Context-Aware Learning with Stochastic Variational Inference

Conor Durkan

Master of Science by Research
Centre for Doctoral Training in Data Science
School of Informatics
University of Edinburgh

2017

Abstract

Stochastic gradient variational methods have provided a principled way to marry the topics of deep learning, statistical inference, and probabilistic graphical models. However, much of the research in this area has thus far restricted itself to relatively simple graphical models, based on the original variational autoencoder with a single latent variable. In this thesis, we present an overview of the two pillars of these modern methods, variational inference and deep learning. We demonstrate how a confluence of these two ideas has led to the widely used stochastic gradient variational Bayes framework. We then utilize the flexibility of stochastic gradient variational Bayes to propose a novel framework featuring a dynamic graphical model with several hierarchical latent variables. Through the introduction of collections of datasets with multiple contexts, we develop the *context-aware learner*, a model suited to identifying patterns across datasets in an unsupervised fashion, which enjoys an exponential increase in representational ability for a linear increase in context count. We show that the theory readily extends to models such as this which describe a meta-learning framework, and describe a fully unsupervised model in full generality. Finally, we demonstrate effective learning in a weakly unsupervised adaptation, using a hierarchy of categorical and continuous latent variables.

Acknowledgements

Firstly, to all the supporters of and contributors to open source software around the world, and to each individual who has left their mark on the enormous body of research that comes before me: this thesis would not have been possible without you.

I would like to thank my supervisor Amos Storkey, for narrowing a broad and vague curiosity, and suggesting that I focus on the papers I would like to have written.

Thanks to all the members of the CDT in Data Science, both students and staff. A special thanks to Harri Edwards, for guidance above and beyond the call of duty, and George Papamakarios, for his invaluable feedback.

To Fionnuala, who is always there.

Finally, and most importantly, thanks to my family: my parents and grandparents, who've been on duty for almost twenty four years now, and my brother Donnchadh, who bears me with great dignity. I am eternally grateful.

This work was supported in part by the EPSRC Centre for Doctoral Training in Data Science, funded by the UK Engineering and Physical Sciences Research Council(grant EP/L016427/1), and the University of Edinburgh.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Conor Durkan*)

Table of Contents

List of Figures

Chapter 1

Introduction

> Any sufficiently advanced technology
> is indistinguishable from magic.
>
> *3^{rd} law of Arthur C. Clarke*

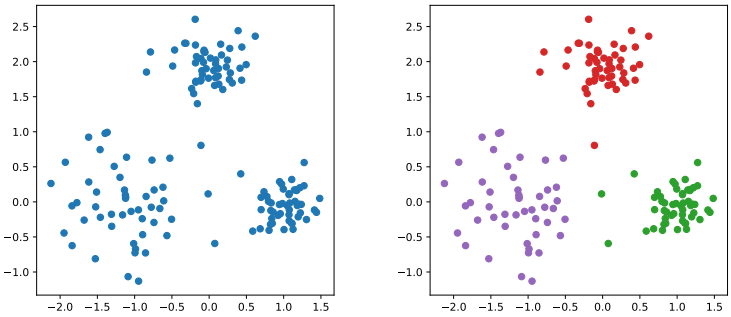
1.1 The Challenge of Unsupervised Learning

The field of machine learning has experienced unparalleled interest and growth in the recent past. Machine learning explores algorithms which can learn from and make predictions about data. Alongside two other main branches, supervised learning and reinforcement learning, the problem of unsupervised learning is central to modern machine learning research. In contrast to supervised learning, this task involves learning from data without explicit labels, in order to extract the latent structure or distribution. Since no labelling is provided, it is also difficult to formulate explicit evaluation metrics by which to judge the performance of an unsupervised learning system. For these reasons, designing models which learn effectively in an unsupervised environment is a particularly difficult challenge.

1.1.1 A canonical example

Perhaps the simplest example of unsupervised learning is that of clustering some input data into a number of groups. We assume each of our n data points belongs to one of k groups. Figure 1.1 shows some synthetic data in the plane, generated from a mixture of three Gaussian distributions. Once the number of clusters has been specified, the k-means algorithm can be used to find a suitable clustering. Even in the plane, computing the optimal solution is NP-hard [Mahajan et al., 2009]. In practice, heuristic

algorithms, such as the iterative approach of Lloyd's algorithm [Kanungo et al., 2002], are used to find an approximate solution.



(a) (b)

Figure 1.1: The canonical example of unsupervised learning using K-means clustering. (a) The data consists of samples in the plane from a mixture of Gaussians. (b) Given $K = 3$, K-means provides a possible clustering of the data.

1.2 What is the motivation for unsupervised learning?

Of the three main branches of modern machine learning research, unsupervised learning is arguably the closest to how humans learn, with a helping of supervision to accelerate the process. Additionally, it does not require expensive labelling of data, which is both time and cost prohibitive. Unsupervised models provide an extremely useful way to derive meaning from unlabeled, unstructured data. In many ways, unsupervised learning stands as the ultimate goal in the pursuit of intelligent machines.

1.3 Our contribution

In this thesis, we focus on one particularly interesting instance of unsupervised learning in modern research, namely that of generative modeling using stochastic variational inference. This topic has received considerable attention over the past few years. Beginning with a background, our contributions are then as follows:

- We examine literature which has shown that stochastic gradient variational Bayes and the variational autoencoding framework is very flexible, and extends naturally to a meta-learning setting.
- We propose theory for a novel method with a view to context-aware learning.

- We implement a weakly supervised alternative as a first approximation, and look towards a completely unsupervised version for future work.

1.4 Thesis Structure

Chapter 2 covers the basics of variational inference, and its uses in solving the general problem of statistical inference. Chapter 3 provides a brief overview of the field of deep learning as it applies to our uses. In Chapter 4, we tie the previous two topics together, and outline stochastic variational inference. We also cover the influential variational autoencoder, based on the theory introduced. In Chapter 5, we examine the neural statistician model, an extension of the variational autoencoder, which demonstrates meta-learning capabilities. In Chapter 6, we generalize this model to the context-aware learner, giving a full theoretical description in the unsupervised case, before carrying out experiments using a supervised version. Finally, Chapter 7 gives our conclusions, and sets out a promising direction for future research.

Chapter 2

Inference and Variational Bayes

2.1 The Problem of Inference

One of the main cases of unsupervised learning is that of *inference* in latent variable models. In this scenario, we have some observed data set $\mathcal{X}$, containing independent and identically distributed (i.i.d.) data points $\mathbf{x}$ which follow a marginal distribution $p(\mathbf{x})$ ($p(\mathcal{X})$ is known as the evidence). We assume this data is generated from some latent, unobserved variable $\mathbf{z}$. We denote the *prior* distribution over $\mathbf{z}$ by $p(\mathbf{z})$, and the likelihood by $p(\mathbf{x}|\mathbf{z})$. Generally, we are interested in the posterior distribution $p(\mathbf{z}|\mathbf{x})$. That is, we would like to infer which $\mathbf{z}$ variables were likely to have given rise to a particular observed $\mathbf{x}$. One major benefit of learning this distribution is that it allows us to encode data $\mathbf{x}$ in a latent code $\mathbf{z}$, forming useful representations of our data. This is the aspect we focus on in this thesis. Bayes' rule gives us a way of writing the posterior in terms of the prior and the likelihood:

$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{p(\mathbf{x})} \tag{2.1}$$

$$= \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{\int p(\mathbf{x}, \mathbf{z})\, \mathrm{d}\mathbf{z}}. \tag{2.2}$$

Once we specify a prior and a likelihood, the numerator in this expression can typically be calculated in a straightforward manner. The problem lies with the denominator. In general, this integral cannot be computed analytically, and the only option is numerical approximation. However, numerical approximations of high dimensional integrals such as these are generally intractable, and so computing this term acts as the primary stumbling block for inference in arbitrary models.

For instance, consider this example from [Blei et al., 2017], which deals with a Bayesian mixture of univariate Gaussians with unit variance. There are K components in the

mixture, each with mean $\mu^{(k)}$, where $\mu^{(k)} \sim \mathcal{N}(0, \sigma^2)$, and σ is a hyperparameter. To sample from the model, we must first choose a cluster $\mathbf{c}^{(n)}$ from a categorical (one-hot) distribution over K classes, and then sample x from that cluster. For a data set $\mathcal{X}$ of N samples, the generative process is as follows:

1. Sample $\mu^{(k)} \sim \mathcal{N}(0, \sigma^2)$ $\qquad k = 1, \dots, K$
2. Sample $\mathbf{c}^{(n)} \sim \text{Categorical}\left(\frac{1}{K}, \dots, \frac{1}{K}\right)$
3. Sample $x^{(n)} | (\mathbf{c}^{(n)}, \boldsymbol{\mu}) \sim \mathcal{N}(\mathbf{c}^{(n)\top} \boldsymbol{\mu}, 1)$ $\qquad n = 1, \dots, N.$

Here we write $\boldsymbol{\mu}$ for the vector of cluster means, and select the correct mean for sampling through an inner product with $\mathbf{c}^{(n)}$. The evidence is

$$\begin{aligned} p(\mathcal{X}) &= \int \sum_{\mathbf{c}} p(\mathcal{X}, \mathbf{c}, \boldsymbol{\mu}) \, \mathrm{d}\boldsymbol{\mu} \\ &= \sum_{\mathbf{c}} p(\mathbf{c}) \int \left[\prod_{n=1}^{N} p(x^{(n)} | \mathbf{c}^{(n)}, \boldsymbol{\mu}) \right] p(\boldsymbol{\mu}) \, \mathrm{d}\boldsymbol{\mu}. \end{aligned}$$

Since the Gaussian likelihoods are conjugate to the Gaussian prior on cluster means, each integral in this sum can be computed with some bookwork. However, there are K^N total integrals in the sum, one for each possible configuration of the cluster assignments. That is, the number of integrals is exponential in the sample size, and thus intractable in general.

Note that in this case we are still dealing with distributions which can be written down explicitly. However, the inference task becomes even more difficult in the case where these distributions are not available in closed form. For example, in the case where the likelihood is chosen to be some parameterized nonlinear function, such as a neural network, computing an exact marginal is certainly out of reach.

2.2 Navigating Intractibility

Intractibility stems from a difficulty in calculating integrals. If we can integrate an arbitrary function, we can do inference. As we've seen, this is far from the case. In this section, we provide an extremely brief overview of some methods which offer solutions to this problem. These fall under the umbrella of approximate inference techniques. Generally, we concede that computing the exact posterior is impossible, and look for an approximate solution which is close in some way.

One of the simplest approaches to approximate inference is the method of Laplace approximation [MacKay, 2003]. This technique fits a Gaussian distribution to some

unknown posterior by setting the mode of the Gaussian to be the mode of the posterior, and matching the curvature of the posterior log density at that mode.[1] The mode is found by minimizing the negative log probability of the distribution up to a constant, also known as an 'energy'. We then calculate the curvature by evaluating the Hessian of the energy at this mode.

The Laplace approximation is cheap to compute, and is accurate when the posterior is well-behaved and sharply peaked around the mode. However, it can often be the case that the mode is uninformative about the overall distribution, which may be multi-modal, or non-Gaussian away from this point.

Another approach to approximate inference is Markov Chain Monte Carlo (MCMC) [Murray, 2007], which remained dominant for many years. In MCMC, we use an ergodic Markov chain whose stationary distribution is the posterior we wish to compute, $p(\mathbf{z}|\mathbf{x})$. We collect samples from this stationary distribution by simulating the chain, and then use these samples to form an empirical estimate of the posterior.

MCMC is a powerful tool, and algorithms such as Metropolis-Hastings, and Gibbs Sampling are invaluable in modern Bayesian statistics. However, if efficiency is a concern, MCMC can struggle to produce estimates of the posterior quickly enough (such as in the case of large amounts of data), and the method becomes impractical.

There are many more approaches to inference, which are far too numerous to cover in this thesis. We merely gave a flavour of the more commonly known techniques in order to convey the fact that inference is an enormous field, with many possible takes on the same problem. From now on, we focus on one particular method, variational inference, which offers something different than what we have seen thus far.

2.3 Variational Inference

Variational inference looks for a solution to the inference problem by finding a parameterized approximate posterior, which is close to the true posterior in some way. More formally, we propose a family of parameterized distributions $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ which are tractable to work with, and seek $\boldsymbol{\phi}$ such that an appropriate measure comparing $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ and $p(\mathbf{z}|\mathbf{x})$ is minimized. Typically, we acknowledge that the true posterior distribution may not be contained in this family, but hope for a close approximation nonetheless, through the choice of a sufficiently flexible family. For thorough discussions of classic variational inference, see [Jordan et al., 1999], [Wainwright et al., 2008], or the aforementioned [Blei

[1] Laplace approximation can also refer to a more general technique of calcualting integrals.

et al., 2017].

2.3.1 Kullback-Leibler Divergence

One of the most commonly used criteria for measuring distance between distributions is the Kullback–Leibler (KL) divergence. Given two distributions $q(\mathbf{x})$ and $p(\mathbf{x})$ over some random variable $\mathbf{x}$ taking values in $\mathbb{R}^n$, the KL divergence between these distributions is defined as

$$D_{KL}(q(\mathbf{x}) \parallel p(\mathbf{x})) = \int q(\mathbf{x}) \log \frac{q(\mathbf{x})}{p(\mathbf{x})} \, \mathrm{d}\mathbf{x} \tag{2.3}$$

$$= \mathbb{E}_{\mathbf{x}\sim q(\mathbf{x})}[\log q(\mathbf{x})] - \mathbb{E}_{\mathbf{x}\sim q(\mathbf{x})}[\log p(\mathbf{x})]. \tag{2.4}$$

Two important properties of the KL divergence are as follows:

- Non-negativity i.e. $D_{KL}(q(\mathbf{x}) \parallel p(\mathbf{x})) \geq 0$.
- Asymmetry i.e. $D_{KL}(q(\mathbf{x}) \parallel p(\mathbf{x})) \neq D_{KL}(p(\mathbf{x}) \parallel q(\mathbf{x}))$ for $q \neq p$.

For a proof of these properties, see (A.1.1) and (A.1.2).

This latter property presents a dilemma: which direction should we use, $D_{KL}(q(\mathbf{x}) \parallel p(\mathbf{x}))$ or $D_{KL}(q(\mathbf{x}) \parallel p(\mathbf{x}))$? When p is the distribution we would like to approximate with distribution q, $D_{KL}(q(\mathbf{x}) \parallel p(\mathbf{x}))$ is known as the reverse KL divergence, while $D_{KL}(p(\mathbf{x}) \parallel q(\mathbf{x}))$ is known as the forward KL divergence. Recalling the definition of the KL divergence, the forward version would require us to take expectations with respect to the distribution we wish to approximate, $p(\mathbf{x})$. However, computing $p(\mathbf{x})$ is exactly the issue we originally faced. Thus, we minimize the reverse KL divergence, which provides us with the evidence lower bound, as we will see in the next section. For a method which can be interpreted as minimizing the forward divergence, see expectation propagation (EP) [Minka, 2001].

With the reverse KL divergence in hand, we can examine some potential shortcomings of the approach. In particular, note that whenever the true distribution $p(\mathbf{x})$ is zero, our approximation $q(\mathbf{x})$ must be zero also, so that the KL divergence does not blow up. For this reason, the approximation in the reverse KL is termed 'zero-avoiding', and will generally be supported on a set of smaller measure than the true distribution. For more details of forward and reverse KL divergence, an excellent discussion is available in [Murphy, 2012].

We now have a target distribution for variational inference. We wish to find $q_{\boldsymbol{\phi}^\star}(\mathbf{z}|\mathbf{x})$,

where

$$\boldsymbol{\phi}^\star = \arg\min_{\boldsymbol{\phi}} D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z}|\mathbf{x})). \tag{2.5}$$

Even with this, it turns out we have the same problem as before. By definition,

$$D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z}|\mathbf{x})) = \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})}[\log q(\mathbf{z}|\mathbf{x})] - \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{z}|\mathbf{x})]. \tag{2.6}$$

Expanding the conditional in the second term,

$$\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{z}|\mathbf{x})] = \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{z},\mathbf{x})] - \log p(\mathbf{x}). \tag{2.7}$$

In other words, we need the log evidence in order to calculate the KL divergence, but the intractibilty of the evidence is the reason we appealed to variational inference in the first place. However, there is a solution which allows us to navigate around this issue.

2.3.2 The Evidence (or Variational) Lower Bound

Rather than minimizing the KL divergence between the approximate and true posterior distributions directly, we typically minimize an alternative quantity, equivalent to the KL up to a constant (the log evidence). This is known as the evidence (or variational) lower bound. Deriving this is a matter of invoking Bayes' rule on the true posterior, and rearranging terms.

$$\begin{aligned}
D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z}|\mathbf{x})) &= \int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \log \frac{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}{p(\mathbf{z}|\mathbf{x})} \,\mathrm{d}\mathbf{z} \\
&= \int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \log \frac{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) p(\mathbf{x})}{p(\mathbf{x}|\mathbf{z}) p(\mathbf{z})} \,\mathrm{d}\mathbf{z} \\
&= \int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \log \frac{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}{p(\mathbf{z})} \,\mathrm{d}\mathbf{z} - \int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \log p(\mathbf{x}|\mathbf{z}) \,\mathrm{d}\mathbf{z} \\
&\qquad + \int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \log p(\mathbf{x}) \,\mathrm{d}\mathbf{z} \\
&= D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})) - \mathbb{E}_{\mathbf{z}\sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z})] + \log p(\mathbf{x}).
\end{aligned} \tag{2.8}$$

The key observation here is that $\log p(\mathbf{x})$ does not depend on the variational parameters $\boldsymbol{\phi}$. Thus, minimizing $D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z}|\mathbf{x}))$ is equivalent to minimizing

$$-\mathcal{L}(\boldsymbol{\phi}) = D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})) - \mathbb{E}_{\mathbf{z}\sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z})]. \tag{2.9}$$

We write the criterion $-\mathcal{L}(\boldsymbol{\phi})$ with a negative sign since $\mathcal{L}(\boldsymbol{\phi})$ is usually understood to be a lower bound on the log evidence. To see this, just recall the non-negative property

of the KL divergence.

$$
\begin{aligned}
& D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z}|\mathbf{x})) \geq 0 \\
\iff & D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})) - \mathbb{E}_{\mathbf{z}\sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z})] + \log p(\mathbf{x}) \geq 0 \\
\iff & \log p(\mathbf{x}) \geq \mathcal{L}(\boldsymbol{\phi}) = \mathbb{E}_{\mathbf{z}\sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z})] - D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})). \qquad (2.10)
\end{aligned}
$$

Finally we have a feasible direction of attack for variational inference. We would like to minimize $-\mathcal{L}(\boldsymbol{\phi})$ in order to find a good approximation for our intractable posterior $p(\mathbf{z}|\mathbf{x})$. In the next chapter, we give a brief overview of the field of deep learning, which will equip us with the tools necessary for performing this optimization procedure across large data sets.

Chapter 3

Deep Learning

The recent surge in machine learning research is due in no small part to the advent of the neural network. Modern computational power, the 'big data' phenomenon, and the development of a wide variety of methods which help to stabilize learning in these networks, has caused an explosion in the study of deep[1] models. In this chapter, we present the main concepts of deep learning relevant to our work in this thesis. For a comprehensive overview of the field, see [Goodfellow et al., 2016].

3.1 Neural Networks

Neural networks (NNs) are parameterized function approximators. As we will see, they can serve as excellent candidates for the parameterized distributions needed in variational inference. Typically they consist of operations which alternate between affine or convolutional transformations, and piecewise application of some nonlinear function. These operations are more commonly referred to as layers.

