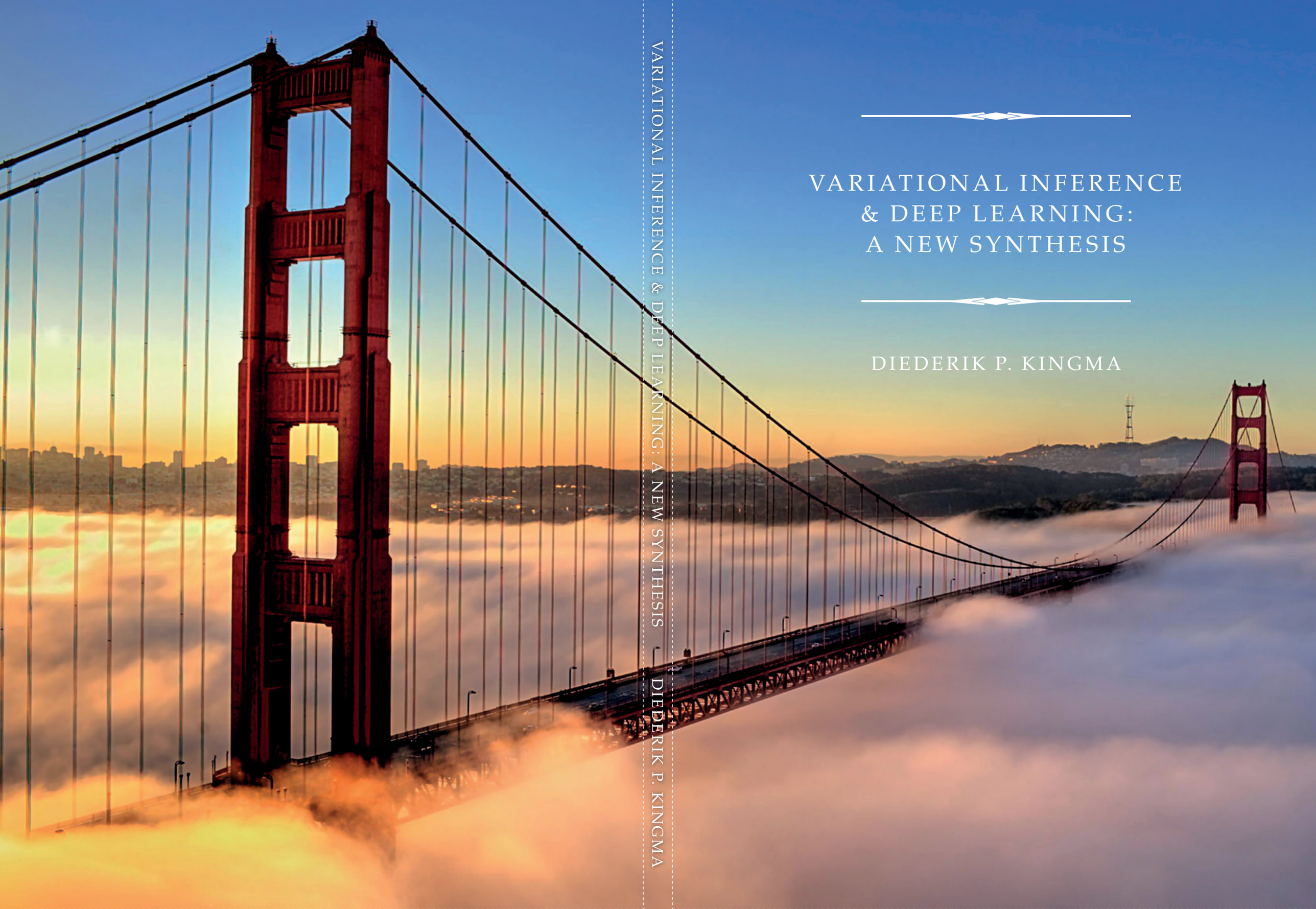
Variational inference & deep learning

Kingma, D.P.



VARIATIONAL INFERENCE & DEEP LEARNING: A NEW SYNTHESIS

DIEDERIK P. KINGMA

VARIATIONAL INFERENCE & DEEP LEARNING: A NEW SYNTHESIS DIEDERIK P. KINGMA

VARIATIONAL INFERENCE & DEEP LEARNING: A NEW SYNTHESIS

Academisch Proefschrift

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnicus
prof. dr. ir. K.I.J. Maex
ten overstaan van een
door het College voor Promoties ingestelde commissie,
in het openbaar te verdedigen in de Aula der Universiteit
op woensdag 25 oktober 2017, te 13 uur

door

Diederik Pieter Kingma

geboren te Utrecht

Promotiecommissie

Promotor:	
Prof. dr. M. Welling	Universiteit van Amsterdam
Copromotor:	
Dr. J.M. Mooij	Universiteit van Amsterdam
Overige leden:	
Prof. dr. D.M. Blei	Columbia University, New York
Prof. dr. Y. Bengio	Université de Montreal
Prof. dr. A. Hyvärinen	University College London
Prof. dr. M. de Rijke	Universiteit van Amsterdam
Dr. I.A. Titov	Universiteit van Amsterdam
Dr. E. Gavves	Universiteit van Amsterdam

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

Funding for this research was provided by the University of Amsterdam and a 2015 Google European PhD Fellowship in Deep Learning.

Printed by Ridderprint, The Netherlands.

ISBN: 978-94-6299-745-5

Summary

In this thesis, *Variational Inference and Deep Learning: A New Synthesis*, we propose novel solutions to the problems of variational (Bayesian) inference, generative modeling, representation learning, semi-supervised learning, and stochastic optimization.

- We propose **an efficient algorithm for variational inference** [Kingma and Welling, 2013] (chapter 2), suitable for solving high-dimensional inference problems with large models. The method uses first-order gradients of the model w.r.t. the latent variables and/or parameters; such gradients are efficient to compute using the backpropagation algorithm. This makes the method especially well-suited for inference and learning with deep neural networks.
- We propose **variational autoencoders** (VAEs) [Kingma and Welling, 2013] (chapter 2). The VAE framework combines a neural-network based inference model with a neural-network based generative model, and provides a simple method for joint optimization of both networks towards a bound on the log-likelihood of the parameters given the data. A doubly stochastic gradient descent procedure allows for scaling to very large datasets. We demonstrate the use of variational autoencoders for **generative modeling and representation learning**.
- We demonstrate how the VAE framework can be used to tackle the problem of **semi-supervised learning** [Kingma et al., 2014] (chapter 3), resulting in state-of-the-art results on standard semi-supervised image classification benchmarks at the time of publication.
- We propose **inverse autoregressive flows** [Kingma et al., 2016] (chapter 5), a flexible class of posterior distributions based on normalizing flows, allowing **inference of highly non-Gaussian posterior distributions** over high-dimensional latent spaces. We demonstrate how the method can be used to learn a VAE whose log-likelihood performance is comparable to autoregressive models, while allowing for orders of magnitude faster synthesis.
- We propose the **local reparameterization trick** (chapter 6) for further improving the efficiency of variational inference of a Gaussian posterior over model parameters [Kingma et al., 2015]. This method pro-

vides an additional (Bayesian) perspective of dropout, a popular regularization method; making using of this connection we propose **variational dropout**, which allows us to learn the dropout rate.

- We propose **Adam** [Kingma and Ba, 2015] (chapter 7), a method for stochastic gradient-based optimization based on adaptive moments.

CONTENTS

1

INTRODUCTION AND BACKGROUND

In this chapter we introduce and explain background material: probabilistic models, directed graphical models, the parameterization of conditional distributions with neural networks, learning in fully observed models, and deep latent-variable models (DLVMs). Please refer to section 9.1 for more information on notation.

1.1 ARTIFICIAL INTELLIGENCE

Modern humans have a natural ability to acquire new knowledge and skills, called *intelligence*. This ability was instrumental for our species' most cherished achievements, from hunting to farming to music and modern medicine and spacecraft. However, while human intelligence is impressive, it is bounded. On an evolutionary scale, we are not far removed from the great apes. While we have developed increasingly sophisticated tools to overcome our limitations, intellectually we are still to a large degree limited to our innate ability. How much more could we achieve if we were more intelligent? More specifically: what if we apply our intelligence to acquire knowledge about intelligence, and create tools that allow us to overcome our own cognitive limitations?

This last question is central to the field of computer science, and especially the field of artificial intelligence. In the rapidly developing subfield of *machine learning*, specifically, we set out to acquire new knowledge and skills to build machines that themselves can acquire new knowledge and skills. The aim of this work is further advancement of the field of artificial intelligence, and through this means, to increase the probability of a future that is bright.

1.2 PROBABILISTIC MODELS AND VARIATIONAL INFERENCE

In the field of machine learning, we are often interested in learning probabilistic models of various natural and artificial phenomena from data. Probabilistic models are mathematical descriptions of such phenomena. They are useful for prediction of unknowns in the future, and are useful for various forms of assisted or automated decision making. As such, models formalize the notion of knowledge and skill, and are the central constructs in the field of machine learning and AI.

Probabilistic models combine mathematical rigor with raw data; their power is increasingly apparent when combined with the strength of modern computational resources. The speed of computation and the amount of memory in modern consumer-grade computers start to rival those of a human brain. An important advantageous difference with the human brain, is that computers are easily programmable, allowing humans to quickly develop and debug novel programs, and share them with peers. Computers can also be wired together into large and high-bandwidth networks, allowing such networks to effectively function as a single computer with orders of magnitude more resources than singletons.

In probabilistic models, we assume some level of uncertainty over the values of variables contained in the model, and the degree and nature of uncertainty of the values of the variables, conditioned on the value of other variables, is explicitly specified in terms of (conditional) probability distributions. In the framework of Bayesian probabilistic inference, we choose a *prior distribution* over the unknown parameters or latent variables, which we update to a *posterior distribution* after seeing data. One method for computation of such a posterior is *variational inference*. In this work we propose new methods for efficient estimation of such posterior distributions.

We are interested in learning accurate probabilistic models of high-dimensional data. Perhaps the most complete form of probabilistic model are those that specify a joint distribution over all the variables of interest. Let's use $\mathbf{x}$ as the vector representing the set of all observed variables whose joint distribution we would like to model. It may consist of continuous variables, discrete variables, or a mix. Note that for notational simplicity and to avoid clutter, we use lower case bold (e.g. $\mathbf{x}$) to denote the underlying set of observed random variables: i.e. flattened and concatenated such that the set is represented as a single vector. See section 9.1 for more on notation.

We assume the observed variable $\mathbf{x}$ is a random sample from an *unknown underlying process*, whose true distribution $p^*(\mathbf{x})$ is unknown. We attempt to approximate this underlying process with a chosen model $p_{\boldsymbol{\theta}}(\mathbf{x})$, with parameters $\boldsymbol{\theta}$:

$$\mathbf{x} \sim p_{\boldsymbol{\theta}}(\mathbf{x}) \tag{1.1}$$

Learning is, most commonly, the process of searching for a value of the parameters $\boldsymbol{\theta}$ such that the probability distribution function given by the model, $p_{\boldsymbol{\theta}}(\mathbf{x})$, approximates the true distribution of the data, denoted by $p^*(\mathbf{x})$, such that for any observed $\mathbf{x}$:

$$p_{\boldsymbol{\theta}}(\mathbf{x}) \approx p^*(\mathbf{x}) \tag{1.2}$$

Naturally, we wish $p_{\boldsymbol{\theta}}(\mathbf{x})$ to be sufficiently **flexible** to be able to adapt to the data, such that we have a chance of obtaining a sufficiently accurate model. At the same time, we wish to be able to incorporate knowledge about the distribution of data into the model that is known a priori.

1.2.1 *Conditional Models*

Often, such as in case of classification or regression problems, we are not interested in learning an unconditional model $p_{\boldsymbol{\theta}}(\mathbf{x})$, but a conditional model $p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})$ that approximates the underlying conditional distribution $p^*(\mathbf{y}|\mathbf{x})$: a distribution over the values of variable $\mathbf{y}$, conditioned on variable $\mathbf{x}$. In this case, $\mathbf{x}$ is often called the *input* of the model. Like in the unconditional case, a model $p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})$ is chosen, and optimized to be close to the unknown underlying distribution, such that for any $\mathbf{x}$ and $\mathbf{y}$:

$$p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}) \approx p^*(\mathbf{y}|\mathbf{x}) \tag{1.3}$$

A relatively common and simple example of conditional modeling is image classification, where $\mathbf{x}$ is an image, and $\mathbf{y}$ is the image's class, as labeled by a human, which we wish to predict. In this case, $p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})$ is typically chosen to be a categorical distribution, whose parameters are computed from $\mathbf{x}$.

Conditional models become more difficult to learn when the predicted variables are very high-dimensional, such as images, video or sound. One example is the reverse of the image classification problem: prediction of a distribution over images, conditioned on the class label. Another example

with both high-dimensional input, and high-dimensional output, is depth prediction: in this case we would input an RGB image, and wish to predict a join distribution over estimates depths of each pixel of the image.

For clarity and notational brevity we will often assume unconditional modeling, but one should always keep in mind that the methods introduced in this work are typically equally applicable to conditional models. The data on which the model is conditioned, can be treated as inputs to the model, similar to the parameters of the model, with the obvious difference that one doesn't optimize over their value.

1.3 PARAMETERIZING CONDITIONAL DISTRIBUTIONS WITH NEURAL NETWORKS

In this work, we parameterize conditional distributions with neural networks [Goodfellow et al., 2016]. With this, we mean that neural networks are part of the function that computes the conditional probability density. A flexible way to parameterize such distributions is with neural networks. Neural networks are compositions as directed and typically differentiable computational units, that are somewhat analogous to neurons in biology. Such models can be efficiently optimized with stochastic gradient descent (SGD), as we will explain. For generality, clarity and notational brevity, we will denote a deep neural network simply as a vector function: $\text{NeuralNet}(.)$.

An early successful example of neural network, was the classification of hand-written digits by LeCun et al. [1998]; at the time of writing, deep learning has been shown to work well for a large variety of classification and regression problems, as summarized in [LeCun et al., 2015, Goodfellow et al., 2016]. In image classification, neural networks parameterize a categorical distribution $p_{\boldsymbol{\theta}}(y|\mathbf{x})$ over a class label y, conditioned on an image $\mathbf{x}$.

$$\mathbf{p} = \text{NeuralNet}(\mathbf{x}) \tag{1.4}$$

$$p_{\boldsymbol{\theta}}(y|\mathbf{x}) = \text{Categorical}(y; \mathbf{p}) \tag{1.5}$$

where the last operation of the neural net is a softmax() function such that $\sum_i p_i = 1$.

1.4 DIRECTED GRAPHICAL MODELS AND NEURAL NETWORKS

In this work we work with *directed (probabilistic) graphical models*, whose conditional distributions are parameterized by neural networks. Directed graphical models are a type of probabilistic models where all the variables are topologically organized into a directed acyclic graph. The joint distribution over the variables of such models factorizes as a product of prior and conditional distributions:

$$p_{\boldsymbol{\theta}}(\mathbf{x}_1, ..., \mathbf{x}_M) = \prod_{j=1}^{M} p_{\boldsymbol{\theta}}(\mathbf{x}_j | Pa(\mathbf{x}_j)) \tag{1.6}$$

where $Pa(\mathbf{x}_j)$ is the set of parent variables of node j in the directed graph. For non-root-nodes, a factor is a conditional distribution, where we condition on the parents. For root nodes, the set of parents is the empty set, such that the distribution is unconditional. Such models are called *directed graphical models*, or *Bayesian networks*.

Traditionally, each conditional probability distribution $p_{\boldsymbol{\theta}}(\mathbf{x}_j | Pa(\mathbf{x}_j))$ is parameterized as a lookup table or a linear model. A more flexible way to parameterize such distributions is with neural networks. When used to parameterize conditional distributions, neural networks take as input the parents of a variable in a directed graph, and produce the parameters $\boldsymbol{\eta}$ of a distribution over $\mathbf{x}$:

$$\boldsymbol{\eta} = \text{NeuralNet}(Pa(\mathbf{x})) \tag{1.7}$$

$$p_{\boldsymbol{\theta}}(\mathbf{x} | Pa(\mathbf{x})) = p_{\boldsymbol{\theta}}(\mathbf{x} | \boldsymbol{\eta}) \tag{1.8}$$

1.5 LEARNING IN FULLY OBSERVED MODELS WITH NEURAL NETS

If all variables in the directed graphical model are observed in the data, then we can compute and differentiate the log-probability of the data under the model, leading to relatively straightforward optimization.

