Variational Inference and Stochastic Optimization for Document Modelling

by

Sakif Hossain Khan

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computational Mathematics

Waterloo, Ontario, Canada, 2019

Abstract

Document modelling is the part of natural language processing (NLP) concerned with the automatic discovery and extraction of abstract semantic structure within a document or a corpus of documents. Contemporary document models are often generative statistical models and the trend can be traced back to the seminal paper [Blei et al., 2003], which presents Latent Dirichlet Allocation (LDA). The LDA model is an example of a latent variable model in which variational inference is employed for inference of the latent variables and estimation of model parameters. Since the publication of that paper, there has been significant progress in development and application of variational inference methods. This has been driven in large part by breakthroughs combining deep learning with variational methods. In this paper, we explore the utility of applying deep variational inference methods to the task of document modeling. A good portion of our work surveys long-standing approaches along with contemporary tools driving current innovation in this research area. Towards the end, we present our own extension of a modern unsupervised generative model of text data.

Acknowledgements

I would like to acknowledge the support of my supervisor Dr. Ali Ghodsi. Many thanks to Dr. Ghodsi for letting me freely pursue my research interests as well his valuable contributions to my career outside of the University of Waterloo. I am extremely thankful to the second reader of my essay, Dr. Yeying Zhu, for volunteering her time. I appreciate Dr. Zhu contributing her time to reading the essay and attending my presentation.

I am also very grateful to all administrative and faculty personnel involved in the Computational Mathematics program for all the hard work they do every day to keep things running smoothly. Amanda Guderian, Chérisse Mike, Dr. Kevin Hare, Dr. Jeff Orchard and Dr. Henry Wolkowicz have been specially helpful throughout my time at the department.

I would like to thank Kira Selby and Nabiha Asghar for several valuable discussions during the early stages of my final project. Relatedly, Nicole Dumont and Kira Selby provided much appreciated friendship in Waterloo.

Lastly, Wanying Zhang has been my favorite companion through distances in both space and time. This thesis is dedicated to her for her brains, beauty, humor, grace and love.

Dedication

To Wanying Zhang

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In machine learning and NLP, document modeling aims to provide probabilistic models for text corpora in order to compute parsimonious representations for documents within a given corpus. Such representations are important since they enable automated and scalable text processing while also facilitating other machine learning task like document classification, text summarization, etc. Moreover, if the document model is generative, it gives an estimate of the probability of the observed texts while simultaneously providing a procedure for sampling synthetic texts which accord with texts seen in the training data. Latent Dirichlet Allocation (LDA) exemplifies the approach whereby a generative probabilistic model is defined for a corpus. The model is then trained by (approximate) maximum likelihood in an unsupervised fashion. We shall first describe the LDA model since it provides an useful jumping-off point for the work we present in this essay.

1.1 Latent Dirichlet Allocation

As stated in the preceding, LDA provides a generative model of text at the corpus level. More precisely, the abstract structure inferred by LDA is the set of topics inherent in a given document. To make sense of this statement, it is useful to think of documents existing within a hierarchy as follows.

(1) A corpus consists of several documents, i.e., if we generate M documents from our model, we get a corpus.

(2) A document consists of several words, i.e., if we generate N words from our model, we get a document.

(3) Each word has a latent (or hidden) topic associated with it, i.e., if we wish to generate N words, we must first generate N (independent) topics and each topic then generates a word. Additionally, the topics are sampled from a finite mixture of latent topics.

Each document is treated as a bag of words but, as detailed below, our model will contain parameters shared between documents within a corpus. The LDA model can be represented using the graphical model in figure 1.1.

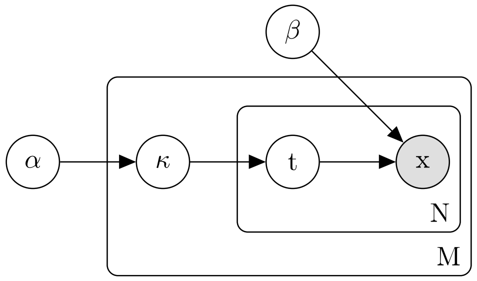


Figure 1.1: Graphical model for Latent Dirichlet Allocation.

The notation is as follows: $\mathbf{x}$ denotes a document, $\mathbf{t}$ the latent topics with t_n being the latent topic for word x_n, κ denotes the topic proportions for document $\mathbf{x}$ and α, β are model parameters. We define auxiliary variables

$$\theta := [\alpha, \beta]$$
$$\mathbf{z} := [\kappa, \mathbf{t}]$$

Observe that θ is shared across documents whereas $\mathbf{z}$ is defined per document. There are two computational tasks. Firstly, we must perform inference, i.e., deduce

$$\Pr(\kappa, \mathbf{t} \mid \mathbf{x}; \alpha, \beta) = \Pr(\mathbf{z} \mid \mathbf{x}; \theta) =: p_\theta(\mathbf{z} \mid \mathbf{x})$$

and secondly, we must estimate the model parameters θ. The quantity $p_\theta(\mathbf{z} \mid \mathbf{x})$ is typically an intractable integral and we must perform approximate inference. To this end, we introduce a surrogate or approximate posterior

$$q_\lambda(\mathbf{z} \mid \mathbf{x}) \approx p_\theta(\mathbf{z} \mid \mathbf{x}),$$

where λ are the variational parameters. We then have the ELBO

$$\mathcal{L}(\theta, \lambda; \mathbf{x}) := \mathbf{E}_{q_\lambda} \left[\log p_\theta(\mathbf{x} \mid \mathbf{z}) - \log q_\lambda(\mathbf{z} \mid \mathbf{x}) + \log p_\theta(\mathbf{z})\right)]$$

Note that the ELBO is completely generic and, in the form we have written it, consists of three ingredients: the likelihood $p_\theta(\mathbf{x} \mid \mathbf{z})$, the surrogate posterior $q_\lambda(\mathbf{z} \mid \mathbf{x})$ and the prior $p_\theta(\mathbf{z})$. Moreover, we have not yet assumed anything about the forms of the various conditional distributions above (except, of course, that each of the distributions is parametrized).

The LDA model obtains under the following assumptions.

(a) $\kappa \sim \text{Dir}(\alpha)$;

(b) $t_n \sim \text{Multi}(\kappa)$;

(c) $x_n \sim \text{Multi}(t_n; \beta)$.

Additionally, when carrying out the variational inference, LDA assumes a factorized surrogate posterior

$$q_\lambda(\mathbf{z} \mid \mathbf{x}) = q_\gamma(\kappa \mid \mathbf{x}) \cdot q_\delta(\mathbf{t} \mid \mathbf{x}),$$

where $\lambda = [\gamma, \delta]$, $q_\gamma(\kappa \mid \mathbf{x})$ is $\text{Dir}(\gamma)$ and $q_\delta(\mathbf{t} \mid \mathbf{x})$ is a product of the multinomials $\text{Multi}(\delta_n)$. These ingredients together constitute a model in which the ELBO can be computed exactly given concrete values for θ and λ and the ELBO can be optimized by gradient descent since the gradients too may be computed exactly. This provides a set of exact update equations for computing optimal values for θ and λ as detailed in Appendix A of [Blei et al., 2003]. Note, however, that since θ is the set of model parameters while λ is the set of variational parameters, the two sets of variables must be updated in an alternating fashion rather than simultaneously per gradient descent step.

1.2 Black-Box Document Model

As should be clear from the foregoing description of inference and estimation in the LDA model, achieving a model in which we can perform exact updates during optimization requires a series of carefully chosen assumptions about the form of the generative model itself along with correspondent assumptions about the surrogate posterior. However, suppose we wish to obtain a richer model for text corpora. We can retain the graphical model which LDA respects but we may need to either choose different parametrizations (i.e., different families of probability distributions) for some of the conditional distributions in the graphical model or we might choose a richer family of distributions for our surrogate posterior. This observation suggests a generic or "black-box" document model which obeys the equations for the ELBO stated above but which no longer needs to accord with the assumptions of LDA. In general, such deviations from LDA assumptions will result in an ELBO which can no longer be computed analytically or whose gradients may not induce closed-form update equations for gradient-based optimization (most likely both).

However, if we can design our overall model so that the ELBO is the output of a neural network, with the implication that θ and λ are parameters of neural networks, then gradient-based optimization of the ELBO reduces to back-propagation. Thus, if we choose to parametrize the probability distributions involved using neural networks, we obtain a document model for which learning is still possible despite analytic intractability of the objective function and its gradients. Furthermore, the latent variables can be understood to represent an abstract parsimonious representation rather than simply topics. We shall elaborate on this distinction in our discussion of models in the sequel.

Going back to our earlier point that a variational inference procedure is specified exactly by choices of likelihood, surrogate posterior and prior, we can construct expressive document models by parametrizing any or all of these three components using neural networks. We shall start exploring this idea in Chapter 4, prior to which Chapters 2 and 3 lay out the necessary tools from current literature on variational inference.

Chapter 2

Primer on Variational Inference

This chapter provides a standalone presentation of basic concepts of variational techniques as well as some recent technical advances in variational inference. We draw heavily from [Jaakkola and Jordan, 1999], [Blei et al., 2016], [Ranganath et al., 2014], [Zhang et al., 2017]. We recommend perusing [Zhang et al., 2017] for a broader survey of recent innovations in VI. We shall start with a conceptual overview of what variational inference aims to do. This discussion will lead us to the key quantity of interest in VI, namely, the ELBO. We then derive expressions for the ELBO and its gradients. Subsequently, we explore issues in computing the ELBO (or its gradients) exactly which provides a segue to stochastic VI.

2.1 Variational Inference Basics

We start by considering a probabilistic model with latent variables. That is, we have a joint density

$$p_\theta(\mathbf{z}, \mathbf{x}) \coloneqq \Pr(\mathbf{z}, \mathbf{x} \mid \theta)$$

over latent variables $\mathbf{z}$ and $\mathbf{x}$. Here, θ is the set of model parameters. We think of the latent variables as unobserved influences on the distribution of the observations. Recall that the joint density can be written as

$$p_\theta(\mathbf{z}, \mathbf{x}) = p_\theta(\mathbf{z}) \cdot p_\theta(\mathbf{x} \mid \mathbf{z}),$$

where $p_\theta(\mathbf{z})$ is a prior density over latent variables specified by our modeling assumptions and $p_\theta(\mathbf{x} \mid \mathbf{z})$ is the likelihood also determined by assumptions. The key quantity of interest is the posterior distribution

$$p_\theta(\mathbf{z} \mid \mathbf{x}) = \frac{p_\theta(\mathbf{z}) \cdot p_\theta(\mathbf{x} \mid \mathbf{z})}{p_\theta(\mathbf{x})}$$

Computing this quantity is known as **inference** and obviously, inference requires calculation of the marginal likelihood or **evidence**

$$p_\theta(\mathbf{x}) = \int_{\mathbf{z}} p_\theta(\mathbf{z}, \mathbf{x})$$

In all but the simplest latent variable models, the evidence is an intractable integral and exact inference is therefore infeasible. Markov Chain Monte Carlo (MCMC) algorithms may be employed to approximate the integral. However, for large datasets, MCMC may be impractical due to long mixing times. As well, the sequential nature of MCMC means that we are unable to exploit modern parallel computing tools.

Variational inference provides a scalable and parallelizable alternative for inference as follows. Suppose $q_\lambda(\mathbf{z} \mid \mathbf{x})$ is any parametrized family of conditional distributions. We can rewrite the log-evidence

$$\begin{aligned}
\log p_\theta(x) &= \log \int_{\mathbf{z}} p_\theta(\mathbf{z}, \mathbf{x}) \\
&= \log \int_{\mathbf{z}} p_\theta(\mathbf{z}, \mathbf{x}) \cdot \frac{q_\lambda(\mathbf{z} \mid \mathbf{x})}{q_\lambda(\mathbf{z} \mid \mathbf{x})} \\
&= \log \int_{\mathbf{z}} \frac{p_\theta(\mathbf{z}, \mathbf{x})}{q_\lambda(\mathbf{z} \mid \mathbf{x})} \cdot q_\lambda(\mathbf{z} \mid \mathbf{x}) \\
&= \log \mathbb{E}_{q_\lambda}\left[\frac{p_\theta(\mathbf{z}, \mathbf{x})}{q_\lambda(\mathbf{z} \mid \mathbf{x})}\right]
\end{aligned}$$

Recall Jensen's Inequality for the log function

$$\log\left(\mathbb{E}\left[X\right]\right) \geq \mathbb{E}\left[\log(X)\right]$$

Applying this inequality to the log-evidence yields

$$\log p_\theta(x) = \log \mathbb{E}_{q_\lambda}\left[\frac{p_\theta(\mathbf{z}, \mathbf{x})}{q_\lambda(\mathbf{z} \mid \mathbf{x})}\right] \geq \mathbb{E}_{q_\lambda}\left[\log \frac{p_\theta(\mathbf{z}, \mathbf{x})}{q_\lambda(\mathbf{z} \mid \mathbf{x})}\right]$$

Hence, we have discovered a lower bound for the evidence. Unsurprisingly, the quantity

$$\mathcal{L}(\theta, \lambda; \mathbf{x}) := \mathbb{E}_{q_\lambda}\left[\log \frac{p_\theta(\mathbf{z}, \mathbf{x})}{q_\lambda(\mathbf{z} \mid \mathbf{x})}\right]$$

is known as the **Evidence Lower BOund (ELBO)**. Note that the ELBO is per-datapoint, i.e., $\mathcal{L}$ is a function of $\mathbf{x}$.

To understand the significance of the ELBO, observe that we can re-express it like so

$$\begin{aligned}
\mathcal{L}(\theta, \lambda; \mathbf{x}) =& \mathbb{E}_{q_\lambda}\left[\log \frac{p_\theta(\mathbf{z}, \mathbf{x})}{q_\lambda(\mathbf{z} \mid \mathbf{x})}\right] \\
=& \mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{z}, \mathbf{x}) - \log q_\lambda(\mathbf{z} \mid \mathbf{x})\right] \\
=& \mathbb{E}_{q_\lambda}\left[\log \left(p_\theta(\mathbf{x}) \cdot p_\theta(\mathbf{z} \mid \mathbf{x})\right) - \log q_\lambda(\mathbf{z} \mid \mathbf{x})\right] \\
=& \mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{x}) + \log p_\theta(\mathbf{z} \mid \mathbf{x}) - \log q_\lambda(\mathbf{z} \mid \mathbf{x})\right] \\
=& \mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{x}) - \left(\log q_\lambda(\mathbf{z} \mid \mathbf{x}) - \log p_\theta(\mathbf{z} \mid \mathbf{x})\right)\right] \\
=& \mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{x})\right] - \mathbb{E}_{q_\lambda}\left[\log q_\lambda(\mathbf{z} \mid \mathbf{x}) - \log p_\theta(\mathbf{z} \mid \mathbf{x})\right] \\
=& \log p_\theta(\mathbf{x}) - \mathbb{E}_{q_\lambda}\left[\log q_\lambda(\mathbf{z} \mid \mathbf{x}) - \log p_\theta(\mathbf{z} \mid \mathbf{x})\right]
\end{aligned}$$

Recall the definition of **Kullback-Leibler (KL) Divergence** between distributions

$$\mathrm{KL}\left[q \,||\, p\right] := \int_{\mathbf{z}} q(\mathbf{z}) \log \frac{q(\mathbf{z})}{p(\mathbf{z})} = \mathbb{E}_q\left[\log \frac{q(\mathbf{z})}{p(\mathbf{z})}\right] = \mathbb{E}_q\left[\log q(\mathbf{z}) - \log p(\mathbf{z})\right]$$

Clearly, we obtain

$$\mathcal{L}(\theta, \lambda; \mathbf{x}) = \log p_\theta(\mathbf{x}) - \mathrm{KL}\left[q_\lambda(\mathbf{z} \mid \mathbf{x}) \,||\, p_\theta(\mathbf{z} \mid \mathbf{x})\right]$$

Hence, for given $\mathbf{x}$ and a given choice of model parameters θ, finding λ which maximizes $\mathcal{L}$ is the same thing as computing λ which minimizes the KL divergence between $q_\lambda(\mathbf{z} \mid \mathbf{x})$ and the true posterior $p_\theta(\mathbf{z} \mid \mathbf{x})$. We can therefore interpret q_λ as a **surrogate** or **approximate posterior** to the intractable true posterior. Moreover, the problem of approximate inference in now reduced to an optimization problem (viz., finding an optimal λ) and we are free to apply whatever optimization tools we have available. Once the optimization problem for λ is solved, we obtain a distribution over latent variables given $\mathbf{x}$ and θ. We call λ the **variational parameters** of our model and **variational inference** simply means approximate inference facilitated by these variational parameters. Observe that the optimization is unconstrained; this reflects the fact that the variational parameters are free parameters which are introduced by the modeler and which exist completely apart from the model parameters.

The derivations above provide a general procedure for approximate inference under the assumption that the model parameters θ are known. However, in practice, we need to perform **parameter estimation**, i.e., we need estimates for θ. Note that given values for

Algorithm 1 Coordinate Ascent for Variational Inference

```
Require: data $\mathbf{x}$, joint distribution family $p_\theta$, surrogate posterior family $q_\lambda$, tolerance $\epsilon$
  Initialize model parameters $\theta^{(0)}$ and variational parameters $\lambda^{(0)}$
  for $k = 0, 1, 2, \ldots$ do
      $\lambda^{(k+1)} \leftarrow \underset{\lambda}{\operatorname{argmax}}\ \mathcal{L}(\theta^{(k)}, \lambda; \mathbf{x})$
      $\theta^{(k+1)} \leftarrow \underset{\theta}{\operatorname{argmax}}\ \mathcal{L}(\theta, \lambda^{(k+1)}; \mathbf{x})$
      if $\left\| \left[\theta^{(k+1)}, \lambda^{(k+1)}\right] - \left[\theta^{(k)}, \lambda^{(k)}\right] \right\| < \epsilon$ then return $\theta^{(k+1)}$, $\lambda^{(k+1)}$
```

the variational parameters, we can optimize $\mathcal{L}$ for θ. This suggests a natural alternating optimization procedure for $\mathcal{L}$ for a given data-point $\mathbf{x}$.

The above procedure amounts to coordinate ascent on $\mathcal{L}$. We mention this scheme not only because it is applied in actual use-cases of variational inference (see [Blei et al., 2003] again, for example) but it sheds some further light on what is taking place when we optimize $\mathcal{L}$. Holding θ fixed and optimizing for λ amounts to committing to a particular instance of the generative model and then finding the best surrogate posterior which accords with the generative process. This is the conceptual content of the fact that maximizing $\mathcal{L}$ is equivalent to minimizing the KL divergence between the true posterior and the surrogate posterior. In the subsequent step, where we hold the variational parameters fixed but search for an optimal θ, we have decided what surrogate posterior to use; in turn, this determines a surrogate to the true data likelihood (viz. $\mathcal{L}(\theta, \lambda^*; \mathbf{x})$) and we must find the θ which maximizes this surrogate likelihood.

This discussion also exposes a limitation of the variational inference approach. We do not have an objective quantification of how well the surrogate posterior approximates the true posterior and, by extension, no quantification of how close the optimal value of the ELBO is to the true data likelihood. Moreover, our choice of probability distributions for the surrogate posterior biases the overall model. See [Dupuy and Bach, 2017] for an exploration of limitations of VI as it pertains to LDA. Typically, after a VI model has been trained, domain-specific empirical metrics are computed on the test set to obtain a rough measure of model quality. For instance, VI models for NLP tasks use perplexity on test data. Finally, we are not restricted to using coordinate descent on $\mathcal{L}$ when training. Many of the algorithms we explore in the sequel jointly optimize for model parameters and variational parameters. Indeed, as already mentioned, we may choose whichever optimization technique holds the most appeal for a given problem.

Before moving on, we make some remarks about terminology. The coordinate ascent procedure just mentioned often goes under the moniker of **variational EM**. The variational E-step is the computation of the variational parameters while the M-step is unchanged compared to the classical EM algorithm. The term "variational inference" loosely refers to any method that employs approximation of intractable distributions using simpler, parametrized ones. However, in certain contexts, the term refers specifically to computation of the variational parameters. Thus, in the coordinate ascent algorithm we outlined, only the step which outputs values for the variational parameters may be referred to as variational inference. In this usage of the phrase, the computation of model parameters, while an essential part of training, would not fall under the umbrella of variational inference. We point this out to make the reader aware that some authors use "variational inference" to mean a procedure which computes variational parameters under the assumption that model parameters are fixed and known. Usually, such authors will present a scheme which uses the "variational inference" algorithm as part of a larger algorithm to compute both variational parameters and model parameters. Of course, for algorithms which simultaneously update variational and model parameters, this way of using the phrase does not make much sense.

2.2 Exact Expressions for the ELBO

As we saw in the preceding section, optimizing $\mathcal{L}$ amounts to joint approximate inference and parameter estimation in the latent variable model. However, if we are to use $\mathcal{L}$ as an objective function in an optimization routine, we need to be able to evaluate $\mathcal{L}$ given appropriate inputs. Additionally, if our optimization is gradient-based, we require evaluation of gradients of $\mathcal{L}$ with respect to both variational parameters and model parameters. At this point, it will be helpful to note that latent variable models typically assume tractable sampling and evaluation of the prior $p_\theta(\mathbf{z})$, the likelihood $p_\theta(\mathbf{x} \mid \mathbf{z})$ and the surrogate posterior $q_\lambda(\mathbf{z} \mid \mathbf{x})$. As a corollary, the joint density $p_\theta(\mathbf{z}, \mathbf{x})$ is also tractable. This obviously implies that we should try to re-express all quantities of interest solely in terms of these densities. That is exactly what we do in the next couple of derivations.