NNs are universal function approximators. That is, given some domain $\Omega \subset \mathbb{R}^n$, and a well-behaved function $\boldsymbol{f} : \mathbb{R}^n \mapsto \mathbb{R}^m$, a network with sufficient representational power can approximate this function to arbitrary precision on this domain. This result is known as the universal approximation theorem. For an intuitive visual explanation, see [Nielsen, 2015]. For a more formal treatment, making explicit the conditions necessary on Ω and $\boldsymbol{f}$, see [Cybenko, 1989] or [Hornik, 1991].

In practice, modern networks cannot be shown to satisfy the conditions of the universal approximation theorem, since it is a limit and existence argument. Generally we rely

[1] While there is no explicit definition of the word 'deep' in this context, it is usually used to refer to models with multiple hidden layers.

on heuristics and empirical validation to justify their use. In addition, the term 'non-parametric limit' has recently come into use, referring to a model's capabilities when the limitations of a finite parameterization are not present. Despite these difficulties, neural networks have proven themselves to be powerful tools for a variety of machine learning tasks.

3.2 Backpropagation

Backpropagation, or reverse-mode automatic differentiation, is the workhorse of deep learning. Given some input data $\mathcal{X}$ and corresponding labels $\mathcal{Y}$, we create a criterion $\mathcal{L}$, which is some function of the network input and output (generally the labels and the network output in the case of supervised learning). The criterion allows us to compare the produced output with the intended result, and adjust the parameters accordingly, with a view to finding the optimal parameters $\boldsymbol{\theta}^\star$ for that particular criterion. More formally, if $\boldsymbol{\theta}$ are the parameters of the network, we calculate $\nabla_{\boldsymbol{\theta}}\mathcal{L}$, the gradient of the criterion with respect to the parameters, and move in the direction (or opposite direction) of this gradient in parameter space. Recall that the gradient points in the direction of greatest increase of the criterion $\mathcal{L}$. If $\mathcal{L}$ is differentiable, the existence of a step size small enough so that $\mathcal{L}$ either increases or decreases is guaranteed (this is the definition of the derivative). We write the updated parameters $\boldsymbol{\theta}'$ as

$$\boldsymbol{\theta}' = \boldsymbol{\theta} + \alpha\nabla_{\boldsymbol{\theta}}\mathcal{L}. \tag{3.1}$$

The magnitude of the step size α determines how much we want to move in the direction of the gradient, and its sign indicates whether we would like to minimize or maximize the criterion $\mathcal{L}$.

At its heart, backpropagation is really just the chain rule of multivariable calculus in disguise, utilised over large computation graphs with many parameters. The clever application of the technique in neural networks lies in the computation of the gradients in reverse, hence the 'back' in backpropagation. Consider the computational graph shown in Figure 3.1, where the criterion $\mathcal{L}$ depends on the variables B and C directly, and A indirectly. We would like to calculate $\frac{\partial\mathcal{L}}{\partial A}$, $\frac{\partial\mathcal{L}}{\partial B}$, and $\frac{\partial\mathcal{L}}{\partial C}$. The latter two terms are computable directly, if we sweep from right to left in the computation graph. For the gradient with respect to A, we have

$$\frac{\partial\mathcal{L}}{\partial A} = \frac{\partial\mathcal{L}}{\partial B}\frac{\partial B}{\partial A} + \frac{\partial\mathcal{L}}{\partial C}\frac{\partial C}{\partial A}. \tag{3.2}$$

Note that although this derivative has four terms, we have already computed $\frac{\partial\mathcal{L}}{\partial B}$ and $\frac{\partial\mathcal{L}}{\partial C}$. This is the key to the backpropagation algorithm, and the cheap calculation of

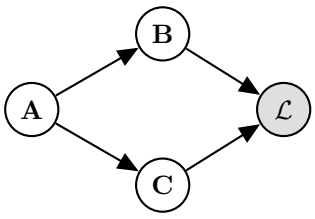


Figure 3.1: Simple computational graph.

gradients. In fact, computing gradients like this has the same order of computational complexity as one forward pass (from left to right) in the graph. This is not the case when calculating gradients using finite differences, where we need to evaluate the function multiple times for each gradient approximation. In addition, reverse-mode is cheaper than forward-mode when we map from a higher to a lower dimensional space, as is often the case with deep learning. For a thorough discussion of forward and reverse-mode differentiation, see [Giles, 2008].

The overall picture then looks as follows: given a network parameterized by $\boldsymbol{\theta}$, and a data set of samples $\mathcal{X}$, we pass the data through the network, calculate the criterion $\mathcal{L}$, compute $\nabla_{\boldsymbol{\theta}}\mathcal{L}$, the gradient of this criterion with respect to the parameters, and then update the parameters accordingly. This process is then repeated until the parameters converge suitably, measured by some metric or heuristic. To make clear the iterative process, we index the parameters by t, which is incremented each time an update is performed.

3.2.1 Stochastic Optimization

While iteratively stepping in the direction of the gradient over an entire data set is a sound approach in theory, in practice it proves extremely difficult due to very large data sets and limited computational power. Instead, we perform stochastic (or mini-batch) gradient descent, by passing a single example (or mini-batch) through the network at each step, and updating the parameters after each iteration. Note that the convention is always to minimize the criterion $\mathcal{L}$, and since any maximization problem can be phrased as minimizing a negative criterion, this is easily achieved. For this reason, we take the step size $\alpha > 0$, and move in the direction of the negative gradient.

The content presented in this section restricts itself to the methods used in training stochastic variational models in this thesis. For a more complete overview of gradient optimization algorithms, we refer the reader to [Ruder, 2016].

3.2.1.1 Stochastic Gradient Descent

Stochastic gradient descent proceeds exactly as outlined in the section on backpropagation. However, instead of using the entire data set to calculate gradients, we use one randomly chosen sample at a time. That is, if $\mathbf{x}$ is a single data point passed to the network, and $\hat{\mathbf{x}}$ is the corresponding output, then we update

$$\boldsymbol{\theta}_{(t+1)} = \boldsymbol{\theta}_{(t)} - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{x}, \hat{\mathbf{x}}). \tag{3.3}$$

In practice, we don't process exactly one sample at a time. Instead, we pass a mini-batch of samples through the network at each iteration. There a number of reasons for this. Using a single sample exhibits high variance in the stochastic gradient, and can destabilize training, though this can be mitigated by reducing the step size. More importantly, the linear algebra operations involved in passing a single sample through the network are trivially parallelizable across a mini-batch, meaning training is quicker.

The choice of the step size is obviously an important one, as is the decision about whether to keep it fixed, or vary it by some schedule. Additionally, we might wonder if stepping in exactly the direction of the negative gradient is the optimal thing to do each time. Indeed, more advanced and nuanced methods are often used, one of which is outlined in the next section.

3.2.1.2 Adam

Adaptive moment estimation (Adam) [Kingma and Ba, 2014] is a stochastic optimization method with an adaptive learning rate for each parameter, and benefits from the effects of momentum. Its 'off-the-shelf' capabilities have made it an extremely popular choice when looking for a gradient descent algorithm that does not require manual tweaking of learning rates and schedules.

Per-parameter update rules are useful when data is sparse, or more generally, when parameter updates happen with an inconsistent frequency. Parameters which are modified infrequently can receive larger updates, and vice versa. On the other hand, momentum (i.e. keeping a running estimate of gradients) eases the pain of noisy gradients, and allows for effective online learning, or learning with a non-stationary objective. The typical analogy used for momentum-based learning is that of a ball released from the edge of a bowl. The ball picks up speed and reaches the centre of the bowl faster than taking repeated steps in the direction of the gradient and ignoring past behaviour.

Like the name suggests, Adam keeps track of exponentially decaying moment estimates of the gradient, namely the mean $\boldsymbol{m}$ and variance $\boldsymbol{v}$. At each step these moments are

updated as

$$\boldsymbol{m}_{(t+1)} = \beta_1 \boldsymbol{m}_{(t)} + (1 - \beta_1)\nabla_{\boldsymbol{\theta}}\mathcal{L} \tag{3.4}$$

$$\boldsymbol{v}_{(t+1)} = \beta_2 \boldsymbol{v}_{(t)} + (1 - \beta_2)(\nabla_{\boldsymbol{\theta}}\mathcal{L})^2 \tag{3.5}$$

Since these quantities are initialized as vectors of zeros, they are biased towards zero, and so the bias-corrected moments are used instead.

$$\hat{\boldsymbol{m}}_{(t+1)} = \frac{\boldsymbol{m}_{(t+1)}}{1 - \beta_1^t} \tag{3.6}$$

$$\hat{\boldsymbol{v}}_{(t+1)} = \frac{\boldsymbol{v}_{(t+1)}}{1 - \beta_2^t} \tag{3.7}$$

These are then used in the update rule for Adam itself, namely

$$\boldsymbol{\theta}_{(t+1)} = \boldsymbol{\theta}_{(t)} - \eta \frac{\hat{\boldsymbol{m}}_{(t+1)}}{\sqrt{\hat{\boldsymbol{v}}_{(t+1)}} + \varepsilon} \tag{3.8}$$

The authors suggest values of $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$.

3.2.1.3 Behind the Scenes

The truth behind stochastic optimization for large parameterized models is a great deal more complex than the content of this section might suggest. Typically we are optimizing non-convex criteria $\mathcal{L}$, with many minima, and there is no guarantee whatsoever that the minimum we end up in is global, let alone desirable. There are many heuristics for coping with this, including early stopping, annealing the learning rate by a particular schedule, or ensembling a number of models. This is often where the field of deep learning becomes more of an art than a science.

3.3 Model Architectures

In this section we examine some of the common architectures used in modern deep learning, with a focus on those models which will be useful for work presented in this thesis.

3.3.1 A Note on General Structure

In their most common form, neural networks alternate between linear operations, and the piecewise application of a particular nonlinear function, called the nonlinearity or activation function. This nonlinearity is typically used across the entire network, except

for special cases where we would like to use a function which constrains a particular output to a certain range. Picking the correct nonlinearity can be of central importance in deep learning, and is an ongoing direction of research. For an overview of the choices used in this thesis, namely the sigmoid and rectified linear unit (ReLU), see Appendix B.1.

3.3.2 Fully Connected Networks

Fully connected networks (FCNNs) consist of alternating linear transformations and nonlinearities. The linear transformation can optionally include a bias term. That is, for an input $\mathbf{x}$, the operation prior to the nonlinearity computes the quantity $\mathbf{W}\mathbf{x} + \mathbf{b}$. $\mathbf{W}$ and $\mathbf{b}$ are called the weight and bias for that particular layer, and a FCNN may stack many of these layers. These are the parameters which are learned during training of the network.

FCNNs are perhaps the simplest form of modern networks. Many architectures use them as building blocks from which to form more complex models. As we will see, FCNNs serve very capably as general purpose function approximators when plugged into models in the right way.

3.3.3 Convolutional Networks

While FCNNs are useful tools, it is often preferable to use models which have been developed with some domain specificity in mind. For example, convolutional[2] neural networks (CNNs) are particularly suited for use with image data. They have come into widespread use in the computer vision community over the past number of years, repeatedly surpassing state of the art on a variety of tasks [Krizhevsky et al., 2012] [Russakovsky et al., 2015]. Where FCNNs alternate between standard linear transformations and nonlinearities, CNNs opt for the use of a convolution operation between these nonlinearities.

The key to understanding CNNs lies in the concepts of local receptive fields and shared weights. RGB images are represented by structures of dimension `(channels, height, width)`. While this spatial structure is essentially ignored by FCNNs, these concepts, along with an inherent translation invariance, allow us to use the structure to our advantage. Beginning with the idea of local receptive fields, consider Figure 3.2. On

[2]Contrary to its name, a convolutional neural network (CNN) does not use convolutions, but instead the closely related operation of cross-correlation. In keeping with the convention, we will use the term convolution throughout.

the left, we have a 5×5 input image, with a single channel. The weights of a CNN, more commonly called kernels or filters, are two dimensional windows which slide over the input, carrying out the convolution operation. In this case, we have a 3×3 square filter. At each step, the dot product with the underlying image is computed, and this becomes the feature learned by the filter at this location. The patch of the input image for which each feature is responsible is called that feature's local receptive field.

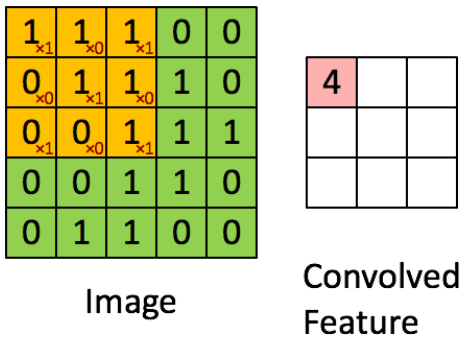


Figure 3.2: Demonstration of local receptive field.[3]The output value (4), is given by taking a dot product of the sliding window, or filter, in yellow, and the underlying image, in green.

In this way, the sharing of weights happens naturally. Because of the sliding window motion inherent with convolution, the filter is learned so that it adapts well to identifying features across the entire input image. That is, the learned filter is a global feature extractor for the entire input image. Note that like FCNNs, a CNN filter can also have a bias, which is a scalar, since the output of each dot product is a scalar. In this way, the bias for a particular filter is also shared across the entire image.

Figure 3.3 demonstrates the typical structure of a CNN, which usually takes a multi-channel input image, and uses many more channels in its intermediate representation of the data. So far we have used a single channel input image, with a single filter, but the idea generalizes very easily. For each input channel, we need a distinct filter. The output, more commonly known as a feature map, is then formed by convolving each filter with each input channel separately, and summing the results. For more than one feature map, we repeat this process with different filters. Thus, for an input with n_{in} channels, a filter of shape $h \times w$, and a desired output with n_{out} channels, we need a total of $n_{\text{out}} \times (n_{\text{in}} \times (h \times w))$ parameters. Typically, this results in networks which are far more parameter efficient than FCNNs, and achieve better performance in tasks to

[3]Taken from `http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution`

which they are suited. A final fully connected layer is usually inserted in CNNs as a 'classifier' which uses the learned convolutional features, and has the added bonus that the output is of the correct dimensionality.

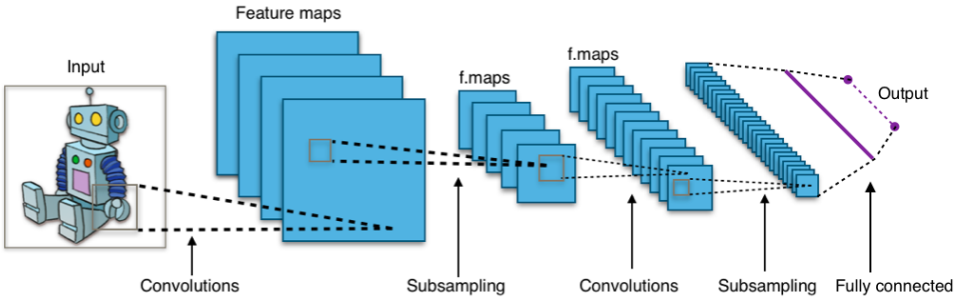


Figure 3.3: Typical example of a multi-layer CNN.[4] Note the alternating convolutions and subsampling, with a final fully connected layer. Nonlinearities not shown explicitly.

Another important detail which should be mentioned in this overview is downsampling within a CNN, which is the reduction in the height and width dimensions of an image-like representation. First, note that convolution as presented above does not preserve the these dimensions. This is known as valid convolution. Same convolution pads the input with zeros such that the dimensions of height and width are preserved. This note aside, we also want a way to downsample intentionally, commonly by a factor of two or four at a given time. In the past, the operation of pooling was common. Rather than using an entire feature map as input, the average or max over sub-blocks of this map would be used such that the dimensionality was reduced appropriately. More recently, striding has become more prevalent. This is where the sliding filter moves more than one step at a time, effectively allowing the network to learn down its own form of downsampling.

3.3.4 More Complex Architectures

3.3.4.1 Autoencoders

Autoencoders [Hinton and Salakhutdinov, 2006] are a nonlinear dimensionality reduction technique. The idea is simple: given an input $\mathbf{x}$, we would like to train a network that returns our input i.e. $\mathbf{f}(\mathbf{x}) = \mathbf{x}$. This is trivial without constraints; we simply use the identity. The problem becomes interesting when the intermediate representations used by our network must be of dimensionality strictly less than that of our input, which we enforce by introducing a lower-dimensional bottleneck into our network. In this case,

[4] Taken from `https://upload.wikimedia.org/wikipedia/commons/6/63/Typical_cnn.png`. By Aphex34 (Own work) [CC BY-SA 4.0 (`http://creativecommons.org/licenses/by-sa/4.0`)], via Wikimedia Commons

we can decompose the autoencoder naturally into two parts:

- an encoder E which takes our n-dimensional input $\mathbf{x}$ and produces a k-dimensional representation or code $\mathbf{z}$, where $k < n$,
- a decoder D which takes the latent representation $\mathbf{z}$, and returns the original input $\mathbf{x}$.

Note that the concept of the autoencoder is decoupled from any particular architecture. We are free to choose whichever structure we like for the encoder and decoder, be it fully connected, convolutional, or a temporal-based architecture like a recurrent neural network. Note also the resemblance of the model to the probabilistic framework of latent variables presented in the previous chapter. As we will see, a variational autoencoder allows us to tie the two settings together nicely.

3.3.4.2 Residual Networks

The idea of residual networks came to the fore in the context of computer vision [He et al., 2016]. In training very deep models, a central issue is the vanishing of gradients in the very long chains of backpropagation. To solve this, the concept of skip connections were introduced. In particular, the notion of a residual block was proposed. These blocks consist of a number of convolutional layers, and process an input $\mathbf{x}$ as any standard feed-forward model. The key difference is that prior to the final nonlinearity of the block, the input $\mathbf{x}$ is added to the quantity calculated by the block up to that point. The idea is illustrated in Figure 3.4.

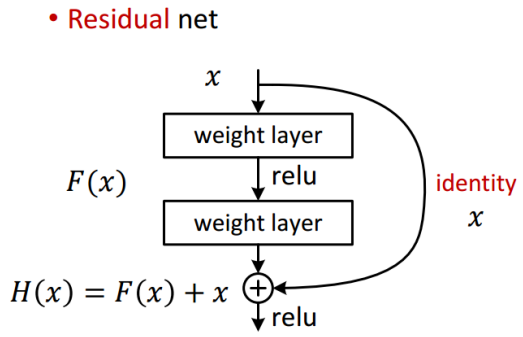


Figure 3.4: Simple figure outlining a residual block. Rather than applying the final nonlinearity to $\mathcal{F}(\mathbf{x})$ which has been computed by the block, we apply the nonlinearity to the sum $\mathcal{F}(\mathbf{x}) + \mathbf{x}$. Figure taken from [He et al., 2016].

The reason for the suggestion of a network architecture such as this was to alleviate the issue of vanishing gradients. When many of these blocks are stacked, the skip

connections provide gradients with a path to travel directly back through the graph. It also allows the higher layers of a network to make use of lower level features typically learned earlier in a deep model. As mentioned, residual networks were first introduced for the purpose of image classification models, but the idea of skip connections is applicable in any deep model. Implementing them is trivial, they are computationally cheap, and they almost always provide improved performance over standard feed-forward networks.

3.4 Training Tools

Neural networks are inherently unwieldy creatures; with so many parameters and so much representational power, getting them to behave and train as we would like is a difficult task. In this section we cover some commonly used techniques to improve stability in training.

