
Program of Electrical Engineering - COPPE /
Department of Electronics - Poli
Federal University of Rio de Janeiro
Rio de Janeiro, Brazil

ISBN 978-3-030-70678-4 ISBN 978-3-030-70679-1 (eBook)
https://doi.org/10.1007/978-3-030-70679-1



This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

*To our families.*

# Preface

This book has its origins in the first author's profound interest in uncertainty as a natural way of thinking. Indeed, behavioral studies support that humans perform nearly optimal Bayesian inference, efficiently integrating multi-sensory information while being energetically efficient. At the same time, modern deep learning methods still are sensitive to overfitting and lack uncertainty estimation even though they achieve human-level results in many tasks. The Bayesian framework fits elegantly as a manner to tackle both issues, simultaneously offering a principled mathematical ground.

While Bayesian ML and approximate inference are rather broad topics, each spawning entire books, the present text is a self-contained introduction to modern variational methods for Bayesian Neural Network (BNN). 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. We also address one specific usage of a variational technique for generative modeling.

The target audience are those already familiar with ML and modern NN. Although basic knowledge of calculus, linear algebra, and probability theory is 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 understanding the book and may actually intimidate the reader. Furthermore, we do not assume the reader to be familiar with statistical inference and thus explain the necessary information throughout the text.

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

Prior to any trending ML technique, we introduce in Chap. 2 the required statistical tools that many students lack nowadays. We discuss what is a model, how information is measured, what is the Bayesian approach, as well as two cornerstones of statistical inference: estimation and hypothesis testing. Even those already familiar with the subject could benefit from the refresher, at the same time acclimating with the notation.

In Chap. 3, we introduce the building blocks of Model-Based Machine Learning (MBML). We explain what it is and discuss its main enabling techniques: Bayesian inference, graphical models, and, more recently, probabilistic programming. We explain approximate inference and broach deterministic distributional approximation methods, focusing on Variational Bayes, Assumed Density Filtering, and Expectation Propagation, going through derivations, advantages, issues, and modern extensions.

In Chap. 4, we introduce the concept and advantages of the Bayesian Neural Network (BNN). We scrutinize four of the most popular algorithms in the area: Bayes by Backpropagation, Probabilistic Backpropagation, Monte Carlo Dropout, and Variational Adam, covering their derivations, benefits, and issues. We finish by comparing the algorithms through a 1-D example as well as more complex scenarios.

In Chap. 5, we introduce generative models. We focus specifically on the Variational Autoencoder (VAE) family, a well-known deep generative model. The ability to model the process that generates the observed data empowers us to simulate new data, create world models, grasp underlying generative factors, and learn with little to no supervision. Starting with a simple example, we build the vanilla VAE, pointing out its shortcomings and various extensions, such as the Conditional VAE, the $\beta$-$VAE$, the Categorical VAE, and others. We end the chapter with numerous VAE experiments on two image data sets, and an illustrative example of semi-supervised learning with VAEs.

We take this opportunity to thank our professors and colleagues who helped us in writing this book. In particular, we thank Dr. Leonardo Nunes and Professor Luís Alfredo de Carvalho who first came up with its conceptual ideal. We also thank our loved ones for putting up with us during the challenging and interesting times of turning the book into a reality.

Rio de Janeiro, Brazil — Lucas Pinheiro Cinelli
Rio de Janeiro, Brazil — Matheus Araújo Marins
Rio de Janeiro, Brazil — Eduardo Antônio Barros da Silva
Rio de Janeiro, Brazil — Sergio Lima Netto
September 2020

# Contents

# Acronyms

| | |
|---|---|
| ADF | Assumed Density Filtering. 46, 47, 50, 51, 53–56, 59, 61, 62, 74, 76, 77, 81, 84, 94, 95, 97, 104, 106, 151 |
| ADVI | Automatic Differentiation Variational Inference. 61 |
| AEVB | Autoencoding Variational Bayes. 119, 120 |
| BBB | Bayes by Backprop. 65, 69, 71–75, 102 |
| BBVI | Black Box Variational Inference. 59–61 |
| BNN | Bayesian Neural Network. vii, viii, 65–67, 74, 104, 106, 111 |
| BO | Bayesian Optimization. 100, 101, 103 |
| CAVI | Coordinate Ascent Variational Inference. 42, 45, 46, 151 |
| CDF | Cumulative Distribution Function. 3 |
| CVAE | Conditional VAE. 120 |
| DL | Deep Learning. 2, 66, 119, 130, 146 |
| DNN | Deep Neural Network. 2 |
| ELBO | Evidence Lower Bound. 38, 39, 41–44, 59, 61, 70, 75, 90, 92, 116, 117, 120, 121, 123–125, 127, 132, 133, 139, 145, 153 |
| EM | Expectation-Maximization. 25–28, 30, 43 |
| EP | Expectation Propagation. 36, 47, 54–58, 60–62, 75–77, 81, 97, 104, 106 |
| GGN | Generalized Gauss-Newton. 73, 151 |
| GPU | Graphical Processing Unit. 103 |
| iid | independent and identically distributed. 8–11, 20, 22, 45, 47, 77, 128, 142 |
| IS | Importance Sampling. 115 |
| IWAE | Importance Weighted Autoencoder. 123 |
| KL | Kullback-Leibler. 15–17, 30, 37, 38, 40, 41, 44, 47–49, 55, 57, 58, 71, 72, 77, 92, 114, 117–122, 126, 127, 129, 137–139, 153, 155, 156 |
| MAP | Maximum a posteriori. 23–26, 30, 39 |
| MBML | Model-Based Machine Learning. viii, 31, 61 |

MC Monte Carlo. 69, 70, 72–74, 107
MCDO Monte Carlo Dropout. 65, 89, 90, 97, 102, 104, 105
MCMC Markov Chain Monte Carlo. 1, 2
MFVI Mean-Field VI. 41, 42, 91
ML Machine Learning. vii, viii, 1–3, 5, 6, 13, 18, 23, 30–32, 34, 35, 43, 130, 140, 144–146
MLE Maximum Likelihood Estimator. 22–26, 28–30, 34, 77, 91, 116, 118

NN Neural Network. vii, 65, 66, 87, 116, 117, 119, 141, 144

PBP Probabilistic Backpropagation. 65, 75–77, 82, 96, 97, 101–104
PCA Principal Component Analysis. 34, 35
PDF Probability Density Function. 3, 6, 9–11, 13, 14, 19
PGM Probabilistic Graphical Model. 32, 72, 74, 76, 89, 140

ReLU Rectified Linear Unit. 75, 82, 131, 144
RMSE Root Mean Squared Error. 99, 104, 105

UMAP Uniform Manifold Approximation and Projection. 130, 132

Vadam Variational Adam. 65
VAE Variational Autoencoder. viii, 111, 119, 120, 123
VI Variational Inference. 36, 37, 39, 43, 44, 47, 57, 59–61, 69, 73–75, 88, 91, 106, 117, 135, 145, 152

# Chapter 1
# Introduction


## 1.1 Historical Context

Over the last two decades, Bayesian methods have largely fallen out of favor in the ML community. The culprit for such unpopularity is their complicated mathematics, which makes it hard for practitioners to access and comprehend them, as well as their heavy computational burden. Conversely, classical techniques relying on bagging and point estimates offer cheap alternatives to measure uncertainty and evaluate hypotheses [9]. Consequently, Bayesian methods remained confined mostly to (Bayesian) statisticians and a handful of other researchers either working in related areas or limited by small amounts of data.

For instance, Markov Chain Monte Carlo (MCMC) methods are powerful Bayesian tools [9]. In a modeling 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 [6], 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 that 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. Although correct, the limit is far from being reached in practical cases. As we discuss in Sect. 2.4.1, there is an important fundamental difference between a large and a *statistically* large data set. A mere $28 \times 28$ 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 ($\sim 10^{80}$) [10]. Even in a simple case as this, being *statistically* large means having a virtually infinite number of examples, which is not practically achievable. Naturally, one frequently assumes that there is an underlying low-



L. P. Cinelli et al., *Variational Methods for Machine Learning with Applications to Deep Networks*, https://doi.org/10.1007/978-3-030-70679-1_1

dimensional structure that explains the observations. In Chap. 2, we formalize this thought and in Chap. 5 we review an algorithm that incorporates this assumption.

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, outstanding results have been achieved in the last ten years or so, breaking through plateaus in many areas of research, e.g., speech [5] and vision [8]. As a consequence, the DL domain became a trending area, attracting a lot of newcomers, media attention, and industry investments.

All this positive feedback reinforces the approach of overlooking probabilistic modeling and reasoning. After all, it seems to be working. However, reliable confidence estimates are essential to many domains, such as healthcare and financial markets, whose demands standard Deep Learning cannot adequately attend. Additionally, Deep Learning requires large quantities of data that when not available leads to models that are likely to overfit and have poor generalization. Contrarily, Bayesian methods perform well even in data-poor regimes and are robust, though not immune, to overfitting.

Recently, researchers found that many ML models, including Deep Neural Network (DNN) with great test set performance, are deceived by adversarial examples [3], which are tampered images apparently normal to humans that are consistently misclassified despite the model's great confidence. Moreover, the authors in [3] describe a method to systematically create adversarial examples. Fortunately, methods that estimate uncertainty are capable of detecting adversarial examples and, more generally, examples outside the domain in which they were trained.

Probabilistic models further lend themselves to semi-supervised and unsupervised learning, allowing us to leverage the performance from unlabeled samples. Moreover, we can recur to active learning, in which the system put forward for the operator to annotate the samples it is most uncertain about, thus maximizing information gain and minimizing annotation labor.

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 can 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 [1]. The toll for its speed is inferior performance, making it adequate to scenarios where a lot of data is available to compensate for such

weakness and it would be otherwise impossible to employ MCMC. Over the last decade, research on variational methods for Bayesian ML started to reemerge [4] and slowly gain momentum. Since 2014, there has been an exponential growth in interest for this field [7, 11, 12], fueled among others by the discovery of critical failure modes for conventional Deep Learning. Nowadays, there are workshop tracks for variational Bayesian ML in major ML conferences and lots of papers accepted to the main tracks, as well as in venues geared toward Statistics, Artificial Intelligence, and uncertainty estimation, all increasing in importance, visibility, and submission count.

## 1.2 On the Notation

The following mathematical elements attend the notation:

- scalar: $a$ and $\sigma$;
- vector: $\mathbf{a}$ and $\boldsymbol{\sigma}$;
- matrix: $\mathbf{A}$ and $\boldsymbol{\Sigma}$;
- set: $\mathcal{A}$ and $\Sigma$.

We denote both Probability Density Function (PDF) and discrete probability distributions with lower-case notation $p$. Although an abuse of language, we decided to simplify notation. We shall make 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 in Chap. 4, whose algorithms rely on continuous functions and variables. Additionally, we always denote random variables and the Cumulative Distribution Function (CDF) in upper case, such as $F(X) = P(X \leqslant x)$.

We write parametric family $\mathcal{P}$ of distributions $p$ as $p(\cdot\,;\,\boldsymbol{\theta})$ with $\boldsymbol{\theta}$ the set of parameters that specify the member of the family. For example, for 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 $\mu$ and variance $\sigma^2$. If the parameters are random variables, we can write the conditional distribution as $p(\cdot \mid \boldsymbol{\Theta})$, and since we deal with Bayesian analysis these two notations get pretty similar although different.

Whenever possible, the variational parameters will write $\boldsymbol{\psi}$ and the model parameters $\boldsymbol{\theta}$, and if both refer to the same entity we opt for $\boldsymbol{\theta}$. If we are to consider parameters as random variables, we write them in bold upper-cased letters, i.e., $\boldsymbol{\Psi}$ and $\boldsymbol{\Theta}$, respectively. Similarly, hidden units or more generally latent variables are $\mathbf{Z}$.

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 parameterized by $\boldsymbol{\theta} = [\theta_1, \theta_2]^t$, we have according to this notation:

$$\frac{\partial f(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \begin{bmatrix} \frac{\partial f(\theta_1, \theta_2)}{\partial \theta_1} \\ \frac{\partial f(\theta_1, \theta_2)}{\partial \theta_2} \end{bmatrix}.$$

## References


1. Blei DM, Kucukelbir A, McAuliffe JD (2017) Variational inference: a review for statisticians. J Am Stat Assoc 112(518):859–877
2. Chen M, Mao S, Liu Y (2014) Big data: a survey. Mobile Netw Appl 19(2):171–209
3. Goodfellow IJ, Shlens J, Szegedy C (2015) Explaining and harnessing adversarial examples. In: Proceedings of the international conference on learning representations, San Diego
4. Graves A (2011) Practical variational inference for neural networks. In: Advances in neural information processing systems, Granada, pp 2348–2356
5. Hinton G, Deng L, Yu D, Dahl GE, Mohamed A, Jaitly N, Senior A, Vanhoucke V, Nguyen P, Sainath TN, Kingsbury B (2012) Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. Sign Process Mag 29(6):82–97
6. Homan MD, Gelman A (2014) The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. J Mach Learn Res 15(1):1593–1623
7. Kingma DP, Welling M (2014) Auto-encoding variational Bayes. In: Proceedings of the international conference on learning representations, Banff
8. Krizhevsky A, Sutskever I, Hinton GE (2012) ImageNet classification with deep convolutional neural networks. In: Advances in neural information processing systems, Lake Tahoe, pp 1097–1105
9. Murphy KP (2012) Machine Learning: a probabilistic perspective. MIT Press, Cambridge
10. Planck Collaboration, Ade PAR, Aghanim N, Arnaud M, Ashdown M, Aumont J, Baccigalupi C, Banday AJ, Barreiro RB, Bartlett JG, et al (2016) Planck 2015 results. XIII. Cosmological parameters. Astron Astrophys 594:A13. arXiv:1502.01589
11. Ranganath R, Gerrish S, Blei D (2014) Black box variational inference. In: Proceedings of the international conference on artificial intelligence and statistics, Reykjavik, pp 814–822
12. Soudry D, Hubara I, Meir R (2014) Expectation backpropagation: parameter-free training of multilayer neural networks with continuous or discrete weights. In: Advances in neural information processing systems, Montreal, pp 963–971

# Chapter 2
# Fundamentals of Statistical Inference


By the end of this chapter, the reader should:

- Appreciate the importance of statistical inference as the basis to popular ML;
- Discern between the frequentist and Bayesian views of probability;
- Comprehend the advantages of the exponential family and its characteristics;
- Understand the concept of entropy and information;
- Be capable of implementing computational algorithms for estimation.

## 2.1 Models

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 the corresponding real-world entity. A model is a description of the world (at a given level) and as such encodes our beliefs and assumptions about it. Specifically, a statistical model is a mathematical description of a process and involves both sample data as well as statistical assumptions about such process.

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 better.



L. P. Cinelli et al., *Variational Methods for Machine Learning with Applications to Deep Networks*, https://doi.org/10.1007/978-3-030-70679-1_2

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 estimations. 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.

### *2.1.1 Parametric Models*

A parametric model $\mathcal{P}$ is a family of distributions $f$ that can be indexed by a finite number of parameters. Let $\boldsymbol{\theta}$ be an element of the parameter space $\Theta$ and $\mathbf{X}$ a random variable, we define the set of possible distribution of the parametric model as

$$\mathcal{P}_{\Theta} = \{f(\mathbf{x}\,;\boldsymbol{\theta}) : \boldsymbol{\theta} \in \Theta\}\,. \tag{2.1}$$

A simple, yet clear example is the uniform distribution $\mathcal{U}(a, b)$ defined by

$$f(x\,;a,b) = \begin{cases} 1/(a-b)\,, \text{ if } x \in [a,b] \\ 0\,, \quad \text{otherwise}\,. \end{cases} \tag{2.2}$$

Note that each pair of parameters $\{a, b\}$ defines a different distribution that follows the same functional form.

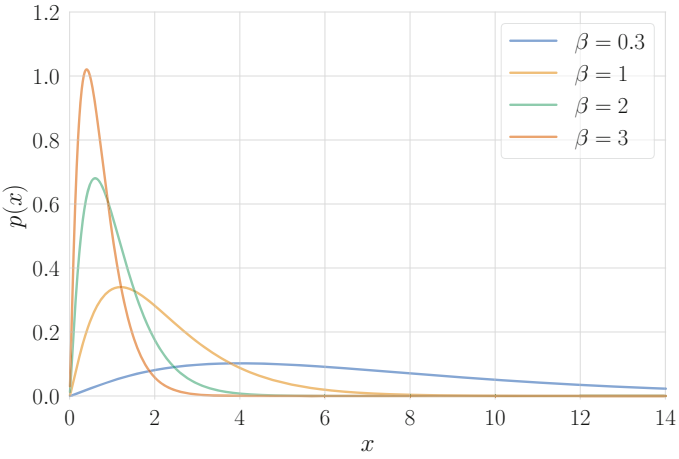
#### 2.1.1.1 Location-Scale Families

We can also generate families of distributions by modifying an original base PDF, hence named standard PDF, in a predefined manner. Concisely, we can either shift, scale, or shift-and-scale the standard distribution.

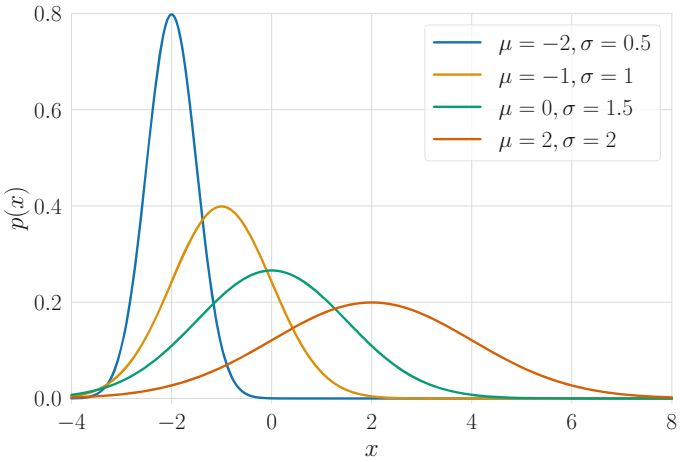
**Theorem 2.1** *Let $f(x)$ be a PDF and $\mu$ and $\sigma > 0$ constants. Then, the following functions are also a PDF:*

$$g(x\,;\mu,\sigma) = \frac{1}{\sigma} f\left(\frac{x-\mu}{\sigma}\right). \tag{2.3}$$

Hence, introducing the scale $\sigma$ and/or the location $\mu$ parameters in the PDF and tweaking their values lead to new PDFs. Examples of families generated from these procedures include many of the well-known distributions. Figure 2.1a shows the




(a)




(b)

**Fig. 2.1** Illustration of location-scale families for the Gamma and Gaussian distributions. In our parametrization of the Gamma function with the rate parameter $\beta$, the scale parameter as defined in Theorem 2.1 is actually $\sigma = 1/\beta$. Note that as the scale $\sigma$ increases the distribution becomes less concentrated around the location parameter. In particular, $\lim_{\sigma \to 0} f(x\,; \mu, \sigma) = \delta(x - \mu)$. (**a**) Members of the same scale family of Gamma distributions with shape parameter $\alpha = 2.2$. (**b**) Members of the same location-scale family of Gaussian distributions

Gamma distribution $\mathrm{Ga}(\alpha, \beta)$ which is a scale family for each value of the shape parameter $\alpha$:

$$f(x\,; \alpha, \beta) = \frac{\beta^{\alpha}}{\Gamma(\alpha)} x^{\alpha - 1} e^{-\beta x}. \tag{2.4}$$

Likewise, Fig. 2.1b exhibits the Gaussian distribution $\mathcal{N}(\mu, \sigma)$ that is a location-scale family for the parameters $\mu$ and $\sigma$, respectively, following

$$f(x\,;\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}. \tag{2.5}$$

### 2.1.2 Nonparametric Models

Nonparametric models assume an infinite dimensional parameter space $\boldsymbol{\Theta}$, instead of a finite one. We interpret $\boldsymbol{\theta}$ as a realization from a stochastic process, what defines a probability distribution over $\Theta$ and further allows us to understand $\boldsymbol{\theta}$ as a random function.

A well-known example is given by infinite mixture models [6], which can have a countably infinite number of components, and uses a Dirichlet Process to define a distribution of distributions [9]. The model allows the number of latent components to grow as necessary to accommodate the data, which is a typical characteristic of nonparametric models.

### 2.1.3 Latent Variable Models

Given observed data $\mathbf{x}$, how should we model the distribution $p(\mathbf{x})$ so that it reflects the true real-world population? This distribution may be arbitrarily complex and to readily assume the data points $\mathbf{x}_i$ to be independent and identically distributed (iid) seems rather naive. After all, they cannot be completely independent, as 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 $\mathbf{Z}$, thus obtaining the joint distribution $p(\mathbf{x}, \mathbf{z})$. Naturally, by marginalizing over $\mathbf{z}$ we obtain

$$p(\mathbf{x}) = \int p(\mathbf{x}, \mathbf{z}) d\mathbf{z} = \int p(\mathbf{x} \,|\, \mathbf{Z}) p(\mathbf{z}) d\mathbf{z}. \tag{2.6}$$

Note that we use latent variables and unknown model parameters interchangeably. For Bayesians, there is no fundamental difference between model parameters and latent variables, as they are all random variables whose values we wish to infer. For example, if our observables $\mathbf{X}_i$ are Bernoulli random variables, then $\mathbf{Z}$ corresponds to the probability of success $p \in (0, 1)$.

### 2.1.4 De Finetti's Representation Theorem

Independence is a strong assumption. Instead we can resort to the infinite exchangeability property. Exchangeability is the minimal assumption of symmetry. A finite sequence of random variables $(\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_n)$ is said to be exchangeable if any permutation of its elements has the same probability distribution [7]. 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 [7] states that if the sequence of random variables is infinitely exchangeable, then there exists $p(\mathbf{z})$ for which the joint distribution can be written as

$$p(\mathbf{x}_1, \mathbf{x}_2, \ldots) = \int \prod_i p(\mathbf{x}_i \mid \mathbf{Z}) p(\mathbf{z}) d\mathbf{z}. \tag{2.7}$$

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 $\{\mathbf{X}\}_{i=1}^{\infty}$, 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 $p(\mathbf{z})$.

In practical problems, even if we deal with unordered data, the number of random variables is always finite. Therefore, the infinite exchangeability assumption may be impractical or wrong, but the result still holds approximately true for large $n$ [5].

### 2.1.5 The Likelihood Function

The likelihood function $L(\theta \mid \mathbf{x})$ measures how well the parameter $\theta$ explains the observations $\mathbf{x}$ of the random variable $\mathbf{X}$. Thus, it measures the model's ability to fit the observed data for different values of $\theta$. The definition is similar to the PDF $f(\mathbf{x}; \theta)$, following

$$L(\theta \mid \mathbf{x}) = f(\mathbf{x}; \theta). \tag{2.8}$$

The higher its value, the more likely the given value of $\theta$, indicating that $\mathbf{x}$ had higher probability of being observed over other realizations of $\mathbf{X}$.

Despite the similarity in the definition, the likelihood considers $\mathbf{x}$ as known and fixed while $\theta$ as the unknown variable. On the other hand, the PDF considers $\theta$ as fixed and $\mathbf{x}$ as the variable. Hence, the likelihood function is not a PDF and, as a consequence, does not necessarily sum to one.

We can understand the role of the likelihood term $L(\theta \mid \mathbf{x})$ more practically by means of an example. Let $f(\cdot\,; w)$ be a regression model parameterized by $w$ that predicts a scalar value $\hat{y}$, such that $\hat{y} = f(x)$. Suppose a probabilistic model that 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 $\sigma^2$. So, now our model does not output the correct value $f(x; w)$, instead it outputs a value that fluctuates around it according to a Gaussian distribution with variance $\sigma^2$. The log-likelihood then has the form

$$\begin{aligned} \log L(w \mid y, x) &= \log \mathcal{N}(y\,;\, f(x\,;w), \sigma^2) \\ &= -\frac{1}{2}\log\left(2\pi\sigma^2\right) - \frac{1}{2\sigma^2}\left(y - f(x; w)\right)^2 . \end{aligned} \tag{2.9}$$

What we wish is maximize $L(w \mid y, x)$, which means minimize $(y - f(x; w))^2$. So, as $y$ gets closer to the predicted output $f(x)$, more likely the value of $w$. Note that the prediction and the noise model could in principle be anything.

## 2.2 Exponential Family

### *2.2.1 Sufficient Statistics*

When working with statistical models, it is necessary to recover some, or even all, of its parameters from a set of randomly drawn samples $x_1, x_2, \cdots, x_n$. Assuming these observations are iid and sampled from a PDF $f(x\,;\boldsymbol{\theta})$, estimating the parameter $\boldsymbol{\theta}$ is the goal of many statisticians and engineers, imposing a real challenge sometimes. A rather common approach is to capture and summarize some information from the observations and use it to estimate the parameter $\boldsymbol{\theta}$ instead of using the observations themselves. This strategy is known as data reduction, whereas engineers and computer scientists call it feature extraction.

The problem of the data reduction is the loss of information. How do we guarantee that with the statistics we computed from the observations, call it $T(\mathbf{X})$, we are not discarding information to estimate $\boldsymbol{\theta}$? The answer to this question is provided by the Sufficiency Principle, which defines $T(\mathbf{X})$ as a sufficient statistics when, for any two samples $\mathbf{x}_1$ and $\mathbf{x}_2$, where $T(\mathbf{x}_1) = T(\mathbf{x}_2)$, the estimation of $\boldsymbol{\theta}$ yields the same results despite observing $\mathbf{X} = \mathbf{x}_1$ or $\mathbf{X} = \mathbf{x}_2$.

Computing the sufficient statistics for a parameter is a rather difficult task in most scenarios, but a rather simple way to do that is by using the Fisher-Neyman Theorem (also known as factorization theorem).

**Theorem 2.2 (Fisher-Neyman Factorization Theorem)** *Let $x_1, x_2, \cdots, x_n$ be random observations from a discrete distribution with PDF $f(\mathbf{x}\,;\boldsymbol{\theta})$, and $\mathbf{x} = [x_1, x_2, \cdots, x_n]^t$. $T(\mathbf{X})$ is a sufficient statistics if, and only if, there exist functions $g(T(\mathbf{x})\,;\boldsymbol{\theta})$ and $h(\mathbf{x})$, such that $h(\mathbf{x}) \geq 0$ and, for all sample points $\mathbf{x}$ and all values of $\boldsymbol{\theta}$, the distribution $f(\mathbf{x}\,;\boldsymbol{\theta})$ can be factorized as follows:*

$$f(\mathbf{x}\,;\boldsymbol{\theta}) = g(T(\mathbf{x})\,;\boldsymbol{\theta})h(\mathbf{x}). \tag{2.10}$$

For example, for a one-dimensional Poisson distribution with unknown mean $\theta$, if we draw a sample $\mathbf{x}$ formed by $n$ iid observations, it is possible to write the probability function as:

$$f(\mathbf{x}\,;\theta) = \prod_{i=1}^{n} f(x_i\,;\theta) = \prod_{i=1}^{n} \frac{e^{-\theta}\theta^{x_i}}{x_i!} = \frac{e^{-n\theta}\theta^{\sum_{i=1}^{n} x_i}}{\prod_{i=1}^{n} x_i!} = g(T(\mathbf{x})\,;\theta)h(\mathbf{x}), \tag{2.11}$$

where $g(T(\mathbf{x})\,;\theta) = e^{-n\theta}\theta^{\sum_{i=1}^{n} x_i}$ and $h(\mathbf{x}) = \left(\prod_{i=1}^{n} x_i!\right)^{-1}$. So, from the factorization theorem, $T(\mathbf{x}) = \sum_{i=1}^{n} x_i$ is a sufficient statistics for $\theta$.

### 2.2.2 Definition and Properties

The exponential family is a parametric family with PDF $f(x\,;\theta)$ that can be written in the form

$$f(x\,;\boldsymbol{\theta}) = h(x)e^{\sum_{i=1}^{k} \eta_i(\boldsymbol{\theta})t_i(x) - B(\boldsymbol{\theta})}, \tag{2.12}$$

where $h(x) \geq 0$ and $\eta_1, \eta_2, \cdots, \eta_k$ are called natural parameters.

Many of the well-known distributions, such as Poisson, beta, and binomial, are members of the exponential family, i.e., their respective PDFs can be expressed in the form of (2.12). Let us verify if the binomial distribution belongs to the exponential family:

$$\begin{aligned} f(x\,;p) &= \binom{n}{x} p^x (1-p)^{n-x} \\ &= \binom{n}{x} (1-p)^n \left(\frac{p}{1-p}\right)^x \end{aligned}$$

$$= \binom{n}{x} e^{n \log(1-p)} e^{x \log \frac{p}{1-p}}$$
$$= h(x) e^{\eta_1(p) t_1(x) - B(p)}, \tag{2.13}$$

where $h(x) = \binom{n}{x}$, $B(p) = -n \log(1-p)$, $t_1(x) = x$, and $\eta_1(p) = \log \frac{p}{1-p}$.

Now, from (2.12) and assuming we have $n$ independent observations, we have

$$f(\mathbf{x}; \theta) = \prod_{j=1}^{n} h(x_j) e^{\sum_{i=1}^{k} \eta_i(\theta) t_i(x_j) - B(\theta)}$$
$$= \prod_{j=1}^{n} \left( h(x_j) \left[ e^{\sum_{i=1}^{k} \eta_i(\theta) \sum_{j=1}^{n} t_i(x_j) - nB(\theta)} \right] \right). \tag{2.14}$$

In the context of Theorem 2.2, we have

$$h(\mathbf{x}) = \prod_{j=1}^{n} h(x_j), \tag{2.15}$$

$$g(t(\mathbf{x}); \theta) = e^{\sum_{i=1}^{k} \eta_i(\theta) \sum_{j=1}^{n} t_i(x_j) - nB(\theta)}, \tag{2.16}$$

which implies the sufficient statistics[1] $T(\mathbf{x}) = (\sum_{j=1}^{n} t_1(x_j), \cdots, \sum_{j=1}^{n} t_k(x_j))$. This result is extremely important, since it gives us a formula to compute the sufficient statistics for a large family of important distributions. The exponential family also provides two equations that allow us to compute derivatives instead of integrations [2], by changing the expected value (which is an integration) for usually easier operations:

$$\mathbb{E}\left[ \sum_{i=1}^{k} \frac{\partial \eta_i(\boldsymbol{\theta})}{\partial \theta_j} t_i(\mathbf{x}) \right] = \frac{\partial B(\boldsymbol{\theta})}{\partial \theta_j}, \tag{2.17}$$

$$\text{Var}\left[ \sum_{i=1}^{k} \frac{\partial \eta_i(\boldsymbol{\theta})}{\partial \theta_j} t_i(\mathbf{x}) \right] = \frac{\partial^2 B(\boldsymbol{\theta})}{\partial \theta_j^2} - \mathbb{E}\left[ \sum_{i=1}^{k} \frac{\partial^2 \eta_i(\boldsymbol{\theta})}{\partial \theta_j^2} t_i(\mathbf{x}) \right], \tag{2.18}$$

$\forall j \in \{1, 2, \cdots, d\}$.

[1] In the case of exponential family, it is also known as natural sufficient statistics.

## 2.3 Information Measures

How much information does a data set gives us about the parameters of our model? How much information may a random variable carry? How do we measure the amount of information gained from observing a random variable? What does information mean? Information theory answers those questions and its concepts are of fundamental importance to ML and, more generally, to information systems, whether they be for quantification, storage, or communication.

In this section, we briefly present some of the fundamental notions of information theory as well as different measures of information that will be needed along book.

### *2.3.1 Fisher Information*

The Fisher information measures the variance in the distribution $f(x\,;\theta)$ inflicted by changes in the parameter space $\Theta$. Intuitively, it quantifies the amount of information about $\theta$ that lies in the random variable $X$.

For the $k$-dimensional parameter space $\boldsymbol{\Theta}$ and random variable $\mathbf{X}$ with PDF $f(\mathbf{x}\,;\boldsymbol{\theta})$, the elements of the Fisher information matrix are

$$\mathcal{I}_{i,j}(\boldsymbol{\theta}) = \mathrm{Cov}\left(\frac{\partial}{\partial\theta_i}\log f(\mathbf{X}\,;\boldsymbol{\theta}),\, \frac{\partial}{\partial\theta_j}\log f(\mathbf{X}\,;\boldsymbol{\theta})\right), \tag{2.19}$$

where $\mathrm{Cov}\,(\cdot,\cdot)$ is the covariance function.

The vector $\frac{\partial}{\partial\boldsymbol{\theta}}\log f(\mathbf{X}\,;\boldsymbol{\theta})$ is called the score function and indicates the sensitivity of the likelihood to the parameter $\boldsymbol{\theta}$. When a likelihood $L(\boldsymbol{\theta}\,|\,\mathbf{x})$, corresponding to the PDF $f(\mathbf{x}\,;\boldsymbol{\theta})$, is very sensitive to variations in $\boldsymbol{\theta}$ it is easier to find strong candidates to the true parameter value: even small changes in $\boldsymbol{\theta}$ are enough to cause the likely observations to be considerably different. However, since the score function has mean equal to zero [9], a high $\mathcal{I}_{i,j}(\boldsymbol{\theta})$ implies that the score function is generally high and then, $\mathbf{X}$ distinguishes well the plausible values of $\boldsymbol{\theta}$. We state "generally" because, being the covariance of the score, the Fisher information is an expectation over all possible values of $\mathbf{x}$.

The Fisher information encodes the curvature of the parameter space and plays an important role in optimization. In Chap. 4 we shall see one method that relies on the Fisher information and in Appendix A.4 we show that the Fisher matrix is the negative of the expected value of the Hessian of the log-likelihood.

### 2.3.2 Entropy

Entropy is a nonnegative measure of the expected information content of a random variable. The information content quantifies how surprising a particular outcome is: the less probable is the event, the more informative it is. If an event has probability 0 or 1, its observation (or lack thereof) is not surprising at all and it does not provide us any additional information. On the other hand, if an event is very unlikely and yet it happens, then such observation brings a lot of valuable information.

As hinted, the information content depends on the probability of a particular event and, hence, the entropy depends on the random variable's probability distribution. For a discrete random variable with probability mass function $p(x)$, the information content of an observation $x$ is

$$h(x) = -\log p(x). \tag{2.20}$$

Thus, the entropy being the expected information content of a random variable is defined as

$$\mathcal{H}(X) = -\sum_i p(x_i) \log p(x_i) = \mathbb{E}_X \big[-\log p(x)\big]. \tag{2.21}$$

If the probability distribution is flat, all events are equally probable and we cannot be sure of any particular outcome, so the entropy is at its maximum.

#### 2.3.2.1 Conditional Entropy

The conditional entropy quantifies the remaining amount of entropy present on the random variable $X$ when $Y$ is known. If both variables are independent, then information about one should not affect our knowledge about the other. Thus, the conditional entropy $\mathcal{H}(X \mid Y)$ should equal $\mathcal{H}(X)$.

We define the conditional entropy as the expected information content of $X \mid Y$ under the joint distribution $p(x, y)$, that is

$$\mathcal{H}(X \mid Y) = -\sum_{i,j} p(x_i, y_j) \log \frac{p(x_i, y_j)}{p(y_j)} = \mathbb{E}_{X \mid Y} \big[-\log p(x \mid y)\big]. \tag{2.22}$$

#### 2.3.2.2 Differential Entropy

Originally, Claude Shannon defined entropy in the context of message symbols in communication theory [10], which are inherently discrete. The entropy of the discrete random variable $Y$ whose probability mass function infinitesimally approximates the PDF $f(x)$ of a continuous random variable $X$ is given by

$$\begin{aligned}
\lim_{\epsilon_i \to 0} \mathcal{H}(X) &= -\sum_i f(x_i)\epsilon_i \log\left(f(x_i)\epsilon_i\right) \\
&= -\sum_i f(x_i)\epsilon_i \log f(x_i) - \sum_i f(x_i)\epsilon_i \log \epsilon_i \\
&= -\int f(x) \log f(x) dx - \log \epsilon_i \int f(x) dx \\
&= -\int f(x) \log f(x) dx + \lim_{\epsilon_i \to 0} -\log \epsilon_i \qquad (2.23) \\
&= -\int f(x) \log f(x) dx + \infty, \qquad (2.24)
\end{aligned}$$

where $\epsilon_i$ is the width of the $i$th discretized segment.

The Shannon entropy diverges and for continuous distributions we offset the limit in (2.24) by $\lim_{\epsilon_i \to 0} \log \epsilon_i = -\infty$. The continuous extension, named differential entropy, is not a direct equivalent to Shannon's entropy and does not share some of its original properties, e.g., nonnegative values, though the formulas look similar. Let $f(x)$ be the pdf of the continuous random variable $X$, then

$$\mathcal{H}(X) = -\int f(x) \log f(x) dx = \mathbb{E}_X\left[-\log f(x)\right]. \qquad (2.25)$$

Throughout the remaining chapters we shall write entropy for both Shannon and differential entropy. The context will be clear in both cases since the former refers to discrete random variables and the latter to continuous ones.

Despite the divergence, when dealing with difference of entropies in the continuous case, the terms $\lim_{\epsilon_i \to 0} -\log \epsilon_i$ cancel, and then the divergence disappears. In the next sections we will present other information measures that are based on difference of entropies, and can be applied for discrete and continuous random variables in the same way.

### 2.3.3 Kullback-Leibler Divergence

Unlike the information measures seen so far, the Kullback-Leibler (KL) divergence is a relative measure in that it assesses the dissimilarity between two distributions $p(x)$ and $q(x)$ over the same random variable $X$ by

$$D_{KL}(p\|q) = \int p(x) \log\left(\frac{p(x)}{q(x)}\right) dx. \qquad (2.26)$$

Alternatively, we can write the KL divergence as

$$D_{KL}(p\|q) = \mathcal{H}_q(p) - \mathcal{H}(p), \tag{2.27}$$

where $\mathcal{H}(p)$ is the entropy when the random variable's probability distribution is $p$ and $\mathcal{H}_q(p)$ is the cross-entropy between $p$ and $q$ given by

$$\mathcal{H}_q(p) = -\int p(x)\log q(x)dx. \tag{2.28}$$

While the entropy $\mathcal{H}(p)$ gives us the expected message length necessary to transmit the information content present in the random variable $X$ whose distribution is given by $p(x)$, the cross-entropy $\mathcal{H}_q(p)$ gives us the expected message length to transmit $X$ when assuming the incorrect distribution $q(x)$ [3]. In that sense, from (2.28) we can understand the KL divergence as a measure of the expected additional length caused by using $q$ instead of $p$. Hence, it cannot be negative nor symmetric and the definition in (2.26) complies to these constraints.

Since the argument of the log in the KL divergence is a ratio between distributions, the quantity is well-defined for continuous random variables. Moreover, it is invariant to affine transformations. However, the distributions $p$ and $q$ must have the same support, that is, they must be defined over the same set of events. Indeed, it does not make sense to compare distributions otherwise.

### 2.3.4 Mutual Information

The mutual information $I(X;Y)$ can be defined as the complement of the conditional entropy $\mathcal{H}(X\,|\,Y)$ relative to the entropy $\mathcal{H}(X)$ as illustrated in Fig. 2.2. Thus, mutual information quantifies the expected reduction in uncertainty about $X$ that comes from observing $Y$. It is a measure of the common information between $X$ and $Y$. Hence, it is a property of their joint distribution.

Let $p(x)$ and $p(y)$ be the marginal distribution of $X$ and $Y$, respectively, and $p(x, y)$ their joint distribution. Then, the mutual information is

$$I(X;Y) = D_{KL}(p(x,y)\|p(x)p(y)). \tag{2.29}$$

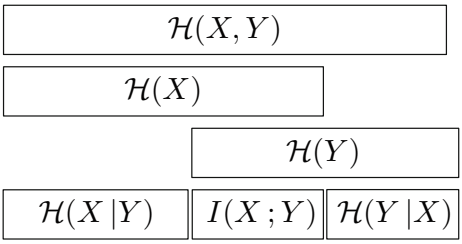



**Fig. 2.2** Relationship between joint information, marginal entropy, conditional entropy, and mutual information

Note that (2.29) is the KL divergence between $p(x, y)$ and the product of their marginals $p(x)p(y)$. We can understand $I(X; Y)$ as the information cost, i.e., the extra number of bits, of encoding the pair $(X, Y)$ as independent. If the variables are actually independent, then $I(X; Y) = 0$.

## 2.4 Bayesian Inference

Bayesian probability inherits its name from the Bayes rule:

$$p(z \mid X) = \frac{p(x \mid Z)p(z)}{p(x)} . \tag{2.30}$$

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

In the above expression each term has a clear interpretation:

- $p(z)$—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 [2];
- $p(x \mid Z)$—the *likelihood*—already seen in Sect. 2.1.5, 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.
- $p(z \mid X)$—the *posterior*—its name reflects the fact that our knowledge about the parameters is updated after accounting for our (new) data, thus it is the probability of $z$ conditioned on such evidence;
- $p(x)$—the *evidence*—as hinted above, the term refers to the observable data and this distribution works as a normalizing factor equal to $\int p(x \mid Z)p(z)dz$ so that the posterior corresponds to a proper probability distribution.

### *2.4.1 Bayesian vs. Classical Approach*

The existence of a prior and a posterior distribution are inherent to the Bayesian framework, in which the parameters and other unknowns are seen not as fixed but as random variables. Thus, all unknown quantities are treated equally and there is no distinction among them.

The Bayesian view is an interpretation of probability itself and its meaning. While frequentists[2] see it as the relative frequency of an event, Bayesians see it as quantification of a belief.

Under a statistically large data set, the Bayesian posterior becomes asymptotically narrow and similar to the likelihood. Intuitively, this behavior makes sense: the evidence provided by the observed data becomes so strong that the effect of the prior knowledge is practically irrelevant. In those scenarios, Bayesian and classical approaches give comparable results. 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 modeling really shines when data is limited and traditional methods are prone to overfitting, the uncertainty in the parameters is significant in these cases [1]. 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 [4] 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.

### 2.4.2 The Posterior Predictive Distribution

A Bayesian treatment consists of computing the full posterior distribution $p(z \mid X)$ instead of only point estimates, therefore the importance of the normalizing constant in (2.30), the evidence $\int p(x \mid Z)p(z)dz$. Both distributions 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 $x'$:

$$p(x' \mid x) = \int p(x' \mid Z)p(z \mid X = x)dz \,. \tag{2.31}$$

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

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 Sect. 3.2 we go through some approximate methods that deal with this issue and in Chap. 4 we discuss algorithms

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

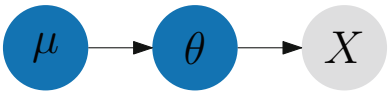



**Fig. 2.3** Hierarchical Bayes model of 3 stages. $X$ is an observed random variable, $\theta$ is an unknown parameter governing its generation process, and $\mu$ a hyper-parameter that determines the distribution of the random variable $\theta$

for performing Bayesian regression in deep neural networks (a class of parametric models).

### *2.4.3 Hierarchical Modeling*

Bayesian models may be further decomposed into a sequence of conditional distributions spanning multiple levels following a hierarchical structure. Bayesian hierarchical models then have multiple levels of hyper-parameters which set the prior distribution of downstream stages and hyper-priors that define the corresponding distributions. In the example of Fig. 2.3, $\mu$ is a hyper-parameter since $\theta$ is already a parameter that influences the distribution of the observations $X$. If we define a distribution for $\mu$, we are defining a hyper-prior, and if we want to take this distribution into consideration in the inference process, by marginalizing $\mu$, we are performing a fully Bayesian approach. On the other hand, if we define a point value for $\mu$ by maximizing its likelihood determined by the observations, we are doing empirical Bayes. In Chap. 4, we will use hierarchical modeling to develop the Probabilistic Backprop.

## 2.5 Conjugate Prior Distributions

### *2.5.1 Definition and Motivation*

In Sect. 2.2, we presented particular characteristics of the exponential family and the advantages of using PDFs from this family. Resorting to the previous section, in Bayesian theory, the prior knowledge of a parameter is used alongside some experiments to update that parameter's beliefs. Mathematically speaking, we have

$$p(z \mid X) \propto p(x \mid Z)p(z). \tag{2.32}$$

So, one can choose the prior distribution to maintain the posterior distribution in the same family. When $p(z \mid X)$ and $p(z)$ are in the same family of distributions, we say that $p(z)$ is a conjugate prior distribution for the likelihood function $p(x \mid Z)$. In that manner, using conjugate priors can save a lot of time as it is only necessary to

compute the updated parameters and not the entire distribution. Although conjugate prior distributions offer a more straightforward way to calculate the posterior, it is essential to choose a prior that can represent your beliefs about the unknown parameters.

### 2.5.2 Conjugate Prior Examples

For the first example on conjugate priors, we will show the thought process used to compute them. Let us suppose we have a Normal distribution with known variance $\sigma$ and unknown mean $\mu$:

$$p(x \mid \mu) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}. \tag{2.33}$$

Then, considering $n$ iid observations:

$$\begin{aligned} p(x \mid \mu) &= \prod_{i=1}^{n} p(x_i \mid \mu) \\ &= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{x_i-\mu}{\sigma}\right)^2} \\ &= \left(2\pi\sigma^2\right)^{-\frac{n}{2}} e^{-\frac{1}{2}\sum_{i=1}^{n}\left(\frac{x_i-\mu}{\sigma}\right)^2}. \end{aligned} \tag{2.34}$$

Using this result in (2.32), the posterior can be written as

$$p(\mu \mid X) \propto p(\mu) \left(2\pi\sigma^2\right)^{-\frac{n}{2}} e^{-\frac{1}{2}\sum_{i=1}^{n}\left(\frac{x_i-\mu}{\sigma}\right)^2}. \tag{2.35}$$

Now, we can notice that the factor that influences the $\mu$ parameter is an exponential with both terms $\mu^2$ and $\mu$, and then, the posterior will also have both terms. Thus, the conjugate prior must have both terms, which indicates that it must also be a Normal distribution. Supposing it is a Normal distribution with mean $\mu_0$ and variance $\sigma_0$, we have

$$\begin{aligned} p(\mu \mid X) &\propto p(\mu) \left(2\pi\sigma^2\right)^{-\frac{n}{2}} e^{-\frac{1}{2}\sum_{i=1}^{n}\left(\frac{x_i-\mu}{\sigma}\right)^2} \\ &\propto \frac{1}{\sqrt{2\pi\sigma_0^2}} e^{-\frac{1}{2}\left(\frac{\mu-\mu_0}{\sigma_0}\right)^2} \left(2\pi\sigma^2\right)^{-\frac{n}{2}} e^{-\frac{1}{2}\sum_{i=1}^{n}\left(\frac{x_i-\mu}{\sigma}\right)^2} \end{aligned}$$

$$\propto e^{-\frac{1}{2}\left[\mu^2\left(\frac{1}{\sigma_0^2}+\frac{n}{\sigma^2}\right)-2\mu\left(\frac{\mu_0}{\sigma_0^2}+\frac{n\bar{x}}{\sigma^2}\right)\right]}. \tag{2.36}$$

In the last step of (2.36) we first ignored every term not related to $\mu$ as they are constant and grouped the remaining terms with either $\mu^2$ or $\mu$. At this point, we want to write $p(\mu \mid X)$ also as a Normal distribution of the form $\mathcal{N}\left(\mu_p, \sigma_p^2\right)$. Therefore,

$$\begin{aligned} p(\mu \mid X) &\propto e^{-\frac{1}{2}\left[\mu^2\left(\frac{1}{\sigma_0^2}+\frac{n}{\sigma^2}\right)-2\mu\left(\frac{\mu_0}{\sigma_0^2}+\frac{n\bar{x}}{\sigma^2}\right)\right]} \\ &= \left(2\pi\sigma_p^2\right)^{-\frac{1}{2}} e^{-\frac{1}{2}\left(\frac{\mu-\mu_p}{\sigma_p}\right)^2} \\ &\propto e^{-\frac{1}{2}\left[\mu^2\frac{1}{\sigma_p^2}-2\mu\frac{\mu_p}{\sigma_p^2}\right]}. \end{aligned} \tag{2.37}$$

Which implies that

$$\sigma_p^{-1} = \left(\frac{1}{\sigma_0^2}+\frac{n}{\sigma^2}\right), \tag{2.38}$$

$$\mu_p = \frac{\mu_0 + n\bar{x}}{\sigma^2 + n\sigma_0^2}. \tag{2.39}$$

This process looks long and not as a simple way to compute the posterior, but the alternative method involves integrations that usually leads to a longer path.

Next, we give another example that will be quite useful in Chaps. 3 and 4. Instead of figuring out the conjugate prior, we will only prove that the Gamma distribution is a conjugate prior for the Gaussian distribution with known mean and unknown precision (the inverse of its variance $1/\sigma^2$), that is

$$p(x \mid \lambda) = \prod_{i=1}^{n} \mathcal{N}\left(x \mid \mu, \lambda^{-1}\right) = \left(\frac{\lambda}{2\pi}\right)^{\frac{n}{2}} e^{-\frac{\lambda}{2}\sum_{i=1}^{n}(x_i-\mu)^2}, \tag{2.40}$$

$$p(\lambda, \alpha_0, \beta_0) = \frac{\beta_0^{\alpha_0}\lambda^{\alpha_0-1}e^{-\beta_0\lambda}}{\Gamma(\alpha_0)} \propto \lambda^{\alpha_0-1}e^{-\beta_0\lambda}. \tag{2.41}$$

Therefore, from (2.32), we have

$$\begin{aligned} p(\lambda \mid X) &\propto \lambda^{\alpha_0-1}e^{-\beta_0\lambda}\left(\frac{\lambda}{2\pi}\right)^{\frac{n}{2}} e^{-\frac{\lambda}{2}\sum_{i=1}^{n}(x_i-\mu)^2} \\ &\propto \lambda^{\alpha_0-1+\frac{n}{2}} e^{-\lambda\left(\frac{1}{2}\sum_{i=1}^{n}(x_i-\mu)^2+\beta_0\right)} \end{aligned}$$

$$= \text{Ga}\left(\lambda \left| \alpha_0 + \frac{n}{2}, \frac{1}{2}\sum_{i=1}^{n}(x_i - \mu)^2 + \beta_0 \right.\right). \tag{2.42}$$

From (2.42) we see the influence of the parameters $\alpha_0$ and $\beta_0$ on the posterior. It is as if we had already collected $2\alpha_0$ observations with sample variance $\beta_0/\alpha_0$.

## 2.6 Point Estimation

A point estimation is the process of identifying the single value that corresponds to the "best possible guess" of some unknown desired quantity from the sample data. The quantity could be a parameter, a function or the future value of a random variable. We consider the function of the sample to be the estimator and the estimate to be the actual value when a sample is taken.

In this section, we briefly review the most common methods of point estimation, some of which are later used in the book.

### *2.6.1 Method of Moments*

The method of moments is straightforward. It relies on solving the system of equations that results from "matching," that is, equating, the sample moments $\{m_i\}_{i=1}^{k}$ to the population moments $\{\mu_i\}_{i=1}^{k}$. The latter typically are functions of the sought parameters $\{\theta_i\}_{i=1}^{k}$. Then, we have $k$ equations:

$$\left(m_i = \frac{1}{n}\sum_{j=0}^{n} X_j^i\right) = \left(\mathbb{E}\left[X^i\right] = \mu_i\right), \forall 1 \leq i \leq k. \tag{2.43}$$

Since this technique consists on matching moments of distributions, it is also called moment matching. We will use it for approximate inference in Sects. 3.2.2 and 3.2.3, which will in turn be quite useful in Chap. 4. As we shall see, it is advisable to use similar distributions when matching moments, otherwise computability is seriously compromised.

### *2.6.2 Maximum Likelihood Estimation*

Maximum Likelihood Estimator (MLE) relies on the maximization of the likelihood function, that is, finding the single parameter value for which the observable data is more likely. We assume data is iid given $\mathcal{Z}$ so $p(\mathbf{x} \mid Z)$ factorizes as follows:

$$
\begin{aligned}
z_{MLE} &= \underset{Z}{\operatorname{argmax}}\, p(\mathbf{x} \mid Z) \\
&= \underset{Z}{\operatorname{argmax}} \prod_i p(x_i \mid Z) \\
&= \underset{Z}{\operatorname{argmax}} \sum_i \log p(x_i \mid Z)\,. \qquad (2.44)
\end{aligned}
$$

The last equality holds as the log is a monotonically increasing function and therefore the optimization 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.