The original expression

$$\mathcal{L}(\theta, \lambda; \mathbf{x}) = \log p_\theta(\mathbf{x}) - \mathrm{KL}\left[q_\lambda(\mathbf{z} \mid \mathbf{x}) \,||\, p_\theta(\mathbf{z} \mid \mathbf{x})\right]$$

which we derived for $\mathcal{L}$ is clearly impractical to compute since the right-hand side contains the evidence as well as the true posterior. However, we can perform an alternative derivation

$$\begin{aligned}
\mathcal{L}(\theta, \lambda; \mathbf{x}) =& \mathbb{E}_{q_\lambda}\left[\log \frac{p_\theta(\mathbf{z}, \mathbf{x})}{q_\lambda(\mathbf{z} \mid \mathbf{x})}\right] \\
=& \mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{z}, \mathbf{x}) - \log q_\lambda(\mathbf{z} \mid \mathbf{x})\right] \\
=& \mathbb{E}_{q_\lambda}\left[\log\left(p_\theta(\mathbf{z}) \cdot p_\theta(\mathbf{x} \mid \mathbf{z})\right) - \log q_\lambda(\mathbf{z} \mid \mathbf{x})\right] \\
=& \mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{z}) + \log p_\theta(\mathbf{x} \mid \mathbf{z}) - \log q_\lambda(\mathbf{z} \mid \mathbf{x})\right] \\
=& \mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{x} \mid \mathbf{z}) - \log q_\lambda(\mathbf{z} \mid \mathbf{x}) + \log p_\theta(\mathbf{z})\right]
\end{aligned}$$

Hence, we obtain an expression for $\mathcal{L}$ in terms of the likelihood, the surrogate posterior and the prior.

But we can also keep going with the above derivation

$$\begin{aligned}
\mathcal{L}(\theta, \lambda; \mathbf{x}) =& \mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{x} \mid \mathbf{z}) - \log q_\lambda(\mathbf{z} \mid \mathbf{x}) + \log p_\theta(\mathbf{z})\right] \\
=& \mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{x} \mid \mathbf{z}) - \left(\log q_\lambda(\mathbf{z} \mid \mathbf{x}) - \log p_\theta(\mathbf{z})\right)\right] \\
=& \mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{x} \mid \mathbf{z})\right] - \mathbb{E}_{q_\lambda}\left[\log q_\lambda(\mathbf{z} \mid \mathbf{x}) - \log p_\theta(\mathbf{z})\right] \\
=& \mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{x} \mid \mathbf{z})\right] - \mathrm{KL}\left[q_\lambda(\mathbf{z} \mid \mathbf{x}) \,||\, p_\theta(\mathbf{z})\right]
\end{aligned}$$

Not only do we obtain another version of $\mathcal{L}$ but we can take advantage of situations in which the KL divergence admits an analytical form (e.g., if the surrogate posterior and prior are both Gaussians). Furthermore, the above expression provides a different perspective on what $\mathcal{L}$ represents. We may think of q_λ as an **encoder** which, given an observation $\mathbf{x}$, “compresses" the observation into a latent “code" $\mathbf{z}$. Choosing a good surrogate posterior can therefore be seen as picking an encoding engine which is able to extract a parsimonious representation of observed data. Conversely, p_θ can be thought of as a **decoder** whose job is to “decompress" or “reconstruct" datapoints given a compressed representations of said datapoints. With this interpretation, the first term

$$\mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{x} \mid \mathbf{z})\right]$$

measures the expected error in reconstruction when using q_λ to encode datapoints and using p_θ to decompress latent codes into points in dataspace. The KL term

$$\mathrm{KL}\left[q_\lambda(\mathbf{z} \mid \mathbf{x}) \,||\, p_\theta(\mathbf{z})\right]$$

which is being subtracted from the expected error is therefore a regularizer which prevents the decoder from straying too far from the prior distribution over latent codes.

Finally, we can rewrite

$$\mathcal{L}(\theta, \lambda; \mathbf{x}) = \mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{x} \mid \mathbf{z}) - \log q_\lambda(\mathbf{z} \mid \mathbf{x}) + \log p_\theta(\mathbf{z})\right]$$

as

$$\begin{aligned}\mathcal{L}(\theta, \lambda; \mathbf{x}) =& \mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{x} \mid \mathbf{z}) + \log p_\theta(\mathbf{z})\right] + \mathbb{E}_{q_\lambda}\left[\log q_\lambda(\mathbf{z} \mid \mathbf{x})\right] \\ =& \mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{x} \mid \mathbf{z}) + \log p_\theta(\mathbf{z})\right] + H\left[q_\lambda\right]\end{aligned}$$

Here, H refers to the **entropy**

$$H[q] = \mathbb{E}_q[-\log q]$$

of a distribution.

Thus, we have exact formulas for the ELBO which we record in the box below.

$$\boxed{\begin{aligned}\mathcal{L}(\theta, \lambda; \mathbf{x}) =& \mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{x} \mid \mathbf{z}) - \log q_\lambda(\mathbf{z} \mid \mathbf{x}) + \log p_\theta(\mathbf{z})\right] \\ \mathcal{L}(\theta, \lambda; \mathbf{x}) =& \mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{x} \mid \mathbf{z})\right] - \mathrm{KL}\left[q_\lambda(\mathbf{z} \mid \mathbf{x}) \,||\, p_\theta(\mathbf{z})\right] \\ \mathcal{L}(\theta, \lambda; \mathbf{x}) =& \mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{x} \mid \mathbf{z}) + \log p_\theta(\mathbf{z})\right] + H\left[q_\lambda\right]\end{aligned}} \qquad \begin{aligned}(2.1)\\(2.2)\\(2.3)\end{aligned}$$

Following [Miller, 2016], these formulas shall, respectively, be called the **Fully Monte Carlo (FMC) form**, the **KL form** and the **entropy form** for the ELBO.

2.3 Exact Expressions for Gradients of the ELBO

As mentioned, we may require gradients of the ELBO for optimization. For our derivations in this section, we shall follow [Mnih and Gregor, 2014] and start from the equation

$$\mathcal{L}(\theta, \lambda; \mathbf{x}) = \mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{z}, \mathbf{x}) - \log q_\lambda(\mathbf{z} \mid \mathbf{x})\right]$$

Firstly, the gradient of $\mathcal{L}$ with respect to θ is straightforward.

$$\begin{aligned}\nabla_\theta \mathcal{L}(\theta, \lambda; \mathbf{x}) =& \nabla_\theta \mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{z}, \mathbf{x}) - \log q_\lambda(\mathbf{z} \mid \mathbf{x})\right] \\ =& \nabla_\theta \left[\mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{z}, \mathbf{x})\right] - \mathbb{E}_{q_\lambda}\left[\log q_\lambda(\mathbf{z} \mid \mathbf{x})\right]\right] \\ =& \nabla_\theta \mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{z}, \mathbf{x})\right] - \nabla_\theta \mathbb{E}_{q_\lambda}\left[\log q_\lambda(\mathbf{z} \mid \mathbf{x})\right]\end{aligned}$$

Since the term

$$\mathbb{E}_{q_\lambda}\left[\log q_\lambda(\mathbf{z} \mid \mathbf{x})\right]$$

is independent[1] of θ, we have

$$\begin{aligned}\nabla_\theta \mathcal{L}(\theta, \lambda; \mathbf{x}) =& \nabla_\theta \mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{z}, \mathbf{x})\right] \\ =& \nabla_\theta \int_{\mathbf{z}} q_\lambda(\mathbf{z} \mid \mathbf{x}) \cdot \log p_\theta(\mathbf{z}, \mathbf{x}) \\ =& \int_{\mathbf{z}} \nabla_\theta \left(q_\lambda(\mathbf{z} \mid \mathbf{x}) \cdot \log p_\theta(\mathbf{z}, \mathbf{x})\right) \\ =& \int_{\mathbf{z}} q_\lambda(\mathbf{z} \mid \mathbf{x}) \cdot \nabla_\theta \log p_\theta(\mathbf{z}, \mathbf{x}) \\ =& \mathbb{E}_{q_\lambda}\left[\nabla_\theta \log p_\theta(\mathbf{z}, \mathbf{x})\right] \\ =& \mathbb{E}_{q_\lambda}\left[\nabla_\theta \log \left(p_\theta(\mathbf{x} \mid \mathbf{z}) \cdot p_\theta(\mathbf{z})\right)\right] \\ =& \mathbb{E}_{q_\lambda}\left[\nabla_\theta \log p_\theta(\mathbf{x} \mid \mathbf{z}) + \nabla_\theta \log p_\theta(\mathbf{z})\right]\end{aligned}$$

So, we obtain the gradient with respect to θ in terms of the likelihood, surrogate posterior and prior. We do an analogous calculation for the gradient with respect to λ. We start by observing that

$$\begin{aligned}\nabla_\lambda \mathcal{L}(\theta, \lambda; \mathbf{x}) =& \nabla_\lambda \mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{z}, \mathbf{x}) - \log q_\lambda(\mathbf{z} \mid \mathbf{x})\right] \\ =& \nabla_\lambda \left[\mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{z}, \mathbf{x})\right] - \mathbb{E}_{q_\lambda}\left[\log q_\lambda(\mathbf{z} \mid \mathbf{x})\right]\right] \\ =& \nabla_\lambda \mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{z}, \mathbf{x})\right] - \nabla_\lambda \mathbb{E}_{q_\lambda}\left[\log q_\lambda(\mathbf{z} \mid \mathbf{x})\right]\end{aligned}$$

Now,

$$\begin{aligned}& \nabla_\lambda \mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{z}, \mathbf{x})\right] \\ =& \nabla_\lambda \int_{\mathbf{z}} q_\lambda(\mathbf{z} \mid \mathbf{x}) \cdot \log p_\theta(\mathbf{z}, \mathbf{x}) \\ =& \int_{\mathbf{z}} \nabla_\lambda \left(q_\lambda(\mathbf{z} \mid \mathbf{x}) \cdot \log p_\theta(\mathbf{z}, \mathbf{x})\right) \\ =& \int_{\mathbf{z}} \log p_\theta(\mathbf{z}, \mathbf{x}) \cdot \nabla_\lambda q_\lambda(\mathbf{z} \mid \mathbf{x})\end{aligned}$$

[1]We emphasize our earlier remark that the variational parameters are free parameters assumed to exist apart from the model parameters. As is often the case, this assumption may not be borne out exactly in practical situations.

On the other hand,

$$\begin{aligned}
&\nabla_\lambda \mathbb{E}_{q_\lambda}[\log q_\lambda(\mathbf{z} \mid \mathbf{x})] \\
=&\nabla_\lambda \int_{\mathbf{z}} q_\lambda(\mathbf{z} \mid \mathbf{x}) \cdot \log q_\lambda(\mathbf{z} \mid \mathbf{x}) \\
=&\int_{\mathbf{z}} \nabla_\lambda \left(q_\lambda(\mathbf{z} \mid \mathbf{x}) \cdot \log q_\lambda(\mathbf{z} \mid \mathbf{x})\right) \\
=&\int_{\mathbf{z}} \left(q_\lambda(\mathbf{z} \mid \mathbf{x}) \cdot \nabla_\lambda \log q_\lambda(\mathbf{z} \mid \mathbf{x}) + \log q_\lambda(\mathbf{z} \mid \mathbf{x}) \cdot \nabla_\theta q_\lambda(\mathbf{z} \mid \mathbf{x})\right) \\
=&\int_{\mathbf{z}} q_\lambda(\mathbf{z} \mid \mathbf{x}) \cdot \nabla_\lambda \log q_\lambda(\mathbf{z} \mid \mathbf{x}) + \int_{\mathbf{z}} \log q_\lambda(\mathbf{z} \mid \mathbf{x}) \cdot \nabla_\theta q_\lambda(\mathbf{z} \mid \mathbf{x}) \\
=&\int_{\mathbf{z}} q_\lambda(\mathbf{z} \mid \mathbf{x}) \cdot \frac{\nabla_\lambda q_\lambda(\mathbf{z} \mid \mathbf{x})}{q_\lambda(\mathbf{z} \mid \mathbf{x})} + \int_{\mathbf{z}} \log q_\lambda(\mathbf{z} \mid \mathbf{x}) \cdot \nabla_\theta q_\lambda(\mathbf{z} \mid \mathbf{x}) \\
=&\int_{\mathbf{z}} \nabla_\lambda q_\lambda(\mathbf{z} \mid \mathbf{x}) + \int_{\mathbf{z}} \log q_\lambda(\mathbf{z} \mid \mathbf{x}) \cdot \nabla_\theta q_\lambda(\mathbf{z} \mid \mathbf{x}) \\
=&\nabla_\lambda \int_{\mathbf{z}} q_\lambda(\mathbf{z} \mid \mathbf{x}) + \int_{\mathbf{z}} \log q_\lambda(\mathbf{z} \mid \mathbf{x}) \cdot \nabla_\theta q_\lambda(\mathbf{z} \mid \mathbf{x}) \\
=&\nabla_\lambda 1 + \int_{\mathbf{z}} \log q_\lambda(\mathbf{z} \mid \mathbf{x}) \cdot \nabla_\theta q_\lambda(\mathbf{z} \mid \mathbf{x}) \\
=&0 + \int_{\mathbf{z}} \log q_\lambda(\mathbf{z} \mid \mathbf{x}) \cdot \nabla_\theta q_\lambda(\mathbf{z} \mid \mathbf{x}) \\
=&\int_{\mathbf{z}} \log q_\lambda(\mathbf{z} \mid \mathbf{x}) \cdot \nabla_\theta q_\lambda(\mathbf{z} \mid \mathbf{x})
\end{aligned}$$

Therefore,

$$\begin{aligned}
&\nabla_\lambda \mathcal{L}(\theta, \lambda; \mathbf{x}) \\
=&\int_{\mathbf{z}} \log p_\theta(\mathbf{z}, \mathbf{x}) \cdot \nabla_\lambda q_\lambda(\mathbf{z} \mid \mathbf{x}) - \int_{\mathbf{z}} \log q_\lambda(\mathbf{z} \mid \mathbf{x}) \cdot \nabla_\theta q_\lambda(\mathbf{z} \mid \mathbf{x}) \\
=&\int_{\mathbf{z}} \log p_\theta(\mathbf{z}, \mathbf{x}) \cdot \nabla_\lambda q_\lambda(\mathbf{z} \mid \mathbf{x}) - \log q_\lambda(\mathbf{z} \mid \mathbf{x}) \cdot \nabla_\theta q_\lambda(\mathbf{z} \mid \mathbf{x}) \\
=&\int_{\mathbf{z}} \left(\log p_\theta(\mathbf{z}, \mathbf{x}) - \log q_\lambda(\mathbf{z} \mid \mathbf{x})\right) \cdot \nabla_\theta q_\lambda(\mathbf{z} \mid \mathbf{x})
\end{aligned}$$

But since

$$\begin{aligned}
\nabla_\lambda \log q_\lambda(\mathbf{z} \mid \mathbf{x}) &= \frac{\nabla_\lambda q_\lambda(\mathbf{z} \mid \mathbf{x})}{q_\lambda(\mathbf{z} \mid \mathbf{x})} \\
\Rightarrow \nabla_\lambda q_\lambda(\mathbf{z} \mid \mathbf{x}) &= q_\lambda(\mathbf{z} \mid \mathbf{x}) \cdot \nabla_\lambda \log q_\lambda(\mathbf{z} \mid \mathbf{x})
\end{aligned}$$

we obtain

$$\begin{aligned}
&\nabla_\lambda \mathcal{L}(\theta, \lambda; \mathbf{x}) \\
=&\int_{\mathbf{z}} (\log p_\theta(\mathbf{z}, \mathbf{x}) - \log q_\lambda(\mathbf{z} \mid \mathbf{x})) \cdot q_\lambda(\mathbf{z} \mid \mathbf{x}) \cdot \nabla_\lambda \log q_\lambda(\mathbf{z} \mid \mathbf{x}) \\
=&\int_{\mathbf{z}} q_\lambda(\mathbf{z} \mid \mathbf{x}) \cdot ((\log p_\theta(\mathbf{z}, \mathbf{x}) - \log q_\lambda(\mathbf{z} \mid \mathbf{x})) \cdot \nabla_\lambda \log q_\lambda(\mathbf{z} \mid \mathbf{x})) \\
=&\mathbb{E}_{q_\lambda}\left[(\log p_\theta(\mathbf{z}, \mathbf{x}) - \log q_\lambda(\mathbf{z} \mid \mathbf{x})) \cdot \nabla_\lambda \log q_\lambda(\mathbf{z} \mid \mathbf{x})\right] \\
=&\mathbb{E}_{q_\lambda}\left[(\log p_\theta(\mathbf{x} \mid \mathbf{z}) + \log p_\theta(\mathbf{z}) - \log q_\lambda(\mathbf{z} \mid \mathbf{x})) \cdot \nabla_\lambda \log q_\lambda(\mathbf{z} \mid \mathbf{x})\right] \\
=&\mathbb{E}_{q_\lambda}\left[(\log p_\theta(\mathbf{x} \mid \mathbf{z}) - \log q_\lambda(\mathbf{z} \mid \mathbf{x}) + \log p_\theta(\mathbf{z})) \cdot \nabla_\lambda \log q_\lambda(\mathbf{z} \mid \mathbf{x})\right]
\end{aligned}$$

We record the main results in the box below.

$$\boxed{\begin{aligned}
\nabla_\theta \mathcal{L}(\theta, \lambda; \mathbf{x}) =&\mathbb{E}_{q_\lambda}\left[\nabla_\theta \log p_\theta(\mathbf{x} \mid \mathbf{z}) + \nabla_\theta \log p_\theta(\mathbf{z})\right] \\
\nabla_\lambda \mathcal{L}(\theta, \lambda; \mathbf{x}) =&\mathbb{E}_{q_\lambda}\left[(\log p_\theta(\mathbf{x} \mid \mathbf{z}) - \log q_\lambda(\mathbf{z} \mid \mathbf{x}) + \log p_\theta(\mathbf{z})) \cdot \nabla_\lambda \log q_\lambda(\mathbf{z} \mid \mathbf{x})\right]
\end{aligned}}$$

(2.4)

(2.5)

Notice that both of the expressions are "fully Monte Carlo" forms in the sense that they are expectations against q_λ.

2.4 Variational Inference using Batched Data

Before proceeding, it will be helpful to introduce some notation and assumptions when discussing training a VI model on a set of data. Note that we have restricted ourselves to discussing the ELBO provided only a single datapoint $\mathbf{x}$. This is mostly due to the fact that the core ideas and derivations are easier to follow when only dealing with a single observation. However, the extension to a collection of datapoints is straightforward.

Given a dataset of observations

$$\mathbf{X} \coloneqq \{\mathbf{x}^{(i)}\}_{i=1}^N$$

the variable λ collects together parameters $\{\lambda_i\}_{i=1}^N$ such that λ_i consists of the variational parameters associated to the i^{th} datapoint. In other words,

$$\lambda = [\lambda_1, \lambda_2, \ldots, \lambda_N]$$

Note that we do not assume that the λ_i are independent or even that they are distinct from one another. The datapoints are assumed to be generated in an iid fashion and thus, the marginal likelihood factorizes

$$p_\theta(\mathbf{X}) = \prod_{i=1}^{N} p_\theta(\mathbf{x}^{(i)})$$

This implies that the log marginal likelihood is a sum

$$\log p_\theta(\mathbf{X}) = \sum_{i=1}^{N} \log p_\theta(\mathbf{x}^{(i)})$$

where the i^{th} summand is bounded below by $\mathcal{L}(\theta, \lambda; \mathbf{x}^{(i)})$. Hence, the goal during training is to compute

$$\underset{\theta,\lambda}{\operatorname{argmax}} \sum_{i=1}^{N} \mathcal{L}(\theta, \lambda; \mathbf{x}^{(i)})$$

Going forward, we will primarily be interested in the case where there are no global latent variables, i.e., datapoints in $\mathbf{X}$ are not assumed to share latent variables[2]. Our notation will therefore favor situations where each datapoint has a local (set of) latent variable(s) associated to it. We shall use $\mathbf{Z}^{(i)}$ to denote a collection of the form

$$\{\mathbf{z}^{(i,l)}\}_{l=1}^{L}$$

where, for all $l \in \{1, \ldots, L\}$,

$$\mathbf{z}^{(i,l)} \sim q_{\lambda_i}(\mathbf{z} \mid \mathbf{x})$$

2.5 Amortized Inference and Mean-Field Inference

We now clarify some nomenclature typically found in the VI literature. Firstly, a VI model is said to have a **mean-field variational family** when the surrogate posterior factorizes as

$$q_\lambda(\mathbf{Z}^{(1)}, \ldots, \mathbf{Z}^{(N)} \mid \mathbf{X}) = \prod_{i=1}^{N} q_{\lambda_i}(\mathbf{Z}^{(i)} \mid \mathbf{X})$$

[2]This is not the same as assuming that datapoints do not have shared variational parameters since the $\mathbf{z}^{(i)}$ are *samples* from the surrogate posterior.

Put differently, there are independent variational parameters associated to each latent variable. Again, in the common case, a latent variable is local to each datapoint and so, the above equation is equivalent to

$$q_\lambda(\mathbf{Z}^{(1)}, \ldots, \mathbf{Z}^{(N)} \mid \mathbf{X}) = \prod_{i=1}^{N} q_{\lambda_i}(\mathbf{Z}^{(i)} \mid \mathbf{x}^{(i)})$$

There is a general variational EM procedure available during training. The variational parameters are iteratively updated using a coordinate ascent[3] algorithm and we refer to [Blei et al., 2016] for the coordinate update formula. Notice that we are forced to make a pass through the entire dataset to update all of the variational parameters before we can perform a single update of the model parameters (or even global variational parameters). The paper [Hoffman et al., 2013] presents a stochastic optimization method for scaling mean-field VI to large datasets. Also, we may choose different parametric families for each of the factors q_{λ_i} in the surrogate posterior. Therefore, even though the true posterior is unlikely to be contained in the mean-field variational family[4], the mean-field family has high representational capacity.

In contrast, we may **amortize** the cost of inference by tying or sharing parameters between variational parameters corresponding to different observations. Besides the obvious advantage of having fewer parameters to compute, amortized inference is crucial when we wish to train models using mini-batches of data. This opens the door to applying stochastic optimization techniques which scale well, as we shall see below. Observe, however, that amortized inference is strictly less expressive than mean-field inference. In practice, we often achieve amortization by having the variational parameters be deterministic functions of the weights and biases of a (fixed) neural network. Lack of representational capacity is therefore not of great concern in many situations.