3.4.1 Initialization

One natural question to ask when dealing with large parameterized models is which values these parameters should initially take. This turns out to be quite an important issue, and has been explored in both [Glorot and Bengio, 2010] and [He et al., 2015], which introduce schemes known as Xavier and He initialization, respectively. Good initialization can mean the difference between a network converging in a reasonable amount of time, or never getting anywhere at all. Without delving into too much detail, the main idea behind these methods is to keep the variance of pre-nonlinearity quantities constant throughout the entire network, in either the forward pass, the backward pass, or a compromise between both. This helps to prevent vanishing or exploding gradients, and can make a considerable difference in helping the model to learn.

3.4.2 Normalization

Once initialized, we would like our parameters to remain well-behaved for the duration of training. Large variation per batch in the input to a particular layer means that that layer must constantly readjust its parameters in order to accomodate a new distribution. Batch normalization [Ioffe and Szegedy, 2015] was introduced to combat this issue, and has become widespread in the training of deep models. As the name suggest, batch normalization first renormalizes the inputs to each layer so that they have zero mean and unit variance. Moreover, it provides the option of learning a mean and variance per

dimension, which should allow the model to adapt to a particular distribution if it is advantageous.

Inspired by batch normalization, more recent methods have been proposed which look to overcome dependence on batch size, though they have not attained the same level of use thus far. Layer normalization [Ba et al., 2016] can be seen as the transpose of batch normalization, where the mean and variance are computed per individual item activations in the batch. This technique was proposed for use in recurrent models, where it is less obvious how batch normalization might be used. Weight normalization [Salimans and Kingma, 2016] decouples parameters' magnitudes from their direction, avoids batch dependence, and acclerates the convergence of stochastic gradient descent. Finally, the recent work on self-normalizing networks [Klambauer et al., 2017] takes a fresh approach, using a specially crafted activation function which yields automatic convergence of network activations to zero mean and unit variance.

Chapter 4

Stochastic Variational Inference

In this chapter, we combine the topics outlined so far and discuss stochastic variational inference [Hoffman et al., 2013], a scalable method for performing inference on large data sets. In particular, we present the ideas of amortizing inference networks and the reparametrization trick, introduced by [Kingma and Welling, 2013] and [Rezende et al., 2014]. We then examine this framework as applied to the autoencoder, using neural networks for the inference and generative models, resulting in the variational autoencoder (VAE) [Kingma and Welling, 2013].

4.1 Phrasing Inference as Stochastic Optimization

Let us restate our setting up until this point. We have available a data set $\mathcal{X}$, consisting of N observations $\mathbf{x}$. We have assumed each $\mathbf{x}$ is generated from a corresponding latent $\mathbf{z}$, sampled from some prior $p(\mathbf{z})$. However, as with the approximate posterior from variational inference, let us now also take this generative process to be parameterized by some $\boldsymbol{\theta}^\star$. That is, the conditional distribution on observations given latents is $p_{\boldsymbol{\theta}^\star}(\mathbf{x}, \mathbf{z})$. We assume this likelihood comes from a family of parameterized distributions $p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$, and that these distributions are differentiable with respect to both $\boldsymbol{\theta}$ and $\mathbf{z}$. Similarly, we also ask for the family of approximate posterior distributions $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ to be differentiable with respect to $\boldsymbol{\phi}$ and $\mathbf{x}$. Finally, assume N is very large, so that we can't work with the entire data set at one time.

Consider the task of learning the inference and generative parameters, $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$, respectively. Since our observations are i.i.d., we can write the log evidence as

$$\log p_{\boldsymbol{\theta}}(\mathcal{X}) = \sum_{\mathbf{x} \in \mathcal{X}} \log p_{\boldsymbol{\theta}}(\mathbf{x}). \tag{4.1}$$

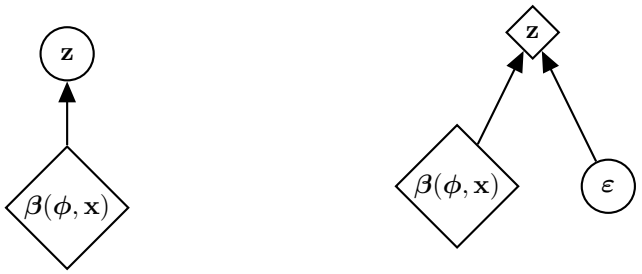
Recall that in Chapter 2 we derived a variational lower bound for the single datum log evidence in a similar scenario. This was introduced as an optimization criterion for determining the optimal parameters $\boldsymbol{\phi}$ of the approximate posterior. As we will show explicitly in the coming section describing the VAE, we can bound the log evidence when using a parameterized p in a similar way. We can write

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) \geq \mathcal{L}_{\mathbf{x}}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \mathbb{E}_{\mathbf{z}\sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\right] - D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})), \tag{4.2}$$

and propose this as a criterion for stochastic optimization of both $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$. However, there is an immediate issue to contend with when taking gradients of this lower bound with respect to $\boldsymbol{\phi}$. Assuming we choose a prior and approximate posterior over $\mathbf{z}$ which admit an analytic solution to the KL divergence term, the issue we must contend with is taking the gradient of the expectation over the log likelihood. Fortunately, there is a solution, commonly referred to as the reparameterization trick [Kingma and Welling, 2013] [Rezende et al., 2014].

4.1.1 The Reparameterization Trick

The reason backpropagation runs into a problem is that we need to be able to take gradients through the sampling procedure which generates $\mathbf{z}$ from $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$, in order to update the parameters $\boldsymbol{\phi}$. However, this is not a differentiable process. The key to overcoming this hurdle is reparameterizing the distribution $q_{\boldsymbol{\phi}}$, yielding a form which allows us to do what we want. In particular, many distributions can be written as some parameterized function of noise, $g = g_{\boldsymbol{\beta}(\boldsymbol{\phi},\mathbf{x})}(\boldsymbol{\varepsilon})$, with $\boldsymbol{\varepsilon} \sim p(\boldsymbol{\varepsilon})$. Here the parameters $\boldsymbol{\beta}(\boldsymbol{\phi}, \mathbf{x})$ are deterministic, and the stochastic element comes from the noise term $\boldsymbol{\varepsilon}$.



(a) $\mathbf{z}$ is stochastic, sampled from a distribution parameterized by $\boldsymbol{\beta}(\boldsymbol{\phi}, \mathbf{x})$.

(b) Reparameterization moves the stochastic element to a noise term $\boldsymbol{\varepsilon}$, making $\mathbf{z}$ deterministic.

Figure 4.1: Illustration of the reparameterization trick. Stochastic elements indicated by round shape. Reparameterization removes stochasticity from $\mathbf{z}$ so that we can take gradients.

With $\mathbf{z} = g_{\boldsymbol{\beta}(\boldsymbol{\phi},\mathbf{x})}(\boldsymbol{\varepsilon})$, we can rewrite the expectation over the log likelihood and construct a Monte Carlo estimator as follows:

$$\begin{aligned}\mathbb{E}_{\mathbf{z}\sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\right] &= \mathbb{E}_{\boldsymbol{\varepsilon}\sim p(\boldsymbol{\varepsilon})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}|g_{\boldsymbol{\beta}(\boldsymbol{\phi},\mathbf{x})}(\boldsymbol{\varepsilon}))\right] \\ &\approx \frac{1}{L}\sum_{l=1}^{L}\log p_{\boldsymbol{\theta}}(\mathbf{x}|g_{\boldsymbol{\beta}(\boldsymbol{\phi},\mathbf{x})}(\boldsymbol{\varepsilon}^{(l)})), \text{ where } \boldsymbol{\varepsilon}^{(l)} \sim p(\boldsymbol{\varepsilon}). \end{aligned} \tag{4.3}$$

Moreover, this estimator is differentiable with respect to the inference parameters $\boldsymbol{\phi}$. Better yet, the number of samples L per datapoint $\mathbf{x}$ can often be set to 1, by compensating with the use of larger mini-batches for training. Overall then, we have an estimator of the variational lower bound which is differentiable with respect to both the inference parameters $\boldsymbol{\phi}$, and generative parameters $\boldsymbol{\theta}$, and is also unbiased.

The canonical example of the reparameterization trick, and by far the most commonly used reparameterization in the VAE framework, is the diagonal normal distribution. Given $\boldsymbol{\beta}(\boldsymbol{\phi},\mathbf{x}) = (\boldsymbol{\mu},\boldsymbol{\sigma}^2)$, the mean and (diagonal) variance of a multivariate normal, we can sample differentiably from $\mathcal{N}(\boldsymbol{\mu},\mathrm{diag}(\boldsymbol{\sigma}^2))$ by taking

$$g_{\boldsymbol{\beta}(\boldsymbol{\phi},\mathbf{x})}(\boldsymbol{\varepsilon}) = \boldsymbol{\mu} + \boldsymbol{\sigma}\odot\boldsymbol{\varepsilon}, \tag{4.4}$$

where $\boldsymbol{\varepsilon}\sim\mathcal{N}(0,\mathrm{I})$, and $\odot$ denotes the Hadamard product or elementwise multiplication. This example extends naturally to all location-scale families.

Using the reparameterization trick to perform stochastic gradient descent on the variational lower bound was termed stochastic gradient variational Bayes (SGVB) by Kingma & Welling, and we use this to describe the procedure here as well. Most notably, SGVB turns a statistical inference problem into a large-scale optimization problem. This allows us to bring the powerful machinery of modern deep learning to bear on the problem of posterior inference and generative modeling.

4.2 Model (The Variational Autoencoder)

The variational autoencoder (VAE) is the model used by Kingma & Welling to demonstrate the above theory. For the rest of the chapter, we make the theory explicit in this context, detail the implementation of such a model, and detail some experiments which illustrate the model's capabilities and behaviour. [Doersch, 2016] provides a good reference for the VAE.

A VAE is a latent variable model, such that for each observation $\mathbf{x}\in\mathcal{X}$, we have the following generative process:

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \int p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})p(\mathbf{z})\,\mathrm{d}\mathbf{z}, \tag{4.5}$$

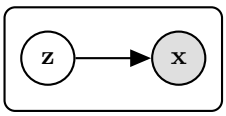


Figure 4.2: Graphical model for the variational autoencoder.

where the likelihood is parameterized by $\boldsymbol{\theta}$. Through the introduction of a (further parameterized) approximate posterior $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$, it is possible to bound the single datum log marginal likelihood from below in the standard variational way i.e. find $\mathcal{L}_{\mathbf{x}}(\boldsymbol{\theta}, \boldsymbol{\phi})$ such that

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) \geq \mathcal{L}_{\mathbf{x}}(\boldsymbol{\theta}, \boldsymbol{\phi}). \tag{4.6}$$

Note that this is just a restatement of the same setting we have outlined previously. We seek to maximise $\mathcal{L}_{\mathcal{X}}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \sum_{\mathbf{x} \in \mathcal{X}} \mathcal{L}_{\mathbf{x}}(\boldsymbol{\theta}, \boldsymbol{\phi})$ across the entire dataset of examples. The parameterized functions are represented by neural networks, and we seek to train the model end-to-end using stochastic gradient descent and the reparameterization trick in order to learn the encoder (or approximate posterior) $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$, and the observation decoder (or likelihood) $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$.

We now turn to the form of the lower bound.

Claim.

$$\mathcal{L}_{\mathbf{x}}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \mathbb{E}_{\mathbf{z} \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) \right] - D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})) \tag{4.7}$$

satisfies (4.6).

Derivation. See (A.2.1). □

This is exactly the lower bound alluded to in the previous section. As mentioned, it is often the case in practice that the form of the approximate posterior and the prior are chosen so that the KL divergence term can be computed analytically. On the other hand, the expectation is evaluated using Monte Carlo. The choice of a standard normal prior and diagonal normal approximate posterior is perhaps the most common, but this has begun to change in the recent past with the use of normalizing flows [Rezende and Mohamed, 2015].

4.2.1 An Intuitive Interpretation of the Lower Bound

The particular form of the lower bound as presented above lends itself to an intuitive interpretation. The expectation over the log likelihood can be viewed as an expected

reconstruction error, using terminology from the autoencoder. If this were the sole term in the lower bound, our latent space would collapse to a set of measure zero. It is the KL divergence term which prevents this, by acting as a 'regularizer' for our approximate posterior, and forcing it to resemble the prior.

While the lower bound for the VAE will be the simplest presented in this work, it is important to stress that the form stays very similar, even with more complex models. As we will show, we can always present the lower bound as a reconstruction error (i.e. the expected log likelihood), with added 'regularization' terms in the form of KL divergences on the latent variables.

4.3 Implementation

With the theory in place, the implementation of a VAE bears a remarkable resemblance to that of an autoencoder. The encoder, which is sometimes referred to as the recognition network, is a neural network which takes as input $\mathbf{x}$, and produces a parameterization of the approximate posterior $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$. Using this, we apply the reparameterization trick in order to sample differentiably from the approximate posterior, yielding a sample $\mathbf{z}$. This is then passed to the decoder, another neural network, which acts as the likelihood $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$. The lower bound is composed of the KL divergence term, which involves the parameterization of the $\mathbf{z}$ variable, and the log likelihood term, which involves the output of the decoder. We can then choose an optimizer, and minimize the negative lower bound by stochastic gradient descent.

4.4 Demonstrations

In this section we outline the implementation of a simple VAE model with a two dimensional $\mathbf{z}$ variable. The trained model is then used to visualize the learned latent space by exploring its encoding and decoding behaviour on a held-out test set. We use the MNIST data set [LeCun et al., 1998], binarized in the standard way [Salakhutdinov and Murray, 2008].

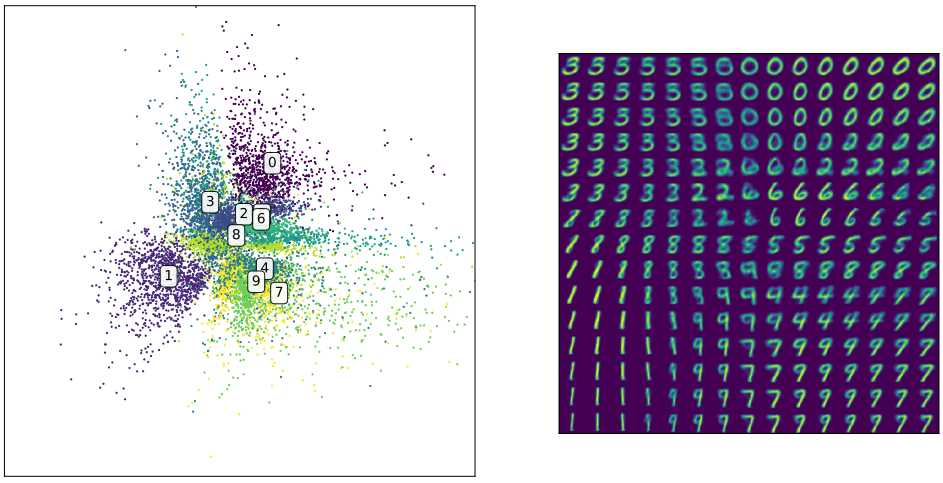
4.4.1 Model Architecture

The encoder consists of two hidden layers, with 500 and 200 units, respectively. The decoder mirrors the encoder, and also contains two hidden layers with the same dimensionality. All hidden layers use batch normalization and a ReLU nonlinearity. The

encoder outputs two unconstrained quantities: the mean and log variance parameterizing the approximate posterior, chosen to be a diagonal Gaussian distribution. We then sample $\mathbf{z}$ from this approximate posterior, and pass this sample to the decoder. The output of the decoder is passed through a sigmoid nonlinearity so that it is constrained to the range $[0, 1]$, like the binarized inputs. The model is trained using mini-batch gradient descent with a batch size of 32, and training takes place for 50 epochs. The optimizer used is Adam, with default parameter settings.

4.4.2 Visualizing the Latent Space

With a trained model in hand, it is possible to visualize the learned latent space in two interesting ways. The first is to create a scatter plot of the embedded test set in the latent space. We do this by passing the test set through the encoder, and use the produced two dimensional means as coordinate values. The coordinates can be coloured by MNIST class, and we would expect to see clustering by digit. This behaviour is demonstrated in Figure 4.3, with label position given by the mean coordinate of the corresponding class.



(a) (b)

Figure 4.3: Visualization of MNIST latent space using VAE.

Next, we can exploit the fact that the posterior is constrained to be similar to a two dimensional standard normal distribution, due to the KL divergence term. Using this, we can grid the latent space in the range $[-b, b]$ along both axes, then pass this grid through the decoder to obtain a grid of MNIST images arrange by their location in the latent space. We use $b = 2.5$ to cover a number of standard deviations of the standard normal prior, and 15 equally spaced points along each axis. The outputted images

are shown in Figure 4.3. Note the correspondence of the latent space representation between the scatter plot and the grid of images.

Chapter 5

The Neural Statistician

Though the VAE restricted itself to a single latent variable per data point, the framework of SGVB extends naturally to more complex graphical models. All we need do is write down a KL divergence between some approximate posterior and the true posterior of a generative process, derive a lower bound, and perform training as before. In this chapter we present one such avenue of research, the neural statistician.

The neural statistician [Edwards and Storkey, 2016] extends SGVB and the VAE to a meta-learning setting. The model now considers datasets $\mathbf{D} \in \mathcal{D}$ consisting of multiple observations $\{\mathbf{x}^{(n)}\}_{n=1}^{N}$, which we explicitly index per data set. Each dataset is assumed to share some context variable $\mathbf{c}$ across the entire set. In other words, $\mathbf{c}$ is sampled once per dataset. Within each dataset, the generative process per item $\mathbf{x}^{(n)}$ remains as before i.e. $\mathbf{z}^{(n)} \to \mathbf{x}^{(n)}$. Two models are presented: a basic extension of the original VAE, and a more complex model with hierarchical latent variables $\{\mathbf{z}^{(l)}\}_{l=1}^{L}$, which are referred to as stochastic layers.

The goal is once again to learn the parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ through optimization of a lower bound on the log marginal likelihood. In this case we seek the variational lower bound $\mathcal{L}_{\mathbf{D}}(\boldsymbol{\theta}, \boldsymbol{\phi})$ such that

$$\log p_{\boldsymbol{\theta}}(\mathbf{D}) \geq \mathcal{L}_{\mathbf{D}}(\boldsymbol{\theta}, \boldsymbol{\phi}), \tag{5.1}$$

and then look to maximize $\mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \sum_{\mathbf{D} \in \mathcal{D}} \mathcal{L}_{\mathbf{D}}(\boldsymbol{\theta}, \boldsymbol{\phi})$ across the entire collection of datasets.

Due to the fact that we now need to consider items grouped by dataset $\mathbf{D}$, along with a hierarchical structure, we stress the slight change notation from the VAE model. As mentioned, we now write $\mathbf{D} = \{\mathbf{x}^{(n)}\}_{n=1}^{N}$, where N is the number of samples per dataset.

Note the latent variables are also grouped by dataset using the notation $\mathbf{z} = \{\mathbf{z}^{(n)}\}_{n=1}^{N}$. To indicate the l^{th} level in the hierarchy of the n^{th} latent $\mathbf{z}$, we write $\mathbf{z}^{(n)(l)}$.

5.1 Models

5.1.1 Basic Model

The basic model is an extension of the VAE which introduces a context variable $\mathbf{c}$. The corresponding graphical model, shown in Figure 5.1, bears a close resemblance to that of a topic model [Blei et al., 2003].

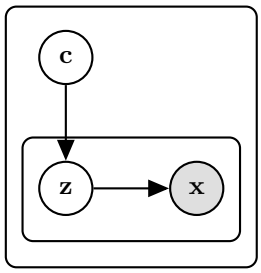


Figure 5.1: Neural statistician basic model. The plates denote the fact that $\mathbf{c}$ is shared across each dataset $\mathbf{D}$, while $\mathbf{z}$ varies per instance $\mathbf{x} \in \mathbf{D}$.