1.5.1 *Dataset*

We often collect a dataset $\mathcal{D}$ consisting of $N \geq 1$ datapoints:

$$\mathcal{D} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(N)}\} \equiv \{\mathbf{x}^{(i)}\}_{i=1}^{N} \equiv \mathbf{x}^{(1:N)} \tag{1.9}$$

The datapoints are assumed to be independent samples from an unchanging underlying distribution. In other words, the dataset is assumed to consist of distinct, independent measurements from the same (unchanging) system. In this case, the observations $\{\mathbf{x}^{(i)}\}_{i=1}^{N}$ are said to be *i.i.d.*, for *independently and identically distributed*. Under the i.i.d. assumption, the probability of the datapoints given the parameters factorizes as a product of individual datapoint probabilities.

1.5.2 *Maximum Likelihood and Minibatch SGD*

The most common criterion for probabilistic models is *maximum log-likelihood* (ML). As we will explain, maximization of the log-likelihood criterion is equivalent to minimization of a Kullback-Leibler divergence between the data and model distributions.

Under the ML criterion, we attempt to find the parameters $\boldsymbol{\theta}$ that maximize the sum, or equivalently the average, of the log-probabilities assigned to the data by the model. With i.i.d. dataset $\mathcal{D}$ of size $N_{\mathcal{D}}$, the maximum likelihood criterion is:

$$\log p_{\boldsymbol{\theta}}(\mathcal{D}) = \sum_{\mathbf{x}\in\mathcal{D}} \log p_{\boldsymbol{\theta}}(\mathbf{x}) \tag{1.10}$$

Using calculus' chain rule and automatic differentiation tools, we can efficiently compute gradients of this objective, and use such gradient to iteratively hill-climb to a local optimum of the ML objective. If we compute such gradients using all datapoints, $\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathcal{D})$, then this is known as *batch* gradient descent. Computation of this derivative is, however, an expensive operation since it scales linearly with the dataset size $N_{\mathcal{D}}$.

A more efficient method for optimization is *stochastic gradient descent* (SGD) (section 9.3), which uses randomly drawn minibatches of data $\mathcal{M} \subset \mathcal{D}$ of size $N_{\mathcal{M}}$. With such minibatches we can form an unbiased estimator of the ML criterion:

$$\frac{1}{N_{\mathcal{D}}} \log p_{\boldsymbol{\theta}}(\mathcal{D}) \simeq \frac{1}{N_{\mathcal{M}}} \log p_{\boldsymbol{\theta}}(\mathcal{M}) = \frac{1}{N_{\mathcal{M}}} \sum_{\mathbf{x}\in\mathcal{M}} \log p_{\boldsymbol{\theta}}(\mathbf{x}) \tag{1.11}$$

The $\simeq$ symbol means that one of the two sides is an *unbiased estimator* of the other side. So one side (in this case the right-hand side) is a random variable due to some noise source, and the two sides are equal when averaged over

the noise distribution. The noise source, in this case, is the randomly drawn minibatch of data $\mathcal{M}$. The unbiased estimator $\log p_{\boldsymbol{\theta}}(\mathcal{M})$ is differentiable, yielding the unbiased stochastic gradients:

$$\frac{1}{N_{\mathcal{D}}}\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathcal{D}) \simeq \frac{1}{N_{\mathcal{M}}}\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathcal{M}) = \frac{1}{N_{\mathcal{M}}} \sum_{\mathbf{x}\in\mathcal{M}} \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{x}) \tag{1.12}$$

These gradients can be plugged into stochastic gradient-based optimizers; see section 9.3 for further discussion. In a nutshell, we can optimize the objective function by repeatedly taking small steps in the direction of the stochastic gradient.

1.5.3 *Bayesian inference*

From a Bayesian perspective, we can improve upon ML through *maximum a posteriori* (MAP) estimation (section section 9.2.1), or, going even further, inference of a full approximate posterior distribution over the parameters (see section 9.1.4).

1.6 LEARNING AND INFERENCE IN DEEP LATENT VARIABLE MODELS

1.6.1 *Latent Variables*

We can extend fully-observed directed models, discussed in the previous section, into directed models with *latent variables*. Latent variables are variables that are part of the model, but which we don't observe, and are therefore not part of the dataset. We typically use $\mathbf{z}$ to denote such latent variables. In case of unconditional modeling of observed variable $\mathbf{x}$, the directed graphical model would then represent a joint distribution $p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$ over both the observed variables $\mathbf{x}$ and the latent variables $\mathbf{z}$. The marginal distribution over the observed variables $p_{\boldsymbol{\theta}}(\mathbf{x})$, is given by:

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \int p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})\, d\mathbf{z} \tag{1.13}$$

This is also called the (single datapoint) *marginal likelihood* or the *model evidence*, when taking as a function of $\boldsymbol{\theta}$.

Such an implicit distribution over $\mathbf{x}$ can be quite flexible. If $\mathbf{z}$ is discrete and $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ is a Gaussian distribution, then $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ is a mixture-of-Gaussians distribution. For continuous $\mathbf{z}$ (which is generally more efficient

to work with due to the reparameterization trick), $p_{\theta}(\mathbf{x})$ can be seen as an infinite mixture, which are potentially more powerful than discrete mixtures. Such marginal distributions ares also called compound probability distributions.

1.6.2 *Deep Latent Variable Models*

We use the term *deep latent variable model* (DLVM) to denote a latent variable model $p_{\theta}(\mathbf{x}, \mathbf{z})$ whose distributions are parameterized by neural networks. Such a model can be conditioned on some context, like $p_{\theta}(\mathbf{x}, \mathbf{z}|\mathbf{y})$. DLVMs are especially useful when inference is computationally affordable, such as representation learning and artificial creativity

One important advantage of DLVMs, is that even when each factor (prior or conditional distribution) in the directed model is relatively simple (such as conditional Gaussian), the marginal distribution $p_{\theta}(\mathbf{x})$ can be very complex, i.e. contain almost arbitrary dependencies. This expressivity makes deep latent-variable models attractive for approximating complicated underlying distributions $p^*(\mathbf{x})$.

Perhaps the simplest, and most common, graphical model with latent variables is one that is specified as factorization with the following structure:

$$p_{\theta}(\mathbf{x}, \mathbf{z}) = p_{\theta}(\mathbf{z})p_{\theta}(\mathbf{x}|\mathbf{z}) \tag{1.14}$$

where $p_{\theta}(\mathbf{z})$ and/or $p_{\theta}(\mathbf{x}|\mathbf{z})$ are specified. The distribution $p(\mathbf{z})$ is often called the *prior distribution* over $\mathbf{z}$, since it is not conditioned on any observations.

1.6.3 *Example DLVM for multivariate Bernoulli data*

A simple example DLVM used in the original VAE publication [Kingma and Welling, 2013] is one for binary data $\mathbf{x}$, with a spherical Gaussian latent space, and a factorized Bernoulli observation model:

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; 0, \mathbf{I}) \tag{1.15}$$

$$\mathbf{p} = \text{DecoderNeuralNet}_{\boldsymbol{\theta}}(\mathbf{z}) \tag{1.16}$$

$$\log p(\mathbf{x}|\mathbf{z}) = \sum_{j=1}^{D} \log p(x_j|\mathbf{z}) = \sum_{j=1}^{D} \log \text{Bernoulli}(x_j; p_j) \tag{1.17}$$

$$= \sum_{j=1}^{D} x_j \log p_j + (1 - x_j) \log(1 - p_j) \tag{1.18}$$

where $\forall p_j \in \mathbf{p} : 0 \leq p_j \leq 1$ (e.g. implemented through a sigmoid nonlinearity as the last layer of the $\text{DecoderNeuralNet}_{\boldsymbol{\theta}}(.)$), where D is the dimensionality of $\mathbf{x}$, and $\text{Bernoulli}(.; p)$ is the probability mass function (PMF) of the Bernoulli distribution.

1.7 INTRACTABILITIES

The main difficulty of maximum likelihood learning in DLVMs is that the marginal probability of data under the model is typically intractable. This is due to the integral in equation (1.13) for computing the marginal likelihood (or model evidence), $p_{\boldsymbol{\theta}}(\mathbf{x}) = \int p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})\, d\mathbf{z}$, not having an analytic solution or efficient estimator. Due to this intractability, we cannot differentiate it w.r.t. its parameters and optimize it, as we can with fully observed models.

The intractability of $p_{\boldsymbol{\theta}}(\mathbf{x})$, is related to the intractability of the posterior distribution $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$. Note that the joint distribution $p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$ is efficient to compute, and that the densities are related through the basic identity:

$$p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) = \frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})}{p_{\boldsymbol{\theta}}(\mathbf{x})} \tag{1.19}$$

Since $p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$ *is* tractable to compute, a tractable marginal likelihood $p_{\boldsymbol{\theta}}(\mathbf{x})$ leads to a tractable posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$, and vice versa. Both are intractable in DLVMs.

Approximate inference techniques (see also section 9.2) allow us to approximate the posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ and the marginal likelihood $p_{\boldsymbol{\theta}}(\mathbf{x})$ in DLVMs.

Traditional inference methods are relatively expensive. Such methods, for example, often require a per-datapoint optimization loop, or yield bad posterior approximations. We would like to avoid such expensive procedures.

Likewise, the posterior over the parameters of (directed models parameterized with) neural networks, $p(\boldsymbol{\theta}|\mathcal{D})$, is generally intractable to compute exactly, and requires approximate inference techniques.

1.8 RESEARCH QUESTIONS AND CONTRIBUTIONS

We formulate the research questions and contributions in this work as follows.

Research question 1: *How can we perform efficient approximate posterior inference and efficient approximate maximum likelihood estimation in deep latent-variable models, in the presence of large datasets?*

In chapter 2 and [Kingma and Welling, 2013] we propose an efficient algorithm for variational inference based on a *reparameterization trick*, suitable for solving high-dimensional inference problems with large models. The method uses first-order gradients of the model w.r.t. the latent variables and/or parameters; such gradients are efficient to compute using the back-propagation algorithm. This makes the method especially well-suited for inference and learning in deep latent-variable models.

The proposed *variational autoencoder* (VAE) framework combines a neural-network based inference model with a neural-network based generative model, and we provide a simple method for joint optimization of both networks towards a bound on the log-likelihood of the parameters given the data. This doubly stochastic gradient descent procedure allows for scaling to very large datasets. We demonstrate the use of variational autoencoders for generative modeling and representation learning.

Research question 2: *Can we use the proposed VAE framework to improve upon state-of-the-art semi-supervised classification results?*

In chapter 3 and [Kingma et al., 2014] we demonstrate how the VAE framework can be used to tackle the problem of semi-supervised learning, resulting in state-of-the-art results on standard semi-supervised image classifica-

tion benchmarks at the time of publication.

The framework of normalizing flows [Rezende and Mohamed, 2015] provides an attractive approach for parameterizing flexible approximate posterior distributions in the VAE framework, but does not scale well to high-dimensional latent spaces. This leads us to the following question:

Research question 3: *Does there exist a practical normalizing flow that scales well to high-dimensional latent space?*

In chapter 5 and [Kingma et al., 2016] we propose *inverse autoregressive flows*, a flexible class of posterior distributions based on normalizing flows, allowing inference of highly non-Gaussian posterior distributions over high-dimensional latent spaces. We demonstrate how the method can be used to learn a VAE whose log-likelihood performance is comparable to autoregressive models, while allowing for orders of magnitude faster synthesis.

As explained in chapter 2, the proposed reparameterization-based inference method can be used to infer approximate posterior distributions over neural network parameters. A naïve implementation, however, is relatively inefficient.

Research question 4: *Can we improve upon the reparameterization-based gradient estimator by constructing a gradient estimator whose variance grows inversely proportional to the minibatch size, without sacrificing parallelizability?*

In chapter 6 and [Kingma et al., 2015], we propose a *local reparameterization trick* for further improving the efficiency of variational inference of a Gaussian posterior over model parameters, which leads to a gradient estimator whose variance shrinks proportional to the minibatch size. This method also provides an additional (Bayesian) perspective of dropout, a popular regularization method; making use of this connection we propose *variational dropout*, which allows us to learn the dropout rate.

Almost all experiments with neural networks require stochastic gradient-based optimization; improvements in such optimizers are worth investigating, since they can potentially translate to improvements in all such results. This leads us to our final research question:

Research question 5: *Can we improve upon existing stochastic gradient-based optimization methods?*

In chapter 7 and [Kingma and Ba, 2015] we propose *Adam*, a scalable method for stochastic gradient-based optimization based on adaptive moments. The method is simple to implement, has low computational overhead, and we demonstrate promising theoretical and empirical results.

2

VARIATIONAL AUTOENCODERS

2.1 INTRODUCTION

How can we perform efficient inference and learning in directed probabilistic models, in the presence of continuous latent variables with intractable posterior distributions, and large datasets? In this chapter, we introduce a stochastic variational inference and learning algorithm that scales to large datasets and, under some mild differentiability conditions, even works in the intractable case. Our contribution is two-fold. First, we show that a reparameterization of the ELBO yields a gradient estimator that can be straightforwardly optimized using standard stochastic gradient methods. Second, we show that for i.i.d. datasets with continuous latent variables per datapoint, posterior inference can be made especially efficient by fitting an approximate inference model (also called a recognition model) to the intractable posterior using the proposed gradient estimator. Theoretical advantages are reflected in experimental results.[1]

2.2 ENCODER OR APPROXIMATE POSTERIOR

Let $p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$ be a latent-variable model with observed variables $\mathbf{x}$ and latent variables $\mathbf{z}$. To turn a DLVMs intractable posterior inference and learning problems into tractable problems, we introduce a parametric *inference model* $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$. This model is also called an *encoder*. With $\boldsymbol{\phi}$ we indicate the parameters of this inference model, also called the *variational parameters*. We optimize the variational parameters $\boldsymbol{\phi}$ such that:

$$q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \approx p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) \tag{2.1}$$

[1] The main contributions in this chapter were first described in [Kingma and Welling, 2013].

As we will explain, this approximation to the posterior help us optimize the marginal likelihood.

Like a DLVM, the inference model can be (almost) any directed graphical model:

$$q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = q_{\boldsymbol{\phi}}(\mathbf{z}_1, ..., \mathbf{z}_M|\mathbf{x}) = \prod_{j=1}^{M} q_{\boldsymbol{\phi}}(\mathbf{z}_j|Pa(\mathbf{z}_j), \mathbf{x}) \tag{2.2}$$

And also similar to a DLVM, the distribution $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ can be parameterized using deep neural networks. In this case, the variational parameters $\boldsymbol{\phi}$ include the weights and biases of the neural network. For example:

$$(\boldsymbol{\mu}, \log \boldsymbol{\sigma}) = \text{EncoderNeuralNet}_{\boldsymbol{\phi}}(\mathbf{x}) \tag{2.3}$$

$$q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma})) \tag{2.4}$$

Typically, we use a single encoder neural network to perform posterior inference over all of the datapoints in our dataset. This can be contrasted to more traditional variational inference methods where the variational parameters are not shared, but instead separately and iteratively optimized per datapoint. The strategy used in VAEs of sharing variational parameters across datapoints is also called *amortized variational inference* [Gershman and Goodman, 2014]. With amortized inference we can avoid a per-datapoint optimization loop, and leverage the efficiency of SGD.

2.3 EVIDENCE LOWER BOUND (ELBO)

The optimization objective of the variational autoencoder, like in other variational methods, is the *evidence lower bound*, abbreviated as ELBO. An alternative term for this objective is *variational lower bound*. Typically, the ELBO is derived through Jensen's inequality. Here we will use an alternative derivation that avoids Jensen's inequality, providing greater insight about its tightness.