### 2.6.3 Maximum a Posteriori Estimation

Maximum a posteriori (MAP) estimation performs the maximization of the a posteriori function $p(z \mid X)$ defined in Eq. (2.30). From an optimization perspective with respect to the parameters, the evidence is fixed so we can ignore it. Therefore, one may write that:

$$
\begin{aligned}
\mathcal{Z}_{MAP} &= \underset{Z}{\operatorname{argmax}}\, p(z \mid X) \\
&= \underset{Z}{\operatorname{argmax}} \frac{p(x \mid Z)p(z)}{p(x)} \\
&= \underset{Z}{\operatorname{argmax}} \left(p(x \mid Z)p(z)\right) \\
&= \underset{Z}{\operatorname{argmax}} \left(\left[\prod_i p(x_i \mid Z)\right] p(z)\right) \\
&= \underset{Z}{\operatorname{argmax}} \log \left(\left[\prod_i p(x_i \mid Z)\right] p(z)\right) \\
&= \underset{Z}{\operatorname{argmax}} \left(\sum_i \log p(x_i \mid Z) + \log p(z)\right)\,. \qquad (2.45)
\end{aligned}
$$

The MAP forcefully regards $Z$ as a random variable since it also takes into consideration the prior, which is a proper probability density of $Z$. The MAP arrives at a different solution from the MLE, as illustrated in Fig. 2.4. The formula in (2.45) is a common utility function in ML algorithms, e.g., in neural networks the second

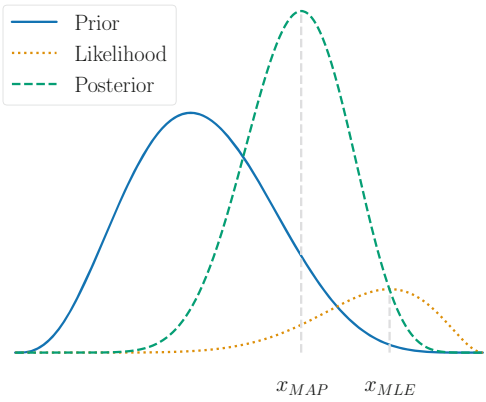



**Fig. 2.4** The solutions for the MLE and MAP are different in general. As more data is gathered, the likelihood becomes expressive and the posterior shifts towards it

term is known as the regularizer and if we take the prior as a standard Gaussian distribution the term reduces to $\ell_2$ regularization.

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.

### *2.6.4 Bayes Estimation*

The full Bayesian estimation is inherently different from the previous approaches where we had an ad-hoc formula to compute the point value. Similarly to the MAP estimator, the procedures start by defining the prior distribution $p(z)$ for the random variable $Z$ we wish to infer. However, we take the marginal distribution $p(x)$ into account, so that we obtain the complete posterior distribution $p(z \mid X)$ instead of a single point in $\mathcal{Z}$.

Naturally, the problem of selecting the best suitable point in $\mathcal{Z}$ arises. To solve such matter, Decision Theory defines the concept of risk $R$ of an estimator $\delta$, which is the expected value of a loss function $L(\delta, Z)$ w.r.t. the posterior distribution $p(z \mid X)$. We thus write

$$R(\delta) = \mathbb{E}_{Z \mid X}\left[L(\delta, Z)\right]. \tag{2.46}$$

The minimization of the risk $R(\delta)$ leads to the estimator $\delta$ whose realizations correspond to the suitable point estimates in $\mathcal{Z}$. The choice of loss function depends on the characteristics of the problem: are large errors just as bad as small ones or considerably worse? Design questions such as this result in different estimators, for example:

- The quadratic loss $L(\delta, Z) = (\delta(X) - Z)^2$ leads to the mean of posterior distribution;
- The absolute loss $L(\delta, Z) = |\delta(X) - Z|$ leads to the median of posterior distribution;
- The $0 - 1$ loss $L(\delta, Z) = \mathbb{1}_{\{\delta:|\delta(X)-Z|>\epsilon\}}$[3] leads to the mode of posterior distribution, just as the MAP;

### 2.6.5 *Expectation-Maximization*

The Expectation-Maximization (EM) is an iterative algorithm that attempts to find the MLE of a parameter $\theta$. The method casts the optimization problem of finding the maximum of a function as a sequence of two simpler problems involving expectation and maximization, which are performed interchangeably at each step. The algorithm excels at handling missing data, what often makes computation difficult, and is widely used in computational statistics.

We can write a distribution $f(x; \theta)$ as the marginal of a "complete-data" distribution $f(x, z; \theta)$ by

$$f(x; \theta) = \int f(x, z; \theta) dz, \tag{2.47}$$

where $Z$ is an unobserved random variable, thus constituting the missing data.

Through the induced conditional distribution $h(z \mid X, \theta)$, we write

$$\log h(Z \mid X, \theta) = \log f(X, z; \theta) - \log f(X; \theta). \tag{2.48}$$

We can take the expectation over the unobserved $Z$, $\left(Z \mid X; \theta^{(0)}\right)$, in (2.48) and rearrange to obtain

$$\log f(X; \theta) = \mathbb{E}_Z \left[\log f(X, Z; \theta) \mid X, \theta^{(0)}\right] - \mathbb{E}_Z \left[\log h(Z \mid X, \theta^{(0)})\right]. \tag{2.49}$$

The second term of the right-hand side of (2.49) may be ignored for the optimization of $f(X; \theta)$ and it suffices to maximize the term $\mathbb{E}_Z \left[\log f(X, Z; \theta) \mid X, \theta^{(0)}\right]$. Thus, we break the computation down into two steps

1. E-step: $Q(\theta \mid \theta^{(r)}, X) = \mathbb{E}_Z \left[\log f(X, Z; \theta) \mid X, \theta^{(r)}\right]$;
2. M-step $\theta^{(r+1)} = \text{argmax}_\theta \; Q(\theta \mid \theta^{(r)}, X)$.

[3] $\mathbb{1}_{(\cdot)}$ is the indicator function, in this case we have $L(\delta, Z) = 1$ if $|\delta(X) - Z| > \epsilon$, and equals to 0, otherwise.

The E-step computes the expectation of the log function over the missing data $Z$ considering the observed data $X$ and $\theta^{(r)}$ fixed. The "M-step" finds the maximum w.r.t. $\theta$.

Although we have deliberately ignored a term in (2.49), the algorithm is guaranteed to improve the parameter's estimation at every iteration. We prove this property by showing that $\mathbb{E}_Z\left[\log f(X, Z; \theta) \mid X, \theta^{(r+1)}\right] \geq \mathbb{E}_Z\left[\log f(X, Z; \theta) \mid X, \theta^{(r)}\right]$ and $\mathbb{E}_Z\left[\log h(Z \mid X, \theta^{(r+1)})\right] \leq \mathbb{E}_Z\left[\log h(Z \mid X, \theta^{(r)})\right]$. The former is guaranteed by construction through the M-step that maximizes that function. The latter we see by noting that

$$
\begin{aligned}
&\mathbb{E}_Z\left[\log h(Z \mid X, \theta^{(r+1)}) \mid X, \theta^{(r)}\right] \leq \mathbb{E}_Z\left[\log h(Z \mid X, \theta^{(r)}) \mid X, \theta^{(r)}\right] \\
&\iff \mathbb{E}_Z\left[\log \frac{h(Z \mid X, \theta^{(r+1)})}{h(Z \mid X, \theta^{(r)})} \mid X, \theta^{(r)}\right] \leq 0 \\
&\iff \mathbb{E}_Z\left[\log \frac{h(Z \mid X, \theta^{(r+1)})}{h(Z \mid X, \theta^{(r)})} \mid X, \theta^{(r)}\right] \leq \log \mathbb{E}_Z\left[\frac{h(Z \mid X, \theta^{(r+1)})}{h(Z \mid X, \theta^{(r)})} \mid X, \theta^{(r)}\right] \\
&\iff \mathbb{E}_Z\left[\log \frac{h(Z \mid X, \theta^{(r+1)})}{h(Z \mid X, \theta^{(r)})} \mid X, \theta^{(r)}\right] \leq \log \int \frac{h(Z \mid X, \theta^{(r+1)})}{h(Z \mid X, \theta^{(r)})} h(Z \mid X, \theta^{(r)}) dZ \\
&\iff \mathbb{E}_Z\left[\log \frac{h(Z \mid X, \theta^{(r+1)})}{h(Z \mid X, \theta^{(r)})} \mid X, \theta^{(r)}\right] \leq \log(1) = 0, \qquad (2.50)
\end{aligned}
$$

where in the third step of Eq. (2.50) we applied the Jensen inequality as $\log(x)$ is concave.

At the first iteration, any guess about $\theta$ is enough to get the algorithm started. Usually, we run the method multiple times from different random starting points $\theta^{(0)}$ to mitigate the local-optimum issue inherent to multi-modal problems. While good guesses enable faster convergence, bad guesses may lead to slower convergence or local optima. The algorithm itself is generally slow with the E-step being the bottleneck, getting slower with higher dimensionality. Besides, the more missing data one has, the more expectations one must take and the slower the method gets.

In spite of the aforementioned disadvantages, the EM method is very useful and employed in a number of different cases where computation of the MLE (or MAP) may be difficult:

- Mixture of distributions;
- Non-conjugate prior;
- Missing or unobserved data;
- Discrete, continuous, or mixed data.

#### 2.6.5.1 EM Example

Since the EM has a strong numerical component, we see fit to illustrate with it an example.

Suppose a factory produces $n$ different products. Let $Y_i$ be the count of registered malfunctioning items of the product $i$ and the root cause for the defects be impurities in the raw materials $X_i$ used for each $i$, which naturally occurs at rate $\tau_i$. We assume the manufacturing process affects all materials equally through the factor $\beta$.

The directors of the factory are doubting the quality of their machinery and have seen a study setting standards for the parameter $\beta$ of a factory. Hence, they want to estimate the efficiency of their own factory to decide whether they should invest in new equipment. Unfortunately, measuring which raw materials are below the required purity level is an expensive procedure and due to budget reasons it is only possible to perform the chemical analysis a small number of times $m < n$. Moreover, not all products have records of the malfunctioning items. Nonetheless, the directors demand a thorough report.

Although we could simply ignore all products for which we do not possess either raw impurity measure or malfunctioning records, the directors would not accept that. Thus, to overcome the missing data, we use the EM algorithm.

We model the counting of malfunctioning items $Y_i$ and the counting of impure materials $X_i$ as two Poisson processes, like

$$X_i \sim \text{Poisson}(\tau_i) \tag{2.51}$$

$$Y_i \sim \text{Poisson}(\beta\tau_i). \tag{2.52}$$

The complete likelihood is given by

$$f(\mathbf{x}, \mathbf{y} \mid \beta, \boldsymbol{\tau}) = \prod_{i=1}^{n} \left( \frac{e^{-\tau_i} \tau_i^{x_i}}{x_i!} \right) \left( \frac{e^{-\beta\tau_i} (\beta\tau_i)^{y_i}}{y_i!} \right). \tag{2.53}$$

For simplicity, we consider $m = n - 1$, with $x_1$ missing. Also, suppose unavailable the record of defective items $y_n$. We write the incomplete-data likelihood as

$$f(\mathbf{x}_{-1}, \mathbf{y} \mid \beta, \boldsymbol{\tau}) = \prod_{i=2}^{n} \left( \frac{e^{-\tau_i} \tau_i^{x_i}}{x_i!} \right) \prod_{i=1}^{n-1} \left( \frac{e^{-\beta\tau_i} (\beta\tau_i)^{y_i}}{y_i!} \right). \tag{2.54}$$

As in Eq. (2.48), defining the conditional $h(x_1, y_n \mid \mathbf{x}_{-1}, \mathbf{y}_{-n}; \boldsymbol{\theta})$, where $\boldsymbol{\theta} = (\beta, \boldsymbol{\tau})$, we write

$$h(x_1, y_n \mid \mathbf{x}_{-1}, \mathbf{y}_{-n}; \boldsymbol{\theta}) = \frac{f(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta})}{f(\mathbf{x}_{-1}, \mathbf{y}_{-n}; \boldsymbol{\theta})}$$

$$\Rightarrow \log f(\mathbf{x}_{-1}, \mathbf{y}_{-n}; \boldsymbol{\theta}) = \log f(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}) - \log h(x_1, y_n \mid \mathbf{x}_{-1}, \mathbf{y}_{-n}; \boldsymbol{\theta})$$

$$\Rightarrow \log f(\mathbf{x}_{-1}, \mathbf{y}_{-n}; \boldsymbol{\theta}) = \mathbb{E}_{X_1, Y_n}\left[\log f(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}) \mid \mathbf{x}_{-1}, \mathbf{y}_{-n}; \boldsymbol{\theta}\right] - \mathbb{E}_{X_1, Y_n}\left[\log h(\mathbf{x} \mid \mathbf{y}; \boldsymbol{\theta}) \mid \mathbf{x}_{-1}, \mathbf{y}_{-n}; \boldsymbol{\theta}\right]. \tag{2.55}$$

The EM algorithm allows us find the most suitable $\boldsymbol{\theta}$ by taking expectations and maximizing $(\log f(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}) \mid \mathbf{x}_{-1}, \mathbf{y}; \boldsymbol{\theta})$. First, we compute the analytical formula for the expected complete-data log-likelihood as

$$\begin{aligned}
&\mathbb{E}_{X_1, Y_n}\left[\log f(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}) \mid \mathbf{x}_{-1}, \mathbf{y}_{-n}; \boldsymbol{\theta}\right] \\
&= \sum_{i=1}^{n-1}\left[-\beta\tau_i + y_i(\log\beta + \log\tau_i) - \log y_i!\right] + \sum_{i=2}^{n}\left[-\tau_i + x_i \log\tau - \log x_i!\right] \\
&\quad + \sum_{x_1=0}^{\infty}\left[-\tau_1 + x_1\log\tau_1 - \log x_1!\right]\frac{e^{-\tau_1^{(r)}}(\tau_1^{(r)})^{x_1}}{x_1!} \\
&\quad + \sum_{y_n=0}^{\infty}\left[-\beta\tau_n + y_n\log\beta\tau_n - \log y_n!\right]\frac{e^{-\beta\tau_n^{(r)}}(\beta\tau_n^{(r)})^{y_n}}{y_n!} \\
&= \left(\sum_{i=1}^{n}\left[y_i(\log\beta + \log\tau_i) - \beta\tau_i\right] + \sum_{i=2}^{n}\left[x_i\log\tau_i - \tau_i\right]\right. \\
&\quad \left. + \sum_{x_1=0}^{\infty}\left[x_1\log\tau_1 - \tau_1\right]\frac{e^{-\tau_1^{(r)}}(\tau_1^{(r)})^{x_1}}{x_1!}\right) - \left(\sum_{i=1}^{n-1}\log y_i! + \sum_{i=2}^{n}\log x_i!\right. \\
&\quad \left. + \sum_{y_n=0}^{\infty}\log y_n!\frac{e^{-\beta\tau_n^{(r)}}(\beta\tau_n^{(r)})^{y_n}}{y_n!} + \sum_{x_1=0}^{\infty}\log x_1!\frac{e^{-\tau_1^{(r)}}(\tau_1^{(r)})^{x_1}}{x_1!}\right).
\end{aligned} \tag{2.56}$$

Since we compute the expectation only to be able to maximize it w.r.t. $\boldsymbol{\theta}$, we can ignore all terms not involving $\boldsymbol{\theta}$, i.e., the second set of parentheses in (2.56). Taking the derivatives w.r.t. $\boldsymbol{\theta}$ and computing the MLEs lead to

$$\hat{\beta}^{(r+1)} = \frac{\hat{\beta}^{(r)}\hat{\tau}_n^{(r)} + \sum_{i=1}^{n-1} y_i}{\sum_{i=1}^{n}\hat{\tau}_i^{(r)}} = \frac{\hat{\beta}^{(r)}\hat{\tau}_n^{(r)} + \sum_{i=1}^{n-1} y_i}{\hat{\tau}_1^{(r)} + \sum_{i=2}^{n} x_i}, \tag{2.57}$$

$$\hat{\tau_1}^{(r+1)} = \frac{\hat{\tau_1}^{(r)} + y_1}{1 + \hat{\beta}^{(r+1)}}, \tag{2.58}$$

$$\hat{\tau}_n^{(r+1)} = \frac{x_n + \hat{\beta}^{(r)}\hat{\tau}_n^{(r)}}{1 + \hat{\beta}^{(r+1)}}, \tag{2.59}$$

$$\hat{\tau}_i^{(r+1)} = \frac{x_i + y_i}{1 + \hat{\beta}^{(r+1)}} \; \forall i \neq 1, n. \tag{2.60}$$

The second equality in (2.57) comes from summing (2.58)–(2.60) and substituting $\hat{\beta}^{(r+1)}$ by (2.57), like

$$\begin{aligned} \sum_i^n \hat{\tau}^{(r+1)} &= \frac{y_1 + x_n + \hat{\tau}_1^{(r)} + \hat{\beta}^{(r)}\hat{\tau}_1^{(r)} + \sum_{i=2}^{n-1} x_i + \sum_{i=2}^{n-1} y_i}{\hat{\beta}^{(r+1)} + 1} \\ &= \frac{\sum_{i=2}^{n} x_i + \sum_{i=1}^{n-1} y_i + \hat{\tau}_1^{(r)} + \hat{\beta}^{(r)}\hat{\tau}_1^{(r)}}{\sum_{i=1}^{n-1} y_i + \sum_{i=1}^{n} \tau_i^{(r)} + \hat{\beta}^{(r)}\hat{\tau}_n^{(r)}} \sum_i^n \hat{\tau}^{(r+1)} \\ \Rightarrow \sum_{i=1}^{n} \tau_i^{(r)} &= \sum_{i=2}^{n} x_i + \hat{\tau}_1^{(r)}. \end{aligned} \tag{2.61}$$

Note that the formulas for $\tau_1$ and $\tau_n$, i.e., (2.58) and (2.59), respectively, are similar to the general formula (2.60) for $\tau_i$ $\forall i \neq i, n$. When the observation is unavailable, we replace it by the mean of the corresponding random variable, which is the rate of Poisson process.

Iterating over Eqs. (2.57)–(2.60) will lead to the stationary point, which must satisfy

$$\hat{\beta} = \frac{\hat{\beta}\hat{\tau}_n + \sum_{i=1}^{n-1} y_i}{\hat{\tau}_1 + \sum_{i=2}^{n} x_i}, \quad \hat{\tau}_1 = \frac{\hat{\tau}_1 + y_1}{1 + \hat{\beta}}, \quad \hat{\tau}_n = \frac{x_n + \hat{\beta}\hat{\tau}_n}{1 + \hat{\beta}}, \quad \hat{\tau}_i = \frac{x_i + y_i}{1 + \hat{\beta}} \; \forall i \neq 1, n. \tag{2.62}$$

After solving the set of equations, we arrive at

$$\hat{\beta} = \sum_{i=1}^{n-1} y_i \Big/ \sum_{i=1}^{n-1} x_i \;, \tag{2.63}$$

$$\hat{\tau}_1 = y_1/\hat{\beta}, \tag{2.64}$$

$$\hat{\tau}_n = x_n, \tag{2.65}$$

$$\hat{\tau}_i = \frac{x_i + y_i}{1 + \hat{\beta}} \; \forall i \neq 1, n, \tag{2.66}$$

which we also achieve by computing the MLE from the incomplete-data likelihood (2.54).

In this simple example, as just pointed out, we could have found the estimates of $\boldsymbol{\theta}$ directly from (2.54). However, this is not usually the case in real scenarios.

## 2.7 Closing Remarks

In this chapter we skimmed through the some of the essentials of statistical inference. Specifically, we focused on parametric models and discussed the exponential family of distributions. For the inference portion, we saw common point estimation methods, i.e., Maximum Likelihood Estimator (MLE), Maximum a posteriori (MAP) and method of moments. Also, we introduced the Expectation-Maximization (EM) algorithm, which we will revisit in Chap. 3. Of fundamental importance for the approximation methods of Chap. 3 is the KL divergence between distributions that we interpreted as the extra cost for transmitting information under the wrong model.

As we have already pointed out, having a good understanding of this subject is paramount for those who wish to work on any ML-related field. Otherwise, all methods will seem mysterious recipes and problem-solving will resemble alchemy. Good comprehensive resource on statistics and statistical inference include [2, 8, 11].

## References

1. Bishop CM (2013) Model-based machine learning. Philos Trans R Soc A: Math Phys Eng Sci 371(1984):1–17
2. Casella G, Berger RL (1990) Statistical inference. Wadsworth and Brooks/Cole, Pacific Grove, CA
3. Cover TM, Thomas JA (2006) Elements of information theory. Wiley-Interscience, New York
4. Deng J, Dong W, Socher R, Li LJ, Li K, Fei-Fei L (2009) ImageNet: a large-scale hierarchical image database. In: IEEE conference on Computer vision and pattern recognition, 2009. CVPR 2009. IEEE, New York, pp 248–255
5. Diaconis P, Freedman D (1980) Finite exchangeable sequences. Ann Probab 8(4):745–764
6. Ghahramani Z (2013) Bayesian non-parametrics and the probabilistic approach to modelling. Philos Trans R Soc A: Math Phys Eng Sci 371
7. Migon HS, Gamerman D, Louzada F (2014) Statistical inference: an integrated approach. CRC Press, Boca Raton
8. Murphy KP (2012) Machine learning: a probabilistic perspective. MIT Press, Cambridge
9. Schervish MJ (1996) Theory of statistics. Springer, New York
10. Shannon CE, Weaver W (1963) A mathematical theory of communication. University of Illinois Press, Champaign, IL
11. Wasserman L (2010) All of statistics: a concise course in statistical inference. Springer Publishing Company, Incorporated, New York

# Chapter 3
# Model-Based Machine Learning and Approximate Inference


By the end of this chapter, the reader should:

- Understand the various advantages of the model-based approach,
- Discern the benefits and issues of Bayesian inference,
- Be capable of understanding and implementing variational Bayes and expectation propagation,
- Understand the mean-field approximation,
- Comprehend the relations between the different variational methods,
- Know the modern landscape of stochastic and black-box inference methods.

## 3.1 Model-Based Machine Learning

Model-Based Machine Learning (MBML) aims at providing a specific solution for each application. It encodes the set of assumptions for a given application explicitly in the model. Consequently, we are able to create a wide range of highly tailored models under a single development framework.

The clear picture of what is the model decouples the model structure from the learning (inference) algorithm. This segregation allows their independent formulation and 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, allowing the derivation of many traditional ML techniques as special cases of certain model-inference configurations (see examples in Sect. 3.1.1).

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



L. P. Cinelli et al., *Variational Methods for Machine Learning with Applications to Deep Networks*, https://doi.org/10.1007/978-3-030-70679-1_3

*Example* 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 min, while $\mathcal{B}$, 17. Which one should the driver choose? Now, the driver further considers that the patient must get to the hospital within 20 min and $\mathcal{A}$ consists of regular urban streets with semaphores and possible traffic jams and that his predicted travel time **may vary** up to eight minutes, whereas $\mathcal{B}$ is an express lane for medical emergencies and the estimated time **varies** by no more than one minute. Would the choice still be the same?

The highlighted keywords above give us a sense of the intrinsic variability in our problem, and disregarding that information 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 more conscious decision. Probability theory provides a principled framework for modeling uncertainty. As seen in our example, probabilistic models allow us to reason and perform decision-making, anticipate the future and plan accordingly, and detect unexpected events, among others; all by learning probability distributions of the data. Not only we can understand almost all ML through probabilistic lenses but also connect it through this perspective to every other computational science [22, p. 2].

### *3.1.1 Probabilistic Graphical Models*

Full joint distributions are generally intractable. Therefore, we resort to structured models [4], which associate probability distributions over only a few variables providing considerable computational simplifications.

One flexible paradigm is Probabilistic Graphical Model (PGM) [12], which use a diagrammatic representation for compactly encoding a complex distribution over a high-dimensional space [12], as depicted in Fig. 3.2, where the most frequent elements are:

- **vertices** or **nodes**: denote random variables (also commonly called nodes), which can be shaded if observed or empty otherwise;
- **edges**: capture the dependency between vertices;
- **plates**: symbolize that the enclosed subgraph is repeated the number of times indicated by the subscript in the bottom right of the plate, as illustrated in Fig. 3.1.

#### 3.1.1.1 Direct Acyclic Graphs

Let $\mathcal{V}$ be the set of all vertices of a graph. Define a parent of the vertex $i$ as the vertex whose directed edge points to $i$. Further define the parent set $\Pi_i$ as the set of all vertices that are parents of the vertex $i$.

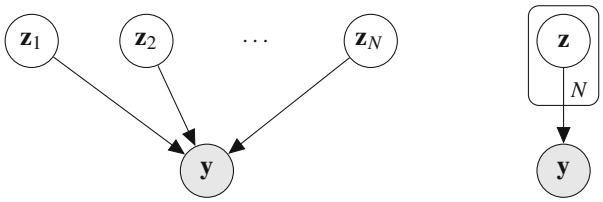



**Fig. 3.1** Equivalence of the shorthand plate notation

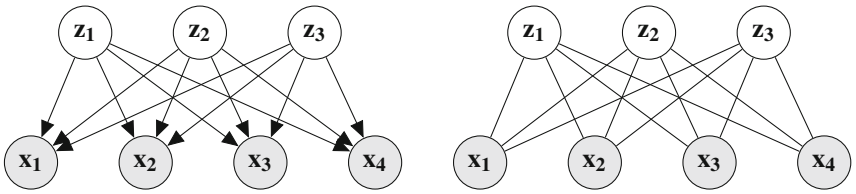



**Fig. 3.2** Examples of probabilistic graphical models. In either cases, vertices represent random variables and edges their dependency relation. Bayesian networks (**a**) encode causation through the edge's direction, i.e., $\mathbf{x}_1$ depends on $\mathbf{z}_1$, $\mathbf{z}_2$, and $\mathbf{z}_3$, being their effect. On the other hand, Markov random fields, also known as undirected graphical models, encode symmetrical dependency through the edges, with no cause–effect relation. (**a**) Bayesian network. (**b**) Markov random field

In the Directed Acyclic Graph (DAG) approach, exemplified in Fig. 3.2a, each vertex $i \in \mathcal{V}$ together with its parent set $\Pi_i$ defines a local probability distribution $p(\mathbf{x}_i \mid \Pi_i)$ over the random variable $\mathbf{x}_i$ associated with the corresponding vertex $i$. The existence of an edge from $i$ to $j$ indicates that $i$ causes $j$, while the absence indicates that the nodes are independent.

The collection of all local probability distributions $p(x_i \mid \Pi_i)$ over the random variables $x_i$ describes the joint probability of the model:

$$p(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{|\mathcal{V}|}) = \prod_{i \in \mathcal{V}} p(\mathbf{x}_i \mid \Pi_i) \ . \tag{3.1}$$

This class of models is frequently referred to as Bayesian network, despite no intrinsic need for Bayesian methods. They are so called because they use the Bayes' rule for defining the probability distributions [23].

### 3.1.1.2 Undirected Graphs

Contrary to DAGs, undirected graphs, exemplified in Fig. 3.2b, have no cause–effect relation between its nodes and cannot describe a generative process. Instead

of describing the full joint distribution in terms of conditionals, undirected graphs factorize the joint distribution over groups $\mathcal{X}_c$ of fully connected nodes (maximal cliques), each characterized by a potential function $\psi(\mathcal{X}_c)$ [12]. The potential function of a group is not a valid probability distribution, but the set $C$ of all such groups gives the joint distribution according to

$$p(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{|\mathcal{V}|}) = \frac{1}{Z} \prod_{c \in C} \psi(\mathcal{X}_c), \tag{3.2}$$

where $Z$ is the normalizing constant.

#### 3.1.1.3 The Power of Graphical Models

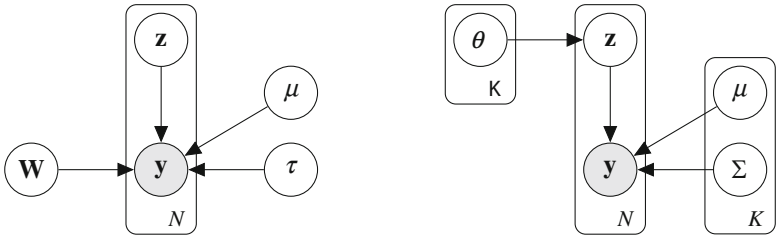
Many traditional ML and signal processing algorithms can be derived as special cases of graphical models combined with the appropriate inference algorithms. Moreover, many of them can be represented by simple graphical structures and be effortlessly combined [4]. For example:

1. Principal Component Analysis (PCA) can be formulated as a generative process with the latent variable $\mathbf{z}$ corresponding to the principal component subspace (Fig. 3.3a). Observations $\mathbf{y}$ are noisy versions of the linear mapping $\mathbf{W}\mathbf{z} + \boldsymbol{\mu}$, where $\tau$ is the noise precision, which is the same for all directions. Assuming all distributions to be Gaussian, using MLE for determining $\mathbf{W}$ and $\boldsymbol{\mu}$, and taking the limit $\tau \to \infty$, one obtains the standard PCA model [3, ch. 12].
2. Gaussian Mixture Model (GMM) (Fig. 3.3b) with $K$ modes is represented by a $K$-dimensional latent indicator variable $\mathbf{z}$ that follows a categorical distribution with probabilities, the mixture weights, given by $\theta$. Each mode has its own mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\Sigma}$. For other types of mixture, the difference is in the type of parameters that define the $K$ modes, i.e., $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$.

PGMs can be easily customized to a specific application or modified if the requirements of the application change.

### *3.1.2 Probabilistic Programming*

Probabilistic programming is a tool for statistical modeling. It borrows lessons from computer science and common programming languages to construct languages that allow the denotation and evaluation of inference problems [32]. It frees the developer from complex low-level details of probabilistic inference, allowing him or 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




(a) Principal component analysis. (b) Gaussian mixture model.

**Fig. 3.3** Probabilistic graphical models of traditional machine learning algorithms. In the PCA model (**a**), the principal component subspace is represented by the latent variable $\mathbf{z}$, which cannot be directly observed. We only know the $\mathbf{y}$ values that are noisy observations of the true underlying generative $\mathbf{y} = \mathbf{W}\mathbf{z} + \boldsymbol{\mu} + \mathcal{N}(0, \tau)$. In the GMM (**b**), each of the $K$ modes has its set of unknown defining parameters, i.e., $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, and is selected by the latent indicator variable $\mathbf{Z}$, whose observation probability is $\theta$

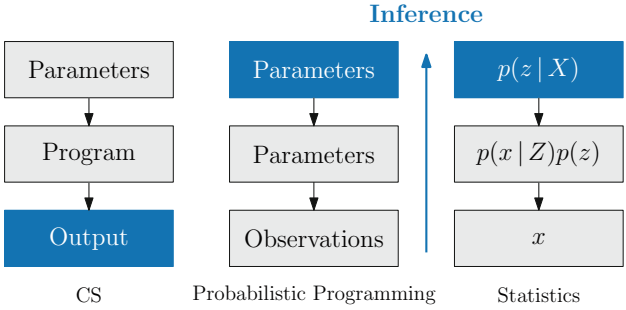



**Fig. 3.4** An intuitive view of probabilistic programming and how it differs from the common computer science paradigm. Shaded boxes indicate the information that is available. Instead of inputting the required parameters to run the program and obtain the desired output, probabilistic programming tries to recover the parameters from the observations generated by the program. This process is similar to inference in statistics

that abstract away architecture-specific implementation details, it boosts system performance and productivity. In Fig. 3.4 we draw a parallel between computer science, statistics, and probabilistic programming.

One of the cornerstones for the deep learning success was the development of specialized programming libraries that facilitate model specification and automatize differentiation, relieving the user from the need of manually deriving the gradients for optimization. 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 differential calculus to try and run a model. Still, it does not mean that whatever the model may be it will be useful or meaningful. Probabilistic programming aims to achieve the same for probabilistic ML [32]. It also allows rapid prototyping and testing of ideas, allowing the field to flourish and pushing industry adoption.

Modern probabilistic programming languages provide a more powerful framework than PGM. Computer programs accept recursion and control flow statements that are otherwise difficult to represent [7]. There is a myriad of different languages, each with its own set of specific features: some are explicitly restrictive, others specialize in a certain type of inference techniques, or yet are general purpose. A non-extensive list includes Pyro [2], Stan [6], WebPPL [8], Infer.NET [21], PyMC3 [28], Edward [30], and BUGS (from 1995) [33].

## 3.2 Approximate Inference

As briefly alluded in the previous section, it is frequently unfeasible to compute posterior distributions and marginals for many models of practical interest. In the continuous case, there may not be a closed-form analytical solution or it may be just too complex for numerical computation. In the discrete case, summing over all possible configurations, though possible in principle, may not be viable if the total number of combinations grows exponentially.

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 [31, 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 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 distribution and, as a consequence, cannot generate exact results. Hence, both methods are complementary.

In this chapter, 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).

### *3.2.1 Variational Inference*

VI, Variational Bayes, and Variational Bayesian Inference are different names for the same algorithm. Its purpose is to construct a deterministic analytical approximation to the posterior distribution. Thus, it is suited to large data sets and to quickly test many models [5]. As other Bayesian methods, it describes all available information about the variables through their probability distributions. Figure 3.5 depicts how VI works: it iteratively finds the best possible distribution $q^*$ among the specified family $\boldsymbol{Q}$, given a dissimilarity criterion $D$.

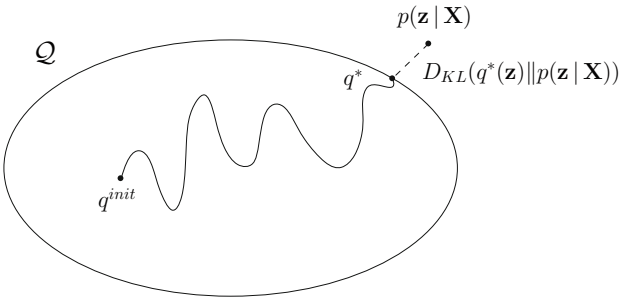



**Fig. 3.5** Illustration of VI optimization process given a family $\mathcal{Q}$ of distributions that does not contain the true posterior distribution $p(\mathbf{z} \mid \mathbf{X})$. The best possible approximation $q^*$ the variational posterior $q$ can achieve is the one that minimizes the chosen dissimilarity criterion $D$, i.e., the KL divergence $D_{KL}\left(q(\mathbf{z} \mid \mathbf{X}) \| p(\mathbf{z} \mid \mathbf{X})\right)$

VI borrows its name from variational calculus. Regular calculus concentrates on maxima, minima, and derivatives of functions, while variational calculus does that for functionals, which are basically functions of functions. Several problems can be cast as functional optimization problems, and variational methods do exactly that for inference: they allow us to find a function, the approximating distribution $q$, that minimizes the quality measure functional $D$.

#### 3.2.1.1 The Evidence Lower Bound

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

$$q^*(\mathbf{z} \mid \mathbf{X}) = \underset{q \in \mathcal{Q}}{\operatorname{argmin}}\, D_{KL}\left(q(\mathbf{z} \mid \mathbf{X}) \| p(\mathbf{z} \mid \mathbf{X})\right), \tag{3.3}$$

where

$$D_{KL}\left(q \| p\right) = \int q(\epsilon) \log \frac{q(\epsilon)}{p(\epsilon)} d\epsilon. \tag{3.4}$$

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

$$
\begin{aligned}
D_{KL}\left(q(\mathbf{z}\,|\,\mathbf{X})\|p(\mathbf{z}\,|\,\mathbf{X})\right)) &= \int q(\mathbf{z}\,|\,\mathbf{X}) \log\left(\frac{q(\mathbf{z}\,|\,\mathbf{X})}{p(\mathbf{z}\,|\,\mathbf{X})}\right) d\mathbf{z} \\
&= -\int q(\mathbf{z}\,|\,\mathbf{X}) \log\left(\frac{p(\mathbf{x},\mathbf{z})}{p(\mathbf{x})q(\mathbf{z}\,|\,\mathbf{X})}\right) d\mathbf{z} \\
&= -\left(\int q(\mathbf{z}\,|\,\mathbf{X}) \log\left(\frac{p(\mathbf{x},\mathbf{z})}{q(\mathbf{z}\,|\,\mathbf{X})}\right) d\mathbf{z} - \int q(\mathbf{z}\,|\,\mathbf{X}) \log p(\mathbf{x}) d\mathbf{z}\right) \\
&= -\int q(\mathbf{z}\,|\,\mathbf{X}) \log\left(\frac{p(\mathbf{x},\mathbf{z})}{q(\mathbf{z}\,|\,\mathbf{X})}\right) d\mathbf{z} + \log p(\mathbf{x}) \int q(\mathbf{z}\,|\,\mathbf{X}) d\mathbf{z} \\
&= -\mathbb{E}_q\left[\log\left(\frac{p(\mathbf{x},\mathbf{z})}{q(\mathbf{z}\,|\,\mathbf{X})}\right)\right] + \log p(\mathbf{x}). \qquad (3.5)
\end{aligned}
$$

Reorganizing the last equation, we obtain

$$
\log p(\mathbf{x}) = \mathbb{E}_q\left[\log\left(\frac{p(\mathbf{x},\mathbf{z})}{q(\mathbf{z}\,|\,\mathbf{X})}\right)\right] + D_{KL}(q(\mathbf{z}\,|\,\mathbf{X})\|p(\mathbf{z}\,|\,\mathbf{X})). \qquad (3.6)
$$

Knowing that $D_{KL}(q\|p) \geqslant 0$, it follows that the first term of the right-hand side of Eq. (3.6) must be a lower bound on $\log p(\mathbf{x})$. For this reason, it is named the Evidence Lower Bound (ELBO). This remark leads to a very important result: since the model evidence $\log p(\mathbf{x})$ is fixed, once we know $\mathbf{x}$, by maximizing the ELBO we are equivalently minimizing $D_{KL}(q\|p)$, our original optimization problem. The equivalence is very convenient because the right-hand side of Eq. (3.6) does not contain the intractable log evidence. The term $\log p(\mathbf{x},\mathbf{z})$ decomposes into the log-likelihood $\log p(\mathbf{x}\,|\,\mathbf{Z})$ and the log-prior $\log p(\mathbf{z})$, which we are able to handle.

Alternatively, we could have obtained the same bound by applying Jensen's inequality for concave functions $\mathbb{E}\left[f(\mathbf{x})\right] \leqslant f(\mathbb{E}\left[\mathbf{x}\right])$ as follows:

$$
\begin{aligned}
\log p(\mathbf{x}) &= \log \int p(\mathbf{x},\mathbf{z}) d\mathbf{z} \\
&= \log \int p(\mathbf{x},\mathbf{z}) \frac{p(\mathbf{z}\,|\,\mathbf{X})}{q(\mathbf{z}\,|\,\mathbf{X})} d\mathbf{z} \\
&= \log \mathbb{E}_q\left[\frac{p(\mathbf{x},\mathbf{z})}{q(\mathbf{z}\,|\,\mathbf{X})}\right] \\
&\geqslant \mathbb{E}_q\left[\log\left(\frac{p(\mathbf{x},\mathbf{z})}{q(\mathbf{z}\,|\,\mathbf{X})}\right)\right]. \qquad (3.7)
\end{aligned}
$$

By comparison with Eq. (3.6), the difference between the left- and right-hand sides of Eq. (3.7) is exactly the KL divergence term, as shown in Fig. 3.6. The visual depiction clearly illustrates the equivalence between the minimization of $D_{KL}\left(q(\mathbf{z}\,|\,\mathbf{X})\|p(\mathbf{z}\,|\,\mathbf{X})\right))$ and the maximization of the ELBO(q).

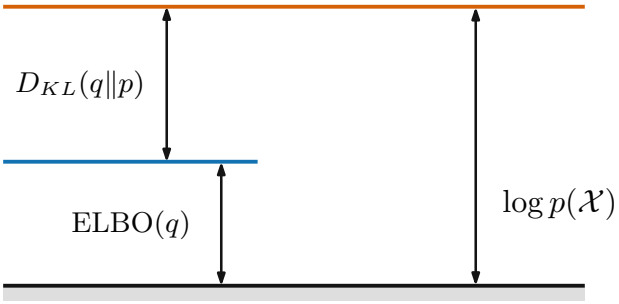



**Fig. 3.6** The decomposition of the marginal log-probability $p(\mathbf{x})$ into the ELBO and the $D_{KL}(q\|p)$ terms

We can rearrange the ELBO into the more interpretable form:

$$\begin{aligned}\text{ELBO}(q) &= \mathbb{E}_q\left[\log p(\mathbf{x},\mathbf{z})\right] - \mathbb{E}_q\left[\log q(\mathbf{z}\,|\,\mathbf{X})\right] \\ &= \mathbb{E}_q\left[\log p(\mathbf{x}\,|\,\mathbf{Z}) + \log p(\mathbf{z})\right] - \mathbb{E}_q\left[\log q(\mathbf{z}\,|\,\mathbf{X})\right] \\ &= \mathbb{E}_q\left[\log p(\mathbf{x}\,|\,\mathbf{Z})\right] - D_{KL}\left(q(\mathbf{z}\,|\,\mathbf{X})\|p(\mathbf{z})\right). \end{aligned} \tag{3.8}$$

The first term is the expected likelihood under the distribution $q(\mathbf{z}\,|\,\mathbf{X})$ and the second is the (negative) divergence between the $q(\mathbf{z}\,|\,\mathbf{X})$ and the prior $p(\mathbf{z})$. When maximizing the ELBO, the former drives the approximation toward better explaining the data, while the latter acts as a regularizer pushing the approximation toward the prior $p(\mathbf{z})$.

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

$$\begin{aligned}\text{ELBO}(q) &= \mathbb{E}_q\left[\log p(\mathbf{x},\mathbf{z})\right] - \mathbb{E}_q\left[\log q(\mathbf{z}\,|\,\mathbf{X})\right] \\ &= \mathbb{E}_q\left[\log p(\mathbf{x},\mathbf{z})\right] + \mathcal{H}[q] \end{aligned} \tag{3.9}$$

$$\begin{aligned}\widetilde{F}(q) &= -\mathbb{E}_q\left[\log p(\mathbf{x},\mathbf{z})\right] - \mathcal{H}[q] \\ &= -\text{ELBO}(q), \end{aligned} \tag{3.10}$$

where $-\mathbb{E}_q\left[\log p(\mathbf{x},\mathbf{z})\right]$ is the average of the energy function under the distribution $q(\mathbf{z}\,|\,\mathbf{X})$ and $\mathcal{H}[q]$ is the entropy of $q(\mathbf{z}\,|\,\mathbf{X})$ [16, ch. 33]. Indeed, the use of the variational free energy framework in statistical learning leads to the VI methodology.

The optimal solution for $q$ in Eq. (3.9) w.r.t. the term $\mathbb{E}_q\left[\log p(\mathbf{x},\mathbf{z})\right]$ corresponds to the MAP estimate of $p$, which maximizes the log joint probability $\log p(\mathbf{x},\mathbf{z})$. However, the entropy term favors disperse distributions. The solution is then a compromise between these two terms.

#### 3.2.1.2 Information Theoretic View on the ELBO

In its very essence, the rate-distortion theory establishes the trade-off between data compression and the entailed distortion [1]. The rate represents the average number of bits needed per sample to transmit the data. Ideally, one wants to maximally compress the data, achieving compact representations with low rates, while preserving all relevant information, such that the reconstructed signal has no distortion whatsoever. However, these are opposite goals.

Clustering algorithms can be naturally seen through the rate-distortion perspective. In K-means [14], the rate is related to the number of centroids and the distortion measure is the sum of the squared error between the original data points and the centroid of their attributed cluster.

Rate-distortion theory asserts that for a given maximum level of distortion $D$, there exists a minimum achievable rate $R$. Thus, for the input random variable $X$ and the compressed output $Z$, we have

$$\begin{aligned} R(D) = & \underset{q(\mathbf{z}\,|\,\mathbf{X})}{\operatorname{argmin}} I(\mathbf{X};\mathbf{Z}) \\ & s.t.\ \mathbb{E}_{p(\mathbf{x})}\left[\mathbb{E}_q\left[d(\mathbf{Z},\mathbf{X})\right]\right] < D, \end{aligned} \tag{3.11}$$

where $d(\cdot,\cdot)$ is the distortion measure (e.g., sum of squared errors in K-means), $I(\mathbf{X};\mathbf{Z})$ the mutual information, and $q(\mathbf{z}|\mathbf{X})$ the channel we wish to optimize.

Introduced in Sect. 2.3.4, the mutual information $I(\mathbf{X};\mathbf{Z})$ between the random variables $X$ and $Z$ quantifies their dependency, that is, how much can we know about one by observing the other. Intuitively, Eq. (3.11) seeks to remove as much information as possible from $\mathbf{X}$, making it independent of $\mathbf{Z}$.

To make the optimization problem manageable, we upper bound $I(\mathbf{X};\mathbf{Z})$ as follows:

$$\begin{aligned} I(\mathbf{X};\mathbf{Z}) &= D_{KL}\left(q(\mathbf{z},\mathbf{x})\|q(\mathbf{z})p(\mathbf{x})\right) \\ &= \mathbb{E}_{p(\mathbf{x})}\left[D_{KL}\left(q(\mathbf{z}\,|\,\mathbf{X})\|m(\mathbf{z})\right)\right] - D_{KL}\left(q(\mathbf{z})\|p(\mathbf{z})\right) \qquad (3.12) \\ &\leq \mathbb{E}_{p(\mathbf{x})}\left[D_{KL}\left(q(\mathbf{z}\,|\,\mathbf{X})\|m(\mathbf{z})\right)\right], \qquad (3.13) \end{aligned}$$

where $q(\mathbf{z})$ is the induced marginal $q(\mathbf{z}) = \int q(\mathbf{z},\mathbf{x})p(\mathbf{x})d\mathbf{x}$, $m(\mathbf{z})$ is an approximation to $q(\mathbf{z})$, and the inequality stems from the nonnegativity of the KL divergence.

For latent variable models, the implicitly defined distortion function is $d(\mathbf{X},\mathbf{Z}) = -\log p(\mathbf{x}\,|\,\mathbf{Z})$. This distortion penalizes latent variables $\mathbf{Z}$ unable to faithfully reconstruct the original sample $\mathbf{x}$. If we further set the marginal approximation $m(\mathbf{z})$ as the prior $p(\mathbf{z})$ over the compressed random variable $\mathbf{Z}$, the optimization problem becomes

$$\min_{q(\mathbf{z}\,|\,\mathbf{X})} \mathbb{E}_{p(\mathbf{x})}\left[D_{KL}\left(q(\mathbf{z}\,|\,\mathbf{X})\|p(\mathbf{z})\right)\right] \tag{3.14}$$

$$s.t.\ \mathbb{E}_{p(\mathbf{x})}\left[\mathbb{E}_q\left[-\log p(\mathbf{x}\,|\,\mathbf{Z})\right]\right] < D.$$

Rewriting Eq. (3.14) as a maximization problem and stating it in terms of its Lagrangian lead to

$$\max_{q(\mathbf{z}\,|\,\mathbf{X})} \mathbb{E}_{p(\mathbf{x})}\left[\mathbb{E}_q\left[\log p(\mathbf{x}\,|\,\mathbf{Z})\right] - \beta D_{KL}\left(q(\mathbf{z}\,|\,\mathbf{X})\|p(\mathbf{z})\right)\right], \tag{3.15}$$

where $\beta$ is the Lagrange multiplier.

Solving Eq. (3.15) is equivalent to maximizing the average ELBO in Eq. (3.8) for the data set $\mathcal{D} = \{\mathcal{X}\}_n$ with empirical distribution $p(\mathbf{d})$ and $\beta = 1$. Thus, we can interpret Variational Bayes as optimizing an upper bound on the distortion-rate function. While the $\mathbb{E}_q\left[\log p(\mathbf{x}\,|\,\mathbf{Z})\right]$ term measures the fidelity (negative distortion) of the compressed representation, the KL term quantifies the extra number of bits needed to represent $\mathcal{X}$ with $\mathcal{Z}$. The connection allows us to leverage insights from the well-established field of information theory onto variational Bayes. For example, there is an upper bound to the ELBO, and its value is the negative of the entropy of the true data distribution, $-\mathcal{H}[p(\mathbf{x})]$.

#### 3.2.1.3 The Mean-Field Approximation

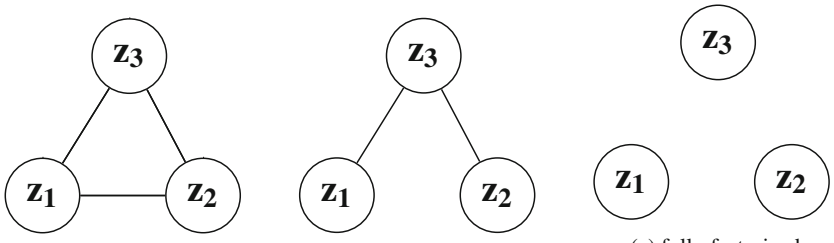
No matter the kind of inference algorithm, we usually impose restrictions to the family of approximating distributions $\mathcal{Q}$ so that we can solve the problem. The family $\mathcal{Q}$ should be as flexible as possible to allow us to achieve better approximations of the true posterior, the only restriction being its tractability. The richer the family of distributions, the closer $q^*(\mathbf{z}\,|\,\mathbf{X})$ will be to $p(\mathbf{z}\,|\,\mathbf{X})$. In cases where $\mathcal{Q}$ does include the true posterior and the latter is tractable, the inference methods generally converge to the exact distribution.

There are two main ways to constrain the family of distributions of a model:

1. by specifying a parametric form for the distribution $q(\mathbf{z}\,|\,\mathbf{X};\Psi)$ with the set $\Psi$ of variational parameters;
2. by assuming that $q$ factorizes over partitions $\mathcal{Z}_{\mathcal{S}_i}$ of $\mathcal{Z}$ such that

$$q(\mathbf{z}\,|\,\mathbf{X}) = \prod_{i=1}^{M} q_i(\mathbf{z}_{\mathcal{S}_i}\,|\,\mathbf{X}). \tag{3.16}$$

The factorized form of Eq. (3.16) where each partition is a single dimension is called Mean-Field VI (MFVI). The mean-field approximation is flexible enough to capture any marginal density of the latent variables but is incapable of modeling correlation between them due to the independence assumption, as illustrated in Fig. 3.7. This assumption is a double-edged sword, helping with scalable optimization while limiting expressibility [5]. Hence the need for other families of approximations




(a) True posterior. (b) structured approximation. (c) fully-factorized approximation.

**Fig. 3.7** Graphical representations as undirected graphs of the different levels of approximation to the posterior distribution. In (**a**), the nodes in the true posterior are all dependent. In (**b**), $\mathbf{Z}_1$ and $\mathbf{Z}_2$ are conditionally independent, and the approximation still preserves their dependency on $\mathbf{Z}_3$. In (**c**), all the nodes are marginally independent. Each approximation renders the distribution less expressive

such as structured mean-field [10], richer covariance models [15, 29], normalizing flow [26], etc.

#### 3.2.1.4 Coordinate Ascent Variational Inference

Coordinate Ascent Variational Inference (CAVI) is an algorithm for MFVI. To find the optimal factors $q_i^*(\mathbf{z}_{\mathcal{S}_i} \mid \mathbf{X})$ for Eq. (3.16), we could solve the Lagrangian composed by the ELBO and the constraints that the factors $q_i^*$ must sum up to 1. However, we do not resort to the calculus of variations. Instead, we take a more laborious route by substituting Eq. (3.16) back into Eq. (3.9) and working out the math (available in Appendix A.2) to get

$$\log q_j^*(\mathbf{z}_{\mathcal{S}_j} \mid \mathbf{X}) = \mathbb{E}_{-j}\left[\log p(\mathbf{x}, \mathbf{z})\right] + \text{const} \tag{3.17}$$

$$q_j^*(\mathbf{z}_{\mathcal{S}_j} \mid \mathbf{X}) \propto \exp\{\mathbb{E}_{-j}\left[\log p(\mathbf{x}, \mathbf{z})\right]\}, \tag{3.18}$$

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

The mutual dependence between the equations for the optimal factors calls for an iterative approach. At each step, we replace each factor by its revised estimate while keeping the others fixed (3.18). CAVI raises the ELBO to a local optimum. An alternative approach to optimization is through gradient-oriented updates, in which the algorithm computes and follows the gradient of the objective w.r.t. the factors at each iteration.

Although we considered all parameters to be within the latent space $\mathbf{Z}$, it is also possible to have parameters $\Theta$ on which we perform point estimation, i.e., $p(\mathbf{z} \mid \mathbf{X}; \Theta)$. In this case, we alternate between two distinct steps:

1. approximating the posterior at each iteration by computing the *expectation* over all $\mathcal{Z}_{\mathcal{S}_i}$ as in Eq. (3.18);

2. performing the *maximization* of the ELBO w.r.t. $\Theta$ under the refined distribution $q^{new}(\mathbf{z} \mid \mathbf{X}) = \prod_i q_i^*(\mathbf{z}_{S_i} \mid \mathbf{X})$.

This is the Variational EM algorithm. 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 $\Theta$ (MAP estimation, Sect. 2.6.3), it computes the entire distribution over $\Theta$ and $\mathbf{Z}$.