The generative process for the basic model is given by

$$p_{\boldsymbol{\theta}}(\mathbf{D}) = \int p_{\boldsymbol{\theta}}(\mathbf{D}|\mathbf{c})p(\mathbf{c})\,\mathrm{d}\mathbf{c}, \tag{5.2}$$

where

$$\begin{aligned} p_{\boldsymbol{\theta}}(\mathbf{D}|\mathbf{c}) &= \prod_{n=1}^{N} p_{\boldsymbol{\theta}}(\mathbf{x}^{(n)}|\mathbf{c}) \\ &= \prod_{n=1}^{N} \int p_{\boldsymbol{\theta}}(\mathbf{x}^{(n)}|\mathbf{z}^{(n)})p_{\boldsymbol{\theta}}(\mathbf{z}^{(n)}|\mathbf{c})\,\mathrm{d}\mathbf{z}^{(n)}. \end{aligned} \tag{5.3}$$

As before, we can derive a variational lower bound on the log evidence.
Claim.

$$\mathcal{L}_{\mathbf{D}}(\boldsymbol{\theta}, \boldsymbol{\phi}) = R_{\mathbf{D}} - (L_{\mathbf{D}} + C_{\mathbf{D}}), \tag{5.4}$$

where

$$R_{\mathbf{D}} = \mathbb{E}_{\mathbf{c}\sim q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{D})}\left[\sum_{n=1}^{N} \mathbb{E}_{\mathbf{z}^{(n)}\sim q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)}|\mathbf{c},\mathbf{x}^{(n)})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(n)}|\mathbf{z}^{(n)})]\right], \tag{5.5}$$

$$L_{\mathbf{D}} = \mathbb{E}_{\mathbf{c}\sim q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{D})}\left[\sum_{n=1}^{N} D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)}|\mathbf{c},\mathbf{x}^{(n)}) \parallel p_{\boldsymbol{\theta}}(\mathbf{z}^{(n)}|\mathbf{c}))\right], \tag{5.6}$$

$$C_{\mathbf{D}} = D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{D}) \parallel p(\mathbf{c})), \tag{5.7}$$

satisfies (5.1).

Derivation. See (A.2.2.1). □

Note that the essential difference between the lower bound term for the neural statistician and that for the VAE is the addition of a context divergence term $C_{\mathbf{D}}$. As we've previously mentioned, the lower bound still consists of a reconstruction term, and KL divergence 'regularization'.

5.1.2 Full Model

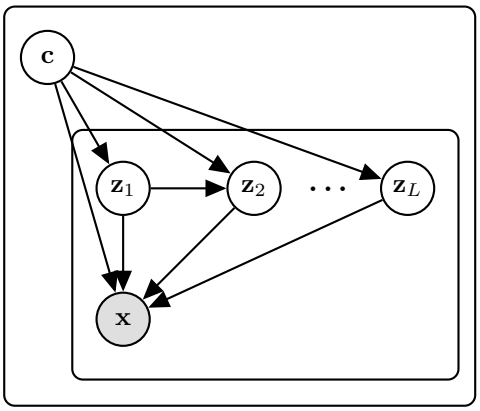


Figure 5.2: Neural statistician full model. The $\{\mathbf{z}^l\}_{l=1}^{L}$ form a hierarchy of latent variables, each of which is passed $\mathbf{c}$ directly, and the context $\mathbf{c}$ is now connected to each observation $\mathbf{x}$ directly.

The full model adds a hierarchical structure for the $\mathbf{z}$ variables. The context is passed to each level of the hierarchy directly, with a view to explaining away generic aspects of the data. This was inspired by [Sønderby et al., 2016]. In addition, the context variable $\mathbf{c}$, along with each of the $\mathbf{z}^{(l)}$, are connected directly to each observation $\mathbf{x}$. The idea behind these skip connections is to prevent later layers in the hierarchy needing to copy information from previous layers, meaning each level can focus on a specific level of abstraction. A similar approach was outlined in [Maaløe et al., 2016]. See Figure 5.2 for

details. Note that the $\mathbf{z}$ variables now have two associated indices: a superscript l to identify position in the hierarchy, and a subscript i, as before, to facilitate grouping per dataset. Where a subscript or superscript is omitted, the notation indicates a grouping over the omitted index e.g. $\mathbf{z}^{(n)} = \{\mathbf{z}^{(n)(l)}\}_{l=1}^{L}$.

The generative process for the full model is given by

$$p(\mathbf{D}) = \int p_{\boldsymbol{\theta}}(\mathbf{D}|\mathbf{c})p(\mathbf{c})\,\mathrm{d}\mathbf{c}, \tag{5.8}$$

where

$$\begin{aligned} p_{\boldsymbol{\theta}}(\mathbf{D}|\mathbf{c}) &= \prod_{n=1}^{N} p_{\boldsymbol{\theta}}(\mathbf{x}^{(n)}|\mathbf{c}) \\ &= \prod_{n=1}^{N} \int p_{\boldsymbol{\theta}}(\mathbf{x}^{(n)}|\mathbf{z}^{(n)}, \mathbf{c}) p_{\boldsymbol{\theta}}(\mathbf{z}^{(n)}|\mathbf{c})\,\mathrm{d}\mathbf{z}^{(n)} \\ &= \prod_{n=1}^{N} \int p_{\boldsymbol{\theta}}(\mathbf{x}^{(n)}|\mathbf{z}^{(n)}, \mathbf{c}) \left[p_{\boldsymbol{\theta}}(\mathbf{z}^{(n)(1)}|\mathbf{c}) \prod_{l=2}^{L} p_{\boldsymbol{\theta}}(\mathbf{z}^{(n)(l)}|\mathbf{z}^{(n)(l-1)}, \mathbf{c}) \right] \mathrm{d}\mathbf{z}^{(n)}. \end{aligned} \tag{5.9}$$

Claim.

$$\mathcal{L}_{\mathbf{D}}(\boldsymbol{\theta}, \boldsymbol{\phi}) = R_{\mathbf{D}} - (L_{\mathbf{D}} + C_{\mathbf{D}}), \tag{5.10}$$

where

$$R_{\mathbf{D}} = \mathbb{E}_{\mathbf{c}\sim q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{D})} \left[\sum_{n=1}^{N} \mathbb{E}_{\mathbf{z}^{(n)}\sim q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)}|\mathbf{c},\mathbf{x}^{(n)})} [\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(n)}|\mathbf{z}^{(n)}, \mathbf{c})] \right], \tag{5.11}$$

$$\begin{aligned} L_{\mathbf{D}} = \mathbb{E}_{(\mathbf{z}^{(n)},\mathbf{c})\sim q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)},\mathbf{c}|\mathbf{D})} \Bigg[& \sum_{n=1}^{N} D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)(1)}|\mathbf{c}, \mathbf{x}^{(n)}) \parallel p_{\boldsymbol{\theta}}(\mathbf{z}^{(n)(1)}|\mathbf{c})) \\ & + \sum_{n=1}^{N} \sum_{l=2}^{L} D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)(l)}|\mathbf{z}^{(n)(l-1)}\mathbf{c}, \mathbf{x}^{(n)}) \parallel p_{\boldsymbol{\theta}}(\mathbf{z}^{(n)(l)}|\mathbf{z}^{(n)(l-1)}\mathbf{c})) \Bigg], \end{aligned} \tag{5.12}$$

$$C_{\mathbf{D}} = D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{D}) \parallel p(\mathbf{c})), \tag{5.13}$$

satisfies (5.1).

Derivation. See (A.2.2.2). □

In both the basic and full models, approximate posteriors and priors are once again chosen so that KL divergence terms can be computed analytically. In particular, all approximate posteriors are chosen to be diagonal normal, the prior on $\mathbf{c}$ is standard normal, and the prior on $\mathbf{z}$ is diagonal normal. As with the VAE, any expectations are computed using Monte Carlo.

5.2 Implementation

One of the most important aspects for understanding the neural statistician is the shape of mini-batches passed to the model. Like most deep learning models, the VAE deals with batches of shape `(batch_size, n_features)`. However, the fact that our 'data set' is now a collection of smaller data sets means that batches are now of shape `(batch_size, sample_size, n_features)`. This detail is important to keep in mind when considering the shape of representations and variables used in training the model.

The implementation of a neural statistician model is slightly more involved than that of a VAE. The framework necessitates the introduction of a statistic network, which parameterizes $q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{D})$, and reduces across the `sample_size` dimension. This is a module consisting of a pre-pooling network, the pooling operation itself (the sample mean), and a post-pooling network which outputs the parameterization of $q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{D})$.

Also contrary to the VAE, the neural statistician has a parameterized prior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{c})$ over the $\mathbf{z}$ variable. As we have mentioned, this is chosen to be diagonal normal, again parameterized by a neural network. The prior over contexts, $p(\mathbf{c})$, is standard normal.

5.3 Demonstrations

5.3.1 Clustering in the Context Space

As with the latent space for the VAE, we can visualize the context space with the neural statistician. This allows us to see how data sets are distributed in relation to one another. Similarly to the synthetic case presented by [Edwards and Storkey, 2016], we consider a collection of one-dimensional distributions, each of one of four types. Each data set is chosen uniformly at random as a mixture of (two) Gaussians, a Laplacian, an exponential, or a reverse exponential distribution. Each set has a mean m and variance v, selected uniformly at random from the intervals $[-1, 1]$ and $[0.5, 2]$, respectively.

We use a model with two stochastic layers, since the classes of distributions are more complex than those used in the original paper. This provides the model with sufficient representational power to disentangle the data sets effectively. The latent variables $\mathbf{c}$ and $\mathbf{z}$ are of dimension 3 and 32, respectively. The latent decoder, $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{c})$, and approximate posterior over $\mathbf{z}$, $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{c}, \mathbf{D})$ are each diagonal Gaussian, with means and log variances given by three-layer fully connected networks, with a hidden dimension of 128. Similarly, the statistic network, $q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{x})$, consists of a three-layer fully-connected network both before and after pooling. The observation decoder, $p_{\boldsymbol{\theta}}(\mathbf{D}|\mathbf{z}, \mathbf{c})$, also uses a

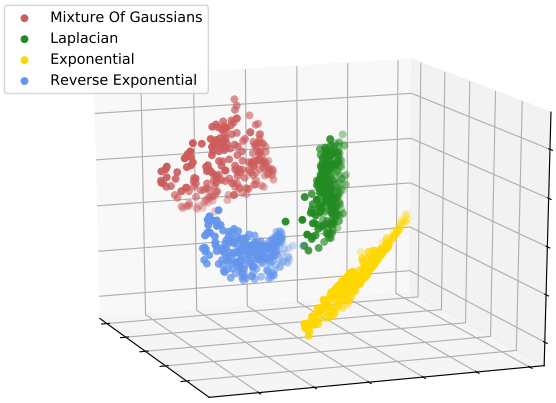
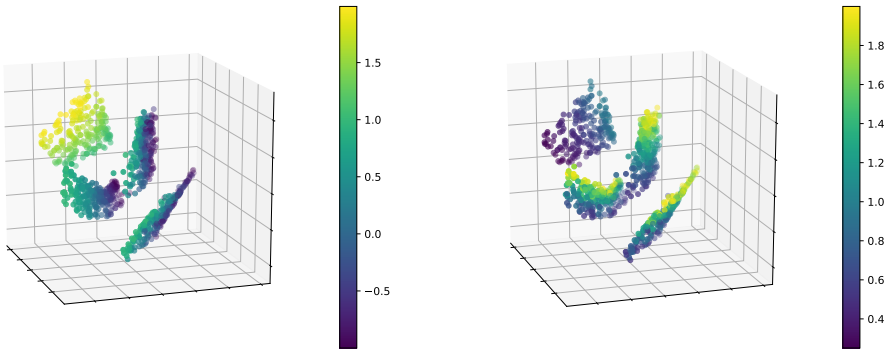


Figure 5.3: Plotting the test set in the context space shows the datasets clearly clustering by distribution.

three-layer sturucture, and we employ a Gaussian likelihood. Each network uses batch normalization and ReLU nonlinearities. Additionally, residual connections were added between the first and last layers of each subnetwork, as these improved the lower bound.



(a) Coloured by mean (b) Coloured by variance

Figure 5.4: Distributions in the context space coloured by moment. The contrasting colour gradients per cluster suggest the moments are learned in orthogonal directions.

The model is trained for 50 epochs using the Adam optimizer with default parameters. A total of 10000 data sets are used, with a batch size of 16, and 200 samples per dataset,. The test set contains 1000 data sets. Gradients are clipped to the range $[-0.5, 0.5]$ to stabilize training. Additionally, a simple weighting scheme is initially applied to the lower bound so that the KL divergence terms do not overly affect the learning of latent variables. In particular, we optimize the following lower bound:

$$\mathcal{L}_{\mathbf{D}}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \omega R_{\mathbf{D}} - \frac{1}{\omega}(L_{\mathbf{D}} + C_{\mathbf{D}}), \tag{5.14}$$

where $\omega = 1 + \alpha$, and α has initial value 1, and is reduced by 0.5 after each epoch. This means we are optimizing the true lower bound after only a few epochs, since α decays exponentially. The weighting allows the approximate posteriors more flexibility initially, since deviating from the prior isn't penalised as much. This idea was also inspired by work on deep VAEs and ladder networks [Sønderby et al., 2016].

To calculate the position of a data set in the context space, we use the statistic network, $q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{D})$, to output the mean and log variance for that data set. The mean is then used as the data set's location in context space. In this way, we can embed a test set in the context space, and visualize how data sets compare. This is shown in Figure 5.3, and we see clearly that the four classes of distribution have clustered accordingly. Furthermore, we can colour the data sets by their mean and variance, shown in Figure 5.4, and examine whether the clustering is meaningful in this respect. Indeed, it seems as if this is the case, with the opposing colour gradients per cluster indicating that these moments have been learned in orthogonal directions.

Chapter 6

The Context-Aware Learner

Following the neural statistician, it is natural to question whether this meta-learning take on SGVB can be further extended. In particular, we might seek a model which could be applied in a setting where datasets exhibit multiple contexts instead of a single shared context. Each observed dataset $\mathbf{D}$ would then consist of some subset of these which are held constant across the entire set, while the rest are free to vary per instance $\mathbf{x}$. The key point is that the contexts which are constant change per data set. Once more, the learning objective remains the same; we look to optimize a lower bound on the log marginal likelihood across the entire collection of datasets in order to learn the parameters $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ of the inference and generative networks, respectively. The theory is outlined in the next section, but first we present an example to demonstrate the usefulness of such a model.

As is common with current work on deep generative models, imagine we have a collection of sets containing images of faces. Each set consists of faces which are consistent in some aspects e.g. one set features only male faces with blond hair, another features only female faces with glasses. Ideally, we would like to create disentangled representations of this data which reflect our high level impressions. The model should learn these in an unsupervised way. Moreover, we might hope that such a model would then be able to reason about joint distributions of which it has seen only some factor. For instance, having never been explicitly shown female faces with blond hair, but instead only these features in isolation, it should be capable of generating suitable samples in the joint style. This is a novel unsupervised and data efficient regime, allowing for powerful modelling even when our data set fails to describe a distribution exhaustively.

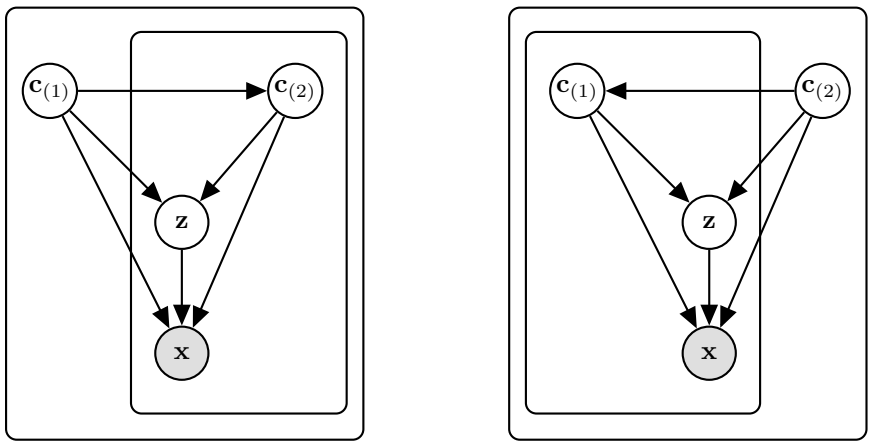
6.1 Model

6.1.1 Theory

Writing down the generative model for the context aware learner is slightly more involved than with the VAE or neural statistician. This is due to the presence of a categorical $\mathbf{a}$ variable, which must be introduced to indicate which contexts are constant, and which are varying in a particular data set. We use the convention $\mathbf{a}_k = 1$ to indicate that the k^{th} context is constant. We also define the constant and varying index sets

$$A_c = \{k : \mathbf{a}_k = 1\} \qquad \text{and} \qquad A_v = \{k : \mathbf{a}_k = 0\},$$

respectively. The generative model is only specified once $\mathbf{a}$ has been sampled. That is, the process is now dynamic per dataset $\mathbf{D}$ in our collection, and the inner 'plate' in Figure 6.1 is effectively free to move. There is not a single distribution over constant contexts, nor is there a single distribution over varying contexts. In fact, we have a total of K distributions for the constant contexts, and likewise for the varying contexts. Once sampled, $\mathbf{a}$ selects which of these should be used in the generative process for each dataset.



(a) (b)

Figure 6.1: Graphical models describing two possible generative processes for the two-context case. (a) $\mathbf{c}_{(1)}$ is held constant while $\mathbf{c}_{(2)}$ varies. (b) Vice versa.

In the most general case, $\mathbf{a}$ is K-dimensional binary vector, taking on 2^K possible values, corresponding to 2^K distinct generative models. In other words, it is equipped to handle datasets which have any subset of K contexts held constant, including the extreme cases of all or none. There are natural simplifications to this setting, namely where we have a guarantee that some fixed number $K' \leq K$ of contexts are constant in each

dataset, or simpler still, where we know exactly one context is held constant in each data set. We present the theory in full generality here.

We use the notation $\mathbf{c}_{(k)}^{(n)}$ to denote the k^{th} context for the n^{th} data point, and use $\mathbf{c}^{(n)}$ to refer to the complete vector of contexts for that data point, similarly to the grouping by omission we utilized previously. We also denote the restriction to either the set of constant or varying contexts by writing $\mathbf{c}_{A_c}$, or $\mathbf{c}_{A_v}$, respectively.

The inclusion of a latent $\mathbf{z}$ variable is made so that the model has the capacity to represent aspects of the data which are neither strictly constant nor strictly varying. To encourage the variable to focus on this type of information, the distribution over $\mathbf{z}$ is not explicitly conditioned on $\mathbf{a}$. Instead, we condition solely on the context variables, so that $\mathbf{z}$ may learn useful features that are not represented by the contexts. In practice, we found that including $\mathbf{z}$ in the model resulted in a better variational lower bound, and more convincing samples.