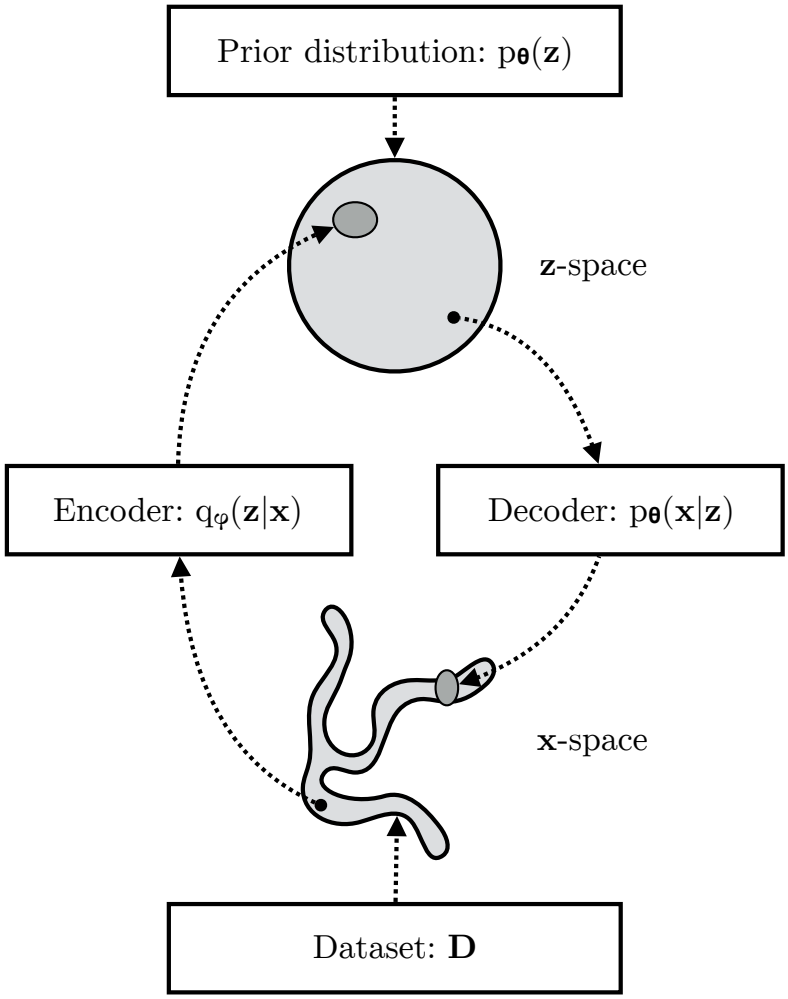


Figure 2.1: A VAE learns stochastic mappings between the observed $\mathbf{x}$-space, whose empirical distribution $q_{\mathcal{D}}(\mathbf{x})$ is typically complicated, and a latent $\mathbf{z}$-space, whose distribution can be relatively simple (such as spherical, as in this figure). The generative model learns a joint distribution $p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) = p_{\boldsymbol{\theta}}(\mathbf{z})p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$, factorized into a prior distribution over latent space, $p_{\boldsymbol{\theta}}(\mathbf{z})$, and a stochastic decoder $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$. The stochastic encoder $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$, also called *inference model*, approximates the true but intractable posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ of the generative model.

For any choice of inference model $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$, including the choice of variational parameters $\boldsymbol{\phi}$, we have:

$$\begin{aligned}
\log p_{\boldsymbol{\theta}}(\mathbf{x}) &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x})\right] && (2.5)\\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})}{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\right]\right] && (2.6)\\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\frac{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\right]\right] && (2.7)\\
&= \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\right]\right]}_{\substack{=\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x})\\ \text{(ELBO)}}} + \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\right]\right]}_{=D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}))} && (2.8)
\end{aligned}$$

The second term in eq. (2.8) is the Kullback-Leibler (KL) divergence between $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ and $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$, which is non-negative:

$$D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})) \geq 0 \tag{2.9}$$

and zero if, and only if, $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ equals the true posterior distribution.

The first term in eq. (2.8) is the *variational lower bound*, also called the *evidence lower bound* (ELBO):

$$\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\right] \tag{2.10}$$

Due to the non-negativity of the KL divergence, the ELBO is a lower bound on the log-likelihood of the data.

$$\begin{aligned}
\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) &= \log p_{\boldsymbol{\theta}}(\mathbf{x}) - D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})) && (2.11)\\
&\leq \log p_{\boldsymbol{\theta}}(\mathbf{x}) && (2.12)
\end{aligned}$$

So, interestingly, the KL divergence $D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}))$ determines two 'distances':

1. By definition, the KL divergence of the approximate posterior from the true posterior;
2. The gap between the ELBO $\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x})$ and the marginal likelihood $\log p_{\boldsymbol{\theta}}(\mathbf{x})$; this is also called the *tightness* of the bound. The better $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ approximates the true (posterior) distribution $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$, in terms of the KL divergence, the smaller the gap.

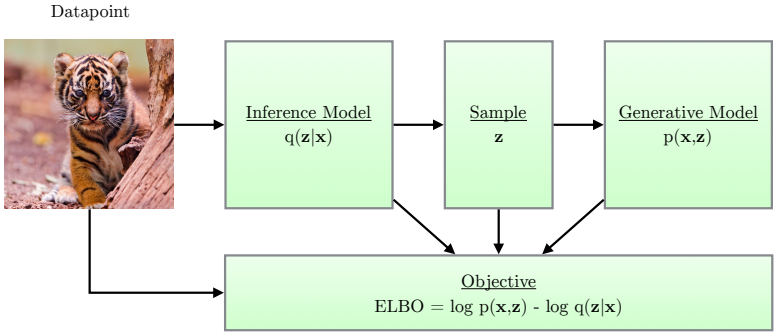


Figure 2.2: Simple schematic of computational flow in a variational autoencoder.

2.3.1 *A Double-Edged Sword*

By looking at equation 2.11, it can be understood that maximization of the ELBO $\mathcal{L}_{\theta,\phi}(\mathbf{x})$ w.r.t. the parameters θ and ϕ, will concurrently optimize the two things we care about:

1. It will approximately maximize the marginal likelihood $p_\theta(\mathbf{x})$. This means that our generative model will become better.
2. It will minimize the KL divergence of the approximation $q_\phi(\mathbf{z}|\mathbf{x})$ from the true posterior $p_\theta(\mathbf{z}|\mathbf{x})$, so $q_\phi(\mathbf{z}|\mathbf{x})$ becomes better.

2.4 STOCHASTIC GRADIENT-BASED OPTIMIZATION OF THE ELBO

An important property of the ELBO, is that it allows joint optimization w.r.t. all parameters (ϕ and θ) using stochastic gradient descent (SGD). We can start out with random initial values of ϕ and θ, and stochastically optimize their values until convergence.

Given a dataset with i.i.d. data, the ELBO objective is the sum (or average) of individual-datapoint ELBO's:

$$\mathcal{L}_{\theta,\phi}(\mathcal{D}) = \sum_{\mathbf{x}\in\mathcal{D}} \mathcal{L}_{\theta,\phi}(\mathbf{x}) \tag{2.13}$$

The individual-datapoint ELBO, and its gradient $\nabla_{\theta,\phi}\mathcal{L}_{\theta,\phi}(\mathbf{x})$ is, in general, intractable. However, good unbiased estimators $\tilde{\nabla}_{\theta,\phi}\mathcal{L}_{\theta,\phi}(\mathbf{x})$ exist, as we will show, such that we can still perform minibatch SGD.

Unbiased gradients of the ELBO w.r.t. the generative model parameters are simple to obtain:

$$\begin{aligned}
\nabla_{\theta}\mathcal{L}_{\theta,\phi}(\mathbf{x}) &= \nabla_{\theta}\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})}\left[\log p_{\theta}(\mathbf{x},\mathbf{z}) - \log q_{\phi}(\mathbf{z}|\mathbf{x})\right] && (2.14)\\
&= \mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})}\left[\nabla_{\theta}(\log p_{\theta}(\mathbf{x},\mathbf{z}) - \log q_{\phi}(\mathbf{z}|\mathbf{x}))\right] && (2.15)\\
&\simeq \nabla_{\theta}(\log p_{\theta}(\mathbf{x},\mathbf{z}) - \log q_{\phi}(\mathbf{z}|\mathbf{x})) && (2.16)\\
&= \nabla_{\theta}(\log p_{\theta}(\mathbf{x},\mathbf{z})) && (2.17)
\end{aligned}$$

The last line (eq. (2.17)) is a simple Monte Carlo estimator of the second line (eq. (2.15)), where $\mathbf{z}$ in (2.17), the last line, is a random sample from $q_{\phi}(\mathbf{z}|\mathbf{x})$.

Unbiased gradients w.r.t. the *variational* parameters ϕ are more difficult to obtain, since the ELBO's expectation is taken w.r.t. the distribution $q_{\phi}(\mathbf{z}|\mathbf{x})$, which is a function of ϕ. I.e., in general:

$$\begin{aligned}
\nabla_{\phi}\mathcal{L}_{\theta,\phi}(\mathbf{x}) &= \nabla_{\phi}\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})}\left[\log p_{\theta}(\mathbf{x},\mathbf{z}) - \log q_{\phi}(\mathbf{z}|\mathbf{x})\right] && (2.18)\\
&\neq \mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})}\left[\nabla_{\phi}(\log p_{\theta}(\mathbf{x},\mathbf{z}) - \log q_{\phi}(\mathbf{z}|\mathbf{x}))\right] && (2.19)
\end{aligned}$$

In case of continuous latent variables, we can use a reparameterization trick for computing unbiased estimated of $\nabla_{\theta,\phi}\mathcal{L}_{\theta,\phi}(\mathbf{x})$, as we will now discuss. This stochastic estimate allows us to optimize the ELBO using SGD; see algorithm 1. See section 2.10.1 for a discussion of variational methods for discrete latent variables.

2.5 REPARAMETERIZATION TRICK

For continuous latent variables and a differentiable encoder and generative model, the ELBO can be straightforwardly differentiated w.r.t. both ϕ and θ through a change of variables, also called the *reparameterization trick* (Kingma and Welling [2013] and Rezende et al. [2014]).

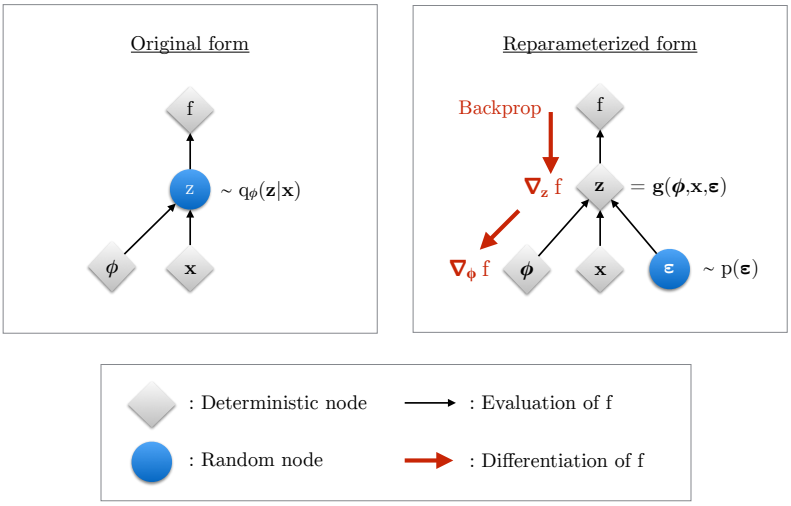


Figure 2.3: Illustration of the reparameterization trick. The variational parameters $\boldsymbol{\phi}$ affect the objective f through the random variable $\mathbf{z} \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$. We wish to compute gradients $\nabla_{\boldsymbol{\phi}} f$ to optimize the objective with SGD. In the original form (left), we cannot differentiate f w.r.t. $\boldsymbol{\phi}$, because we cannot directly backpropagate gradients through the random variable $\mathbf{z}$. We can 'externalize' the randomness in $\mathbf{z}$ by re-parameterizing the variable as a deterministic and differentiable function of $\boldsymbol{\phi}$, $\mathbf{x}$, and a newly introduced random variable $\boldsymbol{\epsilon}$. This allows us to 'backprop through $\mathbf{z}$', and compute gradients $\nabla_{\boldsymbol{\phi}} f$.

Algorithm 1: Stochastic optimization of the ELBO. Since noise originates from both the minibatch sampling and sampling of $p(\boldsymbol{\epsilon})$, this is a doubly stochastic optimization procedure. We also refer to this procedure as the *Auto-Encoding Variational Bayes* (AEVB) algorithm.

Data:
$\mathcal{D}$: Dataset
$q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$: Inference model
$p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$: Generative model
Result:
$\boldsymbol{\theta}, \boldsymbol{\phi}$: Learned parameters

$(\boldsymbol{\theta}, \boldsymbol{\phi}) \leftarrow$ Initialize parameters
while *SGD not converged* **do**
  $\mathcal{M} \sim \mathcal{D}$ (Random minibatch of data)
  $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$ (Random noise for every datapoint in $\mathcal{M}$)
  Compute $\tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathcal{M}, \boldsymbol{\epsilon})$ and its gradients $\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} \tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathcal{M}, \boldsymbol{\epsilon})$
  Update $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ using SGD optimizer
end

2.5.1 *Change of variables*

First, we express the random variable $\mathbf{z} \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ as some differentiable (and invertible) transformation of another random variable $\boldsymbol{\epsilon}$, given $\mathbf{z}$ and $\boldsymbol{\phi}$:

$$\mathbf{z} = \mathbf{g}(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \mathbf{x}) \tag{2.20}$$

where the distribution of random variable $\boldsymbol{\epsilon}$ is independent of $\mathbf{x}$ or $\boldsymbol{\phi}$.

2.5.2 *Gradient of expectation under change of variable*

Given such a change of variable, expectations can be rewritten in terms of $\boldsymbol{\epsilon}$,

$$\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[f(\mathbf{z})\right] = \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[f(\mathbf{z})\right] \tag{2.21}$$

where $\mathbf{z} = \mathbf{g}(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \mathbf{x})$. and the expectation and gradient operators become commutative, and we can form a simple Monte Carlo estimator:

$$\nabla_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} [f(\mathbf{z})] = \nabla_{\boldsymbol{\phi}} \mathbb{E}_{p(\boldsymbol{\epsilon})} [f(\mathbf{z})] \tag{2.22}$$

$$= \mathbb{E}_{p(\boldsymbol{\epsilon})} [\nabla_{\boldsymbol{\phi}} f(\mathbf{z})] \tag{2.23}$$

$$\simeq \nabla_{\boldsymbol{\phi}} f(\mathbf{z}) \tag{2.24}$$

where in the last line, $\mathbf{z} = \mathbf{g}(\boldsymbol{\phi}, \mathbf{x}, \boldsymbol{\epsilon})$ with random noise sample $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$. See figure 2.3 for an illustration and further clarification.

2.5.3 *Gradient of ELBO*

Under the reparameterization, we can replace an expectation w.r.t. $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ with one w.r.t. $p(\boldsymbol{\epsilon})$. The ELBO can be rewritten as:

$$\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\right] \tag{2.25}$$

$$= \mathbb{E}_{p(\boldsymbol{\epsilon})} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\right] \tag{2.26}$$

where $\mathbf{z} = g(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \mathbf{x})$.

As a result we can form a simple Monte Carlo estimator $\tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x})$ of the individual-datapoint ELBO where we use a single noise sample $\boldsymbol{\epsilon}$ from $p(\boldsymbol{\epsilon})$:

$$\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon}) \tag{2.27}$$

$$\mathbf{z} = \mathbf{g}(\boldsymbol{\phi}, \mathbf{x}, \boldsymbol{\epsilon}) \tag{2.28}$$

$$\tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) = \log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \tag{2.29}$$

This series of operations can be expressed as a symbolic graph in software like Tensorflow, and effortlessly differentiated w.r.t. the parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$. The resulting gradient $\nabla_{\boldsymbol{\phi}} \tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x})$ is used to optimize the ELBO using minibatch SGD. See algorithm 1. This algorithm was originally referred to as the Auto-Encoding Variational Bayes (AEVB) algorithm by Kingma and Welling [2013]. More generally, the reparameterized ELBO estimator was referred to as the Stochastic Gradient Variational Bayes (SGVB) estimator. This estimator can also be used to estimate a posterior over the model parameters, as explained in the appendix of Kingma and Welling [2013].