#### 3.2.1.5 Stochastic Variational Inference

Stochastic Variational Inference (SVI) optimizes the ELBO by taking noisy estimates of the gradient $g$ [11], hence the name. Stochastic optimization is ubiquitous on modern ML since it is much faster than assessing a massive data set, which is commonplace nowadays.

The major requirements for the approximation to be valid are:

1. The gradient estimator $\hat{g}$ should be unbiased $\mathbb{E}\left[\hat{g}\right] = \mathbb{E}\left[g\right]$;
2. The step size sequence $\{\alpha_i \mid i \in \mathcal{N}\}$ (learning rate) that nudges the parameters toward the optimal should be annealed so that

$$\sum_{i=0}^{\infty} \alpha_i = \infty \qquad \text{and} \qquad \sum_{i=0}^{\infty} \alpha_i^2 < \infty. \tag{3.19}$$

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

Instead of computing the expectation step in Eq. (3.18) for all $N$ data points (at every iteration), we do it for a uniformly sampled (with replacement) subset of desired size $n$. From these new variational parameters, we compute the maximization 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 Eq. (3.19).

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

SVI is a stochastic optimization algorithm originally developed for fully factorized approximations (MFVI) [11] and later extended to support models with arbitrary dependencies between global and local variables [10].

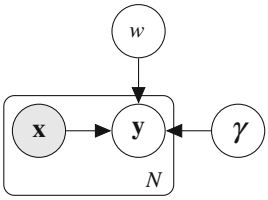



**Fig. 3.8** Graphical representation of a linear regression model with $N$ observations and one weight. The variable $\gamma$ is the observation noise precision

#### 3.2.1.6 VI Issues

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. Outside this family, we end up with distributions for which we cannot write down formulas to optimize. Section 3.2.4 briefly presents methods that address this problem.

Even though minimizing the $D_{KL}\left(q(\mathbf{z}\,|\,\mathbf{X})\|p(\mathbf{z}\,|\,\mathbf{X})\right)$ and maximizing the ELBO are equivalent optimization problems, the KL is bounded below by zero, while the ELBO has no bound whatsoever. Therefore, observing how close the KL is to zero informs us about the quality of 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. Still, it asymptotically converges so we can use the value for model selection.

Minimization of the KL divergence combined with the independence assumption of the mean-field approximation causes the approximating distribution to match a single mode of the target distribution. Additionally, this combo underestimates the marginal variances of the target density [5].

#### 3.2.1.7 VI Example

Consider a one-dimensional linear regression problem where the weight has a Gaussian prior distribution with mean $\mu$ and precision $\tau$. We wish to infer the marginal posterior of the observation noise precision $\gamma$, whose prior follows a Gamma distribution. The model is given by

$$\Gamma \sim \text{Ga}(\gamma; \alpha_0, \beta_0) \tag{3.20}$$

$$W \sim \mathcal{N}(w\,|\,\mu, \tau^{-1}) \tag{3.21}$$

$$Y_i \sim Wx_i + \mathcal{N}(0, \gamma^{-1}), 1 \leq i \leq N. \tag{3.22}$$

Observing the graphical model in Fig. 3.8 and its dependency structure, we can write the joint distribution as

$$p(w, \gamma, \,|\, y_1, \cdots, y_N\mathbf{X}) = p(y_1, \cdots, y_N\,|\,W, \gamma, \mathbf{X})p(\gamma)p(w). \tag{3.23}$$

With the objective of using the CAVI algorithm introduced in Sect. 3.2.1.4, we approximate the posterior $p(\gamma, w|\mathbf{Y}, \mathbf{X})$ over the global variables $w$ and $\gamma$ by $q(\gamma, w) = q(\gamma)q(w)$. The distribution of real interest is the marginal $q(\gamma)$.

From Eq. (3.23) and the assumption of independent and identically distributed (iid) observation samples $x_i$, the true posterior distribution is

$$p(\gamma, w \mid \mathbf{Y}, \mathbf{X}) = \frac{p(\gamma)p(w)}{p(y_1, \cdots, y_N \mid \mathbf{X})} \prod_{i=1}^{N} p(y_i \mid W, \gamma, x_i), \tag{3.24}$$

while the marginal on $\gamma$ is

$$p(\gamma \mid \mathbf{Y}, \mathbf{X}) = \int p(\gamma \mid W, \mathbf{Y}, \mathbf{X}) dw. \tag{3.25}$$

To compute the CAVI's update formula for $q(w)$, we substitute Eq. (3.23) into Eq. (3.18) and label all terms not involving $w$ as constants, what leads to

$$\begin{aligned}
\log q^*(w) &= \mathbb{E}_\gamma \left[\log p(w, \gamma, \mathbf{y} \mid \mathbf{X})\right] + \text{const} \\
&= \mathbb{E}_\gamma \left[\log p(y_1, \cdots, y_N \mid W, \gamma, \mathbf{X})\right] + \mathbb{E}_\gamma \left[\log p(w)\right] + \mathbb{E}_\gamma \left[\log p(\gamma)\right] + \text{const} \\
&= \mathbb{E}_\gamma \left[\log \mathcal{N}(y_1, \cdots, y_N \mid \mathbf{W}^T\mathbf{X}, \gamma^{-1})\right] + \mathbb{E}_\gamma \left[\log \mathcal{N}(w \mid \mu, \tau^{-1})\right] + \text{const} \\
&= \mathbb{E}_\gamma \left[\frac{1}{2}\log\gamma - \frac{1}{2}\log 2\pi - \frac{\gamma}{2}(\mathbf{y} - w\mathbf{x})^T(\mathbf{y} - w\mathbf{x})\right] \\
&+ \mathbb{E}_\gamma \left[\frac{1}{2}\log\tau - \frac{1}{2}\log 2\pi - \frac{\tau}{2}(w - \mu)^2\right] + \text{const} \\
&= \mathbb{E}_\gamma \left[-\frac{\gamma}{2}(\mathbf{y} - w\mathbf{x})^T(\mathbf{y} - w\mathbf{x})\right] - \frac{\tau}{2}(w - \mu)^2 + \text{const} \\
&= -\frac{1}{2}\left\{\mathbb{E}_\gamma [\gamma]\left[(\mathbf{y} - w\mathbf{x})^T(\mathbf{y} - w\mathbf{x})\right] + \tau(w - \mu)^2\right\} + \text{const} \\
&= -\frac{1}{2}\left\{\mathbb{E}_\gamma [\gamma]\left(w^2\mathbf{x}^T\mathbf{x} - 2w\mathbf{x}^T\mathbf{y}\right) + \tau w^2 - 2\tau w\mu\right\} + \text{const} \\
&= -\frac{1}{2}\left[(\mathbf{x}^T\mathbf{x}\mathbb{E}_\gamma [\gamma] + \tau)w^2 - 2(\mathbf{x}^T\mathbf{y}\mathbb{E}_\gamma [\gamma] + \tau\mu)w\right] + \text{const} \\
&= -\frac{\mathbf{x}^T\mathbf{x}\mathbb{E}_\gamma [\gamma] + \tau}{2}\left[w^2 - 2\frac{\mathbf{x}^T\mathbf{y}\mathbb{E}_\gamma [\gamma] + \tau\mu}{\mathbf{x}^T\mathbf{x}\mathbb{E}_\gamma [\gamma] + \tau}w\right] + \text{const} \\
&= -\frac{\mathbf{x}^T\mathbf{x}\mathbb{E}_\gamma [\gamma] + \tau}{2}\left(w - \frac{\mathbf{x}^T\mathbf{y}\mathbb{E}_\gamma [\gamma] + \tau\mu}{\mathbf{x}^T\mathbf{x}\mathbb{E}_\gamma [\gamma] + \tau}\right)^2 + \text{const},
\end{aligned} \tag{3.26}$$

where we considered $\mathbf{y} = [y_1, \cdots, y_N]^t$ and $\mathbf{x} = [x_1, \cdots, x_N]^t$. Note that Eq. (3.26) is the log of the Gaussian distribution's kernel, so we write

$$q^*(w) = \mathcal{N}\left(w \,\middle|\, \frac{\mathbf{x}^T\mathbf{y}\mathbb{E}_\gamma[\gamma] + \tau\mu}{\mathbf{x}^T\mathbf{x}\mathbb{E}_\gamma[\gamma] + \tau}, \left(\mathbf{x}^T\mathbf{x}\mathbb{E}_\gamma[\gamma] + \tau\right)^{-1}\right). \tag{3.27}$$

Applying the same procedure to $q(\gamma)$, we obtain

$$\begin{aligned}
\log q^*(\gamma) &= \mathbb{E}_w\left[\log p(\mathbf{y} \mid w, \gamma, \mathbf{x})\right] + \mathbb{E}_w\left[\log p(\gamma)\right] + \mathbb{E}_w\left[\log p(w)\right] + \text{const} \\
&= \mathbb{E}_w\left[\log \mathcal{N}(\mathbf{y} \mid \mathbf{w}^T\mathbf{x}, \gamma^{-1})\right] + \mathbb{E}_w\left[\text{Ga}(\gamma; \alpha_0, \beta_0)\right] + \text{const} \\
&= \frac{1}{2}\log\gamma - \frac{\gamma}{2}\mathbb{E}_w\left[(\mathbf{y} - w\mathbf{x})^T(\mathbf{y} - w\mathbf{x})\right] \\
&+ \mathbb{E}_w\left[\alpha_0\log\beta_0 - \log\Gamma(\alpha_0) + (\alpha_0 - 1)\log\gamma - \beta_0\gamma\right] + \text{const} \\
&= \frac{1}{2}\log\gamma - \frac{\gamma}{2}\mathbb{E}_w\left[(\mathbf{y} - w\mathbf{x})^T(\mathbf{y} - w\mathbf{x})\right] + (\alpha_0 - 1)\log\gamma - \beta_0\gamma + \text{const} \\
&= \left(\frac{1}{2} + \alpha_0 - 1\right)\log\gamma - \left(\frac{1}{2}\mathbb{E}_w\left[(\mathbf{y} - w\mathbf{x})^T(\mathbf{y} - w\mathbf{x})\right] + \beta_0\right)\gamma + \text{const}.
\end{aligned} \tag{3.28}$$

Note that Eq. (3.28) is the log of the Gamma distribution's kernel, so we write

$$q^*(\gamma) = \text{Ga}\left(\gamma \,\middle|\, \alpha_0 + \frac{1}{2}, \frac{1}{2}\mathbb{E}_w\left[(\mathbf{y} - w\mathbf{x})^T(\mathbf{y} - w\mathbf{x})\right] + \beta_0\right). \tag{3.29}$$

The CAVI algorithm consists in initializing the parameters of $q(w)$ and $q(\gamma)$, e.g., with the values of their priors, and interleaving the update formulas (3.27) and (3.29) until convergence.

### 3.2.2 Assumed Density Filtering

Assumed Density Filtering (ADF) has been independently proposed in the statistics, artificial intelligence, and control domains [17]. Its central idea relies on the model's joint probability $p(\mathbf{x}, \mathbf{z})$ decomposing into a product of independent factors $f_i(\mathbf{z})$ as

$$p(\mathbf{x}, \mathbf{z}) = \prod_{i=1}^{N} f_i(\mathbf{z}), \tag{3.30}$$

$$p(\mathbf{z} \mid \mathbf{X}) = \frac{1}{p(\mathbf{x})} \prod_{i=1}^{N} f_i(\mathbf{z}), \tag{3.31}$$

where the dependency of the factors $f_i$ on $\mathbf{x}$ is made implicit.

The assumption of factorizable distributions is still pretty general. For example, we frequently assume that the observed data is iid given the parameters, which induces factorization over the likelihood term. When considering a graphical model, the distribution can be factored according to its structure, where the factors represent sets of nodes.

Separately approximating each factor and only combining them all at the end to obtain $q^{(N)}(\mathbf{z})$ frequently lead to poor global approximation. Therefore, the ADF sequences through each factor, including one at a time into the current approximation $q^{(i-1)}(\mathbf{z})$, according to

$$q_{tilt}^{(i)}(\mathbf{z}) \propto q^{(i-1)}(\mathbf{z}) f_i(\mathbf{z}). \tag{3.32}$$

However, $q_{tilt}^{(i)}(\mathbf{z})$ gets "slightly" warped and cannot be represented anymore by the initially assumed family of densities $\mathcal{Q}$ from which the prior belong. We thus have to project it back to a distribution in $\mathcal{Q}$. The projection consists in minimizing the KL divergence between the two distributions such that

$$\begin{aligned} q^{(i)}(\mathbf{z}) &= \underset{q \in \mathcal{Q}}{\operatorname{argmin}} \, D_{KL}\left(q_{tilt}^{(i)}(\mathbf{z}) \| q(\mathbf{z})\right) \\ &= \underset{q \in \mathcal{Q}}{\operatorname{argmin}} \, D_{KL}\left(\frac{1}{K_i} q^{(i-1)}(\mathbf{z}) f_i(\mathbf{z}) \| q(\mathbf{z})\right), \end{aligned} \tag{3.33}$$

where $K_i$ is the normalizing constant.

At the $i$th iteration, $q^{(i)}(\mathbf{z})$ is the approximation of the product between the true factors $f_k(\mathbf{z})$, $1 \leqslant k \leqslant i$.

#### 3.2.2.1 Minimizing the Forward KL Divergence

Differently from Sect. 3.2.1, we now employ the *forward* KL divergence $D_{KL}(p \| q)$ for measuring the quality of the approximation. The change in the ordering of the arguments is the reason why ADF (and EP in Sect. 3.2.3) behaves so differently from ADF. KL is a divergence and not a distance, so the symmetry property does not hold and exchanging the arguments leads to a distinct functional with distinct properties.

The reverse KL divergence $D_{KL}(q \| p)$ used in VI severely penalizes the approximating distribution $q$ for placing mass in regions where $p$ has low probability. Rewriting Eq. (3.4) as

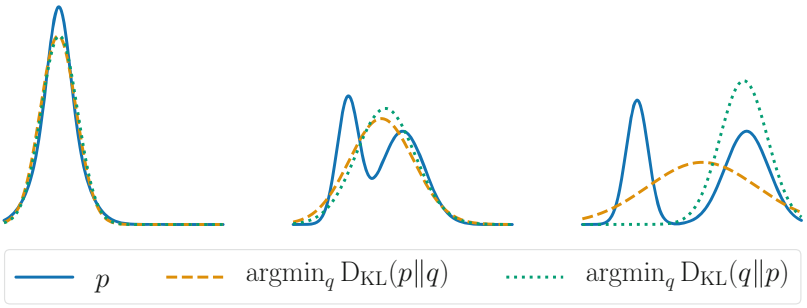



**Fig. 3.9** Comparison of the two alternatives forms of the KL divergence in different scenarios. The blue solid curve is a mixture of two Gaussians, while in the leftmost graph their mean intersects resulting in a single mode, for the two other cases the distribution becomes bi-modal. The green dashed 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

$$D_{KL}(q\|p) = \mathbb{E}_q\left[\log q(x)\right] - \mathbb{E}_q\left[\log p(x)\right], \tag{3.34}$$

we can note that the term $\log p(x)$ rapidly tends to $-\infty$ for such regions. Conversely, by exchanging $p$ and $q$ in Eqs. (3.4) and (3.34) we get

$$D_{KL}(p\|q) = \mathbb{E}_p\left[\log p(x)\right] - \mathbb{E}_p\left[\log q(x)\right]. \tag{3.35}$$

The forward KL has the opposite behavior, that is, it favors 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 $p(x)$ such that $\log q(x)$ tends to $-\infty$. Figure 3.9 neatly illustrates this property for both KL forms.

#### 3.2.2.2 Moment Matching in the Exponential Family

In order to be efficiently calculated, the posterior distribution must be simple to handle. So we further constrain $q_i$ to belong to the exponential family:

$$q_i(\mathbf{z}) = h(\mathbf{z})g(\boldsymbol{\eta})\exp\left(\boldsymbol{\eta}^T\mathbf{u}(\mathbf{z})\right), \tag{3.36}$$

where $\boldsymbol{\eta}^T$ are the natural parameters of the family, $\mathbf{u}(\mathbf{z})$ the sufficient statistics, $g(\boldsymbol{\eta})$ the partition function, and $h(\mathbf{z}) > 0$ the carrier function. See Sect. 2.2 for further details.

Then, the forward KL divergence reduces to

$$
\begin{aligned}
D_{KL}(p\|q) &= \int p(\mathbf{z}) \log p(\mathbf{z}) d\mathbf{z} - \int p(\mathbf{z}) \log q(\mathbf{z}) d\mathbf{z} \\
&= \int p(\mathbf{z}) \log p(\mathbf{z}) d\mathbf{z} - \int p(\mathbf{z}) \log \Big( h(\mathbf{z}) g(\boldsymbol{\eta}) \exp(\boldsymbol{\eta}^T \mathbf{u}(\mathbf{z})) d\mathbf{z} \Big) \\
&= \int p(\mathbf{z}) \log p(\mathbf{z}) d\mathbf{z} - \Big( \mathbb{E}_p[h(\mathbf{z})] + \log g(\boldsymbol{\eta}) + \boldsymbol{\eta}^T \mathbb{E}_p[u(\mathbf{z})] \Big).
\end{aligned} \tag{3.37}
$$

We are interested in finding the natural parameters $\boldsymbol{\eta}$ that specify the distribution that minimizes the KL among the assumed member of the exponential family. Thus, we set

$$
\begin{aligned}
&\nabla_{\boldsymbol{\eta}} D_{KL}(p\|q) = 0 \\
&\implies \nabla_{\boldsymbol{\eta}} \left\{ \int p(\mathbf{z}) \log p(\mathbf{z}) d\mathbf{z} - \Big( \mathbb{E}_p[h(\mathbf{z})] + \log g(\boldsymbol{\eta}) + \boldsymbol{\eta}^T \mathbb{E}_p[\mathbf{u}(\mathbf{z})] \Big) \right\} = 0 \\
&\implies -\nabla_{\boldsymbol{\eta}} \log g(\boldsymbol{\eta}) - \mathbb{E}_p[\mathbf{u}(\mathbf{z})] = 0 \\
&\implies \nabla_{\boldsymbol{\eta}} \log g(\boldsymbol{\eta}) = -\mathbb{E}_p[\mathbf{u}(\mathbf{z})].
\end{aligned} \tag{3.38}
$$

From the fact that any normalized distribution must sum up to 1, we arrive at the following general result for the exponential family:

$$
\begin{aligned}
&\nabla_{\boldsymbol{\eta}} 1 = \nabla_{\boldsymbol{\eta}} \left( \int h(\mathbf{z}) g(\boldsymbol{\eta}) \exp(\boldsymbol{\eta}^T \mathbf{u}(\mathbf{z})) d\mathbf{z} \right) \\
&\implies 0 = \int h(\mathbf{z}) \exp(\boldsymbol{\eta}^T \mathbf{u}(\mathbf{z})) d\mathbf{z} \nabla_{\boldsymbol{\eta}} g(\boldsymbol{\eta}) + \int \mathbf{u}(\mathbf{z}) h(\mathbf{z}) g(\boldsymbol{\eta}) \exp(\boldsymbol{\eta}^T \mathbf{u}(\mathbf{z})) d\mathbf{z} \\
&\implies 0 = \nabla_{\boldsymbol{\eta}} g(\boldsymbol{\eta}) \frac{1}{g(\boldsymbol{\eta})} \int g(\boldsymbol{\eta}) h(\mathbf{z}) \exp(\boldsymbol{\eta}^T \mathbf{u}(\mathbf{z})) d\mathbf{z} + \int \mathbf{u}(\mathbf{z}) q_i(\mathbf{z}) d\mathbf{z} \\
&\implies 0 = \frac{1}{g(\boldsymbol{\eta})} \nabla_{\boldsymbol{\eta}} g(\boldsymbol{\eta}) \int q_i(\mathbf{z}) d\mathbf{z} + \mathbb{E}_q[\mathbf{u}(\mathbf{z})] \\
&\implies 0 = \frac{1}{g(\boldsymbol{\eta})} \nabla_{\boldsymbol{\eta}} g(\boldsymbol{\eta}) + \mathbb{E}_q[\mathbf{u}(\mathbf{z})] \\
&\implies 0 = \nabla_{\boldsymbol{\eta}} \log g(\boldsymbol{\eta}) + \mathbb{E}_q[\mathbf{u}(\mathbf{z})].
\end{aligned} \tag{3.39}
$$

The relation (3.39) means that we can compute moments by taking the derivative w.r.t. $\boldsymbol{\eta}$ of the negative log-partition function.

Substituting Eq. (3.38) in Eq. (3.39), we arrive at

$$
\mathbb{E}_q[\mathbf{u}(\mathbf{z})] = \mathbb{E}_p[\mathbf{u}(\mathbf{z})], \tag{3.40}
$$

which means that when approximating an arbitrary distribution with a member of the exponential family, we should match their expectations over the sufficient statistics $\mathbf{u}(\mathbf{z})$, e.g., the first and second moments, $z$ and $z^2$, for the univariate Gaussian (see Sect. 2.2). Therefore, it all comes down to matching the moments of the new approximation with the moments of the old one tilted by the newly included true factor at each iteration. For computing those moments, Eq. (3.39) is extensively explored.

For example, if we consider a Gaussian posterior approximation $\mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$, we should select $\boldsymbol{\mu}_i$, $\boldsymbol{\Sigma}_i$, and $K_i$ for the distribution $q^{(i)}$ such that

$$\boldsymbol{\mu}_i = \mathbb{E}_{q^{(i-1)} f_i}[\mathbf{z}], \tag{3.41}$$

$$\boldsymbol{\Sigma}_i = \mathrm{Cov}_{q^{(i-1)} f_i}[\mathbf{z}], \tag{3.42}$$

$$\int q^{(i)}(\mathbf{z}) d\mathbf{z} = \frac{1}{K_i} \int q^{(i-1)}(\mathbf{z}) f_i(\mathbf{z}) d\mathbf{z} = 1, \tag{3.43}$$

where $q^{(i-1)} f_i$ is the unnormalized version of the tilted distribution $q^{(i)}_{tilt}$ defined in Eq. (3.32).

#### 3.2.2.3 ADF Issues

Even though the ADF's 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 could mitigate this issue at the expense of losing the online characteristic of the method by revising the initial approximations later on, effectively cycling through all factors.

Similarly to ADF, the variance of the approximating distribution is affected by both the independence assumption needed for the factorization of the distribution and the mass spreading property of the forward KL. However, differently from ADF, the ADF overestimates the marginal variance, giving larger uncertainty estimations and variability than the true posterior would. One should take the variance overestimation property into account when choosing among the different variational methods to solve a given problem.

#### 3.2.2.4 ADF Example

We return to the linear regression problem of Sect. 3.2.1.7, whose model definition was given in Eqs. (3.20)–(3.22) and the posterior distribution provided in Eq. (3.23). For convenience, we rewrite them here:

$$\Gamma \sim \mathrm{Ga}(\gamma; \alpha_0, \beta_0) \tag{3.44}$$

$$W \sim \mathcal{N}(w \mid \mu, \tau^{-1}) \tag{3.45}$$

$$Y_i = Wx_i + \mathcal{N}(0, \gamma^{-1}), 1 \leq i \leq N, \tag{3.46}$$

$$p(\gamma, w \mid \mathbf{Y}, \mathbf{X}) = \frac{p(\gamma)p(w)}{p(\mathbf{y} \mid \mathbf{X})} \prod_{i=1}^{N} p(y_i \mid w, \gamma, x_i). \tag{3.47}$$

Here we use the ADF algorithm to approximate the marginal posterior $p(\gamma \mid \mathbf{Y}, \mathbf{X})$ given by

$$\begin{aligned} p(\gamma \mid \mathbf{Y}, \mathbf{X}) &= \int p(\gamma, w \mid \mathbf{Y}, \mathbf{X}) dw \\ &= \frac{p(\gamma)}{p(\mathbf{y} \mid \mathbf{X})} \prod_{i=1}^{N} \int p(y_i \mid w, \gamma, x_i) p(w) dw \\ &= \frac{p(\gamma)}{p(\mathbf{y} \mid \mathbf{X})} \prod_{i=1}^{N} p(y_i \mid \gamma, x_i), \end{aligned} \tag{3.48}$$

where the likelihood terms $p(y_i \mid \gamma; x_i)$ of the individual observations $Y_i$ are

$$\begin{aligned} p(y_i \mid \gamma; x_i) &= \int p(y_i \mid w, \gamma, x_i) p(w) dw \\ &= \int \mathcal{N}(y_i \mid wx_i, \gamma^{-1}) \mathcal{N}(w; \mu, \tau^{-1}) dw \\ &= \mathcal{N}(y_i; x_i\mu, \tau^{-1}x_i^2 + \gamma^{-1}). \end{aligned} \tag{3.49}$$

We have $N$ likelihood factors to include. We choose $\gamma$ to have a Gamma prior, what constrains the approximate posterior $q(\gamma)$ to follow a Gamma distribution. So, at start the posterior is

$$q(\gamma) = \text{Ga}(\gamma \mid \alpha, \beta), \text{ with } \alpha = \alpha_0, \beta = \beta_0. \tag{3.50}$$

Next, we include the likelihood factors of Eq. (3.49) into $q(\gamma)$. The resulting shifted distribution $s(\gamma)$ after the inclusion of one such factor $p(y_i \mid \gamma; x_i)$ is

$$\begin{aligned} s(\gamma) &\propto \text{Ga}(\gamma \mid \alpha, \beta) \mathcal{N}(y_i; x_i\mu, \tau^{-1}x_i^2 + \gamma^{-1}) \\ &\propto \left[\gamma^{\alpha-1} \exp\{-\beta\gamma\}\right] \left[\left(\tau^{-1}x_i^2 + \gamma^{-1}\right)^{-1/2} \exp\left\{-\frac{1}{2} \frac{(y_i - x_i\mu)^2}{\tau^{-1}x_i^2 + \gamma^{-1}}\right\}\right]. \end{aligned} \tag{3.51}$$

Notice that we cannot write $s(\gamma)$ under the functional form of the Gamma distribution that we established for the approximation $q(\gamma)$. We must project $s(\gamma)$ back to the assumed family. Thus, we compute the update equations responsible for matching the moments. The sufficient statistics for $\gamma$ under the shifted distribution has no closed form, so we only match the first and second moments.

Before proceeding, we compute the normalizing constant $K$, which we need for the moments:

$$\begin{aligned}
K &= \int \text{Ga}(\gamma \mid \alpha, \beta) p(y_i \mid \gamma; x_i) d\gamma \\
&= \int \text{Ga}(\gamma \mid \alpha, \beta) p(y_i \mid z_i, \gamma; x_i) p(z_i \mid w; x_i) dz_i d\gamma \\
&= \int \text{Ga}(\gamma \mid \alpha, \beta) \mathcal{N}(y_i \mid z_i, \gamma^{-1}) \mathcal{N}(z_i \mid x_i \mu, x^2 \tau^{-1}) dz_i d\gamma \\
&= \int \mathcal{T}_{2\alpha}(y_i \mid z_i, \beta/\alpha) \mathcal{N}(z_i \mid x_i \mu, x^2 \tau^{-1}) dz_i \\
&\approx \int \mathcal{N}(y_i \mid x_i \mu, x_i^2 + \beta/(\alpha - 1)) \mathcal{N}(z_i \mid x_i \mu, x^2 \tau^{-1}) dz_i \\
&= \mathcal{N}(y_i \mid x_i \mu, x_i^2 \tau^{-1} + \beta/(\alpha - 1)),
\end{aligned} \tag{3.52}$$

where we have used the fact that the marginalization over the Gamma-distributed prior precision $\gamma$ of the Gaussian-distributed observations $Y_i$ is the student's $t$-distribution $\mathcal{T}$, as shown in Eq. (A.34). We then approximated the distribution $\mathcal{T}$ with a Gaussian with the same mean and variance. Notice that the normalizing constant $K$ depends on $\alpha$ and $\beta$, a fact that we make explicit by writing $K$ as $K_{\alpha,\beta}$.

Labeling the Gaussian term in Eq. (3.51) as $g(\gamma)$, we write the first moment of $\gamma$ under the shifted distribution $s$ as

$$\begin{aligned}
\mathbb{E}_s[\gamma] &= \int \frac{1}{K_{\alpha,\beta}} \gamma \text{Ga}(\gamma \mid \alpha, \beta) g(\gamma) dw d\gamma \\
&= \frac{1}{K_{\alpha,\beta}} \int \gamma \frac{\beta^\alpha}{\Gamma(\alpha)} \gamma^{\alpha-1} e^{-\beta\gamma} g(\gamma) dw d\gamma \\
&= \frac{1}{K_{\alpha,\beta}} \int \frac{\Gamma(\alpha+1)}{\beta\Gamma(\alpha)} \frac{\beta^{\alpha+1}}{\Gamma(\alpha+1)} \gamma^{(\alpha+1)-1} e^{-\beta\gamma} g(\gamma) dw d\gamma \\
&= \frac{1}{K_{\alpha,\beta}} \frac{\alpha}{\beta} \int \text{Ga}(\gamma \mid \alpha+1, \beta) g(\gamma) dw d\gamma \\
&= \frac{K_{\alpha+1,\beta}}{K_{\alpha,\beta}} \frac{\alpha}{\beta}.
\end{aligned} \tag{3.53}$$

The second moment follows a similar procedure

$$\begin{aligned}\mathbb{E}_s\left[\gamma^2\right] &= \int K^{-1}\gamma^2 \text{Ga}(\gamma \mid \alpha, \beta) g(\gamma) dw d\gamma \\ &= \int \frac{\lambda^2}{K_{\alpha,\beta}} \frac{\Gamma(\alpha+2)}{\beta^2\Gamma(\alpha)} \frac{\beta^{\alpha+2}}{\Gamma(\alpha+2)} \gamma^{(\alpha+2)-1} e^{-\beta\gamma} d\gamma \\ &= \frac{1}{K_{\alpha,\beta}} \frac{\alpha(\alpha+1)}{\beta^2} \int \text{Ga}(\gamma \mid \alpha+1, \beta) g(\gamma) dw d\gamma \\ &= \frac{K_{\alpha+2,\beta}}{K_{\alpha,\beta}} \frac{\alpha(\alpha+1)}{\beta^2}. \end{aligned} \tag{3.54}$$

Recalling the mean and variance formulas for the Gamma distribution, we write

$$\begin{aligned}\mathbb{E}_s[\gamma] &= \frac{\alpha_{new}}{\beta_{new}} = \frac{K_{\alpha+1,\beta}}{K_{\alpha,\beta}} \frac{\alpha}{\beta}, \\ Var_s(\gamma) &= \frac{\alpha_{new}}{\beta_{new}^2} = \frac{K_{\alpha+2,\beta}}{K_{\alpha,\beta}} \frac{\alpha(\alpha+1)}{\beta^2}. \end{aligned} \tag{3.55}$$

Solving the system of equations for $\alpha_{new}$ and $\beta_{new}$, we get

$$\alpha_{new} = \left[\frac{K_{\alpha,\beta} K_{\alpha+2,\beta}}{K_{\alpha+1,\beta}} \frac{\alpha+1}{\alpha} - 1\right]^{-1}, \tag{3.56}$$

$$\beta_{new} = \left[\frac{K_{\alpha+2,\beta}}{K_{\alpha+1,\beta}} \frac{\alpha+1}{\beta} - \frac{K_{\alpha+1,\beta}}{K_{\alpha,\beta}} \frac{\alpha}{\beta}\right]^{-1}. \tag{3.57}$$

In summary, the algorithm starts from the prior in Eq. (3.50), then it applies Eqs. (3.56) and (3.57) once for each likelihood factor, where the partition functions for $K$ follow Eq. (3.52).

An established example is given in [17], where the author demonstrates the use of ADF for recovering data from a sea of clutter, projecting a Gaussian mixture posterior onto a single Gaussian distribution.

### 3.2.3 *Expectation Propagation*

As mentioned in the previous section, one of the ADF's weaknesses is its sensitivity to the order in which factors are considered. In a batch setting, where all factors are available, it is unreasonable to see each only once and not refine the approximation repeatedly. However, directly cycling through a factor $n$ times would lead to including such factor into the approximating distribution $n$ times instead of one. This would artificially accumulate evidence, making the likelihood concentrate around a

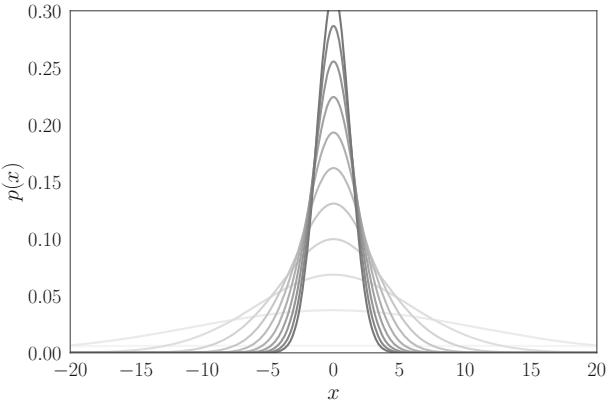



**Fig. 3.10** Continuous inclusion of the same original factor, in lightest shade, causes the distribution to concentrate around its mode, progressively collapsing to that single point and eventually becoming a Dirac distribution

single point, until collapsing the posterior into that point, as shown in Fig. 3.10, which is highly undesired.

#### 3.2.3.1 Recasting ADF as a Product of Approximate Factors

The EP reinterprets the ADF as approximating each new true factor $f_i$ with $\widetilde{f_i}$ such that

$$q^{(i)}(\mathbf{z}) \propto q^{(i-1)}(\mathbf{z})\widetilde{f_i}. \tag{3.58}$$

The approximate factor $\widetilde{f_i}$ can be easily obtained at the end of the $i$th ADF iteration by

$$\widetilde{f_i}(\mathbf{z}) \propto \frac{q^{(i)}(\mathbf{z})}{q^{(i-1)}(\mathbf{z})}. \tag{3.59}$$

This shift in view means that $q$ can be seen as a product of the approximate factors $\widetilde{f_i}$, such that

$$q(\mathbf{z}) \propto \frac{q^{(N)}(\mathbf{z})}{q^{(N-1)}(\mathbf{z})} \cdots \frac{q^{(1)}(\mathbf{z})}{q^{(0)}(\mathbf{z})} = \prod_{i=1}^{N} \widetilde{f_i}(\mathbf{z}), \tag{3.60}$$

where $q^{(0)}(\mathbf{z}) = p_0(\mathbf{z})$ is the prior distribution.

In ADF, initial factors have little context: few to none other factors have been seen; so they are prone to poor approximation. On the other hand, later factors have

large context and potential to be better approximated. The EP handles this issue by observing the entire context when approximating $f_i$ with $\widetilde{f}_i$. Since it keeps track of each $f_i$ and the corresponding $\widetilde{f}_i$ at every iteration, it is possible to compute

$$q_{new}(\mathbf{z}) = \underset{q \in Q}{\operatorname{argmin}} D_{KL}\left(\frac{1}{K_i} f_i(\mathbf{z}) \frac{q(\mathbf{z})}{\widetilde{f}_i(\mathbf{z})} \| q(\mathbf{z})\right), \tag{3.61}$$

where $K_i$ is the normalizing constant. Note that now, at any given iteration $j$, $q$ no longer is the product of factors $1 < k < j$, but of all $N$ factors. That is why we have dropped the superscript in $q$.

Since we always remove $\widetilde{f}_i$ prior to including $f_i$, we will not repeatedly accumulate the $f_i$'s contribution if we repeat this step multiple times. After computing $q_{new}$ according to Eq. (3.61), we revise $\widetilde{f}_i$ in a similar fashion to Eq. (3.59) so that the factor $\widetilde{f}_i$ is responsible for the change from $q$ to $q_{new}$. However, because $q$ is the product of all factors in EP, the update in $\widetilde{f}_i$ follows

$$\widetilde{f}_i(\mathbf{z}) = K_i \frac{q_{new}(\mathbf{z})}{q_{-i}(\mathbf{z})}, \tag{3.62}$$

where $q_{-i}(\mathbf{z})$ is the unnormalized cavity distribution, computed by removing the factor $f_i$ from $q$, like

$$q_{-i}(\mathbf{z}) = q_i(\mathbf{z}) / \widetilde{f}_i(\mathbf{z}). \tag{3.63}$$

Note that, from the definitions above, Eq. (3.62) leads to

$$\frac{q_{new}(\mathbf{z})}{q_{-i}(\mathbf{z})} \propto \frac{\prod_j \widetilde{f}_j(\mathbf{z})}{\prod_{j \neq i} \widetilde{f}_j(\mathbf{z})} = \widetilde{f}_i(\mathbf{z}). \tag{3.64}$$

Broadly speaking, each iteration consists of refining the approximation of $\widetilde{f}_i$ by substituting its contribution by that of true factor $f_i$ and finding the $q^{new}$ that minimizes the KL divergence. Just as seen in Sect. 3.2.2.2 for ADF, KL minimization is done by matching the moments of the new distribution $q_{new}(\mathbf{z})$ with those of the tilted distribution $q_{tilt}(\mathbf{z}) = K_i^{-1} f_i(\mathbf{z}) q_{-i}(\mathbf{z})$. Even though the EP approximates one factor at a time and the resulting $q$ is a valid probability distribution, $\widetilde{f}_i$ alone and the partial products not necessarily represent a valid distribution.

The EP algorithm is summarized in Algorithm 1. Figure 3.11 succinctly shows the difference between EP and ADF for a single iteration: while the EP takes in all approximate factors $\widetilde{f}_j$ except for $j = i$, that is going to be update, the ADF takes in only previously seen factors $\widetilde{f}_j$, which are input through $q^{(i-1)} \propto \prod_{j=1}^{i-1} \widetilde{f}_j$.


**Fig. 3.11** Diagram of ADF and EP updates for a single iteration. The ADF limits itself by looking at the previously included factors, which got approximated from $f_j$ to $\widetilde{f}_j$ in the projection step of $q^j$. The EP considers all factors simultaneously, except for the one to be updated, what avoids factor multiplicity in the approximation $q$

**Algorithm 1:** EP

1: initializing $\widetilde{f}_i = 1, \forall i$ by setting the parameters accordingly
2: **while** not converged **do**
3: choosing a factor $\widetilde{f}_i$ to update
4: computing the unnormalized cavity distribution defined in (3.63)
5: evaluating the normalizing constant $K_i$ in (3.61)
6: performing the projection of (3.61)
7: updating the factor $\widetilde{f}_i$ by (3.62)
8: **end while**

#### 3.2.3.2 Operations in the Exponential Family

Constraining the factors to the functional form of the exponential family renders inclusion and exclusion of factors simple and computationally efficient. It suffices to add and subtract the natural parameters $\boldsymbol{\eta}$, like

$$q_i(\mathbf{z})/\widetilde{f}_i(\mathbf{z}) = \frac{h(\mathbf{z})g(\boldsymbol{\eta})\exp\left(\boldsymbol{\eta}^T\mathbf{u}(\mathbf{z})\right)}{h(\mathbf{z})g(\boldsymbol{\eta})\exp\left(\boldsymbol{\eta}'^T\mathbf{u}(\mathbf{z})\right)} = \exp\left((\boldsymbol{\eta}' - \boldsymbol{\eta})^T\mathbf{u}(\mathbf{z})\right). \tag{3.65}$$

#### 3.2.3.3 Power EP

Not every distribution can be factored into simple terms. Hence, integrating such factors to compute the normalizing terms is not a simple task. Consequently, EP fails to be computationally efficient. Power EP [18] addresses this shortcoming by cleverly raising the factors $f_i$ to a power of $1/n_i, n_i \in \mathbb{R}$, canceling out complicated exponents present in the true factors, and making them easier to compute.

The algorithm is essentially the same, except that we perform it on "fractional factors," that is,

$$f_i'(\mathbf{z}) = f_i(\mathbf{z})^{1/n_i}, \tag{3.66}$$

$$\widetilde{f}'_i(\mathbf{z}) = \widetilde{f}_i(\mathbf{z})^{1/n_i}. \tag{3.67}$$

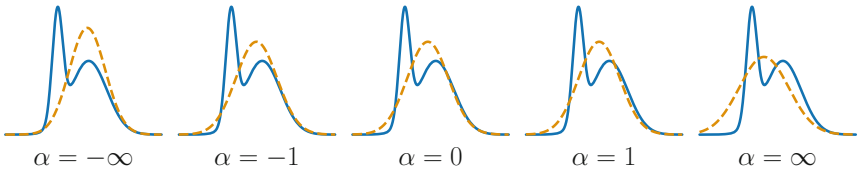



**Fig. 3.12** The $\alpha$-divergence family. For $\alpha \to -1$, it becomes the reverse KL, $D_{KL}(q\|p)$, while for $\alpha \to 1$ it is the forward KL, $D_{KL}(p\|q)$

When $n_i \geqslant 1 \in \mathbb{N}$, we can think of Power EP as an EP that splits the factor $f_i$ into $n_i$ distinct copies. However, instead of performing one EP iteration for each $f_i$, following Eqs. (3.61) and (3.62), Power EP computes the update for a single copy and assumes the result to be the same for the other $n_i - 1$ copies.

In the EP, replacing the minimized objective $D_{KL}(p\|q)$ by

$$D_\alpha(p\|q) = \frac{4}{1-\alpha^2}\left(1 - \int p(x)^{(1+\alpha)/2} q(x)^{(1-\alpha)/2} dx\right), \tag{3.68}$$

with a continuous parameter $\alpha$, results in an algorithm with the same fixed points as the Power EP. Therefore, we can think of Power EP as minimizing the $\alpha$-divergence $D_\alpha$, with $\alpha$ 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 $\alpha$-family defined by Eq. (3.68). Specifically, $\alpha \to 1$ gives the forward KL and $\alpha \to -1$ the reverse KL, which can be verified by remembering that $p(x)^\gamma = \exp\{\gamma \log p(x)\}$ and using L'Hôpital rule for evaluating indeterminate limits. As we can see in Fig. 3.12, values $\alpha \leqslant -1$ induce a zero-forcing behavior, setting $q(x) = 0$ for any values of $x$ for which $p(x) = 0$. Conversely, $\alpha \geqslant 1$ is zero avoiding, imposing $q(x) \geqslant 0$ for regions where $p(x) \geqslant 0$, and typically $q$ stretches to cover all $p$.

One way to understand many message-passing algorithms, including those we discussed, is as the same variational framework with different energy functions corresponding to distinct values of $\alpha$ in Eq. (3.68) [19].

#### 3.2.3.4 EP Issues

Naturally, the enhancement provisioned by EP has costs. Besides being unsuitable to online learning, it has to keep all true and approximating factors stored in memory. Therefore, memory consumption grows linearly with the number of factors of the distribution. This may be inadequate if data sets are too large, because it would be impractical or even impossible to maintain all factors in memory during optimization.

While each step in VI is guaranteed 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 [3, p. 510]. Nonetheless, stable EP fixed points are local minima of the optimization problem [17].

In multi-modal target distributions, the EP can lead to poor approximations because the forward KL divergence causes $q$ to average over all modes [3, p. 510].

#### 3.2.3.5 EP Example

Consider again the linear regression problem of Sects. 3.2.2.4 and 3.2.1.7. The main difference from Sect. 3.2.2.4 is that now we need to track the approximate factors $\widetilde{f}$.

We initialize the approximate posterior with parameters $\alpha = 1$ and $\beta = 0$ so that we have a uniform distribution. The inclusion of the prior factor $p(\gamma)$ shifts the distribution into

$$s(\gamma) \propto \text{Ga}(\gamma \mid \alpha, \beta)\text{Ga}(\gamma \mid \alpha_0, \beta_0), \tag{3.69}$$

$$s(\gamma) = \text{Ga}(\gamma \mid \alpha + \alpha_0 - 1, \beta + \beta_0). \tag{3.70}$$

We see that $s(\gamma)$ is a member of the assumed family and there is no approximation in this step. Since the inclusion of the prior precision $p(\gamma)$ does not throw the approximate posterior $q(\gamma)$ out of the assumed family, there is no need to process such factor multiple times. The update equations are

$$\alpha_{new} = \alpha + \alpha_0 - 1 \qquad \beta_{new} = \beta + \beta_0. \tag{3.71}$$

On the other hand, the inclusion of the likelihood factors $p(y_i \mid \gamma; x_i)$ is not exact as:

1. we approximate a student's $t$-distribution by a Gaussian in deriving Eq. (3.52);
2. we match only the first two moments of the shifted distribution in Eq. (3.51).

Consequently, there is room for improvement and we cycle through the likelihood factors. We conveniently choose the approximate factors to be

$$\widetilde{f}_i(\gamma) = \text{Ga}(\gamma \mid a, b). \tag{3.72}$$

This form allows us to easily compute the cavity distribution

$$q_{-i}(\gamma) \propto \frac{q(\gamma)}{\widetilde{f}_i(\gamma)} = \frac{\text{Ga}(\gamma \mid \alpha, \beta)}{\text{Ga}(\gamma \mid a, b)} = \text{Ga}\left(\gamma \mid \alpha_{-i}, \beta_{-i}\right), \tag{3.73}$$

where

$$\alpha_{-i} = \alpha - a + 1 \qquad \beta_{-i} = \beta - b. \tag{3.74}$$

After computing the cavity distribution $q_{-i}$, we include the true likelihood factor $p(y_i \mid \gamma; x_i)$ and project the resulting distribution back onto the assumed family of $q$. The steps for including and projecting the likelihood factors are still the same as those of Sect. 3.2.2.4 for the ADF algorithm: Eqs. (3.56) and (3.57) for updating $\alpha$ and $\beta$, respectively.

Lastly, we revise the approximate factor $\widetilde{f}_i$, according to

$$a = \alpha - \alpha_{-i} + 1 \qquad b = \beta - \beta_{-i}. \tag{3.75}$$

## 3.2.4 *Further Practical Extensions*

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

### 3.2.4.1 Black Box Variational Inference

As seen in Sect. 3.2.1.5, the SVI computes the distribution updates in a closed form, which requires model-specific knowledge and implementation. Moreover, the gradient of the ELBO must have a closed-form analytical formula. Black Box Variational Inference (BBVI) [25] avoids these problems by estimating the gradient instead of actually computing it.

BBVI uses the score function estimator [34]

$$\nabla_\phi \mathbb{E}_{q(\mathbf{z};\phi)}\left[f(\mathbf{z};\theta)\right] = \mathbb{E}_{q(\mathbf{z};\phi)}\left[f(\mathbf{z};\theta)\nabla_\phi \log q(\mathbf{z};\phi)\right], \tag{3.76}$$

where the approximating distribution $q(\mathbf{z};\phi)$ is a continuous function of $\phi$ (see Appendix A.1). Using this estimator to compute the gradient of the ELBO in Eq. (3.7) gives us

$$\nabla_\phi \text{ELBO} = \mathbb{E}_q\left[(\nabla_\phi \log q(\mathbf{z};\phi))(\log p(\mathbf{x},\mathbf{z}) - \log q(\mathbf{z};\phi))\right]. \tag{3.77}$$

The expectation in Eq. (3.77) is approximated by a Monte Carlo integration.

The sole assumption of the gradient estimator in Eq. (3.77) about the model is the feasibility of computing the log of the joint $p(\mathbf{x},\mathbf{z}_s)$. 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 $p(\mathbf{x},\mathbf{z}_s)$. Hence the name black box: we just need to specify the model $p(\mathbf{x},\mathbf{z}_s)$ and can directly perform VI on it. Actually, $p(\mathbf{x},\mathbf{z}_s)$ does not even need to be normalized, since the log of the normalization constant does not contribute to the gradient in Eq. (3.77).

We generally perform stochastic optimization, observing a subset of the available data at each iteration. The score function estimator gives unbiased estimates when considering $f(\mathbf{z};\theta)$ in Eq. (3.76). However, gradient estimates in Eq. (3.77) are not unbiased due to the presence of the log function. Furthermore, the estimator generally has high variance, what may force the step sizes to be too small for the algorithm to be practical. The authors in [25] further consider variance reduction methods that preserve the black box character of BBVI to address this issue.

#### 3.2.4.2 Black Box $\alpha$ Minimization

Black Box $\alpha$ minimization [9] (BB-$\alpha$) optimizes an approximation of the power EP energy function [19, 20]. Instead of considering $i$ different local compatibility functions $\widetilde{f}_i$, it ties them together so that all $\widetilde{f}_i$ are equal, that is, $\widetilde{f}_i = \widetilde{f}$. We may view it as an average factor approximation, which we use to approximate the average effect of the original $f_i$ [9].

Further restricting these factors to belong to the exponential family amounts to tying their natural parameters. As a consequence, BB-$\alpha$ no longer needs to store an approximating site per likelihood factor, which leads to significant memory savings in large data sets. The fixed points differ from power EP, though they become equal in the limit of infinite data.

BB-$\alpha$ dispenses with the need for double-loop algorithms to directly minimize the energy and employs gradient-descent methods for this matter. This contrasts with the iterative update scheme of Sect. 3.2.3. As other modern methods designed for large-scale learning, it employs stochastic optimization 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 [25], the BB-$\alpha$ uses the pathwise derivative estimator [24] to estimate the gradient (see Appendix A.1). We must be able to express the random variable $\mathbf{z} \sim q(\mathbf{z},\phi)$ as an invertible deterministic transformation $g(\cdot;\phi)$ of a base random variable $\epsilon \sim p(\epsilon)$, so we can write

$$\nabla_\phi \mathbb{E}_{q(\mathbf{z};\phi)}\left[f(\mathbf{z};\theta)\right] = \mathbb{E}_{p(\epsilon)}\left[\nabla_\phi f(g(\epsilon;\phi);\theta)\right]. \tag{3.78}$$

The approach requires not only the distribution $q(\mathbf{z};\phi)$ to be reparameterizable but also $f(\mathbf{z};\theta)$ to be known and a continuous function of $\phi$ for all values of $\mathbf{z}$. Note that it requires, in addition to the likelihood function, its gradients. Still, we can readily obtain them with automatic differentiation tools if the likelihood is analytically defined and differentiable.

As observed in Sect. 3.2.3, the parameter $\alpha$ in Eq. (3.68) controls the divergence function. Hence, the method is able to interpolate between VI ($\alpha \to -1$) and an algorithm similar to EP ($\alpha \to 1$). Interestingly, the authors [9] claim to usually obtain the best results by setting $\alpha = 0$, halfway through VI and EP. This value corresponds to the so-called Hellinger distance, the sole member of the $\alpha$-family that is symmetric.

#### 3.2.4.3 Automatic Differentiation Variational Inference

Automatic Differentiation Variational Inference (ADVI) offers a recipe for automating the computations involved in VI [13]. The user 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.

The ADVI applies a transformation $T : \mathcal{Z} \mapsto \Xi$ that maps the support of the latent variables $\mathbf{z}$ to all real coordinate space, such that the model's joint distribution $p(\mathbf{x}, \mathbf{z})$ becomes $p(\mathbf{x}, \boldsymbol{\xi})$. Then, it approximates $p(\mathbf{x}, \boldsymbol{\xi})$ 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 defined in Eq. (3.7) involves an intractable expectation. The ADVI resorts to the pathwise gradient estimator in Eq. (3.78) to convert the variational distribution into 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 optimization [27].

As the ADVI employs the pathwise gradient estimator, it works only for differentiable models. The derivative of the log joint probability $\nabla_z \log p(x, z)$ must exist. On the other hand, BBVI [25] 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 its manually implemented counterpart, the ADVI works well for a large class of practical models on modern data sets [13]. Therefore, it allows rapid prototyping of new ideas and corrections of complex models.

## 3.3 Closing Remarks

In this chapter, we introduced the concept of MBML and its three pillars: Bayesian inference, graphical models, and probabilistic programming, explaining how each piece connects to construct the MBML landscape and clarifying the need for approximate inference in complex problems.

We have explained the inner workings of and exemplified three central variational inference techniques, namely ADF, ADF, and EP, drawing attention to the advantages and shortcomings of each. In summary,

- **VI**: maximizes a lower bound on the model evidence (ELBO), tends to fit a single mode of the true posterior distribution, underestimates variance, and is guaranteed to converge.
- **EP**: matches moments, requires definition of the approximate posterior family, tends to summarize the entire true posterior distribution, overestimates variance, and is not guaranteed to converge.
- **ADF**: online version of EP with no iterative refinement.

The ADF, ADF, and EP techniques are the bases for many modern methods, such as the ones in Sect. 3.2.4, and are extensively used in many algorithms, which we discuss in Chap. 4, what illustrates the relevance of the topic.