2.6 Estimating the ELBO and its Gradients

The exposition on VI provided thus far is completely general, i.e., nothing in the preceding makes assumptions about what we are modelling and there are no restrictions on any of the

[3]The coordinate ascent this time is not exactly coordinate ascent on $\mathcal{L}$ since $\mathcal{L}$ also has θ as argument. We are only performing coordinate ascent for a subset of $\mathcal{L}$'s arguments (viz. λ).

[4]For instance, suppose we are classifying MNIST digits by first extracting latent representation for each image; images which have the same label are likely to have similar latent representations and therefore the true posterior is unlikely to assign probabilities to an image completely independently of other images in the same class.

Algorithm 2 Variational EM for Mean-Field Inference

```
Require: dataset $\mathbf{X}$, joint distribution family $p_\theta$, surrogate posterior family $q_\lambda$, tolerance $\epsilon$
  Initialize model parameters $\theta^{(0)}$ and variational parameters $\lambda^{(0)}$
  for $s = 0, 1, 2, \ldots$ do
      $\lambda^{(0)} = \lambda^{(s)}$
      for $k = 0, 1, 2, \ldots$ do          ▷ Begin coordinate ascent loop for variational parameters
          for $i = 1, 2, \ldots$ do
              $\lambda_i^{(k+1)} \leftarrow \underset{\lambda_i}{\operatorname{argmax}} \sum_{j=1}^{N} \mathcal{L}(\theta^{(s)}, \lambda_1^{(k+1)}, \ldots, \lambda_{i-1}^{(k+1)}, \lambda_i, \lambda_{i+1}^{(k)}, \ldots, \lambda_N^{(k)}; \mathbf{x}^{(j)})$
          $\lambda^{(k+1)} \leftarrow \left[\lambda_1^{(k+1)}, \ldots, \lambda_N^{(k+1)}\right]$
          if $\left\|\lambda^{(k+1)} - \lambda^{(k)}\right\| < \epsilon$ then
              $\lambda^{(s+1)} = \lambda^{(k+1)}$
              break                                        ▷ Variational inference done
      $\theta^{(s+1)} \leftarrow \underset{\theta}{\operatorname{argmax}} \sum_{j=1}^{N} \mathcal{L}(\theta, \lambda^{(s+1)}; \mathbf{x}^{(j)})$     ▷ Update model parameters
      if $\left\|\theta^{(s+1)} - \theta^{(s)}\right\| < \epsilon$ then return $\theta^{(s+1)}$, $\lambda^{(s+1)}$
```

probability distributions involved. Additionally, the main formulas in the preceding section are only in terms of the likelihood, surrogate posterior and prior. However, they are all also expectations with respect to the surrogate posterior. Thus, computation of the ELBO and/or its gradients necessitates computation of expectations. Since we have scalability as a desideratum of our training procedure, MCMC and related quadrature algorithms are immediately disqualified from the set of tools we may apply to this problem[5]. In this section, we briefly summarize some of the techniques used to compute the ELBO when training a model.

2.6.1 Closed-Form Calculation

The first and obvious strategy to optimize $\mathcal{L}$ is to find a closed-form expression for the expectations, in which case we obtain closed-form updates for gradient-based optimization schemes. This approach has been successfully used in many cases, such as [Blei et al.,

[5]This is not entirely true; MCMC can be used for auxiliary tasks such as approximating the evidence when the latent space is low-dimensional (see Appendix D in [Kingma and Welling, 2013]). However, MCMC quickly becomes impractical if, as in many practical applications, the latent space is high-dimensional.

2003], [Ghahramani and Beal, 2000], [Blei and Lafferty, 2007], [Knowles and Minka, 2011], [Wang and Blei, 2013]. Unfortunately, the derivation of closed-form expressions is both non-trivial and non-transferable. That is, such derivations require careful choice in model construction which ultimately enable closed-form updates and the assumptions thus made cannot be easily transposed into new contexts. Clearly, this presents a major hurdle to quick development and experimentation when applying VI to a diversity of tasks. Given the specialized nature of each application of closed-form training of VI models, it would take us too far afield to explore any one of these applications in depth. We shall therefore not explore this technique further. The interested reader is invited to consult the papers we have just cited and the references therein.

2.6.2 Black-Box Variational Inference

Surmounting the obstacle of model-specific derivations is the motivating goal underlying [Ranganath et al., 2014]. The paper therefore aims to provide a generic framework for using VI in a wide variety of contexts. Their approach is an alternative to approximating expectations via numeric integration (using MCMC) as well as being an alternative to deriving closed-form update equations. The technique we now explore relies on **stochastic optimization**, where we optimize an objective function by constructing an unbiased estimator for the gradient and then plugging in the estimate into a gradient descent scheme. We shall now see how stochastic optimization enables computation of variational parameters.

Let $H(\theta, \lambda; \mathbf{x})$ be a random variable[6] such that

$$\mathbb{E}\left[H(\theta, \lambda; \mathbf{x})\right] = \nabla_\lambda \mathcal{L}(\theta, \lambda; \mathbf{x}).$$

We can use the update equation

$$\lambda^{(k+1)} \leftarrow \lambda^{(k)} - \eta^{(k)} h^{(k)}(\theta, \lambda^{(k)}; \mathbf{x}),$$

where $\eta^{(k)}$ is the step size for the $(k+1)^{th}$ iteration and $h^{(k)}(\theta, \lambda^{(k)}; \mathbf{x})$ is a realization of $H(\theta, \lambda; \mathbf{x})$. This follows the classic Robbins-Monro algorithm from [Robbins and Monro, 1951], which also lays out conditions on the step size and the gradient estimator sufficient to guarantee convergence to an optimum. In particular, a **Robbins-Monro sequence**

[6]The randomness in H is assumed to be independent of any randomness in $(\theta, \lambda; \mathbf{x})$, i.e., we have a separate random variable $H(\theta, \lambda; \mathbf{x})$ for each $(\theta, \lambda; \mathbf{x})$. Thus, when we take the expected value of $H(\theta, \lambda; \mathbf{x})$, the expectation is with respect to the source of randomness in $H(\theta, \lambda; \mathbf{x})$.

consists of learning rate schedule $\{\eta^{(k)}\}_{k=1}^{\infty}$ satisfying

$$\sum_{k=1}^{\infty} \eta^{(k)} = \infty$$
$$\sum_{k=1}^{\infty} \left(\eta^{(k)}\right)^2 < \infty$$

A natural choice for $H(\theta, \lambda; \mathbf{x})$ is provided by the Monte Carlo estimate

$$\nabla_\lambda \mathcal{L}(\theta, \lambda; \mathbf{x}) \approx \frac{1}{L} \sum_{l=1}^{L} \left[\left(\log p_\theta(\mathbf{x} \mid \mathbf{z}^{(l)}) - \log q_\lambda(\mathbf{z}^{(l)} \mid \mathbf{x}) + \log p_\theta(\mathbf{z}^{(l)}) \right) \cdot \nabla_\lambda \log q_\lambda(\mathbf{z}^{(l)} \mid \mathbf{x}) \right]$$

where $\{\mathbf{z}^{(l)}\}_{l=1}^{L}$ is a set of samples from the surrogate posterior[7]. With this estimator for the gradient, we obtain the **black-box variational inference (BBVI) algorithm**.

Algorithm 3 Black-Box Variational Inference

```
Require: data $\mathbf{x}$, joint distribution family $p_\theta$, surrogate posterior family $q_\lambda$, tolerance $\epsilon$,
  Robbins-Monro sequence $\{\eta^{(k)}\}_{k=1}^{\infty}$, model parameters $\theta$
  Initialize variational parameters $\lambda^{(0)}$
  for $k = 0, 1, 2, \ldots$ do
      for $l = 1, \ldots, L$ do
          $\mathbf{z}^{(l)} \sim q_{\lambda^{(k)}}(\mathbf{z} \mid \mathbf{x})$
      $\lambda^{(k+1)} \leftarrow \lambda^{(k)} - \eta^{(k)} \frac{1}{L} \sum_{l=1}^{L} \left[ \left( \log p_\theta(\mathbf{x} \mid \mathbf{z}^{(l)}) - \log q_\lambda(\mathbf{z}^{(l)} \mid \mathbf{x}) + \log p_\theta(\mathbf{z}^{(l)}) \right) \cdot \nabla_\lambda \log q_\lambda(\mathbf{z}^{(l)} \mid \mathbf{x}) \right]$
      if $\left\| \lambda^{(k+1)} - \lambda^{(k)} \right\| < \epsilon$ then return $\lambda^{(k+1)}$
```

While the BBVI algorithm is theoretically sound, it may be slow to converge in practice since the Monte Carlo estimate for the gradients can be very noisy. Even a large number of samples from the surrogate posterior may not be sufficient to reduce the variance by a desirable degree; appendix D of [Rezende et al., 2014] gives some insight as to why. Indeed, [Ranganath et al., 2014] modifies the basic BBVI algorithm using **Rao-Blackwellization** and **control variates**. However, good control variates often require injecting model-dependent information. Note also that the BBVI algorithm only focuses on computation of variational parameters and is agnostic to the way model parameters are updated.

[7]Hence, the randomness in H in this case is due to randomness in sampling from the surrogate posterior.

2.6.3 The Reparametrization Trick

The core idea of BBVI is to express ELBO gradients as expectations and then approximating these expectations via Monte Carlo estimates. As we saw, practicality forces us to apply additional techniques to control the variance of these estimates. The **reparametrization trick** offers a different method for obtaining a lower variance estimate of ELBO gradients. In more detail, observe that the FMC form of the ELBO furnishes the Monte Carlo estimate

$$\mathcal{L}(\theta, \lambda; \mathbf{x}) \approx \frac{1}{L} \sum_{l=1}^{L} \left[\log p_\theta(\mathbf{x} \mid \mathbf{z}^{(l)}) - \log q_\lambda(\mathbf{z}^{(l)} \mid \mathbf{x}) + \log p_\theta(\mathbf{z}^{(l)})\right]$$

where $\{\mathbf{z}^{(l)}\} \sim q_\lambda(\mathbf{z} \mid \mathbf{x})$. We can, of course, take the derivative of the expression on the right-hand side with respect to both variational parameters and model parameters. The latter is often satisfactory for model parameter updates but the former is known to typically be a high-variance estimator for the variational parameter gradients. One possible explanation is that the noise in the term

$$\nabla_\lambda \log q_\lambda(\mathbf{z}^{(l)} \mid \mathbf{x})$$

is influenced both by the Monte Carlo sampling process and the fact that the derivative depends on the distribution q_λ. See [Miller, 2016] and [Roeder et al., 2017] for a deeper analysis. The **reparametrization trick** ameliorates this by first expressing the random variable

$$\mathbf{z} \sim q_\lambda(\mathbf{z} \mid \mathbf{x})$$

as the image of a differentiable transformation

$$\mathbf{z} = g_\lambda(\boldsymbol{\epsilon})$$

where

$$\boldsymbol{\epsilon} \sim r(\boldsymbol{\epsilon})$$

and $r(\boldsymbol{\epsilon})$ is a noise distribution. The classic example of reparametrization is provided by the normal distribution. Any sample

$$\mathbf{z} \sim \mathcal{N}_{\mu,\sigma}(\mathbf{z} \mid \mathbf{x})$$

can be written as

$$\mathbf{z} = \mu + \sigma \cdot \boldsymbol{\epsilon}$$

with

$$\boldsymbol{\epsilon} \sim \mathcal{N}_{0,1}(\boldsymbol{\epsilon})$$

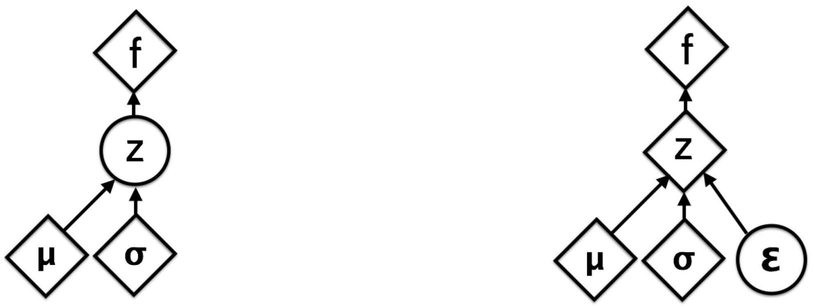


Figure 2.1: Visual for the reparametrization trick. Diamonds indicate deterministic nodes, circles represent stochastic nodes. Taken from [Altosaar, 2018].

The most important thing to note is that we no longer sample $\mathbf{z}^{(l)}$ directly from a distribution; rather, it is the output of a *deterministic* mapping of a random sample $\boldsymbol{\epsilon}^{(l)}$ and the distribution from which $\boldsymbol{\epsilon}^{(l)}$ is sampled is *completely independent* of λ.

Letting

$$f(\mathbf{z}) \coloneqq \log p_\theta(\mathbf{x} \mid \mathbf{z}) - \log q_\lambda(\mathbf{z} \mid \mathbf{x}) + \log p_\theta(\mathbf{z})$$

we now compute, using the same argument as in [Mohamed, 2015],

$$\begin{aligned}\mathcal{L}(\theta, \lambda; \mathbf{x}) =& \mathbb{E}_{q_\lambda}\left[f(\mathbf{z})\right] \\ =& \int_{\mathbf{z}} q_\lambda(\mathbf{z} \mid \mathbf{x}) \cdot f(\mathbf{z}) \\ =& \int_{\boldsymbol{\epsilon}} r(\boldsymbol{\epsilon}) \cdot f(g_\lambda(\boldsymbol{\epsilon})) \\ =& \mathbb{E}_r\left[f(g_\lambda(\boldsymbol{\epsilon}))\right]\end{aligned}$$

Hence, the reparametrization g_λ allows us to express the ELBO as an expectation against the distribution r which is independent of variational parameters. The gradient of the ELBO is now simply

$$\nabla_\lambda \mathcal{L}(\theta, \lambda; \mathbf{x}) = \nabla_\lambda \mathbb{E}_r\left[f(g_\lambda(\boldsymbol{\epsilon}))\right] = \mathbb{E}_r\left[\nabla_\lambda f(g_\lambda(\boldsymbol{\epsilon}))\right]$$

Thus, both the ELBO and its gradients can be estimated using Monte Carlo samples from the noise distribution r. Further intuition is provided by figure 2.1. On the left half of the figure, we see the computation flow for when f is computed naively; on the right, we see that the stochasticity has been "pushed off" from $\mathbf{z}$ and onto $\boldsymbol{\epsilon}$.

We may now compute variational parameters using a stochastic optimization procedure where the gradients are estimated with the derivative of a reparametrized Monte

Carlo estimate of the ELBO[8]. We call such an algorithm **reparametrization gradient variational inference (RGVI)** and we obtain a generic variational inference procedure assuming a reparametrization g_λ exists. The estimator

$$\widetilde{\mathcal{L}}(\theta, \lambda; \mathbf{x}) := \frac{1}{L} \sum_{l=1}^{L} \left[\log p_\theta(\mathbf{x} \mid \mathbf{z}^{(l)}) - \log q_\lambda(\mathbf{z}^{(l)} \mid \mathbf{x}) + \log p_\theta(\mathbf{z}^{(l)})\right] \approx \mathcal{L}(\theta, \lambda; \mathbf{x})$$

where

$$\begin{aligned} \mathbf{z}^{(l)} &= g_\lambda(\boldsymbol{\epsilon}^{(l)}), \\ \boldsymbol{\epsilon}^{(l)} &\sim r(\boldsymbol{\epsilon}) \end{aligned}$$

is called the **Stochastic Gradient Variational Bayes (SGVB)** estimator in [Kingma and Welling, 2013].

Algorithm 4 Reparametrization Gradient Variational Inference

```
Require: data $\mathbf{x}$, joint distribution family $p_\theta$, surrogate posterior family $q_\lambda$, noise distribution $r$, differentiable map $g_\lambda$, number of samples $L$, tolerance $\varepsilon$, Robbins-Monro sequence $\{\eta^{(k)}\}_{k=1}^{\infty}$, model parameters $\theta$
  Initialize variational parameters $\lambda^{(0)}$
  for $k = 0, 1, 2, \ldots$ do
      for $l = 1, \ldots, L$ do
          $\boldsymbol{\epsilon}^{(l)} \sim r(\boldsymbol{\epsilon})$
          $\mathbf{z}^{(l)} = g_\lambda(\boldsymbol{\epsilon}^{(l)})$
      $\lambda^{(k+1)} \leftarrow \lambda^{(k)} - \eta^{(k)} \frac{1}{L} \sum_{l=1}^{L} \nabla_\lambda \left[\log p_\theta(\mathbf{x} \mid \mathbf{z}^{(l)}) - \log q_\lambda(\mathbf{z}^{(l)} \mid \mathbf{x}) + \log p_\theta(\mathbf{z}^{(l)})\right]$
      if $\left\|\lambda^{(k+1)} - \lambda^{(k)}\right\| < \varepsilon$ then return $\lambda^{(k+1)}$
```

As with BBVI, we are agnostic when it comes to model parameter updates. Additionally, we are not restricted to using the Robbins-Monro scheme for stochastic optimization but are free to apply alternative gradient-based schemes such as **stochastic gradient descent (SGD)**, AdaGrad, etc. (see [Duchi et al., 2011]; [Ruder, 2016] provides a survey of gradient descent methods). Moreover, the algorithm can be easily adapted for batches of data. Indeed, given an iid dataset $\mathbf{X}$ with N points and a subset (or mini-batch) $\mathbf{X}^M \subseteq \mathbf{X}$ of M points, the SGVB estimator can be used to construct the approximation

$$\mathcal{L}(\theta, \lambda; \mathcal{X}) \approx \frac{N}{M} \sum_{i=1}^{M} \widetilde{\mathcal{L}}(\theta, \lambda; \mathbf{x}^{(i)})$$

[8]Equivalently, we can estimate the gradient using Monte Carlo samples from r.

to the ELBO for the full dataset. The **auto-encoding variational Bayes (AEVB)** algorithm from [Kingma and Welling, 2013] is an example of a routine which combines the reparametrization trick, joint optimization of variational and model parameters, training using mini-batches of data and application of modern gradient-based optimization algorithms.

Algorithm 5 Auto-Encoding Variational Bayes

```
Require: dataset $\mathbf{X}$, joint distribution family $p_\theta$, surrogate posterior family $q_\lambda$, noise
    distribution $r$, differentiable map $g_\lambda$, number of samples $L$, tolerance $\varepsilon$
    Initialize variational parameters $\lambda^{(0)}$ and model parameters $\theta^{(0)}$
    for $k = 0, 1, 2, \dots$ do
        Randomly sample a mini-batch $\mathbf{X}^M = \{\mathbf{x}^{(i)}\}_{i=1}^M \subseteq \mathbf{X}$
        Sample $\boldsymbol{\epsilon}^{(i,l)} \sim r(\boldsymbol{\epsilon})$ for $(i, l) \in \{1, \dots, M\} \times \{1, \dots, L\}$
        Compute $\mathbf{z}^{(i,l)} = g_\lambda(\boldsymbol{\epsilon}^{(i,l)})$
        Compute $\mathbf{g}^{(k)} := \nabla_{\theta,\lambda} \left[ \frac{N}{M} \sum_{i=1}^M \widetilde{\mathcal{L}}(\theta, \lambda; \mathbf{x}^{(i)}) \right]$
        Use $\mathbf{g}^{(k)}$ in a gradient-based scheme to get updated parameters $\theta^{(k+1)}$, $\lambda^{(k+1)}$
        if $(\theta, \lambda)$ converged then return $\theta^{(k+1)}$, $\lambda^{(k+1)}$
```

Naturally, the requirement that a differentiable transformation g_λ exist restricts the choice of surrogate posteriors to continuous distributions. Hence, BBVI is strictly more general than RGVI. The cost of the extra generality of BBVI is higher variance in the gradient estimates. Lastly, we mention that the reparametrization trick can be used in conjunction with the KL and entropy forms of the ELBO. In cases where the KL term or the entropy term in the ELBO can be computed analytically, we only need a Monte Carlo estimate for part of the ELBO. This may further decrease the variance of the gradient estimates. However, this is not guaranteed to occur and the FMC form of the ELBO can have lower variance in certain regimes. Consult [Roeder et al., 2017] for further details. As we shall see in the sequel, the reparametrization trick and the innovations of the AEVB algorithm provide a powerful framework for latent variable modeling when combined with amortized inference and deep neural networks.

Chapter 3

Normalizing Flows

In the preceding chapter, we elucidated variational inference (VI) as an approach to probabilistic modeling. More precisely, we saw how VI supports a framework for performing inference and parameter estimation in latent variable models (LVM). As we saw, VI entails approximate inference by way of optimization of the ELBO. We then expanded on some modern techniques for making the optimization tractable and scalable. The methods we studied are broadly applicable and we focused on an abstract LVM problem without getting very specific about the three core ingredients which constitute an end-to-end VI model. These ingredients are, as already stated, the following:

(i) likelihood $p_\theta(\mathbf{x} \mid \mathbf{z})$,

(ii) prior $p_\theta(\mathbf{z})$,

(iii) surrogate posterior $q_\lambda(\mathbf{z} \mid \mathbf{x})$.

Any VI model used in practice requires choices for each of these and the choice of surrogate posterior is crucial in determining how successful we will be in a given situation. When deciding what family of surrogate posteriors to employ, we must consider not only the representational capacity but also the computational burden implied. Generally, the more expressive the surrogate posterior, the more difficult it is to optimize the ELBO. However, if we restrict ourselves to simple surrogate posteriors (e.g., Gaussians), we may fail to match properties of the true posterior (e.g., multi-modality). In addition, note that we are always injecting some bias through our choice of surrogate posterior and the amount of bias induced by simple choices may be unacceptably high. Furthermore, it is well-known that the variance of the posterior distribution is underestimated in such cases. See [Turner and Sahani, 2011] for details. Hence, the ideal situation would be one in which we are able to specify flexible, complex and scalable surrogate posteriors while retaining computational feasibility in optimization of the ELBO. **Normalizing flows** are intended as one tool for achieving this goal. The purpose of this chapter is to introduce the basic technique and then catalog a few kinds of normalizing flows found in the current literature on VI.