2.5.3.1 *Unbiasedness*

This gradient is an unbiased estimator of the exact single-datapoint ELBO gradient; when averaged over noise $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$, this gradient equals the single-datapoint ELBO gradient:

$$\begin{aligned}\mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}}\tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x};\boldsymbol{\epsilon})\right] &= \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}}(\log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}))\right] && (2.30)\\ &= \nabla_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\right]) && (2.31)\\ &= \nabla_{\boldsymbol{\theta},\boldsymbol{\phi}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) && (2.32)\end{aligned}$$

2.5.4 *Computation of* $\log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{z})$

Computation of the (estimator of) the ELBO requires computation of the density $\log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{z})$, given a value of $\mathbf{x}$, and given a value of $\mathbf{z}$ or equivalently $\boldsymbol{\epsilon}$. This log-density is a simple computation, as long as we choose the right transformation $\mathbf{g}()$.

Note that we typically know the density $p(\boldsymbol{\epsilon})$, since this is the density of the chosen noise distribution. As long as $\mathbf{g}(.)$ is an invertible function, the densities of $\boldsymbol{\epsilon}$ and $\mathbf{z}$ are related as:

$$\log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = \log p(\boldsymbol{\epsilon}) - \log d_{\boldsymbol{\phi}}(\mathbf{x},\boldsymbol{\epsilon}) \tag{2.33}$$

where the second term is the log of the absolute value of the determinant of the Jacobian matrix $(\partial\mathbf{z}/\partial\boldsymbol{\epsilon})$:

$$\log d_{\boldsymbol{\phi}}(\mathbf{x},\boldsymbol{\epsilon}) = \log\left|\det\left(\frac{\partial\mathbf{z}}{\partial\boldsymbol{\epsilon}}\right)\right| \tag{2.34}$$

We call this the log-determinant of the transformation from $\boldsymbol{\epsilon}$ to $\mathbf{z}$. We use the notation $\log d_{\boldsymbol{\phi}}(\mathbf{x},\boldsymbol{\epsilon})$ to make explicit that this log-determinant, similar to $\mathbf{g}()$, is a function of $\mathbf{x}$, $\boldsymbol{\epsilon}$ and $\boldsymbol{\phi}$. The Jacobian matrix contains all first derivatives of the transformation from $\boldsymbol{\epsilon}$ to $\mathbf{z}$:

$$\frac{\partial\mathbf{z}}{\partial\boldsymbol{\epsilon}} = \frac{\partial(z_1,...,z_k)}{\partial(\epsilon_1,...,\epsilon_k)} = \begin{pmatrix}\frac{\partial z_1}{\partial\epsilon_1} & \cdots & \frac{\partial z_1}{\partial\epsilon_k}\\ \vdots & \ddots & \vdots\\ \frac{\partial z_k}{\partial\epsilon_1} & \cdots & \frac{\partial z_k}{\partial\epsilon_k}\end{pmatrix} \tag{2.35}$$

As we will show, we can build very flexible transformations $\mathbf{g}()$ for which $\log d_{\boldsymbol{\phi}}(\mathbf{x},\boldsymbol{\epsilon})$ is simple to compute, resulting in highly flexible inference models $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$.

2.6 FACTORIZED GAUSSIAN POSTERIORS

A common choice is a simple factorized Gaussian encoder $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2))$:

$$(\boldsymbol{\mu}, \log \boldsymbol{\sigma}) = \text{EncoderNeuralNet}_{\boldsymbol{\phi}}(\mathbf{x}) \tag{2.36}$$

$$q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = \prod_i q_{\boldsymbol{\phi}}(z_i|\mathbf{x}) = \prod_i \mathcal{N}(z_i; \mu_i, \sigma_i^2) \tag{2.37}$$

where $\mathcal{N}(z_i; \mu_i, \sigma_i^2)$ is the PDF of the univariate Gaussian distribution. After reparameterization, we can write:

$$\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}) \tag{2.38}$$

$$(\boldsymbol{\mu}, \log \boldsymbol{\sigma}) = \text{EncoderNeuralNet}_{\boldsymbol{\phi}}(\mathbf{x}) \tag{2.39}$$

$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon} \tag{2.40}$$

where $\odot$ is the element-wise product. The Jacobian of the transformation from $\boldsymbol{\epsilon}$ to $\mathbf{z}$ is:

$$\frac{\partial \mathbf{z}}{\partial \boldsymbol{\epsilon}} = \text{diag}(\boldsymbol{\sigma}), \tag{2.41}$$

i.e. a diagonal matrix with the elements of $diag(\boldsymbol{\sigma})$ on the diagonal. The determinant of a diagonal (or more generally, triangular) matrix is the product of its diagonal terms. The log determinant of the Jacobian is therefore:

$$\log d_{\boldsymbol{\phi}}(\mathbf{x}, \boldsymbol{\epsilon}) = \log \left| \det \left(\frac{\partial \mathbf{z}}{\partial \boldsymbol{\epsilon}} \right) \right| = \sum_i \log \sigma_i \tag{2.42}$$

and the posterior density is:

$$\log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = \log p(\boldsymbol{\epsilon}) - \log d_{\boldsymbol{\phi}}(\mathbf{x}, \boldsymbol{\epsilon}) \tag{2.43}$$

$$= \sum_i \log \mathcal{N}(\epsilon_i; 0, 1) - \log \sigma_i \tag{2.44}$$

when $\mathbf{z} = \mathbf{g}(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \mathbf{x})$.

2.6.1 *Full-covariance Gaussian posterior*

The factorized Gaussian posterior can be extended to a Gaussian with full covariance:

$$q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{2.45}$$

A reparameterization of this distribution is given by:

$$\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}) \tag{2.46}$$

$$\mathbf{z} = \boldsymbol{\mu} + \mathbf{L}\boldsymbol{\epsilon} \tag{2.47}$$

where $\mathbf{L}$ is a lower (or upper) triangular matrix, with non-zero entries on the diagonal. The off-diagonal elements define the correlations (covariances) of the elements in $\mathbf{z}$.

The reason for this parameterization of the full-covariance Gaussian, is that the Jacobian determinant is remarkably simple. The Jacobian in this case is trivial: $\frac{\partial \mathbf{z}}{\partial \boldsymbol{\epsilon}} = \mathbf{L}$. Note that the determinant of a triangular matrix is the product of its diagonal elements. Therefore, in this parameterization:

$$\log |\det(\frac{\partial \mathbf{z}}{\partial \boldsymbol{\epsilon}})| = \sum_i \log |L_{ii}| \tag{2.48}$$

And the log-density of the posterior is:

$$\log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = \log p(\boldsymbol{\epsilon}) - \sum_i \log |L_{ii}| \tag{2.49}$$

This parameterization corresponds to the Cholesky decomposition $\boldsymbol{\Sigma} = \mathbf{L}\mathbf{L}^T$ of the covariance of $\mathbf{z}$:

$$\boldsymbol{\Sigma} = \mathbb{E}\left[(\mathbf{z} - \mathbb{E}\left[\mathbf{z}|\right])(\mathbf{z} - \mathbb{E}\left[\mathbf{z}|\right])^T\right] \tag{2.50}$$

$$= \mathbb{E}\left[\mathbf{L}\boldsymbol{\epsilon}(\mathbf{L}\boldsymbol{\epsilon})^T\right] = \mathbf{L}\mathbb{E}\left[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^T\right]\mathbf{L}^T \tag{2.51}$$

$$= \mathbf{L}\mathbf{L}^T \tag{2.52}$$

Note that $\mathbb{E}\left[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^T\right] = \mathbf{I}$ since $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$.

One way to build a matrix $\mathbf{L}$ with the desired properties, namely triangularity and non-zero diagonal entries, is by constructing it as follows:

$$(\boldsymbol{\mu}, \log \boldsymbol{\sigma}, \mathbf{L}') \leftarrow \text{EncoderNeuralNet}_{\boldsymbol{\phi}}(\mathbf{x}) \tag{2.53}$$

$$\mathbf{L} \leftarrow \mathbf{L}_{mask} \odot \mathbf{L}' + \text{diag}(\boldsymbol{\sigma}) \tag{2.54}$$

$\mathbf{L}_{mask}$ is a masking matrix with zeros on and above the diagonal, and ones below the diagonal. The log-determinant is identical to the factorized Gaussian case:

$$\log \left|\det\left(\frac{\partial \mathbf{z}}{\partial \boldsymbol{\epsilon}}\right)\right| = \sum_i \log \sigma_i \tag{2.55}$$

More generally, we can replace $\mathbf{z} = \mathbf{L}\boldsymbol{\epsilon} + \boldsymbol{\mu}$ with a chain of (differentiable and nonlinear) transformations; as long as the Jacobian of each step in the chain is triangular with non-zero diagonal entries, the log determinant remains simple. This principle is used by *inverse autoregressive flow* (IAF) as explored by Kingma et al. [2016] and discussed in chapter 5.

Algorithm 2: Computation of unbiased estimate of single-datapoint ELBO for example VAE with a full-covariance Gaussian inference model and a factorized Bernoulli generative model. $\mathbf{L}_{mask}$ is a masking matrix with zeros on and above the diagonal, and ones below the diagonal. Note that $\mathbf{L}$ must be a triangular matrix with positive entries on the diagonal.

Data:
$\mathbf{x}$: a datapoint, and optionally other conditioning information
$\boldsymbol{\epsilon}$: a random sample from $p(\boldsymbol{\epsilon}) = \mathcal{N}(0, \mathbf{I})$
$\boldsymbol{\theta}$: Generative model parameters
$\boldsymbol{\phi}$: Inference model parameters
$q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$: Inference model
$p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$: Generative model
Result:
$\tilde{\mathcal{L}}$: unbiased estimate of the single-datapoint ELBO $\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x})$

$(\boldsymbol{\mu}, \log \boldsymbol{\sigma}, \mathbf{L}') \leftarrow \text{EncoderNeuralNet}_{\boldsymbol{\phi}}(\mathbf{x})$
$\mathbf{L} \leftarrow \mathbf{L}_{mask} \odot \mathbf{L}' + \text{diag}(\boldsymbol{\sigma})$
$\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$
$\mathbf{z} \leftarrow \mathbf{L}\boldsymbol{\epsilon} + \boldsymbol{\mu}$
$\text{logqz} \leftarrow -\texttt{sum}(\frac{1}{2}(\boldsymbol{\epsilon}^2 + \log(2\pi) + \log \boldsymbol{\sigma}))$ $\triangleright = q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$
$\text{logpz} \leftarrow -\texttt{sum}(\frac{1}{2}(\mathbf{z}^2 + \log(2\pi)))$ $\triangleright = p_{\boldsymbol{\theta}}(\mathbf{z})$
$\mathbf{p} \leftarrow \text{DecoderNeuralNet}_{\boldsymbol{\theta}}(\mathbf{z})$
$\text{logpx} \leftarrow \texttt{sum}(\mathbf{x} \odot \log \mathbf{p} + (1 - \mathbf{x}) \odot \log(1 - \mathbf{p}))$ $\triangleright = p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$
$\tilde{\mathcal{L}} = \text{logpx} + \text{logpz} - \text{logqz}$

2.7 ESTIMATION OF THE MARGINAL LIKELIHOOD

After training a VAE, we can estimate the probability of data under the model using an *importance sampling* technique, as originally proposed by

Rezende et al. [2014]. The marginal likelihood of a datapoint can be written as:

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) = \log \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \left[p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) / q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \right] \tag{2.56}$$

Taking random samples from $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$, a Monte Carlo estimator of this is:

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) \approx \log \frac{1}{L} \sum_{l=1}^{L} p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}^{(l)}) / q_{\boldsymbol{\phi}}(\mathbf{z}^{(l)}|\mathbf{x}) \tag{2.57}$$

where each $\mathbf{z}^{(l)} \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ is a random sample from the inference model. By making L large, the approximation becomes a better estimate of the marginal likelihood, and in fact since this is a Monte Carlo estimator, for $L \to \infty$ this converges to the actual marginal likelihood.

Notice that when setting $L = 1$, this equals the ELBO estimator of the VAE. When $L > 1$. We can also use the estimator of 2.57 as our objective function; this is the objective used in *importance weighted autoencoders* [Burda et al., 2015] (IWAE). In that paper, it was also shown that the objective has increasing tightness for increasing value of L. It was later shown by Cremer and Duvenaud [2017] that the IWAE objective can be re-interpreted as an ELBO objective with a particular inference model. The downside of these approaches for optimizing a tighter bound, is that importance weighted estimates have notoriously bad scaling properties to high-dimensional latent spaces.

2.8 MARGINAL LIKELIHOOD AND ELBO AS KL DIVERGENCES

One way to improve the potential tightness of the bound, is increasing the flexibility of the generative model. This can be understood through a connection between the ELBO and the KL divergence.

With i.i.d. dataset $\mathcal{D}$ of size $N_{\mathcal{D}}$, the maximum likelihood criterion is:

$$\log p_{\boldsymbol{\theta}}(\mathcal{D}) = \frac{1}{N_{\mathcal{D}}} \sum_{\mathbf{x} \in \mathcal{D}} \log p_{\boldsymbol{\theta}}(\mathbf{x}) \tag{2.58}$$

$$= \mathbb{E}_{q_{\mathcal{D}}(\mathbf{x})} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x}) \right] \tag{2.59}$$

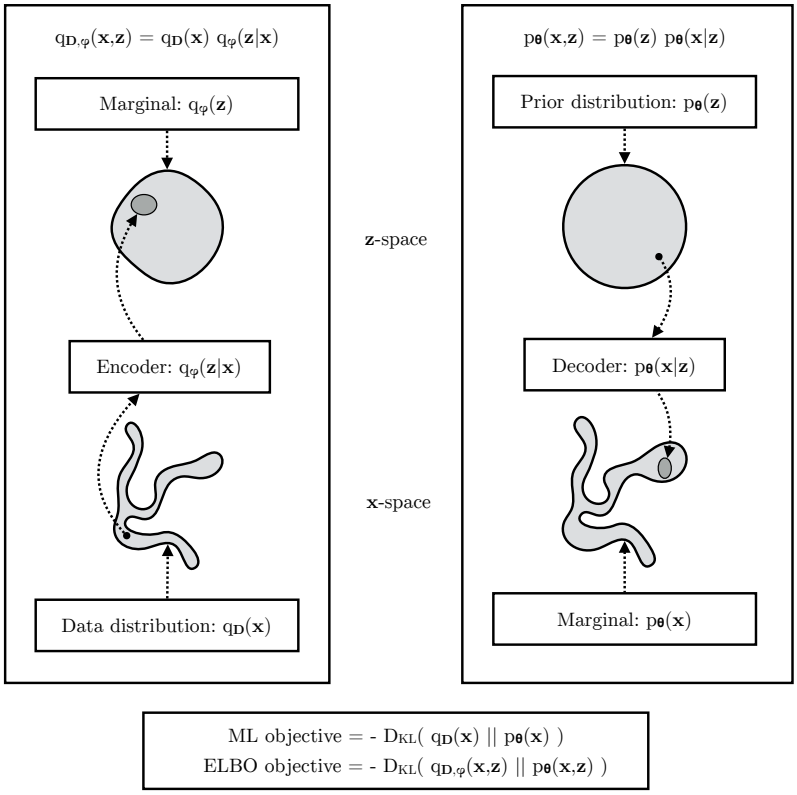


Figure 2.4: The maximum likelihood (ML) objective can be viewed as the minimization of $D_{KL}(q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{x}))$, while the ELBO objective can be viewed as the minimization of $D_{KL}(q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{x},\mathbf{z})||p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}))$, which upper bounds $D_{KL}(q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{x}))$. If a perfect fit is not possible, then $p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})$ will typically end up with higher variance than $q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{x},\mathbf{z})$, because of the direction of the KL divergence.

where $q_{\mathcal{D}}(\mathbf{x})$ is the empirical (data) distribution, which is a mixture distribution:

$$q_{\mathcal{D}}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} q_{\mathcal{D}}^{(i)}(\mathbf{x}) \tag{2.60}$$

where each component $q_{\mathcal{D}}^{(i)}(\mathbf{x})$ typically corresponds to a Dirac delta distribution centered at value $\mathbf{x}^{(i)}$ in case of continuous data, or a discrete distribution with all probability mass concentrated at value $\mathbf{x}^{(i)}$ in case of discrete data. The Kullback Leibler (KL) divergence between the data and model distributions, can be rewritten as the negative log-likelihood, plus a constant:

$$\begin{aligned} D_{KL}(q_{\mathcal{D}}(\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{x})) &= -\mathbb{E}_{q_{\mathcal{D}}(\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x})\right] + \mathbb{E}_{q_{\mathcal{D}}(\mathbf{x})}\left[\log q_{\mathcal{D}}(\mathbf{x})\right] && (2.61)\\ &= -\log p_{\boldsymbol{\theta}}(\mathcal{D}) + \text{constant} && (2.62)\end{aligned}$$

where constant $= -\mathcal{H}(q_{\mathcal{D}}(\mathbf{x}))$. So minimization of the KL divergence above is equivalent to maximization of the data log-likelihood $\log p_{\boldsymbol{\theta}}(\mathcal{D})$.