## References


1. Berger T (1975) Rate distortion theory and data compression. Springer Vienna, pp 1–39. https://doi.org/10.1007/978-3-7091-2928-9_1
2. Bingham E, Chen JP, Jankowiak M, Obermeyer F, Pradhan N, Karaletsos T, Singh R, Szerlip P, Horsfall P, Goodman ND (2019) Pyro: deep universal probabilistic programming. J Mach Learn Res 20(28):1–6
3. Bishop CM (2006) Pattern recognition and machine learning. Springer, Berlin
4. Bishop CM (2013) Model-based machine learning. Philos Trans R Soc A: Math Phys Eng Sci 371(1984):1–17
5. Blei DM, Kucukelbir A, McAuliffe JD (2017) Variational inference: a review for statisticians. J Am Stat Assoc 112(518):859–877
6. Carpenter B, Gelman A, Hoffman M, Lee D, Goodrich B, Betancourt M, Brubaker M, Guo J, Li P, Riddell A (2017) Stan: A probabilistic programming language. J Stat Softw 76(1):1–32. Articles
7. Ghahramani Z (2015) Probabilistic machine learning and artificial intelligence. Nature 521(7553):452–459
8. Goodman ND, Stuhlmüller A (2014) (electronic). The Design and Implementation of Probabilistic Programming Languages. Retrieved 2021-4-5 from http://dippl.org
9. Hernandez-Lobato J, Li Y, Rowland M, Bui T, Hernandez-Lobato D, Turner R (2016) Black-box alpha divergence minimization. In: Proceedings of the international conference on machine learning, New York, vol 48, pp 1511–1520
10. Hoffman M, Blei D (2015) Stochastic structured variational inference. In: International conference on artificial intelligence and statistics, San Diego, vol 38, pp 361–369
11. Hoffman MD, Blei DM, Wang C, Paisley J (2013) Stochastic variational inference. J Mach Learn Res 14:1303–1347
12. Koller D, Friedman N, Bach F (2009) Probabilistic graphical models: principles and techniques. MIT Press, Cambridge
13. Kucukelbir A, Blei D, Gelman A, Ranganath R, Tran D (2017) Automatic differentiation variational inference. J Mach Learn Res 18:1–45
14. Lloyd S (1982) Least squares quantization in PCM. IEEE Trans Inf Theor 28(2):129–137
15. Louizos C, Welling M (2016) Structured and efficient variational deep learning with matrix Gaussian posteriors. In: Proceedings of the international conference on machine learning, New York, vol 48, pp 1708–1716
16. MacKay DJ (2003) Information theory, inference and learning algorithms. Cambridge University Press, Cambridge
17. Minka TP (2001) Expectation propagation for approximate Bayesian inference. In: Conference in uncertainty in artificial intelligence, San Francisco, pp 362–369
18. Minka T (2004) Power EP. Tech. rep., Microsoft Research
19. Minka T (2005) Divergence measures and message passing. Tech. rep., Microsoft Research
20. Minka T (2007) The EP energy function and minimization schemes. Tech. rep., Microsoft Research
21. Minka T, Winn J, Guiver J, Zaykov Y, Fabian D, Bronskill J (2018) Inf. NET 0.3. Microsoft Research Cambridge
22. Mohamed S (2018) Planting the seeds of probabilistic thinking: foundations, tricks and algorithms. Tutorial presentation

23. Murphy KP (2012) Machine learning: a probabilistic perspective. MIT Press, Cambridge
24. Price R (1958) A useful theorem for nonlinear devices having Gaussian inputs. Trans Inf Theor 4(2):69–72
25. Ranganath R, Gerrish S, Blei D (2014) Black box variational inference. In: Proceedings of the international conference on artificial intelligence and statistics, Reykjavik, pp 814–822
26. Rezende D, Mohamed S (2015) Variational inference with normalizing flows. In: Proceedings of the international conference on machine learning, Lille, vol 37, pp 1530–1538
27. Robbins H, Monro S (1951) A stochastic approximation method. Ann Math Stat 22(3):400–407
28. Salvatier J, Wiecki TV, Fonnesbeck C (2016) Probabilistic programming in Python using PyMC3. PeerJ Comput Sci 2:e55
29. Sun S, Chen C, Carin L (2017) Learning structured weight uncertainty in Bayesian neural networks. In: International conference on artificial intelligence and statistics, Fort Lauderdale, vol 54, pp 1283–1292
30. Tran D, Kucukelbir A, Dieng AB, Rudolph M, Liang D, Blei DM (2016) Edward: a library for probabilistic modeling, inference, and criticism. arXiv e-prints 1610.09787
31. Tukey JW (1962) The future of data analysis. Ann Math Stat 33(1):1–67
32. van de Meent JW, Paige B, Yang H, Wood F (2018) An introduction to probabilistic programming. arXiv e-prints 1809.10756
33. Vidakovic B (2011) Bayesian inference using Gibbs sampling—BUGS project. Springer, New York, pp 733–745
34. Williams RJ (1992) Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach Learn 8(3):229–256

# Chapter 4
# Bayesian Neural Networks


This chapter presents the ideas, derivations, advantages, and issues of four different algorithms for Bayesian Neural Network (BNN):

- Bayes by Backprop (BBB) [6];
- Probabilistic Backpropagation (PBP) [19];
- Monte Carlo Dropout (MCDO) [13];
- Variational Adam (Vadam) [26].

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 distribution.

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 its needs;
- Know where to search deeper if in need of structured BNNs.

## 4.1 Why BNNs?

Recently, 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 $\mathbf{w}^*$, they infer the posterior weight distribution $p(\mathbf{w} \mid \mathbf{D})$ given the data set $\mathcal{D}$. Thus, $\mathbf{w}^*$, the maximizer of the distribution, is only a single point in the entire support, as illustrated in Fig. 4.1.



L. P. Cinelli et al., *Variational Methods for Machine Learning with Applications to Deep Networks*, https://doi.org/10.1007/978-3-030-70679-1_4

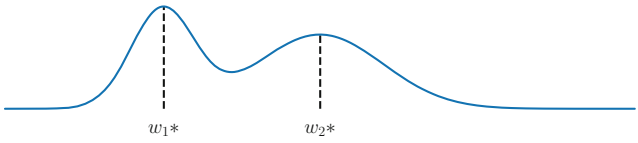



**Fig. 4.1** Maximizer $w_1^*$ of the posterior distribution and runner-up $w_2^*$ corresponding to the maximum of another mode

First introduced in [55], BNNs saw a great advance during the following years (the 1990s) [20, 35, 39, 40]. However, due to their 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 [21] and effectively gaining attention in 2012 after a Deep Learning (DL) method [30] won an important image classification competition [48] 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 [16, 57] allied to the concerns raised by adversarial attacks [43] and the cry for uncertainty measures quintessential for some practical applications sparked interest in Bayesian methods for DL.

The main reason for the late acceptance of BNNs (which is still to come) is that their computational complexity impedes scalability. Modern models and data sets have 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 optimization 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 still are 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 DL.

Even though backpropagation and maximum likelihood optimization allow fitting large non-linear 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, such as $\ell_1$ or $\ell_2$ penalty, is equivalent to maximum a posteriori optimization (with Laplace and Gaussian priors, respectively). However, in spite of alleviating overfitting, it is far from solving the problem. What is more, it makes the solution dependent on the parameterization, that is, different parameterizations may lead to different optimal points. Then, the question arises of which parameterization leads to the best possible solution and how sensitive it is.

Even when resorting to invariant methods, we still have no measure of confidence. Although bootstrapping alleviates the issue, it does not solve the underlying problem: it approximates the probability distribution of the observed data, considering the unknown variables to be fixed. The Bayesian framework solves all this at once by allowing models to represent not only single point estimates but complete distributions over all possible parameter values. It offers a unified framework for

model building, inference, prediction, and decision-making. Moreover, it provides a straightforward way to score models and select among them. BNNs have built-in regularization, offer the advantages of ensemble learning, allow uncertainty estimation and continual learning, besides weight quantization and compression.

There is no free lunch, and as already hinted above, BNNs have burdensome inference. They rely on conditioning and marginalization, so the main operation is integration. Thus, high-dimensional and/or complex models impose a real barrier to their deployment. We discuss approaches that mitigate this issue by employing distributional approximations (Sect. 3.2) 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, Sect. 3.2.1).

For ease of notation, we shall use $\mathbf{w}$ as the random variable instead of $\mathbf{z}$. 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.

## 4.2 Assessing Uncertainty Quality

Bayesian and, more generally, probabilistic models output some measure of uncertainty on which we rely to make decisions. Can we really believe in these models? Do they reflect, approximately at least, the truth? We next present common approaches to address these questions.

### *4.2.1 Predictive Log-Likelihood*

As explained in Sect. 2.4, the likelihood term $p(\mathbf{d} \mid \mathbf{W})$ measures how likely a specific configuration of the model is of generating the observed data. The predictive log-likelihood captures how well the model fits the data, taking the variance (or other measure of spread) of the prediction into account. It is an estimate of how well the model fits both the mean and uncertainty.

Intuitively, the lower the variance, the more reliable the prediction should be and, hence, the lower the score for being wrong. Still, the predictions ought to be reliable so large variances also receive lower scores.

Let us take as example a regression model $f(\cdot; w)$, parameterized 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 true output is corrupted by a known process. For an additive Gaussian noise with variance $\sigma^2$, the log-likelihood estimate has the form

$$\begin{aligned} \log p(y \mid x, w) &= \log \mathcal{N}(y; f(x; w), \sigma^2) \\ &= -\frac{1}{2} \log \left(2\pi\sigma^2\right) - \frac{1}{2\sigma^2} \left(y - f(x)\right)^2 . \end{aligned} \tag{4.1}$$

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.

### 4.2.2 Calibration

Although posterior or predictive credible intervals are not necessarily calibrated, it is a natural measure of reliability. In a classification task, one would expect being correct $X\%$ of the time when the model assigns an $X\%$ probability of being correct. In a regression setting, one hopes that $X\%$ of the time the true value falls within an $X\%$ credible interval. A model with such coverage property is said to be well-calibrated and implies that the Bayesian credible intervals coincide with the frequentist confidence intervals.

The approach that asserts that inferences under a particular model should be Bayesian, but model assessment can and should involve frequentist ideas is called Calibrated Bayes [33].

A common diagnostic tool for calibration is the reliability (or calibration) plot, shown in Fig. 4.2. 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 expected error between the predicted and empirical frequencies for $m$ different confidence intervals.

Still analyzing Fig. 4.2, one can notice that there are two other curves besides the identity. The one in red, with triangle markers, refers to the uncalibrated model, as the blue one, with square markers, is provided by the calibrated method applied after the model has been trained. This is a toy example so the reader can realize how significant the calibration process can be, clearly moving the uncalibrated curve towards the identity. However, the performance of this method varies according to the model you are calibrating, as pointed by the authors in [42]. In that paper, the authors also present and analyze the behavior of two well-known learning techniques that perform calibration: Platt Scaling [45] and Isotonic Regression.

Calibration is not enough for a good overall model, forecasts also need to be sharp [31]. 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.

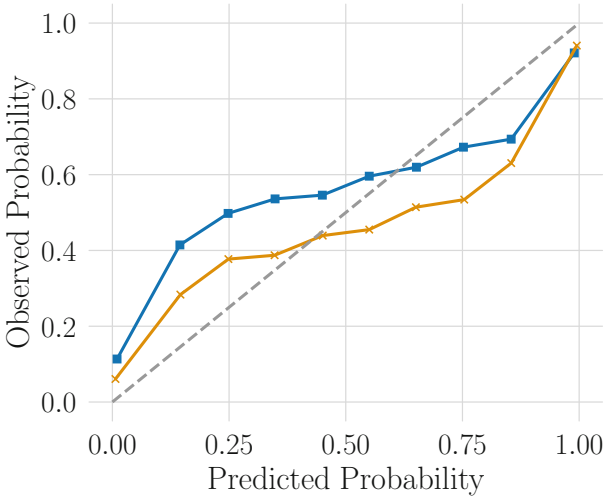



**Fig. 4.2** Example of calibration plot. The gray dashed line is the identity $y = x$, the red curve is the uncalibrated model, and the blue curve refers to the post-calibration model, which were calibrated using the Platt's method [45]. Ideally, we want the blue and gray line to be superposed, indicating a perfectly well-calibrated model

### *4.2.3 Downstream Applications*

It is worthwhile noting that even though the two previous metrics, predictive log-likelihood and expected calibration error, are standard measures for assessing uncertainty quality, it is still important to consider the context in which the uncertainty measures are applied. One should also evaluate uncertainty quality by measuring the performance of the downstream application of interest, e.g., outlier detection, active learning, or uncertainty-driven exploration, with the appropriate relevant metrics.

## 4.3 Bayes by Backprop

BBB has a quite long history preceding it. Bayes by Backprop [6], or BBB for short, continues the work of [16] on practical VI for NNs, who in turn extends on [20], the first to propose VI for NNs.

The essence of BBB's approach is choosing a variational posterior $q$ from which probable samples can be drawn efficiently so that it becomes amenable to Monte Carlo (MC) integration.

Specifying a diagonal Gaussian posterior implies that all network weights $w_i$ are independent, requiring separate means $\mu_i$ and variances $\sigma_i^2$. Consequently, each

weight $w_i$ is characterized by $\Psi_i = \{\mu_i, \sigma_i^2\}$ and the set of all parameters by $\Psi = \{\boldsymbol{\mu}, \boldsymbol{\sigma}^2\}$. We express the approximating variational posterior distribution by

$$q(\mathbf{w}; \Psi) = \prod_i q(w_i; \Psi_i) = \prod_i \mathcal{N}(w_i; \mu_i, \sigma_i^2). \tag{4.2}$$

As seen in Sect. 3.2.1, optimizing the variational approximation amounts to minimizing the negative ELBO as in (3.8), which writes

$$\begin{aligned} \mathcal{L}(q) &= -\text{ELBO}(q) \\ &= -\mathbb{E}_{q(\mathbf{w};\Psi)} \left[\log p(\mathcal{D} \,|\, \mathbf{W})\right] + D_{KL}\left(q(\mathbf{w}; \Psi) \| p(\mathbf{w})\right) \\ &= \mathcal{L}_{data} + \mathcal{L}_{prior}, \end{aligned} \tag{4.3}$$

where we 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 regularizer pushing towards the prior $p(\mathbf{w})$, as already explained in Sect. 3.2.1.

The diagonal Gaussian posterior (4.2) results in a non-closed analytical form for the expectation in $\mathcal{L}_{data}$ and for its derivatives w.r.t. $\mu_i$ and $\sigma_i$, rendering direct evaluation and backpropagation unfeasible. To get around this issue one may resort to MC integration, i.e., drawing different weights $\mathbf{w}_t$ from the posterior $q(\mathbf{w}; \Psi)$, performing the desired computation for each sample and averaging the results. The main contribution from [6] is a reparameterization that gives unbiased gradient estimators and is actually not restricted to Gaussian distributions. It relies on the reparameterization trick (Sect. A.1) for a variational posterior $q(\mathbf{w}; \Psi)$ and a cost function $h(\mathbf{w}; \Psi)$, both dependent on the parameters $\Psi$, according to

$$\nabla_\Psi \mathbb{E}_{q(\mathbf{w};\Psi)} \left[h(\mathbf{w}; \Psi)\right] = \mathbb{E}_{p(\boldsymbol{\epsilon})} \left[\frac{\partial h(\mathbf{w}; \Psi)}{\partial \mathbf{w}} \frac{\partial \mathbf{w}}{\partial \Psi} + \frac{\partial h(\mathbf{w}; \Psi)}{\partial \Psi}\right], \tag{4.4}$$

where, as before, $\mathbf{w} = g(\boldsymbol{\epsilon}; \Psi)$, with $g(\cdot; \Psi)$ a smooth invertible deterministic transformation, and $\boldsymbol{\epsilon}$ is a base random variable.

Indeed, the above representation works for any distribution $q(\mathbf{w}; \Psi)$ that can be recast as a transformation $g(\boldsymbol{\epsilon}; \cdot)$ of base distribution $p(\boldsymbol{\epsilon})$. Still, the present case only deals with $q(\mathbf{w}; \Psi)$ as the product of independent univariate Gaussians (4.2) with parameters $\Psi = \{\boldsymbol{\mu}, \boldsymbol{\sigma}^2\}$. A convenient choice of transformation 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 $\boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$ for the uncorrelated Gaussian case, i.e., diagonal covariance matrix $\boldsymbol{\Sigma}$.

On a practical numerical note, one needs to prevent the $\sigma_i$ from assuming negative values during the optimization since $\sigma_i \geqslant 0$. Instead of imposing explicit

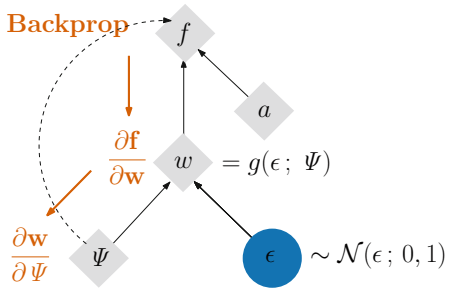



**Fig. 4.3** Computational graph after the reparameterization trick. The blue round node is a random node, while the gray rhombus nodes are deterministic. Black arrows represent the forward pass of the model and the red ones (part of) the backpropagation path. The black dashed line indicates the path for the computation of the KL divergence, that takes the distribution parameters $\Psi$ as input. Note that thanks to the reparameterization trick the node $w$ is no longer random and so we can compute its gradient as usual

constraints, the authors [6] suggest the *softplus* transform $\sigma_i = \log(1 + \exp \rho_i)$ that maps $\sigma_i$ to $\rho_i$, whose value is confined to the range $(0, \infty)$.

Computing the derivatives of (4.4) w.r.t. both elements of $\Psi = \{\boldsymbol{\mu}, \boldsymbol{\sigma}^2\}$ and using the chosen transformation give

$$\frac{\partial \mathcal{L}}{\partial \mu_i} = \frac{\partial h(\mathbf{w}, \Psi)}{\partial w_i} + \frac{\partial h(\mathbf{w}, \Psi)}{\partial \mu_i}, \tag{4.5}$$

$$\frac{\partial \mathcal{L}}{\partial \rho_i} = \frac{\partial h(\mathbf{w}, \Psi)}{\partial w_i} \frac{\epsilon}{1 + \exp(-\rho_i)} + \frac{\partial h(\mathbf{w}, \Psi)}{\partial \rho_i}. \tag{4.6}$$

The modification places the random component out of the gradient path followed by backpropagation, as illustrated in Fig. 4.3, where we depict a computational graph that computes a function $f$ with weights $\mathbf{w}$ from input activations $\mathbf{a}$. This modification allows the direct computation of the gradients w.r.t. $\mathbf{w}$ and $\Psi$ nodes in the computational graph just as done in any other deterministic node. Automatic differentiation tools available in common frameworks [1, 44, 53] handle this transparently, the only implementation difference being the need to explicitly reparameterize the weights $w = g(\boldsymbol{\epsilon}; \Psi)$ in the network definition and specify $\Psi = \{\boldsymbol{\mu}, \boldsymbol{\rho}\}$ as the learnable parameters. More modern versions of the frameworks include built-in functions that automatically perform this reparameterization implicitly.

Figure 4.4 shows the final graphical model for BBB with independent Gaussian priors with parameters $\{\mu_p, \sigma_p^2\}$, an example for which the KL term in (4.3) can be evaluated analytically through the closed-form solution

$$\mathcal{L}_{prior} = \sum_i^W \log \frac{\sigma_p}{\sigma_i} + \frac{1}{2\sigma_p^2}\left[(\mu_i - \mu_p)^2 + \sigma_i^2 - \sigma_p^2\right], \tag{4.7}$$

whose derivatives w.r.t. $\sigma_i$ and $\mu_i$ are trivial to calculate.

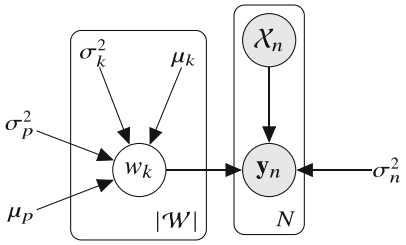



**Fig. 4.4** PGM representation of the model underlying the BBB method. The observed output $\mathbf{y}_n$ is a noisy observation of the model output for the input $\mathbf{x}_n$ with the variance noise determined by the fixed parameter $\sigma_n^2$. The constant values $\{\mu_p, \sigma_p^2\}$ govern the Gaussian prior distributions over the weights, while $\{\mu_k, \sigma_k^2\}$ their posteriors

For non-conjugate priors, such as a mixture of Gaussians, we can instead compute the KL numerically with the samples drawn from the posterior. This estimation has the immediate advantage of 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, what potentially leads to better results. In light of this change, instead of plugging (4.7) into (4.3), we write

$$\mathcal{L} \approx \sum_{i=1}^{T} -\log p(\mathbf{d} \mid \mathbf{W}^{(i)}) + \log q(\mathbf{w}^{(i)}; \Psi) - \log p(\mathbf{w}^{(i)}), \tag{4.8}$$

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

When using mini-batch optimization such that $\mathcal{D} = \{\mathbf{d}_j \mid 1 \leq j \leq M\}$, it is important to scale the complexity cost $\mathcal{L}_{prior}$ in the objective accordingly. Equation (4.7) accounts for the whole data set, so naively computing the loss $\mathcal{L}_j$ in (4.3) $M$ 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 $B_j = 1/M$ seem a natural choice, there are different ways of distributing them as long as $\sum_{j=1}^{M} B_j = 1$. In [6], 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 optimizing a BNN in Algorithm 1. 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 unbiased, the MC predictive log-likelihood estimator is biased, because a non-linear function, i.e., the log, warps

**Algorithm 1:** 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 $\mathbf{x}_i$
  $i \leftarrow (i + 1) \bmod N$
  $\pi_i \leftarrow 2^{N-i}/2^{N-1}$
  **for** $\mathbf{s} \in \{\mathbf{w}, \boldsymbol{\mu}, \boldsymbol{\rho}\}$ **do**
    $\mathbf{g}_\mathbf{s} \leftarrow -\nabla_\mathbf{s} \log p(\mathbf{x}_i|\mathbf{W}) + \pi_i \left(\nabla_\mathbf{s} \log q(\mathbf{w}; \Psi) - \nabla_\mathbf{s} \log p(\mathbf{w})\right)$
  **end for**
  $\Delta\boldsymbol{\mu} \leftarrow \mathbf{g}_\mathbf{w} + \mathbf{g}_{\boldsymbol{\mu}}$
  $\Delta\boldsymbol{\rho} \leftarrow \mathbf{g}_\mathbf{w} \odot \boldsymbol{\epsilon}/(1 + \exp(-\boldsymbol{\rho})) + \mathbf{g}_{\boldsymbol{\rho}}$
  $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} - k\Delta\boldsymbol{\mu}$
  $\boldsymbol{\rho} \leftarrow \boldsymbol{\rho} - k\Delta\boldsymbol{\rho}$
**end while**

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

### 4.3.1 Practical VI

The BBB algorithm [6] actually builds upon the work of Graves [16], which gets around the non-closed analytical form of the derivatives of $\mathcal{L}_{data}$ in a different manner. Instead of using the reparameterization trick to compute the derivatives, Practical VI uses the fact that the expectations are over the Gaussian distribution and employs the identities [7, 46]

$$\frac{\partial \mathbb{E}_q\left[f(\mathbf{w})\right]}{\partial \mu_i} = \mathbb{E}_q\left[\frac{\partial f(\mathbf{w})}{\partial w_i}\right], \tag{4.9}$$

$$\frac{\partial \mathbb{E}_q\left[f(\mathbf{w})\right]}{\partial \sigma_i^2} = \frac{1}{2}\mathbb{E}_q\left[\frac{\partial^2 f(\mathbf{w})}{\partial w_i^2}\right]. \tag{4.10}$$

Here, the generic function $f = -\log p(\mathbf{d} \mid \mathbf{W})$ and its expected value $\mathbb{E}_q\left[f(\mathbf{w})\right] = \mathcal{L}_{data}$, the likelihood cost term of (4.3). These identities are useful because they enable unbiased gradient estimates and have low variance when doing MC integration. Nevertheless, (4.10) requires second-order derivatives and even though the mean-field assumption saves us from computing the full Hessian matrix $\nabla^2_\mathbf{w}\mathcal{L}_{data}$, its diagonal is still necessary.

Using the Generalized Gauss-Newton (GGN) approximation [8] to the Hessian in (4.10) (see Appendix A.3), we obtain

$$\frac{\partial \mathbb{E}_q\left[f(\mathbf{w})\right]}{\partial \sigma_i^2} = \frac{1}{2}\mathbb{E}_q\left[\frac{\partial^2 f(\mathbf{w})}{\partial w_i^2}\right] \approx \frac{1}{2}\mathbb{E}_q\left[\left(\frac{\partial f(\mathbf{w})}{\partial w_i}\right)^2\right]. \tag{4.11}$$

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

Putting together the gradients for both $\mathcal{L}_{prior}$ and $\mathcal{L}_{data}$ terms, we have

$$\frac{\partial \mathcal{L}}{\partial \mu_i} \approx \frac{\mu_i - \mu_p}{\sigma_p^2} + \sum_{\mathbf{x}\in\mathcal{D}} \frac{1}{T}\sum_{k=1}^{T} \frac{\partial \log p(\mathbf{x}\,|\,\mathbf{W}^{(k)})}{\partial w_i}, \tag{4.12}$$

$$\frac{\partial \mathcal{L}}{\partial \sigma_i^2} \approx \frac{1}{2}\left(\frac{1}{\sigma_p^2} - \frac{1}{\sigma_i^2}\right) + \sum_{\mathbf{x}\in\mathcal{D}} \frac{1}{T}\sum_{k=1}^{T} \left[\frac{\partial \log p(\mathbf{x}\,|\,\mathbf{W}^{(k)})}{\partial w_i}\right]^2, \tag{4.13}$$

where $\{w_i\}_{i=0}^T$ are the MC samples, $\mathbf{x}$ are the data points, i.e., input, target pairs. We then optimize the objective (4.3) with a gradient-descent method $\Psi_{m+1} = \Psi_m - k\frac{\partial \mathcal{L}}{\partial \Psi_m}$.

As with the BBB method, observing (4.13) we note that this parameterization may cause $\sigma_i$ to assume negative values, thus calling for external constraints. Also similar to BBB, the Probabilistic Graphical Model (PGM) underlying Practical VI is the same as the one in Fig. 4.4. The difference between the two algorithms is rather a practical implementation issue, not a modeling assumption.

We summarize the resulting algorithm for optimizing a BNN with Practical ADF [16] in Algorithm 2. The case we illustrate is for a diagonal Gaussian variational posterior with parameters $\Psi = \{\boldsymbol{\mu}, \boldsymbol{\sigma}^2\}$ and centered 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.

**Algorithm 2:** Practical ADF

**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 $\mathbf{x}_i$
  $\mathbf{g} \leftarrow -\nabla \log p(\mathbf{x}_i|\mathbf{w})$
  $\Delta\boldsymbol{\mu} \leftarrow (\boldsymbol{\mu} - \mu_p\mathbf{1})/(N\sigma_p^2) + \mathbf{g}$
  $\Delta\boldsymbol{\sigma}^2 \leftarrow (\boldsymbol{\sigma}^2 - \sigma_p^2\mathbf{1})/\left(N\sigma_p^2\boldsymbol{\sigma}^2\right) + (\mathbf{g} \odot \mathbf{g})$
  $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} - k\Delta\boldsymbol{\mu}$
  $\boldsymbol{\sigma}^2 \leftarrow \boldsymbol{\sigma}^2 - k\Delta\boldsymbol{\sigma}^2$
**end while**

## 4.4 Probabilistic Backprop

Probabilistic Backpropagation (PBP) [19] solves the same problem as BBB but in a rather very different manner. While the algorithm of the previous section relies on optimizing the ELBO for the VI equation, PBP employs Assumed Density Filtering (ADF) and Expectation Propagation (EP), discussed in Sects. 3.2.2 and 3.2.3, 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 information about the new data points into the posterior approximation at each iteration. Although another EP-based method had been proposed before [50], it focused on binary weights and its continuous extension performed poorly, not estimating the posterior variance.

In the year following PBP's publication [19], other researchers developed a variant for binary and multi-class classification problems [15]. In [52], the authors adopted 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 book.

Similar to the previous method, PBP assumes independence among the network weights and the existence of additive Gaussian noise $\mathcal{N}(\epsilon \mid 0, \gamma^{-1})$ with precision $\gamma$ 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 specializes in fully connected layers with Rectified Linear Unit (ReLU) [38], 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 can glance upon this section.

The graphical model for PBP is illustrated in Fig. 4.5 and its full posterior distribution over the parameters is given by

$$\begin{aligned} p(w, \gamma, \lambda \mid \mathbf{X}) &= \frac{p(y \mid W, \mathbf{X}, \gamma) p(w \mid \lambda) p(\lambda) p(\gamma)}{p(y \mid \mathbf{X})} \\ &\propto p(y \mid W, \mathbf{X}, \gamma) p(w \mid \lambda) p(\lambda) p(\gamma), \end{aligned} \tag{4.14}$$

where $p(y \mid X)$ is the model evidence, $p(y \mid W, X, \gamma)$ the observation model defining the likelihood factors, $p(w \mid \lambda)$ the prior distribution over the weights composed of univariate Gaussians with precision $\lambda$, that is,

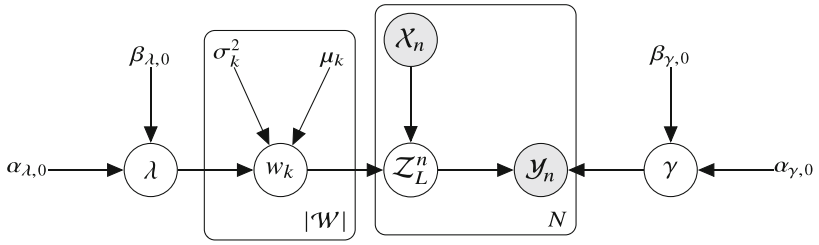



**Fig. 4.5** PGM representation of the PBP model. The observed output $y_n$ is a noisy observation of the model output $z_L^n$ for the input $x_n$. The hyper-parameter $\lambda$ governs the precision of the Gaussian prior distributions over the weights, whereas $\gamma$ governs the precision noise of the Gaussian observation model

$$p(w_1, \cdots, w_{|\mathcal{W}|} \mid \lambda) = \prod_{w \in \mathcal{W}} \mathcal{N}(w \mid 0, \lambda^{-1}), \tag{4.15}$$

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 $\text{Ga}(z \mid \alpha, \beta)$, given by

$$p(z \mid \alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} z^{\alpha-1} \exp(-\beta z), \tag{4.16}$$

for both hyper-priors. In Sect. 2.5, we proved that Gamma is the conjugate prior for the Gaussian distribution with known mean and unknown precision parameter.

From the analysis of the influence of the hyper-parameters on the Gamma posterior (2.42), we choose them such that they impose a weak prior, not affecting the posterior distribution. Exactly the same reasoning is valid for the hyper-prior on $\gamma$.

PBP uses EP and ADF (Sects. 3.2.3 and 3.2.2 respectively) to update the parameters $w_1, \cdots, w_{|\mathcal{W}|}$, $\alpha_\gamma$, $\beta_\gamma$, $\alpha_\lambda$, and $\beta_\lambda$ of the approximating distribution

$$\begin{aligned} &q(w_1, \cdots, w_{|\mathcal{W}|}, \lambda, \gamma) \\ &= \left[ \prod_{i=1}^{|\mathcal{W}|} \mathcal{N}(w_i \mid \mu_i, \sigma_i^2) \right] \text{Ga}\left(\lambda \mid \alpha_\lambda, \beta_\lambda\right) \text{Ga}\left(\gamma \mid \alpha_\gamma, \beta_\gamma\right), \end{aligned} \tag{4.17}$$

by cycling through the factors in (4.14) 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 two 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) independent and identically distributed (iid). However, this is clearly not the case when repeating the observations. Thus, the PBP should not run for many epochs. The authors [19] advise fewer than 100 and in our case study (Sect. 4.7) we run it for 40 epochs. Nevertheless, PBP is specifically designed for large data sets so this restriction does not matter much in practice. Yet, this is important to keep it in mind.

The models we analyze here and those employed in the original work have rather small sizes according to the current standards, i.e., one hidden layer with 50 units, so running EP updates is still feasible. Indeed, it is what the authors in [19] 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 $f_i(w_1, \cdots, w_{|\mathcal{W}|}, \lambda, \gamma)$ into the current approximation $q(w_1, \cdots, w_{|\mathcal{W}|}, \lambda, \gamma)$, such that the updated approximation is

$$K^{-1} f(w_1, \cdots, w_{|\mathcal{W}|}, \lambda, \gamma) q(w_1, \cdots, w_{|\mathcal{W}|}, \lambda, \gamma), \tag{4.18}$$

where $K^{-1}$ is a normalization constant that assures $q(w_1, \cdots, w_{|\mathcal{W}|}, \lambda, \gamma)$ 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 the same distribution class we had before the inclusion of the true factor, namely we minimize the KL divergence between the term in (4.18) and $q_{new}(w_1, \cdots, w_{|\mathcal{W}|}, \lambda, \gamma)$ w.r.t. $w_1, \cdots, w_{|\mathcal{W}|}, \lambda, \gamma$, the parameters of the new distribution $q_{new}$. As already shown in (3.40), 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 for the other methods we discuss.

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

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

### 4.4.1 Incorporating the Hyper-Priors $p(\lambda)$ and $p(\gamma)$

The first factors to incorporate into the approximation are the priors over $\gamma$ and $\lambda$. As shown in (2.42), 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 (4.14), that is

$$q_{new}(w_1, \cdots, w_{|\mathcal{W}|}, \lambda, \gamma) \propto \left[\lambda^{\alpha_\lambda - 1} \exp(-\lambda\beta_\lambda)\right]\left[\lambda^{\alpha_{\lambda,0}-1} \exp\left(-\lambda\beta_{\lambda,0}\right)\right]$$
$$\propto \lambda^{(\alpha_\lambda + \alpha_{\lambda,0} - 1) - 1} \exp\left(-\lambda\left(\beta_{\lambda,0} + \beta_\lambda\right)\right). \tag{4.19}$$

Thus, including the Gamma prior factors into $q$, and considering that $\alpha_\lambda = 0$, $\beta_\lambda = 1$, amounts to increment the values of the parameters $\gamma$ and $\lambda$ by

$$\alpha_{\gamma,\text{new}} = \alpha_\gamma + \alpha_{\gamma,0} - 1 = \alpha_{\gamma,0}\,,$$
$$\beta_{\gamma,\text{new}} = \beta_\gamma + \beta_{\gamma,0} = \beta_{\gamma,0}\,, \tag{4.20}$$

where we have used the values defined above, i.e., $\alpha_\lambda = \alpha_\gamma = 0$, $\beta_\lambda = \beta_\gamma = 1$.

Since there are no approximations in these relationship, and, hence no loss of information, the hyper-priors need to be included only once.

### 4.4.2 Incorporating the Priors on the Weights $p(\mathbf{w} \mid \boldsymbol{\lambda})$

Next, we incorporate the priors over the weights $w \in \mathcal{W}$. The unnormalized shifted distribution after the inclusion of one such factor is

$$q(w_1, \cdots, w_{|\mathcal{W}|}, \gamma, \lambda)\mathcal{N}(w_j \mid 0, \lambda^{-1}), \tag{4.21}$$

and the normalization constant is

$$K = \int q(w_1, \cdots, w_{|\mathcal{W}|}, \gamma, \lambda)\mathcal{N}(w_j \mid 0, \lambda^{-1}) d\mathbf{w} d\gamma d\lambda$$
$$= \int \underbrace{\prod_{i=1}^{|\mathcal{W}|} \mathcal{N}(w_i \mid \mu_i, \sigma_i^2)\text{Ga}(\lambda \mid \alpha_\lambda, \beta_\lambda)\,\text{Ga}\left(\gamma \mid \alpha_\gamma, \beta_\gamma\right)}_{q(w_1, \cdots, w_{|\mathcal{W}|}, \gamma, \lambda)} \times$$
$$\mathcal{N}(w_j \mid 0, \lambda^{-1}) dw_1 \cdots dw_{|\mathcal{W}|} d\gamma d\lambda$$
$$= \int \mathcal{N}(w_j \mid \mu_j, \sigma_j^2)\left[\int \mathcal{N}(w_j \mid 0, \lambda^{-1})\text{Ga}(\lambda \mid \alpha_\lambda, \beta_\lambda)\, d\lambda\right] dw_j$$
$$= \int \mathcal{N}(w_j \mid \mu_j, \sigma_j^2)\mathcal{T}_{2\alpha_\lambda}(w_j \mid 0, \beta_\lambda/\alpha_\lambda) dw_j, \tag{4.22}$$

where we have used the result demonstrated in (A.34), that the integral of the product of a Gamma and a Gaussian distributions is the t-Student's distribution

defined in (A.35). We continue the computation of $K$ by approximating the t-Student's distribution $\mathcal{T}_{2\alpha_\lambda}(w_j \,|\, 0, \beta_\lambda/\alpha_\lambda)$ with a Gaussian with same mean and variance, what as we saw in Fig. A.1 is within reason for enough degrees of freedom $\nu$, i.e., large $\alpha_\lambda$. Thus, continuing the calculation of $K$:

$$\begin{aligned} K &\approx \int \mathcal{N}(w_j \,|\, \mu_j, \sigma_j^2)\mathcal{N}(w_j \,|\, 0, \beta_\lambda/(\alpha_\lambda - 1))dw_j \\ &= \int \mathcal{N}\left(\mu_j \middle| 0, \sigma_j^2 + \frac{\beta_\lambda}{\alpha_\lambda - 1}\right) \mathcal{N}\left(w_j \,\middle|\, \frac{\lambda(\alpha_\lambda - 1)}{\beta\lambda + \alpha - 1}\frac{\mu}{\sigma^2}, \frac{\lambda(\alpha_\lambda - 1)}{\beta\lambda + \alpha - 1}\right) dw_j \\ &= \mathcal{N}\left(\mu_j \middle| 0, \sigma_j^2 + \frac{\beta_\lambda}{\alpha_\lambda - 1}\right) \int \mathcal{N}\left(w_j \,\middle|\, \frac{\lambda(\alpha_\lambda - 1)}{\beta\lambda + \alpha - 1}\frac{\mu}{\sigma^2}, \frac{\lambda(\alpha_\lambda - 1)}{\beta\lambda + \alpha - 1}\right) dw_j \\ &= \mathcal{N}\left(\mu_j \middle| 0, \sigma_j^2 + \frac{\beta_\lambda}{\alpha_\lambda - 1}\right), \end{aligned} \tag{4.23}$$

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

$$\mathcal{N}(w_j \,|\, \mu_1, \sigma_1^2)\mathcal{N}(w_j \,|\, \mu_2, \sigma_2^2) = \mathcal{N}(\mu_1 \,|\, \mu_2, \sigma_1^2 + \sigma_2^2)\mathcal{N}(w_j \,|\, \mu, \sigma^2), \tag{4.24}$$

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

#### 4.4.2.1 Update Equations for $\alpha_\lambda$ and $\beta_\lambda$

Updating the posterior approximation means matching its moments with those of the shifted distribution $s = K^{-1}q(w_1, \cdots, w_{|\mathcal{W}|}, \gamma, \lambda)\mathcal{N}(w_j \,|\, 0, \lambda^{-1})$. However, the sufficient statistics for $\lambda$ does not have closed form so we revise its parameters $\beta_\lambda$ and $\alpha_\lambda$ by matching only its first and second moments, which still produces good results [36].

Let us now derive those update formulas. We start by noting that $K$ in (4.23) is a function of $\mu_j, \sigma_j^2$, $\beta_\lambda$, and $\alpha_\lambda$, and make the dependency in the two latter terms explicit by writing $K(\beta_\lambda, \alpha_\lambda)$. Additionally, for brevity we denote $q(w_1, \cdots, w_{|\mathcal{W}|}, \gamma, \lambda)\mathcal{N}(w_j \,|\, 0, \lambda^{-1})$ as $f(\lambda)\text{Ga}(\lambda \,|\, \alpha_\lambda, \beta_\lambda)$ and compute

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

$$= \frac{K(\alpha_\lambda + 1, \beta_\lambda)\alpha_\lambda}{K(\alpha_\lambda, \beta_\lambda)\beta_\lambda}. \tag{4.25}$$

Similarly, we obtain for the second moment

$$\mathbb{E}_q\left[\lambda^2\right] = \frac{K(\alpha_\lambda + 2, \beta_\lambda)\alpha_\lambda\,(\alpha_\lambda + 1)}{K(\alpha_\lambda, \beta_\lambda)\beta_\lambda^2}. \tag{4.26}$$

Recalling the mean and variance formulas for the Gamma distribution (4.16), we equate them to the above expressions to obtain

$$\frac{\alpha_{\lambda,\text{new}}}{\beta_{\lambda,\text{new}}} = \frac{K(\alpha_\lambda + 1, \beta_\lambda)\alpha_\lambda}{K(\beta_\lambda, \alpha_\lambda)\beta_\lambda}, \tag{4.27}$$

$$\frac{\alpha_{\lambda,\text{new}}}{\beta_{\lambda,\text{new}}^2} = \frac{K(\alpha_\lambda + 2, \beta_\lambda)\alpha_\lambda(\alpha_\lambda + 1)}{K(\beta_\lambda, \alpha_\lambda)\beta^2} - \left[\frac{K(\alpha_\lambda + 1, \beta_\lambda)\alpha_\lambda}{K(\beta_\lambda, \alpha_\lambda)\beta_\lambda}\right]^2. \tag{4.28}$$

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

$$\alpha_{\lambda,\text{new}} = \left[K_0 K_2 K_1^{-2}(\alpha_\lambda + 1)\alpha_\lambda^{-1} - 1\right]^{-1}, \tag{4.29}$$

$$\beta_{\lambda,\text{new}} = \left[K_2 K_1^{-1}(\alpha_\lambda + 1)\beta_\lambda^{-1} - K_1 K_0^{-1}\alpha_\lambda\beta_\lambda^{-1}\right]^{-1}, \tag{4.30}$$

which are the update equations for the Gamma distribution over the precision parameter $\lambda$.

#### 4.4.2.2 Update Equations for the $\mu$ and $\sigma^2$

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 [17].

We first note that the shifted distribution can be conveniently written as $s = K^{-1} f(w_i)\mathcal{N}(w_i \mid \mu_i, \sigma_i^2)$, where $f(w_i)$ comprises all factors in

$$q(w_1, \cdots, w_{|\mathcal{W}|}, \gamma, \lambda)\mathcal{N}(w_i \mid 0, \gamma^{-1}) \tag{4.31}$$

except the $\mathcal{N}(w_i \mid \mu_i, \sigma_i^2)$, which we make explicit.

For $\mu_i$, we start from the easily verifiable identity

$$\nabla_{\mu_i}\mathcal{N}(w_i \mid \mu_i, \sigma_i^2) = \sigma_i^{-2}(w_i - \mu_i)\mathcal{N}(w_i \mid \mu_i, \sigma_i^2), \tag{4.32}$$

which we rearrange to

$$w_i \mathcal{N}(w_i \mid \mu_i, \sigma_i^2) = \mu_i \mathcal{N}(w_i \mid \mu_i, \sigma_i^2) + \sigma_i^2 \nabla_\mu \mathcal{N}(w_i \mid \mu_i, \sigma_i^2). \tag{4.33}$$

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

$$\begin{aligned} \int w_i K^{-1} f(w_i) \mathcal{N}(w_i \mid \mu_i, \sigma^2) dw_i = &\int \mu K^{-1} f(w_i) \mathcal{N}(w_i \mid \mu_i, \sigma^2) dw_i \\ &+ \int \sigma^2 K^{-1} f(w_i) \nabla_\mu \mathcal{N}(w_i \mid \mu_i, \sigma^2) dw_i \end{aligned} \tag{4.34}$$

$$\begin{aligned} \mathbb{E}_s[w_i] &= \mu + \sigma^2 K^{-1} \left[ \nabla_\mu \int f(w_i) \mathcal{N}(w_i \mid \mu_i, \sigma^2) dw_i \right] \\ &= \mu + \sigma^2 K^{-1} \nabla_\mu K \\ &= \mu + \sigma^2 \nabla_\mu \log K. \end{aligned} \tag{4.35}$$

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

$$\mu_{i,\text{new}} = \mu + \sigma^2 \nabla_\mu \log K. \tag{4.36}$$

Through a similar identity for the derivative w.r.t. $\sigma_i^2$:

$$\nabla_{\sigma_i^2} \mathcal{N}(w_i \mid \mu_i, \sigma^2) = \frac{\sigma_i^{-2}}{2} \left( -1 + \sigma_i^{-2} (w_i - \mu_i)^2 \right) \mathcal{N}(w_i \mid \mu_i, \sigma^2), \tag{4.37}$$

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

$$\text{Var}(w_i) = \mathbb{E}_s\left[w_i^2\right] - (\mathbb{E}_s[w_i])^2 = \sigma_i^2 - \left(\sigma_i^2\right)^2 \left[ \left(\nabla_\mu \log K\right)^2 - 2\nabla_{\sigma_i^2} \log K \right]. \tag{4.38}$$

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

$$\sigma_{i,\text{new}}^2 = \sigma_i^2 - \left(\sigma_i^2\right)^2 \left[ \left(\nabla_\mu \log K\right)^2 - 2\nabla_{\sigma_i^2} \log K \right]. \tag{4.39}$$

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 updated, 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.

### 4.4.3 *Incorporating the Likelihood Factors* $p(\mathbf{y} \mid \mathbf{W}, \mathbf{X}, \gamma)$

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 multilayer 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 also Gaussian. At the last layer, we obtain the distribution of the output $y_i$ given $\mathbf{x}_i$, to which we further apply the observation model, i.e., additive Gaussian noise with precision $\gamma$, which gives us $p(y_i \mid \mathbf{x}_i, \mathbf{W}, \gamma) = \mathcal{N}(y_i \mid f(\mathbf{x}_i, \mathbf{W}), \gamma^{-1})$. The likelihood factor is then included into the posterior approximation as usual: we shift the posterior by multiplying it by the likelihood factor stemming from the data point under consideration, 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 (4.29), (4.30), (4.36), (4.39) we have not assumed any specific format for the factors being included into the posterior approximation. So the same equations can be used once again for the likelihood factors, the sole change being what the normalizing constant $K$ is. In what follows we unveil the expression for $K$ for the likelihood factors $\mathcal{N}(y_i \mid f(\mathbf{x}_i, \mathbf{W}), \gamma^{-1})$.

#### 4.4.3.1 The Normalizing 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 [38].

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 normalizing constant $K$ of the associated shifted distribution as

$$K = \int q(\mathbf{w}, \gamma, \lambda)\mathcal{N}(\mathbf{y}_i \mid f(\mathcal{X}_i, \mathbf{w}), \gamma^{-1})d\mathbf{w}d\gamma d\lambda$$

$$
\begin{aligned}
&\approx \int q(\mathbf{w}, \gamma, \lambda)\mathcal{N}(\mathbf{y}_i \mid \mathbf{z}_L, \gamma^{-1})\mathcal{N}(\mathbf{z}_L \mid \mu_{\mathbf{z}_L}, \sigma^2_{\mathbf{z}_L})d\mathbf{w}d\mathbf{z}_L d\gamma d\lambda \\
&= \int \mathrm{Ga}\left(\gamma \mid \alpha_\gamma, \beta_\gamma\right)\mathcal{N}(\mathbf{y}_i \mid \mathbf{z}_L, \gamma^{-1})\mathcal{N}(\mathbf{z}_L \mid \mu_{\mathbf{z}_L}, \sigma^2_{\mathbf{z}_L})d\mathbf{z}_L d\gamma \\
&= \int \mathcal{T}_{2\alpha_\gamma}(\mathbf{y}_i \mid \mathbf{z}_L, \beta_\gamma/\alpha_\gamma)\mathcal{N}(\mathbf{z}_L \mid \mu_{\mathbf{z}_L}, \sigma^2_{\mathbf{z}_L})d\mathbf{z}_L \\
&\approx \int \mathcal{N}\left(\mathbf{y}_i \mid \mathbf{z}_L, \beta_\gamma/(\alpha_\gamma - 1)\right)\mathcal{N}(\mathbf{z}_L \mid \mu_{\mathbf{z}_L}, \sigma^2_{\mathbf{z}_L})d\mathbf{z}_L \\
&= \mathcal{N}(\mathbf{y}_i \mid \mu_{\mathbf{z}_L}, \beta_\gamma/(\alpha_\gamma - 1) + \sigma^2_{\mathbf{z}_L}),
\end{aligned} \tag{4.40}
$$

where we have followed the same steps and performed the same approximations as in the derivation of (4.23).

Computing the mean $\mu_{\mathbf{z}_L}$ and variance $\sigma^2_{\mathbf{z}_L}$ of the last layer $\mathbf{z}_L$ amounts to propagating the input through the entire network. If we assume that the layer $l-1$ has output $\mathbf{z}_{l-1}$ 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 $\mathbf{a}_l$ at the following layer according to

$$
\mu_{\mathbf{a}_l} = \mathbb{E}\left[\mathbf{W}_l\mathbf{z}_{l-1}/\sqrt{V_{l-1}+1}\right] = \bar{\mathbf{W}}_l\mathbf{z}_{l-1}/\sqrt{V_{l-1}+1} \tag{4.41}
$$

$$
\begin{aligned}
\sigma^2_{\mathbf{a}_l} &= \mathrm{Var}\left(\mathbf{W}_l\mathbf{z}_{l-1}/\sqrt{V_{l-1}+1}\right), \\
&= \frac{1}{V_{l-1}+1}\left[\left(\mathbb{E}\left[\mathbf{W}_l\right]\right)^2 \mathrm{Var}\left(\mathbf{z}_{l-1}\right) + \mathrm{Var}\left(\mathbf{W}_l\right)\left(\mathbb{E}\left[\mathbf{z}_{l-1}\right]\right)^2 + \mathrm{Var}\left(\mathbf{W}_l\right)\mathrm{Var}\left(\mathbf{z}_{l-1}\right)\right] \\
&= \frac{1}{V_{l-1}+1}\left[\left(\bar{\mathbf{W}}_l \odot \bar{\mathbf{W}}_l\right)\sigma^2_{\mathbf{z}_{l-1}} + \mathbf{V}_l\left(\mu_{\mathbf{z}_{l-1}} \odot \mu_{\mathbf{z}_{l-1}}\right) + \mathbf{V}_l\sigma^2_{\mathbf{z}_{l-1}}\right],
\end{aligned} \tag{4.42}
$$

where $\bar{\mathbf{W}}_l$ 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 that the pre-activation $\mathbf{a}_l$ is Normally distributed with the above mean and variance [50].

We are now to consider the effect of the non-linear activation function on $\mathbf{a}_l$. The $\max(0, a_{i,l})$ operation causes all probability density spread over $\mathbb{R}^-$ to concentrate at zero as Fig. 4.6 indicates. The resulting distribution is called rectified Gaussian and has its PDF given by

$$
\mathcal{N}^R\left(a_{i,l}; \mu_{i,l}, \sigma^2_{i,l}\right) = \Phi\left(-\frac{\mu_{i,l}}{\sigma_{i,l}}\right)\delta(a_{i,l}) + \frac{1}{\sqrt{2\pi\sigma^2_{i,l}}}e^{-\frac{(a_{i,l}-\mu_{i,l})^2}{2\sigma^2_{i,l}}}\mathrm{U}(a_{i,l}), \tag{4.43}
$$

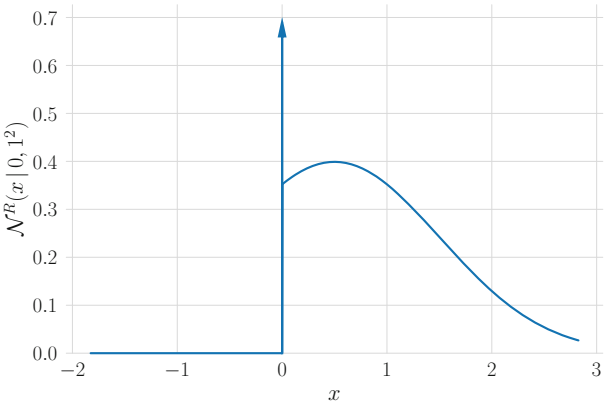



**Fig. 4.6** PDF of the rectified Gaussian distribution $\mathcal{N}^R(x; 0.5, 1^2)$

where $\mu$, $\sigma^2$ 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

$$\mu_{z_{i,l}} = \Phi\left(\frac{\mu_{a_{i,l}}}{\sigma_{a_{i,l}}}\right)\mu_{a_{i,l}} + \sigma_{a_{i,l}}\phi\left(-\frac{\mu_{a_{i,l}}}{\sigma_{a_{i,l}}}\right) \tag{4.44}$$

$$\sigma^2_{a_{i,l}} = m\left(\mu_{a_{i,l}} + \sigma_{a_{i,l}}\kappa\right)\Phi\left(-\frac{\mu_{a_{i,l}}}{\sigma_{a_{i,l}}}\right) + \Phi\left(\frac{\mu_{a_{i,l}}}{\sigma_{a_{i,l}}}\right)\sigma^2_{a_{i,l}}\left(1 - \kappa\left(\kappa + \frac{\mu_{a_{i,l}}}{\sigma_{a_{i,l}}}\right)\right), \tag{4.45}$$

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 $\mu_{z_L}$ and $\sigma^2_{z_L}$ consists in iteratively computing Eqs. (4.41), (4.42), (4.44), (4.45) from the first until the last layer for each data point $(\mathbf{x}_i, \mathbf{y}_i)$.