3.1 Transforming Distributions

In order to understand normalizing flows, we will first review some results from probability theory. Let $\mathbf{z}$ be a (continuous) random variable with distribution q and let $f : \mathbb{R}^d \to \mathbb{R}^d$ be an differentiable function with smooth inverse. Using the inverse function theorem and rules for Jacobians of invertible functions, we may deduce that the random variable

$$\mathbf{z}' \coloneqq f(\mathbf{z})$$

has distribution

$$q'(\mathbf{z}') = q(\mathbf{z}) \left| \det \frac{\partial f}{\partial \mathbf{z}} \right|^{-1}$$

Given a sequence of invertible functions $\{f_k\}_{k=1}^K$ and an initial random variable $\mathbf{z}_0$ with **base distribution** q_0, we inductively define random variables

$$\mathbf{z}_k \coloneqq f_k(\mathbf{z}_{k-1})$$

and compute the **transformed distribution** of z_K to be

$$q_K(\mathbf{z}_K) = q_0(\mathbf{z}_0) \prod_{k=1}^{K} \left| \det \frac{\partial f_k}{\partial \mathbf{z}_{k-1}} \right|^{-1}$$

The log-density of q_K is therefore

$$\log q_K(\mathbf{z}_K) = \log q_0(\mathbf{z}_0) - \sum_{k=1}^{K} \log \left| \det \frac{\partial f_k}{\partial \mathbf{z}_{k-1}} \right|$$

We have thus taken an initial, and possibly simple, distribution and transformed to a perhaps more complex distribution through a sequence of **bijectors** $\{f_k\}_{k=1}^K$. The terms base distribution, transformed distribution and bijector are used in accordance with the nomenclature of [Dillon et al., 2017]. The integer K is known as the **length** of the flow. Note that each bijector can be an arbitrary differentiable and invertible function. Following [Rezende and Mohamed, 2015], the path traversed by the random variables $\mathbf{z}_k$ is called a **flow** and the path formed by their distributions is called a **normalizing flow**. A visual representation of a normalizing flow is provided in figure 3.1.

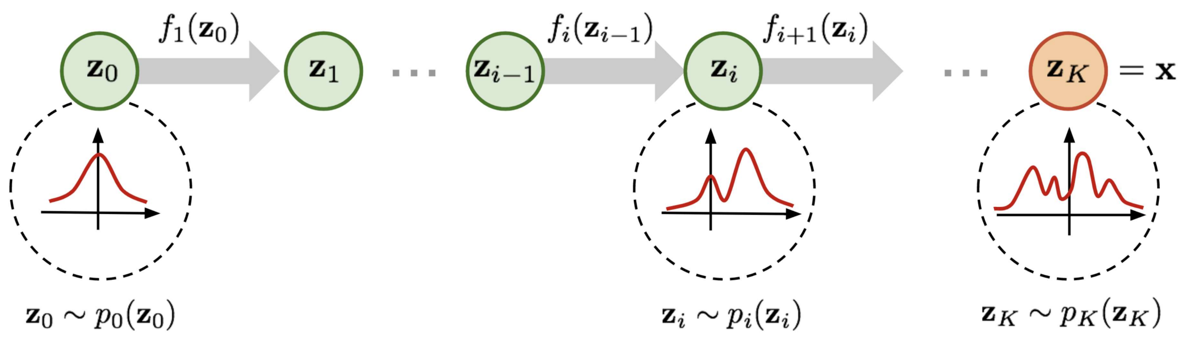

$f_1(\mathbf{z}_0)$
$\mathbf{z}_0$
$\mathbf{z}_1$
$f_i(\mathbf{z}_{i-1})$
$\mathbf{z}_{i-1}$
$\mathbf{z}_i$
$f_{i+1}(\mathbf{z}_i)$
$\mathbf{z}_K = \mathbf{x}$
$\mathbf{z}_0 \sim p_0(\mathbf{z}_0)$
$\mathbf{z}_i \sim p_i(\mathbf{z}_i)$
$\mathbf{z}_K \sim p_K(\mathbf{z}_K)$

Figure 3.1: Transformation of distribution through normalizing flow. Taken from [Weng, 2018].

We can apply a normalizing flow to VI as follows: start with a simple base distribution (e.g., Gaussian) q_0 and define the surrogate posterior $q_\lambda(\mathbf{z} \mid \mathbf{x})$ to be the transformed distribution q_K, i.e,

$$q_\lambda(\mathbf{z} \mid \mathbf{x}) := q_K(\mathbf{z}_K)$$

In slightly more detail, given $\mathbf{x}$, we can generate a sample from the surrogate posterior by sampling a $\mathbf{z}_0$ and then applying the sequence of bijectors. Note that the variational parameters λ are now determined by the parameters of the bijectors used to construct the flow (this will become clearer when we consider some examples below). The FMC form of the ELBO can now be expressed as

$$\begin{aligned}
\mathcal{L}(\theta,\lambda;\mathbf{x}) =& \mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{x}\mid\mathbf{z}) - \log q_\lambda(\mathbf{z}\mid\mathbf{x}) + \log p_\theta(\mathbf{z})\right] \\
=& \mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{x}\mid\mathbf{z}) + \log p_\theta(\mathbf{z}) - \log q_\lambda(\mathbf{z}\mid\mathbf{x})\right] \\
=& \mathbb{E}_{q_\lambda}\left[\log\left(p_\theta(\mathbf{x}\mid\mathbf{z})\cdot p_\theta(\mathbf{z})\right) - \log q_\lambda(\mathbf{z}\mid\mathbf{x})\right] \\
=& \mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{z},\mathbf{x}) - \log q_\lambda(\mathbf{z}\mid\mathbf{x})\right] \\
=& \mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{z},\mathbf{x})\right] - \mathbb{E}_{q_\lambda}\left[\log q_\lambda(\mathbf{z}\mid\mathbf{x})\right] \\
=& \mathbb{E}_{q_K}\left[\log p_\theta(\mathbf{z}_K,\mathbf{x})\right] - \mathbb{E}_{q_K}\left[\log q_K(\mathbf{z}_K)\right] \\
=& \mathbb{E}_{q_K}\left[\log p_\theta(\mathbf{z}_K,\mathbf{x})\right] - \mathbb{E}_{q_K}\left[\log q_0(\mathbf{z}_0) - \sum_{k=1}^{K}\log\left|\det\frac{\partial f_k}{\partial \mathbf{z}_{k-1}}\right|\right] \\
=& \mathbb{E}_{q_K}\left[\log p_\theta\left(f_K\circ\cdots\circ f_1(\mathbf{z}_0),\mathbf{x}\right)\right] - \mathbb{E}_{q_K}\left[\log q_0(\mathbf{z}_0) - \sum_{k=1}^{K}\log\left|\det\frac{\partial f_k}{\partial \mathbf{z}_{k-1}}\right|\right]
\end{aligned}$$

Observe that each of the terms in the last line is of the form

$$\mathbb{E}_{q_K}\left[h(\mathbf{z}_K)\right]$$

where h does not depend on q_K. The law of the unconscious statistician (LOTUS) can be applied inductively to yield

$$\mathbb{E}_{q_K}\left[h(\mathbf{z}_K)\right] = \mathbb{E}_{q_0}\left[h\left(f_K\circ\cdots\circ f_1(\mathbf{z}_0)\right)\right]$$

and hence,

$$\begin{aligned}
\mathcal{L}(\theta,\lambda;\mathbf{x}) =& \mathbb{E}_{q_K}\left[\log p_\theta\left(f_K\circ\cdots\circ f_1(\mathbf{z}_0),\mathbf{x}\right)\right] - \mathbb{E}_{q_K}\left[\log q_0(\mathbf{z}_0) - \sum_{k=1}^{K}\log\left|\det\frac{\partial f_k}{\partial \mathbf{z}_{k-1}}\right|\right] \\
=& \mathbb{E}_{q_0}\left[\log p_\theta\left(f_K\circ\cdots\circ f_1(\mathbf{z}_0),\mathbf{x}\right)\right] - \mathbb{E}_{q_0}\left[\log q_0(\mathbf{z}_0) - \sum_{k=1}^{K}\log\left|\det\frac{\partial f_k}{\partial \mathbf{z}_{k-1}}\right|\right] \\
=& \mathbb{E}_{q_0}\left[\log p_\theta\left(f_K\circ\cdots\circ f_1(\mathbf{z}_0),\mathbf{x}\right)\right] - \mathbb{E}_{q_0}\left[\log q_0(\mathbf{z}_0)\right] + \mathbb{E}_{q_0}\left[\sum_{k=1}^{K}\log\left|\det\frac{\partial f_k}{\partial \mathbf{z}_{k-1}}\right|\right] \\
=& \mathbb{E}_{q_0}\left[\log p_\theta\left(\mathbf{x}\mid f_K\circ\cdots\circ f_1(\mathbf{z}_0)\right)\right] + \mathbb{E}_{q_0}\left[\log p_\theta\left(f_K\circ\cdots\circ f_1(\mathbf{z}_0)\right)\right] \\
&- \mathbb{E}_{q_0}\left[\log q_0(\mathbf{z}_0)\right] + \mathbb{E}_{q_0}\left[\sum_{k=1}^{K}\log\left|\det\frac{\partial f_k}{\partial \mathbf{z}_{k-1}}\right|\right]
\end{aligned}$$

The latter quantity is called the **flow-based free energy bound** in [Rezende and Mohamed, 2015]. Notice that all expectations are against the base distribution q_0 and so, for Monte Carlo estimates, we only require samples from the base distribution. Additionally, if the reparametrization trick is available for the base distribution, we can obtain variance-reduced gradient estimators for a gradient-based optimization scheme. However, the flow-based bound is applicable in any VI procedure since, from a computational perspective, all that a normalizing flow does is to produce a new formula for the ELBO.

In choosing a flow-based surrogate posterior, the choice of surrogate posterior is reduced to picking a base distribution and a sequence of bijectors. The base distribution is typically just a Gaussian distribution and the main effort goes towards designing bijectors with good properties. The usual strategy is to define a parametric family of functions for the bijectors such that it is relatively inexpensive to compute the log-determinants

$$\log \left| \det \frac{\partial f_k}{\partial \mathbf{z}_{k-1}} \right|$$

since calculation of these quantities is a key bottleneck for VI using normalizing flows. In the sequel, we enumerate some of the normalizing flows which have been recently devised to provide rich surrogate posteriors while maintaining reasonable computational complexity.

3.2 Planar Flow

The authors of [Rezende and Mohamed, 2015] introduced two simple families of bijectors for normalizing flows. The first family consists of functions of the form

$$f(\mathbf{z}) = \mathbf{z} + \mathbf{u}h\left(\mathbf{w}^T\mathbf{z} + b\right)$$

with $\{\mathbf{u} \in \mathbb{R}^d, \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}\}$ being the parameters for this family and h being a specified smooth non-linearity such as the tanh function. The choice of non-linearity determines the conditions under which $f(\mathbf{z})$ is invertible[1] and we only get a normalizing flow under the constraints which imply bijectivity of f. Notice that the equation

$$\mathbf{w}^T\mathbf{z} + b = 0$$

defines a hyperplane in latent space and so, geometrically, a function $f(\mathbf{z})$ as above contracts and expands vectors perpendicular to this hyperplane. This is why a flow defined

[1]For instance, if h is the tanh function, $\mathbf{w}^T\mathbf{z} + 1 \geq 0$ is a sufficient condition.

by bijectors of this form is called a **planar flow**. It is easy to see that

$$\frac{\partial f}{\partial \mathbf{z}} = \mathbf{I} + \mathbf{u}^T \mathbf{w} h'(\mathbf{w}^T \mathbf{z} + b)$$

and therefore,

$$\left| \det \frac{\partial f}{\partial \mathbf{z}} \right| = \left| \det \left(\mathbf{I} + \mathbf{u}^T \mathbf{w} h'(\mathbf{w}^T \mathbf{z} + b) \right) \right| = \left| 1 + \mathbf{u}^T \mathbf{w} h'(\mathbf{w}^T \mathbf{z} + b) \right|$$

This implies that the log-determinant terms required in the flow-based ELBO are efficient to compute. Finally, if we construct a sequence of bijectors $\{f_k\}_{k=1}^K$ with

$$\lambda_k \coloneqq \{\mathbf{u}_k, \mathbf{w}_k, b_k\}$$

parametrizing the k^{th} bijector, then the set of variational parameters is simply

$$\lambda = \{\lambda_k\}_{k=1}^K = \{\{\mathbf{u}_k, \mathbf{w}_k, b_k\}\}_{k=1}^K$$

3.3 Radial Flow

The second family of functions considered in [Rezende and Mohamed, 2015] is given by

$$f(\mathbf{z}) = \mathbf{z} + \beta h(\alpha, r) (\mathbf{z} - \mathbf{z}_0)$$

where

$$r \coloneqq \|\mathbf{z} - \mathbf{z}_0\|,$$
$$h(\alpha, r) \coloneqq \frac{1}{\alpha + r}$$

The set of parameters this time is $\{\mathbf{z}_0 \in \mathbb{R}^d, \alpha \in \mathbb{R}^+, \beta \in \mathbb{R}\}$. As with planar flow, extra conditions are needed to guarantee that $f(\mathbf{z})$ is invertible. In this case, it suffices to have $\alpha + \beta \geq 0$. Geometrically, f applies a contraction or expansion to vectors in a sphere around the **reference point** $\mathbf{z}_0$. This justifies the name **radial flow**. The log-determinant is inexpensive for a radial flow since it can be shown that

$$\left| \det \frac{\partial f}{\partial \mathbf{z}} \right| = [1 + \beta h(\alpha, r)]^{d-1} [1 + \beta h(\alpha, r) + \beta h'(\alpha, r) r]$$

For VI with a radial flow of length K, the variational parameters are

$$\lambda = \{\{\mathbf{z}_{0,k}, \alpha_k, \beta_k\}\}_{k=1}^K$$

Figure 3.2 shows how planar and radial flows transform two kinds of standard base distributions.

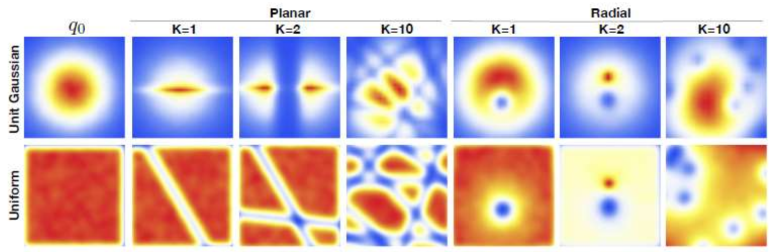


Figure 3.2: Effect of planar and radial flows on unit Gaussian base distribution and uniform base distribution. Taken from [Rezende and Mohamed, 2015].

3.4 Autoregressive Flow

Given a distribution $q(\mathbf{z})$, we can use the chain rule to obtain the factorization

$$q(\mathbf{z}) = \prod_{i=1}^{d} q(z_i \mid \mathbf{z}_{1:i-1})$$

where each of the factors is a one-dimensional conditional distribution. We can use this factorization to estimate a complex joint density $q(\mathbf{z})$ by choosing a parametric family of distributions for the conditionals $q(z_i \mid \mathbf{z}_{1:i-1})$. This strategy is known as **autoregressive density estimation**. As pointed out in [Papamakarios et al., 2017], the ordering of the variables influences the overall model and an unfortunate choice of ordering can produce very poor estimates for the joint distribution.

It is observed in both [Kingma et al., 2016] and [Papamakarios et al., 2017] that certain kinds of autoregressive models can be viewed within the framework of normalizing flows.

Consider the model where, for each $i \in \{1, \ldots, d\}$,

$$\begin{aligned}
q(z_i \mid \mathbf{z}_{1:i-1}) :=& \mathcal{N}_{\mu_i,\sigma_i}(z_i), \\
\mu_i :=& f_{\mu_i}(\mathbf{z}_{1:i-1}), \\
\sigma_i :=& \exp(\alpha_i), \\
\alpha_i :=& f_{\alpha_i}(\mathbf{z}_{1:i-1})
\end{aligned}$$

Here, f_{μ_i} and f_{α_i} are parametrized functions. Following [Jang, 2018a] and [Jang, 2018b], we shall refer to the above as the **conditional-Gaussian autoregressive model** and $\{\mu_i, \alpha_i\}_{i=1}^d$ is known as the set of **scale-and-shift statistics**. Applying the usual scale-and-shift for Gaussians, we see that a sample from $q(z_i \mid \mathbf{z}_{1:i-1})$ can be generated using

$$\begin{aligned}
z_i =& \mu_i + u_i \sigma_i, \\
u_i \sim& \mathcal{N}(0, 1)
\end{aligned}$$

and therefore, we have that

$$\mathbf{z} = f(\mathbf{u}) := \boldsymbol{\mu} + (\mathbf{u} \odot \boldsymbol{\sigma}) = \boldsymbol{\mu} + (\mathbf{u} \odot \exp(\boldsymbol{\alpha}))$$

with $\mathbf{u} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\odot$ being element-wise multiplication (i.e., Hadamard product). Note that f has inverse

$$g(\mathbf{z}) := (\mathbf{z} - \boldsymbol{\mu}) \oslash \boldsymbol{\sigma} = (\mathbf{z} - \boldsymbol{\mu}) \odot \exp(-\boldsymbol{\alpha})$$

where $\oslash$ indicates element-wise division. Hence, we see that $\mathbf{z}$ can be interpreted as being a sample from a transformed distribution such that the base distribution is a standard multivariate normal and the bijector is defined using the set of functions $\{\{f_{\mu_i}, f_{\alpha_i}\}\}_{i=1}^d$. Observe that the equation

$$u_i = \frac{z_i - \mu_i}{\sigma_i}$$

hold for all $i \in \{1, \ldots, d\}$ and, indeed, we could also have defined the autoregressive model using

$$\begin{aligned}
q(z_i \mid \mathbf{z}_{1:i-1}) :=& \mathcal{N}_{\mu_i,\sigma_i}(z_i), \\
\mu_i :=& f_{\mu_i}(\mathbf{u}_{1:i-1}), \\
\sigma_i :=& \exp(\alpha_i), \\
\alpha_i :=& f_{\alpha_i}(\mathbf{u}_{1:i-1})
\end{aligned}$$

That is, we can either compute the scale-and-shift statistics by regressing on latent variable components or by regressing on noise variable components. In both cases, we obtain an invertible map between $\mathbf{u}$ and $\mathbf{z}$.

The two different ways of defining an autoregressive model will become important when we study autoregressive flows. The most salient point is that an autoregressive model as above can be interpreted as implementing a normalizing flow of length 1. Thus, we can stack a sequence of autoregressive models together to construct a more complex normalizing flow. Furthermore, we can re-order the components of $\mathbf{z}$ in each bijector to ameliorate issues with choice of permutation of coordinates. Additionally, even though the bijector f is invertible, there are no invertibility constraints on the functions f_{μ_i} and f_{α_i}. In particular, this implies that we can parametrize them using rich classes of function approximaters such as neural networks. This enables us to introduce complex dependencies between components of a random variable. Next, the Jacobian of the bijector f has some nice computational properties. If

$$\mathbf{z} = f(\mathbf{u})$$

then the autoregressive structure means that z_i is only dependent on $\mathbf{u}_{1:i}$. This immediately implies that the Jacobian is a triangular matrix and that the determinant of this Jacobian is the product of the elements on the diagonal. Indeed, we can use the equalities

$$(f(\mathbf{u}))_i = z_i = \mu_i + u_i \sigma_i$$

to deduce that

$$\log \left| \det \frac{\partial f}{\partial \mathbf{u}} \right| = \log \prod_{i=1}^{d} \sigma_i = \sum_{i=1}^{d} \log \sigma_i = \sum_{i=1}^{d} \alpha_i$$

The log-density of final transformed distribution of an autoregressive flow is

$$\log q_K(\mathbf{z}_K) = \log q_0(\mathbf{z}_0) - \sum_{k=1}^{K} \sum_{i=1}^{d} \alpha_{k,i}$$

We shall henceforth use the term **autoregressive flow** to refer to a normalizing flow which uses a bijector derived from a conditional-Gaussian autoregressive model. Unless specified otherwise, the base distribution for an autoregressive flow is a standard Gaussian and each of the functions f_{μ_i} and f_{σ_i} is parametrized by the weights and biases of a neural network. The set of variational parameters for a VI model which uses an autoregressive flow is therefore the union of the weights and biases of these neural networks.

When implementing an autoregressive flow, the most significant computational consideration is how we compute the scale-and-shift statistics. In principle, we are free to use a separate neural network for each of the functions f_{μ_i} and f_{σ_i}, which would result in up to $2d$ distinct networks with distinct sets of weights and biases for each bijector in the flow. Clearly, for any latent space of respectable dimension, the computational load would be too heavy for practical tasks. The autoregressive flows we study below apply various techniques which make the scale-and-shift statistics feasible to compute in a broad range of scenarios. In particular, we will see that all of the flows use a single neural network per bijector. Their difference lies in the ways they manage the balance between the following three qualities of the trained distribution q_K:

(i) expressiveness of q_K;

(ii) computational complexity of generating a sample from q_K;

(iii) computational complexity of evaluating the density $q_K(\mathbf{z})$ of a test sample $\mathbf{z}$.

Intuitively, less expressive q_K make it easy to sample and evaluate the density of test samples. Indeed, NICE and RNVP sacrifice complexity to achieve good sampling and evaluation. On the other hand, IAF and MAF are able to produce rich transformed distributions. But, while it is easy to sample from IAF, it is relatively expensive to perform density estimation. For MAF, it is the reverse. Lastly, a small technical note: if sampling (resp. evaluation) from q_K is easy at test time, then sampling (resp. evaluation) is also easy during training. The converse, however, is false; for instance, IAF makes it easy to evaluate density during training even though it is expensive to evaluate density of a test sample.