Taking the combination of the empirical data distribution $q_{\mathcal{D}}(\mathbf{x})$ and the inference model, we get a joint distribution over data $\mathbf{x}$ and latent variables $\mathbf{z}$: $q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{x},\mathbf{z}) = q_{\mathcal{D}}(\mathbf{x})q(\mathbf{z}|\mathbf{x})$.

The KL divergence of this joint distribution, can be rewritten as the negative ELBO, plus a constant:

$$\begin{aligned} &D_{KL}(q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{x},\mathbf{z})||p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})) && (2.63)\\ &= -\mathbb{E}_{q_{\mathcal{D}}(\mathbf{x})}\left[\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\right] - \log q_{\mathcal{D}}(\mathbf{x})\right] && (2.64)\\ &= -\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathcal{D}) + \text{constant} && (2.65)\end{aligned}$$

where constant $= -\mathcal{H}(q_{\mathcal{D}}(\mathbf{x}))$. So maximization of the ELBO, is equivalent to the minimization of this KL divergence $D_{KL}(q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{x},\mathbf{z})||p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}))$. The relationship between the ML and ELBO objectives can be summarized in the following simple equation:

$$\begin{aligned} &D_{KL}(q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{x},\mathbf{z})||p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})) && (2.66)\\ &= D_{KL}(q_{\mathcal{D}}(\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{x})) + \mathbb{E}_{q_{\mathcal{D}}(\mathbf{x})}\left[D_{KL}(q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}))\right] && (2.67)\\ &\leq D_{KL}(q_{\mathcal{D}}(\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{x})) && (2.68)\end{aligned}$$

One additional perspective is that the ELBO can be viewed as a maximum likelihood objective *in an augmented space*. For some fixed choice of encoder

$q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$, we can view the joint distribution $p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$ as an augmented empirical distribution over the original data $\mathbf{x}$ and (stochastic) auxiliary features $\mathbf{z}$ associated with each datapoint. The model $p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$ then defines a joint model over the original data, and the auxiliary features.

2.9 CHALLENGES

2.9.1 *Optimization issues*

In our work, consistent with findings in [Bowman et al., 2015] and [Kaae Sønderby et al., 2016], we found that stochastic optimization with the unmodified lower bound objective can gets stuck in an undesirable stable equilibrium. At the start of training, the likelihood term $\log p(\mathbf{x}|\mathbf{z})$ is relatively weak, such that an initially attractive state is where $q(\mathbf{z}|\mathbf{x}) \approx p(\mathbf{z})$. In this state, encoder gradients have a relatively low signal-to-noise ratio, resulting in a stable equilibrium from which it is difficult to escape. The solution proposed in [Bowman et al., 2015] and [Kaae Sønderby et al., 2016] is to use an optimization schedule where the weights of the latent cost $D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))$ is slowly annealed from 0 to 1 over many epochs.

An alternative proposed in [Kingma et al., 2016] is the method of *free bits*: a modification of the ELBO objective, that ensures that on average, a certain minimum number of bits amount of information are encoded per latent variable, or per group of latent variables.

The latent dimensions are divided into the K groups. We then use the following minibatch objective, which ensures that using less than λ nats of information per subset j (on average per minibatch $\mathcal{M}$) is not advantageous:

$$\widetilde{\mathcal{L}}_{\lambda} = \mathbb{E}_{\mathbf{x}\sim\mathcal{M}}\left[\mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\log p(\mathbf{x}|\mathbf{z})\right]\right] \tag{2.69}$$

$$- \sum_{j=1}^{K} \text{maximum}(\lambda, \mathbb{E}_{\mathbf{x}\sim\mathcal{M}}\left[D_{KL}(q(z_j|\mathbf{x})||p(z_j))\right] \tag{2.70}$$

Since increasing the latent information is generally advantageous for the first (unaffected) term of the objective (often called the *negative reconstruction error*), this results in $\mathbb{E}_{\mathbf{x}\sim\mathcal{M}}\left[D_{KL}(q(\mathbf{z}_j|\mathbf{x})||p(\mathbf{z}_j))\right] \geq \lambda$ for all j, in practice. In Kingma et al. [2016] and chapter 5, it is shown that the method works well for a fairly wide range of values ($\lambda \in [0.125, 0.25, 0.5, 1, 2]$), resulting

in significant improvement in the resulting log-likelihood on a benchmark result.

2.9.2 *Blurriness of generative model*

In section 2.8 we saw that optimizing the ELBO is equivalent to minimizing $D_{KL}(q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{x},\mathbf{z})||p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}))$. If a perfect fit between $q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{x},\mathbf{z})$ and $p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})$ is not possible, then the variance of $p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})$ and $p_{\boldsymbol{\theta}}(\mathbf{x})$ will end up larger than the variance $q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{x},\mathbf{z})$ and the data $q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{x})$. This is due to the direction of the KL divergence; if there are values of $(\mathbf{x},\mathbf{z})$ which are likely under $q_{\mathcal{D},\boldsymbol{\phi}}$ but not under $p_{\boldsymbol{\theta}}$, the term $\mathbb{E}_{q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{x},\mathbf{z})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})\right]$ will go to infinity. However, the reverse is not true: the generative model is only slightly penalized when putting probability mass on values of $(\mathbf{x},\mathbf{z})$ with no support under $q_{\mathcal{D},\boldsymbol{\phi}}$.

Issues with 'blurriness' can thus can be countered by choosing a sufficiently flexible inference model, and/or a sufficiently flexible generative model. In the next two chapters we will discuss techniques for constructing flexible inference models and flexible generative models.

2.10 RELATED PRIOR AND CONCURRENT WORK

Here we briefly discuss relevant literature prior and concurrent our work in [Kingma and Welling, 2013]. The wake-sleep algorithm [Hinton et al., 1995] is, to the best of our knowledge, the only other on-line learning method in the literature that is applicable to the same general class of continuous latent variable models. Like our method, the wake-sleep algorithm employs a recognition model that approximates the true posterior. A drawback of the wake-sleep algorithm is that it requires a concurrent optimization of two objective functions, which together do not correspond to optimization of (a bound of) the marginal likelihood. An advantage of wake-sleep is that it also applies to models with discrete latent variables. Wake-Sleep has the same computational complexity as AEVB per datapoint.

Stochastic variational inference Hoffman et al. [2013] has received increasing interest. Blei et al. [2012] introduced a control variate schemes to reduce the variance of the score function gradient estimator, and applied the estimator to exponential family approximations of the posterior. In [Ranganath et al., 2014] some general methods, e.g. a control variate scheme, were introduced for reducing the variance of the original gradient estimator. In Sali-

mans and Knowles [2013], a similar reparameterization as in this work was used in an efficient version of a stochastic variational inference algorithm for learning the natural parameters of exponential-family approximating distributions.

In Graves [2011] a similar estimator of the gradient is introduced; however the estimator of the variance is not an unbiased estimator w.r.t. the ELBO gradient.

The VAE training algorithm exposes a connection between directed probabilistic models (trained with a variational objective) and auto-encoders. A connection between *linear* auto-encoders and a certain class of generative linear-Gaussian models has long been known. In [Roweis, 1998] it was shown that PCA corresponds to the maximum-likelihood (ML) solution of a special case of the linear-Gaussian model with a prior $p(\mathbf{z}) = \mathcal{N}(0, \mathbf{I})$ and a conditional distribution $p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}; \mathbf{W}\mathbf{z}, \epsilon\mathbf{I})$, specifically the case with infinitesimally small ϵ. In this limiting case, the posterior over the latent variables $p(\mathbf{z}|\mathbf{x})$ is a Dirac delta distribution: $p(\mathbf{z}|\mathbf{x}) = \delta(\mathbf{z} - \mathbf{W}'\mathbf{x})$ where $\mathbf{W}' = (\mathbf{W}^T\mathbf{W})^{-1}\mathbf{W}^T$, i.e., given $\mathbf{W}$ and $\mathbf{x}$ there is no uncertainty about latent variable $\mathbf{z}$. Roweis [1998] then introduces an EM-type approach to learning $\mathbf{W}$. Much earlier work [Bourlard and Kamp, 1988] showed that optimization of linear autoencoders retrieves the principal components of data, from which it follows that learning linear autoencoders correspond to a specific method for learning the above case of linear-Gaussian probabilistic model of the data. However, this approach using linear autoencoders is limited to linear-Gaussian models, while our approach applies to a much broader class of continuous latent variable models.

When using neural networks for both the inference model and the generative model, the combination forms a type of autoencoder [Goodfellow et al., 2016] with a specific regularization term:

$$\tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x};\boldsymbol{\epsilon}) = \underbrace{\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})}_{\text{Negative reconstruction error}} + \underbrace{\log p_{\boldsymbol{\theta}}(\mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}_{\text{Regularization terms}} \tag{2.71}$$

In an analysis of plain autoencoders [Vincent et al., 2010] it was shown that the training criterion of unregularized autoencoders corresponds to maximization of a lower bound (see the infomax principle [Linsker, 1989]) of the mutual information between input X and latent representation Z. Maximizing (w.r.t. parameters) of the mutual information is equivalent to maximizing the conditional entropy, which is lower bounded by the expected log-likelihood of the data under the autoencoding model [Vincent et al., 2010],

i.e. the negative reconstruction error. However, it is well known that this reconstruction criterion is in itself not sufficient for learning useful representations Bengio et al. [2013]. Regularization techniques have been proposed to make autoencoders learn useful representations, such as denoising, contractive and sparse autoencoder variants [Bengio et al., 2013]. The VAE objective contains a regularization term dictated by the variational bound, lacking the usual nuisance regularization hyper-parameter required to learn useful representations. Related are also encoder-decoder architectures such as the predictive sparse decomposition (PSD) Kavukcuoglu et al. [2008], from which we drew some inspiration. Also relevant are the recently introduced Generative Stochastic Networks Bengio and Thibodeau-Laufer [2013] where noisy auto-encoders learn the transition operator of a Markov chain that samples from the data distribution. In Salakhutdinov and Larochelle [2010] a recognition model was employed for efficient learning with Deep Boltzmann Machines. These methods are targeted at either unnormalized models (i.e. undirected models like Boltzmann machines) or limited to sparse coding models, in contrast to our proposed algorithm for learning a general class of directed probabilistic models.

The proposed DARN method [Gregor et al., 2013], also learns a directed probabilistic model using an auto-encoding structure, however their method applies to binary latent variables. In concurrent work, Rezende et al. [2014] also make the connection between auto-encoders, directed probabilistic models and stochastic variational inference using the reparameterization trick we describe in [Kingma and Welling, 2013]. Their work was developed independently of ours and provides an additional perspective on the VAE.

2.10.1 *Score function estimator*

An alternative unbiased stochastic gradient estimator of the ELBO is the *score function estimator* [Kleijnen and Rubinstein, 1996]:

$$\nabla_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \left[f(\mathbf{z}) \right] = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \left[f(\mathbf{z}) \nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \right] \tag{2.72}$$

$$\simeq f(\mathbf{z}) \nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \tag{2.73}$$

where $\mathbf{z} \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$.

This is also known as the *likelihood ratio* estimator [Glynn, 1990, Fu, 2006] and the REINFORCE gradient estimator [Williams, 1992]. The method has been successfully used in various methods like *neural variational infer-*

ence [Mnih and Gregor, 2014], *black-box variational inference* [Ranganath et al., 2014], *automated variational inference* [Wingate and Weber, 2013], and *variational stochastic search* [Paisley et al., 2012], often in combination with various novel control variate techniques [Glasserman, 2013] for variance reduction. An advantage of the likelihood ratio estimator is its applicability to discrete latent variables.

We do not directly compare to these techniques, since we concern ourselves with continuous latent variables, in which case we have (computationally cheap) access to gradient information $\nabla_{\mathbf{z}} \log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$, courtesy of the backpropagation algorithm. The score function estimator solely uses the scalar-valued $\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$, ignoring the gradient information about the function $\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$, generally leading to much higher variance. This has been experimentally confirmed by e.g. [Kucukelbir et al., 2016], which finds that a sophisticated score function estimator requires two orders of magnitude more samples to arrive at the same variance as a reparameterization based estimator.

The difference in efficiency of our proposed reparameterization-based gradient estimator, compared to score function estimators, can intuitively be understood as removing an information bottleneck during the computation of gradients of the ELBO w.r.t. $\boldsymbol{\phi}$ from current parameters $\boldsymbol{\theta}$: in the latter case, this computation is bottlenecked by the scalar value $\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$, while in the former case it is bottlenecked by the much wider vector $\nabla_{\mathbf{z}} \log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$.

2.11 EXPERIMENTS

This section describes the experiments published in [Kingma and Welling, 2013]. We learned generative models of images from the MNIST and Frey Face datasets[2] and compared learning algorithms in terms of the variational lower bound, and the estimated marginal likelihood.

In variational auto-encoders, neural networks are used as probabilistic encoders and decoders. There are many possible choices of encoders and decoders, depending on the type of data and model. In our example we used relatively simple neural networks, namely multi-layered perceptrons (MLPs). For the encoder we used a MLP with Gaussian output, while for the decoder we used MLPs with either Gaussian or Bernoulli outputs, depending on the type of data.

[2] Available at `http://www.cs.nyu.edu/~roweis/data.html`

In case of a Bernoulli MLP as decoder, we let $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ be a multivariate Bernoulli whose probabilities are computed from $\mathbf{z}$ with a fully-connected neural network with a single hidden layer:

$$\begin{aligned} \log p(\mathbf{x}|\mathbf{z}) &= \sum_{i=1}^{D} x_i \log y_i + (1 - x_i) \cdot \log(1 - y_i) \\ \text{where } \mathbf{y} &= f_\sigma(\mathbf{W}_2 \tanh(\mathbf{W}_1 \mathbf{z} + \mathbf{b}_1) + \mathbf{b}_2) \end{aligned} \tag{2.74}$$

where $f_\sigma(.)$ is the element-wise sigmoid activation function, and where $\boldsymbol{\theta} = \{\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2\}$ are the weights and biases of the MLP.

In the case of a Gaussian MLP as encoder or decoder, we let the encoder or decoder be a multivariate Gaussian with a diagonal covariance structure:

$$\begin{aligned} \log p(\mathbf{x}|\mathbf{z}) &= \log \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \sigma^2 \mathbf{I}) \\ \text{where } \boldsymbol{\mu} &= \mathbf{W}_4 \mathbf{h} + \mathbf{b}_4 \\ \log \sigma^2 &= \mathbf{W}_5 \mathbf{h} + \mathbf{b}_5 \\ \mathbf{h} &= \tanh(\mathbf{W}_3 \mathbf{z} + \mathbf{b}_3) \end{aligned} \tag{2.75}$$

where $\{\mathbf{W}_3, \mathbf{W}_4, \mathbf{W}_5, \mathbf{b}_3, \mathbf{b}_4, \mathbf{b}_5\}$ are the weights and biases of the MLP and part of $\boldsymbol{\theta}$ when used as decoder. Note that when this network is used as an encoder $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$, then $\mathbf{z}$ and $\mathbf{x}$ are swapped, and the weights and biases are variational parameters $\boldsymbol{\phi}$.