We summarize the many factors that compose PBP's distribution in Table 4.1 and highlight how many of each there are. The steps of the ADF-update-only method are available in Algorithm 3. We condense the forward pass responsible for computing the output distribution $\mathcal{N}(\mathcal{Y}_i \mid f(\mathcal{X}_i, \mathbf{w}), \gamma^{-1})$ into a single step in line 17.

It is important to note that the authors [19] assume the inputs are normalized, i.e., zero mean and unit variance. Hence, we need to normalize the data points before feeding them to the model and then to denormalize the obtained outputs.

**Table 4.1** Summary of PBP's factors, their distributions, and quantities

| Type | Symbol | Distribution | Quantity |
|---|---|---|---|
| Hyper-prior | $p(\lambda)$ | $\text{Ga}\,(\lambda \mid \alpha_\lambda, \beta_\lambda)$ | 1 |
| Hyper-prior | $p(\gamma)$ | $\text{Ga}\left(\gamma \mid \alpha_\gamma, \beta_\gamma\right)$ | 1 |
| Prior | $p(w_j \mid \lambda)$ | $\mathcal{N}(w_j \mid 0, \lambda^{-1})$ | $\lvert\mathcal{W}\rvert$ |
| Likelihood | $p(\mathcal{Y}_j \mid \mathbf{W}, \mathcal{X}_j, \gamma)$ | $\mathcal{N}(\mathcal{Y}_j \mid f(\mathcal{X}_j, \mathbf{w}), \gamma^{-1})$ | $N$ |

**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**
    $\mu_{\mathbf{z}_L}, \sigma_{\mathbf{z}_L}^2 \leftarrow f(\mathcal{X}_j, \mathcal{W})$
    **for** $s = 0$ **to** $2$ **do**
      $K_s \leftarrow \mathcal{N}(\mathbf{y}_i \mid \mu_{\mathbf{z}_L}, \sigma_{\mathbf{z}_L}^2) + \beta_\gamma/(\alpha_\gamma - 1 + s)$
    **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**

## 4.5 MC Dropout

The Monte Carlo Dropout [13], usually referred to as MC Dropout, stems from reinterpreting Dropout [51] as doing approximate Bayesian inference. Consequently, it suffices to use Dropout both during training and testing to obtain the advantages of Bayesian inference and model uncertainty measures.

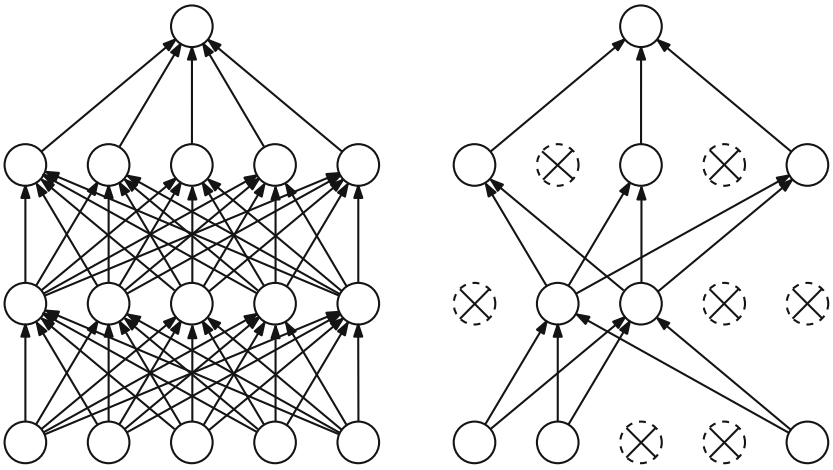



**Fig. 4.7** Effect of Bernoulli dropout on the network. Setting the output of a unit to zero is equivalent to removing that unit from the network. That nodes' inputs also become irrelevant because they are no longer propagated forward and so we remove them from the drawing in (**b**). (**a**) Standard neural network. (**b**) Neural network after dropout

### *4.5.1 Dropout*

First, we review Dropout [51]. Succinctly, it is a stochastic regularization 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 realization of a random vector $\boldsymbol{\epsilon}_l$, such that $\hat{\mathbf{h}_l} = \mathbf{h}_l \odot \boldsymbol{\epsilon}_l$.

In the case of Bernoulli dropout, each unit $h_{j,l}$ at layer $l$ is randomly *dropped out* with probability $1 - p$, i.e., its output value is set to zero, at each iteration according to $\epsilon_{j,l} \sim Bern(p)$, as illustrated in Fig. 4.7. Dropping units causes different subnetworks with considerably less parameters to be used at each iteration (12 instead of 55 in Fig. 4.7, for instance). 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 [28, 51], the authors study corrupting the activations with multiplicative Gaussian noise, and in [56], independently injecting noise on each weight, instead of on the input. The latter technique is called DropConnect.

### 4.5.2 A Bayesian View

Optimizing a model with dropout and an approximate Bayesian inference model leads to similar objective functions with similar stochastic gradient update steps. This similarity is so strong that under some conditions they are indeed equivalent [13]. Although we shall only consider here the Bernoulli Dropout, a similar development is possible for other types of noise.

Let us first review the cost function of a standard deterministic neural network $f(\cdot; \Theta)$ with deterministic parameters $\Theta$:

$$\mathcal{L} = \mathcal{L}_{data}\left(\mathcal{D}, f(\cdot; \Theta)\right) + \mathcal{L}_{reg}(\Theta), \tag{4.46}$$

where the first term is data-dependent and measures the model's prediction error, and the second is a regularization 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 $\ell_2$-norm with strength factors $\lambda_{\mathbf{M}}$, (4.46) becomes

$$\mathcal{L} = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{D}} \frac{1}{2} \left(\mathbf{y} - f(\mathbf{x}; \Theta)\right)^2 + \sum_{\mathbf{M}\in\Theta} \lambda_{\mathbf{M}} \|\mathbf{M}\|_2{}^2 \,. \tag{4.47}$$

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

$$\begin{aligned} \mathbf{h}_l &= g_l\left(\mathbf{M}_l \hat{\mathbf{h}}_{l-1}\right) \\ &= g_l\left(\mathbf{M}_l \left(\boldsymbol{\epsilon}_l \odot \mathbf{h}_{l-1}\right)\right) \\ &= g_l\left(\mathbf{M}_l \left(\text{diag}\left(\boldsymbol{\epsilon}_l\right)\mathbf{h}_{l-1}\right)\right) \\ &= g_l\left(\left(\mathbf{M}_l \text{diag}\left(\boldsymbol{\epsilon}_l\right)\right)\mathbf{h}_{l-1}\right) \\ &= g_l\left(\mathbf{W}_l \mathbf{h}_{l-1}\right), \end{aligned} \tag{4.48}$$

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

We have demonstrated that multiplying the input is equivalent to multiplying the columns of the upcoming weight matrix. Considering a Bernoulli distribution on each element of $\boldsymbol{\epsilon}_l$, when one of its entries assumes value equal to 0, it zeros the corresponding column of $\mathbf{W}_l$ (as $\mathbf{W}_l = \mathbf{M}_l \text{diag}\left(\boldsymbol{\epsilon}_l\right)$). Zeroing the column is equivalent to dropping every input of a neuron, which in turn is the same as dropping the neuron itself, as illustrated in Fig. 4.8.

Hence, applying dropout on a deterministic neural network can be interpreted as a transformation to a NN whose weights are sampled from a distribution. Looking

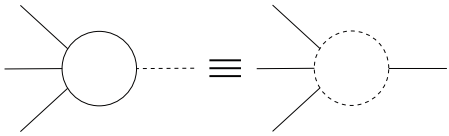


**Fig. 4.8** Dropping every input of a neuron (figure in the left) is equivalent to dropping the neuron itself

from this perspective, dropout is a way of using BNNs. Figure 4.10 shows the resulting weight matrix after being transformed by realizations of different types of noise.

If we now rewrite (4.47) taking this into consideration and make the sampling explicit, we get

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}\left(\mathbf{y}_i - f^{(i)}(\mathbf{x}_i;\Theta)\right)^2 + \sum_{l=1}^{L}\lambda_l\|\mathbf{M}_l\|_2{}^2, \tag{4.49}$$

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 $\Theta$ now defines distribution parameters, we replace it by $\Psi$ in order to keep compliance with our notation of variational parameters and ease the comparison with other methods.

Substituting the first term of the above equation according to (4.1), we obtain

$$\begin{aligned}\mathcal{L} &= -\frac{1}{N}\sum_{i=1}^{N}\sigma_n^2 \log p(\mathbf{y}_i \,|\, \mathbf{X}_i, \mathbf{W}^{(i)}) + \sum_{l=1}^{L}\lambda_l\|\mathbf{M}_l\|_2{}^2 - \frac{\sigma_n^2}{2}\log\left(2\pi\sigma_n^2\right)\\ &= -\frac{1}{N}\sum_{i=1}^{N}\sigma_n^2 \log p(\mathbf{y}_i \,|\, \mathbf{X}_i, \mathbf{W}^{(i)}) + \sum_{l=1}^{L}\lambda_l\|\mathbf{M}_l\|_2{}^2 + \text{const}\,,\end{aligned} \tag{4.50}$$

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

Equation (4.50) is pretty similar to a one-sample MC estimator of the VI cost function $\hat{\mathcal{L}}_{VI}$ defined in (3.8), and (4.3), which after approximating with MC integration becomes

$$\begin{aligned}\hat{\mathcal{L}}_{VI} &= -\frac{1}{T}\sum_{k=1}^{T}\log p(\mathbf{d} \,|\, \mathbf{W}^{(k)}) + D_{KL}\left(q(\mathbf{W}^{(k)};\Psi)\|p(\mathbf{W}^{(k)})\right)\\ &= -\log p(\mathbf{d} \,|\, \mathbf{W}^{(1)}) + D_{KL}\left(q(\mathbf{W}^{(1)};\Psi)\|p(\mathbf{W}^{(1)})\right)\\ &= -\sum_{i=1}^{N}\log p(\mathbf{y}_i \,|\, \mathbf{X}_i, \mathbf{W}^{(1)}) + D_{KL}\left(q(\mathbf{W}^{(1)};\Psi)\|p(\mathbf{W}^{(1)})\right).\end{aligned} \tag{4.51}$$

Taking the derivative of both (4.50) and (4.51) w.r.t. their parameters, we note that they possess the same objective (up to a constant scale factor), as long as we assure that

$$\frac{\partial}{\partial \Psi} D_{KL}\left(q(\mathbf{W}^{(1)}; \Psi) \| p(\mathbf{W}^{(1)})\right) = \frac{N}{\sigma_n^2} \frac{\partial}{\partial \Psi} \sum_{l=1}^{L} \lambda_l \|\mathbf{M}_l\|_2^2. \tag{4.52}$$

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 (4.52) to hold, we have to choose the hyper-parameters $\sigma_n$ and $\Lambda = \{\lambda_l | 1 \leqslant l \leqslant L\}$ such that they induce a sensible prior $p(\mathbf{W})$ for the underlying variational distribution $q(\mathbf{W}; \Psi)$. In the Appendix of [12], the author goes deeper in the conditions necessary to be attended in order for Eq. (4.52) to hold, where they assume that the weights of the neural network are sampled from a centered Gaussian distribution, i.e., $w_{j,l} \sim \mathcal{N}(0, \gamma_l^{-1})$.

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 Fig. 4.9, the other being readily attended: to every choice of dropout probability $1 - p_l$, observation noise $\sigma_n^2$ (or, equivalently, noise precision $\tau_n$), and regularization strength $\lambda_l$, corresponds a prior precision $\gamma_l$ (or, according to [13] 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 [12] and recurrent [14], 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 pretrained 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.

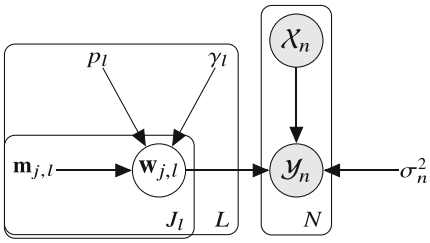



**Fig. 4.9** PGM representation of the MCDO model. The observed output $\mathbf{y}_n$ is a noisy observation of the model output for the input $\mathbf{x}_n$ with the variance noise determined by the fixed parameter $\sigma_n^2$. The $j$-th weight vector $\mathbf{m}_{j,l}$ out of the $J_l$ from the $l$-th layer get selected by Bernoulli random variables with success rate $p_l$ and $\mathbf{w}_{j,l}$ have centered Gaussian priors with fixed precision $\gamma_l$, whose value is readily determined by the choice of the two previous hyper-parameters together with the regularization strength $\lambda_l$

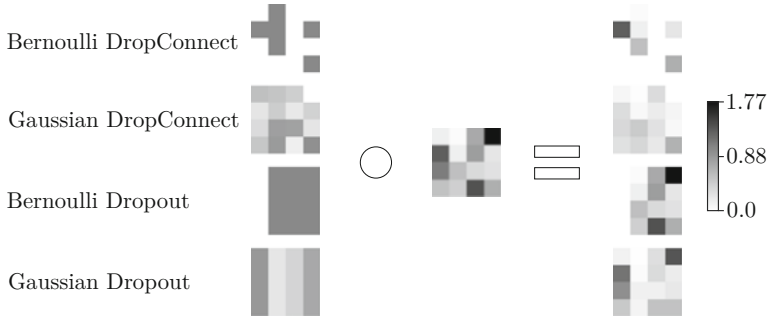



**Fig. 4.10** The effect on the network weights of using different stochastic regularization techniques on the same deterministic weight matrix **M**. Each technique corresponds to a distinct base distribution and leads to distinct variational distribution. For the same weight matrix, different base variational distributions

Also, the posterior approximation for Bernoulli Dropout factorizes 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, MCDO is less restrictive.

However, the author of [12] warns that to get well-calibrated uncertainty estimates the dropout probability must be optimized as well. Since this is a variational parameter, it cannot be directly chosen by observing the ELBO objective [12]. The recommendation is then to set it by maximizing the log-likelihood over a validation set.

We summarize the resulting procedure in Algorithm 4, where 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 regularizers can be recast as performing approximate Bayesian inference by following a similar derivation, as Fig. 4.10 illustrates. For example, for DropConnect [56] the only difference is in using separate random variables for each weight instead of one for each column of the weight matrix. It is important to note that not all resignifications go without problems, Gaussian Dropout [51] as Bayesian inference with a log-uniform prior [28] has had some issues pointed out in [24]. If instead of using multiplicative we consider additive Gaussian noise for each weight parameter, we recover the algorithm of Sect. 4.3.1.

## 4.6 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 [2]. Natural gradient

**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**
  $\mathbf{g} \leftarrow \frac{1}{2}\nabla \left(\mathbf{y}_i - f(\mathbf{x}_i; \{\mathbf{W}_l\}_{l=0}^{L})\right)^2 + \sum_{l=1}^{L} \lambda_l \nabla \|\mathbf{W}_l \text{diag}(\boldsymbol{\epsilon}_l)\|_2{}^2$
  $\mathbf{m}_{j,l} \leftarrow \mathbf{m}_{j,l} - k\mathbf{g}$
**end while**

methods do that by warping the gradient according to the information geometry encoded into the Fisher information matrix (see Appendix A.4). As a consequence, they are invariant (up to first order) to changes in the parameterization of the problem, what is in stark contrast to standard gradient descent, whose efficiency and convergence rate are sensitive to the parameterization.

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 Sects. 4.3 and 4.4 have twice the number of parameters of a deterministic model with the same architecture, besides the additional implementation effort. Adaptive optimizers further enlarge the costs since each parameter has its own scaling variable that regulates the learning rate.

The authors in [26] build upon previous work [25] on natural gradient for Gaussian MFVI and propose a series of progressively more practical but less accurate optimizers. It is a lengthy read to grasp all the details, but certainly worth the effort. Here, we review and rederive the core algorithm of [26], named Vadam.

### *4.6.1 Vadam*

From all reviewed methods, Vadam [26] is the more recent and practical one. Similar to the Adam optimizer [27] by construction, it is a natural gradient method (see Appendix A.4) with momentum designed specifically for MFVI. Starting from a parameter update equation proposal, the authors [26] embed several approximations defining different algorithms until reaching the method they name Vadam, for Variational Adam.

Gradient optimizers with momentum establish the update step as a linear combination between the steepest descent direction and the last displacement [8], such as

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \bar{\alpha}_t \nabla_{\mathbf{w}} f(\mathbf{w}_t) + \bar{\gamma}_t (\mathbf{w}_t - \mathbf{w}_{t-1}), \tag{4.53}$$

where $\{\bar{\alpha}_t\}$ and $\{\bar{\gamma}_t\}$ form a sequence of scalars that determines the contribution of each term and must obey the convergence conditions discussed in Sect. 3.2.1.5.

The latter term in (4.53) 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 [8].

Instead of (4.53), the authors of [26] propose

$$\boldsymbol{\eta}_{t+1} = \boldsymbol{\eta}_t - \bar{\alpha}_t \widetilde{\nabla}_{\boldsymbol{\eta}} f(\boldsymbol{\eta}_t) + \bar{\gamma}_t(\boldsymbol{\eta}_t - \boldsymbol{\eta}_{t-1}), \tag{4.54}$$

where $\widetilde{\nabla}$ is the natural gradient and the optimization is on the natural parameter $\boldsymbol{\eta}$ of an exponential family member. For such family, the natural gradient assumes a simple and efficient form, requiring less memory and computations. 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 [3]

$$\widetilde{\nabla}_{\boldsymbol{\eta}} f(\boldsymbol{\eta}) = \mathcal{I}^{-1}(\boldsymbol{\eta}) \nabla_{\boldsymbol{\eta}} f(\boldsymbol{\eta}) = \nabla_{\mathbf{m}} f(\mathbf{m}), \tag{4.55}$$

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 parameterized according to $\mathbf{m} = \mathbb{E}[\mathbf{u}(\mathbf{w})]$. The identity in (4.55) frees us from computing the Fisher matrix and its inverse, that is why it is so useful and many other practical natural gradient algorithms resort to it [22, 23, 25].

In the specific case of independent univariate Gaussian weights (mean-field assumption), writing (4.54) as a minimization problem with a KL constraint (see Appendix A.4), using (4.55) and solving the resulting Lagrangian lead to

$$\mu_{t+1} = \mu_t - \frac{\beta_t}{1-\alpha_t}\sigma_{t+1}^2 \nabla_\mu \mathcal{L}_t + \frac{\alpha_t}{1-\alpha_t}\sigma_{t+1}^2 \sigma_{t-1}^{-2}(\mu_t - \mu_{t-1}), \tag{4.56}$$

$$\sigma_{t+1}^{-2} = \frac{1}{1-\alpha_t}\sigma_t^{-2} - \frac{\alpha_t}{1-\alpha_t}\sigma_{t-1}^{-2} + \frac{2\beta_t}{1-\alpha_t}\nabla_{\sigma^2}\mathcal{L}_t. \tag{4.57}$$

The pair of update Eqs. (4.56) and (4.57) is the natural momentum extension of [25]. We immediately note that the learning rate of $\mu$ gets scaled by the variance. Additionally, $\sigma^2$ may assume negative values just like the methods in Sect. 4.3, thus 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. However, now we take into consideration that the cost function is the negative ELBO defined in (4.3) and also 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 (4.7)–(4.13) and again using the identities (4.9) and (4.10), we get

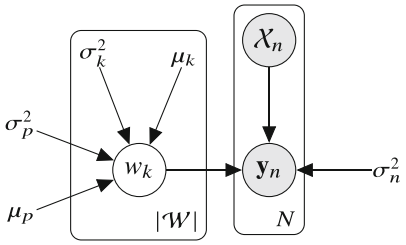



**Fig. 4.11** PGM representation of the Vadam model. It is the same as the one in Sect. 4.3. The observed output $\mathbf{y}_n$ is a noisy observation of the model output for the input $\mathbf{x}_n$ with the variance noise determined by the fixed parameter $\sigma_n^2$. The constant values $\{\mu_p, \sigma_p^2\}$ govern the Gaussian prior distributions over the weights, while $\{\mu_k, \sigma_k^2\}$ their posteriors

$$\nabla_\mu \mathcal{L} = N\mathbb{E}_q\left[\nabla_w h(w)\right] + \frac{\mu}{\sigma_p^2}, \tag{4.58}$$

$$\nabla_{\sigma^2} \mathcal{L} = \frac{N}{2}\mathbb{E}_q\left[\nabla_w^2 h(w)\right] + \frac{1}{2}\left(\frac{1}{\sigma_p^2} - \frac{1}{\sigma^2}\right), \tag{4.59}$$

where $N$ is the data set size and $h(w) = -\frac{1}{N}\sum_{i=1}^{N} \log(x_i \mid w)$, the average negative log-likelihood.

Now, after determining the prior and posterior distributions over the weights, we have completely defined the underlying Vadam model, shown in Fig. 4.11. t has the same structure as the one used in Sect. 4.3 (Fig. 4.4), the difference between both methods being the approximations included in Vadam to make it more computationally efficient.

As one might already expect, we use one-sample MC estimators for the expectations in (4.58) and (4.59), as well as replace the gradients, so far computed from the entire data set, with their stochastic versions $\widehat{\nabla}_w$ and $\widehat{\nabla}_w^2$, computed from a mini-batch. Since second derivatives are computationally expensive, besides (4.59) being able to lead $\sigma^2$ to negatives values, we resort to the GGN approximation for $\widehat{\nabla}_w^2$ (see Appendix A.3). This last step requires calculating the square of the first-order derivative for each mini-batch element; however, modern frameworks are not optimized to operate separately on each element of a batch after computing its derivatives. Thus, we incorporate yet another approximation

$$\widehat{\nabla}_w^2 h(w_t) \approx \frac{1}{M}\sum_{i=1}^{M} \widehat{\nabla}_w h(w_t; \mathbf{x}_i)^2 \approx \left(\frac{1}{M}\sum_{i=1}^{M} \widehat{\nabla}_w h(w_t; \mathbf{x}_i)\right)^2. \tag{4.60}$$

While the first approximation in (4.60) is the GGN, the last is known as the gradient magnitude approximation and employed by several usual optimizers [11, 27, 54]. It causes $\sigma^2$ to act as diagonal rescaling that simply assures equal progress along each axis of $\mu$ rather than closely approximating the curvature [8] (disregarding the

momentum term of the update equation that counter-balances this effect by favoring 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. As the mini-batch size increases, the bias also increases: 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.

For practicality, the authors in [26] define the scaled prior precision $\widetilde{\lambda} = \sigma_p^{-2}/N$ and the new parameter $s_t = (\sigma_t^{-2} - \sigma_p^{-2})/N$. Moreover, they arbitrarily apply square root on the scaling vector $s_t$ in the $\mu$ 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 [26]. The Vadam weight update equations are then

$$\mu_{t+1} = \mu_t - \bar{\alpha}_t \left[\frac{1}{\sqrt{s_t} + \widetilde{\lambda}}\right] \left(\widehat{\nabla}_w h(w_t) + \mu_t \widetilde{\lambda}\right) + \bar{\gamma}_t \left[\frac{\sqrt{s_t} + \widetilde{\lambda}}{\sqrt{s_{t+1}} + \widetilde{\lambda}}\right] (\mu_t - \mu_{t+1}), \tag{4.61}$$

$$s_{t+1} = (1 - \bar{\alpha}_t)\, s_t + \bar{\alpha}_t \widehat{\nabla}_w^2 h(w_t), \tag{4.62}$$

where $\widehat{\nabla}_w$ and $\widehat{\nabla}_w^2$ are the unbiased stochastic approximations of $\nabla_w$ and $\nabla_w^2$, respectively.

Unwinding these update equations and using different step sizes $\gamma_1$ and $\gamma_2$ for $\mu$ and $s$ instead of $\bar{\alpha}_t$ and $(1-\bar{\alpha}_t)$, respectively, we get the Algorithm 5. Remember that the scale factor $s$ actually relates to $\sigma^2$ by $\sigma_t^{-2} = N s_t + \sigma_p^{-2}$ and each weight sample $w_t$ is drawn from the distribution $\mathcal{N}(\mu_t, \sigma_t^2)$. The implementation differences from Adam [27], in red in Algorithm 5, are responsible for enabling ADF.

**Algorithm 5:** 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 $\mathbf{x}_i$
  $\mathbf{g} \leftarrow -\nabla \log p(\mathbf{x}_i|\mathbf{W})$
  $\mathbf{m} \leftarrow \gamma_1 \mathbf{m} + (1 - \gamma_1)\,(\mathbf{g} + \sigma_p^{-2}\mu/N)$
  $\mathbf{s} \leftarrow \gamma_2 \mathbf{s} + (1 - \gamma_2)\,(\mathbf{g} \odot \mathbf{g})$
  $\hat{\mathbf{m}} \leftarrow \mathbf{m}/(1 - \gamma_1^t), \quad \hat{\mathbf{s}} \leftarrow \mathbf{s}/(1 - \gamma_2^t)$
  $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} - \alpha\, \hat{\mathbf{m}}/(\sqrt{\hat{\mathbf{s}}} + \sigma_p^{-2}/N)$
  $t \leftarrow t + 1$
**end while**

Throughout the development of the Vadam algorithm, it is 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 [27]. In order to reduce this effect, the authors [26] introduce a bias-correcting factor that decays exponentially as the optimization runs.

The final method is indeed very similar to Adam [27], 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 ADF. Differently from the previous methods of this chapter, the user has only to define the model as if it were deterministic and optimize it with Vadam. There is no silver bullet and the price for such easiness and speed is inferior posterior estimates.

## 4.7 Comparing the Methods

In the remainder of this section, we compare the four algorithms studied during the chapter. We begin with one-dimensional toy examples, for which we can visually analyze the predicted curves and better grasp some of the discussed ideas. Next, we benchmark them on more complex regression tasks, whose results work as better guidelines on possible practical scenarios.

### *4.7.1 1-D Toy Example*

As a first experiment, we evaluate the predictive distribution obtained from the approximation algorithms on toy regression data sets whose targets are given by

$$y = -(x + 1)\sin(3\pi x) + \epsilon, \text{ where } \epsilon = \mathcal{N}(0, 0.3^2). \tag{4.63}$$

For this task, we uniformly sample a 20-point and a 400-point sets. We train one-hidden-layer networks with 100 units until convergence using the Adam optimizer (except for the Vadam algorithm, which is an optimizer itself). Figure 4.12 shows the results we obtain.

Let us first recall what was our intention with BNNs: to better model the underlying distribution of our problem and quantify the unknown. Thus, we would like to see our models' uncertainty increase in regions with few to none samples. In that sense, no matter how dense our knowledge about the function may be in the center region of Fig. 4.12, we essentially do not know much outside it, so our uncertainty estimation should not change much in these off-center regions. Conversely, the more samples we have in a given region, the better our accuracy should get and the more certain our model should be in that region.

We can notice that Vadam's uncertainty in the 400-sample scenario remain high whereas the other methods' estimate shrink considerably more in proportion to their 20-sample case. Even BBB, whose predictions look similar to Vadam's, more severely underestimates the variance of the posterior, specially in the 400-point set.

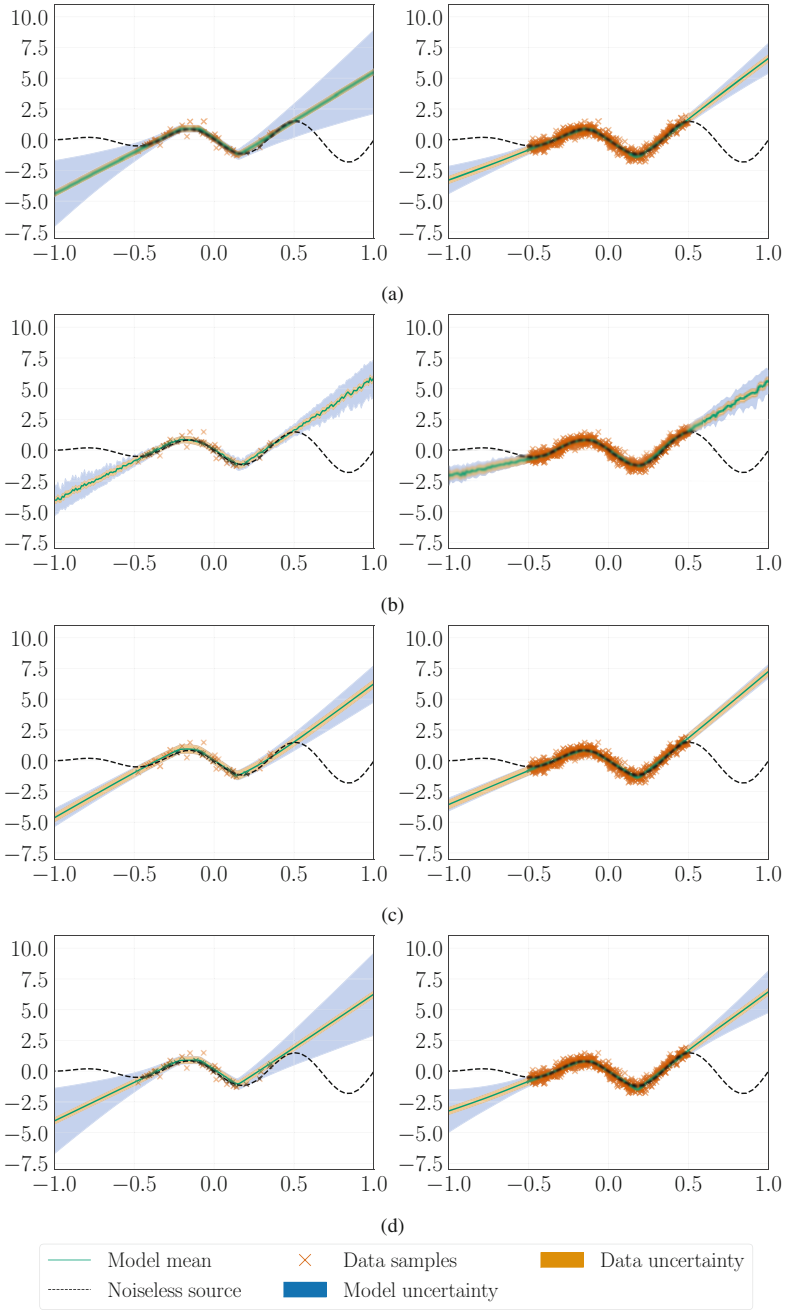



**Fig. 4.12** Comparison of the resulting predictive distribution of a one-dimensional toy example with either 40 (left) or 400 (right) data point obtained by (**a**) BBB, (**b**) MCDO, (**c**) PBP, and (**d**) Vadam

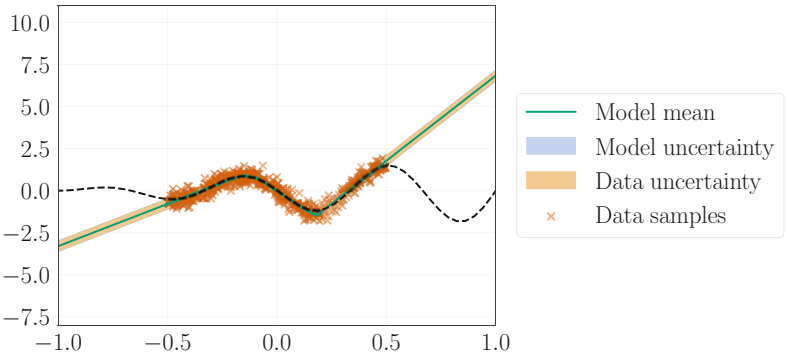



**Fig. 4.13** Predictive posterior distribution for the PBP method when trained for 200 epochs. Note that the blue shades (model uncertainty) have disappeared

MCDO's predictions are considerably less smooth than those from other algorithms, needing more MC samples to obtain stable results. Nonetheless, the mean predictions accurately capture the underlying behavior of the target function. Although the algorithm's iterations are individually less computationally expensive, more iterations are required to adequately model the data. Even with a small number of samples, MCDO obtains good estimates for both the mean and the variance.

In larger data sets, where only a reduced number of passes over the data (fewer than 100) are possible, performing EP requires a unreasonably large memory footprint, so PBP actually performs multiple ADF passes through the data, treating each as a novel example. A disadvantage of this approach is that it can lead to underestimation of the posterior variance when too many passes are done over the data. This is the behavior we observe in Fig. 4.13 compared to the one in Fig. 4.12c for the same number of samples.

### *4.7.2 UCI Data Sets*

Now, with a better understanding, we benchmark the algorithms in eight different regression data sets of the UCI Machine Learning Repository [10], a procedure that has recently become a standard in the related literature [13, 18, 19, 26, 34, 52, 58]. Below we give a brief description of each data set.

#### 4.7.2.1 Boston Housing

Data about homes from various suburbs in Boston, Massachusetts, collected in 1978 by the US Census Service. The task is to predict the median value of owner-occupied homes from 13 variables measuring property and neighborhood characteristics such

as average number of rooms per dwelling, nitric oxides concentration, per capita crime rate, among others. From now on, we refer to it as Boston only.

#### 4.7.2.2 Concrete Compressive Strength

Concrete strength is one of the most important engineering properties of concrete and very important in civil engineering. It is a highly non-linear function of age and ingredients, usually obtained by testing samples under a compression testing machine. The aim of the data set is to predict the compressive strength given the age and seven ingredients, such as cement, water, fine and coarse aggregate, among other component concentrations. From now on, we call it Concrete.

#### 4.7.2.3 Energy Efficiency

During the design of a building, simulations are performed to estimate its energy efficiency. The task is to predict the efficiency, which is expressed by the 2 different metrics heating and cooling load, from eight attributes, such as glazing area, surface area, orientation. The data set is composed of a collection of 768 simulated buildings with different characteristics and 12 different shapes. For brevity, we call this data set Energy.

#### 4.7.2.4 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.

#### 4.7.2.5 Condition Based Maintenance of Naval Propulsion Plants

The behavior and interaction of the main components of propulsion systems cannot be easily modeled with a priori physical knowledge. Still, it is important to continuously monitor the propulsion equipment and take decisions based on their condition. The aim is to predict the compressor decay state coefficient using input features such as ship speed, fuel flow, torques from turbine and propellers, temperatures and pressures coming in and out of the compressor. The data set was generated from a numerical simulator of a Navy frigate characterized by a combined diesel-electric and gas propulsion plant. The simulator was fine-tuned and validated with real-data. In what follows, we refer to this data set as Naval.

#### 4.7.2.6 Combined Cycle Power Plant

The aim of this data set is to predict the net hourly electrical energy output of a power plant. Power output prediction is an important element in managing a plant and its connection to the power grid. The features come from a real plant and were collected for over 6 years, when the plant was set to work with full load. They consist of hourly averages of temperature, ambient pressure, relative humidity, and exhaust vacuum. From this point on, we call this data set Powerplant.

#### 4.7.2.7 Wine Quality

The data consists of 11 physicochemical characteristics of different brands of red and variants of the Portuguese "Vinho Verde" wine. The objective is to predict the quality of the wine, a score between 0 and 10. This data set is hereafter called Wine.

#### 4.7.2.8 Yacht Hydrodynamics

Estimation of the residuary resistance of sailing yachts at the initial design stage is essential for evaluating the ship's performance and for assessing the required propulsive power. The data set contains results from 308 full-scale experiments of 22 different hull forms. The input features are aspects of hull geometry. From now on, we call this data set Yacht.

### *4.7.3 Experimental Setup*

For each data set in Sect. 4.7.2, we compare algorithms according to their training time, predictive (Gaussian) log-likelihood (4.1), and Root Mean Squared Error (RMSE), which is defined as

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{N}(\mathbf{y}_i - \hat{\mathbf{y}}_i)^2}{N}}. \tag{4.64}$$

While RMSE exclusively measures the prediction accuracy, thus assessing how close the predictions are to the target values, the log-likelihood takes into account the prediction variance and thus incorporates the prediction uncertainty into the evaluation. 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 gives us a sense of the model uncertainty, though indirectly.

We follow the setup proposed in [13]: for each data set we run the models on 20 random train-test splits after doing hyper-parameter search with 30 iterations of Bayesian Optimization (BO) [49] on each split. It is important to note that Bayesian Optimization (BO) has nothing to do with the previously discussed methods for training BNNs, we use it as a tool for efficiently searching the hyper-parameter space. We could have employed random or grid search instead and the arguments developed throughout the chapter would still be the same, as would our pipeline illustrated in Fig. 4.14.

#### 4.7.3.1 Hyper-Parameter Search with Bayesian Optimization (BO)

Bayesian Optimization (BO) is a black-box approach to optimize objective functions that take a long time or are costly to evaluate. Bayesian Optimization (BO) builds a surrogate for the objective and quantifies the uncertainty in that surrogate through Gaussian Process regression [47]. At each iteration, we observe the objective at a new point (a new hyper-parameter configuration), update the posterior distribution that describes the potential objective values at each point, and sample a new point whose values maximize a given acquisition function, i.e., the expected improvement. Bayesian Optimization (BO) factors in all previously seen configurations to decide what point of the parameter space to investigate next, achieving 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.

We use the same 20 data splits[1] for the methods to avoid fluctuations in the results due to the reduced size of the data sets and the effect different splits may have. For each split, we set the optimal hyper-parameter configurations of the prior precision $\lambda$ (or equivalently the prior variance $\sigma_p^2$), the observation noise precision $\gamma$, and, in the MC Dropout case, the dropout probability $p$ by running 30 iterations of Bayesian Optimization (BO) on the training set for 40 epochs. Additionally, the

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

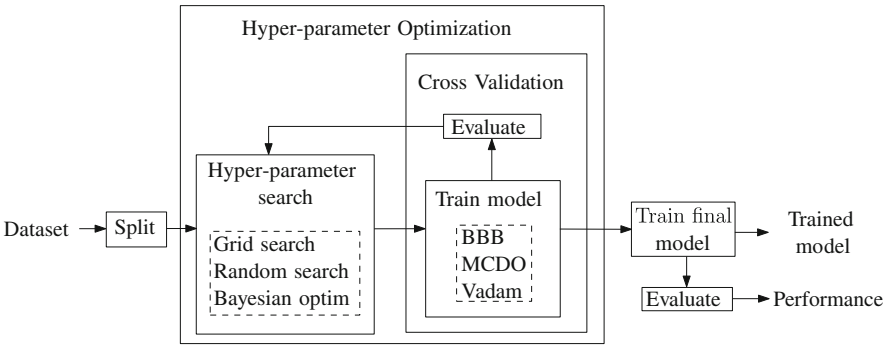



**Fig. 4.14** Experimental pipeline for evaluating BNNs. Each data set is split as in [13], then we do 30 rounds of hyper-parameter optimization, training the model on a random subset of the split with one of the discussed methods in a cross-validation setting on each round. At the end, we use the best hyper-parameters found to train the final model on the whole split

performance of the hyper-parameter configuration of each Bayesian Optimization (BO) iteration is averaged over a 5-fold cross-validation, so for each setting we train and evaluate the model 5 times. After finding the best configuration for each split, we fit the model to the whole training set. All this procedure follows from [13]. The general pipeline of the approach is summarized in Fig. 4.14.

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

$$\underbrace{20}_{\text{splits}} \times \left( \underbrace{30}_{\text{BO iters}} \times \underbrace{\left(5 \times \frac{4}{5}\right)}_{\text{CV iters}} + \underbrace{1}_{\text{full run}} \right) \times \underbrace{40}_{\text{epochs}} = \underbrace{96800}_{\text{epochs}}, \tag{4.65}$$

plus the time each Bayesian Optimization (BO) iteration takes to decide on the next point to test. We do small scale studies with single hidden-layer networks with 50 units to keep the computation under a viable amount of time.

The source codes for PBP[2] and Vadam[3] were borrowed from the authors' repositories. With exception of PBP which is implemented in Theano 1.0 [53], all remaining algorithms and supporting code are in PyTorch 1.0 [44].

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

[3] https://github.com/emtiyaz/vadam.

**Table 4.2** Number of MC samples during training for each algorithm. PBP does not employ MC integration, instead it uses analytical approximations

| | Smaller sets | Larger sets |
|---|---|---|
| BBBx1 | 10 | 5 |
| BBBx2 | 20 | 10 |
| MCDOx1 | 1 | 1 |
| MCDOx10 | 10 | 10 |
| Vadam | 10 | 5 |
| PBP | – | – |

### *4.7.4 Training Configuration*

We maintain a training configuration similar to [26]. We use a mini-batch of size 32 on the four smaller data sets (Boston, Concrete, Energy, and Yacht), and of 128 on the other four (Wine, Powerplant, Naval, Kin8nm). Table 4.2 contains the number of MC samples used during training for each algorithm, for evaluation they all use 100 MC samples.

We run BBB and MCDO under two different conditions to further investigate their behavior. BBBx2 draws twice the number of MC samples of BBBx1 during training. As for MC Dropout, our MCDOx1 configuration follows the original MC Dropout implementation [13], i.e., just one MC sample during training and longer training time after hyper-parameter selection, namely 400 epochs instead of 40, since MC Dropout takes longer to converge [13]. On the other hand, MCDOx10's training procedure is more similar to the other algorithms' setup: 10 training MC samples and 40 epochs.

In PBP's original implementation [19], which we use, samples are individually processed, but mini-batching is possible at the cost of slightly reduced performance. PBP also has no MC approximation of the weights' posterior since it propagates entire distributions through the layers, analytically performing its approximations.

BBB, MCDO, and Vadam use gradient-descent optimizers. Both BBB and MC Dropout use the Adam optimizer [27], while Vadam is itself a (variational) optimizer and the experiment consists of using it in lieu of Adam. Following [26], in all three methods we set 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 initial precision for the posterior approximation is set to 10 (attention, this is not the prior precision) for BBB and Vadam.

### *4.7.5 Analysis*

For comparing the algorithms according to the performance in each individual data set, we use the Bayesian Correlated t-test [9]. This test is used for the analysis of cross-validation results and accounts for the correlation due to the overlapping

**Table 4.3** 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

| Data set | *Size* | Dim | Absolute avg. running time (s) | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | BBBx1 | BBBx2 | Vadam | **PBP** | MCDOx1 | MCDOx10 |
| Boston | 506 | 13 | 1813 | 3279 | 2214 | 16 | 1286 | 1339 |
| Concrete | 1030 | 8 | 3510 | 6101 | 4333 | 28 | 2280 | 2442 |
| Energy | 768 | 8 | 2680 | 4312 | 3283 | 20 | 1541 | 912 |
| Kin8nm | 8192 | 8 | 4563 | 8433 | 4985 | 190 | 4493 | 4631 |
| Naval | 11,934 | 16 | 6923 | 14036 | 6835 | 279 | 6759 | 6916 |
| Powerplant | 9568 | 4 | 5349 | 9993 | 6117 | 188 | 5356 | 5500 |
| Wine | 1599 | 11 | 1009 | 2269 | 1226 | 41 | 1098 | 1076 |
| Yacht | 308 | 6 | 1139 | 1634 | 1291 | 10 | 612 | 275 |

training sets [5]. It is thus suited to our case where we have 20 random splits with 90% for training and 10% for testing.

PBP automatically sets all its hyper-parameters by the Bayesian framework thanks to the hyper-priors, thus dispensing with the hyper-parameter search. In this case, the number in (4.65) reduces from 96800 epochs to $20 \times 40 = 800$, that is, one 40-epoch run per random split. Table 4.3 shows the required (wall-clock) time each algorithm takes to complete the full training schedule (including Bayesian Optimization (BO)). Figure 4.15 illustrates a similar information, but depicts the ratio w.r.t. PBP training time for easier visualization.

PBP outspeeds all others being 34 times faster than the runner-up. 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 optimization inference. Instead of requiring computer time, PBP requires human time to workout all its derivations and approximations. However, if we were not performing hyper-parameter tuning, 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 [4], 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 were implemented in Pytorch [44], a more recent and rapidly growing framework powered by Facebook Artificial Intelligence Research and developed by dedicated personnel, hence the operations are better optimized to GPU processing.

BBBx2 is by far the slowest, taking on average 80% more time to train than BBBx1. Although Vadam has a poorer average performance than BBBx2 as seen in the performance bar chars of Figs. 4.16 and 4.17, it trains faster: only 16%

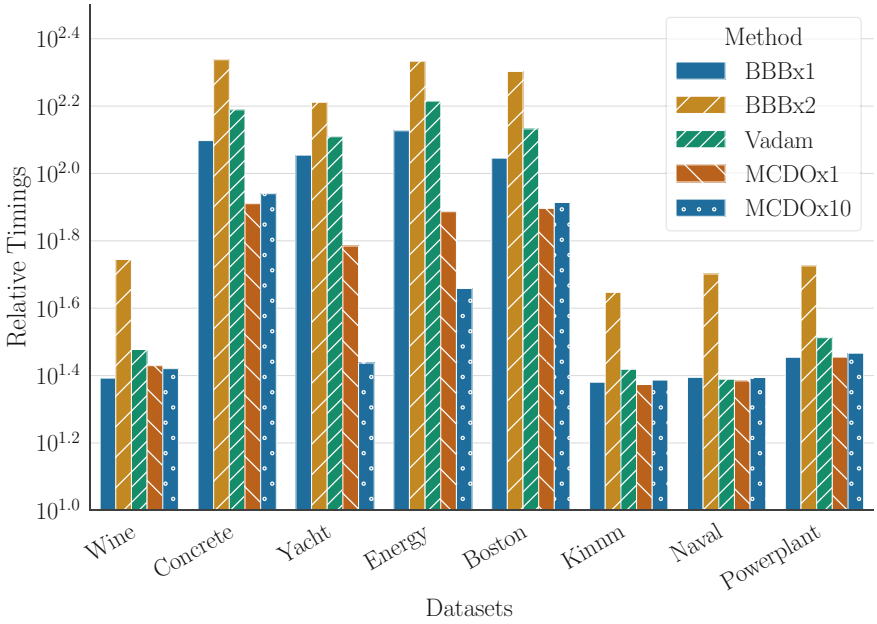



**Fig. 4.15** Amount of time in logarithm scale for training each algorithm relative to PBP. We take into account hyper-parameter search and training of the final model

slower than BBBx1, instead of 80%. Vadam does not have additional parameters for the variances of weights; instead it directly computes them from the intermediate variable used to normalize the directions of the parameter space, something the optimizer Adam already does.

Both MCDO configurations take roughly the same amount of time and have similar performance as Figs. 4.15, 4.16, and 4.17 show. MCDOx1 is 10% slower to train and has slightly better performance than MCDOx10. Even though this difference is small, it is consistent. Overall, MCDOx1 is the best w.r.t both RMSE and log-likelihood, with MCDOx10 being a close second. PBP follows them in third place.

Despite its not so stellar performance, PBP has no need for hyper-parameter search and trains incredibly fast. Another strength PBP possesses, inherited from EP, is being naturally well-suited to data-parallelization across machines, and if using only ADF updates, to online learning.

We summarize the conclusions in Table 4.4 to make future reference easier. It rates the BNN algorithms without any number nor formula w.r.t. three fundamental practical aspects:

- 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.

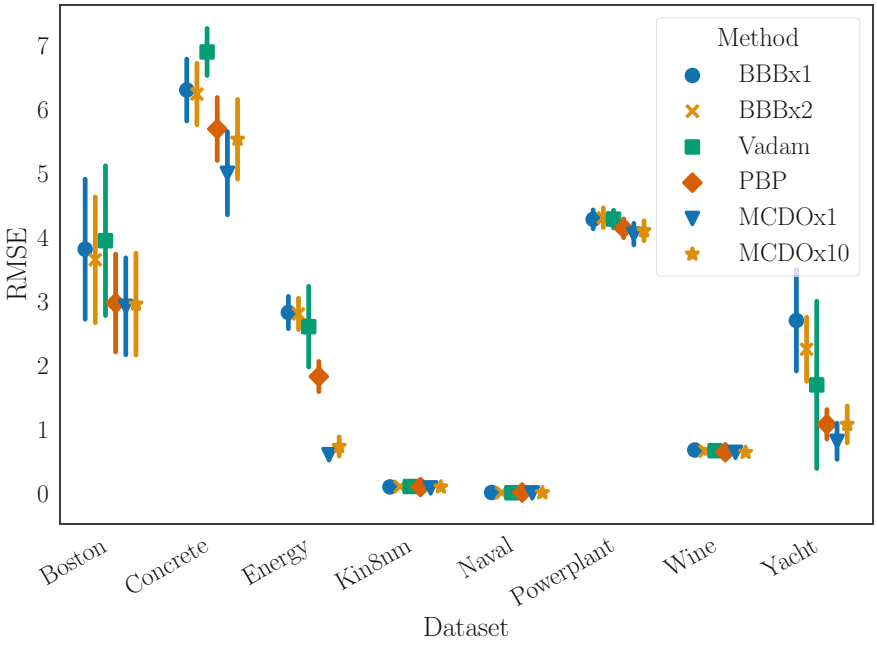



**Fig. 4.16** The average RMSE (low values are better) over the 20 random resampled splits of the UCI regression data sets. Error bars represent the standard deviations over the 20 random splits

On a final note, we leave a general recommendation for those needing to develop a custom solution for a certain task: use Vadam [26], it is fast, out-of-the-box and has reasonable performance. It still needs hyper-parameter tuning, but at this point, almost every algorithm does. If the problem calls for better predictive accuracy or uncertainty estimation, resort to MCDO or other methods not covered here, a few of which are mentioned in Sect. 4.9.

## 4.8 Further References

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

- Matrix variate Gaussian prior [52] and posterior approximation [34];
- Structured covariance with noisy natural gradient [58];
- Low-rank covariance approximation with natural gradient [37].

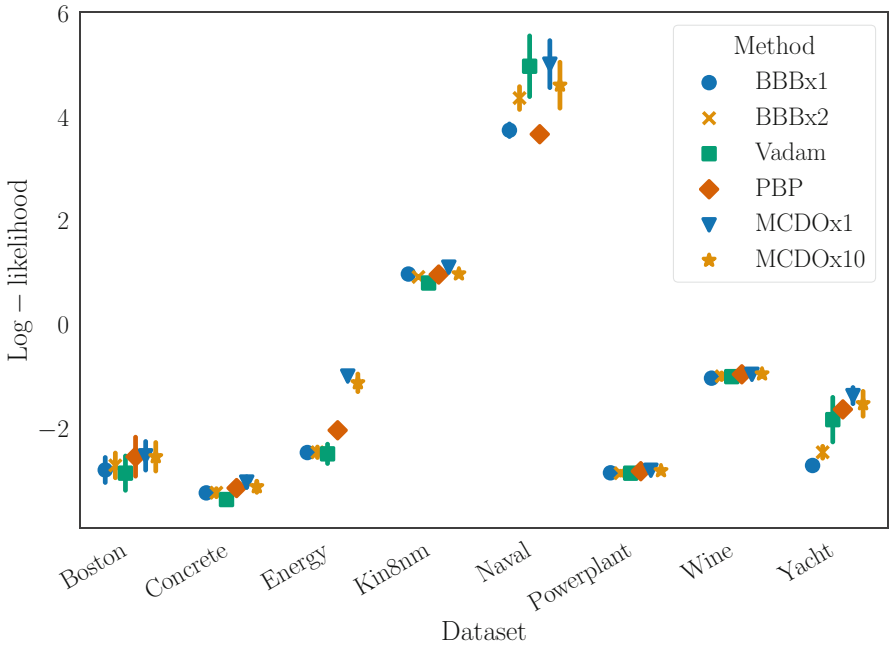



**Fig. 4.17** The average log-likelihood (high values are better) over the 20 random resampled splits of the UCI regression data sets. Error bars represent the standard deviations over the 20 random splits

**Table 4.4** Practitioner's Table: a rough comparison between the variational methods studied for BNNs. Although BBBx2 performance is better than Vadam's, it takes longer to train and a fairer comparison regarding the time would include BBBx1 instead

| Method | Effort | Quality | Timing |
|---|---|---|---|
| BPB | Medium | Poor | Slow |
| PBP | Very hard | Good | Very fast |
| MCDO | Very easy | Good | OK |
| Vadam | None | OK | OK |

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

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

- Hamiltonian MC [41];
- Stochastic gradient Langevin dynamics [32, 57];
- Posterior distribution distillation [29].

## 4.9 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 four key variational algorithms throughout the chapter. Namely, Bayes by Backprop [6], Probabilistic Backprop [19], MC Dropout [13], and Vadam [26]. They all consider unstructured approximations to the posterior. Even though MC Dropout [13] is the only one that does not rely on mean-field approximation, it assumes dependency among groups of weights in a rather non-well-defined manner.