3.5 Non-Linear Independent Component Estimation Flow

Although the paper [Dinh et al., 2014] was not originally written using the language of normalizing flows, the **Non-Linear Independent Component Estimation (NICE)** algorithm constructs a bijector and then chains a sequence of bijectors together to transform samples from a noise distribution into samples from an autoregressive model. Apart from the aforementioned fact that only a single neural network is used for computation of shift-and-scale statistics, NICE is an efficient procedure for two reasons. Firstly, NICE only computes shift statistics and secondly, the shifts statistics are constructed to depend only on the first k components of a noise sample with $1 < k < d$. More mathematically,

$$f(\mathbf{u}) = \boldsymbol{\mu} + \mathbf{u}$$

where

$$\begin{aligned}\boldsymbol{\mu}_{1:k} &\coloneqq \mathbf{0},\\ \boldsymbol{\mu}_{k+1:d} &\coloneqq f_\mu(\mathbf{u}_{1:k})\end{aligned}$$

The function $f_\mu : \mathbb{R}^k \to \mathbb{R}^{d-k}$ is just a neural network.

In constructing a normalizing flow using the NICE bijector, the authors of [Dinh et al., 2014] reverse the ordering of the dimensions in each step of the flow so that components which are only copied in one step may have a transformation applied to them in the succeeding step. Observe also that the Jacobian of f is simply the identity matrix and its log-determinant is zero. For a NICE flow of length K, the log-density of the final transformed distribution is

$$\log q_K(\mathbf{z}_K) = \log q_0(\mathbf{z}_0) - \sum_{k=1}^{K} \log \left| \det \frac{\partial f_k}{\partial \mathbf{z}_{k-1}} \right| = \log q_0(\mathbf{z}_0) - \sum_{k=1}^{K} 0 = \log q_0(\mathbf{z}_0)$$

Generating a sample from a NICE flow at test time is efficient since we only need to sample from the base distribution and feed it through a succession of K trained neural networks. Evaluating the density $q_K(\mathbf{z})$ of a test sample $\mathbf{z}$ is also easy. Once training is complete, we have the final set of shift statistics $\{\boldsymbol{\mu}\}_{k=1}^{K}$ and we can use the recursion

$$\begin{aligned}\mathbf{z}_K =& \mathbf{z},\\ \mathbf{z}_k =& \mathbf{z}_{k+1} - \boldsymbol{\mu}_{k+1} \forall k \in \{1, \ldots, K-1\}\end{aligned}$$

to get $\mathbf{z}_0$ whose density is trivial to compute.

3.6 Real Non-Volume Preserving Flow

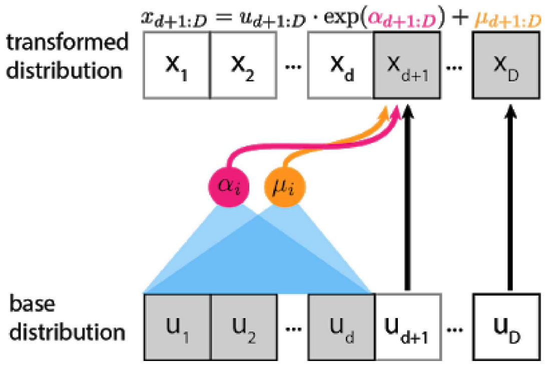
The paper [Dinh et al., 2016] generalizes the NICE bijector by introducing scale statistics. That is, the **Real Non-Volume Preserving (RNVP)** bijector is of the form

$$f(\mathbf{u}) = \boldsymbol{\mu} + (\mathbf{u} \odot \boldsymbol{\sigma})$$

where

$$\begin{aligned}\boldsymbol{\mu}_{1:k} &\coloneqq \mathbf{0}\\ \boldsymbol{\mu}_{k+1:d} &\coloneqq \left(f_{\mu,\alpha}(\mathbf{u}_{1:k})\right)_{1:d-k},\\ \boldsymbol{\sigma}_{1:k} &\coloneqq \exp(\mathbf{0}),\\ \boldsymbol{\sigma}_{k+1:d} &\coloneqq \exp\left(f_{\mu,\alpha}(\mathbf{u}_{1:k})\right)_{d-k+1:2(d-k)}\end{aligned}$$

The function $f_{\mu,\alpha} : \mathbb{R}^k \to \mathbb{R}^{2(d-k)}$ is a single neural network whose first $d-k$ outputs are the shift statistics and whose last $d-k$ outputs are the log of the scale statistics. Note that an RNVP flow is slightly more expensive than a NICE flow since the Jacobian of the RNVP bijector is not the identity matrix; rather, the last $d-k$ diagonal entries consist of $\{\sigma_i\}_{i=k+1}^d$. Apart from that, RNVP enjoys computational efficiencies similar to NICE vis-a-vis sampling and evaluation.



$x_{d+1:D} = u_{d+1:D} \cdot \exp(\alpha_{d+1:D}) + \mu_{d+1:D}$
transformed distribution
x_1 x_2 ... x_d x_{d+1} ... x_D
α_i μ_i
base distribution
u_1 u_2 ... u_d u_{d+1} ... u_D

Figure 3.3: Computation of shift-and-scale statistics in RNVP. Taken from [Jang, 2018b].

3.7 Masked Autoregressive Flow

Although RNVP provides an autoregressive flow, it is not as general as an autoregressive flow is allowed to be. This is due to the fact that the non-trivial scale-and-shift statistics of an RNVP (and therefore also NICE) bijector are only made to depend on the first k components of a noise variable. The authors of [Papamakarios et al., 2017] introduce a bijector which allows us to implement an efficient autoregressive flow while taking full advantage of the representational capacity provided by autoregressive flows.

The **Masked Autoregressive Flow (MAF)** bijector uses the equations

$$\begin{aligned} z_i =& \mu_i + u_i \exp(\alpha_i), \\ \mu_i =& f_{\mu_i}(\mathbf{z}_{1:i-1}), \\ \alpha_i =& f_{\alpha_i}(\mathbf{z}_{1:i-1}) \end{aligned}$$

directly to define the transformation of samples. The novelty is in the way they compute the set of scale-and-shift statistics $\{\mu_i, \alpha_i\}_{i=1}^d$ per bijector. Clearly, it would be inefficient to use a separate neural network for each f_{μ_i} and f_{α_i}; moreover, it seems desirable to share learned representations between all the scale-and-shift statistics computed. The authors of [Germain et al., 2015] put forward a solution which computes all the statistics using a single neural network but applies binary masks to enforce the autoregressive property. Such a neural network is called a **masked autoencoder**. Since the paper [Germain et al., 2015] is titled **Masked Autoencoder for Density Estimation (MADE)**, we shall also call such a neural network a **MADE network**. The MADE network makes it possible to have the autoregressive property for each bijector while retaining scalability of training when using MAF in a VI model. More precisely, [Papamakarios et al., 2017] uses a MADE network which, given $\mathbf{z}$, computes all μ_i and α_i using a single pass through the network. This makes it very efficient to perform density estimation for test samples. Recall that computation of the log-density $q_K(\mathbf{z})$ requires the accumulated scale statistics $\alpha_{k,i}$ and the log-density $q_0(\mathbf{z}_0)$, where $\mathbf{z}_0$ is obtained from inverting $\mathbf{z}$ through the flow. But notice that obtaining $\mathbf{z}_0$ just means repeatedly applying the inverse of the MAF bijector. This inverse is the mapping

$$\mathbf{z} \mapsto (\mathbf{z} - \boldsymbol{\mu}) \odot \exp(-\boldsymbol{\alpha})$$

and thus, inversion of the flow is only as hard as accumulating the scale-and-shift statistics. The MAF is designed precisely to make accumulation of these statistics scalable and efficient.

Unfortunately, once the parameters of a MAF have been learned, it can be expensive to generate a new sample from MAF. This is so since the defining equation of the MAF bijector is inherently sequential: we must use the recursion

$$\begin{aligned} z_1 =& \mu_1 + u_1 \exp(-\alpha_1), \\ z_{i+1} =& \mu_{i+1} + u_{i+1} \exp(-\alpha_{i+1}) = f_{\mu_{i+1}}(\mathbf{z}_{1:i}) + u_{i+1} \exp\left(f_{\alpha_{i+1}}(\mathbf{z}_{1:i})\right) \end{aligned}$$

and there is no way to parallelize this process. This implies making d sequential passes through the MADE network to accumulate the necessary scale-and-shift statistics. Contrast this with RNVP, where generating a sample requires a single pass through a neural network.

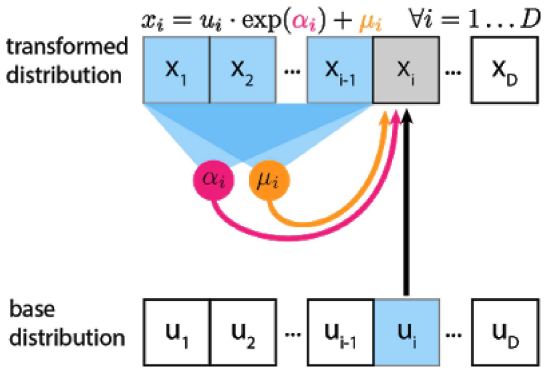


Figure 3.4: Computation of shift-and-scale statistics in MAF. Taken from [Jang, 2018b].

3.8 Inverse Autoregressive Flow

The **Inverse Autoregressive Flow (IAF)** bijector of [Kingma et al., 2016] makes the opposite trade-off as the MAF bijector. The IAF bijector uses the equations

$$\begin{aligned} z_i &= \mu_i + u_i \exp(\alpha_i), \\ \mu_i &= f_{\mu_i}\left(\mathbf{u}_{1:i-1}\right), \\ \alpha_i &= f_{\alpha_i}\left(\mathbf{u}_{1:i-1}\right) \end{aligned}$$

when transforming a sample from the base distribution. Again, a MADE network is used to compute all the scale-and-shift statistics in one pass through the network. The difference with MAF is that, in the IAF MADE network, the input to the network is the noise variable $\mathbf{u}$ rather than the latent sample $\mathbf{z}$. This means that an IAF can generate samples efficiently but density estimation is expensive. Sampling is easy since we just draw a sample from the base distribution, pass it once through the MADE network to get all the scale-and-shift statistics and apply the IAF bijector. As with MAF, density evaluation requires the bijector inverse

$$\mathbf{z} \mapsto \mathbf{u} = (\mathbf{z} - \boldsymbol{\mu}) \odot \exp(-\boldsymbol{\alpha})$$

However, for IAF, the components of $\mathbf{u}$ have to be computed in sequence. That is, we are forced to carry out the recursion

$$\begin{aligned} u_1 &= (z_1 - \mu_1) \exp(-\alpha_1), \\ u_{i+1} &= (z_{i+1} - \mu_{i+1}) \exp(-\alpha_{i+1}) = \left(z_{i+1} - f_{\mu_{i+1}}\left(\mathbf{u}_{1:i}\right)\right) \exp\left(f_{\alpha_{i+1}}\left(\mathbf{u}_{1:i}\right)\right) \end{aligned}$$

We therefore require d sequential passes to invert the IAF bijector. There is a precise sense in which MAF and IAF are inverse processes to each other. See [Papamakarios et al., 2017] for details.

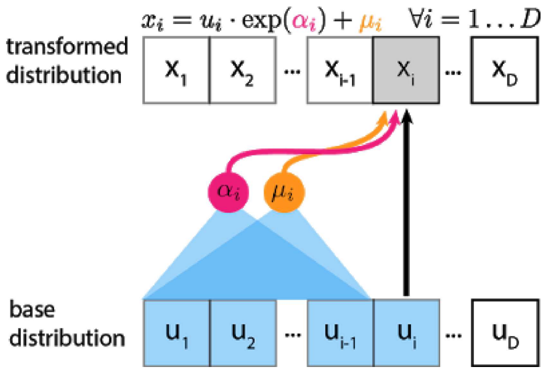


Figure 3.5: Computation of shift-and-scale statistics in IAF. Taken from [Jang, 2018b].

Chapter 4

Variational Inference and Deep Learning

Our discussion in the foregoing chapters focused on a suite of general techniques for practical VI using powerful surrogate posteriors. In our discussion of normalizing flows, we saw that autoregressive flows such as MAF and IAF allow us to endow VI models with rich approximate posteriors which are capable of matching features in real-world data. As we also learned, the choice of surrogate posterior, likelihood and prior ultimately comes down to choosing a sufficiently expressive class of parametrized functions. Computational efficiency of approximate inference and estimation of model parameters is thus contingent on the chosen class of functions having good computational properties.

Neural networks are precisely such a family of functions. This provides a natural point of entry for deep learning into VI. Current best practice therefore combines amortized inference and normalizing flows with deep learning to obtain VI models which achieve state-of-the-art performance on several tasks (see [Zhang et al., 2017] for specifics). Since we have already analyzed the key ideas and derived the main equations which enable VI on practical tasks, the purpose of the current chapter is to point out the places where deep neural networks lend their representational power. Thus, we focus on relatively terse pointers on the interaction between VI and deep learning. We mention in advance that the VAE is the primary model of interest for our applications to document modeling.

4.1 Deep Latent Gaussian Models

A **deep latent Gaussian model (DLGM)** is a latent variable model which specifies a standard multivariate normal for the prior $p(\mathbf{z})$ over latent space and the likelihood

$p_\theta(\mathbf{x} \mid \mathbf{z})$ is parametrized by a neural network. More precisely, the neural network involved takes as input a sample $\mathbf{z}$ from latent space and outputs parameters for the probability distribution over datapoints $\mathbf{x}$ given the $\mathbf{z}$. The distribution in this case is Gaussian and the means and variance of the Gaussian are exactly the outputs of the neural network.

Equivalently, as per [Hoffman, 2017], a DLGM can be specified using the following generative process:

(i) sample $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$;

(ii) compute vector-valued $g_\theta(\mathbf{z})$, where g_θ represents a neural network with θ the set of weights and biases;

(iii) sample $\mathbf{x}$ from $\mathcal{N}(\mathbf{x} \mid g_\theta(\mathbf{z}))$, where $g_\theta(\mathbf{z})$ is the concatenation of the mean and (co)variance matrices.

The paper [Rezende et al., 2014] which originally provided an efficient and scalable method for learning in DLGMs presented an alternative view which is strictly more general. That paper also derived equations for training DLGMs with stochastic backpropagation. However, we shall use the term DLGM to refer to the slightly simpler version defined by the generative process above. The main takeaway is that, since g_θ is a neural network, the model parameters are the weights and biases θ and that we can apply deep learning to perform parameter estimation. In practice, we apply VI to approximately optimize the marginal likelihood of the data and the surrogate posterior chosen is also parametrized by a neural network. Note, however, that a DLGM only specifies the prior and likelihood. Additionally, we can change the form of the likelihood depending on the type of data we are modeling. Indeed, in the final step of the generative process, we can sample from $f(g_\theta(\mathbf{z}))$, where f produces a distribution over datapoints provided the outputs of g_θ. So, for instance, f can compute a Bernoulli distribution in the case of discrete data. Hence, DLGMs form a very flexible class of density estimators.

4.2 Variational Auto-Encoders

The **Variational Auto-Encoder (VAE)** of [Kingma and Welling, 2013] can be thought of as a more specific version of the DLGM. In fact, both [Rezende et al., 2014] and [Kingma and Welling, 2013] were published around the same time and offered similar algorithms for learning latent variable models using VI. Like the DLGM, VAE uses a neural network with weights and biases θ to parametrize the likelihood. The original model in [Kingma and

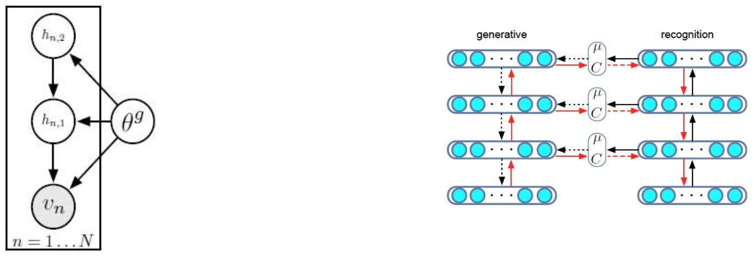


Figure 4.1: Graphical model (left) and computational graph (right) for DLGMs. Red arrows represent forward propagation, black arrows backpropagation. Solid arrows are deterministic calculations and dashed arrows are stochastic computations. Taken from [Rezende et al., 2014].

Welling, 2013] also uses a standard multivariate normal prior over latent space. However, the VAE is more specific in performing amortized inference using a neural network. That is, to obtain the distribution $q_\lambda(\mathbf{z} \mid \mathbf{x})$ over latent space when given a datapoint $\mathbf{x}$, we pass $\mathbf{x}$ through a neural network with weights and biases ϕ and the outputs are the parameters of $q_\lambda(\mathbf{z} \mid \mathbf{x})$. We shall slightly abuse notation and write q_ϕ for the surrogate posterior in this situation. The neural network with parameters ϕ is called the **recognition network** or **inference network**. We can view the inference network as predicting local variational parameters using datapoints. Given local variational parameters, we are able to sample from the surrogate posterior. This is done repeatedly during training, where we optimize the ELBO using the AEVB algorithm. Furthermore, as with DLGMs, a second neural network called the **generative network** takes a sample from latent space as input and outputs a point in data space. When we place the inference network and generative network next to each other, we get an architecture which resembles an **autoencoder (AE)** (see [Baldi, 2012]). Much like a deterministic AE, the complete VAE architecture takes a datapoint as input and produces a reconstruction of the input at the top layer. But there is a stochastic operation in the bottleneck layer which is key to variational inference. Notice also that since the AEVB algorithm is used, the reparametrization trick needs to be available for the family of distributions for the surrogate posterior.

Observe that the VAE provides a general framework for performing VI as long as the following ingredients are available: a prior, a generative network, an inference network and the reparametrization trick. Indeed, [Dillon et al., 2017] takes this idea seriously and gives examples of VAEs with very rich surrogate posteriors and likelihoods to produce models which perform well on standard benchmarks for a diversity of tasks. Note that even the prior can be implemented using a neural network. The TensorFlow Distributions library, as described in [Dillon et al., 2017], combines **Automatic Differentiation Variational Inference (ADVI)** from [Kucukelbir et al., 2017] with the generic VAE model to provide a very powerful class of VI models. Very briefly, ADVI uses the generic formulas we derived for the ELBO and its gradients in conjunction with a generalized form of the reparametrization trick to automatically produce a VI procedure. ADVI only requires specification of differentiable likelihood, prior and surrogate posterior. We have made extensive use of the TensorFlow Distributions library in our own work.

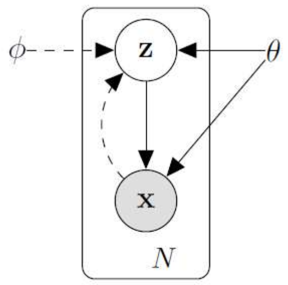


Figure 4.2: Graphical model representing computation in a VAE. Taken from [Kingma and Welling, 2013].

4.3 Neural Variational Inference and Learning

For completeness, we also mention the **Neural Variational Inference and Learning (NVIL)** technique of [Mnih and Gregor, 2014]. Just like BBVI, NVIL optimizes the ELBO by estimating the gradients using Monte Carlo samples and then applies variance reduction techniques to the estimate of $\nabla_\lambda \mathcal{L}(\theta, \lambda; \mathbf{x})$. Unlike BBVI, the variance reduction techniques of NVIL are more generally applicable and NVIL does not assume a mean-field factorization for the surrogate posterior. Recall that the gradient with respect to variational parameters is

$$\nabla_\lambda \mathcal{L}(\theta, \lambda; \mathbf{x}) = \mathbb{E}_{q_\lambda}\left[(\log p_\theta(\mathbf{x} \mid \mathbf{z}) - \log q_\lambda(\mathbf{z} \mid \mathbf{x}) + \log p_\theta(\mathbf{z})) \cdot \nabla_\lambda \log q_\lambda(\mathbf{z} \mid \mathbf{x})\right]$$

The term

$$l_\lambda(\mathbf{x}, \mathbf{z}) := \log p_\theta(\mathbf{x} \mid \mathbf{z}) - \log q_\lambda(\mathbf{z} \mid \mathbf{x}) + \log p_\theta(\mathbf{z})$$

is called the **learning signal** in [Mnih and Gregor, 2014] and the paper offers tricks for variance reduction by studying the learning signal. We now give short descriptions of these tricks

(1) Centering the learning signal: It can be shown that

$$\mathbb{E}_{q_\lambda}\left[l_\lambda(\mathbf{x}, \mathbf{z}) \cdot \nabla_\lambda \log q_\lambda(\mathbf{z} \mid \mathbf{x})\right] = \mathbb{E}_{q_\lambda}\left[(l_\lambda(\mathbf{x}, \mathbf{z}) - c) \cdot \nabla_\lambda \log q_\lambda(\mathbf{z} \mid \mathbf{x})\right]$$

for any c independent of $\mathbf{z}$. We can use a neural network with weights and biases ψ to compute an appropriate $C_\psi(\mathbf{x})$ for each observation $\mathbf{x}$ and reduce the variance contributed by $\mathcal{L}(\theta, \lambda; \mathbf{x})$. The quantity $C_\psi(\mathbf{x})$ is called the **baseline** associated to $\mathbf{x}$.

(2) Variance normalization: Once the learning signal $l_\lambda(\mathbf{x}, \mathbf{z})$ is centered, we can divide it by a running estimate of it standard deviation.

(3) Local learning signals: When using an MLP inference network as part of our VI model, we can think of our surrogate posterior as introducing a hierarchy of latent variables. We can reduce the gradient variance further by explicitly using this hierarchical structure and carrying out centering and variance normalization on a layer-by-layer basis.

The reader is invited to consult [Mnih and Gregor, 2014] for further details.

4.4 Sigmoid Belief Networks

Recall that a **belief network** (see [Pearl, 1986]) is specified by a directed acyclic graph where the nodes represent random variables and the joint distribution of these variables can be factorized as

$$\Pr(\mathbf{v}) = \prod_{t=1}^{T} \Pr\left(v_t \mid \mathbf{v}_{\mathrm{pa}(t)}\right)$$

Here, $\mathbf{v}_{\mathrm{pa}(t)}$ is the set of parent nodes of v_t as indicated by the directed edges of the graph. As the name suggests, a **sigmoid belief network (SBN)** (see [Neal, 1992]) is a type of belief network; in addition, an SBN assumes that all random variables are binary and that each of the conditional distributions

$$\Pr\left(v_t \mid \mathbf{v}_{\mathrm{pa}(t)}\right)$$

is a logistic regressor. Thus, an SBN is parametrized by a vector θ which includes parameters for each of the logistic functions. Note that this implies that an SBN estimates the density over a collection of bit vectors.