The generative process for the context-aware learner is given by

$$p(\mathbf{D}) = \sum_{\mathbf{a}} p_{\boldsymbol{\theta}}(\mathbf{D}|\mathbf{a})p(\mathbf{a}) \tag{6.1}$$

where

$$p_{\boldsymbol{\theta}}(\mathbf{D}|\mathbf{a}) = \int \left[\prod_{n=1}^{N} p_{\boldsymbol{\theta}}(\mathbf{x}^n|\mathbf{z}^{(n)}, \mathbf{c}^{(n)}) p_{\boldsymbol{\theta}}(\mathbf{z}^{(n)}|\mathbf{c}^{(n)}) \, \mathrm{d}\mathbf{z}^{(n)} \right] p_{\boldsymbol{\theta}}(\mathbf{c}|\mathbf{a}) \, \mathrm{d}\mathbf{c}. \tag{6.2}$$

Furthermore, we can factorize the context distributions using the Dirac delta as a point mass for the 'copying' of constant contexts, yielding

$$p_{\boldsymbol{\theta}}(\mathbf{c}^{(1)}|\mathbf{a}) = \prod_{k \in A_c} p_{\boldsymbol{\theta}}(\mathbf{c}_{(k)}^{(1)}|\mathbf{a}) \prod_{k \in A_v} p_{\boldsymbol{\theta}}(\mathbf{c}_{(k)}^{(1)}|\mathbf{c}_{A_c}, \mathbf{a}) \tag{6.3}$$

$$p_{\boldsymbol{\theta}}(\mathbf{c}^{(n)}|\mathbf{a}) = \prod_{k \in A_c} \delta\left(\mathbf{c}_{(k)}^{(n)} - \mathbf{c}_{(k)}^{(1)}\right) \prod_{k \in A_v} p_{\boldsymbol{\theta}}(\mathbf{c}_{(k)}^{(n)}|\mathbf{c}_{A_c}, \mathbf{a}), \ n > 1. \tag{6.4}$$

We make the assumption that the approximate posterior factors in an analagous way. It is important make the distinction here between the random variable $\mathbf{c} = \{\mathbf{c}_{(k)}\}_{k=1}^{K}$, and the choice we make to model this random variable with $2K$ distributions, K of which handle the constant case, while the others are responsible for the varying case. In addition, we note that the varying context distributions are conditioned on the constant contexts, so that the model is better informed as to how exactly these varying aspects should be expressed.

With some work, it is possible to derive the variational lower bound for the single datum log marginal likelihood. We find it advantageous to separate the KL divergence terms for the constant and varying contexts.

Claim.

$$\mathcal{L}_{\mathbf{D}}(\boldsymbol{\theta}, \boldsymbol{\phi}) = R_{\mathbf{D}} - (L_{\mathbf{D}} + C_{v\mathbf{D}} + C_{c\mathbf{D}} + A_{\mathbf{D}}), \tag{6.5}$$

where

$$R_{\mathbf{D}} = \mathbb{E}_{\mathbf{z}^{(n)} \sim q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)}|\mathbf{c}^{(n)}, \mathbf{x}^{(n)})} \left[\mathbb{E}_{\mathbf{c}^{(n)} \sim q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{a}, \mathbf{D})} \left[\sum_{n=1}^{N} \log p_{\boldsymbol{\theta}}(\mathbf{x}^{(n)}|\mathbf{z}^{(n)}, \mathbf{c}^{(n)}) \right] \right] \tag{6.6}$$

$$L_{\mathbf{D}} = \mathbb{E}_{\mathbf{c}^{(n)} \sim q_{\boldsymbol{\phi}}(\mathbf{c}^{(n)}|\mathbf{a}, \mathbf{x}^{(n)})} \left[\sum_{n=1}^{N} D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)}|\mathbf{c}^{(n)}, \mathbf{x}^{(n)}) \parallel p_{\boldsymbol{\theta}}(\mathbf{z}^{(n)}|\mathbf{c}^{(n)})) \right] \tag{6.7}$$

$$C_{v\mathbf{D}} = \mathbb{E}_{(\mathbf{c}_{A_c}, \mathbf{a}) \sim q_{\boldsymbol{\phi}}(\mathbf{c}, \mathbf{a}|\mathbf{D})} \left[\sum_{n=1}^{N} \sum_{k \in A_v} D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{c}_{(k)}^{(n)}|\mathbf{c}_{A_c}, \mathbf{a}, \mathbf{x}^{(n)}) \parallel p_{\boldsymbol{\theta}}(\mathbf{c}_{(k)}^{(n)}|\mathbf{c}_{A_c}, \mathbf{a})) \right] \tag{6.8}$$

$$C_{c\mathbf{D}} = \mathbb{E}_{\mathbf{a} \sim q_{\boldsymbol{\phi}}(\mathbf{a}|\mathbf{D})} \left[\sum_{k \in A_c} D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{c}_{(k)}^{(1)}|\mathbf{a}, \mathbf{D}) \parallel p_{\boldsymbol{\theta}}(\mathbf{c}_{(k)}^{(1)}|\mathbf{a})) \right] \tag{6.9}$$

$$A_{\mathbf{D}} = D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{a}|\mathbf{D}) \parallel p(\mathbf{a})) \tag{6.10}$$

satisfies (5.1).

Derivation. See (A.2.3). □

6.1.2 A Weakly Supervised Alternative

Though we have presented the theory in full generality above, one interesting narrowing of focus concerns the case where we know $\mathbf{a}$, a priori. This corresponds to a weak form of supervision in which we tell the model which data sets have a particular context held constant. The model must still learn representations which correspond to these factors of variation, but the task is now simpler since we do not need to worry about additionally inferring $\mathbf{a}$. Indeed, the $\mathbf{a}$ variable can be thought of a binary context label which informs the model of commonalities between data sets, but does not specify the exact nature of these common traits. The theory outlined above still applies; we just remove the KL term from the loss, and provide the model with a context label for each input. This is the aspect we choose to focus on in this thesis, leaving a promising open direction for future research. As we will see, this alternative model can still demonstrate novel capabilities.

6.2 Related Work

The context-aware learner touches on several interesting areas. Transfer learning remains a central interest in modern machine learning, and [Pan and Yang, 2010] provide a

useful survey. We share distributions and network parameters across data sets, and look to leverage learned concepts to generate novel 'out-of-distribution' sets. A similar idea is presented in [Bengio, 2012], with a view to improving classification methods on test samples which may come from a different distribution.

Very recent work has directly explored VAE extensions toward a similar goal. [Bouchacourt et al., 2017] introduce the multi-level VAE, who seek to learn representations which disentangle factors of variation per groups with some common attribute. This is achieved through the introduction of style and content variables, which account for different aspects of the data. A closely related semi-supervised model is proposed by [Siddharth et al., 2017], who also look to encode various facets of data into distinct latent variables. Our work differs in that it naturally extends to many factors of variation, or contexts.

Though we have not explored the generative adversarial network (GAN) [Goodfellow et al., 2014] in this work, it remains a highly influential and popular method for generative modeling (for a brief outline, see section 7.2.4.2). InfoGAN [Chen et al., 2016] takes an information theoretic interpretation of latent variables, and maximizes the mutual information between a conditioning variable and an observation. This results in a subset of the latent variables successfully disentangling factors of variation in an unsupervised fashion, which is useful when we do not possess common grouping knowledge as in the setting of the context-aware learner. Finally, [Donahue et al., 2017] propose an algorithm which can learn subject identity, and a specific observation (e.g. configuration of lighting, pose etc.) of that subject, in a separate manner. Our model is capable of more granular representations, but again relies on meaningful grouping of the data in advance.

6.3 Implementation

The overall sketch of the context-aware learner is similar to what we have seen so far. Each generative and approximate posterior distribution is diagonal Gaussian parameterized by a neural network, and we have a statistic network which collapses across the `sample_size` dimension to parameterize the distributions over constant contexts. However, the introduction of the categorical variable necessitates some extra work, and this is detailed below.

6.3.1 Dealing with a Dynamic Generative Model

The fact that the categorical variable $\mathbf{a}$ can take on 2^K values, and thus specify 2^K possible generative models, also introduces practical difficulties. A priori, we don't know which generative model we need, even when we assume $\mathbf{a}$ is given to us. The solution we propose is to parameterize every distribution which may be used, and use $\mathbf{a}$ as a binary mask to zero out extraneous information. That is, we always generate a mean and log variance for each of the K constant context distributions, and each of the K varying context distributions, and use the mask implied by $\mathbf{a}$ to dynamically select the correct subset. In this way, the samples from the constant and varying context distributions complement one another; where one is zeroed out, the other has nonzero values, and vice versa. The mask also allows us to select the correct mean and log variance terms when computing the various KL divergence contributions.

6.3.2 Categorical Reparamterization

As mentioned, the introduction of multiple contexts also necessitates the addition of a categorical random variable which indicates which contexts are held constant, and which are free to vary. Applying the reparameterization trick to categorical distributions has received attention in the recent past, in an effort to extend the variational autoencoding framework to models with discrete latent variables. Both [Jang et al., 2016] and [Maddison et al., 2016] independently proposed the same reparameterizable distribution to handle this use case, which is a continuous relaxation of a categorical distribution. The former work terms the distribution Gumbel-Softmax, while the latter chooses the portmanteau Concrete. Though both papers consider identical density functions for the relaxation of a K-dimensional categorical variable $\mathbf{y}$, namely

$$p_{\boldsymbol{\pi},\tau}(\mathbf{y}) = (K-1)!\ \tau^{K-1} \prod_{k=1}^{K} \left(\frac{\boldsymbol{\pi}_k \mathbf{y}_k^{-\tau-1}}{\sum_{j=1}^{K} \boldsymbol{\pi}_j \mathbf{y}_j^{-\tau}} \right), \tag{6.11}$$

the manner in which they are integrated into a discrete latent variable model differs (here $\boldsymbol{\pi}$ is the vector of unnormalized class probabilities, and τ is the temperature parameter, which controls how closely the relaxation matches a true categorical distribution). The Gumbel-Softmax samples directly from the distribution using the reparameterization

$$\mathbf{y}_k = \frac{\exp((\log \boldsymbol{\pi}_k + \boldsymbol{g}_k)/\tau)}{\sum_{j=1}^{K} \exp((\log \boldsymbol{\pi}_j + \boldsymbol{g}_j)/\tau)}, \tag{6.12}$$

where $\boldsymbol{g}$ is noise from a Gumbel distribution, which is easily generated using the inverse cumulative density function. The KL divergence term is computed using the categorical implied by the normalized class probabilities (for details see A.4).

On the other hand, the Concrete distribution performs reparameterization based on a log-space treatment, and an auxiliary distribution termed the Exp-Concrete, since taking the exponent of samples from this distribution yields a Concrete distribution. The method views the KL divergence as the expectation of a log density ratio, and this expected ratio is computed by evaluating the true relaxed densities in a Monte Carlo fashion across mini-batches. The paper notes that this approach is the only method giving a guaranteed lower bound on the log evidence in the relaxed setting, unlike the use of the implied categorical by the Gumbel-Softmax.

6.4 Experiments

6.4.1 Black & White MNIST

As a basic proof of concept, we consider binarized MNIST with varying background. This yields $K = 2$ candidate contexts: digit and background colour. Images are either black on a white background, or white on a black background. Each dataset consists of a number of samples which are consistent in exactly one of digit or background colour. Figure 6.2 demonstrates a collection of five datasets with five samples in each. The task is based on seen data, and we look to produce samples in two ways:

- sampling a data set conditioned on a particular context
- sampling a data set conditioned on some input data set with a particular context held constant.

We note that these tasks are likely achievable by a conditional generative model, but we include them here as a first demonstration of our model's ability to also act in this capacity.

Concerning architecture, we initially use a convolutional encoder to produce features for each item in each of our data sets. In other words, the model does not work with the data directly, but instead with extracted features. This encoder consists of nine convolutional layers, divided into groups of three, with 16, 32, and 64 filters, respectively. The observation decoder, parameterizing $p_{\boldsymbol{\theta}}(\mathbf{x}^{(n)}|\mathbf{z}^{(n)}, \mathbf{c}^{(n)})$, mirrors this, using transpose convolution operations. The model still features a statistic network, as in the neural statistician, which now parameterizes a distribution over the constant contexts. Both pre and post-pooling modules feature 3 fully connected layers with 256 units. Networks parameterizing distributions over varying contexts and $\mathbf{z}$ variables also use 3 layers, with 128 units. We use skip connections between the first and last hidden layers in all

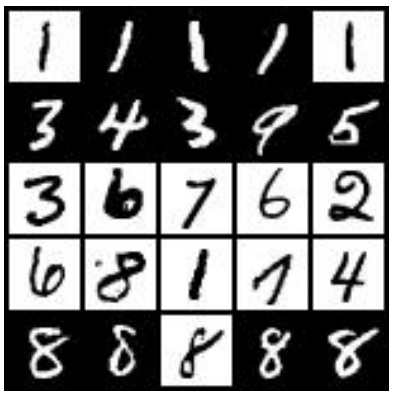


Figure 6.2: Five data sets from black & white MNIST, each containing five samples. Either the digit or the background colour is constant across rows.

fully connected subnetworks. The context variables are 64 dimensional, and $\mathbf{z}$ is 16 dimensional. All layers, fully connected or convolutional, use batch normalization and ReLU activations, with the exception of output layers.

The model is trained for 100 epochs using the Adam optimizer with a learning rate of 3×10^{-4}. We create 20000 data sets in advance, where the proportion of constant digit data sets to constant background data sets is 5:1, due to the fact that we have ten digits, but only two background colours. The batch size is 32, and we use 10 samples per data set. The weighting scheme is similar to that used in the neural statistician, in order to facilitate flexibility of the latent variables during initial training. The only modification we make is to decrease the weighting of the latent variables, giving a criterion

$$\mathcal{L}_{\mathbf{D}}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \omega R_{\mathbf{D}} - \frac{1}{\omega^2}(L_{\mathbf{D}} + C_{v\mathbf{D}} + C_{c\mathbf{D}}), \tag{6.13}$$

where $\omega = 1 + \alpha$, and α is annealed as before. We use a Bernoulli likelihood.

To sample based on a particular context, all we need do is provide the model with a batch of context labels, and run the generative model forward. The output should consist of one collection of data sets with constant digit and varying background, and another collection of data sets with constant background and varying digit. This is shown in Figure 6.3, which demonstrates 20 total data sets, 2 per row, with constant digit on the left, and constant background on the right.

Next we look to condition on a batch of data sets, and produce new samples with the appropriate contexts held constant. To do this, we calculate the approximate posteriors over constant contexts, and condition on the mean of these distributions to generate samples using the most likely contexts given the data. The results are shown in Figure 6.4. The format is similar to previous figure, but now the conditioning data set is displayed on the left, with the conditioned output on the right. The samples

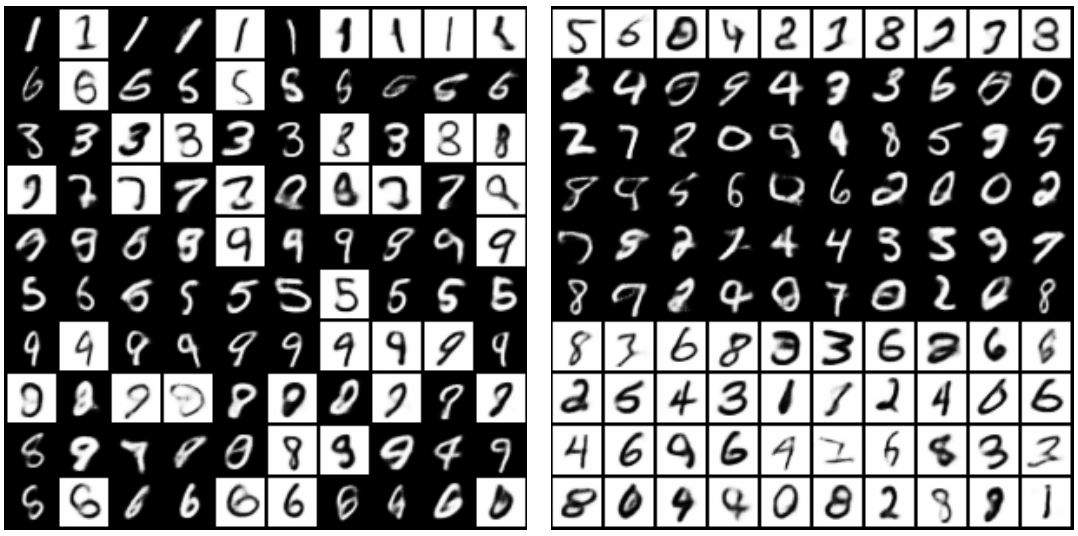


Figure 6.3: Samples generated by the model given context labels, for black & white MNIST. Left side depicts constant digit, right side constant background.

correspond to the correct constant context in each data set. Additionally, there are two interesting properties to note. In the fourth row, the model has produced a data set with black background and varying digit, but contains digits which are not present in the conditioning set. In addition, the row second from bottom shows a data set constant in the digit seven, but each sample also features a black background, which happened by chance during the creation of the data. The model is able to handle this detail, and produces samples of the digit seven, but with varying background.

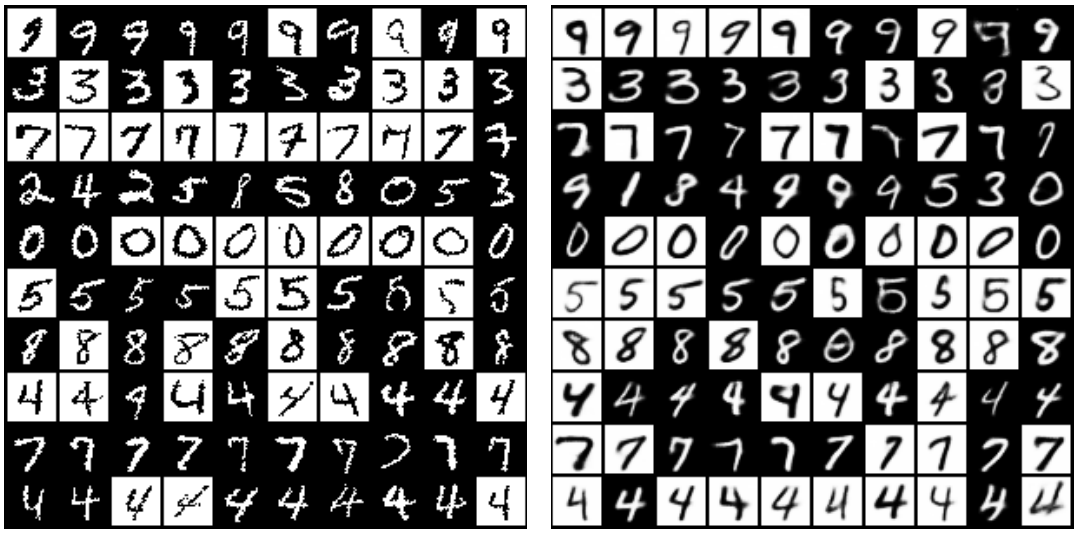


Figure 6.4: Conditional samples generated by the model for black & white MNIST. Left side shows data sets provided to the model, right side shows samples conditioned on these data sets.

6.4.2 Rotated MNIST

In this experiment, we pose a slightly more challenging problem. Background is a relatively simple factor of variation to discover, so we choose to replace this context with digit rotation instead. Thus, we again have $K = 2$ contexts, digit and angle of rotation, exactly one of which is held constant in each dataset. Figure 6.5 demonstrates a collection of five data sets with five examples in each. We use 8 distinct rotations, spaced evenly between 0° and 360°.

Figure 6.5: Five data sets from rotated MNIST, each containing five samples. Either the digit or the angle of rotation is constant across rows.

So far the context-aware learner has acted as a conditional generative model. However, the framework is designed for more interesting generalization abilities. For instance, we might train the model using data sets that hold exactly one context constant, but at test time ask it to produce samples of data sets with neither or both of these held constant. Furthermore, we might look to condition on two data sets with distinct contexts held constant, and synthesize a data set with the specific constant properties of both. These 'out-of-distribution' tasks are the focus of this section.