For the experiments, we let the prior over the latent variables be the centered isotropic multivariate Gaussian $p_{\boldsymbol{\theta}}(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$. Note that in this case, the prior lacks parameters. We let $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ be a multivariate Gaussian (in case of real-valued data) or Bernoulli (in case of binary data) whose distribution parameters are computed from $\mathbf{z}$ with a MLP (a fully-connected neural network with a single hidden layer). Note the true posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ is in this case intractable. While there is much freedom in the form $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$, we'll assume the true (but intractable) posterior takes on a approximate Gaussian form with an approximately diagonal covariance. In this case, we can let the variational approximate posterior be a multivariate Gaussian with a diagonal covariance structure[3]:

$$\log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = \log \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \sigma^2 \mathbf{I}) \tag{2.76}$$

[3] Note that this is just a (simplifying) choice, and not a limitation of our method.

where the mean and s.d. of the approximate posterior, $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$, are outputs of the encoding MLP, i.e. nonlinear functions of datapoint $\mathbf{x}$ and the variational parameters $\boldsymbol{\phi}$.

As explained in section 2.5, we sample from the posterior $\mathbf{z} \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ using $\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$ where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. With $\odot$ we signify an element-wise product. In this model both $p_{\boldsymbol{\theta}}(\mathbf{z})$ (the prior) and $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ are Gaussian; in this case, we can use the ELBO gradient estimator where the KL divergence can be computed and differentiated without estimation). The resulting estimator for this model and datapoint $\mathbf{x}$ is:

$$\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) \simeq \frac{1}{2} \sum_{j=1}^{J} \left(1 + \log((\sigma_j)^2) - (\mu_j)^2 - (\sigma_j)^2\right) + \frac{1}{L} \sum_{l=1}^{L} \log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$$
$$\text{where} \quad \mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon} \quad \text{and} \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}) \tag{2.77}$$

In these experiments, the decoding term $\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ is a Bernoulli or Gaussian MLP, depending on the type of data we are modeling.

The used encoder and decoder have an equal number of hidden units. Since the Frey Face data are continuous, we used a decoder with Gaussian outputs, identical to the encoder, except that the means were constrained to the interval $(0, 1)$ using a sigmoidal activation function at the decoder output. Note that with *hidden units* we refer to the hidden layer of the neural networks of the encoder and decoder.

Parameters are updated using stochastic gradient ascent where gradients are computed by differentiating the lower bound estimator $\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}} \mathbf{x}$, plus a small weight decay term corresponding to a prior $p(\boldsymbol{\theta}) = \mathcal{N}(0, \mathbf{I})$. Optimization of this objective is equivalent to approximate MAP estimation, where the likelihood gradient is approximated by the gradient of the lower bound.

We compared performance of our method to the wake-sleep algorithm Hinton et al. [1995]. We employed the same encoder (also called recognition model) for the wake-sleep algorithm and the variational auto-encoder. All parameters, both variational and generative, were initialized by random sampling from $\mathcal{N}(0, 0.01)$, and were jointly stochastically optimized using the MAP criterion. Stepsizes were adapted with Adagrad Duchi et al. [2010]; the Adagrad global stepsize parameters were chosen from {0.01, 0.02, 0.1} based on performance on the training set in the first few iterations. Minibatches of size $M = 100$ were used, with one sample from the inference model per datapoint.

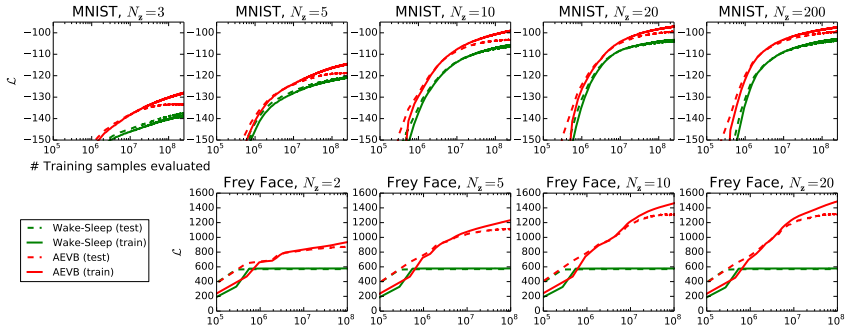


Figure 2.5: Comparison, on log-log scales, of our method to the wake-sleep algorithm, in terms of optimizing the lower bound, for different dimensionality of latent space (N_z). Our method converged considerably faster and reached a better solution in all experiments. Interestingly enough, more latent variables does not result in more overfitting, which is explained by the regularizing effect of the lower bound. Vertical axis: the estimated average variational lower bound per datapoint. The estimator variance was small (< 1) and omitted. Horizontal axis: amount of training points evaluated. Computation took around 20-40 minutes per million training samples with a Intel Xeon CPU running at an effective 40 GFLOPS.

2.11.1 *Likelihood lower bound*

We trained generative models (decoders) and corresponding encoders (a.k.a. recognition models) having 500 hidden units in case of MNIST, and 200 hidden units in case of the Frey Face dataset (to prevent overfitting, since it is a considerably smaller dataset). The chosen number of hidden units is based on prior literature on auto-encoders, and the relative performance of different algorithms was not very sensitive to these choices. Figure 2.5 shows the results when comparing the lower bounds. Interestingly, superfluous latent variables did not result in overfitting, which is explained by the regularizing nature of the variational bound.

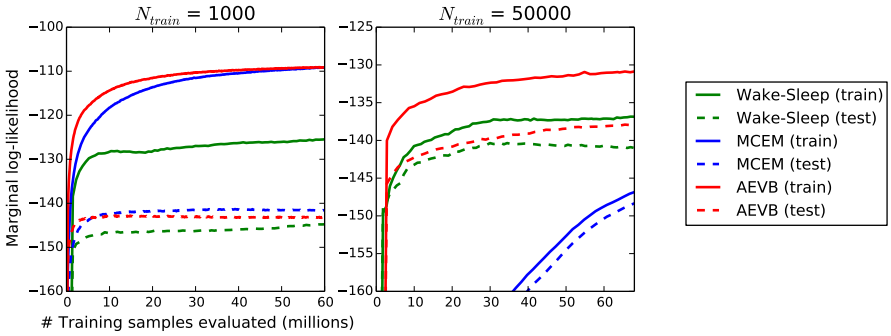


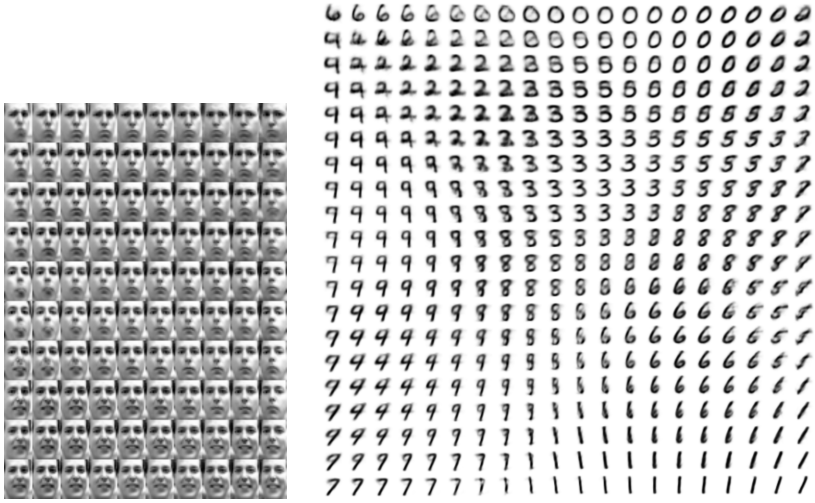
Figure 2.6: Comparison of our method (here called AEVB, for *auto-encoding variational Bayes*) to the wake-sleep algorithm and Monte Carlo EM, in terms of the estimated marginal likelihood, for a different number of training points. Monte Carlo EM is not an on-line algorithm, and (unlike our method and the wake-sleep method) can't be applied efficiently for the full MNIST dataset.

2.11.2 *Marginal likelihood*

For the encoder and decoder we again used neural networks, this time with 100 hidden units, and 3 latent variables; for higher dimensional latent space the estimates became less reliable. Again, the MNIST dataset was used. Our method and Wake-Sleep methods were compared to Monte Carlo EM (MCEM) with a Hybrid Monte Carlo (HMC) Duane et al. [1987] sampler; details are in the appendix. We compared the convergence speed for the three algorithms, for a small and large training set size. Results are in figure 2.6.

2.11.3 *Visualization of high-dimensional data*

If we choose a low-dimensional latent space (e.g. 2D), we can use the learned encoders (recognition model) to project high-dimensional data to a low-dimensional manifold. See figures 2.7 and 2.8 for visualizations of latent space and corresponding observed space of models learned with our proposed method. In 2.9 we visualize random samples from a VAE-based generative model of faces.



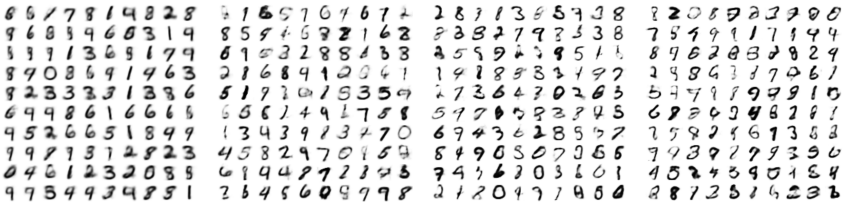
(a) Learned Frey Face manifold

(b) Learned MNIST manifold

Figure 2.7: Visualizations of learned data manifold for generative models with two-dimensional latent space, learned with AEVB. Since the prior of the latent space is Gaussian, linearly spaced coordinates on the unit square were transformed through the inverse CDF of the Gaussian to produce values of the latent variables $\mathbf{z}$. For each of these values $\mathbf{z}$, we plotted the corresponding generative $p_{\theta}(\mathbf{x}|\mathbf{z})$ with the learned parameters $\boldsymbol{\theta}$.

2.12 CONCLUSION

We have introduced a reparameterization-based estimator of the gradient of the ELBO, for efficient approximate inference with continuous latent variables. The proposed estimator can be straightforwardly optimized using standard stochastic gradient methods. For the case of i.i.d. datasets and continuous latent variables per datapoint we introduce an efficient framework for efficient inference and learning, the variational autoencoder (VAE), that learns an approximate inference model using the proposed estimator. The theoretical advantages are reflected in experimental results.



(a) 2-D latent space (b) 5-D latent space (c) 10-D latent space (d) 20-D latent space

Figure 2.8: Random samples from learned generative models of MNIST for different dimensionalities of latent space.

Figure 2.9: Random full-color samples from a generative model of faces. We used the same relatively simple VAE as the one used for the previous Frey Face example, this time trained on a low-resolution version of the *Labeled Faces in the Wild* (LFW) dataset. Shown are random samples from the model. Better models are possible through more sophisticated generative models (chapter 4) and inference models (chapter 5), see e.g. [White, 2016].

CHAPTER APPENDIX

2.A SOLUTION OF $-D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z})||p_{\boldsymbol{\theta}}(\mathbf{z}))$, GAUSSIAN CASE

The ELBO contains a KL term that can often be integrated analytically. Here we give the solution when both the prior $p_{\boldsymbol{\theta}}(\mathbf{z}) = \mathcal{N}(0, \mathbf{I})$ and the posterior approximation $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})$ are Gaussian. Let J be the dimensionality of $\mathbf{z}$. Let $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ denote the variational mean and s.d. evaluated at datapoint i, and let μ_j and σ_j simply denote the j-th element of these vectors. Then:

$$\begin{aligned}\int q_{\boldsymbol{\theta}}(\mathbf{z}) \log p(\mathbf{z})\, d\mathbf{z} &= \int \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2) \log \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})\, d\mathbf{z} \\ &= -\frac{J}{2}\log(2\pi) - \frac{1}{2}\sum_{j=1}^{J}(\mu_j^2 + \sigma_j^2)\end{aligned}$$

And:

$$\begin{aligned}\int q_{\boldsymbol{\theta}}(\mathbf{z}) \log q_{\boldsymbol{\theta}}(\mathbf{z})\, d\mathbf{z} &= \int \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2) \log \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2)\, d\mathbf{z} \\ &= -\frac{J}{2}\log(2\pi) - \frac{1}{2}\sum_{j=1}^{J}(1 + \log \sigma_j^2)\end{aligned}$$

Therefore:

$$\begin{aligned}-D_{KL}((q_{\boldsymbol{\phi}}(\mathbf{z})||p_{\boldsymbol{\theta}}(\mathbf{z})) &= \int q_{\boldsymbol{\theta}}(\mathbf{z}) \left(\log p_{\boldsymbol{\theta}}(\mathbf{z}) - \log q_{\boldsymbol{\theta}}(\mathbf{z})\right)\, d\mathbf{z} \\ &= \frac{1}{2}\sum_{j=1}^{J}\left(1 + \log((\sigma_j)^2) - (\mu_j)^2 - (\sigma_j)^2\right)\end{aligned}$$

When using a recognition model $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ then $\boldsymbol{\mu}$ and s.d. $\boldsymbol{\sigma}$ are simply functions of $\mathbf{x}$ and the variational parameters $\boldsymbol{\phi}$, as exemplified in the text.

2.B MONTE CARLO EM

The Monte Carlo EM algorithm does not employ an encoder, instead it samples from the posterior of the latent variables using gradients of the posterior

computed with $\nabla_{\mathbf{z}} \log p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) = \nabla_{\mathbf{z}} \log p_{\boldsymbol{\theta}}(\mathbf{z}) + \nabla_{\mathbf{z}} \log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$. The Monte Carlo EM procedure consists of 10 HMC leapfrog steps with an automatically tuned stepsize such that the acceptance rate was 90%, followed by 5 weight updates steps using the acquired sample. For all algorithms the parameters were updated using the Adagrad step-sizes (with accompanying annealing schedule).

2.C FULL VB

As explained, it is possible to perform variational inference on both the parameters $\boldsymbol{\theta}$ and the latent variables $\mathbf{z}$, as opposed to just the latent variables as we did in this work. Here, we'll derive our estimator for that case.

Let $p_{\boldsymbol{\alpha}}(\boldsymbol{\theta})$ be some prior for the parameters introduced above, parameterized by $\boldsymbol{\alpha}$. The marginal likelihood can be written as:

$$\log p_{\boldsymbol{\alpha}}(\mathbf{X}) = D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{\theta})||p_{\boldsymbol{\alpha}}(\boldsymbol{\theta}|\mathbf{X})) + \mathcal{L}(\boldsymbol{\phi}; \mathbf{X}) \tag{2.78}$$

where the first RHS term denotes a KL divergence of the approximate from the true posterior, and where $\mathcal{L}(\boldsymbol{\phi}; \mathbf{X})$ denotes the variational lower bound to the marginal likelihood:

$$\mathcal{L}(\boldsymbol{\phi}; \mathbf{X}) = \int q_{\boldsymbol{\phi}}(\boldsymbol{\theta}) \left(\log p_{\boldsymbol{\theta}}(\mathbf{X}) + \log p_{\boldsymbol{\alpha}}(\boldsymbol{\theta}) - \log q_{\boldsymbol{\phi}}(\boldsymbol{\theta}) \right) d\boldsymbol{\theta} \tag{2.79}$$

Note that this is a lower bound since the KL divergence is non-negative; the bound equals the true marginal when the approximate and true posteriors match exactly. The term $\log p_{\boldsymbol{\theta}}(\mathbf{X})$ is composed of a sum over the marginal likelihoods of individual datapoints $\log p_{\boldsymbol{\theta}}(\mathbf{X}) = \sum_{i=1}^{N} \log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})$, which can each be rewritten as:

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) = D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}^{(i)})) + \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)}) \tag{2.80}$$

where again the first RHS term is the KL divergence of the approximate from the true posterior, and $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x})$ is the variational lower bound of the marginal likelihood of datapoint i:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)}) = \int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \left(\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z}) + \log p_{\boldsymbol{\theta}}(\mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \right) d\mathbf{z} \tag{2.81}$$

The expectations on the right-hand side of equations (2.79) and (2.81) can obviously be written as a sum of three separate expectations, of which

the second and third component can sometimes be analytically solved, e.g. when both $p_{\theta}(\mathbf{x})$ and $q_{\phi}(\mathbf{z}|\mathbf{x})$ are Gaussian. For generality we will here assume that each of these expectations is intractable.