We can create a generative model for data by partitioning the random variable $\mathbf{v}$ (also called the **state** of the SBN in [Neal, 1992]) into two subsets

$$\mathbf{v} = (\mathbf{z}, \mathbf{x})$$

where $\mathbf{z}$ is the set of latent variables or **hidden units** and $\mathbf{x}$ is the set of observations or **visible units**. There is a clear connection with neural networks if we arrange the random variables into layers such that each hidden layer consists of a subset of the hidden units. This corresponds to a partitioning of the latent variables

$$\mathbf{z} = (\mathbf{h}_1, \ldots, \mathbf{h}_M)$$

with $\mathbf{h}_m$ being the set of hidden units in the m^{th} layer and M being the number of hidden layers. As with the VAE, the network thus constructed is called a **generative network**.

As might be expected, inference of the hidden units given a set of observations is generally intractable. Moreover, parameter estimation requires inference as a subroutine. The paper [Neal, 1992] provides a learning procedure which uses Gibbs sampling for inference and gradient descent for model parameter updates. This is obviously not very scalable and for the SBN, mixing times for a Gibbs sampler are typically very long since the latent variables tend to be highly correlated. Both [Hinton et al., 1995] and [Saul et al., 1996] present approaches for approximating the exact posterior. The latter uses a mean-field variational inference approach in conjunction with the generative network and optimizes a form of the ELBO specially derived for an SBN. However, the **wake-sleep algorithm** of [Hinton et al., 1995] deviates from the usual VI approach. This algorithm is specifically designed for an SBN which takes the form of a layered feed-forward network and uses an **inference network** for computing parameters of an approximate posterior distribution. The inference network topology is simply a mirror of the generative network and a form of variational EM is applied for inference and parameter estimation. This combination is

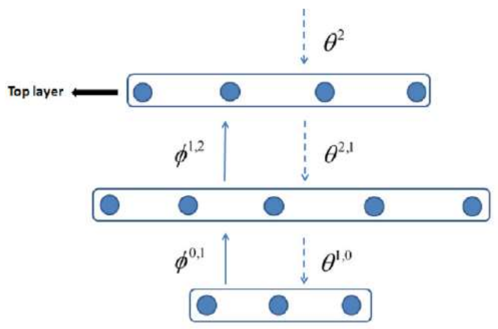


Figure 4.3: Helmholtz machine. Taken from [Hu et al., 2017].

known as a **Helmholtz machine** (see figure 4.4), which was introduced in [Dayan et al., 1995] and which also applies variational techniques. Unfortunately, the construction of the training procedure for Helmholtz machines does not optimize a lower bound of the evidence and there are no formal guarantees for convergence. This limitation is pointed out in [Saul et al., 1996], [Mnih and Gregor, 2014] and [Kingma and Welling, 2013]. In particular, both [Saul et al., 1996] and [Mnih and Gregor, 2014] provide theoretically sound objective functions for training an SBN. As we discussed in our section on NVIL, the approach of [Mnih and Gregor, 2014] is quite general but [Saul et al., 1996] provides lower variance gradients of the objective since the mean-field lower bound in that paper is specifically tailored for an SBN.

To summarize, we have several ways to train an SBN which scales well with the size of the data. We can apply the wake-sleep algorithm or a mean-field variational inference procedure or NVIL. All of these methods apply variational optimization in one way or another. Wake-sleep and NVIL use an inference network whereas the mean-field VI of [Saul et al., 1996] does not. On the other hand, mean-field VI and NVIL optimize a lower bound to the evidence but wake-sleep does not. Finally, even though SBN and related techniques such as Boltzmann machines, Helmholtz machines, deep belief networks, etc., are not so fashionable at time of writing, the SBN provides a solid algorithm against which to compare newer methods. As well, these ideas are an important precursor to more modern techniques like DLGMs and VAEs. In particular, the idea of using an inference network for performing approximate inference has been very influential.

4.5 Deep Autoregressive Networks

The **deep autoregressive network (DARN)** of [Gregor et al., 2014] is very similar in spirit to the VAE. Just like the VAE, DARN specifies a prior over latent space, an inference network which computes local variational parameters from global parameters ϕ and a generative network whose weights and biases θ serve as model parameters. As well, DARN jointly optimizes the two sets of parameters with stochastic optimization rather than relying on a variational EM procedure and we get an AE-like architecture when the inference network is placed next to the generative network. Unlike the VAE, however, DARN maximizes the function

$$\mathcal{L}_{\text{Helm}}(\theta, \lambda; \mathbf{x}) = \mathbb{E}_{q_\lambda}\left[\log_2 p_\theta(\mathbf{x} \mid \mathbf{z}) - \log_2 q_\lambda(\mathbf{z} \mid \mathbf{x}) + \log_2 p_\theta(\mathbf{z})\right]$$

The quantity $\mathcal{L}_{\text{Helm}}$ is known as the **Helmholtz variational free energy** and lower bounds $\log_2 p_\theta(\mathbf{x})$. Clearly, the Helmholtz free energy is just the ELBO with natural logarithms replaced by binary logarithms (more concisely, replace nats with bits). The reason for the binary logarithms is that, as shown in [Hinton and Zemel, 1993], the Helmholtz free energy is derived using the **minimum description lengths (MDL)** of the probability distributions involved. Briefly, the MDL is a concept from information theory which measures compression and minimizing MDL is a desirable goal when training models which extract parsimonious representations (or codes) from data. See [Hinton and van Camp, 1993] for a detailed discussion.

Additionally, DARN uses autoregressive models for the prior, likelihood and surrogate posterior. To make this more precise, let us introduce some notation similar to [Gregor et al., 2014]. Consider an AE with successive hidden layers $\{\mathbf{h}^{(l)}\}_{l=1}^{n_{\text{layers}}}$ with $\mathbf{h}^{(l)}$ consisting of $n_h^{(l)}$ units. For $l > 0$, each of the $\mathbf{h}^{(l)}$ is allowed to be stochastic. We also define $\mathbf{h}^{(n_{\text{layers}+1})} := \emptyset$ and write $\mathbf{h}^{(0)}$ for the input layer which represents points in data space.

The likelihood and surrogate posterior for DARN then satisfy the conditions

$$p_\theta\left(\mathbf{h}^{(l)} \mid \mathbf{h}^{(l+1)}\right) = \prod_{j=1}^{n_h^{(l)}} p_\theta\left(\mathbf{h}_j^{(l)} \mid \mathbf{h}_{1:j-1}^{(l)}, \mathbf{h}^{(l+1)}\right)$$

$$q_\phi\left(\mathbf{h}^{(k)} \mid \mathbf{h}^{(k-1)}\right) = \prod_{j=1}^{n_h^{(k)}} q_\phi\left(\mathbf{h}_j^{(k)} \mid \mathbf{h}_{1:j-1}^{(k)}, \mathbf{h}^{(k-1)}\right)$$

for all $l \in \{0, \ldots, n_{\text{layers}}\}$ and for all $k \in \{1, \ldots, n_{\text{layers}}\}$. Furthermore, the prior factorizes as

$$p_\theta(\mathbf{z}) = \prod_{j=1}^{n_h^{(\text{latent})}} p_\theta\left(z_j \mid \mathbf{z}_{1:j-1}\right)$$

where $n_h^{(\text{latent})}$ is equal to the dimensionality of the latent space. Observe that the autoregressive property holds *within* each layer. The distribution over vectors output by $\mathbf{h}^{(l)}$ (resp. $\mathbf{h}^{(k)}$) is dependent on the output of $\mathbf{h}^{(l+1)}$ (resp. $\mathbf{h}^{(k-1)}$) as in an ordinary feed-forward network but also the j^{th} unit of $\mathbf{h}^{(l)}$ (resp. $\mathbf{h}^{(k)}$) depends on the output of the preceding $j-1$ units of $\mathbf{h}^{(l)}$ (resp. $\mathbf{h}^{(k)}$).

In principle, we can use arbitrarily complex autoregressive models for each of the conditional distributions above. However, computational efficiency may force us to stick with relatively simple distributions. Indeed, [Gregor et al., 2014] initially parametrizes these distributions using sigmoid functions. Moreover, the paper introduces extra efficiency by choosing sparse stochastic hidden layers. This more specific version of DARN is called **fast deep autoregressive network (fDARN)**. Lastly, note that we are free to train the DARN architecture using either NVIL or wake-sleep to maximize the ELBO. The original stochastic optimization scheme in [Gregor et al., 2014] uses Monte Carlo estimates for the Helmholtz free energy and its gradients with a relatively simple baseline (in the sense of NVIL) to reduce variance of gradient estimates.

4.6 Deep Boltzmann Machines

Recall that an **energy-based model (EBM)** assigns probabilities, to vectors in a specified domain, using a formula

$$\Pr(\mathbf{v}) = \frac{1}{Z} \exp\left(-E(\mathbf{v})\right)$$

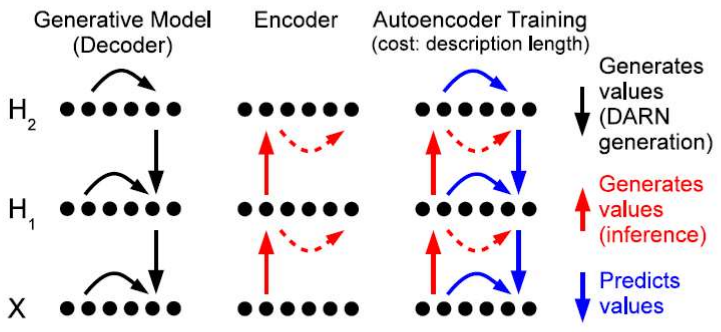

Generative Model (Decoder)
Encoder
Autoencoder Training (cost: description length)
H2
H1
X
Generates values (DARN generation)
Generates values (inference)
Predicts values

Figure 4.4: Visual representation of deep autoregressive network (DARN). Taken from [Gregor et al., 2014].

where $E(\mathbf{v})$ is an **energy function** and

$$Z := \int_{\mathbf{v}} \exp\left(-E(\mathbf{v})\right)$$

is the **partition function**. For an energy-based latent variable model, we partition $\mathbf{v}$ as $\mathbf{v} = (\mathbf{z}, \mathbf{x})$ with $\mathbf{z}$ being hidden variables or units and $\mathbf{x}$ being observed variables or units. We can compute the negative log-probability of an observation $\mathbf{x}$ as

$$\begin{aligned} -\log \Pr(\mathbf{x}) &= \log\left(\int_{\mathbf{z}} \Pr(\mathbf{z}, \mathbf{x})\right) \\ &= -\log\left(\int_{\mathbf{z}} \frac{1}{Z} \exp\left(-E(\mathbf{z}, \mathbf{x})\right)\right) \\ &= -\log\left(\frac{1}{Z} \int_{\mathbf{z}} \exp\left(-E(\mathbf{z}, \mathbf{x})\right)\right) \\ &= \log Z + \mathcal{F}(\mathbf{x}) \end{aligned}$$

with

$$\mathcal{F}(\mathbf{x}) := -\log\left(\int_{\mathbf{z}} \exp\left(-E(\mathbf{z}, \mathbf{x})\right)\right)$$

being the **free energy**. It is easy to see that

$$\Pr(\mathbf{x}) = \frac{1}{Z} \exp\left(-\mathcal{F}(\mathbf{x})\right)$$

and that

$$Z = \int_{\mathbf{x}} \exp\left(-\mathcal{F}(\mathbf{x})\right)$$

That is, the probability distribution over observations is also an energy-based model but using the free energy function derived from the defining energy function.

Model design for an EBM amounts to choosing an energy function. Typically, the energy function is parametrized by a family of functions indexed by θ and parameter estimation for the model means determining the parameters θ which maximize the likelihood of a given training set of observations. As shown in [Bengio, 2009], the gradient with respect to θ of the log-evidence can be expressed in terms of the energy

$$\frac{\partial}{\partial\theta} \log p_\theta(\mathbf{x}) = -\mathbb{E}_{p_\theta(\mathbf{z}|\mathbf{x})}\left[\frac{\partial}{\partial\theta} E(\mathbf{z}, \mathbf{x})\right] + \mathbb{E}_{p_\theta(\mathbf{z},\tilde{\mathbf{x}})}\left[\frac{\partial}{\partial\theta} E(\mathbf{z}, \tilde{\mathbf{x}})\right]$$

or in terms of the free energy

$$\frac{\partial}{\partial\theta} \log p_\theta(\mathbf{x}) = -\frac{\partial}{\partial\theta}\mathcal{F}(\mathbf{x}) + \mathbb{E}_{p_\theta}\left[\frac{\partial}{\partial\theta}(\mathbf{x})\mathcal{F}(\mathbf{x})\right]$$

In either case, we are required to compute expectations to compute the gradient and to calculate the derivative of either the energy or the free energy. Since the energy function is a model choice, we can usually design the energy such that its derivative, as well as the free energy and its derivative, is easy. Calculation of the expectations is the more difficult task and, indeed, the expectation terms are typically not analytically tractable. Thus, we compute stochastic estimators for the gradient using Monte Carlo samples from the relevant distributions. For the gradient in terms of the energy, we require samples from the posterior $p_\theta(\mathbf{z} \mid \mathbf{x})$ and from the joint distribution $p_\theta(\mathbf{z}, \tilde{\mathbf{x}})$. But note that if we are able to sample from the joint, then we can also compute a Monte Carlo approximation of the gradient in terms of the free energy.

A **deep Boltzmann machine (DBM)** ([Salakhutdinov and Hinton, 2009a], figure 4.5) is an EBM which admits an affine energy function based on an undirected graphical model. The random variables of a DBM are organized into successive layers with hidden units belonging to one of several hidden layers. Borrowing the notation of DARN, the energy function of a DBM can be written as

$$E(\mathbf{h}^{(1)}, \ldots, \mathbf{h}^{(n_{\text{layers}})}, \mathbf{x}) = -\sum_{l=1}^{n_{\text{layers}}} \left(\mathbf{h}^{(l)}\right)^T \mathbf{W}^{(l)} \mathbf{h}^{(l-1)} - \sum_{l=1}^{n_{\text{layers}}} \left(\mathbf{h}^{(l)}\right)^T \mathbf{b}^{(l)}$$

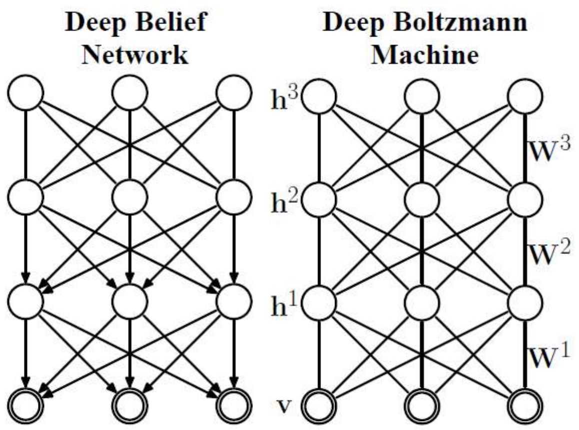


Figure 4.5: A DBM is an undirected version of DBN. Taken from [Salakhutdinov and Hinton, 2009a].

Note that $\mathbf{h}^{(0)} = \mathbf{x}$. The weights $\{\mathbf{W}^{(l)}\}_{l=1}^{n_{\text{layers}}}$ and biases $\{\mathbf{b}^{(l)}\}_{l=1}^{n_{\text{layers}}}$ form the set of model parameters for the DBM. When there is only one hidden layer in a DBM, we get a **restricted Boltzmann machine (RBM)** or **Harmonium** (see 4.6). Observe that all random variables in a DBM or RBM are assumed to be binary. Thus, a DBM acts a density estimator for bit vectors.

As with any other latent variable model, we wish to perform inference and parameter estimation in a DBN given a set of observations. Additionally, inference is necessary for parameter estimation in any case. Directly optimizing for the model parameters using the formula for the gradient of the log-evidence is computationally challenging due to the expectation terms. Boltzmann machine learning admits an MCMC procedure which uses two Gibbs sampling chains (one for the posterior and one for the joint; consult [Bengio, 2009]). However, this algorithm is impractical for DBMs trained on large datasets. For RBMs, minimizing a different function known as **contrastive divergence (CD)** provides an efficient alternative. CD uses K steps of blocked Gibbs sampling (for this reason, CD is sometimes called CD-K) and this sampling is required for every single parameter update. In practice, very small values of K (even $K = 1$) often suffice.

Unfortunately, even CD-K is not efficient enough for DBMs with multiple hidden layers and we must once again resort to approximating the exact posterior $p_\theta(\mathbf{z} \mid \mathbf{x})$. Thus, [Salakhutdinov and Hinton, 2009a] employs a mean-field variational inference algorithm

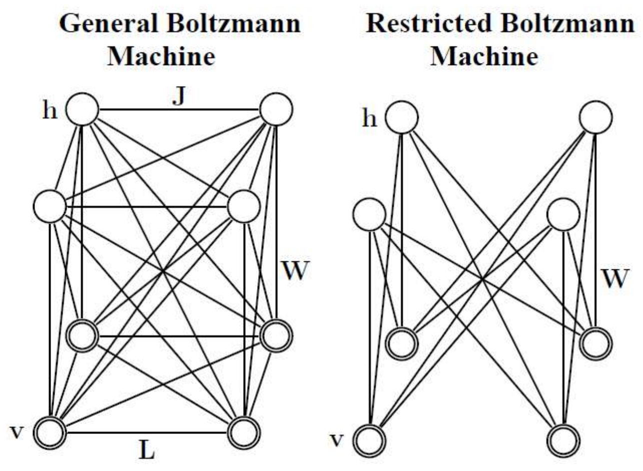


Figure 4.6: An RBM has only one hidden layer. Taken from [Salakhutdinov and Hinton, 2009a].

for training a DBM. A form of variational EM is used for training. The ELBO in this case involves the partition function Z, which is a function of θ, and MCMC must be used to estimate the partition function when updating model parameters (given variational parameters from the variational E-step). Follow-up work in [Salakhutdinov and Larochelle, 2010] accelerates inference in the DBM by also applying a set of recognition weights. Although not explicitly mentioned in the paper, the end-to-end DBM training procedure (minus the weight initialization with pre-training) of [Salakhutdinov and Larochelle, 2010] is like an undirected version of the Helmholtz machine. That is, there is a set of inference weights and a set of recognition weights and these are alternately updated with a variational EM algorithm.

Chapter 5

Neural Document Models

Our ultimate goal is, of course, document modeling. More accurately, we wish to automatically discover good representations for sentences or even entire documents using unlabeled data[1]. Our task falls under the umbrella of **unsupervised representation learning**. The representations (or features) are treated as hidden variables which must be inferred given observations. Latent variable models are therefore a natural framework for this type of machine learning.

In the previous chapters, we have explored techniques for creating powerful latent variable models using variational inference. We then saw that deep neural networks provide efficient learning for a broad class of expressive VI models. In particular, the use of an inference network, which predicts variational parameters using data, is crucial for representational capacity and scalable optimization of the objective function. Hence, we now have a robust framework for applying latent variable models to document modeling. The present chapter is dedicated to such applications. We first describe some existing approaches which use deep learning to attack document modeling. Some of these apply variational inference while others do not. The latter are presented to provide strong baselines and also to give some flavor of document modeling without VI models. Subsequently, we present our own additions and modifications to the approaches combining VI and deep learning.

[1]What counts as a "good" representation is always a little subjective. If the goal is to extract features which are useful in a downstream task (e.g., classification), then representations are "good" insofar as they improve outcomes in the downstream task. However, like many authors working in the field, we wish to be agnostic to context and "good" will usually mean one of two things: (a) achieving low perplexity on a held-out test set or (b) achieving good performance on a document retrieval task constructed using a held-out test set.

5.1 Replicated Softmax

Previously, we saw that the RBM provides a latent variable model which can be trained from a given set of observations. An MFVI technique allows scalable training of the RBM. Recall, however, that each observation is assumed to be a (high-dimensional) bit vector. So, additional steps are required in order to apply RBMs to the problem of document modeling. Let us introduce some notation before proceeding. We shall use $\mathbf{x}$ to denote a document which occurs as a sequence

$$\mathbf{x} = (x_1, \dots, x_d)$$

of integer indexes, i.e., $x_i \in \{1, \dots, V\}$ with V being the size of the vocabulary we are working with. It is assumed that we have a dictionary whose keys are all the integers between 1 and V and whose values are the words in the vocabulary. We write $\mathbf{n}(\mathbf{x})$ for the **Bag-of-Words (BoW)** vector associated to $\mathbf{x}$. That is, $\mathbf{n}(\mathbf{x})$ is a vector of size V where the i^{th} element is the number of times that the word with index i occurs in $\mathbf{x}$.

The **replicated softmax (RSM)** model of [Salakhutdinov and Hinton, 2009b] is an EBM whose energy function is defined by

$$E(\mathbf{z}, \mathbf{h}) := -\mathbf{z}^T \mathbf{W} \mathbf{n}(\mathbf{x}) - \mathbf{b}^T \mathbf{n}(\mathbf{x}) - d\mathbf{c}^T \mathbf{z}$$

where $\theta = \{\mathbf{W} \in \mathbb{R}^{H \times V}, \mathbf{b} \in \mathbb{R}^V, \mathbf{c} \in \mathbb{R}^H\}$ is the set of model parameters and H is the dimensionality of the latent space. Additionally, $\mathbf{z}$ represents a latent, stochastic binary variable. Clearly, the energy function of RSM is just a slight modification of that of the RBM. The major differences lies in two places: the visible layer is no longer binary and the bias term associated to the hidden layer is scaled by d.