The architecture remains as in the experiments featuring varying background, as does the training scheme. We also use the same weighting for the lower bound, and we found that this was important for good convergence. The proportion of constant digit data sets to constant rotation data sets was evenly split despite the slight difference in number of classes per context, and this also seemed to benefit learning.

Sampling new context combinations is straightforward; we provide the model with context labels it hasn't seen before, and examine the output of the generative process. Here, since each training data set has exactly one context held constant, corresponding to two-dimensional one-hot context labels, this means we are providing the model with either two-dimensional vectors of zeros, or two-dimensional vectors of ones. The results

are shown in Figure 6.6. On the left, we have data sets consisting of varying digit and varying rotation. That is, there is no discernible pattern, since neither of the contexts are constant. On the other hand, the right hand side demonstrates samples which are consistent in both digit and rotation, and have both contexts held constant. We stress that the model has not been trained on either of these types of data sets, and can readily transfer the concepts it has learned to generate new combinations of contexts.

Figure 6.6: Left side shows samples with neither rotation nor digit held constant, right side shows samples with both held constant.

Synthesizing a data set from two others is similar to the conditional sampling we demonstrated in the previous experiment. In this case, we need to calculate the approximate posteriors over constant contexts for the data sets with those corresponding contexts held constant. As before, these quantities can then be passed through the generative process so that we condition on the most likely contexts given the data. The results are shown in Figure 6.7. The top row shows the data set with the constant digit '5', while the second displays the data set with constant rotation, 90° clockwise. Thus, we would expect the model to produce data sets of the digit '5', rotated 90° clockwise. Eight such synthesized data sets are shown in the remaining rows.

6.4.3 The Fully Unsupervised Case

Finally, we present some results from the fully unsupervised case i.e. when $\mathbf{a}$ is not known, and must be inferred. This is where categorical reparameterization is necessary. We emphasize that this section presents preliminary results, and is included solely as a proof of concept.

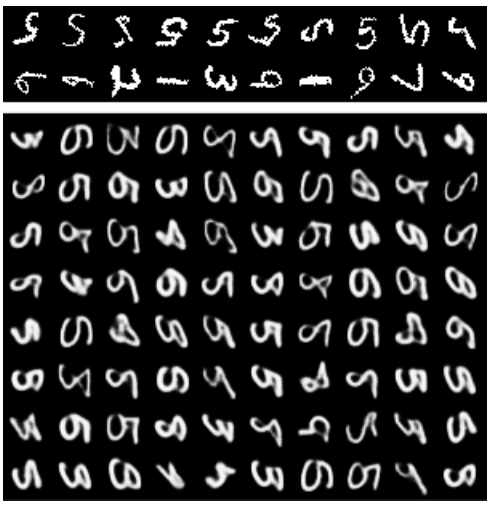


Figure 6.7: Top two rows show data sets with specific contexts held constant. Bottom eight rows are samples of data sets synthesized by combining these two constant properties.

The major architectural change required is the addition of a network which parameterizes $q_\phi(\mathbf{a}|\mathbf{D})$. This is another module similar to the statistic network of the neural statistician, which must collapse across the `sample_size` dimension. We make the choice to once again perform mean pooling, and the module closely resembles the structure we have seen thus far. We use a pre-pooling network with 3 layers of 512 units, and a post-pooling network with 3 layers of 128 units, both of which feature skip connections. For categorical reparameterization, we choose the Gumbel-Softmax, and use a temperature annealing scheme as presented in that paper, beginning at $\tau = 1$, and reduced to $\tau = 0.5$ over the course of training. Since this method outputs soft categories, and we require hard binary labels, we perform the 'straight-through' approximation outlined in the paper, where we binarize the samples, but take gradients with respect to the soft samples.

The remaining architecture, along with the training method, remain as before, with two important exceptions:

- we use black & white MNIST with only two digits, '0' and '1', with an even proportion of constant digit and constant background data sets, to set an easier task overall,

- we use an extreme weighting of the KL terms in the lower bound in order to strongly encourage the model to rely on the categorical variable $\mathbf{a}$.

In particular, we use the following weighting scheme

$$\mathcal{L}_{\mathbf{D}}(\boldsymbol{\theta}, \boldsymbol{\phi}) = R_{\mathbf{D}} - \omega(L_{\mathbf{D}} + C_{v\mathbf{D}} + C_{c\mathbf{D}}) - \frac{1}{\omega} A_{\mathbf{D}}, \tag{6.14}$$

where $\omega = 1 + \alpha$, $\alpha = 100$ is the starting value, and α is annealed much more slowly so that the weighting effects are present across the entirety of training. It is vital to remember that we are not optimizing the variational lower bound here, and we tread on theoretically weak ground. Instead, we have a heuristic which makes it extremely expensive for the model to store information in latent variables other than $\mathbf{a}$, and conversely, extremely cheap to make $\mathbf{a}$ flexible, since it receives such a small penalty for deviating from the prior. This forces reliance on $\mathbf{a}$ as a consequence.

Various metrics for the training procedure are shown in Figure 6.8. One particularly novel feature of the fully unsupervised case is that we can fashion a pseudo-accuracy metric by comparing the model's sampled $\mathbf{a}$ variable to the true context label for a given data set. This is a unique aspect of the context-aware learner; we can induce something akin to an easily interpretable classic evaluation method for fully unsupervised learning. We use the term pseudo-accuracy here since the configuration of $\mathbf{a}$ is only unique up to permutation; the model may assign the k contexts in any order, and each of these is perfectly valid. In other words, an 'accuracy' of 0% is also optimal here, and indicates the model has learned the contexts in exactly the opposite way we have arbitrarily assigned them. This perhaps gives an explanation as to why the fully unsupervised model is difficult to train; in the weakly-supervised case, we observe $\mathbf{a}$, the assigment of contexts is clear, and the inference task becomes easier.

The highest value achieved for the pseudo-accruacy was 92.3%, but this is unstable, and oscillates significantly. Across the whole training run, we see the KL divergence terms for $\mathbf{c}_{A_v}$ and $\mathbf{c}_{A_c}$ increase steadily, and the KL term for $\mathbf{z}$ increase right at the end, indicating their corresponding approximate posterior distributions gaining expressivity. This is mirrored by a steady decrease in the accuracy during the second half of training. Once more, we stress that these results are based on heuristic manipulation of the true lower bound, and significant exploration is required to achieve a reliable and theoretically sound approach. Nevertheless, the outcome is encouraging, and suggests the fully unsupervised case is an attainable goal.

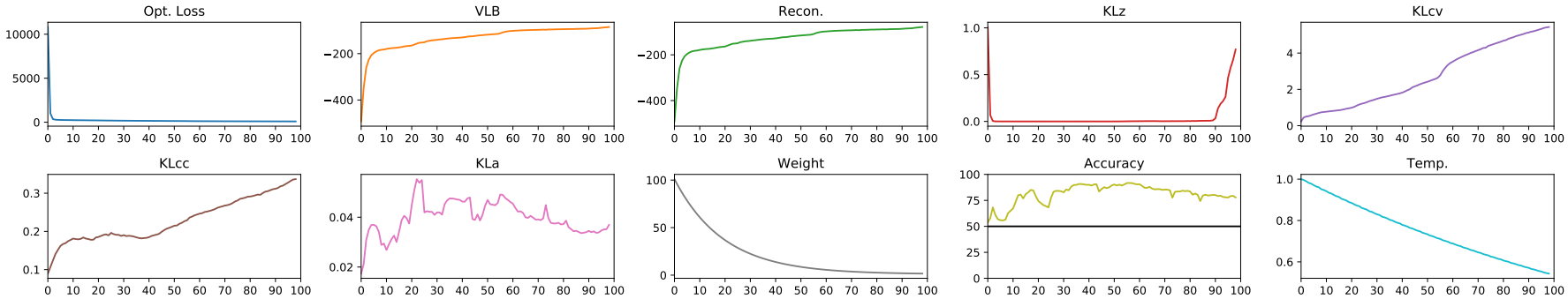


Figure 6.8: Experimental metrics for fully unsupervised context-aware learner over 100 epochs of training. Top row, left to right: Criterion optimized by Adam, true variational lower bound, reconstruction term (expected log likelihood), KL divergence on $\mathbf{z}$, KL divergence on $\mathbf{c}_v$. Bottom row, left to right: KL divergence on $\mathbf{c}_c$, KL divergence on $\mathbf{a}$, weighting for lower bound terms, 'accuracy' in prediction of context labels, temperature for Gumbel-Softmax sampling.

Chapter 7

Conclusion & Further Work

In this chapter, we recapitulate our work so far, and look to a variety of promising avenues for future development.

7.1 Our Contribution

We have given an overview of modern stochastic variational inference, presented as a confluence of classic variational inference and modern deep learning. Using this, we have explored the variational autoencoder as a realization of SGVB, and seen how the framework has been readily extended to the neural statistician, which offers a meta-learning take on learning representations of data. Based on this, we have proposed the context-aware learner, a model suited to disentangling numerous factors of variation across data sets. Following a comprehensive theory, a series of experiments demonstrate the usefulness of such a model, and we observe results which align with our expectations. We now turn our attention to the next step: extending and improving the model.

7.2 Improving the Current Model

The context-aware learner, as described and implemented in the previous chapter, stands to benefit from a number of interesting modifications and avenues of exploration.

7.2.1 The Fully Unsupervised Case

Perhaps the most prominent extension involves the the case where we must also infer $\mathbf{a}$ i.e. the model is fully unsupervised. The success of this modification hinges

on a reliable method for integrating a categorical random variable into the SGVB framework. This is a problem with a number of aspects to consider; we must successfully incorporate categorical reparameterization in our model, as well as determine the best way for our network to collapse across samples per data set in order to provide a good parameterization. The former issue has been broached in the previous chapter, and it seems as if this will receive increased attention in the near future. The latter problem of reasoning across data sets has also benefitted from recent developments, and we look to the novel work on relational networks [Santoro et al., 2017] as a possible improvement to the mean-pooling we currently employ. A relational structure might be particularly advantageous for our use case. The question we ask of the model is implcit: do there exist constant contexts in this data set?

7.2.2 Exploring Data Sets

As noted, for each context added to a prospective data set, the model can learn to hold that context constant, or allow it to vary. In other words, we get an exponential increase in representational ability for each context added. This is especially useful when we have a multi-context data set which is not exhaustive in all possible categories. We have seen the powerful generalization ability of the context-aware learner, and look to exploit this advantage to the fullest extent.

We might also look to explore a more diverse range of data sets. This requires some measure of creativity, since many modern resources offer a simpler structure, amenable to classification tasks or straightforward generative modeling, instead of the multi-context setting we require. Nevertheless, achieving success with a task involving face images like that specified at the outset of the previous chapter poses an exciting challenge. We might also consider the task of disentangling style and content in various art media, such as paintings or music.

7.2.3 Architecture

The context-aware learner can benefit from architectural upgrades in a natural way. As a direct result of the surge in deep learning research, new methods improving training stability and model behaviour are continuously available. We have mentioned a few these in terms of normalization strategies (layer, weight, and self-normalization), skip connections (residual networks), and improved optimizers (Adam). As new work in these areas comes into widespread practice, it is easily incorporated into the context-aware learner. We note that the model as presented in the previous chapter offers great

flexibility with image data in particular, since it is bookended by a convolutional encoder and decoder. When dealing with more complex images, we need only leverage recent developments in computer vision [Huang et al., 2016] to compensate.

7.2.4 Digging Deeper into the Lower Bound

As we added increased complexity to the variational autoencoder, we saw how the lower bound changed by the addition of KL divergence terms for each of the latent variables. In each case, we used a diagonal Gaussian to parameterize these distributions. This choice is made almost exclusively for analytic convenience; it is not too much work to write down the KL divergence between two diagonal Gaussian distributions (see A.3 for details). However, there is generally no reason to believe these latent variables factor as such, and are totally uncorrelated. Closed form expressions are helpful, but if better tools are available to specify more flexible distibutions, our methods should be updated accordingly. In this section, we explore recent developments which improve upon these analytically tractable, but often prohibitive, choices.

7.2.4.1 Normalizing Flows

We have briefly mentioned normalizing flows when talking about the standard choice of posterior in SGVB models, which has up until recently been the diagonal Gaussian. Normalizing flows for variational inference [Rezende and Mohamed, 2015] offers an interesting progression from this standard. The central idea is to begin with a simple density, such as the diagonal Gaussian we have been utilizing all along, and applying a series of invertible transformations until the density is sufficiently complex. This idea was further developed with the introduction of inverse autoregressive flow [Kingma et al., 2016], whose flow offers particularly rich and expressive approximate posteriors. This is an active area of research which we can integrate directly into our work as new findings are released, in order to improve the lower bound, achieve better latent variable representations, and generate more convincing samples.

7.2.4.2 Implicit Distributions and Adversarial Training

The topic of implicit distributions in the context of variational inference has received considerable attention in the very recent past [Huszár, 2017] [Tran et al., 2017]. The working definition for these distributions encompasses models whose density may be intractable, but possess the following properties:

- we can sample from them, and/or compute expectations with respect to them,
- we can compute gradients of these expectations with respect to the model parameters.

Implicit distributions allow us to parameterize extremely flexible densities, and have seen extensive use in generative modeling over the past number of years. A widely used example is the generative adversarial network (GAN) [Goodfellow et al., 2014], which looks to directly model a data distribution by passing samples from a simple distribution (such as a uniform or Gaussian) through a neural network, or generator. Through the introduction of a discriminator, which attempts to differentiate between samples from the generator, and samples from the true (empirical) distribution, the model is trained using a two player mini-max game. The GAN literature is considerable, and we direct the reader to [Goodfellow, 2016] for more details.

While GANs are typically used to directly model data distributions, we might also wonder whether they can be applied in the approximation of latent variable distributions for variational inference. This leads to the topic of adversarial training for VAE-based models, and more generally, the fusion of GAN-type training with VAE-type training, which has a wide variety of approaches [Makhzani et al., 2015] [Mescheder et al., 2017]. The α-GAN [Rosca et al., 2017] encompasses many of these ideas, and also provides interesting extensions. We expect this direction of research to equip us with yet another set of tools with which we might enrich our the expressivity of the distributions in our model.

7.3 Final Thoughts

The SGVB framework offers a great degree of flexibility for specifying and training complex graphical models using the machinery of deep learning. We emphasize the VAE as one particular instance of this framework, and highlight that SGVB facilitates the implementation of more elaborate generative models, with many interesting capabilities. The context-aware learner is one such model, whose framework offers significant potential for future work, and we anticipate many more exciting developments at the crossroads of stochastic variational inference and deep learning for generative modeling.

Bibliography

[Ba et al., 2016] Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

[Bengio, 2012] Bengio, Y. (2012). Deep learning of representations for unsupervised and transfer learning. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pages 17–36.

[Blei et al., 2017] Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American Statistical Association*, (just-accepted).

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

[Bouchacourt et al., 2017] Bouchacourt, D., Tomioka, R., and Nowozin, S. (2017). Multi-level variational autoencoder: Learning disentangled representations from grouped observations. *arXiv preprint arXiv:1705.08841*.

[Chen et al., 2016] Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., and Abbeel, P. (2016). Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2172–2180.

[Cybenko, 1989] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314.

[Doersch, 2016] Doersch, C. (2016). Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*.

[Donahue et al., 2017] Donahue, C., Balsubramani, A., McAuley, J., and Lipton, Z. C. (2017). Semantically decomposing the latent spaces of generative adversarial networks. *arXiv preprint arXiv:1705.07904*.

[Edwards and Storkey, 2016] Edwards, H. and Storkey, A. (2016). Towards a neural statistician. *arXiv preprint arXiv:1606.02185*.

[Giles, 2008] Giles, M. (2008). An extended collection of matrix derivative results for forward and reverse mode automatic differentiation.

[Glorot and Bengio, 2010] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256.

[Goodfellow, 2016] Goodfellow, I. (2016). NIPS 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

[Goodfellow et al., 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[Hinton and Salakhutdinov, 2006] Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507.

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

[Hornik, 1991] Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257.

[Huang et al., 2016] Huang, G., Liu, Z., Weinberger, K. Q., and van der Maaten, L. (2016). Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*.

[Huszár, 2017] Huszár, F. (2017). Variational inference using implicit distributions. *arXiv preprint arXiv:1702.08235*.

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Bach, F. and Blei, D., editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France. PMLR.

[Jang et al., 2016] Jang, E., Gu, S., and Poole, B. (2016). Categorical reparameterization with Gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

[Jordan et al., 1999] Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., and Saul, L. K. (1999). An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233.

[Kanungo et al., 2002] Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C. D., Silverman, R., and Wu, A. Y. (2002). An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):881–892.

[Kingma and Ba, 2014] Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Kingma et al., 2016] Kingma, D. P., Salimans, T., and Welling, M. (2016). Improving variational inference with inverse autoregressive flow. *arXiv preprint arXiv:1606.04934*.

[Kingma and Welling, 2013] Kingma, D. P. and Welling, M. (2013). Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*.

[Klambauer et al., 2017] Klambauer, G., Unterthiner, T., Mayr, A., and Hochreiter, S. (2017). Self-normalizing neural networks. *arXiv preprint arXiv:1706.02515*.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[Kuczma, 2009] Kuczma, M. (2009). *An introduction to the theory of functional equations and inequalities: Cauchy's equation and Jensen's inequality*. Springer Science & Business Media.

[LeCun et al., 1998] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[Maaløe et al., 2016] Maaløe, L., Sønderby, C. K., Sønderby, S. K., and Winther, O. (2016). Auxiliary deep generative models. *arXiv preprint arXiv:1602.05473*.

[MacKay, 2003] MacKay, D. J. (2003). *Information theory, inference and learning algorithms*. Cambridge University Press.

[Maddison et al., 2016] Maddison, C. J., Mnih, A., and Teh, Y. W. (2016). The Concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*.

[Mahajan et al., 2009] Mahajan, M., Nimbhorkar, P., and Varadarajan, K. (2009). The planar k-means problem is NP-hard. In *International Workshop on Algorithms and Computation*, pages 274–285. Springer.

[Makhzani et al., 2015] Makhzani, A., Shlens, J., Jaitly, N., Goodfellow, I., and Frey, B. (2015). Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*.

[Mescheder et al., 2017] Mescheder, L., Nowozin, S., and Geiger, A. (2017). Adversarial variational bayes: Unifying variational autoencoders and generative adversarial networks. *arXiv preprint arXiv:1701.04722*.

[Minka, 2001] Minka, T. P. (2001). Expectation propagation for approximate Bayesian inference. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 362–369. Morgan Kaufmann Publishers Inc.

[Murphy, 2012] Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. MIT Press.

[Murray, 2007] Murray, I. (2007). *Advances in Markov chain Monte Carlo methods*. University of London, University College London (United Kingdom).

[Nielsen, 2015] Nielsen, M. A. (2015). Neural networks and deep learning.

[Pan and Yang, 2010] Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359.

[Petersen et al., 2008] Petersen, K. B., Pedersen, M. S., et al. (2008). The matrix cookbook. *Technical University of Denmark*, 7:15.

[Rezende and Mohamed, 2015] Rezende, D. J. and Mohamed, S. (2015). Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*.

[Rezende et al., 2014] Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*.

[Rosca et al., 2017] Rosca, M., Lakshminarayanan, B., Warde-Farley, D., and Mohamed, S. (2017). Variational approaches for auto-encoding generative adversarial networks. *arXiv preprint arXiv:1706.04987*.

[Ruder, 2016] Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

[Russakovsky et al., 2015] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252.