## References

1. Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M, Ghemawat S, Goodfellow I, Harp A, Irving G, Isard M, Jia Y, Jozefowicz R, Kaiser L, Kudlur M, Levenberg J, Mané D, Monga R, Moore S, Murray D, Olah C, Schuster M, Shlens J, Steiner B, Sutskever I, Talwar K, Tucker P, Vanhoucke V, Vasudevan V, Viégas F, Vinyals O, Warden P, Wattenberg M, Wicke M, Yu Y, Zheng X (2015) TensorFlow: large-scale machine learning on heterogeneous systems. Software available from tensorflow.org
2. Amari SI (1998) Natural gradient works efficiently in learning. Neural Comput 10(2):251–276
3. Amari SI (2016) Natural gradient learning and its dynamics in singular regions. Springer Japan, Tokyo, pp 279–314
4. Benatan M, Daresbury ST, Pyzer-Knapp EO (2018) Practical considerations for probabilistic backpropagation. In: Workshop on Bayesian deep learning (NeurIPS 2018), Montreal
5. Benavoli A, Corani G, Demšar J, Zaffalon M (2017) Time for a change: a tutorial for comparing multiple classifiers through Bayesian analysis. J Mach Learn Res 18(77):1–36
6. Blundell C, Cornebise J, Kavukcuoglu K, Wierstra D (2015) Weight uncertainty in neural networks. In: Proceedings of the international conference on machine learning, Lille, vol 37, pp 1613–1622
7. Bonnet G (1964) Transformations des signaux aléatoires a travers les systèmes non linéaires sans mémoire. Annales des Télé Communications 19(9):203–220
8. Bottou L, Curtis FE, Nocedal J (2018) Optimization methods for large-scale machine learning. SIAM Rev 60(2):223–311. https://doi.org/10.1137/16M1080173
9. Corani G, Benavoli A (2015) A Bayesian approach for comparing cross-validated algorithms on multiple data sets. Mach Learn 100(2-3):285–304
10. Dua D, Graff C (2017) UCI machine learning repository. http://archive.ics.uci.edu/ml
11. Duchi J, Hazan E, Singer Y (2011) Adaptive subgradient methods for online learning and stochastic optimization. J Mach Learn Res 12:2121–2159
12. Gal Y (2016) Uncertainty in deep learning. PhD thesis, University of Cambridge
13. Gal Y, Ghahramani Z (2016a) Dropout as a bayesian approximation: representing model uncertainty in deep learning. In: Proceedings of the international conference on machine learning, New York, vol 48, pp 1050–1059
14. Gal Y, Ghahramani Z (2016b) A theoretically grounded application of dropout in recurrent neural networks. In: Advances in neural information processing systems, Barcelona, pp 1019–1027

15. Ghosh S, Fave FD, Yedidia J (2016) Assumed density filtering methods for learning Bayesian neural networks. In: Proceedings of the AAAI conference on artificial intelligence, Phoenix, pp 1589–1595
16. Graves A (2011) Practical variational inference for neural networks. In: Advances in neural information processing systems, Granada, pp 2348–2356
17. Herbrich R (2005) Minimising the Kullback-Leibler divergence. Tech. rep., Microsoft Research
18. Hernandez-Lobato J, Li Y, Rowland M, Bui T, Hernandez-Lobato D, Turner R (2016) Black-box alpha divergence minimization. In: Proceedings of the international conference on machine learning, New York, vol 48, pp 1511–1520
19. Hernández-Lobato JM, Adams RP (2015) Probabilistic backpropagation for scalable learning of bayesian neural networks. In: Proceedings of the international conference on machine learning, Lille, vol 37, pp 1861–1869
20. Hinton GE, van Camp D (1993) Keeping the neural networks simple by minimizing the description length of the weights. In: Annual conference on computational learning theory, COLT '93, Santa Cruz, pp 5–13
21. Hinton GE, Osindero S, Teh YW (2006) A fast learning algorithm for deep belief nets. Neural Comput 18(7):1527–1554
22. Hoffman MD, Blei DM, Wang C, Paisley J (2013) Stochastic variational inference. J Mach Learn Res 14:1303–1347
23. Honkela A, Tornio M, Raiko T, Karhunen J (2007) Natural conjugate gradient in variational inference. In: Proceedings of the international conference on neural information processing, Kitakyushu, pp 305–314
24. Hron J, Matthews A, Ghahramani Z (2018) Variational Bayesian dropout: pitfalls and fixes. In: Proceedings of the international conference on machine learning, Stockholm, vol 80, pp 2019–2028
25. Khan M, Lin W (2017) Conjugate-computation variational inference :converting variational inference in non-conjugate models to inferences in conjugate models. In: International conference on artificial intelligence and statistics, Fort Lauderdale, vol 54, pp 878–887
26. Khan M, Nielsen D, Tangkaratt V, Lin W, Gal Y, Srivastava A (2018) Fast and scalable Bayesian deep learning by weight-perturbation in Adam. In: Proceedings of the international conference on machine learning, Stockholm, vol 80, pp 2611–2620
27. Kingma DP, Ba J (2015) Adam: a method for stochastic optimization. In: Proceedings of the international conference on learning representations, San Diego
28. Kingma DP, Salimans T, Welling M (2015) Variational dropout and the local reparameterization trick. In: Advances in neural information processing systems, Montreal, pp 2575–2583
29. Korattikara A, Rathod V, Murphy K, Welling M (2015) Bayesian dark knowledge. In: Advances in neural information processing systems, Montreal, pp 3438–3446
30. Krizhevsky A, Sutskever I, Hinton GE (2012) ImageNet classification with deep convolutional neural networks. In: Advances in neural information processing systems, Lake Tahoe, pp 1097–1105
31. Kuleshov V, Fenner N, Ermon S (2018) Accurate uncertainties for deep learning using calibrated regression. In: Proceedings of the international conference on machine learning, Stockholm, vol 80, pp 2796–2804
32. Li C, Chen C, Carlson D, Carin L (2016) Preconditioned stochastic gradient Langevin dynamics for deep neural networks. In: Proceedings of the AAAI conference on artificial intelligence, Phoenix, pp 1788–1794
33. Little RJ (2006) Calibrated Bayes: a Bayes/frequentist roadmap. Am Stat 60(3):213–223
34. Louizos C, Welling M (2016) Structured and efficient variational deep learning with matrix Gaussian posteriors. In: Proceedings of the international conference on machine learning, New York, vol 48, pp 1708–1716
35. MacKay DJC (1992) A practical Bayesian framework for backpropagation networks. Neural Comput 4(3):448–472
36. Minka TP (2001) Expectation propagation for approximate Bayesian inference. In: Conference in uncertainty in artificial intelligence, San Francisco, pp 362–369

37. Mishkin A, Kunstner F, Nielsen D, Schmidt M, Khan ME (2018) SLANG: fast structured covariance approximations for Bayesian deep learning with natural gradient. In: Advances in neural information processing systems, Montreal, pp 6245–6255
38. Nair V, Hinton GE (2010) Rectified linear units improve restricted Boltzmann machines. In: Proceedings of the international conference on machine learning, Haifa, pp 807–814
39. Neal RM (1993) Bayesian learning via stochastic dynamics. In: Advances in neural information processing systems, San Francisco, pp 475–482
40. Neal RM (1996) Bayesian learning for neural networks. PhD thesis, University of Toronto, Toronto
41. Neal RM (2011) MCMC using Hamiltonian dynamics. In: Handbook of Markov chain, chap 5. Monte Carlo, CRC Press, Boca Raton, pp 113–162
42. Niculescu-Mizil A, Caruana R (2005) Predicting good probabilities with supervised learning. In: Proceedings of the international conference on machine learning, Bonn, pp 625–632
43. Papernot N, McDaniel P, Goodfellow I, Jha S, Celik ZB, Swami A (2017) Practical black-box attacks against machine learning. In: Proceedings of the ACM Asia conference on computer and communications security, Abu Dhabi, pp 506–519
44. Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, Desmaison A, Kopf A, Yang E, DeVito Z, Raison M, Tejani A, Chilamkurthy S, Steiner B, Fang L, Bai J, Chintala S (2019) PyTorch: an imperative style, high-performance deep learning library. In: Advances in neural information processing systems, Vancouver, pp 8024–8035
45. Platt JC (1999) Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In: Advances in large margin classifiers, pp 61–74
46. Price R (1958) A useful theorem for nonlinear devices having Gaussian inputs. Trans Inf Theor 4(2):69–72
47. Rasmussen CE, Williams CKI (2005) Gaussian processes for machine learning. The MIT Press, Cambridge
48. Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, Huang Z, Karpathy A, Khosla A, Bernstein M, Berg AC, Fei-Fei L (2015) ImageNet large scale visual recognition challenge. Int J Comput Vis 115(3):211–252
49. Snoek J, Larochelle H, Adams RP (2012) Practical Bayesian optimization of machine learning algorithms. In: Advances in neural information processing systems, Lake Tahoe, pp 2951–2959
50. Soudry D, Hubara I, Meir R (2014) Expectation backpropagation: parameter-free training of multilayer neural networks with continuous or discrete weights. In: Advances in neural information processing systems, Montreal, pp 963–971
51. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 15(1):1929–1958
52. Sun S, Chen C, Carin L (2017) Learning structured weight uncertainty in Bayesian neural networks. In: International conference on artificial intelligence and statistics, Fort Lauderdale, vol 54, pp 1283–1292
53. Theano Development Team (2016) Theano: a python framework for fast computation of mathematical expressions. arXiv e-prints 1605.02688
54. Tieleman T, Hinton G (2012) Lecture 6.5-rmsprop: divide the gradient by a running aof its recent magnitude
55. Tishby N, Levin E, Solla SA (1989) Consistent inference of probabilities in layered networks: predictions and generalization. In: International joint conference on neural networks, vol 2, pp 403–409
56. Wan L, Zeiler M, Zhang S, Cun YL, Fergus R (2013) Regularization of neural networks using dropconnect. In: Proceedings of the international conference on machine learning, Atlanta, vol 28, pp 1058–1066
57. Welling M, Teh YW (2011) Bayesian learning via stochastic gradient Langevin dynamics. In: Proceedings of the international conference on machine learning, Bellevue, pp 681–688
58. Zhang G, Sun S, Duvenaud D, Grosse R (2018) Noisy natural gradient as variational inference. In: Proceedings of the international conference on machine learning, Stockholm, vol 80, pp 5852–5861

# Chapter 5
# Variational Autoencoder


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 class 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 characterize generative models,
- understand when they are useful,
- know the challenges on assessing their quality,
- possess breadth of knowledge on VAE's core idea,
- comprehend the ideas behind its extensions and what effectively changes.

## 5.1 Motivations

Generative models are statistical models of data that attempt to capture the entire probability distribution from the observables, that is, to estimate $p(\mathbf{x})$ from $\mathcal{D} = \{\mathcal{X}\}_n$. They are 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 much more.

Differently from discriminative models that use the target $\mathbf{Y}$ as supervisory signal to estimate the conditional distribution of $\mathbf{Y} \mid \mathbf{X}$, generative models do not



L. P. Cinelli et al., *Variational Methods for Machine Learning with Applications to Deep Networks*, https://doi.org/10.1007/978-3-030-70679-1_5

need to be supervised. Thus, the latter naturally fits unsupervised algorithms, which use unlabeled data, an abundant resource. Nonetheless, generative models can also perform discriminative tasks, such as classification, by modeling the joint distribution $p(\mathbf{y}, \mathbf{x})$. The models of Chap. 4 are examples of discriminative models, as they estimate the conditional $p(\mathbf{y} \mid \mathbf{X})$: given some inputs, they output a distribution of probable values for the unknown quantity. However, we cannot infer information such as the most probable input–output pairs or the expected input values for a given observed output, which would require the joint $p(\mathbf{y}, \mathbf{x})$.

Knowledge of the complete probabilistic model means we can simulate how the world evolves [20, 29], anticipate and plan for the future [14, 18], reason out and make decisions [11, 14], and understand elements and their factors of variation [10, 11], among other high-level abstract tasks. Exciting applications that have direct application in industry are image super-resolution [28], compression [55], denoising [8], and audio synthesis [57]. Other examples outside the multimedia domain and arguably even more compelling are in chemistry, for efficient exploration of new compounds [13], biology, for prediction of the effects of mutations in proteins and RNAs [46], and astronomy, for images of optical telescope [44].

A classical example of a generative model is the Naive Bayes that constructs the joint probability for classification. In this chapter, the focus is on modern methods that use approximate inference. Specifically, we discuss the family of VAE models, theirs issues, and advantages.

## 5.2 Evaluating Generative Networks

There is no universal metric for measuring the performance of a generative model, thus assessing its quality is not straightforward and often misleading. Models trained by optimizing the same criteria relate well across different properties and can be directly compared, i.e., the training criteria can be used as an objective metric for model selection. However, for different objective functions, model comparison becomes problematic. The higher the dimension of the space, the less correlated the metrics become [53].

There are many different properties we may wish for in a model: high quality samples, diverse samples, compact representations, useful representation, interpretable representations, and so on. Naturally, we cannot have all at the same time, so we must compromise.

A model may have a high log-likelihood but an average matching behavior, causing it to assign probability mass to low-density regions. Although the approximation may be overall good, unusual regions will end up getting too much attention, at least more than it normally should. Consequently, generated samples will be more likely to be very distinct from samples from the true distribution in spite of the approximation's good log-likelihood. Although it may seem as an undesirable property at first, it really depends on the context. For drug discovery, it can be great

because it means we are exploring other regions of the space, where we can find new useful compounds. For images, it can be a disaster because the resulting image can be amorphous and weird or represent nothing at all. Still, how many abstract, odd artworks end up being highly valuable and praised?

On the other hand, a model with moment-matching behavior produces realistic samples but has poorer log-likelihood for multi-modal distributions. The approximation fits a single mode, assuming a unimodal model, and misses out on a great deal of information. Hence, the resulting sample diversity will also be low. Images are a good example of highly multi-modal distributions over space: there is dependence between neighboring pixels and groups of pixels (superpixels), and on the scene as a whole, besides a lot of structure built-in from the physical world. A model fitted to a single mode would then struggle to mix different factors of variation in image elements.

Besides log-likelihood and sample quality communicating different things, they do not inform us about sample diversity though they are connected. If samples seem to come from the true distribution and, at the same time, have great diversity, then probably the two densities match well overall, which implies good average log-likelihood for the model. Sample diversity relates to the entropy of the distribution.

Average log-likelihood has become the de facto standard measure of quality for generative models. However, the computation may not be possible, at least in a viable amount of time, depending on the model type. For such models, where a direct measure of the log-likelihood is not possible, it is common to resort to Parzen window estimates [53]. The Parzen window method is a nonparametric approach that consists in drawing samples from the original model and constructing a tractable approximate model, for which we can compute the average log-likelihood. Nevertheless, it has been experimentally demonstrated that evaluation with Parzen window estimates does not correlate well with likelihood nor sample quality, hence being strongly discouraged [53].

Visual sample inspection is the natural metric for assessing sample quality, and it is the go-to metric when dealing with image synthesis, as it allows us to find out in an intuitive manner about what is happening into the model. Still, this approach is only effective for audiovisual samples. Samples represent an interesting diagnostic tool but should not be used as a proxy for other tasks [53].

Perceptual quality metrics do not take into account generalization capacity. A model that simply outputs training examples without ever producing a new one would score high but would be of no use. Trivial algorithms such as nearest neighbors for detecting whether generated samples are similar to training ones are not effective: perceptually similar images can have large distances, e.g., a one-pixel shift of a texture-rich image. Additionally, overfit models do not necessarily reproduce the images from the data set [53].

A different approach for assessing the model's performance is to measure it directly on a surrogate task, what is especially 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 for. For example, one could use a small capacity 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 these aspects and allow to objectively compare between competing algorithms. For images, the Inception Score [48] and the Frechet Inception Distance [16] are two popular metrics that use pretrained classification models to compare the generated samples with a hold-out test set. The former measures sharpness and diversity, whereas the latter measures similarities in the feature representation space. Both empirically correlate well with the perceived quality of samples. However, the Inception Score is only valid when the classifier used for the evaluation was pretrained on the same data set as the generative model under evaluation [47].

In summary, there is no universal metric with sample quality, classification accuracy, and log-likelihood being largely independent properties in high-dimensional spaces. Thus, proper assessment of the model's performance depends on the application: different applications require different metrics, e.g., sample quality for content generation, downstream task for representation learning, and log-likelihood for compression and density estimation.

## 5.3 Variational Autoencoders

We begin this section by posing a modeling problem and steadily progress toward constructing the KL. We start from the set of observations $\mathcal{D} = \{\mathcal{X}\}_{n=1}^{N}$ and latent variable models, whose value we already discussed in Sect. 2.1.3. Thus, we have $p(\mathbf{x}) = \int p(\mathbf{x}, \mathbf{z}) d\mathbf{z}$, where $\mathbf{Z}$ is the unknown latent variables which we assume to be the hidden causes for the observed $\mathbf{x}$, and $p(\mathbf{x}, \mathbf{z})$ is the generative model of Fig. 5.1a.

Although KLs can in principle work with any kind of data, we use it on images to illustrate our points and keep the discussion attractive. Hence, $\mathbf{x}$ corresponds to an image sample. This type of data has great appeal, is intuitive, and has broad support on modern programming frameworks, i.e., Pytorch [41].

We can rewrite the integral over the joint distribution according to the chain rule for conditional probability, which gives us $p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x} \mid \mathbf{Z}) p(\mathbf{z})$, where $p(\mathbf{z})$ is the prior distribution over the latent space and $p(\mathbf{x} \mid \mathbf{Z})$ the likelihood function. For all but very simple models, the integral $\int p(\mathbf{x} \mid \mathbf{Z}) p(\mathbf{z}) d\mathbf{z}$ is intractable and cannot be analytically calculated; thus, we estimate it by Monte Carlo (MC) sampling with $T$ samples, such as

$$p(\mathbf{x}) = \int p(\mathbf{x}, \mathbf{z}) d\mathbf{z} = \int p(\mathbf{x} \mid \mathbf{Z}) p(\mathbf{z}) d\mathbf{z} \approx \frac{1}{T} \sum_{i=1}^{T} p(\mathbf{x} \mid \mathbf{Z}^{(i)}), \tag{5.1}$$

with $\mathbf{z} \sim p(\mathbf{z})$. However, finding samples of $\mathbf{z}$ for which $p(\mathbf{x} \mid \mathbf{Z})$ is large is challenging in high-dimensional latent spaces: we need millions of draws to obtain

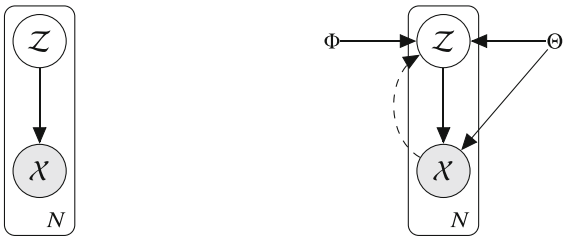



**Fig. 5.1** Graphical representations of the generative model $p(\mathbf{x}, \mathbf{z})$. Figure 5.1a is the initial model we formulate: the latent variables $\mathbf{z}_i$ are the hidden cause behind the sample $\mathbf{x}_i$. Figure 5.1b depicts the parameterized model where we assume the parameters responsible for generating the data to be $\Theta$ and the posterior approximation to be determined by the parameters $\Phi$. The dashed line represents the inference process from which we determine the posterior approximation $q(\mathbf{z} \mid \mathbf{X}; \Phi)$ from the samples $\mathbf{x}$. (**a**) The latent variable model. (**b**) The parameterized model

reasonable estimates for $p(\mathbf{x})$. Then, how to choose $p(\mathbf{z})$ such that we obtain plausible values of $\mathbf{z}$, for which $p(\mathbf{x} \mid \mathbf{Z})$ is high, with high probability?

Once more, we rewrite the problem as

$$\begin{aligned} p(\mathbf{x}) &= \int p(\mathbf{x} \mid \mathbf{Z}) p(\mathbf{z}) d\mathbf{z} \\ &= \int p(\mathbf{x} \mid \mathbf{Z}) p(\mathbf{z}) \frac{q(\mathbf{z} \mid \mathbf{X})}{q(\mathbf{z} \mid \mathbf{X})} d\mathbf{z} \\ &= \mathbb{E}_q \left[ \frac{p(\mathbf{x} \mid \mathbf{Z}) p(\mathbf{z})}{q(\mathbf{z} \mid \mathbf{X})} \right] \\ &\approx \frac{1}{T} \sum_{i=1}^{T} \frac{p(\mathbf{x} \mid \mathbf{Z}^{(i)}) p(\mathbf{z}^{(i)})}{q(\mathbf{z}^{(i)} \mid \mathbf{X})}, \end{aligned} \tag{5.2}$$

with $\mathbf{z} \sim q(\mathbf{z} \mid \mathbf{X})$, and approximate the integral with an unbiased MC estimate of $T$ samples.

Under this new perspective, the sampling process occurs according to the proposal distribution $q(\mathbf{z} \mid \mathbf{X})$, and to obtain the same result as before, we need to properly weight the values of $p(\mathbf{x} \mid \mathbf{Z})$ by $p(\mathbf{z})/q(\mathbf{z} \mid \mathbf{X})$. The problem has now become finding suitable candidates for $p(\mathbf{z})$ and $q(\mathbf{z} \mid \mathbf{X})$.

The approach in (5.2) corresponds to Importance Sampling (IS) [35]. 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. The optimal proposal distribution $q^*(\mathbf{z} \mid \mathbf{X})$ is

$$q^*(\mathbf{z} \mid \mathbf{X}) = \frac{p(\mathbf{x} \mid \mathbf{Z}) p(\mathbf{z})}{p(\mathbf{x})} = p(\mathbf{z} \mid \mathbf{X}), \tag{5.3}$$

for which we obtain for the single-sample estimator $\hat{p}_{T=1}(\mathbf{x})$ the true distribution we seek, that is,

$$\hat{p}_{T=1}(\mathbf{x}) = \frac{p(\mathbf{x} \mid \mathbf{Z}^{(1)})p(\mathbf{z}^{(1)})}{q(\mathbf{z}^{(1)} \mid \mathbf{X})} = \frac{p(\mathbf{x} \mid \mathbf{Z}^{(1)})p(\mathbf{z}^{(1)})}{\frac{p(\mathbf{x} \mid \mathbf{Z}^{(1)})p(\mathbf{z}^{(1)})}{p(\mathbf{x})}} = p(\mathbf{x}). \tag{5.4}$$

Yet, the inability to compute $p(\mathbf{x}) = \int p(\mathbf{x} \mid \mathbf{Z})p(\mathbf{z})d\mathbf{z}$ was the very reason that motivated us to search for other solutions. We shall parameterize the distributions as $p(\mathbf{x} \mid \mathbf{Z}; \Theta)$ and $q(\mathbf{z} \mid \mathbf{X}; \Phi)$ and jointly optimize for $\Theta$ and $\Phi$. The corresponding Probabilistic Graphical Model (PGM), the concept introduced in Sect. 3.1.1, is represented in Fig. 5.1b. While the posterior distribution $q(\mathbf{z} \mid \mathbf{X}; \Phi)$ allows inferring latent distributions relating to the observables, the likelihood function $p(\mathbf{x} \mid \mathbf{Z}; \Theta)$ enables the generation of new samples when paired with the prior, what effectively means sampling from the joint distribution.

We implement the likelihood model as a neural network (NN) and fit its parameters, $\Theta$, by Maximum Likelihood Estimator (MLE), instead of variational Bayesian inference, though possible. For the variational parameters $\Phi$, we would need to compute local estimates for each sample $\mathbf{x}_i \in \mathcal{D}$ [3]. Besides not scaling well, it implies computing new variational parameters for each test data before estimating the posterior distribution over the latent variables.

Instead, we optimize a separate model, called recognition model, to output the local variational parameters $\Phi$ that define the posterior distribution $q(\mathbf{z} \mid \mathbf{X}; \Phi)$. Hence, each new data point $\mathbf{x}'$ goes through a function $f(\mathbf{x}'; \Psi) \mapsto \Phi$. The problem becomes solving for the global variational parameters $\Psi$ that define the mapping $f(\cdot; \Psi)$. We shall also use an NN for the recognition model. The approach of sharing the variational parameters across data points is called *amortized inference* and is common in settings with large data sets because it effectively amortizes the inference cost, allowing for faster training and testing.

We immediately note that the optimal value for $\Phi$ is such that $q(\mathbf{z} \mid \mathbf{X}; \Phi) = p(\mathbf{z} \mid \mathbf{X}; \Theta)$ or at least as close to it as possible. We have arrived at a familiar framework in which we wish to maximize the evidence $p(\mathbf{x}; \Theta)$, and for this, we do

$$\max_{\Theta} p(\mathbf{x}; \Theta) = \max_{\Theta} \log p(\mathbf{x}; \Theta) = \max_{\Theta, \Phi} \log \mathbb{E}_q \left[ \frac{p(\mathbf{x} \mid \mathbf{Z}; \Theta)p(\mathbf{z})}{q(\mathbf{z} \mid \mathbf{X}; \Phi)} \right]. \tag{5.5}$$

Applying Jensen's inequality exactly as we did in Sect. 3.2.1.1 leads us once again to the ELBO objective (3.7, 3.8) as we verify in

$$\log \mathbb{E}_q \left[ \frac{p(\mathbf{x} \mid \mathbf{Z}; \Theta)p(\mathbf{z})}{q(\mathbf{z} \mid \mathbf{X}; \Phi)} \right] \geqslant \mathbb{E}_q \left[ \log \frac{p(\mathbf{x} \mid \mathbf{Z}; \Theta)p(\mathbf{z})}{q(\mathbf{z} \mid \mathbf{X}; \Phi)} \right] = \text{ELBO}(\Theta, \Phi). \tag{5.6}$$

Even though the final utility function has a similar form to those seen in Chap. 4, there are some subtle but important differences. Here, we perform

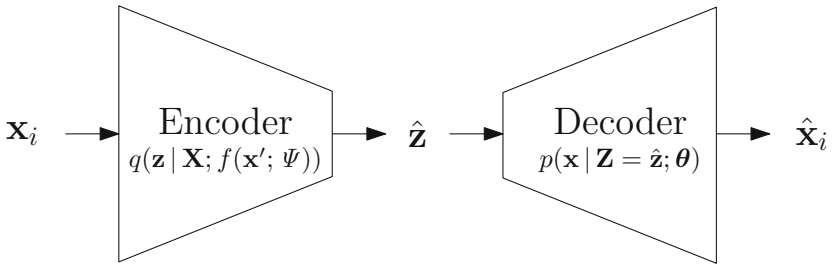



**Fig. 5.2** Schematic of the KL model. The image $\mathbf{x}_i$ gets mapped, thanks to the encoder NN, to a distribution over the latent variable $\mathbf{Z}$, from which we draw a sample $\hat{\mathbf{z}}$ (one-sample MC estimator). Next, we pass the sample through the likelihood model NN to obtain the distribution $p(\mathbf{x} \mid \mathbf{Z} = \hat{\mathbf{z}}; \theta)$, whose most probable values should be the $\mathbf{x}_i$ that generated the latent sample $\hat{\mathbf{z}}$

Variational Inference (VI), on the latent variables $\mathbf{Z}$, but point estimation on the variables $\Theta$ of the likelihood model. In Chap. 4, we performed VI on all variables, which were global, since they were the same for all data points, and there were no local latent variables. A brief treatment on the Full Variational Bayes version can be found in [23].

Note that the target density $p(\mathbf{x} \mid \mathbf{Z}; \Theta)$ changes over the course of training, not being static as the target densities of Chap. 4. Hence, $q(\mathbf{z} \mid \mathbf{X}; f(\mathbf{x}'; \Psi))$ must track this evolution so that the approximation remains "close" to the true (modeled) distribution.

From the complete model developed so far, shown in Fig. 5.2, we observe that the distribution over the latent space $\mathcal{Z}$ is in between the recognition and the likelihood models, creating an information bottleneck if $dim(\mathcal{Z}) < dim(\mathcal{X})$ [1]. Generally, this is the case since we assume that 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 $\mathbf{x}$ 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, $q(\mathbf{z} \mid \mathbf{X}; \Psi)$ can be understood as a probabilistic encoder and $p(\mathbf{x} \mid \mathbf{Z}; \Theta)$ as a probabilistic decoder. Indeed, if we write the ELBO for the data set $\mathcal{D}$ with $N$ samples in its most usual form, we have

$$\text{ELBO}(\Theta, \Psi) = \sum_{n=1}^{N} \mathbb{E}_q \left[\log p(\mathbf{x}_n \mid \mathbf{Z}_n; \Theta)\right] - D_{KL}\left(q(\mathbf{z}_n \mid \mathbf{X}_n; \Psi) \| p(\mathbf{z})\right). \tag{5.7}$$

The first term in (5.7) aims at maximizing the likelihood of the reconstructed samples. The second term, on the other hand, works as a regularization factor imposing structure to the latent space. It must be structured so that the conditional on $\mathbf{x}_i$ is as similar as possible to the prior. Deviations from the prior should be meaningful enough, so the decoder can achieve a better reconstruction and pay off the toll imposed by the Kullback-Leibler (KL). Without the KL term, (5.7) would

correspond to MLE maximization and the latent distribution would degenerate to a point estimate. This would entail a conventional autoencoder that deterministically maps a data point $\mathbf{x}_i$ to $\mathbf{z}_i$ and deterministically reconstructs it. Consequently, nearby latent points would not necessarily represent similar data points, just as in a lookup table. Thus, the latent space would not have a significant structure. The autoencoder with the KL regularization term in the latent space receives the name of *Variational Autoencoder* [23].

From the information bottleneck perspective [54], we can see the reconstruction error as a measure of distortion and the posterior misalignment as the communication rate between the prior and posterior distributions [1, 2]. Indeed, in information theory, the $D_{KL}\left(q \| p\right)$ can be interpreted as the extra number of bits required to send a message under $q$ with a coding scheme that was optimally designed for the $p$. When the KL is zero, we have $q(\mathbf{z} \,|\, \mathbf{D}; \Psi) = p(\mathbf{z})$, and there is no information about the input $\mathbf{x}$ flowing through the model, meaning that the latent channels have zero capacity. A larger overlap between the distributions corresponds to a less informative posterior, with respect to the input $x_i$, and a higher reconstruction error (distortion). This constraint forces similar data points to have similar posterior distributions, imposing smoothness and locality to the latent space.

The latent distribution in between the encoder and decoder raises difficulty when trying to use gradient descent to optimize the model. We cannot numerically compute the gradient of an expectation w.r.t. its distribution; see Appendix A.1 for an in-depth discussion. However, for continuous latent distributions and differentiable likelihood models, we resort to the pathwise gradient estimator (Appendix A.1), more commonly known as the reparameterization trick [23], which after $T$ MC samples leads to

$$\widehat{\text{ELBO}}_1(\Theta, \Psi) = \sum_{n=1}^{N} \frac{1}{T} \sum_{i=1}^{T} \Big[ \log p(\mathbf{x}_n, \mathbf{z}_n^{(i)}; \Theta) - \log q(\mathbf{z}_n^{(i)} \,|\, \mathbf{X}_n; \Psi) \Big], \tag{5.8}$$

where $\mathbf{z}_n^{(i)} = g(\epsilon^{(i)}, \mathbf{x}_n; \Psi)$ is a deterministic transformation and $\epsilon^{(i)}$ the $i$-th sample from the base distribution $p(\epsilon)$. The above estimator is equivalent to (4.8), put forth in [4] for BNNs.

By choosing families of distributions for $p(\mathbf{z})$ and for $q(\mathbf{z} \,|\, \mathbf{X}; \Psi)$ such that the KL term in (5.7) has closed-form analytical formula, we rewrite the estimator in (5.8) as

$$\widehat{\text{ELBO}}_2(\Theta, \Psi) = \sum_{n=1}^{N} \left[ \frac{1}{T} \left[ \sum_{i=1}^{T} \log p(\mathbf{x}_n \,|\, \mathbf{Z}_n^{(i)}; \Theta) \right] - D_{KL}\left(q(\mathbf{z}_n \,|\, \mathbf{X}_n; \Psi) \| p(\mathbf{z})\right) \right], \tag{5.9}$$

which is the form used throughout Chap. 4. Figure 5.3 illustrates the reparameterization trick applied to a random node $\mathbf{z}$ in the computational graph with the KL divergence being analytically calculated.

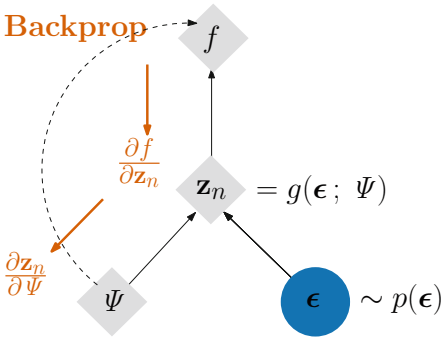



**Fig. 5.3** Computational graph after the reparameterization trick. The blue round node is a random node, while the gray rhombus nodes are deterministic. Black arrows represent the forward pass of the model and the red ones the backpropagation path. The black dashed line indicates the path for the computation of the KL divergence, which takes the distribution parameters $\Psi$ as input. Note that, thanks to the reparameterization trick, the node $\mathbf{z}_n$ is no longer random, and so we can directly compute its gradient

The main technical contribution of [23] was introducing for the first time, in 2013, to the Deep Learning (DL) community the reparameterization trick to obtain a low-variance gradient estimator. The widely used KL model is simply a use case example of this estimator the authors give midway through the paper [23]. The name for the generic formulation of Fig. 5.1a optimized with (5.8) is Autoencoding Variational Bayes (AEVB). Here we consider the most common instantiation of VAE: $q(\mathbf{z} \,|\, \mathbf{X}; \Psi)$ and $p(\mathbf{x} \,|\, \mathbf{Z}; \Theta)$ both implemented with feedforward NNs. Still, we could implement the same general model with other blocks, such as autoregressive models.

Later, in Sect. 5.6, we will work with binary images. Consequently, we describe the pixels as realizations of independent Bernoulli distributions and interpret each output pixel of the generative NN as an estimate of Bernoulli's parameter $p$ in the original sample.

In what follows, we suppose the input to be real-valued images, using $\mathcal{N}(\mu_i, \sigma_i^2)$ as the likelihood function $p(\mathbf{x}_i \,|\, \mathbf{Z}; \Theta)$, and a centered diagonal $\mathcal{N}(\mathbf{0}, \mathbf{I})$ for the a priori $p(\mathbf{z})$. Thus the KL term in (5.7) simplifies to

$$D_{KL}\left(q(\mathbf{z}_n \,|\, \mathbf{X}_n; \Psi) \| p(\mathbf{z})\right) = \sum_i^{|\mathbf{z}|} \log \frac{1}{\sigma_i} + \frac{1}{2}\left(\mu_i^2 + \sigma_i^2 - 1\right). \tag{5.10}$$

In addition, approximate the posterior $q(\mathbf{z}_i \,|\, \mathbf{X}_i; \Psi)$ with a Gaussian distribution with diagonal covariance matrix. These choices for the distributions are not at all due to restrictions in the algorithm, but rather motivated by their simplicity. The deterministic transformation $g(\boldsymbol{\epsilon}; \mathbf{x}_n, \Psi)$ is then

$$g(\boldsymbol{\epsilon}; \mathbf{x}, \Psi) = \mu(f(\mathbf{x}; \Psi)) + \sigma(f(\mathbf{x}; \Psi)) \odot \boldsymbol{\epsilon}, \tag{5.11}$$

with $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, $f(\cdot; \Psi)$ is the recognition model and $\odot$ the operator for elementwise multiplication.

Although we employ the same transformation all the time, it does not mean that it is the only possible one, and 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 to specify $g$ as the inverse Cumulative Density Function (CDF) of the desired distribution and $\boldsymbol{\epsilon} \sim \mathcal{U}(\mathbf{0}, \mathbf{1})$. While we can use full covariance Gaussian posterior, the optimization problem becomes considerably harder with $O(K^2)$ parameters instead of $O(K)$, where $K$ is the number of dimensions in the latent space, besides needing to ensure that the covariance matrix is positive semidefinite.

Experimentally, the authors [23] verified that when using mini-batch optimization with size $M$, one sample from the approximate posterior $\mathbf{z}^{(1)} \sim q(\mathbf{z} \,|\, \mathbf{X}; \Theta)$ is enough as long as $M$ is large enough, e.g., 100. Nevertheless, it has become common for practitioners to use one sample even when the mini-batch size is not that large because of the computational gains.

We summarize the (vanilla) VAE algorithm at high level with an arbitrary base distribution $p(\epsilon)$ in Algorithm 6.

**Algorithm 6:** VAE (or more generally, AEVB algorithm)

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

### 5.3.1 Conditional VAE

The vanilla KL 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 coloring a person's hair in a photograph prior to actually dyeing it. A question then arises on how to endow the model to create targeted samples rather than completely random ones.

What we really wish is to *condition* the model output on some kind of information $\mathbf{Y}$, hence the name Conditional VAE (CVAE) [50]. The aim becomes to maximize $\log p(\mathbf{x}_i \,|\, \mathbf{Y}_i)$ for each observed variable $\mathbf{x}_i$. Following the same derivation as in (5.6) and (5.7), we arrive at

$$\begin{aligned} \log p(\mathbf{x}_i \,|\, \mathbf{Y}_i; \Theta) &\geqslant \mathbb{E}_{q(\mathbf{z} \,|\, \mathbf{X}_i, \mathbf{Y}_i; \Psi)} \left[ p(\mathbf{x}_i, \mathbf{z} \,|\, \mathbf{Y}_i; \Theta) - q(\mathbf{z} \,|\, \mathbf{X}_i, \mathbf{Y}_i; \Psi) \right] \\ &= \mathbb{E}_{q(\mathbf{z} \,|\, \mathbf{X}_i, \mathbf{Y}_i; \Psi)} \left[ \log p(\mathbf{x}_i \,|\, \mathbf{Z}, \mathbf{Y}_i; \Theta) \right] \\ &\quad - D_{KL} \left( q(\mathbf{z} \,|\, \mathbf{X}_i, \mathbf{Y}_i; \Psi) \| p(\mathbf{z} \,|\, \mathbf{X}_i, \mathbf{Y}_i; \Theta) \right). \end{aligned} \tag{5.12}$$

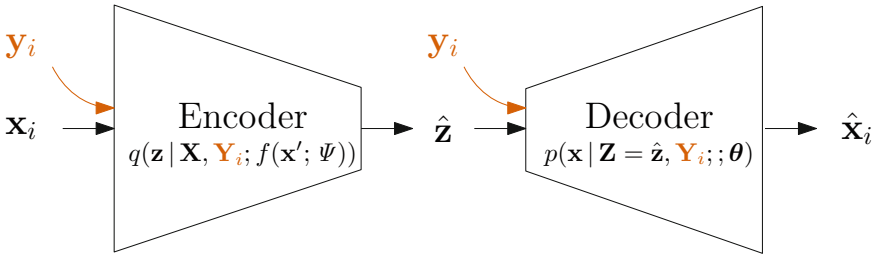



**Fig. 5.4** Schematic illustration of the CVAE model. Note that this is basically the same as the KL, the sole difference being the inclusion of additional conditioning information **Y** to the input **x** and the sampled latent variable **z**. The former so that the recognition model can infer the distribution corresponding to that condition, and the latter so that the generation network also knows to which condition that distribution refers

Recalling that for the KL, the inference model $q(\mathbf{z}_i \mid \mathbf{X}_i; \Psi)$ has input $\mathbf{x}_i$ and output $\mathbf{z}_i$, and it becomes straightforward that for the conditional counterpart $q(\mathbf{z}_i \mid \mathbf{X}_i, \mathbf{Y}_i; \Psi)$, we must just add $\mathbf{y}_i$ as input. One possible way is to concatenate the condition $\mathbf{y}_i$ at the end of $\mathbf{x}_i$ before passing through the inference model. Similar reasoning applies to the generator model. Thus, by changing nothing more than the input to the models, we obtain a CVAE [50].

From an implementation perspective, we can encode category information as a one-hot representation, indicating to the model which class is at the input (or latent code). 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 modeling capacity and sample quality of CVAEs (Fig. 5.4).

### *5.3.2 β-VAE*

As seen from (5.7), optimizing the ELBO is a compromise between the reconstruction quality and posterior alignment with the prior. Depending on the application, we might want to prioritize either realistic high-quality samples or rich latent representations or even something in between. However, the formulation (5.7) offers no control over the individual objectives. Hence, higher ELBOs do not imply better learned representations. Thus, the ELBO is not a suitable objective function for representation learning [2].

The dilemma of representation size and fidelity is well established in rate-distortion theory, already discussed in Sect. . Ideally, we want the model to find a solution along the Pareto front of the distortion-rate plane, which corresponds to the set of minimal solutions. In the $\beta$-KL context, we cast fidelity as the log-likelihood of the model output and connect the concept of rate to the divergence

between the posterior and prior distributions because the misalignment creates the need for sending extra bits through the channel to correct the posterior samples inaccurately coded by the prior.

As we have shown, writing the Lagrangian of the equivalent maximization problem leads us to

$$\mathcal{F}(\theta, \phi, \beta) = \mathbb{E}_q \left[\log p(\mathbf{x} \mid \mathbf{Z}; \Theta)\right] - \beta D_{KL}\left(q(\mathbf{z} \mid \mathbf{X}; \Psi) \| p(\mathbf{z})\right), \tag{5.13}$$

where $\beta$ is the Lagrangian multiplier.

Equation (5.13) is the objective function of the $\beta$-KL algorithm [17], and it is equal to (3.15) after taking the whole data distribution into account. Large values of $\beta$, i.e., $\beta > 1$, give higher weight to the posterior misalignment, forcing a lower corresponding $\delta$ for the rate $R$, thus limiting the representation capacity of the latent space. Hence, the data locality property is further encouraged and, consequently, so does the alignment of independent generative factors of variation along separate latent dimensions [7]. The latent representations become better disentangled. The modeled diagonal structure of the covariance matrix of the posterior distribution also contributes to aligning the factors of variation with the axes.

Due to the fidelity and compression trade-off, log-likelihood is not suited for assessing the quality of the representations learned. However, we can use a linear classifier with low capacity to predict labels from the latent space, $p(\mathbf{y} \mid \mathbf{Z})$. If the classifier achieves high accuracy values, it means that the latent space is linearly separable and easily decoded, hence disentangled [17].

## 5.4 Importance Weighted Autoencoder

As we discussed so far, the VAE objective heavily penalizes approximate posterior samples that fail to explain the data. The log-likelihood term in (5.7) must be high enough to be worth the misalignment penalty imposed by the KL regularization. The VAE criterion may be too strict and limit the model flexibility. If we relax this constraint and become more lenient with samples that are unlikely to explain the observations, we obtain more flexibility on the generative model.

We replace the expectation over the likelihood ratio $p(\mathbf{x}, \mathbf{z}; \Theta)/q(\mathbf{z} \mid \mathbf{X}; \Psi)$ in (5.5) with the expectation over its estimator. Thus, we have the expectation of the importance sampler of (5.2), like

$$\log p(\mathbf{x}) = \log \mathbb{E}_q \left[\frac{1}{T}\sum_{i=1}^{T} \frac{p(\mathbf{x}, \mathbf{z}; \Theta)}{q(\mathbf{z} \mid \mathbf{X}; \Psi)}\right] \geq \mathbb{E}_q \left[\log \frac{1}{T}\sum_{i=1}^{T} \frac{p(\mathbf{x}, \mathbf{z}; \Theta)}{q(\mathbf{z} \mid \mathbf{X}; \Psi)}\right] = \text{ELBO}_{\text{IS}}\,, \tag{5.14}$$

where once again we have applied Jensen's inequality, and $T$ is the number of drawn samples.

We immediately note that (5.14) is equal to (5.7) when $T = 1$, so it can be understood as a generalization of the latter under the point of view of importance sampling. By taking multiple samples, we get progressively tighter bounds and lower variance [6]. Actually, both bias and variance of the Importance Weighted Autoencoder (IWAE) estimator are reduced at a rate $O(1/T)$, leading to a consistent but biased estimate of the true likelihood $p(\mathbf{x})$ [39]. Hence, the gradient of $\text{ELBO}_{\text{IS}}$ points toward better directions as $T$ increases.

The update rule for IWAE is the average over the samples with weights proportional to the importance weights $\mathbf{w}_i = p(\mathbf{x}, \mathbf{z}; \Theta)/q(\mathbf{z} \mid \mathbf{X}; \Psi)$, as shown by

$$\begin{aligned}
\nabla_{\Theta} \text{ELBO}_{\text{IS}} &= \nabla_{\Psi,\Theta} \mathbb{E}_q \left[ \log \frac{1}{T} \sum_{i=1}^{T} \frac{p(\mathbf{x}, \mathbf{z}; \Theta)}{q(\mathbf{z} \mid \mathbf{X}; \Psi)} \right] \\
&= \mathbb{E}_{p(\boldsymbol{\epsilon})} \left[ \nabla_{\Psi,\Theta} \log \frac{1}{T} \sum_{i=1}^{T} \frac{p(\mathbf{x}, g(\boldsymbol{\epsilon}, \mathbf{x}_n; \Psi); \Theta)}{g(\boldsymbol{\epsilon}, \mathbf{x}_n; \Psi) \mid \mathbf{X}; \Psi)} \right] \\
&= \mathbb{E}_{p(\boldsymbol{\epsilon})} \left[ \nabla_{\Psi,\Theta} \log \frac{1}{T} \sum_{i=1}^{T} w_i \right] \\
&= \mathbb{E}_{p(\boldsymbol{\epsilon})} \left[ \nabla_{\Psi,\Theta} \sum_{i=1}^{T} \tilde{w}_i \log w_i \right],
\end{aligned} \tag{5.15}$$

where we have used the reparameterization trick, $\mathbf{z}_n = g(\boldsymbol{\epsilon}, \mathbf{x}_n; \Psi)$ with $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$, to move the gradient inside the expectation, and $\tilde{\mathbf{w}}_i$ are the normalized importance weights $\tilde{\mathbf{w}}_i = \mathbf{w}_i / \sum_{i=1}^{T} \mathbf{w}_i$.

Respective to the log-importance weights in (5.15), the normalized weights $\tilde{\mathbf{w}}_i$ can be seen as their softmax version,

$$l_i = \log \mathbf{w}_i \rightarrow \tilde{\mathbf{w}}_i = \frac{e^{l_i}}{\sum_{j=1}^{T} e^{l_j}} = \text{softmax}(\mathbf{l})_i. \tag{5.16}$$

The importance weights then prioritize the sample with the highest log-likelihood ratio, the one that best explains the data. Since the samples with low likelihood are given less importance, the penalty they impute is attenuated, and the approximate posterior is given more liberty with respect to the VAE constraints, allowing it to spread over the modes of the true posterior and become a better approximation.

However, it is important to keep in mind that the model was optimized to have better performance when drawing multiple samples from the posterior. Thus, one cannot expect IWAEs to have good performance for tasks in which single samples will be used. Training using IWAE will result in models where each individual sample from the model will often have a low (standard) ELBO that is why IWAEs might not be a good fit if we wish to use single samples from the approximate

posterior for downstream tasks. Besides, importance weighted estimates have notoriously bad scaling properties to high-dimensional latent spaces [24].

Although $\text{ELBO}_{\text{IS}}$ gets tighter and the gradient variance smaller with more samples $T$, the magnitude gets even smaller. More precisely, the signal-to-noise ratio of the inference model actually converges at $O(1/\sqrt{T})$ [42]. So, while $T >> 1$ is great for the generative model, it hampers the training of the inference model. This issue can be somewhat alleviated by averaging over $M$ gradient samples, changing the convergence rate to $O(\sqrt{M}/\sqrt{T})$ [42]. However, there are now two sample sizes to tune: while $M$ regulates the variance of the gradient, $T$ regulates the tightness of the bound. In practice, increasing mini-batch size has a similar effect. One can also use different objective functions for each model, generative and inference [42].

## 5.5 VAE Issues

VAE and its variants are a great tool for generative modeling, but they are not without shortcomings. In what follows we succinctly present their main known problems so far.

### *5.5.1 Inexpressive Posterior*

The ELBO simultaneously tries to fit the data distribution through optimization of the generative model and to perform amortized inference through the optimization of the inference model. However, due to limited capacity, it is not possible to adequately perform both and failures can emerge, imposing trade-offs onto the ELBO.

The independent Gaussian assumption for the posterior limits the expressiveness of the model. One way we can circumvent this weakness is by using more flexible families for the posterior approximation. However, the proposal distribution must still be at the same time efficient to sample from, compute, and differentiate if we wish to operate with large amounts of data.

#### 5.5.1.1 Full Covariance Gaussian

One straightforward possibility is replacing the isotonic Gaussian with diagonal covariance matrix by one with correlation between the dimensions. The arbitrary multivariate Gaussian with full-rank covariance $\mathbf{\Sigma}$ allows per axis scaling and rotation, establishing preferential directions in the latent space.

Directly learning the covariance is troublesome because the number of parameters grows at $O(d^2)$ with the number of dimensions $d$ of the latent space, and we need to assure that $\mathbf{\Sigma}$ is semipositive definite. Alternatively, we can again write the

latent random variable as the deterministic transformation of a base random variable. We use $\mathbf{z} = \boldsymbol{\mu} + \mathbf{L}\boldsymbol{\epsilon}$ with $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\mathbf{L}$ a lower triangular matrix with non-zero entries on the diagonal.

#### 5.5.1.2 Auxiliary Latent Variables

We can use auxiliary latent variables $\mathbf{A}$ to augment the inference and generative models [30, 43], as for example,

$$q(\mathbf{z} \mid \mathbf{X}) = \int q(\mathbf{a}, \mathbf{z} \mid \mathbf{X}) d\mathbf{a} = \int q(\mathbf{a} \mid \mathbf{X}) q(\mathbf{z} \mid \mathbf{A}, \mathbf{X}) d\mathbf{a}. \tag{5.17}$$

This hierarchical specification allows the latent variables to be correlated through $\mathbf{A}$, defining a general non-Gaussian implicit marginal distribution while maintaining the computational efficiency of fully factorized models.

Similarly, for the generative model, we have

$$p(\mathbf{z}, \mathbf{x}) = \int p(\mathbf{z}, \mathbf{x}, \mathbf{a}) d\mathbf{a} = \int p(\mathbf{a} \mid \mathbf{Z}, \mathbf{X}) p(\mathbf{z}, \mathbf{x}) d\mathbf{a}. \tag{5.18}$$

The ELBO objective for the auxiliary VAE follows derivations of (3.7) in Sect. 3.2.1.1 straightforwardly, giving

$$\begin{aligned} \log p(\mathbf{x}) &= \log \int p(\mathbf{a}, \mathbf{z}, \mathbf{x}) d\mathbf{a} d\mathbf{z} \\ &\geq \mathbb{E}_q \left[ \log \frac{p(\mathbf{a} \mid \mathbf{Z}, \mathbf{X}; \boldsymbol{\theta}) p(\mathbf{x} \mid \mathbf{Z}; \boldsymbol{\theta}) p(\mathbf{z})}{q(\mathbf{a} \mid \mathbf{X}; \boldsymbol{\psi}) q(\mathbf{z} \mid \mathbf{A}, \mathbf{X}; \boldsymbol{\psi})} \right] \qquad (5.19) \\ &= \text{ELBO}_{\text{aux}}. \qquad (5.20) \end{aligned}$$

#### 5.5.1.3 Normalizing Flow

A normalizing flow is a sequence of invertible mappings that transforms an initial base distribution into a complex one [45]. At each step $k$, it transforms the distribution $q_{K-1}(\mathbf{z}_{k-1})$ into $q_K(\mathbf{z}_k)$ through the change of variable specified by the mapping $\mathbf{z}_k = f_k(\mathbf{z}_{k-1})$, specifically

$$q_K(\mathbf{z}_k) = q_{K-1}(\mathbf{z}_{k-1}) \left| \det \left( \frac{\partial f_k^{-1}}{\partial \mathbf{z}_{k-1}} \right) \right| = q_{K-1}(\mathbf{z}_{k-1}) \left| \det \left( \frac{\partial f_k}{\partial \mathbf{z}_{k-1}} \right) \right|^{-1}, \tag{5.21}$$

where $\frac{\partial f}{\partial \mathbf{z}}$ is the Jacobian and det the determinant of the matrix.