Under certain mild conditions, which we outlined, for chosen approximate posteriors $q_{\phi}(\theta)$ and $q_{\phi}(\mathbf{z}|\mathbf{x})$ we can reparameterize conditional samples $\widetilde{\mathbf{z}} \sim q_{\phi}(\mathbf{z}|\mathbf{x})$ as

$$\widetilde{\mathbf{z}} = g_{\phi}(\epsilon, \mathbf{x}) \quad \text{with} \quad \epsilon \sim p(\epsilon) \tag{2.82}$$

where we choose a prior $p(\epsilon)$ and a function $g_{\phi}(\epsilon, \mathbf{x})$ such that the following holds:

$$\begin{aligned}\mathcal{L}(\theta, \phi; \mathbf{x}^{(i)}) &= \int q_{\phi}(\mathbf{z}|\mathbf{x}) \left(\log p_{\theta}(\mathbf{x}^{(i)}|\mathbf{z}) + \log p_{\theta}(\mathbf{z}) - \log q_{\phi}(\mathbf{z}|\mathbf{x})\right) d\mathbf{z} \\ &= \int p(\epsilon) \left(\log p_{\theta}(\mathbf{x}^{(i)}|\mathbf{z}) + \log p_{\theta}(\mathbf{z}) - \log q_{\phi}(\mathbf{z}|\mathbf{x})\right) \Big|_{\mathbf{z}=g_{\phi}(\epsilon, \mathbf{x}^{(i)})} d\epsilon\end{aligned} \tag{2.83}$$

The same can be done for the approximate posterior $q_{\phi}(\theta)$:

$$\widetilde{\theta} = h_{\phi}(\zeta) \quad \text{with} \quad \zeta \sim p(\zeta) \tag{2.84}$$

where we, similarly as above, choose a prior $p(\zeta)$ and a function $h_{\phi}(\zeta)$ such that the following holds:

$$\begin{aligned}\mathcal{L}(\phi; \mathbf{X}) &= \int q_{\phi}(\theta) \left(\log p_{\theta}(\mathbf{X}) + \log p_{\alpha}(\theta) - \log q_{\phi}(\theta)\right) d\theta \\ &= \int p(\zeta) \left(\log p_{\theta}(\mathbf{X}) + \log p_{\alpha}(\theta) - \log q_{\phi}(\theta)\right) \Big|_{\theta=h_{\phi}(\zeta)} d\zeta\end{aligned} \tag{2.85}$$

For notational conciseness we introduce a shorthand notation $f_{\phi}(\mathbf{x}, \mathbf{z}, \theta)$:

$$\begin{aligned}&f_{\phi}(\mathbf{x}, \mathbf{z}, \theta) \\ &= N \cdot (\log p_{\theta}(\mathbf{x}|\mathbf{z}) + \log p_{\theta}(\mathbf{z}) - \log q_{\phi}(\mathbf{z}|\mathbf{x})) + \log p_{\alpha}(\theta) - \log q_{\phi}(\theta)\end{aligned} \tag{2.86}$$

The Monte Carlo estimate of the variational lower bound, given datapoint $\mathbf{x}^{(i)}$, is:

$$\mathcal{L}(\phi; \mathbf{X}) \simeq \frac{1}{L} \sum_{l=1}^{L} f_{\phi}(\mathbf{x}^{(l)}, g_{\phi}(\epsilon^{(l)}, \mathbf{x}^{(l)}), h_{\phi}(\zeta^{(l)})) \tag{2.87}$$

where $\boldsymbol{\epsilon}^{(l)} \sim p(\boldsymbol{\epsilon})$ and $\zeta^{(l)} \sim p(\boldsymbol{\zeta})$. The estimator only depends on samples from $p(\boldsymbol{\epsilon})$ and $p(\boldsymbol{\zeta})$ which are obviously not influenced by $\boldsymbol{\phi}$, therefore the estimator can be differentiated w.r.t. $\boldsymbol{\phi}$. The resulting stochastic gradients can be used in conjunction with stochastic optimization methods such as SGD or Adagrad Duchi et al. [2010].

2.C.1 *Example*

Let the prior over the parameters and latent variables be the centered isotropic Gaussian $p_{\boldsymbol{\alpha}}(\boldsymbol{\theta}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$ and $p_{\boldsymbol{\theta}}(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$. Note that in this case, the prior lacks parameters. Let's also assume that the true posteriors are approximately Gaussian with an approximately diagonal covariance. In this case, we can let the variational approximate posteriors be multivariate Gaussians with a diagonal covariance structure:

$$\begin{aligned} \log q_{\boldsymbol{\phi}}(\boldsymbol{\theta}) &= \log \mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu}_{\boldsymbol{\theta}}, \sigma^2_{\boldsymbol{\theta}} \mathbf{I}) \\ \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) &= \log \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}_{\mathbf{z}}, \sigma^2_{\mathbf{z}} \mathbf{I}) \end{aligned} \tag{2.88}$$

where $\boldsymbol{\mu}_{\mathbf{z}}$ and $\boldsymbol{\sigma}_{\mathbf{z}}$ are yet unspecified functions of $\mathbf{x}$. Since they are Gaussian, we can parameterize the variational approximate posteriors:

$$\begin{aligned} q_{\boldsymbol{\phi}}(\boldsymbol{\theta}) \quad &\text{as} \quad \widetilde{\boldsymbol{\theta}} = \boldsymbol{\mu}_{\boldsymbol{\theta}} + \boldsymbol{\sigma}_{\boldsymbol{\theta}} \odot \boldsymbol{\zeta} \qquad \text{where} \quad \boldsymbol{\zeta} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \\ q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \quad &\text{as} \quad \widetilde{\mathbf{z}} = \boldsymbol{\mu}_{\mathbf{z}} + \boldsymbol{\sigma}_{\mathbf{z}} \odot \boldsymbol{\epsilon} \qquad \text{where} \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \end{aligned}$$

With $\odot$ we signify an element-wise product. These can be plugged into the lower bound defined above (eqs (2.86) and (2.87)).

In this case it is possible to construct an alternative estimator with a lower variance, since in this model $p_{\boldsymbol{\alpha}}(\boldsymbol{\theta})$, $p_{\boldsymbol{\theta}}(\mathbf{z})$, $q_{\boldsymbol{\phi}}(\boldsymbol{\theta})$ and $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ are Gaussian, and therefore four terms of $f_{\boldsymbol{\phi}}$ can be solved analytically. The resulting estimator is:

$$\begin{aligned} &\mathcal{L}(\boldsymbol{\phi}; \mathbf{X}) \\ &\simeq \frac{1}{L} \sum_{l=1}^{L} N \cdot \left(\frac{1}{2} \sum_{j=1}^{J} \left(1 + \log((\sigma^{(l)}_{\mathbf{z},j})^2) - (\mu^{(l)}_{\mathbf{z},j})^2 - (\sigma^{(l)}_{\mathbf{z},j})^2 \right) + \log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)} \mathbf{z}^{(i)}) \right) \\ &+ \frac{1}{2} \sum_{j=1}^{J} \left(1 + \log((\sigma^{(l)}_{\boldsymbol{\theta},j})^2) - (\mu^{(l)}_{\boldsymbol{\theta},j})^2 - (\sigma^{(l)}_{\boldsymbol{\theta},j})^2 \right) \end{aligned} \tag{2.89}$$

$\mu^{(i)}_j$ and $\sigma^{(i)}_j$ simply denote the j-th element of vectors $\boldsymbol{\mu}^{(i)}$ and $\boldsymbol{\sigma}^{(i)}$.

2.D APPLICATIONS

Some important applications of variational autoencoders are:

- Representation learning: learning better representations of the data. Some uses of this are data-efficient learning, such as semi-supervised learning, and visualization of data as low-dimensional manifolds.

- Generative modeling: artificial creativity by plausible interpolation between data and extrapolation from data.

Here we will review some applications of VAEs.

2.D.1 *Representation Learning*

In case of *supervised learning*, we typically aim to learn a conditional distribution: to predict the distribution over the possible values of a variable, given the value of some another variable. One such problem is that of image classification: given an image, the prediction of a distribution over the possible class labels. Through the yearly ImageNet competition [Russakovsky et al., 2015], it has become clear that deep convolutional neural networks [LeCun et al., 1998, Goodfellow et al., 2016] (CNNs), *given a large amount of labeled images*, are extraordinarily good at solving the image classification task. Modern versions of CNNs based on residual networks, which is a variant of LSTM-type neural networks [Hochreiter and Schmidhuber, 1997], now arguably achieves human-level classification accuracy on this task [He et al., 2015b,a].

When the number of labeled examples is low, solutions found with purely supervised approaches tend to exhibit poor generalization to new data. In such cases, generative models can be employed as an effective type of regularization. One particular strategy, presented in Kingma et al. [2014], is to optimize the classification model jointly with a variational autoencoder over the input variables, sharing parameters between the two. The variational autoencoder, in this case, provides an auxiliary objective, improving the data efficiency of the classification solution. Through sharing of statistical strength between modeling problems, this can greatly improve upon the supervised classification error. Techniques based on VAEs are now among state of the art for semi-supervised classification [Maaløe et al., 2016], with

on average under 1% classification error in the MNIST classification problem, when trained with only 10 labeled images per class, i.e. when more than 99.8% of the labels in the training set were removed. In concurrent work [Rezende et al., 2016], it was shown that VAE-based semi-supervised learning can even do well when only a single sample per class is presented.

A standard supervised approach, *GoogLeNet* [Szegedy et al., 2015], which normally achieves near state-of-the-art performance on the ImageNet validation set, achieves only around 5% top-1 classification accuracy when trained with only 1% of the labeled images, as shown by Pu et al. [2016]. In contrast, they show that a semi-supervised approach with VAEs achieves around 45% classification accuracy on the same task, when modeling the labels jointly with the labeled and unlabeled input images.

2.D.2 *Understanding of data, and artificial creativity*

Generative models with latent spaces allow us to transform the data into a simpler latent space, explore it in that space, and understand it better. A related branch of applications of deep generative models is the synthesis of plausible pseudo-data with certain desirable properties, sometimes coined as *artificial creativity*.

2.D.2.1 *Chemical Design*

One example of a recent scientific application of artificial creativity, is shown in Gómez-Bombarelli et al. [2016]. In this paper, a fairly straightforward VAE is trained on hundreds of thousands of existing chemical structures. The resulting continuous representation (latent space) is subsequently used to perform gradient-based optimization towards certain properties; the method is demonstrated on the design of drug-like molecules and organic light-emitting diodes. See figure 2.D.1.

2.D.2.2 *Natural Language Synthesis*

A similar approach was used to generating natural-language sentences from a continuous space by Bowman et al. [2015]. In this paper, it is shown how a VAE can be successfully trained on text. The model is shown to successfully interpolate between sentences, and for imputation of missing words. See figure 2.D.2.

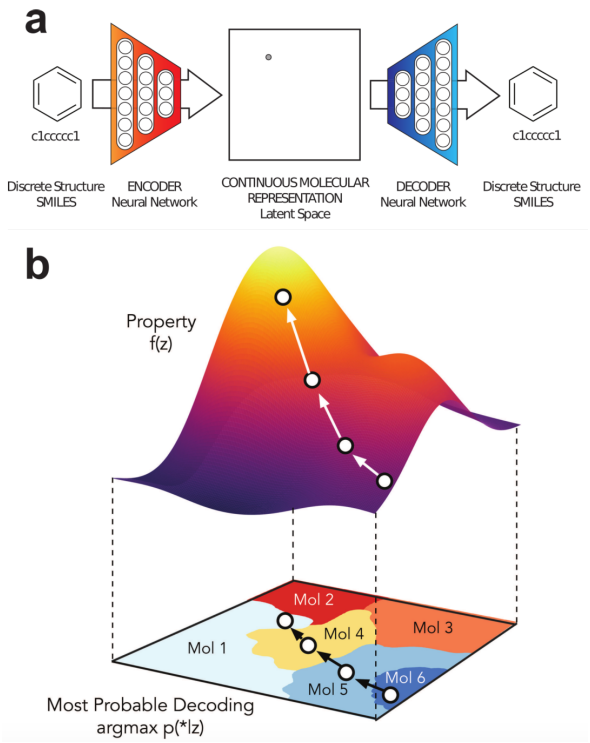


Figure 2.D.1: Example application of a VAE in [Gómez-Bombarelli et al., 2016]: design of new molecules with desired chemical properties. (a) A latent continuous representation $\mathbf{z}$ of molecules is learned on a large dataset of molecules. (b) This continuous representation enables gradient-based search of new molecules that maximizes some chosen desired chemical property given by objective function $f(\mathbf{z})$.

“ i want to talk to you . ”
“i want to be with you . ”
“i do n’t want to be with you . ”
i do n’t want to be with you .
she did n’t want to be with him .
he was silent for a long moment .
he was silent for a moment .
it was quiet for a moment .
it was dark and cold .
there was a pause .
it was my turn .

Figure 2.D.2: An application of VAEs to interpolation between pairs of sentences, from [Bowman et al., 2015]. The intermediate sentences are grammatically correct, and the topic and syntactic structure are typically locally consistent.

2.D.2.3 *Astronomy*

In [Ravanbakhsh et al., 2016], VAEs are applied to simulate observations of distant galaxies. This helps with the calibration of systems that need to indirectly detect the shearing of observations of distant galaxies, caused by weak gravitational lensing in the presence of dark matter between earth and those galaxies. Since the lensing effects are so weak, such systems need to be calibrated with ground-truth images with a known amount of shearing. Since real data is still limited, the proposed solution is to use deep generative models for synthesis of pseudo-data.

2.D.2.4 *Image (Re-)Synthesis*

A popular application is image (re)synthesis. One can optimize a VAE to form a generative model over images. One can synthesize images from the generative model, but the inference model (or *encoder*) also allows one to encode real images into a latent space. One can modify the encoding in this latent space, then decode the image back into the observed space. Relatively simply transformations in the observed space, such as linear transformations, often translate into semantically meaningful modifications of the original image. One example, as demonstrated by White [2016], is the modification

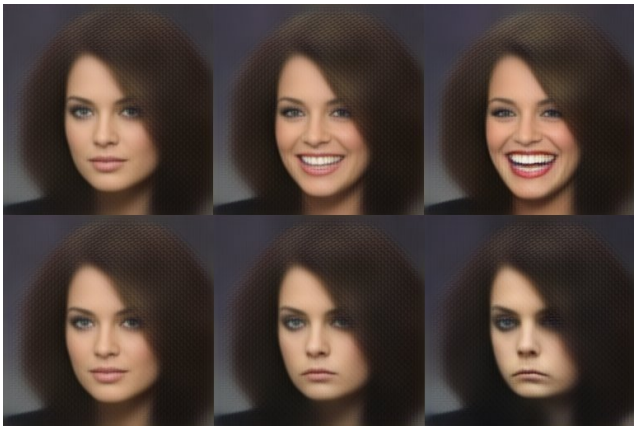


Figure 2.D.3: VAEs can be used for image re-synthesis. In this example by White [2016], an original image (left) is modified in a latent space in the direction of a *smile vector*, producing a range of versions of the original, from smiling to sadness. Notice how changing the image along a single vector in latent space, modifies the image in many subtle and less-subtle ways in pixel space.

of images in latent space along a "smile vector" in order to make them more happy, or more sad looking. See figure 2.D.3 for an example.

3

SEMI-SUPERVISED LEARNING WITH DEEP GENERATIVE MODELS

The ever-increasing size of modern data sets combined with the difficulty of obtaining label information has made semi-supervised learning one of the problems of significant practical importance in modern data analysis. We revisit the approach to semi-supervised learning with generative models and develop new models that allow for effective generalization from small labeled data sets to large unlabeled ones. Generative approaches have thus far been either inflexible, inefficient or non-scalable. We show that deep generative models and approximate Bayesian inference exploiting recent advances in variational methods can be used to provide significant improvements, making generative approaches highly competitive for semi-supervised learning.[1]

3.1 INTRODUCTION

Semi-supervised learning considers the problem of classification when only a small subset of the observations have corresponding class labels. Such problems are of immense practical interest in a wide range of applications, including image search [Fergus et al., 2009], genomics [Shi and Zhang, 2011], natural language parsing [Liang, 2005], and speech analysis [Liu and Kirchhoff, 2013], where unlabeled data is abundant, but obtaining class labels is expensive or impossible to obtain for the entire data set. The question that is then asked is: how can properties of the data be used to improve decision boundaries and to allow for classification that is more accurate than that based on classifiers constructed using the labeled data alone. Here, we an-

[1] This chapter is adapted from our publication on semi-supervised learning with VAEs, published at NIPS 2014 [Kingma et al., 2014].

swer this question by developing probabilistic models for semi-supervised learning by utilizing an explicit model of the data density, building upon recent advances in deep generative models and scalable variational inference [Kingma and Welling, 2013, Rezende et al., 2014].