[Salakhutdinov and Murray, 2008] Salakhutdinov, R. and Murray, I. (2008). On the quantitative analysis of deep belief networks. In *Proceedings of the 25th international conference on Machine learning*, pages 872–879. ACM.

[Salimans and Kingma, 2016] Salimans, T. and Kingma, D. P. (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pages 901–909.

[Santoro et al., 2017] Santoro, A., Raposo, D., Barrett, D. G., Malinowski, M., Pascanu, R., Battaglia, P., and Lillicrap, T. (2017). A simple neural network module for relational reasoning. *arXiv preprint arXiv:1706.01427*.

[Siddharth et al., 2017] Siddharth, N., Paige, B., de Meent, V., Desmaison, A., Wood, F., Goodman, N. D., Kohli, P., Torr, P. H., et al. (2017). Learning disentangled representations with semi-supervised deep generative models. *arXiv preprint arXiv:1706.00400*.

[Sønderby et al., 2016] Sønderby, C. K., Raiko, T., Maaløe, L., Sønderby, S. K., and Winther, O. (2016). How to train deep variational autoencoders and probabilistic ladder networks. *arXiv preprint arXiv:1602.02282*.

[Tran et al., 2017] Tran, D., Ranganath, R., and Blei, D. M. (2017). Deep and hierarchical implicit models. *arXiv preprint arXiv:1702.08896*.

[Wainwright et al., 2008] Wainwright, M. J., Jordan, M. I., et al. (2008). Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1–2):1–305.

Appendix A

Supporting Results

A.1 Properties of KL Divergence

A.1.1 Non-negativity

Lemma. $D_{KL}(q(\mathbf{x}) \parallel p(\mathbf{x})) \geq 0$.

The non-negativity of the KL divergence can be proven using Jensen's inequality. The inequality is stated as follows.

Lemma (Jensen's Inequality)**.** *Let* $\mathbf{x}$ *be a random variable taking values in* $\mathbb{R}^n$*, let* $p(\mathbf{x})$ *be some distribution over* $\mathbf{x}$*, let* $y = y(\mathbf{x}) : \mathbb{R}^n \mapsto \mathbb{R}$ *be a function defining a real-valued random variable in terms of* $\mathbf{x}$*, and let* $\varphi : \mathbb{R} \mapsto \mathbb{R}$ *be convex. Then*

$$\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[\varphi(y(\mathbf{x}))\right] \leq \varphi\left(\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[y(\mathbf{x})\right]\right).$$

Proof of Jensen's inequality. The proof is beyond the scope of this thesis. We refer the reader to [Kuczma, 2009] for a thorough discussion. □

Proof of non-negativity of KL divergence. Consider Jensen's inequality in the following

case: $y(\mathbf{x}) = \frac{q(\mathbf{x})}{p(\mathbf{x})}$ and $\phi(y) = \log(y)$. We have

$$\begin{aligned}
D_{KL}(q(\mathbf{x}) \parallel p(\mathbf{x})) &= \int q(\mathbf{x}) \log \frac{q(\mathbf{x})}{p(\mathbf{x})} \, \mathrm{d}\mathbf{x} \\
&= -\int q(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})} \, \mathrm{d}\mathbf{x} \\
&= -\mathbb{E}_{\mathbf{x}\sim q(\mathbf{x})} \left[\log \frac{p(\mathbf{x})}{q(\mathbf{x})} \right] \\
&\geq -\log \left(\mathbb{E}_{\mathbf{x}\sim q(\mathbf{x})} \left[\frac{p(\mathbf{x})}{q(\mathbf{x})} \right] \right) \quad \text{(Jensen's inequality)} \\
&= -\log \left(\int q(\mathbf{x}) \frac{p(\mathbf{x})}{q(\mathbf{x})} \, \mathrm{d}\mathbf{x} \right) \\
&= -\log \left(\int p(\mathbf{x}) \, \mathrm{d}\mathbf{x} \right) \\
&= -\log(1) \quad \text{(properties of pdf)} \\
&= \log(1) \\
&= 0.
\end{aligned}$$

□

A.1.2 Asymmetry

Lemma. $D_{KL}(q(\mathbf{x}) \parallel p(\mathbf{x})) \neq D_{KL}(p(\mathbf{x}) \parallel q(\mathbf{x}))$.

Proof. Consider $D_{KL}(q(\mathbf{x}) \parallel p(\mathbf{x})) - D_{KL}(p(\mathbf{x}) \parallel q(\mathbf{x}))$.

$$\begin{aligned}
D_{KL}(q(\mathbf{x}) \parallel p(\mathbf{x})) - D_{KL}(p(\mathbf{x}) \parallel q(\mathbf{x})) &= \int q(\mathbf{x}) \log \frac{q(\mathbf{x})}{p(\mathbf{x})} \, \mathrm{d}\mathbf{x} - \int p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})} \, \mathrm{d}\mathbf{x} \\
&= \int (q(\mathbf{x}) + p(\mathbf{x})) \log \frac{q(\mathbf{x})}{p(\mathbf{x})} \, \mathrm{d}\mathbf{x}.
\end{aligned}$$

This integral is non-zero in general. □

A.2 Variational Lower Bound Derivations

A.2.1 Variational Autoencoder

Derivation. Consider the KL divergence between the approximate posterior and the true posterior,

$$D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z}|\mathbf{x})) = \int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \log \frac{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})} \, \mathrm{d}\mathbf{z}.$$

Substituting $p(\mathbf{z}|\mathbf{x}) = \frac{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})p(\mathbf{z})}{p_{\boldsymbol{\theta}}(\mathbf{x})}$ by Bayes' rule, we have

$$\begin{aligned} D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z}|\mathbf{x})) &= \int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \log \frac{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})p_{\boldsymbol{\theta}}(\mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})p(\mathbf{z})} \, \mathrm{d}\mathbf{z} \\ &= \int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \log p_{\boldsymbol{\theta}}(\mathbf{x}) \, \mathrm{d}\mathbf{z} + \int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \log \frac{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}{p(\mathbf{z})} \, \mathrm{d}\mathbf{z} \\ &\quad - \int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) \, \mathrm{d}\mathbf{z} \\ &= \log p_{\boldsymbol{\theta}}(\mathbf{x}) + D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})) - \mathbb{E}_{\mathbf{z}\sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})] \end{aligned}$$

Rearranging, we have

$$\begin{aligned} \log p_{\boldsymbol{\theta}}(\mathbf{x}) &= \mathbb{E}_{\mathbf{z}\sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})] - D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})) + D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z}|\mathbf{x})) \\ &\geq \mathbb{E}_{\mathbf{z}\sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})] - D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})), \end{aligned}$$

since the KL divergence is a non-negative quantity. □

A.2.2 Neural Statistician

A.2.2.1 Basic Model

Derivation. Using the same method as above, we consider the KL divergence between the approximate posterior and the prior

$$D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}, \mathbf{c}|\mathbf{D}) \parallel p(\mathbf{z}, \mathbf{c}|\mathbf{D})) = \int q_{\boldsymbol{\phi}}(\mathbf{z}, \mathbf{c}|\mathbf{D}) \log \frac{q_{\boldsymbol{\phi}}(\mathbf{z}, \mathbf{c}|\mathbf{D})}{p(\mathbf{z}, \mathbf{c}|\mathbf{D})} \, \mathrm{d}\mathbf{z} \, \mathrm{d}\mathbf{c}.$$

Substituting $p(\mathbf{z}, \mathbf{c}|\mathbf{D}) = \frac{p_{\boldsymbol{\theta}}(\mathbf{D}|\mathbf{z},\mathbf{c})p(\mathbf{z},\mathbf{c})}{p_{\boldsymbol{\theta}}(\mathbf{D})}$ by Bayes' rule, we have

$$\begin{aligned} D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}, \mathbf{c}|\mathbf{D}) \parallel p(\mathbf{z}, \mathbf{c}|\mathbf{D})) &= \int q_{\boldsymbol{\phi}}(\mathbf{z}, \mathbf{c}|\mathbf{D}) \log \frac{q_{\boldsymbol{\phi}}(\mathbf{z}, \mathbf{c}|\mathbf{D})p_{\boldsymbol{\theta}}(\mathbf{D})}{p(\mathbf{D}|\mathbf{z}, \mathbf{c})p(\mathbf{z}, \mathbf{c})} \, \mathrm{d}\mathbf{z} \, \mathrm{d}\mathbf{c} \\ &= \int q_{\boldsymbol{\phi}}(\mathbf{z}, \mathbf{c}|\mathbf{D}) \log p_{\boldsymbol{\theta}}(\mathbf{D}) \, \mathrm{d}\mathbf{z} \, \mathrm{d}\mathbf{c} \\ &\quad + \int q_{\boldsymbol{\phi}}(\mathbf{z}, \mathbf{c}|\mathbf{D}) \log \frac{q_{\boldsymbol{\phi}}(\mathbf{z}, \mathbf{c}|\mathbf{D})}{p(\mathbf{D}|\mathbf{z}, \mathbf{c})p(\mathbf{z}, \mathbf{c})} \, \mathrm{d}\mathbf{z} \, \mathrm{d}\mathbf{c} \\ &= \log p_{\boldsymbol{\theta}}(\mathbf{D}) + \int q_{\boldsymbol{\phi}}(\mathbf{z}, \mathbf{c}|\mathbf{D}) \log \frac{q_{\boldsymbol{\phi}}(\mathbf{z}, \mathbf{c}|\mathbf{D})}{p(\mathbf{D}|\mathbf{z}, \mathbf{c})p(\mathbf{z}, \mathbf{c})} \, \mathrm{d}\mathbf{z} \, \mathrm{d}\mathbf{c}. \end{aligned}$$

Factorizing $q_{\boldsymbol{\phi}}(\mathbf{z}, \mathbf{c}|\mathbf{D}) = q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{c}, \mathbf{D})q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{D})$ and $p(\mathbf{z}, \mathbf{c}) = p(\mathbf{z}|\mathbf{c})p(\mathbf{c})$ yields

$$
\begin{aligned}
D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}, \mathbf{c}|\mathbf{D}) \parallel p(\mathbf{z}, \mathbf{c}|\mathbf{D})) &= \log p_{\boldsymbol{\theta}}(\mathbf{D}) + \int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{c}, \mathbf{D})q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{D}) \log \frac{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{c}, \mathbf{D})q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{D})}{p(\mathbf{D}|\mathbf{z}, \mathbf{c})p(\mathbf{z}|\mathbf{c})p(\mathbf{c})} \, \mathrm{d}\mathbf{z} \, \mathrm{d}\mathbf{c} \\
&= \log p_{\boldsymbol{\theta}}(\mathbf{D}) + \int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{c}, \mathbf{D})q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{D}) \log \frac{q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{D})}{p(\mathbf{c})} \, \mathrm{d}\mathbf{z} \, \mathrm{d}\mathbf{c} \\
&\quad + \int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{c}, \mathbf{D})q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{D}) \log \frac{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{c}, \mathbf{D})}{p(\mathbf{z}|\mathbf{c})} \, \mathrm{d}\mathbf{z} \, \mathrm{d}\mathbf{c} - \int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{c}, \mathbf{D})q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{D}) \log p(\mathbf{D}|\mathbf{z}, \mathbf{c}) \, \mathrm{d}\mathbf{z} \, \mathrm{d}\mathbf{c} \\
&= \log p_{\boldsymbol{\theta}}(\mathbf{D}) + \int q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{D}) \log \frac{q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{D})}{p(\mathbf{c})} \, \mathrm{d}\mathbf{c} \\
&\quad + \mathbb{E}_{\mathbf{c} \sim q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{D})} \left[\int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{c}, \mathbf{D}) \log \frac{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{c}, \mathbf{D})}{p(\mathbf{z}|\mathbf{c})} \, \mathrm{d}\mathbf{z} \right] - \mathbb{E}_{\mathbf{c} \sim q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{D})}[\mathbb{E}_{\mathbf{z} \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{c}, \mathbf{D})}[\log p(\mathbf{D}|\mathbf{z}, \mathbf{c})]] \\
&= \log p_{\boldsymbol{\theta}}(\mathbf{D}) + D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{D}) \parallel p(\mathbf{c})) + \mathbb{E}_{\mathbf{c} \sim q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{D})} \left[D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{c}, \mathbf{D}) \parallel p(\mathbf{z}|\mathbf{c})) \right] \\
&\quad - \mathbb{E}_{\mathbf{c} \sim q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{D})}[\mathbb{E}_{\mathbf{z} \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{c}, \mathbf{D})}[\log p(\mathbf{D}|\mathbf{z}, \mathbf{c})]].
\end{aligned}
$$

It remains to simplify the KL divergence terms and the log likelihood, by decomposing $\mathbf{D}$.

Rewriting the log likelihood is straightforward.

$$
\begin{aligned}
\log p(\mathbf{D}|\mathbf{z}, \mathbf{c}) &= \log \left(\prod_{n=1}^{N} p(\mathbf{x}^{(n)}|\mathbf{z}^{(n)}, \mathbf{c}) \right) \\
&= \sum_{n=1}^{N} \log p(\mathbf{x}^{(n)}|\mathbf{z}^{(n)}, \mathbf{c}).
\end{aligned}
$$

Moving to the KL divergence terms for the $\mathbf{z}$ variables,

$$
\begin{aligned}
D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{c}, \mathbf{D}) \parallel p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{c})) &= \int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{c}, \mathbf{D}) \log \frac{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{c}, \mathbf{D})}{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{c})} \, \mathrm{d}\mathbf{z} \\
&\quad \int \prod_{n=1}^{N} q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)}|\mathbf{c}, \mathbf{x}^{(n)}) \log \left(\prod_{n=1}^{N} \frac{q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)}|\mathbf{c}, \mathbf{x}^{(n)})}{p_{\boldsymbol{\theta}}(\mathbf{z}^{(n)}|\mathbf{c})} \right) \mathrm{d}\mathbf{z} \\
&= \int \prod_{n=1}^{N} q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)}|\mathbf{c}, \mathbf{x}^{(n)}) \sum_{n=1}^{N} \log \frac{q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)}|\mathbf{c}, \mathbf{x}^{(n)})}{p_{\boldsymbol{\theta}}(\mathbf{z}^{(n)}|\mathbf{c})} \, \mathrm{d}\mathbf{z} \\
&= \sum_{n=1}^{N} \int \prod_{n=1}^{N} q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)}|\mathbf{c}, \mathbf{x}^{(n)}) \log \frac{q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)}|\mathbf{c}, \mathbf{x}^{(n)})}{p_{\boldsymbol{\theta}}(\mathbf{z}^{(n)}|\mathbf{c})} \, \mathrm{d}\mathbf{z} \\
&= \sum_{n=1}^{N} D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)}|\mathbf{c}, \mathbf{x}^{(n)}) \parallel p_{\boldsymbol{\theta}}(\mathbf{z}^{(n)}|\mathbf{c})).
\end{aligned}
$$

Putting it all together, rearranging, and again using the fact that the KL divergence is a non-negative quantity, we have

$$
\log p_{\boldsymbol{\theta}}(\mathbf{D}) \geq R_{\mathbf{D}} - (L_{\mathbf{D}} + C_{\mathbf{D}}),
$$

where

$$R_{\mathbf{D}} = \mathbb{E}_{\mathbf{c}\sim q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{D})}\left[\sum_{n=1}^{N} \mathbb{E}_{\mathbf{z}^{(n)}\sim q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)}|\mathbf{c},\mathbf{x}^{(n)})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(n)}|\mathbf{z}^{(n)},\mathbf{c})]\right],$$
$$L_{\mathbf{D}} = \mathbb{E}_{\mathbf{c}\sim q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{D})}\left[\sum_{n=1}^{N} D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)}|\mathbf{c},\mathbf{x}^{(n)}) \parallel p_{\boldsymbol{\theta}}(\mathbf{z}^{(n)}|\mathbf{c}))\right],$$
$$C_{\mathbf{D}} = D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{D}) \parallel p(\mathbf{c})).$$

□

A.2.2.2 Full Model

Derivation. Having laid the groundwork with the basic model, the full model now just requires modification of the KL divergence terms for the hierarchical $\mathbf{z}$ variables. Effectively, only $L_{\mathbf{D}}$ changes in the above three-term lower bound. To this end, consider the factorization of the prior on $\mathbf{z}^{(n)}$,

$$p_{\boldsymbol{\theta}}(\mathbf{z}^{(n)}|\mathbf{c}) = p_{\boldsymbol{\theta}}(\mathbf{z}^{(n)(1)}|\mathbf{c}) \prod_{l=2}^{L} p_{\boldsymbol{\theta}}(\mathbf{z}^{(n)(l)}|\mathbf{z}^{(n)(l-1)},\mathbf{c}),$$

and analagously, the factorization of the approximate posterior,

$$q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)}|\mathbf{c},\mathbf{x}^{(n)}) = q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)(1)}|\mathbf{c},\mathbf{x}^{(n)}) \prod_{l=2}^{L} q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)(l)}|\mathbf{z}^{(n)(l-1)},\mathbf{c},\mathbf{x}^{(n)}).$$

Plugging these into the KL divergence term, we have

$$\begin{aligned}
D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)}|\mathbf{c},\mathbf{x}^{(n)}) \parallel p_{\boldsymbol{\theta}}(\mathbf{z}^{(n)}|\mathbf{c})) &= \int q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)}|\mathbf{c},\mathbf{x}^{(n)}) \log \frac{q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)}|\mathbf{c},\mathbf{x}^{(n)})}{p_{\boldsymbol{\theta}}(\mathbf{z}^{(n)}|\mathbf{c})} \,\mathrm{d}\mathbf{z}^{(n)} \\
&= \int q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)(1)}|\mathbf{c},\mathbf{x}^{(n)}) \log \frac{q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)(1)}|\mathbf{c},\mathbf{x}^{(n)})}{p_{\boldsymbol{\theta}}(\mathbf{z}^{(n)(1)}|\mathbf{c})} \,\mathrm{d}\mathbf{z}^{(n)(1)} \\
&\quad + \sum_{l=2}^{L} \int q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)(l)}|\mathbf{z}^{(n)(l-1)},\mathbf{c},\mathbf{x}^{(n)}) \log \frac{q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)(l)}|\mathbf{z}^{(n)(l-1)},\mathbf{c},\mathbf{x}^{(n)})}{p_{\boldsymbol{\theta}}(\mathbf{z}^{(n)(l)}|\mathbf{z}^{(n)(l-1)},\mathbf{c})} \,\mathrm{d}\mathbf{z}^{(n)(l)} \\
&= D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)(1)}|\mathbf{c},\mathbf{x}^{(n)}) \parallel p_{\boldsymbol{\theta}}(\mathbf{z}^{(n)(1)}|\mathbf{c})) \\
&\quad + \sum_{l=2}^{L} D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)(l)}|\mathbf{z}^{(n)(l-1)},\mathbf{c},\mathbf{x}^{(n)}) \parallel p_{\boldsymbol{\theta}}(\mathbf{z}^{(n)(l)}|\mathbf{z}^{(n)(l-1)},\mathbf{c})).
\end{aligned}$$

Thus

$$\log p_{\boldsymbol{\theta}}(\mathbf{D}) \geq R_{\mathbf{D}} - (L_{\mathbf{D}} + C_{\mathbf{D}}),$$

where

$$R_{\mathbf{D}} = \mathbb{E}_{\mathbf{c}\sim q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{D})}\left[\sum_{n=1}^{N}\mathbb{E}_{\mathbf{z}^{(n)}\sim q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)}|\mathbf{c},\mathbf{x}^{(n)})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(n)}|\mathbf{z}^{(n)},\mathbf{c})]\right],$$

$$L_{\mathbf{D}} = \mathbb{E}_{(\mathbf{z}^{(n)},\mathbf{c})\sim q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)},\mathbf{c}|\mathbf{D})}\left[\sum_{n=1}^{N} D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)(1)}|\mathbf{c},\mathbf{x}^{(n)}) \parallel p_{\boldsymbol{\theta}}(\mathbf{z}^{(n)(1)}|\mathbf{c})) + \sum_{n=1}^{N}\sum_{l=2}^{L} D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)(l)}|\mathbf{z}^{(n)(l-1)}\mathbf{c},\mathbf{x}^{(n)}) \parallel p_{\boldsymbol{\theta}}(\mathbf{z}^{(n)(l)}|\mathbf{z}^{(n)(l-1)}\mathbf{c}))\right],$$

$$C_{\mathbf{D}} = D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{D}) \parallel p(\mathbf{c})).$$

□

A.2.3 Context-Aware Learner

The derivation of the lower bound for the context-aware learner follows the same structure we have seen in the previous two derivations. The additional latent variables just mean extra-care is needed with book-keeping. For this reason, we are not as explicit with each step, but the direction should be clear from the previous details.