Under this model, we obtain the following conditional probabilities

$$p_\theta(\mathbf{z} \mid \mathbf{x}) = \prod_{j=1}^{H} p_\theta(z_j \mid \mathbf{x}),$$

$$p_\theta(\mathbf{x} \mid \mathbf{z}) = \prod_{i=1}^{d} p_\theta(x_i \mid \mathbf{z})$$

In words, the distribution over latent variables conditioned on observations factorizes completely and vice-versa. Moreover, each of the one-dimensional factors can be expressed as follows.

$$p_\theta(z_j = 1 \mid \mathbf{x}) = \sigma\left(dc_j + \sum_{i=1}^{d} W_{j,x_i}\right),$$
$$p_\theta(x_i = w \mid \mathbf{z}) = \frac{\exp\left(b_w + \mathbf{z}^T\mathbf{W}_{:,w}\right)}{\sum_{w'} \exp\left(b_{w'} + \mathbf{z}^T\mathbf{W}_{:,w'}\right)}$$

where w is the index of an arbitrary word in our vocabulary. Observe that $p_\theta(x_i = w \mid \mathbf{z})$ is just a softmax function. As with an ordinary RBM, the conditional probabilities $p_\theta(x_i = w \mid \mathbf{z})$ are approximated with a factorized approximate distribution and inference amounts to MFVI. The RSM model extracts a hidden representation $\mathbf{z}$ given a document $\mathbf{x}$ and, once the model parameters have been learned, the conditional probability equations above can be used to calculate the likelihood of an arbitrary document.

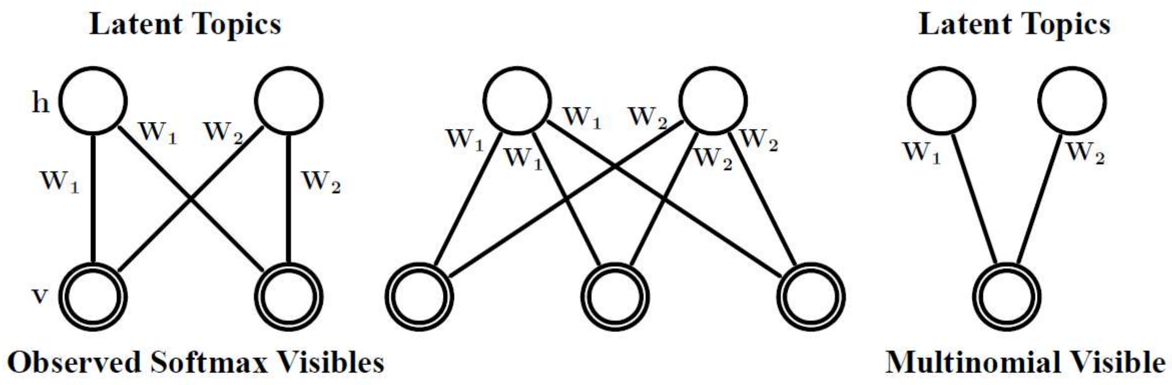


Figure 5.1: Visual representation of the RSM. Taken from [Salakhutdinov and Hinton, 2009b].

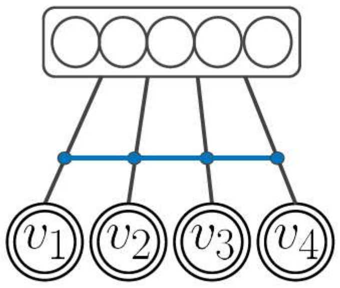


Figure 5.2: Alternative representation of the RSM. Blue lines indicate shared parameters. Taken from [Lauly et al., 2017].

5.2 Document Neural Autoregressive Distribution Estimation

The **Neural Autoregressive Distribution Estimator (NADE)** of [Larochelle and Murray, 2011] is, like the DBM or RBM, a technique for estimating a distribution $p_\theta(\mathbf{x})$ over bit vectors $\mathbf{x} \in \{0,1\}^D$ given a set of observations $\{\mathbf{x}^{(j)}\}_{j=1}^N$. As the name suggests, NADE is an autoregressive model, i.e., we first factorize the overall distribution

$$p_\theta(\mathbf{x}) = \prod_{i=1}^{d} p_\theta(x_i \mid \mathbf{x}_{1:i-1})$$

and specify a parametrized model for each of the factors $p_\theta(x_i \mid \mathbf{x}_{1:i-1})$. In this case, $p_\theta(x_i \mid \mathbf{x}_{1:i-1})$ is computed using a fully-connected neural network with three layers: an input layer with $i-1$ binary units, a hidden layer with H units and an output layer with one unit. The input layer accepts a bit vector which represents the first $i-1$ bits, $\mathbf{x}_{1:i-1}$, of an observation $\mathbf{x}$. The input is then mapped into $\mathbb{R}^H$ with the dimensionality H and the hidden layer activation being hyperparameters for NADE. Lastly, the output of the hidden layer is passed through a sigmoid layer to produce a scalar value between zero and one.

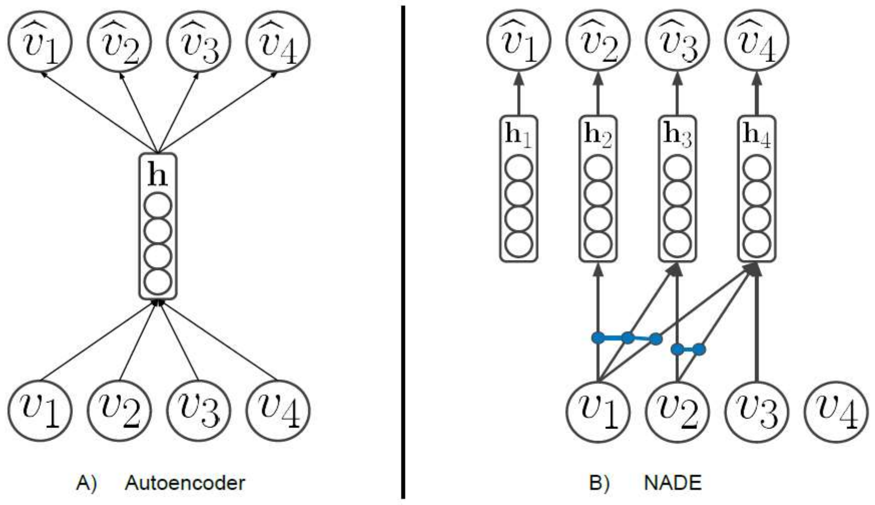


Figure 5.3: Comparison between autoencoder and NADE. The AE has a single hidden bottleneck whereas the hidden layers in NADE enforce autoregressivity. Taken from [Lauly et al., 2017].

Observe that NADE employs d separate neural networks, each containing a hidden layer with H units and an output layer with one unit. Clearly, it would training and sampling would be slow if we had separate sets of parameters for the different neural networks. Hence, NADE introduces parameter sharing as follows. The complete set of parameters consists only of two matrices and two vectors

$$\theta = \{\mathbf{W} \in \mathbb{R}^{H \times d}, \mathbf{V} \in \mathbb{R}^{d \times H}, \mathbf{b} \in \mathbb{R}^{d}, \mathbf{c} \in \mathbb{R}^{H}\}$$

During a forward pass, a list of hidden vectors $\{\mathbf{h}_i\}_{i=1}^{d}$ is computed using the formula

$$\mathbf{h}_i \coloneqq \mathbf{g}\left(\mathbf{c} + \mathbf{W}_{:,1:i-1}\mathbf{x}_{1:i-1}\right)$$

Subsequently, we calculate the probability values $\{p_i\}_{i=1}^{d}$ as

$$p_i \coloneqq \sigma\left(b_i + \mathbf{V}_{i,:}^{T}\mathbf{h}_i\right)$$

Note that

$$p_\theta(x_i = 1 \mid \mathbf{x}_{1:i-1}) = p_i$$

when the NADE input is $\mathbf{x}$. We see that the parameters used to compute $p_\theta(x_i = 1 \mid \mathbf{x}_{1:i-1})$ are re-used to compute $p_\theta(x_{i+1} = 1 \mid \mathbf{x}_{1:i})$ and moreover, NADE computes all the hidden layer outputs using $\mathcal{O}(dH)$ operations. As well, by construction, it is easy to compute the log-likelihood of an observation given values for model parameters. Parameters in NADE can thus be learned by maximizing the log-likelihood of the training data using SGD.

The NADE algorithm must be modified somewhat when modeling a distribution over documents. Suppose we have a vocabulary of size V and we have established a dictionary between integers $\{1, \ldots, V\}$ and the set of words in our vocabulary. Assume for now that each document is represented as a sequence of d indexes, i.e., $\mathbf{x} \in \{1, \ldots, V\}^d$. In this case, instead of a weight matrix $\mathbf{W} \in \mathbb{R}^{H \times d}$, we shall have a word representation matrix $\mathbf{W} \in \mathbb{R}^{H \times V}$ which facilitates calculation of the hidden vectors

$$\mathbf{h}_i \coloneqq \mathbf{g}\left(\mathbf{c} + \sum_{k<i} \mathbf{W}_{:,x_k}\right)$$

for all $i \in \{1, \ldots, d\}$. The j^{th} column of $\mathbf{W}$ is thought of as an H-dimensional word embedding for the word with index j. Thus, the hidden layers in this case simply sum a bunch of word vectors, add a bias and pass the result through a non-linearity. Finally, for a word w in our vocabulary, we calculate the probability $p_\theta(x_i = w \mid \mathbf{x}_{1:i-1})$ using a softmax with shared weights and biases

$$p_w \coloneqq p_\theta(x_i = w \mid \mathbf{x}_{1:i-1}) = \frac{\exp\left(b_w + \mathbf{V}_{w,:}^T \mathbf{h}_i\right)}{\sum_{w'} \exp\left(b_{w'} + \mathbf{V}_{w',:}^T \mathbf{h}_i\right)}$$

Unfortunately, computation of the softmax scales linearly with the size of the vocabulary. We therefore apply the hierarchical softmax function (see [Mnih and Hinton, 2007]) to compute the probabilities instead. This completes the description of the **Document NADE (DocNADE)** (see 5.4) model of [Larochelle and Lauly, 2012].

However, DocNADE assumes that documents are presented to us in the form of sequences of indexes when, in practice, documents are commonly represented as **Bag-of-Words (BoW)** counts which are inherently orderless. Given a document $\mathbf{x}$ represented as a sequence of indexes, let $\mathbf{n}(\mathbf{x})$ be the vector of word counts for $\mathbf{x}$ and let $\mathcal{V}(\mathbf{x})$ be the set of all documents $\tilde{\mathbf{x}}$ such that $\mathbf{n}(\mathbf{x}) = \mathbf{n}(\tilde{\mathbf{x}})$. As shown in [Larochelle and Lauly, 2012], we can sample uniformly at random from $\mathcal{V}(\mathbf{x})$ and we have

$$\Pr(\mathbf{x}) = \frac{1}{|\mathcal{V}(\mathbf{x})|} \sum_{\tilde{x} \in \mathcal{V}(\mathbf{x})} p_\theta(\tilde{x})$$

with $p_\theta(\tilde{x})$ modeled by DocNADE. We then train DocNADE by approximating $\Pr(\mathbf{x})$ using samples from $\mathcal{V}(\tilde{\mathbf{x}})$.

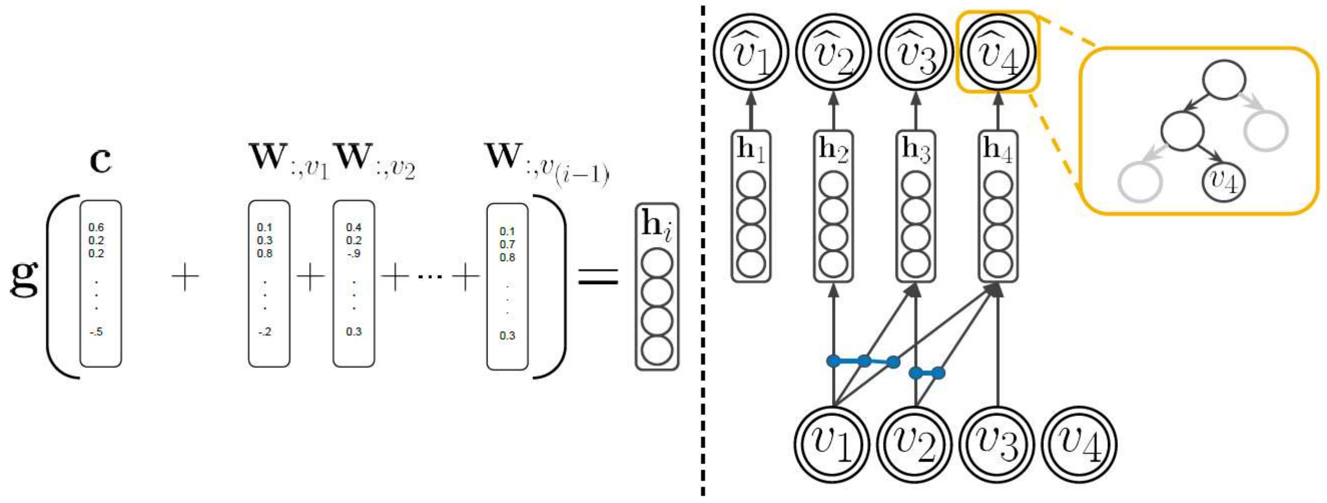


Figure 5.4: Visual representation of the DocNADE model. Taken from [Lauly et al., 2017].

Finally, [Lauly et al., 2017] presents several extensions to DocNADE. One such advance is to extract a sensible language model using the DocNADE framework while another improvement is obtained by using deep neural networks to augment the expressiveness of DocNADE. We refer the reader to [Lauly et al., 2017] for further discussion.

5.3 Neural Variational Document Model

As we have seen, the VAE provides a powerful and flexible method for extracting (continuous) latent representations from unlabeled data. In particular, if our unlabeled data consists of texts represented as BoW vectors, then passing a piece of text through a VAE yields a semantic latent variable for this text. Moreover, this provides a generative model of text. [Miao et al., 2016] explores this idea of taking a VAE architecture and applying it to BoW vectors. The resulting model is known as the **Neural Variational Document Model (NVDM)**. In more detail, NVDM uses a two-layer feed-forward neural network with ReLU activations for the inference network, a softmax regression model for the generative network and a standard Gaussian for the prior over latent variables. Note that the inference network $q_\phi(\mathbf{z} \mid \mathbf{x})$ produces a distribution over latent variables given a BoW vector $\mathbf{x}$ and the generative network $p_\theta(\mathbf{x} \mid \mathbf{z})$ independently generates words given a sample from the latent distribution, i.e.,

$$p_\theta(\mathbf{x} \mid \mathbf{z}) = \prod_i p_\theta(x_i \mid \mathbf{z})$$

As the prior is standard Gaussian and the approximate posterior is Gaussian, the KL form of the ELBO is used when computing the objective function. This is owing to the fact that the KL divergence between Gaussians is available in closed form. Evidently, the architecture employed is quite straightforward as compared to, for instance, deepDocNADE. However, NVDM still achieves competitive results on unsupervised document modeling.

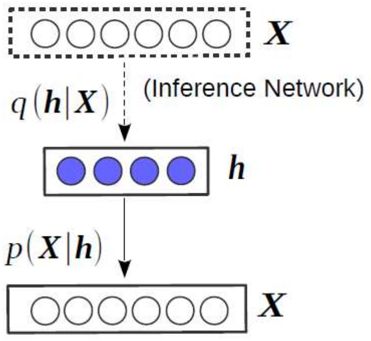

$\boldsymbol{X}$
$q(\boldsymbol{h}|\boldsymbol{X})$
(Inference Network)
$\boldsymbol{h}$
$p(\boldsymbol{X}|\boldsymbol{h})$
$\boldsymbol{X}$

Figure 5.5: The NVDM is an instance of a VAE. Taken from [Miao et al., 2016].

5.4 Flow-Augmented Neural Variational Document Model

Given that the NVDM model is simply an instantiation of the VAE and that normalizing flows provide an elegant tool for constructing more expressive probability distributions for VI, it is natural to extend the NVDM model using such flows. Indeed, as shown in [Dillon et al., 2017], we are free to increase complexity of the prior or the generative network or the inference network by introducing normalizing flows into the appropriate component of the VAE architecture. When applied to BoW vectors of texts, we call such a model a

Flow-Augmented Neural Variational Document Model (FANVDM). At time of writing, we are unaware of extant literature which considers such an approach for document modeling.

While the most obvious FANVDM is one where we simply stack a normalizing flow on top of the inference network, we expect to obtain better performance gains from augmenting the prior distribution instead. This is so since the NVDM inference network is already quite expressive relative to the simple softmax regression generative network. It is well-known that VAEs suffer mode collapse when the inference network is too expressive and it might therefore be undesirable to augment it when using the NVDM architecture. On the other hand, the prior used in NVDM is a simple standard Gaussian and this constrains the document model's capability to efficiently represent complex latent factors. Hence, we hope to be able to represent complicated, multi-modal distributions in latent space by changing the prior distribution. In slightly more detail, we use a standard Gaussian as the base distribution for our prior and transform this Gaussian through a normalizing flow to obtain the final prior distribution over latent variables. Note, however, that the prior no longer being Gaussian implies that we are forced to use the FMC form of the ELBO when computing the objective. Thus, while we can employ an arbitrary normalizing flow for the prior, IAF's efficient sampling property makes it an attractive flow for augmentation. The main novel algorithm we pursue in this work is a FANVDM with IAF as an augmenting flow for the prior. The inference and generative networks for our main FANVDM are identical to those of NVDM.

5.5 Evaluation of Document Models

We have by now encountered a diversity of document models which extract representations of documents given an unlabeled set of training data. Judging the relative merits of these different models necessitates evaluation criteria, which we now discuss. The criteria we provide here are precisely the ones used and prescribed by [Lauly et al., 2017]. We note that these criteria, or some subset thereof, is standard in much of the literature on document and topic modeling.

5.5.1 Perplexity on Test Set

Recall that a document model assigns probabilities to arbitrary documents (assuming fixed vocabulary, standard pre-processing, ect.). Thus, once a document model p_θ has been trained, we can evaluate the quality of the trained model through a metric which reflects the probability assigned to documents in a held-out test set. Given a collection $\mathbf{X}_{\text{test}} = \{\mathbf{x}_i\}_{i=1}^{M}$ of M test documents, the **(average) corpus perplexity** of the p_θ on $\mathbf{X}_{\text{test}}$ is

$$\text{perplexity}\left(\mathbf{X}_{\text{test}}\right) \coloneqq \exp\left(-\frac{1}{M}\sum_{i=1}^{M}\log p_\theta(\mathbf{x}_i)\right)$$

The key property which makes perplexity useful is that it is a monotonically decreasing function of the evidence of the test data. In particular, this means that lower perplexity is better for document models.

Obviously, computation of the perplexity requires calculating the log-evidence $\log p_\theta(\mathbf{x})$ for test documents. While some models such as DocNADE admit exact computation of this quantity, most of the document models we have enumerated above have intractable evidence. Hence, we can only approximate the perplexity on the test set in most cases. For models which employ variational inference, we use the ELBO in place of log-evidence. The ELBO itself is again intractable and, in general, we estimate the ELBO for test samples using the Monte Carlo estimate used during training. The approximation to the perplexity thus computed is still a monotonically decreasing function of the evidence of the test data. However, since the ELBO can sometimes be a poor indicator of the true evidence, the approximate perplexity may be an inaccurate indicator of the true perplexity. Other models such as LDA and RSM estimate the log-evidence using specialized sampling techniques instead. See [Wallach et al., 2009] and [Newman et al., 2010] for more detailed discussions of perplexity as a metric and alternatives.

Lastly, we remark that the actual metric used to evaluate document models is the **perplexity-per-document** defined as

$$\text{perplexity} - \text{per} - \text{doc}\left(\mathbf{X}_{\text{test}}\right) \coloneqq \exp\left(-\frac{1}{M}\sum_{i=1}^{M}\frac{1}{M_i}\log p_\theta\left(\mathbf{x}_i\right)\right)$$

where M_i is the number of words in $\mathbf{x}_i$. It is well-known that there are several problems with using corpus perplexity and we shall therefore use perplexity-per-document to rank document models.

5.5.2 Performance on Document Retrieval Task

Each of the document models we have studied extracts a (distribution over) latent or hidden representation $\mathbf{z}$ when the model is presented with an input document $\mathbf{x}$. Most of our document models are latent variable models which assume a generative process whereby latent codes "emit" observations according to a probability model. For DocNADE, the hidden layer outputs serve a similar role even though DocNADE is not specifically set up as a latent variable model. In any case, we can estimate the quality of our document model by trying to measure whether the model is able to extract good latent representations. Of course, this requires us to define how we measure the "goodness" of such representations.

Intuitively, each document model produces a representation which captures the semantic content of an input document. This implies that the model should produce similar latent representations for similar documents. Since the latent representations are vectors in a "semantic space", we can measure the closeness of two representations using cosine similarity. There are some benchmark datasets which come with labels which indicate categories or topics for each document in the dataset. Hence, we are able to group together documents by their labels and if a model computes close representations for documents with similar labels, we have some assurance of model quality.

We can translate this heuristic into quantitative terms as follows. Suppose we are given a dataset with datapoints $\mathbf{X}$ and labels $\mathbf{y}$ and that the dataset has been split into a training set $\{\mathbf{X}_{\text{train}}, \mathbf{y}_{\text{train}}\}$, a validation set[2] $\{\mathbf{X}_{\text{val}}, \mathbf{y}_{\text{val}}\}$ and test set $\{\mathbf{X}_{\text{test}}, \mathbf{y}_{\text{test}}\}$. Assume that we have trained a document model p_θ in an unsupervised fashion using $\mathbf{X}_{\text{train}}$ and $\mathbf{X}_{\text{val}}$. We create a document retrieval task in which the database to be queried is $\mathbf{X}_{\text{train}} \cup \mathbf{X}_{\text{val}}$ and each document in $\mathbf{X}_{\text{test}}$ is treated as a query to the database. Given a query, the task is to rank documents in the database by similarity to the query. In our case, similarity between documents is simply cosine similarity between latent representations. Thus, documents in the database which have closer semantic representations to the query document are ranked higher. Since we have labels for the datapoints, we can compute precision and recall for each query and hence obtain a precision-recall (PR) curve for each document model. Note, however, that a document in the dataset can have multiple labels. We therefore compute individual PR curves for each label and the average the curves to get a single, final PR curve associated to a document model. The PR curve thus computed indicates how well similar documents are clustered together in latent space.