If the Jacobian determinant can be computed, it is possible to sample from the estimated density by sampling from the base distribution and applying the chain of mappings $f_1 \circ \cdots \circ f_K$. Moreover, because the mappings are invertible, we can also perform inference by applying the inverse mappings in the reversed order $f_K^{-1} \circ \cdots \circ f_1^{-1}$. The log-likelihood of the final distribution $q_K(\mathbf{z}_k)$ can be written as

$$\begin{aligned} \log q_K(\mathbf{z}_k) &= \log \left[ q_0(\mathbf{z}_0) \prod_{k=1}^{K} \left| \det \left( \frac{\partial f_k}{\partial \mathbf{z}_{k-1}} \right) \right|^{-1} \right] \\ &= \log q_0(\mathbf{z}_0) - \sum_{k=1}^{K} \log \left| \det \left( \frac{\partial f_k}{\partial \mathbf{z}_{k-1}} \right) \right| . \end{aligned} \tag{5.22}$$

We can then use normalizing flows parameterized by the output of the inference network to approximate the posterior distribution $q(\mathbf{z} \mid \mathbf{X})$ from a simple base distribution $p(\boldsymbol{\epsilon})$, i.e., standard multivariate Gaussian, and obtain more complex distributions [45]. For example, we can use a single hidden-layer feedforward network with scalar output to define a non-linear transformation [45], like

$$f(\mathbf{z}) = \mathbf{z} + \mathbf{u} h(\mathbf{w}^T \mathbf{z} + \mathbf{b}), \tag{5.23}$$

where the weight $\mathbf{w}$, bias $\mathbf{b}$, and scale $\mathbf{u}$ are learned parameter vectors and $h(\cdot)$ a smooth elementwise non-linearity.

The above mapping is not very expressive, and we need a large stack of chained transformations to capture high-dimensional dependencies [26]. Nonetheless, we can introduce such dependencies by using autoregressive functions that preserve the Jacobian computation simple enough [9, 26, 40]. Autoregressive flows with affine transformations are very popular, but their expressiveness is limited due to their affine nature.Still, chaining multiple layers of autoregressive transformations leads to complex and multivariate distributions.

### 5.5.2 *The Posterior Collapse*

The posterior collapse is the common effect of achieving the undesirable local optimum $q(\mathbf{z} \mid \mathbf{X}; \boldsymbol{\psi}) = p(\mathbf{z} \mid \mathbf{X}; \boldsymbol{\theta}) = p(\mathbf{z})$ during the VAE optimization. In the state where the variational posterior and true model posterior collapse to the prior, the posterior encodes no information about the input $\mathbf{x}$, and no useful latent representation was learned. The KL divergence is zero and no extra bits are communicated. This corresponds to the $(D, 0)$ of the distortion-rate plane discussed in Sects. 5.3.2 and 5.3.2.

The true model posterior is a moving target for the inference network, so it naturally lags behind, especially at the beginning of training where the variables $\mathbf{X}$ and $\mathbf{Z}$ are nearly independent as a consequence of the random initialization of

the models' parameters. When $q(\mathbf{z} \mid \mathbf{X}; \boldsymbol{\psi})$ and $p(\mathbf{z} \mid \mathbf{X}; \boldsymbol{\theta})$ start to diverge, but $\mathbf{Z}$ still is independent of $\mathbf{X}$ such that $q(\mathbf{z} \mid \mathbf{X}; \boldsymbol{\psi}) \approx q(\mathbf{z}; \boldsymbol{\psi})$, and no information is passed through the model, the regularization signal from the KL divergence term in the ELBO objective may be too strong compared to the weak signal coming from the data likelihood [15]. As a result, the model is encouraged to ignore the latent encoding and converges to the local optimum $q(\mathbf{z} \mid \mathbf{X}; \boldsymbol{\psi}) = p(\mathbf{z} \mid \mathbf{X}; \boldsymbol{\theta}) = p(\mathbf{z})$. Then, it is the generative model that is responsible for reconstructing $\mathbf{X}$ and effectively maximizing the bound.

The information about the data distribution can be encoded both on the latent space representation and on the weights of the generative model. An excessively powerful generator facilitates the posterior collapse because it has enough capacity to store sufficient information in its weights. Indeed, it is frequently reported in the literature that autoregressive generators are prone to this effect [2, 15].

Many modifications have been proposed to solve posterior collapse since it was first detected, such as using an annealing schedule on weighting factor on the KL term [51]; modifying the ELBO objective to ensure that the KL has on average a minimum value, guaranteeing a minimum amount of information transmission [26]; using the $\beta$-VAE framework with $\beta < 1$ [2]; and aggressively updating the inference network by optimizing it in an inner loop [15]. The latter is currently the most promising one, obtaining better performance for both inference and generative models, but at a higher computational cost, i.e., 2–3 times slower [15].

### 5.5.3 *Latent 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 under expectation to be continuous and cannot be used for the discrete case. This characteristic restraints our modeling capacity, and we cannot, for example, use categorical distributions for the latent space.

There are works that use the score function estimator instead to estimate the gradient, such as [36, 37]. The alternative estimator on which these algorithms rely makes no assumption about the underlying distribution and can work with both discrete and continuous data, eliminating the issue whatsoever. Although more general, as shown in Appendix A.1, the score function estimator has large variance, so it requires usage of variance reduction techniques to work properly.

There are works on how to reparameterize discrete random variables by relaxing them into continuous distributions [19, 33] 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 annealed during training.

#### 5.5.3.1 Continuous Relaxation

Similarly to the reparameterization trick where we sampled from an arbitrary Gaussian distribution through transformation of a standard Gaussian random variable, we can use the Gumbel-max trick to allow us to sample a discrete random variable $X$ from the unnormalized $K$-Categorical distribution $\widetilde{\Pi_K}$, whose PDF is $\pi_K(x)$, using a continuous distribution [32], following

$$x = \underset{k}{\text{argmax}} \log \pi_k(\cdot) + G_k, \tag{5.24}$$

where $G_k$ is one element from the $K$-sample independent and identically distributed (iid) sequence of standard Gumbel distributed random variables.

The Gumbel is useful to model the distribution of extreme values of samples from the exponential family and its CDF defined by $F(x) = \exp(-\exp(-x))$. Thus, we can use the inverse formula to obtain

$$G = -\log(-\log(U)), \text{ with } U \sim \mathcal{U}[0, 1], \tag{5.25}$$

and combine it with (5.24) to efficiently get discrete samples by drawing from a standard uniform distribution [32].

However, the argmax function is not differentiable and cannot be used within a gradient learning setting. Thus, we replace it with the softmax function to obtain a continuous relaxation with a temperature parameter $\tau$ over the probability simplex [19, 33], as follows:

$$x = \text{softmax}\left(\log \alpha + G\right). \tag{5.26}$$

The softened version of the reparameterization is known as the Gumbel-softmax trick and the resulting distribution as Concrete. The temperature $\tau$ regulates the discreteness of the representation such that $\lim_{\tau \to 0} \text{Concrete}_K(x) = \Pi_K(x)$. Higher temperatures result in a smoother distribution but lower variance of the gradients, while lower temperatures give more accurate samples but higher variance. The additional hyper-parameter $\tau$ is robust and generally follows an annealing schedule from high to low temperature during the optimization [19].

#### 5.5.3.2 Vector Quantization

A popular method for learning discrete latent representations is the Vector Quantized VAE, where vector quantization maps the output of the recognition model to the nearest of $M$ reference elements in the codebook, which is then passed to the generative model [58]. Since the nearest-neighbor match is not differentiable, for the algorithm to work the gradients must be copied from the generator input to the decoder output and the codebook updated using nearest-neighbor lookup to match.

Although the algorithm achieves great generation results, the original and prevalent formulation is not probabilistic: all operations are deterministic [58]. Nonetheless, we can replace the nearest-neighbor lookup by sampling over a K-Categorical distribution $\Pi_K$, defined as

$$\Pi_K = \prod_{i=1}^{k} p_i^{[x=i]}, \tag{5.27}$$

where $[x = i]$ evaluates to 1 if $x = i$, 0 otherwise.

The probabilities $p_i$ should be the distance between the recognition model output $h(\mathcal{X})$ and the codebook elements $\{\mathbf{c}\}_M$ [52], such that

$$q(\mathbf{z} \mid \mathbf{X}) = \Pi_K \left(\mathbf{z} \mid \text{softmax} \left(\|\{\mathbf{c}\}_M - h(\mathcal{X})\|_2\right)\right) \tag{5.28}$$

## 5.6 Experiments

We train and analyze the VAE and CVAE methods with different latent dimensions on two well-known image toy data sets. In addition, we study the effect on the distortion-rate plane of varying the weight $\beta$ of the KL term in (5.13) as well as the effect of normalizing flows on the posterior distribution.

### *5.6.1 Data Sets*

#### 5.6.1.1 MNIST

The MNIST data set is composed of 60,000 training and 10,000 testing $28 \times 28$ grayscale images of handwritten digits [27]. Each sample depicts a single digit out the 10 possibilities. Figure 5.5 presents one example of each digit class.

**Fig. 5.5** Mosaic of the 10 different digit classes of the MNIST data set

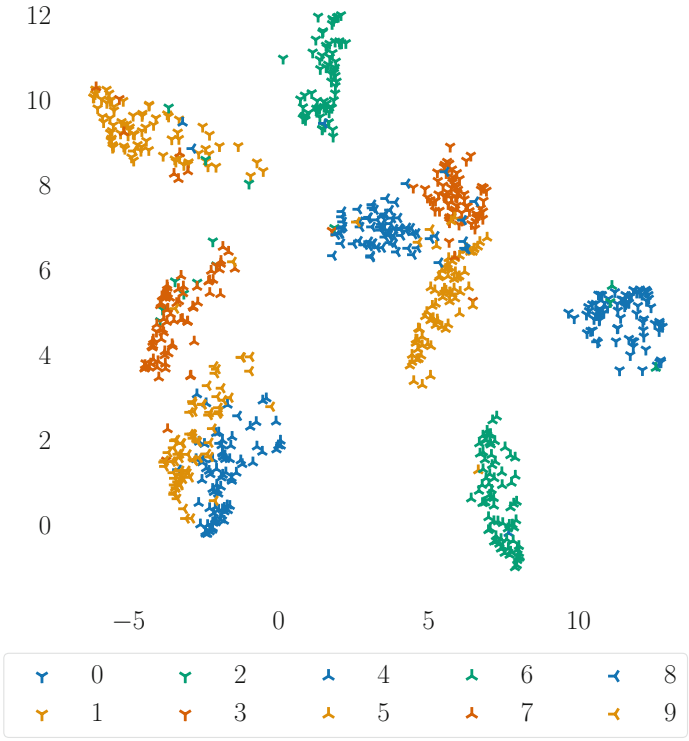



**Fig. 5.6** UMAP 2D projection of the raw pixel space of MNIST

Uniform Manifold Approximation and Projection (UMAP) [34] can be used as an out-of-the-box visualization tool similar to t-distributed Stochastic Neighbor Embedding (t-SNE) [31], while being faster, better scaling to high dimensions and better preserving aspects of global data structure. Using this dimension reduction technique, we observe in Fig. 5.6 the structure of MNIST data set. It has well-defined clusters for all of its classes.

Standard ML algorithms obtain over 97% classification accuracy on MNIST, i.e., random forest [5] and support vector machine with Gaussian kernel [49], while DL models over 99.5% [59]. There is almost no space left for researchers to evaluate if observed performance improvements are statistically relevant. Hence, MNIST has little use 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.

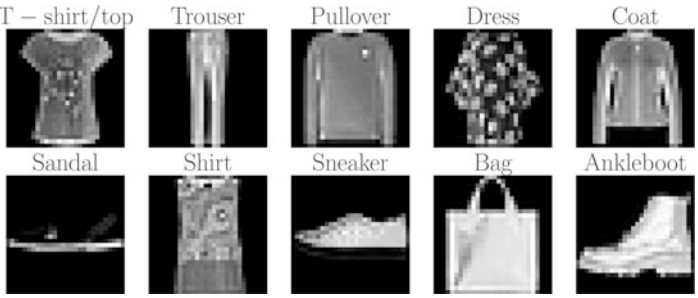



**Fig. 5.7** Mosaic of the 10 different classes of the Fashion-MNIST data set

#### 5.6.1.2 Fashion-MNIST

Fashion-MNIST presents the same general layout: 60,000 training and 10,000 test samples, 28×28 grayscale images, and 10 possible exclusive classes [60]. It is constructed to be a drop-in replacement for MNIST with each class associated with a different piece of clothing, as shown in Fig. 5.7. It is noticeable the higher level of details in the images.

Comparing the raw pixel structure of Fashion-MNIST data set, shown in Fig. 5.8, with that from MNIST in Fig. 5.6, we note that the former has clusters corresponding to garments for the same body region partially overlapping. Although simple, Fashion-MNIST is not as easy as MNIST and still has margin for improvements [60].

### *5.6.2 Experimental Setup*

We implement the encoder and generator of all models as fully connected networks with ReLU activations and Gaussian distributions for the latent spaces. In all experiments, we binarize the input images $\mathbf{x}_i$ in the range $[0, 1]$ to $\{0, 1\}$ with a threshold of 0.5, considering each element $j$ as a realization of a independent Bernoulli distribution. Similarly, we model the output as the parameters of independent Bernoulli distributions $\hat{X}_{ij}$. Hence, the log-likelihood function $\log p(\mathbf{x}_i \,|\, \mathbf{Z}_i; \Theta)$ becomes the binary cross entropy, like

$$\mathcal{H}[p_i, \hat{p}_i] = \sum_{j \in |\mathcal{X}_i|} -x_{ij} \log x_{ij} - (1 - \hat{x}_{ij}) \log(1 - \hat{x}_{ij}), \tag{5.29}$$

where $p_i$ is the binomial distribution induced by the binarized sample $\mathbf{x}_i$, and the $\hat{\cdot}$ symbol denotes quantities related to the model output.

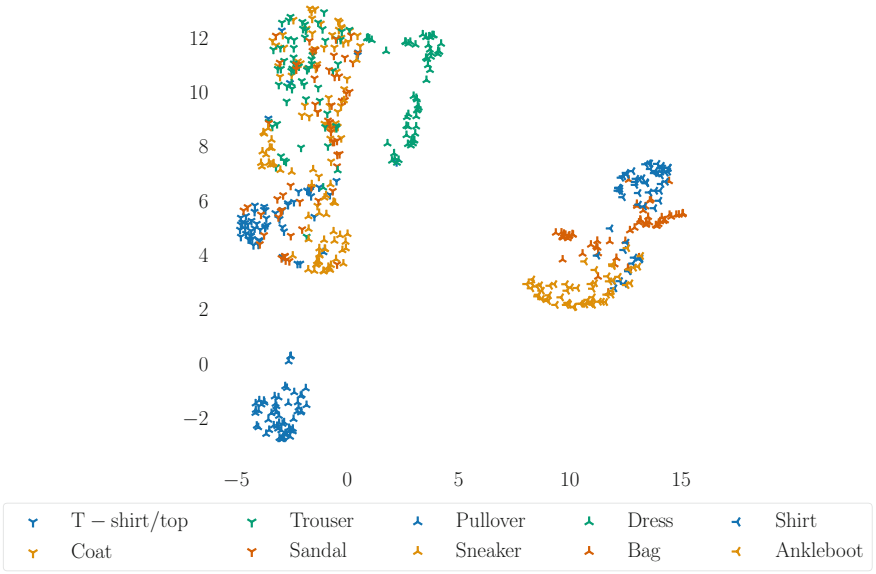



**Fig. 5.8** UMAP 2D projection of the raw pixel space of Fashion-MNIST

Additionally, we use mini-batches of size $bs = 128$, draw $T = 1$ MC sample of the latent space for each input example, and train for $epc = 100$ epochs with Adam [21] using a learning rate of $lr = 0.001$.

We train both the VAE and CVAE with varying latent space sizes $d = \{2, 8, 32, 128\}$. Since the MNIST data set is simpler, we use $hl = 1$ hidden layer for both the encoder and the generator, whereas for the Fashion-MNIST we use $hl = 2$. For simplicity, we design the decoders as mirrored versions of the encoders. Thus, the constructed models have the structure:

- MNIST: $784 \rightarrow 200 \rightarrow d \rightarrow 200 \rightarrow 784$;
- Fashion-MNIST: $784 \rightarrow 400 \rightarrow 200 \rightarrow d \rightarrow 200 \rightarrow 400 \rightarrow 784$.

### *5.6.3 Results*

Overall, the behavior of VAE and CVAE models is similar across both data sets as Fig. 5.9 shows. When there are few latent dimensions, the models lose a lot of information in compression and cannot properly represent the different features of the data set, what explains the low ELBO curves for $d = 2$ in Fig. 5.9. Increasing the dimension $d$ of the latent space allows the models to encode relevant information that was previously ignored, bringing expressive gains to the ELBO. Although this boost on performance saturates with larger $d$, it has almost no negative effect on performance for excessively large $d$, i.e., the ELBO for the 128-dimensional

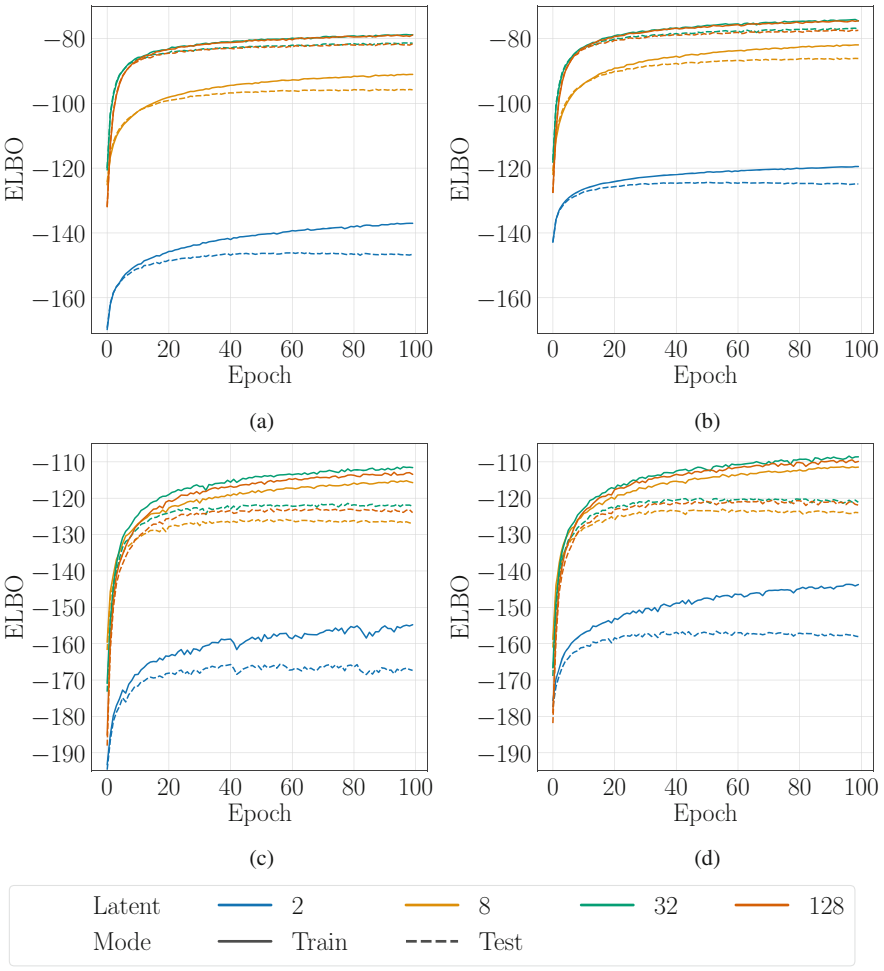



**Fig. 5.9** Training and evaluation ELBO for the MNIST and Fashion-MNIST data sets for different sizes of the latent dimension space. Performance on Fashion-MNIST is lower than on MNIST, and the CVAE models achieve performances similar to the VAEs. (**a**) VAE model on MNIST. (**b**) CVAE model on MNIST. (**c**) VAE model on Fashion-MNIST. (**d**) CVAE model on Fashion-MNIST

models is similar to that for the 32D ones in Figs. 5.9a,b, and only slightly lower in Figs. 5.9c,d. VAEs are robust to overfitting, at least with respect to the size of the latent space.

As expected, the Fashion-MNIST models have remarkably worse results. Indeed, we can confirm it visually by observing the generated samples from Fig. 5.10. While the original MNIST samples are not very rich in details, this is not true for the Fashion-MNIST objects, and the VAE struggles to recover the finer details and more complex shapes. The main reason why the trained CVAEs do not increase

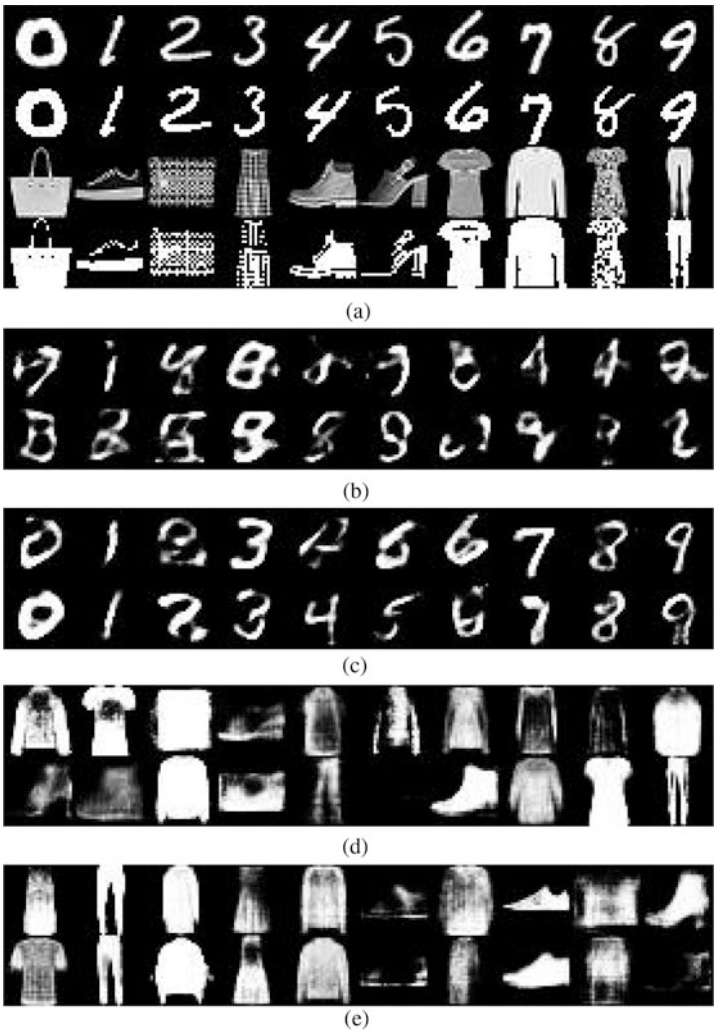



**Fig. 5.10** Samples generated by the VAE and CVAE models with $d = 32$. (**a**) Original and corresponding binarized samples from MNIST and FashionMNIST data sets. (**b**) VAE model—MNIST. (**c**) CVAE model—MNIST. (**d**) VAE model—Fashion-MNIST. (**e**) CVAE model—Fashion-MNIST

performance in our experiments is because, in most cases, i.e., $d = \{8, 32, 128\}$, further augmenting the latent dimension does not translate on better ELBOs. Thus, appending 10 extra dimensions for the conditioning does not make the models any more expressible.

Even though the CVAE does a slightly better job at generating new MNIST images, what we can observe by comparing Figs. 5.10b,c, for Fashion-MNIST data

the sample quality is indistinguishable for both CVAE and VAE, because images in Fig. 5.10e are only marginally better than those of the VAE model, displayed in Fig. 5.10d. The quality of both models is overall poor if compared to the real samples of Fig. 5.7.

One central argument during the construction of the VAE was the latent space structure. This property allows us to smoothly interpolate between different latent representations of arbitrary dimension to create new image samples. In Fig. 5.11a, we interpolate between pairs of randomly drawn high-dimensional latent space samples conditioned on the same MNIST digits. We can see the samples gradually morphing, i.e., the 0 gets thinner and the 1 gets simultaneously bolder and straighter. When $d = 2$, we can span the whole latent space and plot its reconstructions. Figure 5.11b shows samples generated from evenly spaced percentiles of the latent Gaussian prior, note the smoothness in the transition between concepts. The models effectively encode factors of variation of the data into the latent space.

We note from Fig. 5.12 that the latent representations of samples from digits 4 and 9, as well as 5 and 3, are generally overlapped, which indicates that the model cannot properly tell them apart. A classifier built from the latent feature space would have a poor accuracy for samples from these classes. Similarly, as already observed in Fig. 5.8, 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 that 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. More modern types of flow, such as Sylvester flows [56], a generalization of the planar flow, use more powerful transformations and is better suited to real applications.

In Sect. 5.5.1, we discussed different approaches for obtaining more expressive posteriors. Normalizing flows are one of the most prominent approaches nowadays, with several works relying exclusively on it to efficiently perform density estimation and sample generation [22, 26, 40]. In our experiments we use it to enhance VI and obtain models with better posterior approximations and, consequently, higher likelihoods. Table 5.1 shows the estimated marginal log-likelihood on both MNIST and Fashion-MNIST of the VAE with increasing number of steps $K$ in the planar flow, whose transformation was defined in (5.23). Although not statistically significant for $K = \{2, 4, 8\}$, the gain in performance is clear when comparing with the plain VAE. Planar transformations are an elementary case and affect only a small volume of the space at each step, thus calling for a large number of steps to effectively obtain the desired effect, especially on high-dimensional spaces.

Unfortunately, all reconstructed and generated samples were considerably blurry. Blurriness is a general characteristic of VAEs, originating from the objective that seeks to minimize the *average* log-likelihood of data. On average, they may be good, but individually they are not sharp. This effect is more pronounced on Fashion-MNIST as clothes are more diverse and have more details than digits and precisely what we observe in Figs. 5.10 and 5.13. Still, much of the fine-grained details are lost in the binarization procedure applied on the input samples. We could use

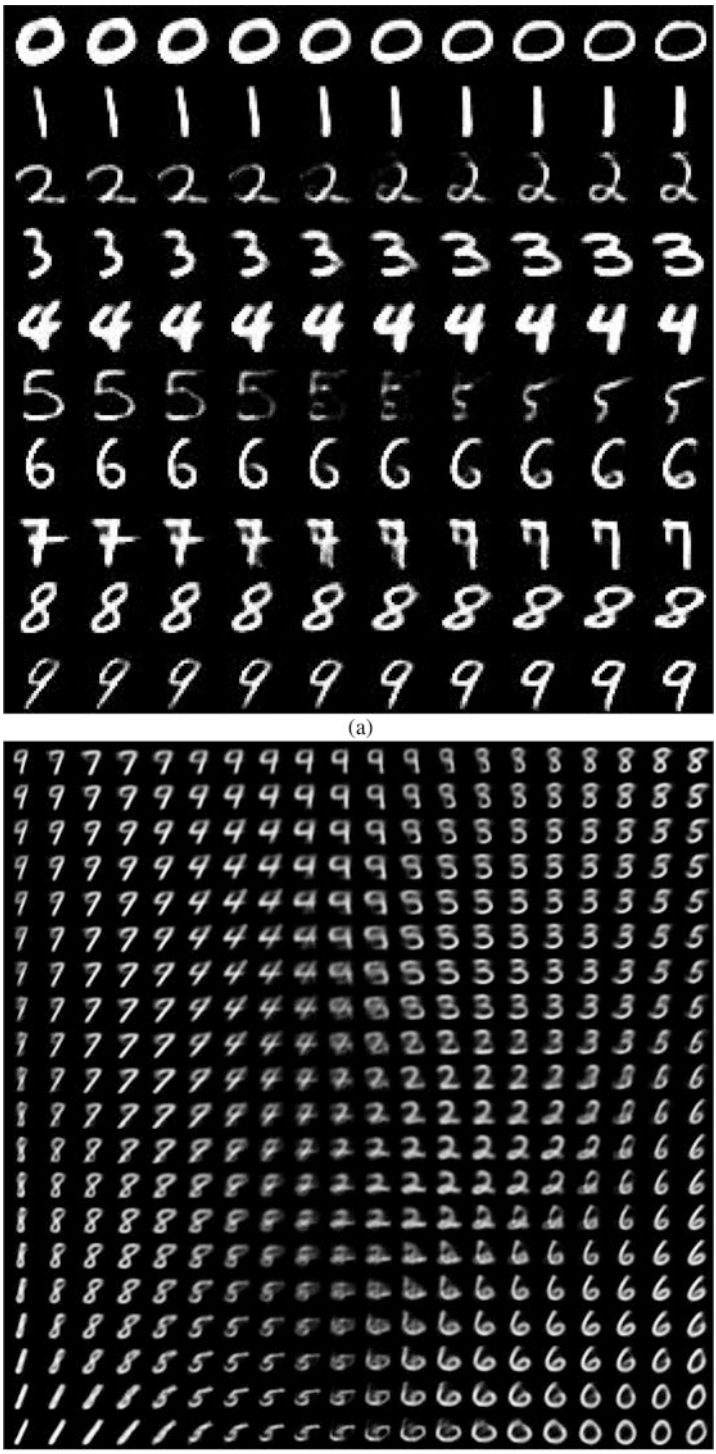


**Fig. 5.11** Interpolation of the latent space of VAE and CVAE models trained on MNIST. Human concepts as thickness, orientation, and digit-specific traits vary smoothly between samples, signaling the latent space effectively captures factors of variation in the data. (**a**) CVAE model with $d = 32$ on MNIST. (**b**) VAE model with $d = 2$ on MNIST

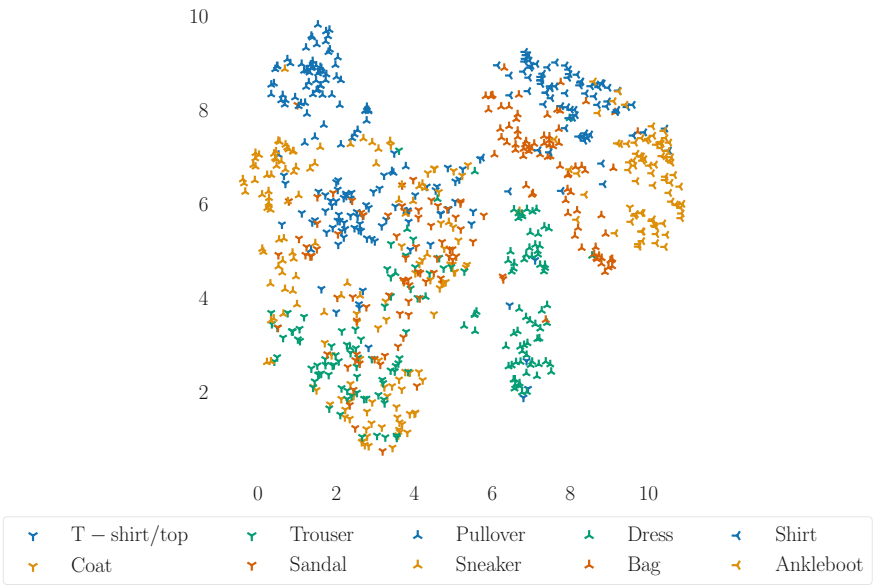



**Fig. 5.12** Visualization of the 2D projection of the VAE with $d = 32$ trained on Fashion-MNIST

**Table 5.1** Estimated marginal log-likelihood $p(\mathcal{X})$ of the VAE model with planar normalizing flows for varying length $K$. Estimations are computed by importance sampling with 1024 samples for each instance of the test set

| Steps $K$ | Marginal log-likelihood $p(\mathcal{X})$ | |
|---|---|---|
| | MNIST | FashionMNIST |
| 0 | $-73.4 \pm 0.2$ | $-116.6 \pm 0.7$ |
| 1 | $-71.4 \pm 0.1$ | $-112.3 \pm 0.6$ |
| 2 | $-71.7 \pm 0.6$ | $-112.6 \pm 0.7$ |
| 4 | $-71.8 \pm 0.2$ | $-112.5 \pm 0.7$ |
| 8 | $-71.8 \pm 0.3$ | $-112.5 \pm 0.5$ |
| 16 | $-71.0 \pm 0.3$ | $-112.9 \pm 0.6$ |
| 32 | $-70.0 \pm 0.2$ | $-113.1 \pm 0.7$ |

the original grayscale values, but the binary cross entropy that stems from the Bernoulli log-likelihood would cease to be adequate, and it would be necessary to employ one from a suitable continuous real-valued distribution, such as the logit-Normal. In general, to achieve better log-likelihood and sample quality, we need to employ better models. Indeed, plain fully connected networks have pretty much been replaced nowadays by convolutional architectures, especially in the image domain.

The KL divergence remains stable throughout the whole training, varying very little, as shown in Fig. 5.14. Although we only exhibit the case of the CVAE trained on Fashion-MNIST, this is a general behavior observed in all experiments. This is a consequence of the powerful regularizing effect the KL term has on the model, seen in Sect. 5.5.2. The learned posterior distribution moves away from the prior within the first epoch and, even though not much has been learned yet as confirmed by

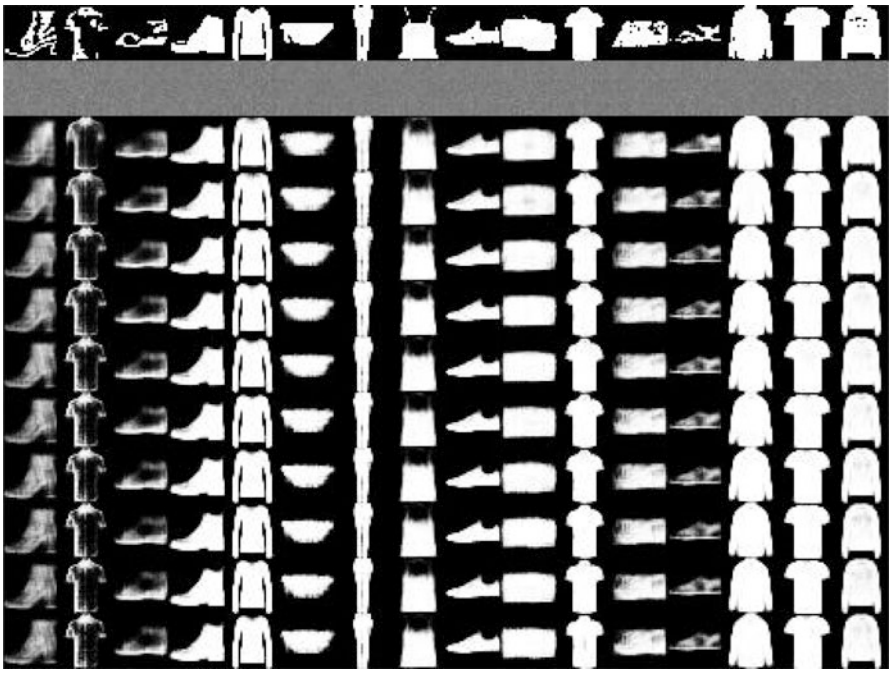


**Fig. 5.13** Reconstruction of Fashion-MNIST samples throughout training of the VAE model with $d = 32$ latent dimensions. The original samples are in the first row, while the others are snapshots at every 10 epochs. Most of the information is learned within the first 10 epochs. Still, it is perceptible the higher level of details in the images in the last row

Fig. 5.13, the posterior stays at approximately the same "distance" during the rest of the optimization procedure. Had we used a powerful generator with greater ability to reconstruct the input, e.g., autoregressive model, the KL term strength would have succeeded in keeping the posterior aligned with the prior, causing the undesired posterior collapse. The most straightforward way to sidestep this issue is by directly decreasing the value of $\beta$ in the $\beta$-VAE model. Adjusting the value of $\beta$ allows us to weight the relative importance of the regularization effect of the KL term on the ELBO. The hyper-parameter balances the compromise between distortion, the reconstruction error, communication rate, and the posterior misalignment, given the model's limited capacity. This characteristic can be observed from Fig. 5.15, where model performance is plotted on the distortion-rate plane.

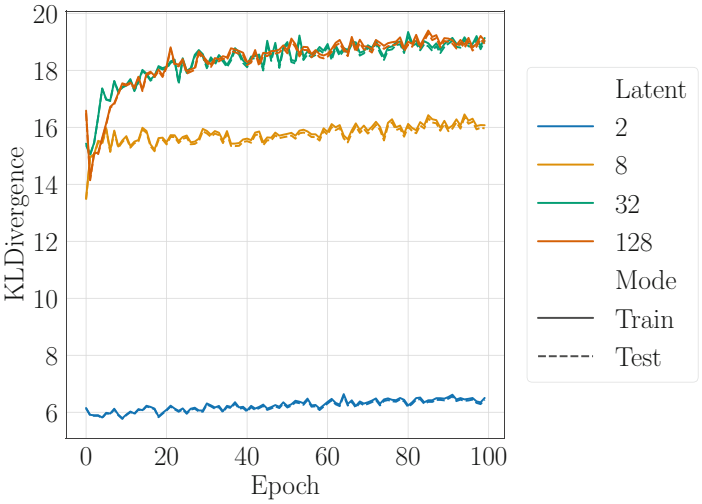



**Fig. 5.14** The KL divergence curve during the training and evaluation of the CVAE models with different latent dimension sizes in the Fashion-MNIST data set

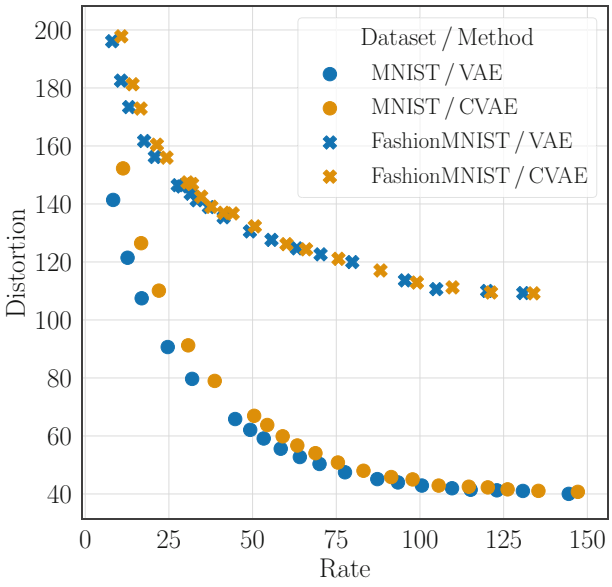



**Fig. 5.15** Representation of the ELBO on the distortion-rate plane (in nats). The ELBO can be decomposed into the fidelity term, measured by the log-likelihood of the data set and a rate term, quantifying the average number of extra bits needed to correct the samples inaccurately represented with the prior distribution. In the plane graphic we use negative fidelity, the distortion. Given a model with finite capacity, not capable of achieving the data entropy lower bound, we must set the hyper-parameter $\beta$ and make other design decisions with this behavior trade-off in mind

## 5.7 Application: Generative Models on Semi-supervised Learning

Popular modern approaches to ML rely on models with millions of parameters and require large amounts of annotated data, currently the most expensive and desirable asset in AI. When annotation is not extensively available, supervised models overfit to the presented data and achieve poor generalization. Generative models allow us to leverage performance from unlabeled samples, effectively reducing the reliance on annotations, under an approach we call semi-supervised learning.

We can optimize a discriminative classifier together with a VAE, sharing their parameters, and use them for semi-supervised learning of the target variable $\mathbf{Y}$ [25, 30]. For the unlabeled samples, we treat $\mathbf{Y}$ as a discrete latent stochastic variable distributed according to a categorical distribution, what enables us to infer the target label $\mathbf{Y}$. Figure 5.16 shows the PGMs for the generative and the inference models. Note that we use $\mathbf{Y}$ to condition the latent variable $\mathbf{Y}$, segmenting the latent space in different regions according to the class, similarly to the CVAE in Sect. 5.3.1. Figure 5.17 illustrates the high-level computational graph of such model.

More complex versions of the model in Fig. 5.17 were able to achieve an average classification error below 1% on MNIST while using only 10 labeled images per class, a total of 100 out of the 60,000 available in the training set [30]. The auxiliary deep generative model extends the above VAE with an auxiliary latent variable (see Sect. 5.5.1.2), making it a two-layered stochastic model [30]. This increases the flexibility of the variational approximation, allowing it to better fit complex latent distributions, hence improving the variational lower bound. The underlying PGM for the generative model is shown in Fig. 5.18a, whereas Fig. 5.18b depicts the inference model. They are basically the same as those of Fig. 5.18a, except for the inclusion of the auxiliary node $\mathbf{a}$. Figure 5.19 illustrates the high-level computational graph of the auxiliary deep generative model used for

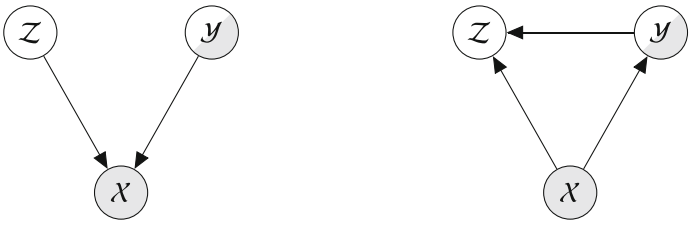



**Fig. 5.16** Graphical model of the semi-supervised VAE. The partially colored node $\mathbf{y}$ denotes the partially observed target labels. We assume that $\mathbf{y}$ and $\mathbf{z}$ are conditionally independent in the generative process, so while $\mathbf{y}$ captures digits' semantics, $\mathbf{z}$ captures styles and position. Since $\mathbf{z}$ is never observed and different digits possess different styles, $\mathbf{y}$ is used during inference to estimate $\mathbf{z}$, such relation is depicted in (**b**) by the arrow $\mathbf{y} \rightarrow \mathbf{z}$. (**a**) Graphical model for the generative network. (**b**) Graphical model for the inference network

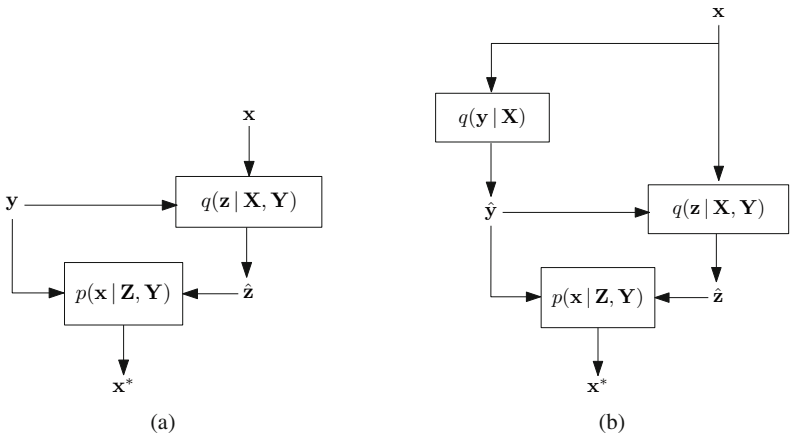



**Fig. 5.17** Overview of the computational diagram of the semi-supervised VAE model in the example. $\mathbf{x}^*$ is the reconstruction of the original sample $\mathbf{x}$. For labeled samples, we have once again the CVAE seen in Sect. 5.3.1, where $\hat{\mathbf{z}}$ represents the estimated value of $\mathbf{z}$. For unlabeled data, however, $\mathbf{y}$ is unknown, and its value must first be inferred from the categorical distribution $q(\mathbf{y} \mid \mathbf{X})$, which gives the estimate $\hat{\mathbf{y}}$. Although each box is implemented by a separate fully connected NN, the complete model is optimized simultaneously. (**a**) Diagram for labeled samples. (**b**) Diagram for unlabeled samples

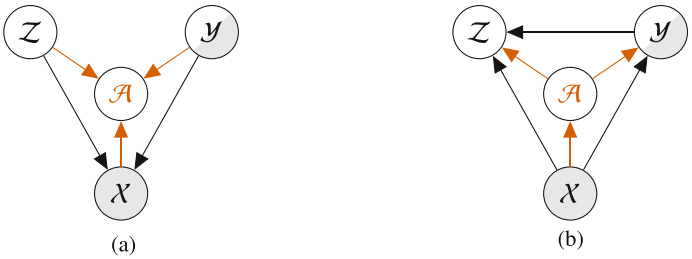



**Fig. 5.18** Graphical representations of both inference and generative parts of the auxiliary deep generative model. The partially colored node $\mathbf{y}$ denotes the partially observed target labels. The novelty here w.r.t. Fig. 5.16 is the inclusion of the stochastic node $\mathbf{a}$, which gives more flexibility to the variational posterior. (**a**) Graphical model for the generative network. (**b**) Graphical model for the inference network

semi-supervised learning. Although only one variable was added to the model, two distributions were included: $q(\mathbf{a} \mid \mathbf{X})$ for the inference network and $p(\mathbf{a} \mid \mathbf{X}, \mathbf{Z}, \mathbf{Y})$ for its generative counterpart.

In what follows, we use the model in Figs. 5.17 and 5.16, i.e., without the auxiliary variable $\mathcal{A}$, to construct our toy example.

We model the generative process of the $\mathbf{x}_i$ as also being dependent on the partially observed latent class variable $\mathbf{Y}_i$ that specifies the digit. Both latent variables $\mathbf{Y}$ and $\mathbf{Z}$ are conditionally independent so that the first captures digits' semantics and the second digits' styles, independently. Hence, we can write $p(\mathbf{y}_i, \mathbf{z}_i \mid \mathbf{X}_i) =$

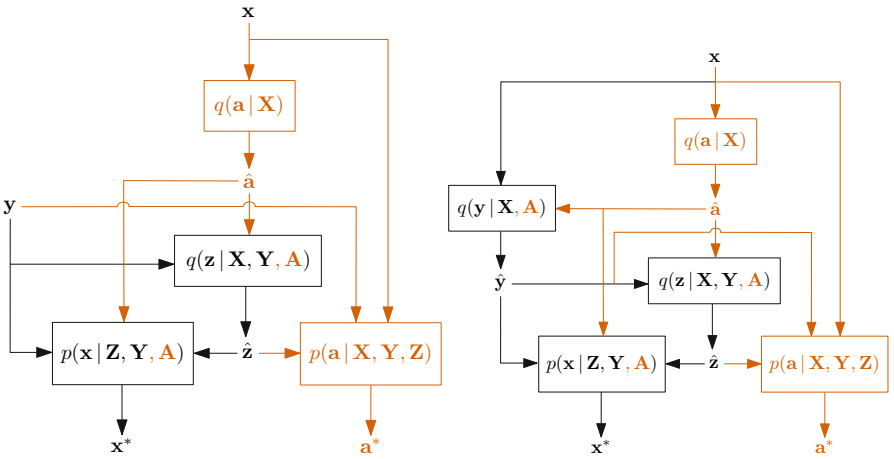



**Fig. 5.19** Computational diagram overview of the auxiliary deep generative model. All differences w.r.t. Fig. 5.17, seen above in color, stem from the inclusion of the auxiliary variable $\mathbf{A}$, which represents an intermediate step in the inference process, illustrated in Fig. 5.18b. Then, $\mathbf{a}$ feeds $\mathbf{Y}$ and $\mathbf{Z}$ encoders (the $q(\cdot)$ blocks), as well as the $\mathbf{X}$ decoder (the $q(\mathbf{x} \mid \cdot)$ block). Similarly to Fig. 5.17, $\mathbf{x}^*$ and $\mathbf{a}^*$ both represent the reconstruction of the samples $\mathbf{x}$ and $\mathbf{a}$, respectively. (**a**) Diagram for labeled samples. (**b**) Diagram for unlabeled samples

$p(\mathbf{y}_i \mid \mathbf{X}_i)p(\mathbf{z}_i \mid \mathbf{X}_i)$. We define the prior $p(\mathbf{y}_i)$ to be a K-Categorical distribution $\Pi_K$ over the class variable and the prior $p(\mathbf{z})$ a multivariate standard Gaussian, similarly to Sect. 5.3. Then, the complete generative process for one sample is defined by

$$p(\mathbf{x}_i, \mathbf{z}_i, \mathbf{y}_i) = p(\mathbf{x}_i \mid \mathbf{Z}_i, \mathbf{Y}_i)p(\mathbf{z})p(\mathbf{y}) \tag{5.30}$$

$$p(\mathbf{y}) = \Pi_K(\mathbf{y} \mid \pi) \tag{5.31}$$

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z} \mid \mathbf{0}, \mathbf{I}) \tag{5.32}$$

$$p(\mathbf{x}_i \mid \mathbf{Z}_i, \mathbf{Y}_i) = \prod_{j=1}^{|\mathcal{X}|} p_j(\mathbf{x}_i \mid \mathbf{Z}_i, \mathbf{Y}_i), \tag{5.33}$$

where $\pi$ is a probability vector and the elements of $\mathcal{X}$, i.e., the dimensions, iid. Specifically, we use Bernoulli variables to model the binary black-and-white pixel value.

Optimizing this model involves the marginal likelihood of observed and unobserved class variables $\mathbf{Y}$, $p(\mathbf{x}, \mathbf{y})$ and $p(\mathbf{x})$, respectively. As in Sect. 5.3, we cannot directly compute those marginals and resort to the variational framework introduced in Sect. 3.2.1, like

$$\begin{aligned}\log p(\mathbf{x}_i, \mathbf{y}_i) &= \log \int p(\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i) d\mathbf{z}_i \\ &\geq \mathbb{E}_{q(\mathbf{z}_i \mid \mathbf{X}_i, \mathbf{Y}_i; \boldsymbol{\psi})} \left[ \log \frac{p(\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i; \boldsymbol{\theta})}{q(\mathbf{z}_i \mid \mathbf{X}_i, \mathbf{Y}_i; \boldsymbol{\psi})} \right] = \mathcal{L}(\mathbf{x}_i, \mathbf{y}_i) \end{aligned} \tag{5.34}$$

$$\begin{aligned}\log p(\mathbf{x}_i) &= \log \int p(\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i) d\mathbf{z}_i d\mathbf{y}_i \\ &\geq \mathbb{E}_{q(\mathbf{z}_i, \mathbf{y}_i \mid \mathbf{X}_i; \boldsymbol{\psi})} \left[ \log \frac{p(\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i; \boldsymbol{\theta})}{q(\mathbf{z}_i, \mathbf{y}_i \mid \mathbf{X}_i; \boldsymbol{\psi})} \right] \\ &= \mathbb{E}_{q(\mathbf{y}_i \mid \mathbf{X}_i; \boldsymbol{\psi})} \left[ \mathbb{E}_{q(\mathbf{z}_i \mid \mathbf{Y}_i, \mathbf{X}_i; \boldsymbol{\psi})} \left[ \log \frac{p(\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i; \boldsymbol{\theta})}{q(\mathbf{z}_i, \mathbf{y}_i \mid \mathbf{X}_i; \boldsymbol{\psi})} \right] \right] = \mathcal{U}(\mathbf{x}_i), \end{aligned} \tag{5.35}$$

where $q(\cdot)$ is again the proposal distribution learned by the inference model. For detailed explanation on how to obtain the inequalities in (5.35) and (5.34), read Sect. 3.2.1.1.

We train one recognition model for each latent variable and assume that their distribution follows

$$q(\mathbf{y}_i \mid \mathbf{X}_i) = \Pi_K(\mathbf{y}_i \mid \pi(\mathbf{x}_i; \boldsymbol{\psi})) \tag{5.36}$$

$$q(\mathbf{z}_i \mid \mathbf{Y}_i, \mathbf{X}_i) = \mathcal{N}\left(\mathbf{z}_i \mid \mu(\mathbf{x}_i, \mathbf{y}_i; \boldsymbol{\phi}), \operatorname{diag}\left(\sigma^2(\mathbf{y}_i, \mathbf{x}_i; \boldsymbol{\phi})\right)\right). \tag{5.37}$$

Thus, we are able to perform inference in the latent space using (5.37) conditioned on the additional attributes $\mathbf{y}_i$, similarly to the CVAE in Sect. 5.3.1. Besides, we can infer the unknown label of $\mathbf{x}_i$ through the distribution defined by (5.36).

Unfortunately, the reparameterization trick does not apply to $q(\mathbf{y}_i \mid \mathbf{X}_i)$ because the distribution is discrete. We can use the score function estimator of the gradient instead, and however it entails high-variance estimates (see Appendix A.1). Alternatively, we can marginalize over $\mathbf{Y}_i$ in (5.37) and perform inference on $q(\mathbf{z}_i \mid \mathbf{Y}_i, \mathbf{X}_i; \boldsymbol{\psi})$ for each value of $\mathbf{y}$ [25]. However, marginalizing over all classes rapidly becomes costly since it is necessary to repeat the same operation $K$ times, where $K$ is the number of classes in a K-Categorical distribution, defined in (5.27). Another option is to relax the discrete distributions $p(\mathbf{y})$ and $q(\mathbf{z}_i \mid \mathbf{Y}_i, \mathbf{X}_i; \boldsymbol{\psi})$ onto continuous approximations using the Gumbel-softmax trick [19, 33] (see Sect. 5.5.3.1), making it possible again to apply the pathwise gradient estimator. The continuous relaxation allows us to take MC samples instead of marginalizing. When using 1 MC sample, it was experimentally verified to increase the overall training speed by $2\times$ for 10 classes and $10\times$ for 100 classes compared to marginalization [19].