Derivation. Once more, consider the KL divergence between the approximate and true posteriors over latent variables.

$$D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z},\mathbf{c},\mathbf{a}|\mathbf{D}) \parallel p_{\boldsymbol{\theta}}(\mathbf{z},\mathbf{c},\mathbf{a}|\mathbf{D})) = \sum_{\mathbf{a}}\int q_{\boldsymbol{\phi}}(\mathbf{z},\mathbf{c},\mathbf{a}|\mathbf{D})\log\frac{q_{\boldsymbol{\phi}}(\mathbf{z},\mathbf{c},\mathbf{a}|\mathbf{D})}{p_{\boldsymbol{\theta}}(\mathbf{z},\mathbf{c},\mathbf{a}|\mathbf{D})}\,\mathrm{d}\mathbf{z}\,\mathrm{d}\mathbf{c}.$$

As was the case in previous derivations, we can use Bayes' rule to rewrite the true posterior in terms of the marginal, likelihood, and prior on latent variables, then use this to massage the KL divergence into a more amenable form.

$$D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z},\mathbf{c},\mathbf{a}|\mathbf{D}) \parallel p_{\boldsymbol{\theta}}(\mathbf{z},\mathbf{c},\mathbf{a}|\mathbf{D})) = \log p(\mathbf{D}) + D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z},\mathbf{c},\mathbf{a}|\mathbf{D}) \parallel p_{\boldsymbol{\theta}}(\mathbf{z},\mathbf{c},\mathbf{a})) - \mathbb{E}_{(\mathbf{z},\mathbf{c})\sim q_{\boldsymbol{\phi}}(\mathbf{z},\mathbf{c}|\mathbf{D})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{D}|\mathbf{z},\mathbf{c})\right].$$

The latter two terms in the expression on the right hand side are the KL divergence between the approximate posterior and prior over latent variables, and the reconstruction term in the from of an expectation over the log likelihood, just as we have seen before. It remains to write these terms in their most granular forms.

We begin with the likelihood. We have

$$\begin{aligned}\log p_{\boldsymbol{\theta}}(\mathbf{D}|\mathbf{z},\mathbf{c}) &= \log\prod_{n=1}^{N} p_{\boldsymbol{\theta}}(\mathbf{x}^{(n)}|\mathbf{z}^{(n)},\mathbf{c}^{(n)}) \\ &= \sum_{n=1}^{N}\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(n)}|\mathbf{z}^{(n)},\mathbf{c}^{(n)}),\end{aligned}$$

and we can plug this directly into the expectation.

Next we decompose the KL divergence between the approximate posterior and prior. For the approximate posterior, we choose a factorization analogous to that specified for the generative model, meaning we are left with three KL divergence terms.

1. $D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{a}|\mathbf{D}) \parallel p(\mathbf{a}))$

2. $\mathbb{E}_{\mathbf{a}\sim q_{\boldsymbol{\phi}}(\mathbf{a}|\mathbf{D})}\left[D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{a},\mathbf{D}) \parallel p_{\boldsymbol{\theta}}(\mathbf{c}|\mathbf{a}))\right]$

3. $\mathbb{E}_{\mathbf{c}\sim q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{a},\mathbf{D})}\left[D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{c},\mathbf{D}) \parallel p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{c}))\right]$

We need only simplify terms 2 and 3. Beginning with term 2, we can split the context divergence into two parts, one corresponding to constant contexts, and the other to varying contexts.

$$\begin{aligned} D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{a},\mathbf{D}) \parallel p_{\boldsymbol{\theta}}(\mathbf{c}|\mathbf{a})) &= \sum_{k\in A_c} D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{c}_{(k)}|\mathbf{a},\mathbf{D}) \parallel p_{\boldsymbol{\theta}}(\mathbf{c}_{(k)}|\mathbf{a})) \\ &\quad + \sum_{k\in A_v} D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{c}_{(k)}|\mathbf{c}_{A_c},\mathbf{a},\mathbf{D}) \parallel p_{\boldsymbol{\theta}}(\mathbf{c}_{(k)}|\mathbf{c}_{A_c},\mathbf{a})) \\ &= \sum_{k\in A_c} D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{c}^{(1)}_{(k)}|\mathbf{a},\mathbf{D}) \parallel p_{\boldsymbol{\theta}}(\mathbf{c}^{(1)}_{(k)}|\mathbf{a})) \\ &\quad + \sum_{n=1}^{N}\sum_{k\in A_v} D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{c}^{(n)}_{(k)}|\mathbf{c}_{A_c},\mathbf{a},\mathbf{x}^n) \parallel p_{\boldsymbol{\theta}}(\mathbf{c}^{(n)}_{(k)}|\mathbf{c}_{A_c},\mathbf{a})). \end{aligned}$$

Here we've used the fact that for the constant divergence terms, we can just take the sampled value for the first data point, since it is copied across the entire set.

Finally, we have

$$D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{c},\mathbf{D}) \parallel p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{c})) = \sum_{n=1}^{N} D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)}|\mathbf{c}^{(n)},\mathbf{x}^{(n)}) \parallel p_{\boldsymbol{\theta}}(\mathbf{z}^{(n)}|\mathbf{c}^{(n)})).$$

Thus, the variational lower bound is given by

$$\mathcal{L}_{\mathbf{D}}(\boldsymbol{\theta},\boldsymbol{\phi}) = R_{\mathbf{D}} - (L_{\mathbf{D}} + C_{v\mathbf{D}} + C_{c\mathbf{D}} + A_{\mathbf{D}}),$$

where

$$R_{\mathbf{D}} = \mathbb{E}_{\mathbf{z}^{(n)}\sim q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)}|\mathbf{c}^{(n)},\mathbf{x}^{(n)})}\left[\mathbb{E}_{\mathbf{c}^{(n)}\sim q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{a},\mathbf{D})}\left[\sum_{n=1}^{N}\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(n)}|\mathbf{z}^{(n)},\mathbf{c}^{(n)})\right]\right]$$

$$L_{\mathbf{D}} = \mathbb{E}_{\mathbf{c}^{(n)}\sim q_{\boldsymbol{\phi}}(\mathbf{c}^{(n)}|\mathbf{a},\mathbf{x}^{(n)})}\left[\sum_{n=1}^{N} D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}^{(n)}|\mathbf{c}^{(n)},\mathbf{x}^{(n)}) \parallel p_{\boldsymbol{\theta}}(\mathbf{z}^{(n)}|\mathbf{c}^{(n)}))\right]$$

$$C_{v\mathbf{D}} = \mathbb{E}_{(\mathbf{c}_{A_c},\mathbf{a})\sim q_{\boldsymbol{\phi}}(\mathbf{c},\mathbf{a}|\mathbf{D})}\left[\sum_{n=1}^{N}\sum_{k\in A_v} D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{c}_{(k)}^{(n)}|\mathbf{c}_{A_c},\mathbf{a},\mathbf{x}^{(n)}) \parallel p_{\boldsymbol{\theta}}(\mathbf{c}_{(k)}^{(n)}|\mathbf{c}_{A_c},\mathbf{a}))\right]$$

$$C_{c\mathbf{D}} = \mathbb{E}_{\mathbf{a}\sim q_{\boldsymbol{\phi}}(\mathbf{a}|\mathbf{D})}\left[\sum_{k\in A_c} D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{c}_{(k)}^{(1)}|\mathbf{a},\mathbf{D}) \parallel p_{\boldsymbol{\theta}}(\mathbf{c}_{(k)}^{(1)}|\mathbf{a}))\right]$$

$$A_{\mathbf{D}} = D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{a}|\mathbf{D}) \parallel p(\mathbf{a}))$$

□

A.3 KL Divergence for multivariate Gaussians

The derivation of the KL Divergence between two multivariate Gaussians is presented here since it is central to the lower bound computation for the models presented. As mentioned, the approximate posterior and prior are often chosen so that this term can be computed analytically. In particular, Gaussian distributions are commonly used for both.

For further details on the linear algebra in this section, we refer the reader to [Petersen et al., 2008].

Let $q(\mathbf{x}) = \mathcal{N}(\mathbf{x};\boldsymbol{\mu}_1,\Sigma_1)$, $p(\mathbf{x}) = \mathcal{N}(\mathbf{x};\boldsymbol{\mu}_2,\Sigma_2)$, with $\mathbf{x}\in\mathbb{R}^n$.
Claim.

$$D_{KL}(q(\mathbf{x}) \parallel p(\mathbf{x})) = \frac{1}{2}\left[(\boldsymbol{\mu}_1-\boldsymbol{\mu}_2)^\top\Sigma_2^{-1}(\boldsymbol{\mu}_1-\boldsymbol{\mu}_2) + \mathrm{Tr}\left(\Sigma_2^{-1}\Sigma_1\right) - n + \log\frac{|\Sigma_2|}{|\Sigma_1|}\right]$$

The following lemma is useful for the proof of the claim.
Lemma. *If* $p(\mathbf{x}) = \mathcal{N}(\mathbf{x};\boldsymbol{\mu},\Sigma)$, *then*

$$\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[\mathbf{x}^\top A\mathbf{x}\right] = \boldsymbol{\mu}^\top A\boldsymbol{\mu} + \mathrm{Tr}[A\Sigma]$$

Proof of lemma.

$$\begin{aligned}
\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[\mathbf{x}^\top A\mathbf{x}\right] &= \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[\sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}\mathbf{x}_i\mathbf{x}_j\right] \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}[\mathbf{x}_i\mathbf{x}_j] \quad \text{(linearity of expectation)} \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}[\Sigma_{ij} + \boldsymbol{\mu}_i\boldsymbol{\mu}_j] \quad \text{(definition of covariance)} \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}\Sigma_{ji} + \sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}\boldsymbol{\mu}_i\boldsymbol{\mu}_j \quad \text{(symmetry of covariance)} \\
&= \sum_{i=1}^{n}[A\Sigma]_{ii} + \boldsymbol{\mu}^\top A\boldsymbol{\mu} \\
&= \mathrm{Tr}[A\Sigma] + \boldsymbol{\mu}^\top A\boldsymbol{\mu} \quad \text{(definition of trace).}
\end{aligned}$$

□

A corollary of this result is that for any constant vector $\mathbf{c}$, we have

$$\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[(\mathbf{x}-\mathbf{c})^\top A(\mathbf{x}-\mathbf{c})\right] = (\boldsymbol{\mu}-\mathbf{c})^\top A(\boldsymbol{\mu}-\mathbf{c}) + \mathrm{Tr}[A\Sigma].$$

Proof of claim. We first rewrite the KL Divergence as an expectation.

$$\begin{aligned}
D_{KL}(q(\mathbf{x}) \parallel p(\mathbf{x})) &= \int q(\mathbf{x}) \log \frac{q(\mathbf{x})}{p(\mathbf{x})}\,\mathrm{d}\mathbf{x} \\
&= \int q(\mathbf{x})\left[\log q(\mathbf{x}) - \log p(\mathbf{x})\right]\mathrm{d}\mathbf{x} \\
&= \mathbb{E}_{\mathbf{x}\sim q(\mathbf{x})}\left[\log q(\mathbf{x}) - \log p(\mathbf{x})\right].
\end{aligned}$$

The logarithm of a multivariate Gaussian density is given by

$$\begin{aligned}
\log\left(\mathcal{N}(\mathbf{x};\boldsymbol{\mu},\Sigma)\right) &= \log\left(\frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}}\exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)\right) \\
&= -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}) + \log\left(\frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}}\right) \\
&= -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}) - \frac{n}{2}\log 2\pi - \frac{1}{2}\log|\Sigma|.
\end{aligned}$$

Thus

$$\log q(\mathbf{x}) - \log p(\mathbf{x}) = \frac{1}{2}\left[(\mathbf{x}-\boldsymbol{\mu}_2)^\top\Sigma_2^{-1}(\mathbf{x}-\boldsymbol{\mu}_2) - (\mathbf{x}-\boldsymbol{\mu}_1)^\top\Sigma_1^{-1}(\mathbf{x}-\boldsymbol{\mu}_1) + \log\frac{|\Sigma_2|}{|\Sigma_1|}\right],$$

and

$$\begin{aligned}
\mathbb{E}_{\mathbf{x}\sim q(\mathbf{x})}\left[\log q(\mathbf{x}) - \log p(\mathbf{x})\right] = \frac{1}{2}\Big[&\mathbb{E}_{\mathbf{x}\sim q(\mathbf{x})}\left[(\mathbf{x}-\boldsymbol{\mu}_2)^\top\Sigma_2^{-1}(\mathbf{x}-\boldsymbol{\mu}_2)\right] \\
&-\mathbb{E}_{\mathbf{x}\sim q(\mathbf{x})}\left[(\mathbf{x}-\boldsymbol{\mu}_1)^\top\Sigma_1^{-1}(\mathbf{x}-\boldsymbol{\mu}_1)\right] + \mathbb{E}_{\mathbf{x}\sim q(\mathbf{x})}\left[\log\frac{|\Sigma_2|}{|\Sigma_1|}\right]\Big].
\end{aligned}$$

Note $\mathbb{E}_{\mathbf{x}\sim q(\mathbf{x})}\left[\log \frac{|\Sigma_2|}{|\Sigma_1|}\right] = \log \frac{|\Sigma_2|}{|\Sigma_1|}$, since this quantity is constant. Then we can make use of the lemma presented above (in particular its corollary) to simplify the other two terms in the expression.

First we have

$$\mathbb{E}_{\mathbf{x}\sim q(\mathbf{x})}\left[(\mathbf{x}-\boldsymbol{\mu}_2)^\top \Sigma_2^{-1}(\mathbf{x}-\boldsymbol{\mu}_2)\right] = (\boldsymbol{\mu}_1-\boldsymbol{\mu}_2)^\top \Sigma_2^{-1}(\boldsymbol{\mu}_1-\boldsymbol{\mu}_2) + \mathrm{Tr}[\Sigma_1\Sigma_2^{-1}].$$

Then

$$\begin{aligned}\mathbb{E}_{\mathbf{x}\sim q(\mathbf{x})}\left[(\mathbf{x}-\boldsymbol{\mu}_1)^\top \Sigma_1^{-1}(\mathbf{x}-\boldsymbol{\mu}_1)\right] &= (\boldsymbol{\mu}_1-\boldsymbol{\mu}_1)^\top \Sigma_1^{-1}(\boldsymbol{\mu}_1-\boldsymbol{\mu}_1) + \mathrm{Tr}[\Sigma_1\Sigma_1^{-1}] \\ &= \mathrm{Tr}[\mathbb{I}_{n\times n}] \\ &= n.\end{aligned}$$

Thus the final result is

$$\frac{1}{2}\left[D_{KL}(q(\mathbf{x}) \parallel p(\mathbf{x})) = (\boldsymbol{\mu}_1-\boldsymbol{\mu}_2)^\top \Sigma_2^{-1}(\boldsymbol{\mu}_1-\boldsymbol{\mu}_2) + \mathrm{Tr}[\Sigma_1\Sigma_2^{-1}] - n + \log\frac{|\Sigma_2|}{|\Sigma_1|}\right].$$

□

Computing this expression involves solving a linear system, as well as the calculation of determinants. In practice, two special cases are used in the models presented above, which simplify the computation significantly. In the following, $\mathrm{diag}(\mathbf{x})$ is used to denote a diagonal matrix with vector $\mathbf{x}$ along the diagonal.

Corollary (Two Diagonal Normals)**.** *For* $q(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_1, \mathrm{diag}(\boldsymbol{\sigma}_1^2))$, $p(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_2, \mathrm{diag}(\boldsymbol{\sigma}_2^2))$,

$$D_{KL}(q(\mathbf{x}) \parallel p(\mathbf{x})) = \frac{1}{2}\sum_{i=1}^{n}\left[\frac{1}{\boldsymbol{\sigma}_{2i}^2}\left[(\boldsymbol{\mu}_{1i}-\boldsymbol{\mu}_{2i})^2 + \boldsymbol{\sigma}_{1i}^2\right] - 1 + \log\frac{\boldsymbol{\sigma}_{2i}^2}{\boldsymbol{\sigma}_{1i}^2}\right].$$

Corollary (Diagonal Normal & Standard Normal)**.** *For* $q(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2))$, $p(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \mathbf{0}, \mathbb{I})$,

$$D_{KL}(q(\mathbf{x}) \parallel p(\mathbf{x})) = \frac{1}{2}\sum_{i=1}^{n}\left[\boldsymbol{\mu}_i^2 + \boldsymbol{\sigma}_i^2 - 1 - \log\boldsymbol{\sigma}_i^2\right].$$

Both results follow directly from the original claim.

A.4 KL Divergence for Gumbel-Softmax

For the Gumbel-Softmax distribution, we use the implied categorical distribution to calculate a KL divergence. That is, for $q(\mathbf{y}) = \mathrm{GS}(\mathbf{y}; \pi_1, \tau_1)$, $p(\mathbf{y}) = \mathrm{GS}(\mathbf{y}; \pi_2, \tau_2)$, where $\mathbf{y} \in \mathbb{R}^K$ is one-hot,

$$D_{KL}(q(\mathbf{y}) \parallel p(\mathbf{y})) = \sum_{i=1}^{K}\pi_{1i}\log\frac{\pi_{1i}}{\pi_{2i}}.$$

Appendix B

Miscellaneous

B.1 Nonlinearities

In this section we give a brief overview of the two nonlinearities used in this work.

B.1.1 Sigmoid

The sigmoid activation was commonly used at the outset of modern deep learning, but has been replaced over the past few years. It now serves effectively as a tool for constraining a particular quantity to the range $[0, 1]$. Its functional form is given by

$$f(x) = \frac{1}{1 + e^{-x}},$$

with derivative

$$f'(x) = f(x)(1 - f(x)).$$

Note that

$$\lim_{x \to -\infty} f(x) = 0 \qquad \text{and} \qquad \lim_{x \to \infty} f(x) = 1.$$

In fact, the sigmoid saturates very quickly in both limits, and this is the primary reason it has fallen out of favour in current deep learning. Intermediate activations often lie in these saturated ranges, meaning the gradient is zero, and effective learning is hindered

B.1.2 Recitifed Linear Unit

The rectified linear unit (ReLU) is activation predominantly used in this work, and has seen widespread use in the literature. Its functional form is given by

$$f(x) = \max(0, x),$$

with derivative

$$f'(x) = \begin{cases} 1 & x \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

This solves the problem of saturating activations, promoting desirable gradient behaviour, and also induces a degree of sparsity in the network, since half of the inputs to any given layer are now zero on average.