5.5.3 Qualitative Evaluation of Representations

Given a trained document model which extracts latent or hidden representations from documents, we can attempt to see whether the representations thus learned can be organized into sensible topics. This requires some post-hoc reasoning based on our own observations

[2]A separate validation dataset is almost always required for tuning hyperparameters.

of the representations extracted by the model for individual words. However, there is no uniform method for examining or categorizing extracted topics since different models implement different mechanisms for feature extraction. For instance, LDA is an explicit topic model, i.e., LDA provides a probability distribution $p_\theta(\mathbf{x} \mid t)$ over documents $\mathbf{x}$ for a fixed number of topics t. In particular, we can get a probability distribution $p_\theta(w \mid t)$ for individual words w by treating words as documents with a single word. We can then deduce the n^{th} topics by looking at the words assigned the most probability by $p_\theta(w \mid t_n)$. For example if the highest-probability words are "encryption", "pgp" and "crypto", we may deduce that t_n corresponds to the topic of encryption.

Unfortunately, for document models which are not topic models in a manner analogous to LDA, the task of deducing topics is a bit harder. Some models like DocNADE interpret a weight matrix $\mathbf{W}$ as a word embedding matrix (where columns are word embeddings) and deduce the n^{th} topic by looking the ten or so words w which have embeddings closest to $\mathbf{W}_{n,w}$. Other models such as NVDM, which learns a continuous space of latent representations, assume that each dimension of the latent space represents a topic and looks at the words generated by the likelihood $p_\theta(\mathbf{x} \mid \mathbf{z})$ when $\mathbf{z}$ is a unit vector in latent space. We will be more specific about qualitative representations and discuss examples in the sequel on experimental results.

Chapter 6

Experimental Results and Conclusion

We now detail the experiments we carried out using our proposed FANVDM algorithm. We then offer some reflections on the results we obtained and on how we plan to build on the work carried presented here.

6.1 Datasets

Within the document modeling literature, there are two datasets which are used as standard testbeds for novel algorithms. Firstly, **20NewsGroups** (see [Lang, 1995]) consists of 20,000 messages posted to 20 Usenet newsgroups. There are 11,314 messages comprising the training set and 7,531 in the test set. Our procedure for preprocessing this dataset is identical to the procedure used by [Miao et al., 2016]. In particular, the vocabulary size is set to 2,000.

The second dataset used in the literature is the **RCV1** (see [Lewis et al., 2004]), which is a collection of over 800,000 Reuters newswire stories. There are 794,414 training documents, 10,000 test documents and [Miao et al., 2016] set the vocabulary size to 10,000 for this much larger dataset.

6.2 Experimental Setup

The inference network and generative network of our FANVDM are identical to that of NVDM. Thus, the inference network $q_\phi(\mathbf{z} \mid \mathbf{x})$ is an MLP with two hidden layers and is defined using the following set of equations (copied from Appendix A of [Miao et al., 2016]):

$$\begin{aligned}\boldsymbol{\lambda} &:= \mathrm{ReLU}\left(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1\right)\\ \boldsymbol{\pi} &:= \mathrm{ReLU}\left(\mathbf{W}_2\boldsymbol{\lambda} + \mathbf{b}_2\right)\\ \boldsymbol{\mu} &:= \mathbf{W}_3\boldsymbol{\pi} + \mathbf{b}_3\\ \log\boldsymbol{\sigma} &:= \mathbf{W}_4\boldsymbol{\pi} + \mathbf{b}_4\\ \mathbf{z} &\sim \mathcal{N}\left(\boldsymbol{\mu}, \mathrm{diag}\left(\boldsymbol{\sigma}^2\right)\right)\end{aligned}$$

Note that $\{\mathbf{W}_3, \mathbf{W}_4, \mathbf{b}_3, \mathbf{b}_4\}$ are all parameters for the final layer of the inference network and that $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are functions of $\mathbf{x}$. The generative network $p_\theta(\mathbf{x} \mid \mathbf{z})$ is a softmax regression (also copied from Appendix A of [Miao et al., 2016]):

$$\begin{aligned}\mathbf{e}_i &:= \exp\left(\mathbf{z}^T\mathbf{R}\mathbf{x}_i + \mathbf{x}_{\mathbf{x}_i}\right)\\ p_\theta\left(\mathbf{x}_i \mid \mathbf{z}\right) &:= \frac{\mathbf{e}_i}{\sum_{j=1}^{V}\mathbf{e}_j}\\ p_\theta\left(\mathbf{x} \mid \mathbf{z}\right) &:= \prod_{i=1}^{N} p_\theta\left(\mathbf{x}_i \mid \mathbf{z}\right)\end{aligned}$$

where V is the vocabulary size and $\mathbf{R} \in \mathbb{R}^{H\times V}$ is interpreted as a matrix of semantic word embeddings. Here, H is the dimensionality of the latent space. The prior was constructed using a normalizing flow with base distribution a unit Gaussiana and two IAF bijectors. Each bijector was implemented as an MLP with a single hidden layer containing 32 neurons. More precisely,

$$\mathbf{z} := f_2 \circ f_1\left(\tilde{\mathbf{z}}\right)$$

where $\tilde{\mathbf{z}} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and f_1, f_2 are the IAF bijectors. Finally, the model is trained to maximize the FMC form of the ELBO, which in our case can be approximated with samples to yield

$$\begin{aligned}\mathcal{L}(\theta, \phi; \mathbf{x}) &= \mathbb{E}_{q_\phi}\left[\log p_\theta(\mathbf{x} \mid \mathbf{z}) - \log q_\phi(\mathbf{z} \mid \mathbf{x}) + \log p_\theta(\mathbf{z})\right]\\ &= \mathbb{E}_{q_\phi}\left[\sum_{i=1}^{N}\log p_\theta(\mathbf{x}_i \mid \mathbf{z}) - \log q_\phi(\mathbf{z} \mid \mathbf{x}) + \log p_\theta(\mathbf{z})\right]\\ &\approx \frac{1}{L}\sum_{l=1}^{L}\left[\sum_{i=1}^{N}\log p_\theta\left(\mathbf{x}_i \mid \mathbf{z}^{(l)}\right) - \log q_\phi\left(\mathbf{z}^{(l)} \mid \mathbf{x}\right) + \log p_\theta\left(\mathbf{z}^{(l)}\right)\right]\end{aligned}$$

We follow [Miao et al., 2016] in only drawing a single sample for each forward pass through the model.

6.3 Results

Due to computational constraints, we only experimented with the 20NewsGroups dataset and we only used a single FANVDM model with a 50-dimensional latent space. We report the test set perplexities for our model as well several other document models explained in the foregoing chapter on document models. All perplexity scores are perplexity per document and every number except the one for FANVDM is lifted from Table 1(a) in [Miao et al., 2016]. As can be seen, we obtained some marginal improvements over the NVDM. Additionally, we plot changes in perplexities and KL divergences for the FANVDM. Note that since we used the FMC form of the ELBO, the KL divergence value is simply the empirical estimate

$$\frac{1}{L}\sum_{l=1}^{L}\left(-\log q_\lambda(\mathbf{z}^{(l)} \mid \mathbf{x}) + \log p_\theta(\mathbf{z}^{(l)})\right)$$

of the quantity

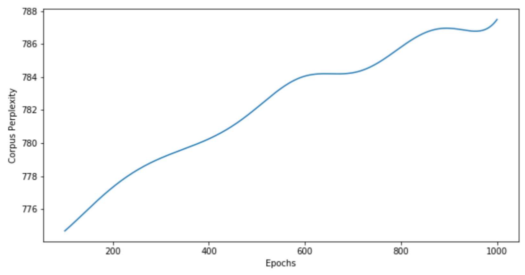
$$\mathbb{E}_{q_\lambda}\left[-\log q_\lambda(\mathbf{z} \mid \mathbf{x}) + \log p_\theta(\mathbf{z})\right]$$

We remark that we do not want the KL divergence to be too low since a very low KL divergence would imply that the latent variable is not capturing much information beyond what is already contained in the prior.

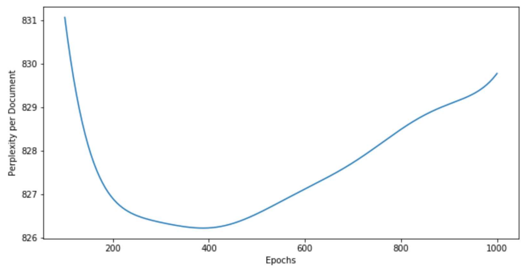
6.4 Discussion and Future Work

The results we have presented for the FANVDM should be considered as a proof-of-concept for showing that it is indeed feasible to add complexity to a VAE, particularly for document models, through normalizing flows. One of the virtues of normalizing flows is that they are a generic tool. That is, given any base distribution we can get a complex transformed distribution by simply applying a sequence of bijectors. Hence, even though we motivated normalizing flows as a tool for obtaining complex surrogate posteriors (including the flow-based free energy bound), we were able to successfully integrate a flow into the prior. Therefore, an obvious and interesting direction to pursue would be to see whether and how much performance gains we get if normalizing flows are applied to the likelihood or surrogate posterior instead.

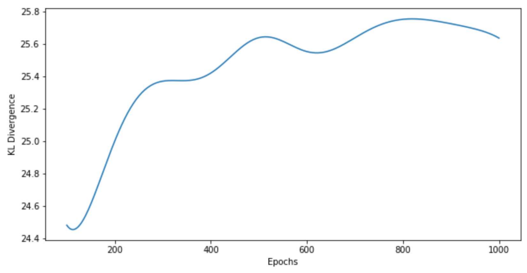
More immediately, however, we hope to carry out experiments applying FANVDM to RCV1. It appears from our table of results in figure 6.1 that more complex models



(a) Corpus perplexity



(b) Perplexity per Document



(c) KL Divergence

Figure 6.1: Change in test perplexities and KL divergence through training epochs. All curves were smoothed using splines.

Model	**Latent Dimension**	**20News**	**RCV1**
LDA	50	1091	1437
LDA	200	1058	1142
RSM	50	953	988
docNADE	50	896	742
SBN	50	909	784
fDARN	50	917	724
fDARN	200	–	598
NVDM	50	836	563
NVDM	200	852	**550**
FANVDM	50	**826**	–
FANVDM	200	–	–

Table 6.1: Perplexities per Document for various models

perform better on larger datasets. Since we managed to gain at least some improvement over NVDM on 20NewsGroups, we may be able to gain even larger improvement on RCV1. On a related note, we wish to set up more extensive evaluation of FANVDM; namely, it would be interesting to observe how FANVDM performs on the document retrieval task and also to what extent latent representations from FANVDM can help improve classification and clustering tasks.

Lastly, we can develop even more powerful versions of FANVDM by constructing more complex likelihoods so that a flow-augmented surrogate posterior does not lead to mode collapse during training. Indeed, augmenting *both* the likelihood and the surrogate posterior with normalizing flows might be a virtue of this approach. The main disadvantage of our FANVDM is, clearly, the fact that normalizing flows add time complexity to training. This is particularly true in the case of MAF and IAF which parametrize all distribution parameters with neural networks. However, for larger and more diverse datasets, this additional complexity may be a price worth paying for a model which is expressive enough to compare rich patterns in data. In conclusion, there are many directions to explore going forward and we hope that our work provides a jumping-off point for further work in this exciting area.

References

[Altosaar, 2018] Altosaar, J. (2018). Tutorial - what is a variational autoencoder? https://jaan.io/what-is-variational-autoencoder-vae-tutorial/. Accessed 2018-11-26.

[Baldi, 2012] Baldi, P. (2012). Autoencoders, unsupervised learning, and deep architectures. In *Unsupervised and Transfer Learning - Workshop held at ICML 2011, Bellevue, Washington, USA, July 2, 2011*, pages 37–50.

[Bengio, 2009] Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127.

[Blei et al., 2016] Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. (2016). Variational inference: A review for statisticians. *CoRR*, abs/1601.00670.

[Blei and Lafferty, 2007] Blei, D. M. and Lafferty, J. D. (2007). A correlated topic model of science. *The Annals of Applied Statistics*, 1(1):17–35.

[Blei et al., 2003] Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022.

[Carreira-Perpiñán and Hinton, 2005] Carreira-Perpiñán, M. Á. and Hinton, G. E. (2005). On contrastive divergence learning. In *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics, AISTATS 2005, Bridgetown, Barbados, January 6-8, 2005*.

[Dayan et al., 1995] Dayan, P., Hinton, G. E., Neal, R. M., and Zemel, R. S. (1995). The helmholtz machine. *Neural Computation*, 7(5):889–904.

[Dillon et al., 2017] Dillon, J. V., Langmore, I., Tran, D., Brevdo, E., Vasudevan, S., Moore, D., Patton, B., Alemi, A., Hoffman, M. D., and Saurous, R. A. (2017). Tensorflow distributions. *CoRR*, abs/1711.10604.

[Dinh et al., 2014] Dinh, L., Krueger, D., and Bengio, Y. (2014). NICE: non-linear independent components estimation. *CoRR*, abs/1410.8516.

[Dinh et al., 2016] Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2016). Density estimation using real NVP. *CoRR*, abs/1605.08803.

[Duchi et al., 2011] Duchi, J. C., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159.

[Dupuy and Bach, 2017] Dupuy, C. and Bach, F. (2017). Online but accurate inference for latent variable models with local gibbs sampling. *Journal of Machine Learning Research*, 18:126:1–126:45.

[Germain et al., 2015] Germain, M., Gregor, K., Murray, I., and Larochelle, H. (2015). MADE: masked autoencoder for distribution estimation. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 881–889.

[Ghahramani and Beal, 2000] Ghahramani, Z. and Beal, M. J. (2000). Propagation algorithms for variational bayesian learning. In *Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000, Denver, CO, USA*, pages 507–513.

[Gregor et al., 2014] Gregor, K., Danihelka, I., Mnih, A., Blundell, C., and Wierstra, D. (2014). Deep autoregressive networks. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 1242–1250.

[Hinton et al., 1995] Hinton, G., Dayan, P., Frey, B., and Neal, R. (1995). The "wake-sleep" algorithm for unsupervised neural networks. *Science*, 268(5214):1158–1161.

[Hinton, 2002] Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800.

[Hinton and van Camp, 1993] Hinton, G. E. and van Camp, D. (1993). Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the Sixth Annual ACM Conference on Computational Learning Theory, COLT 1993, Santa Cruz, CA, USA, July 26-28, 1993.*, pages 5–13.

[Hinton and Zemel, 1993] Hinton, G. E. and Zemel, R. S. (1993). Autoencoders, minimum description length and helmholtz free energy. In *Advances in Neural Information Processing Systems 6, [7th NIPS Conference, Denver, Colorado, USA, 1993]*, pages 3–10.

[Hoffman, 2017] Hoffman, M. D. (2017). Learning deep latent gaussian models with markov chain monte carlo. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 1510–1519.

[Hoffman et al., 2013] Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. W. (2013). Stochastic variational inference. *Journal of Machine Learning Research*, 14(1):1303–1347.

[Hu et al., 2017] Hu, J., Zhang, J., Ji, N., and Zhang, C. (2017). A modified version of helmholtz machine by using a restricted boltzmann machine to model the generative probability of the top layer. *Neurocomputing*, 267:1–17.

[Jaakkola and Jordan, 1999] Jaakkola, T. S. and Jordan, M. I. (1999). Variational probabilistic inference and the QMR-DT network. *J. Artif. Intell. Res.*, 10:291–322.

[Jang, 2018a] Jang, E. (2018a). Normalizing flows tutorial, part 1: Distributions and determinants. `https://blog.evjang.com/2018/01/nf1.html`. Accessed 2018-11-26.

[Jang, 2018b] Jang, E. (2018b). Normalizing flows tutorial, part 2: Modern normalizing flows. `https://blog.evjang.com/2018/01/nf2.html`. Accessed 2018-11-26.

[Kingma et al., 2016] Kingma, D. P., Salimans, T., Józefowicz, R., Chen, X., Sutskever, I., and Welling, M. (2016). Improving variational autoencoders with inverse autoregressive flow. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 4736–4744.

[Kingma and Welling, 2013] Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *CoRR*, abs/1312.6114.

[Knowles and Minka, 2011] Knowles, D. A. and Minka, T. (2011). Non-conjugate variational message passing for multinomial and binary regression. In *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain.*, pages 1701–1709.

[Kucukelbir et al., 2017] Kucukelbir, A., Tran, D., Ranganath, R., Gelman, A., and Blei, D. M. (2017). Automatic differentiation variational inference. *Journal of Machine Learning Research*, 18:14:1–14:45.

[Lang, 1995] Lang, K. (1995). Newsweeder: Learning to filter netnews. In *Machine Learning, Proceedings of the Twelfth International Conference on Machine Learning, Tahoe City, California, USA, July 9-12, 1995*, pages 331–339.

[Larochelle and Lauly, 2012] Larochelle, H. and Lauly, S. (2012). A neural autoregressive topic model. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 2717–2725.

[Larochelle and Murray, 2011] Larochelle, H. and Murray, I. (2011). The neural autoregressive distribution estimator. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, pages 29–37.

[Lauly et al., 2017] Lauly, S., Zheng, Y., Allauzen, A., and Larochelle, H. (2017). Document neural autoregressive distribution estimation. *Journal of Machine Learning Research*, 18:113:1–113:24.

[Lewis et al., 2004] Lewis, D. D., Yang, Y., Rose, T. G., and Li, F. (2004). RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397.

[Miao et al., 2016] Miao, Y., Yu, L., and Blunsom, P. (2016). Neural variational inference for text processing. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 1727–1736.

[Miller, 2016] Miller, A. (2016). Monte carlo gradient estimators and variational inference. `http://andymiller.github.io/2016/12/19/elbo-gradient-estimators.html`. Accessed 2018-11-26.

[Miller, 2017] Miller, A. (2017). Reducing reparametrization gradient variance. `http://andymiller.github.io/2017/05/23/rv-rges.html`. Accessed 2018-11-26.

[Miller et al., 2017] Miller, A. C., Foti, N., D'Amour, A., and Adams, R. P. (2017). Reducing reparameterization gradient variance. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 3711–3721.

[Mnih and Gregor, 2014] Mnih, A. and Gregor, K. (2014). Neural variational inference and learning in belief networks. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 1791–1799.

[Mnih and Hinton, 2007] Mnih, A. and Hinton, G. E. (2007). Three new graphical models for statistical language modelling. In *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20-24, 2007*, pages 641–648.

[Mohamed, 2015] Mohamed, S. (2015). Machine Learning Trick of the Day (4): Reparameterisation Tricks. `http://blog.shakirm.com/2015/10/machine-learning-trick-of-the-day-4-reparameterisation-tricks/`. Accessed 2018-11-26.

[Neal, 1992] Neal, R. M. (1992). Connectionist learning of belief networks. *Artificial Intelligence*, 56(1):71–113.

[Newman et al., 2010] Newman, D., Lau, J. H., Grieser, K., and Baldwin, T. (2010). Automatic evaluation of topic coherence. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 2-4, 2010, Los Angeles, California, USA*, pages 100–108.

[Paisley et al., 2012] Paisley, J. W., Blei, D. M., and Jordan, M. I. (2012). Variational bayesian inference with stochastic search. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*.

[Papamakarios et al., 2017] Papamakarios, G., Murray, I., and Pavlakou, T. (2017). Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 2335–2344.

[Pearl, 1986] Pearl, J. (1986). Fusion, propagation, and structuring in belief networks. *Artif. Intell.*, 29(3):241–288.

[Ranganath et al., 2014] Ranganath, R., Gerrish, S., and Blei, D. M. (2014). Black box variational inference. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics, AISTATS 2014, Reykjavik, Iceland, April 22-25, 2014*, pages 814–822.

[Rezende and Mohamed, 2015] Rezende, D. J. and Mohamed, S. (2015). Variational inference with normalizing flows. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 1530–1538.

[Rezende et al., 2014] Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 1278–1286.

[Robbins and Monro, 1951] Robbins, H. and Monro, S. (1951). A stochastic approximation method. *Ann. Math. Statist.*, 22(3):400–407.

[Roeder et al., 2017] Roeder, G., Wu, Y., and Duvenaud, D. K. (2017). Sticking the landing: Simple, lower-variance gradient estimators for variational inference. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 6928–6937.

[Ruder, 2016] Ruder, S. (2016). An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747.

[Salakhutdinov and Hinton, 2009a] Salakhutdinov, R. and Hinton, G. E. (2009a). Deep boltzmann machines. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, AISTATS 2009, Clearwater Beach, Florida, USA, April 16-18, 2009*, pages 448–455.

[Salakhutdinov and Hinton, 2009b] Salakhutdinov, R. and Hinton, G. E. (2009b). Replicated softmax: an undirected topic model. In *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009. Proceedings of a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada.*, pages 1607–1614.

[Salakhutdinov and Larochelle, 2010] Salakhutdinov, R. and Larochelle, H. (2010). Efficient learning of deep boltzmann machines. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, pages 693–700.

[Saul et al., 1996] Saul, L. K., Jaakkola, T. S., and Jordan, M. I. (1996). Mean field theory for sigmoid belief networks. *J. Artif. Intell. Res.*, 4:61–76.

[Turner and Sahani, 2011] Turner, R. E. and Sahani, M. (2011). Two problems with variational expectation maximisation for time-series models. In Barber, D., Cemgil, T., and

Chiappa, S., editors, *Bayesian Time series models*, chapter 5, pages 109–130. Cambridge University Press.

[Wallach et al., 2009] Wallach, H. M., Murray, I., Salakhutdinov, R., and Mimno, D. M. (2009). Evaluation methods for topic models. In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*, pages 1105–1112.

[Wang and Blei, 2013] Wang, C. and Blei, D. M. (2013). Variational inference in nonconjugate models. *J. Mach. Learn. Res.*, 14(1):1005–1031.

[Weng, 2018] Weng, L. (2018). Flow-based deep generative models. https://lilianweng.github.io/lil-log/2018/10/13/flow-based-deep-generative-models.html. Accessed 2019-03-24.

[Zhang et al., 2017] Zhang, C., Bütepage, J., Kjellström, H., and Mandt, S. (2017). Advances in variational inference. *CoRR*, abs/1711.05597.