There is still a practical issue in the proposed model: direct optimization of the label predictive distribution $q(\mathbf{y}_i \mid \mathbf{X}_i)$ is restricted to the unlabeled portion of the data via (5.35). Let $S$ be the set of indexes of all labeled samples in the data set.

**Table 5.2** Results of both semi-supervised VAE and supervised NN classifier on the Fashion-MNIST and MNIST for different amounts of training labels. The values inside parentheses represent the percentage of the original set size used

| Model accuracy on test set | | | | |
|---|---|---|---|---|
| | Fashion-MNIST | | MNIST | |
| Label Count | Semi-supervised | Supervised | Semi-supervised | Supervised |
| 6000 (10%) | 79.2 | 77.1 | 93.6 | 91.4 |
| 3000 (5%) | 77.9 | 71.6 | 91.4 | 88.0 |
| 600 (1%) | 72.0 | 56.4 | 86.1 | 44.6 |
| 300 (0.5%) | 70.9 | 46.7 | 81.7 | 30.1 |
| 100 (0.17%) | 63.5 | 21.3 | 68.0 | 20.4 |

For $i \in S$, the model does not directly learn to infer classes. Thus, we augment the objective by adding an auxiliary cross-entropy term that constrains $q(\mathbf{y}_i \mid \mathbf{X}_i)$ to distributions that correctly classify the sample $\mathbf{x}_i$ according to the observed class label $\mathbf{y}_i$, which leads to

$$\mathcal{J} = \left[\sum_{i \in S} \mathcal{U}(\mathbf{x}_i) + \sum_{i \notin S} \mathcal{L}(\mathbf{x}_i, \mathbf{y}_i)\right] + \alpha \sum_{i \notin S} \log q(\mathbf{y}_i \mid \mathbf{X}_i), \tag{5.38}$$

where the weight $\alpha$ is a hyper-parameter that balances the regularization strength of the cross-entropy term.

Although we arbitrarily appended the cross-entropy term to (5.38) to enhance learning, we could have achieved the same result directly from the variational framework by also inferring the parameters $\pi$ in (5.36) coupling it with a symmetric Dirichlet prior for $p(\pi)$, instead of defining a categorical prior of $\mathbf{Y}$ [25].

Table 5.2 presents the classification accuracy obtained from the semi-supervised VAE described in this section for the MNIST and Fashion-MNIST data sets at different levels of annotation.

We construct the model by following the diagram in Fig. 5.17, using 2 hidden-layer fully connected networks with ReLU activations [38] for each module. The Gumbel-softmax approximation to the categorical distribution $\Pi_K(\mathbf{y}_i \mid \pi(\mathbf{x}_i; \boldsymbol{\psi}))$ introduces a temperature hyper-parameter $T_k$, which we set to $T_k = 0.6$. Additionally, we set the latent space dimension $d = 32$, draw $T = 1$ MC samples from the approximating distributions $q(\mathbf{y}_i \mid \mathbf{X}_i)$ and $q(\mathbf{z}_i \mid \mathbf{Y}_i, \mathbf{X}_i)$, and use the regularization weight $\alpha = 25$. Regarding the optimization procedure, we employ the Adam [21] optimizer, training for $epc = 40$ epochs with fixed learning rate $lr = 0.003$ and batch size $bs = 128$.

The generative approach allows to leverage the latent information from the unlabeled data and obtain reasonable performances at an annotation effective standpoint. This translates to cheaper and faster development cycles, since cleaning and labeling are by far the toughest and more expensive parts of any ML project in business and industrial applications. Together with active learning, the approach in

which the system presents to the user which samples should be annotated next for optimal performance gain, semi-supervision is a promising and refreshing frontier to the field, with several real-world use cases.

## 5.8 Closing Remarks

In this chapter we discussed the family of VAE models, how the models arise by applying VI to the latent variable modeling, and how they are related to one another. Moreover, we presented their major drawbacks, i.e., inexpressive posterior, posterior collapse, and discrete latent variable, as well as different manners to mitigate them. Many of these issues are current research topics.

In our experiments, we illustrated the generative and inference capabilities of VAEs and their main variations as well as properties of the ELBO and the latent space. Finally, we demonstrated the incredible potential of generative modeling for semi-supervised learning, a learning approach that uses very few labeled examples and takes advantage of the unlabeled samples.

There exist several other types of generative algorithms, the most popular being Generative Adversarial Networks [12], whose training dynamic can also be interpreted as through probabilistic lenses. Recently, pure flow-based models [9, 22] have gained a lot of attention of the research community due to their ability to perform inference and/or generation impressively fast and straightforward training.

As final words, we would link to point out that throughout the book we have seen the value of the Bayesian approach to probabilistic modeling and how it seamlessly allows us to reason under uncertainty, make predictions, and simulate new data, all achieved through the marginalization and conditioning operations. Besides, model fitting and comparison naturally arise from within the framework, which also has the additional advantage of being better equipped to handle data-poor regimes.

Bayesian deep learning is proving to be a central topic in current ML conferences with venues for applications crystallizing by the day: out-of-domain detection, adversarial robustness, compound exploration, audio synthesis, and image super-resolution, among many others. From the start of the writing of this book to the moment of its publication, many new interesting applications and methods came to light, rendering impossible the mission of keeping up with advancements made by the scientific community. However, we hope that our presentation was able to motivate the reader to confidently go further in this field. This is an exciting time to statisticians and ML practitioners alike, as there is a new wave of innovation ahead of us.

## 5.9 Final Words

Throughout the book we have seen the value of the Bayesian approach to probabilistic modeling and how it seamlessly allows us to reason under uncertainty, make predictions, and simulate new data, all achieved through the marginalization and conditioning operations. Besides, model fitting and comparison naturally arise from within the framework, which also has the additional advantage of being better equipped to handle data-poor regimes.

Bayesian DL is proving to be a central topic in current ML conferences with venues for applications crystallizing by the day: out-of-domain detection, adversarial robustness, compound exploration, audio synthesis, and image super-resolution, among many others. From the start of the writing of this book to the moment of its publication, many new interesting applications and methods came to light, rendering impossible the mission of keeping up with advancements made by the scientific community. However, we hope that our presentation was able to motivate the reader to confidently go further in this field. This is an exciting time to statisticians and ML practitioners alike, as there is a new wave of innovation ahead of us.

## References

1. Alemi A, Fischer I, Dillon J, Murphy K (2017) Deep variational information bottleneck. In: Proceedings of the international conference on learning representations, Toulon, France
2. Alemi A, Poole B, Fischer I, Dillon J, Saurous RA, Murphy K (2018) Fixing a broken ELBO. In: Dy J, Krause A (eds) Proceedings of the international conference on machine learning, Stockholm, Sweden, vol 80, pp 159–168
3. Beal MJ, Ghahramani Z (2003) The variational Bayesian EM algorithm for incomplete data: with application to scoring graphical model structures. In: Bayesian Statistics 7: the Seventh Valencia International Meeting, Tenerife, Spain pp. 453–464
4. Blundell C, Cornebise J, Kavukcuoglu K, Wierstra D (2015) Weight uncertainty in neural networks. In: Proceedings of the international conference on machine learning, Lille, France, vol 37, pp 1613–1622
5. Breiman L (2001) Random forests. Machine Learning 45(1):5–32. https://doi.org/10.1023/A:1010933404324
6. Burda Y, Grosse R, Salakhutdinov R (2016) Importance weighted autoencoders. In: Proceedings of the international conference on learning representations, San Juan, Puerto Rico
7. Burgess CP, Higgins I, Pal A, Matthey L, Watters N, Desjardins G, Lerchner A (2018) Understanding disentangling in $\beta$-VAE. arXiv e-prints `1804.03599`
8. Chen J, Chen J, Chao H, Yang M (2018) Image blind denoising with generative adversarial network based noise modeling. In: Proceedings of the conference on computer vision and pattern recognition, Salt Lake City, USA
9. Dinh L, Sohl-Dickstein J, Bengio S (2017) Density estimation using real NVP. In: Proceedings of the international conference on learning representations, Toulon, France
10. Eslami SMA, Heess N, Weber T, Tassa Y, Szepesvari D, kavukcuoglu k, Hinton GE (2016) Attend, infer, repeat: Fast scene understanding with generative models. In: Advances in neural information processing systems, pp 3225–3233

11. Eslami SMA, Jimenez Rezende D, Besse F, Viola F, Morcos AS, Garnelo M, Ruderman A, Rusu AA, Danihelka I, Gregor K, Reichert DP, Buesing L, Weber T, Vinyals O, Rosenbaum D, Rabinowitz N, King H, Hillier C, Botvinick M, Wierstra D, Kavukcuoglu K, Hassabis D (2018) Neural scene representation and rendering. Science 360(6394):1204–1210
12. Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y (2014) Generative adversarial nets. In: Advances in neural information processing systems, Montreal, Canada, pp 2672–2680
13. Gómez-Bombarelli R, Wei JN, Duvenaud D, Hernández-Lobato JM, Sánchez-Lengeling B, Sheberla D, Aguilera-Iparraguirre J, Hirzel TD, Adams RP, Aspuru-Guzik A (2018) Automatic chemical design using a data-driven continuous representation of molecules. ACS Central Sci 4(2):268–276
14. Ha D, Schmidhuber J (2018) Recurrent world models facilitate policy evolution. In: Advances in neural information processing systems, Montreal, Canada, pp 2450–2462
15. He J, Spokoyny D, Neubig G, Berg-Kirkpatrick T (2019) Lagging inference networks and posterior collapse in variational autoencoders. In: Proceedings of the international conference on learning representations, New Orleans, USA
16. Heusel M, Ramsauer H, Unterthiner T, Nessler B, Hochreiter S (2017) GANs trained by a two time-scale update rule converge to a local Nash equilibrium. In: Advances in neural information processing systems, Long Beach, USA, pp 6626–6637
17. Higgins I, Matthey L, Pal A, Burgess C, Glorot X, Botvinick M, Mohamed S, Lerchner A (2017) $\beta$-VAE: Learning basic visual concepts with a constrained variational framework. In: Proceedings of the international conference on learning representations, Toulon, France
18. Houthooft R, Chen X, Chen X, Duan Y, Schulman J, De Turck F, Abbeel P (2016) VIME: Variational information maximizing exploration. In: Advances in neural information processing systems, Barcelona, Spain, pp 1109–1117
19. Jang E, Gu S, Poole B (2017) Categorical reparameterization with Gumbel-softmax. In: Proceedings of the international conference on learning representations, Toulon, France
20. Kalchbrenner N, van den Oord A, Simonyan K, Danihelka I, Vinyals O, Graves A, Kavukcuoglu K (2017) Video pixel networks. In: Proceedings of the international conference on machine learning, Sydney, NSW, Australia, vol 70, pp 1771–1779
21. Kingma DP, Ba J (2015) Adam: A method for stochastic optimization. In: Proceedings of the international conference on learning representations, San Diego, USA
22. Kingma DP, Dhariwal P (2018) Glow: Generative flow with invertible 1x1 convolutions. In: Advances in neural information processing systems, Montreal, Canada, pp 10215–10224
23. Kingma DP, Welling M (2014) Auto-encoding variational Bayes. In: Proceedings of the international conference on learning representations, Banff, Canada
24. Kingma DP, Welling M (2019) An introduction to variational autoencoders. Found Trends Mach Learn 12(4):307–392. https://doi.org/10.1561/2200000056
25. Kingma DP, Mohamed S, Jimenez Rezende D, Welling M (2014) Semi-supervised learning with deep generative models. In: Advances in neural information processing systems, Montreal, Canada, pp 3581–3589
26. Kingma DP, Salimans T, Jozefowicz R, Chen X, Sutskever I, Welling M (2016) Improved variational inference with inverse autoregressive flow. In: Advances in neural information processing systems, Barcelona, Spain, pp 4743–4751
27. Lecun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11):2278–2324
28. Ledig C, Theis L, Huszar F, Caballero J, Cunningham A, Acosta A, Aitken A, Tejani A, Totz J, Wang Z, Shi W (2017) Photo-realistic single image super-resolution using a generative adversarial network. In: Proceedings of the conference on computer vision and pattern recognition, Honolulu, USA
29. Lee AX, Zhang R, Ebert F, Abbeel P, Finn C, Levine S (2018) Stochastic adversarial video prediction. arXiv e-prints `1804.01523`

30. Maaløe L, Sønderby CK, Sønderby SK, Winther O (2016) Auxiliary deep generative models. In: Proceedings of the international conference on machine learning, New York, USA, vol 48, pp 1445–1453
31. Maaten Lvd, Hinton G (2008) Visualizing data using t-SNE. J Mach Learn Res 9:2579–2605
32. Maddison CJ, Tarlow D, Minka T (2014) A* sampling. In: Advances in neural information processing systems, Montreal, Canada, pp 3086–3094
33. Maddison C, Mnih A, Teh YW (2017) The concrete distribution: a continuous relaxation of discrete random variables. In: Proceedings of the international conference on learning representations, Toulon, France
34. McInnes L, Healy J, Melville J (2018) UMAP: Uniform manifold approximation and projection for dimension reduction. arXiv e-prints 1802.03426
35. Migon HS, Gamerman D, Louzada F (2014) Statistical inference: An integrated approach. CRC press, Boca Raton, USA
36. Mnih A, Gregor K (2014) Neural variational inference and learning in belief networks. In: Proceedings of the international conference on machine learning, Bejing, China, vol 32, pp 1791–1799
37. Mnih A, Rezende D (2016) Variational inference for Monte Carlo objectives. In: Proceedings of the international conference on machine learning, New York, USA, vol 48, pp 2188–2196
38. Nair V, Hinton GE (2010) Rectified linear units improve restricted Boltzmann machines. In: Proceedings of the international conference on machine learning, Haifa, Israel, pp 807–814
39. Nowozin S (2018) Debiasing evidence approximations: On importance-weighted autoencoders and jackknife variational inference. In: Proceedings of the international conference on learning representations, Vancouver, Canada
40. Papamakarios G, Pavlakou T, Murray I (2017) Masked autoregressive flow for density estimation. In: Advances in neural information processing systems, Long Beach, USA, pp 2338–2347
41. Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, Desmaison A, Kopf A, Yang E, DeVito Z, Raison M, Tejani A, Chilamkurthy S, Steiner B, Fang L, Bai J, Chintala S (2019) PyTorch: An imperative style, high-performance deep learning library. In: Advances in neural information processing systems, Vancouver, Canada, pp 8024–8035
42. Rainforth T, Kosiorek A, Le TA, Maddison C, Igl M, Wood F, Teh YW (2018) Tighter variational bounds are not necessarily better. In: Proceedings of the international conference on machine learning, Stockholm, Sweden, vol 80, pp 4277–4285
43. Ranganath R, Tran D, Blei D (2016) Hierarchical variational models. In: Proceedings of the international conference on machine learning, New York, USA, vol 48, pp 324–333
44. Regier J, Miller A, McAuliffe J, Adams R, Hoffman M, Lang D, Schlegel D, Prabhat M (2015) Celeste: Variational inference for a generative model of astronomical images. In: Proceedings of the international conference on machine learning, Lille, France, vol 37, pp 2095–2103
45. Rezende D, Mohamed S (2015) Variational inference with normalizing flows. In: Proceedings of the international conference on machine learning, Lille, France, vol 37, pp 1530–1538
46. Riesselman AJ, Ingraham JB, Marks DS (2018) Deep generative models of genetic variation capture the effects of mutations. Nature Methods 15:816–822
47. Rosca M, Lakshminarayanan B, Warde-Farley D, Mohamed S (2017) Variational approaches for auto-encoding generative adversarial networks. arXiv e-prints 1706.04987
48. Salimans T, Goodfellow I, Zaremba W, Cheung V, Radford A, Chen X, Chen X (2016) Improved techniques for training GANs. In: Advances in neural information processing systems, Barcelona, Spain, pp 2234–2242
49. Scholkopf B, Sung KK, Burges CJ, Girosi F, Niyogi P, Poggio T, Vapnik V (1997) Comparing support vector machines with Gaussian kernels to radial basis function classifiers. IEEE Trans Signal Process 45(11):2758–2765
50. Sohn K, Lee H, Yan X (2015) Learning structured output representation using deep conditional generative models. In: Advances in neural information processing systems, Montreal, Canada, pp 3483–3491

51. Sønderby CK, Raiko T, Maaløe L, Sønderby SK, Winther O (2016) Ladder variational autoencoders. In: Advances in neural information processing systems, Barcelona, Spain, pp 3738–3746
52. Sønderby CK, Poole B, Mnih A (2017) Continuous relaxation training of discrete latent variable image models. In: Neural information processing systems - workshop on bayesian deep learning, Long Beach, USA
53. Theis L, Oord Avd, Bethge M (2016) A note on the evaluation of generative models. In: Proceedings of the international conference on learning representations, San Juan, Puerto Rico
54. Tishby N, Pereira FC, Bialek W (2000) The information bottleneck method. arXiv e-prints physics/0004057
55. Tschannen M, Agustsson E, Lucic M (2018) Deep generative models for distribution-preserving lossy compression. In: Advances in neural information processing systems, Montreal, Canada, pp 5929–5940
56. van den Berg R, Hasenclever L, Tomczak J, Welling M (2018) Sylvester normalizing flow for variational inference. In: Proceedings of the international conference on learning representations, Monterey, USA
57. van den Oord A, Dieleman S, Zen H, Simonyan K, Vinyals O, Graves A, Kalchbrenner N, Senior A, Kavukcuoglu K (2016) WaveNet: A generative model for raw audio. In: ISCA speech synthesis workshop, Sunnyvale, USA, pp 125–125
58. van den Oord A, Vinyals O, kavukcuoglu k (2017) Neural discrete representation learning. In: Advances in neural information processing systems, Long Beach, USA, pp 6306–6315
59. Wan L, Zeiler M, Zhang S, Cun YL, Fergus R (2013) Regularization of neural networks using dropconnect. In: Proceedings of the international conference on machine learning, Atlanta, USA, vol 28, pp 1058–1066
60. Xiao H, Rasul K, Vollgraf R (2017) Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms. arXiv e-prints 1708.07747

# Appendix A
# Support Material

In this appendix, we extend on subjects that we deem important to the overall comprehension of the chapters but are only briefly discussed throughout the book. Specifically, we start by explaining the score function and pathwise derivative estimators, crucial to any gradient-based method. We go into the natural gradient, motivating it and showing a rough mathematical derivation. Moreover, we cover the detailed derivations of the

- Coordinate Ascent Variational Inference (CAVI) algorithm, seen in Sect. 3.2.1.4;
- Generalized Gauss-Newton (GGN) approximation used in the derivations of the Practical ADF [3] and Vadam [4] methods for Bayesian Neural Networks (BNNs), seen in Sects. 4.3.1 and 4.6;
- Bonnet's and Price's theorems, known as the Gaussian gradient identities, equally employed on Practical ADF [3] and Vadam [4].

## 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. $\phi$ of the expectation of the function $f(z;\theta)$ under the distribution $q(z;\phi)$, with $\theta$ and $\phi$ 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 $\phi$, though not necessarily of $z$, we derive the *reinforce* or *score function estimator* [8] through

$$\nabla_\phi \mathbb{E}_{q(z;\phi)}[f(z;\theta)] = \nabla_\phi \left[ \int q(z;\phi) f(z;\theta) dz \right]$$



L. P. Cinelli et al., *Variational Methods for Machine Learning with Applications to Deep Networks*, https://doi.org/10.1007/978-3-030-70679-1

$$
\begin{aligned}
&= \int \nabla_\phi \left[q(z;\phi)\right] f(z;\theta) dz \\
&= \int q(z;\phi) \nabla_\phi \left[\log q(z;\phi)\right] f(z;\theta) dz \\
&= \mathbb{E}_{q(z;\phi)} \left[ f(z;\theta) \nabla_\phi \left[\log q(z;\phi)\right]\right]. \qquad \text{(A.1)}
\end{aligned}
$$

Note that the 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 $f(z;\theta)$. 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 differentiable transformation $g(\cdot;\phi)$ of a base random variable $\mathcal{E} \sim p(\epsilon)$, we arrive at the *pathwise derivative estimator* [7]

$$
\begin{aligned}
\nabla_\phi \mathbb{E}_{q(z;\phi)} \left[f(z;\theta)\right] &= \nabla_\phi \left[ \int p(\epsilon) f(g(\epsilon;\phi);\theta) d\epsilon \right] \\
&= \nabla_\phi \left[ \mathbb{E}_{p(\epsilon)} \left[f(g(\epsilon;\phi);\theta)\right]\right] \\
&= \mathbb{E}_{p(\epsilon)} \left[ \nabla_\phi \left[f(g(\epsilon;\phi);\theta)\right]\right]. \qquad \text{(A.3)}
\end{aligned}
$$

This approach requires not only the distribution $q(z;\phi)$ to be reparameterizable but also $f(z;\theta)$ to be known and continuous on $z$ for all values of $\theta$. It is also known as *reparameterization trick* and recently popularized by [5].

While both estimators yield unbiased estimates, the score function estimator generally has higher variance due to the derivative of the log function. This behavior makes sense if we think that the score function estimator computes the derivative only on $q$ and does not include information about the function $f(z;\theta)$, which is the objective function.

## A.2 Update Formula for CAVI

Without resorting to variational calculus, we derive the update equations for the optimal factors of the approximating distribution $q(\mathbf{z} \mid \mathbf{X})$ under the VI algorithm in Sect. 3.2.1.

We rewrite here the formulas for the factorized approximating distribution defined in Eq. (3.16) and the ELBO:

$$q(\mathbf{z} \mid \mathbf{X}) = \prod_{i=1}^{M} q_i(\mathbf{z}_{\mathcal{S}_i} \mid \mathbf{X}), \tag{A.4}$$

$$\text{ELBO}(q) = \mathbb{E}_q\left[\log p(\mathbf{x}, \mathbf{z})\right] - \mathbb{E}_q\left[\log q(\mathbf{z} \mid \mathbf{X})\right]. \tag{A.5}$$

We substitute Eq. (A.4) into Eq. (A.5) and extract the dependence on one of the factors $q_i(\mathbf{z}_{\mathcal{S}_i} \mid \mathbf{X})$. Simplifying the notation from $q_i(\mathbf{z}_{\mathcal{S}_i} \mid \mathbf{X})$ to $q_i$, we get

$$\begin{aligned}
\text{ELBO}(q) &= \int \left(\prod_i q_i\right) \log p(\mathbf{x}, \mathbf{z}) d\mathbf{z} - \int \left(\prod_k q_k\right) \log \prod_l q_l d\mathbf{z} \\
&= \int q_j \left[\int \log p(\mathbf{x}, \mathbf{z}) \prod_{-j} (q_i dz_i)\right] dz_j - \sum_k \int \prod_l q_l \log q_k d\mathbf{z} \\
&= \int q_j \mathbb{E}_{-j}\left[\log p(\mathbf{x}, \mathbf{z})\right] dz_j - \sum_k \int q_k \log q_k \left[\prod_{l \neq k} \int q_l dz_l\right] dz_k,
\end{aligned} \tag{A.6}$$

where the symbol $\mathbb{E}_{-j}[\cdot]$ in the first term denotes expectation with respect to the $q$ distribution over all variables $\mathbf{Z}_{\mathcal{S}_i}$ except for $i = j$.

Since each $q_l$ in the second term is an independent factor, it is normalized such that it sums up to 1. In addition, we define a new distribution $\tilde{p}_{-j}$ such that $\log \tilde{p}_{-j} = \mathbb{E}_{-j}\left[\log p(\mathbf{x}, \mathbf{z})\right] + c$, where the constant term $c$ compensates for the normalization. Then,

$$\begin{aligned}
\text{ELBO}(q) &= \int q_j \log \widetilde{p}_{-j} dz_j - \sum_i \int q_k \log q_k dz_k + c \\
&= \int q_j \log \widetilde{p}_{-j} dz_j - \int q_j \log q_j dz_j - \sum_{k \neq j} \int q_k \log q_k dz_k + c \\
&= \int q_j \log \left(\frac{\widetilde{p}_{-j}}{q_j}\right) dz_j + c \\
&= -D_{KL}(q_j \| \tilde{p}_{-j}) + c.
\end{aligned} \tag{A.7}$$

We keep all $q_{-j}$ fixed and maximize the ELBO w.r.t. to $q_j$. Since the maximum of Eq. (A.7) happens when the KL term is zero, we find the optimal $q^*$ by setting it to be equal to $\tilde{p}_{-j}$

$$q_j^*(\mathbf{z}_{\mathcal{S}_j} \mid \mathbf{X}) = \tilde{p}_{-j}(\mathbf{x}, \mathbf{z}_{\mathcal{S}_j}), \tag{A.8}$$

$$\log q_j^*(\mathbf{z}_{\mathcal{S}_j} \mid \mathbf{X}) = \mathbb{E}_{-j}\left[\log p(\mathbf{x}, \mathbf{z})\right] + c, \tag{A.9}$$

$$q_j^*(\mathbf{z}_{\mathcal{S}_j} \mid \mathbf{X}) \propto \exp\{\mathbb{E}_{-j}\left[\log p(\mathbf{x}, \mathbf{z})\right]\}. \tag{A.10}$$

The $M$ equations (A.10), each for a latent variable set $\mathcal{S}_j$, are coupled. Thus, solving them for the objective function requires an iterative approach. At each step, one replaces a factor $q_j$ by its revised estimate according to (A.10) and does so in a coordinate manner such as round robin.

## A.3 Generalized 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

$$\begin{aligned}
\frac{\partial^2 \ell(f(\mathbf{x}))}{\partial x_j \partial x_i} &= \frac{\partial}{\partial x_j}\left(\frac{\partial \ell(f(\mathbf{x}))}{\partial x_i}\right) \\
&= \frac{\partial}{\partial x_j}\left(\sum_{k=0}^{K} \frac{\partial \ell}{\partial f_k(\mathbf{x})} \frac{\partial f_k(\mathbf{x})}{\partial x_i}\right) \\
&= \sum_{k=0}^{K} \frac{\partial}{\partial x_j}\left(\frac{\partial \ell}{\partial f_k(\mathbf{x})}\right) \frac{\partial f_k(\mathbf{x})}{\partial x_i} + \sum_{k=0}^{K} \frac{\partial \ell}{\partial f_k(\mathbf{x})} \frac{\partial^2 f_k(\mathbf{x})}{\partial x_j \partial x_i} \\
&= \sum_{k=0}^{K} \sum_{m=0}^{K} \left(\frac{\partial^2 \ell}{\partial f_m(\mathbf{x}) f_k(\mathbf{x})}\right) \frac{\partial f_m(\mathbf{x})}{\partial x_j} \frac{\partial f_k(\mathbf{x})}{\partial x_i} + \sum_{k=0}^{K} \frac{\partial \ell}{\partial f_k(\mathbf{x})} \frac{\partial^2 f_k(\mathbf{x})}{\partial x_j \partial x_i}.
\end{aligned} \tag{A.11}$$

The first term in Eq. (A.11) 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. Thus, we have

$$\frac{\partial^2 \ell(f(\mathbf{x}))}{\partial x_j \partial x_i} \approx \sum_{k=0}^{K} \sum_{m=0}^{K} \left(\frac{\partial^2 \ell}{\partial f_m(\mathbf{x}) f_k(\mathbf{x})}\right) \frac{\partial f_m(\mathbf{x})}{\partial x_j} \frac{\partial f_k(\mathbf{x})}{\partial x_i} \triangleq G_{ij}. \tag{A.12}$$

The resulting approximation is what we call the Generalized Gauss–Newton and is exact when the loss $\ell = 0$, since the term $\partial \ell / \partial f_k(\mathbf{x})$ in Eq. (A.11) becomes zero. However, the approximation gets increasingly worse at larger $\ell$.

Writing the GGN in matrix form gives

$$G = J_f(\mathbf{x})^T H_\ell(f(\mathbf{x})) J_f(\mathbf{x}), \tag{A.13}$$

which is always positive semidefinite. In the particular case where $\ell(f(\mathbf{x})) = -\log f(\mathbf{x})$, it becomes

$$G = J_f^T(\mathbf{x}) \frac{1}{f(\mathbf{x})^2} J_f(\mathbf{x}) = \nabla_{\mathbf{x}} f(\mathbf{x}) \nabla_{\mathbf{x}}^T f(\mathbf{x}). \tag{A.14}$$

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 [2].

## A.4 Natural Gradient and the Fisher Information Matrix

We motivate the natural gradient and explain how it appears in our optimization context. The derivation closely follows [6]. For brevity, we use the notation $p_{\boldsymbol{\psi}}$ instead of the usual $p(\cdot; \boldsymbol{\psi})$ to indicate a family of distributions $p$ parameterized by $\boldsymbol{\psi}$.

Ideally, we would like the update step in the gradient descent algorithm to have constant velocity through training. However, it can vary abruptly at each iteration, what slows down and can even hamper optimization. Neither clipping nor fixing its magnitude guarantees a clear limit on the change induced in the model. Furthermore, the gradient depends on the coordinate system, so constraining its norm amounts to different restrictions in different coordinate systems. While small changes in the parameters have large effects on the probability represented by the model for some distributions, for others it is the opposite. If we wish to move along the distribution manifold with constant speed, regardless of its curvature, we must measure distance in the distribution space and constrain it.

Suppose we want to minimize the loss function $\mathcal{L}$ w.r.t. the distribution $p_{\boldsymbol{\psi}}(\mathbf{z})$. Hence, at each iteration we wish to find the step

$$\underset{\delta\boldsymbol{\psi}}{\operatorname{argmin}}\, \mathcal{L}(\boldsymbol{\psi} + \delta\boldsymbol{\psi}) \tag{A.15}$$

$$\text{s.t. } D_{KL}(p_{\boldsymbol{\psi}} \| p_{\boldsymbol{\psi}+\delta\boldsymbol{\psi}}) = c, \tag{A.16}$$

where $c$ is a constant term. The KL divergence constraint assures that the distribution space changes by a constant value. Although not a proper metric due to its asymmetry, KL divergence is asymptotically symmetric for $\delta\boldsymbol{\psi} \to 0$, what enables us to use it nonetheless. For small enough $\delta\boldsymbol{\psi}$, we can approximate the KL divergence around the vicinity of $\boldsymbol{\psi}$ by its Taylor expansion up to the second order, like

$$\begin{aligned} D_{KL}\left(p_{\boldsymbol{\psi}} \| p_{\boldsymbol{\psi}+\delta\boldsymbol{\psi}}\right) \approx D_{KL}\left(p_{\boldsymbol{\psi}} \| p_{\boldsymbol{\psi}}\right) + \delta\boldsymbol{\psi}\ \nabla_{\boldsymbol{\psi}'} D_{KL}\left(p_{\boldsymbol{\psi}} \| p_{\boldsymbol{\psi}'}\right)\big|_{\boldsymbol{\psi}'=\boldsymbol{\psi}} \\ + \frac{1}{2}\delta\boldsymbol{\psi}^T\ H(\boldsymbol{\psi}')\delta\boldsymbol{\psi}\big|_{\boldsymbol{\psi}'=\boldsymbol{\psi}}, \end{aligned} \tag{A.17}$$

where $H(\boldsymbol{\psi}')$ is the shorthand for the Hessian of the $D_{KL}$ computed at $\boldsymbol{\psi}'$, whose adequate formula is

$$H(\boldsymbol{\psi}') = \nabla^2_{\boldsymbol{\psi}'} D_{KL}\left(p_{\boldsymbol{\psi}} \| p_{\boldsymbol{\psi}'}\right) = -\nabla^2_{\boldsymbol{\psi}'} \mathbb{E}\left[\log p_{\boldsymbol{\psi}'}\right]. \tag{A.18}$$

At $\boldsymbol{\psi}' = \boldsymbol{\psi}$, the KL divergence is at the minimum, so both the first and second right-hand terms in Eq. (A.17) vanish and only the second order term remains. Then, the Hessian is given by

$$\begin{aligned}
H(\boldsymbol{\psi}) &= -\nabla^2_{\boldsymbol{\psi}} \mathbb{E}\left[\log p_{\boldsymbol{\psi}}\right] \\
&= \mathbb{E}\left[-\nabla^2_{\boldsymbol{\psi}} \log p_{\boldsymbol{\psi}}\right] \\
&= \mathbb{E}_{p_{\boldsymbol{\psi}}}\left[-\nabla_{\boldsymbol{\psi}}\left[\frac{1}{p_{\boldsymbol{\psi}}} \nabla_{\boldsymbol{\psi}} p^T_{\boldsymbol{\psi}}\right]\right] \\
&= \mathbb{E}_{p_{\boldsymbol{\psi}}}\left[\frac{1}{p^2_{\boldsymbol{\psi}}} \nabla_{\boldsymbol{\psi}} p_{\boldsymbol{\psi}} \nabla_{\boldsymbol{\psi}} p^T_{\boldsymbol{\psi}}\right] - \mathbb{E}_{p_{\boldsymbol{\psi}}}\left[\frac{1}{p_{\boldsymbol{\psi}}} \nabla^2_{\boldsymbol{\psi}} p_{\boldsymbol{\psi}}\right] \\
&= \mathbb{E}\left[\nabla_{\boldsymbol{\psi}} \log p_{\boldsymbol{\psi}} \nabla_{\boldsymbol{\psi}} \log p^T_{\boldsymbol{\psi}}\right] - \int \nabla^2_{\boldsymbol{\psi}} p_{\boldsymbol{\psi}} d\mathcal{Z} \\
&= \mathbb{E}\left[\nabla_{\boldsymbol{\psi}} \log p_{\boldsymbol{\psi}} \nabla_{\boldsymbol{\psi}} \log p^T_{\boldsymbol{\psi}}\right] \\
&= \mathbb{E}\left[\nabla_{\boldsymbol{\psi}} \log p_{\boldsymbol{\psi}} \nabla_{\boldsymbol{\psi}} \log p^T_{\boldsymbol{\psi}}\right] - \underbrace{\mathbb{E}\left[\nabla_{\boldsymbol{\psi}} \log p_{\boldsymbol{\psi}}\right]}_{=0} \mathbb{E}\left[\nabla_{\boldsymbol{\psi}} \log p^T_{\boldsymbol{\psi}}\right] \\
&= \mathrm{Cov}(\nabla_{\boldsymbol{\psi}} \log p_{\boldsymbol{\psi}}, \nabla_{\boldsymbol{\psi}} \log p_{\boldsymbol{\psi}}) \\
&= \mathcal{I}(\boldsymbol{\psi}),
\end{aligned} \tag{A.19}$$

where $\mathcal{I}(\boldsymbol{\psi})$ is the Fisher information matrix and the last equality comes from its definition, as seen in Sect. 2.3.1.

From Eq. (A.19), 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 Eqs. (A.17) and (A.19), we can write the Lagrangian form of Eq. (A.15) as

$$\underset{\delta\boldsymbol{\psi}}{\mathrm{argmin}}\, \mathcal{L}(\boldsymbol{\psi} + \delta\boldsymbol{\psi}) + \lambda\left(\frac{1}{2}\delta\boldsymbol{\psi}^T \mathcal{I}(\boldsymbol{\psi})\delta\boldsymbol{\psi} - \mathrm{const}\right). \tag{A.20}$$

If we further assume that the linearization of $\mathcal{L}(\boldsymbol{\psi} + \delta\boldsymbol{\psi})$ around the vicinity of $\boldsymbol{\psi}$ is valid, we obtain

$$\underset{\delta\boldsymbol{\psi}}{\operatorname{argmin}}\, \mathcal{L}(\boldsymbol{\psi}) + \nabla_{\boldsymbol{\psi}}\mathcal{L}(\boldsymbol{\psi})^T\delta\boldsymbol{\psi} + \lambda\left(\frac{1}{2}\delta\boldsymbol{\psi}^T\mathcal{I}(\boldsymbol{\psi})\delta\boldsymbol{\psi} - \text{const}\right). \tag{A.21}$$

Finally, setting the gradient w.r.t. $\delta\boldsymbol{\psi}$ equal to zero in order to solve the problem, we arrive at

$$0 = \nabla_{\boldsymbol{\psi}}\mathcal{L}(\boldsymbol{\psi}) + \lambda\mathcal{I}(\boldsymbol{\psi})\delta\boldsymbol{\psi} \Rightarrow \delta\boldsymbol{\psi} = -k\widetilde{\nabla}_{\boldsymbol{\psi}}\mathcal{L}(\boldsymbol{\psi}), \tag{A.22}$$

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.

We have thus obtained an algorithm that not only is robust to one-to-one reparameterizations (accounted for in the Fisher matrix), and moves along the manifold with constant speed, but also follows the steepest descent direction [1].

## A.5 Gaussian Gradient Identities

Here, we review the derivations of Bonnet's and Price's theorems, (4.9) and (4.10), respectively. Before these proofs, we derive two other intermediary useful results.

First, given a multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu}, \mathbf{C})$, where $dim(\boldsymbol{\xi}) = d$, the gradient with respect to $\mu_i$ can be rewritten as

$$\begin{aligned}
\nabla_{\mu_i}\mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu}, \mathbf{C}) &= \frac{\partial\left((2\pi)^{-d/2}|\mathbf{C}|^{-1/2}e^{-\frac{1}{2}(\boldsymbol{\xi}-\boldsymbol{\mu})^T\mathbf{C}^{-1}(\boldsymbol{\xi}-\boldsymbol{\mu})}\right)}{\partial\mu_i} \\
&= (2\pi)^{-d/2}|\mathbf{C}|^{-1/2}e^{-\frac{1}{2}(\boldsymbol{\xi}-\boldsymbol{\mu})^T\mathbf{C}^{-1}(\boldsymbol{\xi}-\boldsymbol{\mu})}\frac{\partial\left(-\frac{1}{2}(\boldsymbol{\xi}-\boldsymbol{\mu})^T\mathbf{C}^{-1}(\boldsymbol{\xi}-\boldsymbol{\mu})\right)}{\partial\mu_i} \\
&= \mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu}, \mathbf{C})\left(\sum_{k=1}^{d}(\xi_k - \mu_k)l_{ik}\right),
\end{aligned} \tag{A.23}$$

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

$$\nabla_{\mu_i}\mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu}, \mathbf{C}) = -\nabla_{\xi_i}\mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu}, \mathbf{C}). \tag{A.24}$$

Second, we obtain a relation on the derivative of the Gaussian w.r.t. its covariance matrix elements:

$$\nabla_{c_{i,j}}\mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu}, \mathbf{C}) = \frac{\partial\left((2\pi)^{-d/2}|\mathbf{C}|^{-1/2}e^{-\frac{1}{2}(\boldsymbol{\xi}-\boldsymbol{\mu})^T\mathbf{C}^{-1}(\boldsymbol{\xi}-\boldsymbol{\mu})}\right)}{\partial c_{i,j}}$$

$$
\begin{aligned}
&= (2\pi)^{-d/2}|\mathbf{C}|^{-1/2}e^{-\frac{1}{2}(\boldsymbol{\xi}-\boldsymbol{\mu})^T\mathbf{C}^{-1}(\boldsymbol{\xi}-\boldsymbol{\mu})}\frac{\partial\left(-\frac{1}{2}(\boldsymbol{\xi}-\boldsymbol{\mu})^T\mathbf{C}^{-1}(\boldsymbol{\xi}-\boldsymbol{\mu})\right)}{\partial c_{i,j}} \\
&\quad + (2\pi)^{-d/2}e^{-\frac{1}{2}(\boldsymbol{\xi}-\boldsymbol{\mu})^T\mathbf{C}^{-1}(\boldsymbol{\xi}-\boldsymbol{\mu})}\frac{\partial|\mathbf{C}|^{-1/2}}{\partial c_{i,j}} \\
&= \mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu},\mathbf{C})\left(\frac{1}{2}(\boldsymbol{\xi}-\boldsymbol{\mu})^T\mathbf{C}^{-1}\frac{\partial\mathbf{C}}{\partial c_{i,j}}\mathbf{C}^{-1}(\boldsymbol{\xi}-\boldsymbol{\mu})\right) \\
&\quad + (2\pi)^{-d/2}e^{-\frac{1}{2}(\boldsymbol{\xi}-\boldsymbol{\mu})^T\mathbf{C}^{-1}(\boldsymbol{\xi}-\boldsymbol{\mu})}\left(-\frac{1}{2}|\mathbf{C}|^{-3/2}|\mathbf{C}|\mathrm{tr}\left(\mathbf{C}^{-1}\frac{\partial\mathbf{C}}{\partial c_{i,j}}\right)\right) \\
&= -\frac{1}{2}\mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu},\mathbf{C})\left(-\sum_{k_1=1}^{d}\left((\xi_{k_1}-\mu_{k_1})l_{i,k_1}\sum_{k_2=1}^{d}(\xi_{k_2}-\mu_{k_2})l_{j,k_2}\right)+l_{i,j}\right).
\end{aligned}
\tag{A.25}
$$

Now, taking the derivative of $\nabla_{\xi_i}\mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu},\mathbf{C})$ w.r.t. $\xi_j$, we have

$$
\begin{aligned}
\nabla_{\xi_i,\xi_j}\mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu},\mathbf{C}) &= -\frac{\partial\mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu},\mathbf{C})}{\partial\xi_j}\left(\sum_{k=1}^{d}(\xi_k-\mu_k)l_{i,k}\right)-\mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu},\mathbf{C})l_{i,j} \\
&= -\mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu},\mathbf{C})\left(l_{i,j}-\left(\sum_{k=1}^{d}(\xi_k-\mu_k)l_{i,k}\right)\left(\sum_{k=1}^{d}(\xi_k-\mu_k)l_{j,k}\right)\right),
\end{aligned}
\tag{A.26}
$$

which implies on

$$
\nabla_{c_{i,j}}\mathcal{N}(\boldsymbol{\xi}|\mu,\mathbf{C}) = \frac{1}{2}\nabla_{\xi_i,\xi_j}\mathcal{N}(\boldsymbol{\xi}|\mu,\mathbf{C}). \tag{A.27}
$$

**Theorem A.1 (Bonnet's Theorem)** *Let $f(\boldsymbol{\xi}) : R^d \mapsto R$ be an 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 $f(\boldsymbol{\xi})$, that is,*

$$
\nabla_{\mu_i}\mathbb{E}_{\mathcal{N}(\mu,\mathbf{C})}\left[f(\boldsymbol{\xi})\right] = \mathbb{E}_{\mathcal{N}(\boldsymbol{\mu},\mathbf{C})}\left[\nabla_{\xi_i}f(\boldsymbol{\xi})\right]. \tag{A.28}
$$

***Proof***

$$
\begin{aligned}
\nabla_{\mu_i}\mathbb{E}_{\mathcal{N}(\boldsymbol{\mu},\mathbf{C})}\left[f(\boldsymbol{\xi})\right] &= \int\nabla_{\mu_i}\mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu},\mathbf{C})f(\boldsymbol{\xi})d\boldsymbol{\xi} \\
&= -\int\nabla_{\xi_i}\mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu},\mathbf{C})f(\boldsymbol{\xi})d\boldsymbol{\xi}
\end{aligned}
$$

$$
\begin{aligned}
&= -\int \nabla_{\xi_i} \left(\mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu}, \mathbf{C}) f(\boldsymbol{\xi})\right) d\boldsymbol{\xi} \\
&\quad + \int \mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu}, \mathbf{C}) \nabla_{\xi_i} f(\boldsymbol{\xi}) d\boldsymbol{\xi} \\
&= -\int_{\xi_1} \cdots \int_{\xi_n} \underbrace{\int_{\xi_i} \nabla_{\xi_i} \left(\mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu}, \mathbf{C}) f(\boldsymbol{\xi})\right) d\xi_i}_{=[\mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu}, \mathbf{C}) f(\boldsymbol{\xi})]_{\xi_i=-\infty}^{\xi_i=+\infty}} d\xi_n \cdots d\xi_1 \\
&\quad + \int \mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu}, \mathbf{C}) \nabla_{\xi_i} f(\boldsymbol{\xi}) d\boldsymbol{\xi} \\
&= \left[\int \mathcal{N}(\boldsymbol{\xi}|\boldsymbol{\mu}, \mathbf{C}) f(\boldsymbol{\xi}) d\boldsymbol{\xi}_{\neg i}\right]_{\xi_i=-\infty}^{\xi_i=+\infty} + \mathbb{E}_{\mathcal{N}(\boldsymbol{\mu}, \mathbf{C})}\left[\nabla_{\xi_i} f(\boldsymbol{\xi})\right] \\
&= \mathbb{E}_{\mathcal{N}(\boldsymbol{\mu}, \mathbf{C})}\left[\nabla_{\xi_i} f(\boldsymbol{\xi})\right], \qquad \text{(A.29)}
\end{aligned}
$$

where we have used the identity (A.24) in moving from step 1 to 2, and the product rule for derivatives from step 2 to 3. In moving from step 3 to 4, we have rewritten the first term. At the last step, we eliminated the first term, which equals zero. □

**Theorem A.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*

$$
\nabla_{C_{i,j}} \mathbb{E}_{\mathcal{N}(\mathbf{0}, C)}\left[f(\boldsymbol{\xi})\right] = \frac{1}{2} \mathbb{E}_{\mathcal{N}(\mathbf{0}, C)}\left[\nabla_{\xi_i, \xi_j} f(\boldsymbol{\xi})\right]. \qquad \text{(A.30)}
$$

***Proof***

$$
\begin{aligned}
\nabla_{C_{i,j}} \mathbb{E}\left[\mathcal{N}(\mathbf{0}, \mathbf{C})\right]\left[f(\boldsymbol{\xi})\right] &= \int \nabla_{C_{i,j}} \mathcal{N}(\boldsymbol{\xi}|\mathbf{0}, \mathbf{C}) f(\boldsymbol{\xi}) d\boldsymbol{\xi} \\
&= \frac{1}{2} \int \nabla_{\xi_i, \xi_j} \mathcal{N}(\boldsymbol{\xi}|\mathbf{0}, \mathbf{C}) f(\boldsymbol{\xi}) d\boldsymbol{\xi} \\
&= \frac{1}{2} \int \mathcal{N}(\boldsymbol{\xi}|\mathbf{0}, \mathbf{C}) \nabla_{\xi_i, \xi_j} f(\boldsymbol{\xi}) d\boldsymbol{\xi} \\
&= \frac{1}{2} \mathbb{E}_{\mathcal{N}(\mathbf{0}, \mathbf{C})}\left[\nabla_{\xi_i, \xi_j} f(\boldsymbol{\xi})\right]. \qquad \text{(A.31)}
\end{aligned}
$$

In moving from step 1 to 2, we have used the identity (A.27). From step 2 to 3, we have used the product rule for integrals twice. □

## A.6 *t*-Student Distribution

Here we elaborate on a result that we used on more than one occasion throughout the book. Consider $w$ randomly sampled from a normal distribution with zero mean and precision $\lambda$, such as

$$p(w \mid \lambda) = \left(\frac{\lambda}{2\pi}\right)^{\frac{1}{2}} \exp\left(-\lambda \frac{w^2}{2}\right) \tag{A.32}$$

and consider that $\lambda$ is sampled from a Gamma distribution with parameters $\alpha_0$ and $\beta_0$

$$p(\lambda\,;\beta_0,\alpha_0) = \frac{\beta_0^{\alpha_0}}{\Gamma(\alpha_0)}\lambda^{\alpha_0-1}\exp\left(-\lambda\beta_0\right), \tag{A.33}$$

we can marginalize the distribution of $w$ so it only depends on $\alpha_0$ and $\beta_0$, as follows:

$$\begin{aligned}
p(w \mid \beta_0,\alpha_0) &= \int_0^\infty p(w \mid \lambda)\, p(\lambda\,;\beta_0,\alpha_0) d\lambda \\
&= \int_0^\infty \left[\frac{\beta_0^{\alpha_0}}{\Gamma(\alpha_0)}\lambda^{\alpha_0-1}\exp\left(-\lambda\beta_0\right)\right]\left[\left(\frac{\lambda}{2\pi}\right)^{\frac{1}{2}}\exp\left(-\lambda\frac{w^2}{2}\right)\right] d\lambda \\
&= (2\pi)^{-\frac{1}{2}}\frac{\beta_0^{\alpha_0}}{\Gamma(\alpha_0)}\int_0^\infty \lambda^{(\alpha_0+\frac{1}{2})-1}\exp\left[-\lambda\left(\beta_0+\frac{w^2}{2}\right)\right] d\lambda \\
&= (2\pi)^{-\frac{1}{2}}\frac{\Gamma\left(\alpha_0+\frac{1}{2}\right)}{\Gamma(\alpha_0)}\beta_0^{\alpha_0}\left(\beta_0+\frac{w^2}{2}\right)^{-\left(\alpha_0+\frac{1}{2}\right)} \\
&\qquad \times \int_0^\infty \mathrm{Ga}\left(\lambda \,\middle|\, \alpha_0+\frac{1}{2}, \beta_0+\frac{w^2}{2}\right) d\lambda \\
&= (2\pi)^{-\frac{1}{2}}\frac{\Gamma\left(\alpha_0+1/2\right)}{\Gamma(\alpha_0)}\beta_0^{\alpha_0}\left(\beta_0+\frac{w^2}{2}\right)^{-\left(\alpha_0+\frac{1}{2}\right)} \\
&= \frac{\Gamma\left(\alpha_0+1/2\right)}{\Gamma(\alpha_0)}(2\pi\beta_\lambda)^{-\frac{1}{2}}\left(1+\frac{w^2}{2\beta_0}\right)^{-\left(\alpha_0+\frac{1}{2}\right)}.
\end{aligned} \tag{A.34}$$

Compare Eq. (A.34) to the location-scale family for the student's $t$-distribution (parameterized in terms of the inverse scaling parameter $\lambda$)

$$\mathcal{T}_\nu(x \mid \mu,\lambda) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\Gamma\left(\frac{\nu}{2}\right)}\left(\frac{\lambda}{\pi\nu}\right)^{\frac{1}{2}}\left(1+\frac{\lambda(x-\mu)^2}{\nu}\right)^{-\frac{\nu+1}{2}}, \tag{A.35}$$

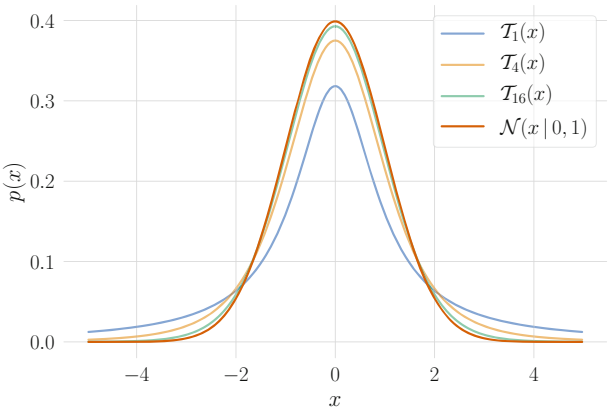



**Fig. A.1** PDF of the standard Student's t-distribution $\mathcal{T}_\nu(x)$ for different values of the parameter $\nu$. The higher the value of $\nu$, the closer (in the KL divergence sense) the distribution becomes to the Gaussian distribution. The standard family member has the location $\mu$ and inverse scale $\lambda$ parameters of Eq. (A.35) equal to 0 and 1, respectively

whose mean and variance are $\mathbb{E}[X] = \mu$, and $\text{Var}(X) = \frac{1}{\lambda}\frac{\nu}{\nu-2}$, 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 Fig. A.1.

## References

1. Amari Si (1998) Natural gradient works efficiently in learning. Neural Computation 10(2):251–276
2. Bottou L, Curtis FE, Nocedal J (2018) Optimization methods for large-scale machine learning. SIAM Review 60(2):223–311. https://doi.org/10.1137/16M1080173
3. Graves A (2011) Practical variational inference for neural networks. In: Advances in neural information processing systems, Granada, Spain, pp 2348–2356
4. Khan M, Nielsen D, Tangkaratt V, Lin W, Gal Y, Srivastava A (2018) Fast and scalable Bayesian deep learning by weight-perturbation in Adam. In: Proceedings of the international conference on machine learning, Stockholm, Sweden, vol 80, pp 2611–2620
5. Kingma DP, Welling M (2014) Auto-encoding variational Bayes. In: Proceedings of the international conference on learning representations, Banff, Canada
6. Pascanu R, Bengio Y (2014) Revisiting natural gradient for deep networks. In: Proceedings of the international conference on learning representations, Banff, Canada
7. Price R (1958) A useful theorem for nonlinear devices having Gaussian inputs. Trans Inf Theory 4(2):69–72
8. Williams RJ (1992) Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning 8(3):229–256

# Index





L. P. Cinelli et al., *Variational Methods for Machine Learning with Applications to Deep Networks*, https://doi.org/10.1007/978-3-030-70679-1