Amongst existing approaches, the simplest algorithm for semi-supervised learning is based on a *self-training* scheme [Rosenberg et al., 2005] where the the model is bootstrapped with additional labeled data obtained from its own highly confident predictions; this process being repeated until some termination condition is reached. These methods are heuristic and prone to error since they can reinforce poor predictions. *Transductive SVMs* (TSVM) [Joachims, 1999] extend SVMs with the aim of max-margin classification while ensuring that there are as few unlabeled observations near the margin as possible. These approaches have difficulty extending to large amounts of unlabeled data, and efficient optimization in this setting is still an open problem. *Graph-based methods* are amongst the most popular and aim to construct a graph connecting similar observations; label information propagates through the graph from labeled to unlabeled nodes by finding the minimum energy (MAP) configuration [Blum et al., 2004, Zhu et al., 2003, Belkin and Adviser-Niyogi, 2003]. Modern approaches have shown good results on label imputation tasks [Yang et al., 2012]. Graph-based approaches are sensitive to the graph structure and require eigen-analysis of the graph Laplacian, which limits the scale to which these methods can be applied – though efficient spectral methods are now available [Fergus et al., 2009].

Neural network-based approaches combine unsupervised and supervised learning by training feed-forward classifiers with an additional penalty from an auto-encoder or other unsupervised embedding of the data [Ranzato and Szummer, 2008, Weston et al., 2012]. The Manifold Tangent Classifier (MTC) [Rifai et al., 2011] trains contrastive auto-encoders (CAEs) to learn the manifold on which the data lies, followed by an instance of TangentProp to train a classifier that is approximately invariant to local perturbations along the manifold. The idea of manifold learning using graph-based methods has most recently been combined with kernel (SVM) methods in the Atlas RBF model [Pitelis et al., 2014] and provides amongst most competitive performance currently available.

Here, we instead choose to exploit the power of *generative models*, which recognize the semi-supervised learning problem as a specialized missing data imputation task for the classification problem. Existing generative approaches based on models such as Gaussian mixture or hidden Markov mod-

els [Zhu, 2006], have not been very successful due to the need for a large number of mixtures components or states to perform well. More recent solutions have used non-parametric density models, either based on trees [Kemp et al., 2003] or Gaussian processes [Adams and Ghahramani, 2009], but scalability and accurate inference for these approaches is still lacking. Variational approximations for semi-supervised clustering have also been explored previously [Wang et al., 2009, Li et al., 2009].

Thus, while a small set of generative approaches have been previously explored, a generalized and scalable probabilistic approach for semi-supervised learning is still lacking. It is this gap that we address through the following contributions:

- We describe a new framework for semi-supervised learning with generative models, employing rich parametric density estimators formed by the fusion of probabilistic modeling and deep neural networks.
- We show for the first time how variational inference can be brought to bear upon the problem of semi-supervised classification.
- We show qualitatively generative semi-supervised models learn to separate the data classes (content types) from the intra-class variabilities (styles), allowing in a very straightforward fashion to simulate analogies of images on a variety of datasets.

3.2 DEEP GENERATIVE MODELS FOR SEMI-SUPERVISED LEARNING

We are faced with data that appear as pairs $(\mathbf{X}, \mathbf{Y}) = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})\}$, with the i-th observation $\mathbf{x}^{(i)} \in \mathbb{R}^D$ and the corresponding class label $y^{(i)} \in \{1, \ldots, L\}$. Observations will have corresponding latent variables, which we denote by $\mathbf{z}^{(i)}$. We will omit the index i whenever it is clear that we are referring to terms associated with a single data point. In semi-supervised classification, only a subset of the observations have corresponding class labels; we refer to the empirical distribution over the labeled and unlabeled subsets as $\widetilde{p}_l(\mathbf{x}, y)$ and $\widetilde{p}_u(\mathbf{x})$, respectively. We now develop models for semi-supervised learning that exploit generative descriptions of the data to improve upon the classification performance that would be obtained using the labeled data alone.

3.2.1 *Latent-feature discriminative model (M1)*

A commonly used approach is to construct a model that provides an embedding or feature representation of the data. Using these features, a separate classifier is thereafter trained. The embeddings allow for a clustering of related observations in a latent feature space that allows for accurate classification, even with a limited number of labels. Instead of a linear embedding, or features obtained from a regular auto-encoder, we construct a deep generative model of the data that is able to provide a more robust set of latent features. The generative model we use is:

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I}); \qquad p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) = f(\mathbf{x}; \mathbf{z}, \boldsymbol{\theta}), \tag{3.1}$$

where $f(\mathbf{x}; \mathbf{z}, \boldsymbol{\theta})$ is a suitable likelihood function (e.g., a Gaussian or Bernoulli distribution) whose probabilities are formed by a non-linear transformation, with parameters $\boldsymbol{\theta}$, of a set of latent variables $\mathbf{z}$. This non-linear transformation is essential to allow for higher moments of the data to be captured by the density model, and we choose these non-linear functions to be deep neural networks. Approximate samples from the posterior distribution over the latent variables $p(\mathbf{z}|\mathbf{x})$ are used as features to train a classifier that predicts class labels y, such as a (transductive) SVM or multinomial regression. Using this approach, we can now perform classification in a lower dimensional space since we typically use latent variables whose dimensionality is much less than that of the observations. These low dimensional embeddings should now also be more easily separable since we make use of independent latent Gaussian posteriors whose parameters are formed by a sequence of non-linear transformations of the data. This simple approach results in improved performance for SVMs, as we demonstrate this in this chapter.

3.2.2 *Generative semi-supervised model (M2)*

We propose a probabilistic model that describes the data as being generated by a latent class variable y in addition to a continuous latent variable $\mathbf{z}$. The data is explained by the generative process:

$$p(y) = \text{Cat}(y|\boldsymbol{\pi}); \qquad p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I}); \qquad p_{\boldsymbol{\theta}}(\mathbf{x}|y, \mathbf{z}) = f(\mathbf{x}; y, \mathbf{z}, \boldsymbol{\theta}), \tag{3.2}$$

where $\text{Cat}(y|\boldsymbol{\pi})$ is the multinomial distribution, the class labels y are treated as latent variables if no class label is available and $\mathbf{z}$ are additional latent

variables. These latent variables are marginally independent and allow us, in case of digit generation for example, to separate the class specification from the writing style of the digit. As before, $f(\mathbf{x}; y, \mathbf{z}, \boldsymbol{\theta})$ is a suitable likelihood function, e.g., a Bernoulli or Gaussian distribution, parameterized by a non-linear transformation of the latent variables. In our experiments, we choose deep neural networks as this non-linear function. Since most labels y are unobserved, we integrate over the class of any unlabeled data during the inference process, thus performing classification as inference. Predictions for any missing labels are obtained from the inferred posterior distribution $p_{\boldsymbol{\theta}}(y|\mathbf{x})$. This model can also be seen as a hybrid continuous-discrete mixture model where the different mixture components share parameters.

3.2.3 *Stacked generative semi-supervised model (M1+M2)*

We can combine these two approaches by first learning a new latent representation $\mathbf{z}_1$ using the generative model from M1, and subsequently learning a generative semi-supervised model M2, using embeddings from $\mathbf{z}_1$ instead of the raw data $\mathbf{x}$. The result is a deep generative model with two layers of stochastic variables: $p_{\boldsymbol{\theta}}(\mathbf{x}, y, \mathbf{z}_1, \mathbf{z}_2) = p(y)p(\mathbf{z}_2)p_{\boldsymbol{\theta}}(\mathbf{z}_1|y, \mathbf{z}_2)p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}_1)$, where the priors $p(y)$ and $p(\mathbf{z}_2)$ equal those of y and $\mathbf{z}$ above, and both $p_{\boldsymbol{\theta}}(\mathbf{z}_1|y, \mathbf{z}_2)$ and $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}_1)$ are parameterized as deep neural networks.

3.3 SCALABLE VARIATIONAL INFERENCE

3.3.1 *Lower Bound Objective*

In all our models, computation of the exact posterior distribution is intractable due to the nonlinear, non-conjugate dependencies between the random variables. To allow for tractable and scalable inference and parameter learning, we exploit recent advances in variational inference [Kingma and Welling, 2013, Rezende et al., 2014], also introduced in the preceding chapters. For all the models described, we introduce a fixed-form distribution $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ with parameters $\boldsymbol{\phi}$ that approximates the true posterior distribution $p(\mathbf{z}|\mathbf{x})$. We then follow the variational principle to derive a lower bound on the marginal likelihood of the model – this bound forms our objective function and ensures that our approximate posterior is as close as possible to the true posterior.

We construct the approximate posterior distribution $q_{\boldsymbol{\phi}}(\cdot)$ as an inference or recognition model, which has become a popular approach for efficient variational inference [Kingma and Welling, 2013, Rezende et al., 2014, Dayan, 2000, Stuhlmüller et al., 2013]. Using an inference network, we avoid the need to compute per data point variational parameters, but can instead compute a set of global variational parameters $\boldsymbol{\phi}$. This allows us to amortize the cost of inference by generalizing between the posterior estimates for all latent variables through the parameters of the inference network, and allows for fast inference at both training and testing time (unlike with VEM, in which we repeat the generalized E-step optimization for every test data point). An inference network is introduced for all latent variables, and we parameterize them as deep neural networks whose outputs form the parameters of the distribution $q_{\phi}(\cdot)$. For the latent-feature discriminative model (M1), we use a Gaussian inference network $q_{\phi}(\mathbf{z}|\mathbf{x})$ for the latent variable $\mathbf{z}$. For the generative semi-supervised model (M2), we introduce an inference model for each of the latent variables $\mathbf{z}$ and y, which we we assume has a factorized form $q_{\phi}(\mathbf{z}, y|\mathbf{x}) = q_{\phi}(\mathbf{z}|\mathbf{x})q_{\phi}(y|\mathbf{x})$, specified as Gaussian and multinomial distributions respectively.

$$\text{M1: } q_{\phi}(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_{\phi}(\mathbf{x}), \text{diag}(\boldsymbol{\sigma}^2_{\phi}(\mathbf{x}))), \tag{3.3}$$

$$\text{M2: } q_{\phi}(\mathbf{z}|y, \mathbf{x}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_{\phi}(y, \mathbf{x}), \text{diag}(\boldsymbol{\sigma}^2_{\phi}(y, \mathbf{x}))), \tag{3.4}$$

$$q_{\phi}(y|\mathbf{x}) = \text{Cat}(y|\boldsymbol{\pi}_{\phi}(\mathbf{x})), \tag{3.5}$$

where $\boldsymbol{\sigma}_{\phi}(\mathbf{x})$ is a vector of standard deviations, $\boldsymbol{\pi}_{\phi}(\mathbf{x})$ is a probability vector, and the functions $\boldsymbol{\mu}_{\phi}(\mathbf{x})$, $\boldsymbol{\sigma}_{\phi}(\mathbf{x})$ and $\boldsymbol{\pi}_{\phi}(\mathbf{x})$ are represented as MLPs.

3.3.1.1 *Latent Feature Discriminative Model Objective*

For this model, the variational bound $\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x})$ on the marginal likelihood for a single data point is:

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) \geq \mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\right] - KL[q_{\phi}(\mathbf{z}|\mathbf{x})\|p_{\boldsymbol{\theta}}(\mathbf{z})] = \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) \tag{3.6}$$

The inference network $q_{\phi}(\mathbf{z}|\mathbf{x})$ (3.3) is used during training of the model using both the labeled and unlabeled data sets. This approximate posterior is then used as a feature extractor for the labeled data set, and the features used for training the classifier.

3.3.1.2 *Generative Semi-supervised Model Objective*

For this model, we have two cases to consider. In the first case, the label corresponding to a data point is observed and the variational bound is a simple extension of equation (3.6):

$$\begin{aligned}\log p_{\boldsymbol{\theta}}(\mathbf{x}, y) &\geq \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x},y)}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}|y,\mathbf{z}) + \log p_{\boldsymbol{\theta}}(y) + \log p(\mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x},y)\right] \\ &= \mathcal{L}(\mathbf{x}, y),\end{aligned} \tag{3.7}$$

For the case where the label is missing, it is treated as a latent variable over which we perform posterior inference and the resulting bound for handling data points with an unobserved label y is:

$$\begin{aligned}\log p_{\boldsymbol{\theta}}(\mathbf{x}) &\geq \mathbb{E}_{q_{\boldsymbol{\phi}}(y,\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}|y,\mathbf{z}) + \log p_{\boldsymbol{\theta}}(y) + \log p(\mathbf{z}) - \log q_{\boldsymbol{\phi}}(y,\mathbf{z}|\mathbf{x})\right] \\ &= \textstyle\sum_y q_{\boldsymbol{\phi}}(y|\mathbf{x})(\mathcal{L}(\mathbf{x}, y)) + \mathcal{H}(q_{\boldsymbol{\phi}}(y|\mathbf{x})) \\ &= \mathcal{U}(\mathbf{x}).\end{aligned} \tag{3.8}$$

The bound on the marginal likelihood for the entire dataset is now:

$$\mathcal{J} = \textstyle\sum_{(\mathbf{x},y)\sim\widetilde{p}_l} \mathcal{L}(\mathbf{x}, y) + \sum_{\mathbf{x}\sim\widetilde{p}_u} \mathcal{U}(\mathbf{x}) \tag{3.9}$$

The distribution $q_{\boldsymbol{\phi}}(y|\mathbf{x})$ (3.5) for the missing labels has the form a discriminative classifier, and we can use this knowledge to construct the best classifier possible as our inference model. This distribution is also used at test time for predictions of any unseen data.

In the objective function (3.9), the label predictive distribution $q_{\boldsymbol{\phi}}(y|\mathbf{x})$ contributes only to the second term relating to the unlabeled data, which is an undesirable property if we wish to use this distribution as a classifier. Ideally, all model and variational parameters should learn in all cases. To remedy this, we add a classification loss to (3.9), such that the distribution $q_{\boldsymbol{\phi}}(y|\mathbf{x})$ also learns from labeled data. The extended objective function is:

$$\mathcal{J}^{\alpha} = \mathcal{J} + \alpha \cdot \mathbb{E}_{\widetilde{p}_l(\mathbf{x},y)}\left[-\log q_{\boldsymbol{\phi}}(y|\mathbf{x})\right], \tag{3.10}$$

where the hyper-parameter α controls the relative weight between generative and purely discriminative learning. We use $\alpha = 0.1 \cdot N$ in all experiments. While we have obtained this objective function by motivating the need for all model components to learn at all times, the objective 3.10 can also be derived directly using the variational principle by instead performing inference over the parameters $\boldsymbol{\pi}$ of the categorical distribution, using a symmetric Dirichlet prior over these parameters.

3.3.2 *Optimization*

The bounds in equations (3.6) and (3.10) provide a unified objective function for optimization of both the parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ of the generative and inference models, respectively. This optimization can be done jointly, without resort to the variational EM algorithm, by using deterministic reparameterizations of the expectations in the objective function, combined with Monte Carlo approximation – referred to as *stochastic gradient variational Bayes* (SGVB) [Kingma and Welling, 2013] or as *stochastic backpropagation* [Rezende et al., 2014]. We describe the core strategy for the latent-feature discriminative model M1, since the same computations are used for the generative semi-supervised model.

When the prior $p(\mathbf{z})$ is a spherical Gaussian distribution $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$ and the variational distribution $q_\phi(\mathbf{z}|\mathbf{x})$ is also a Gaussian distribution as in (3.3), the KL term in equation (3.6) can be computed analytically and the log-likelihood term can be rewritten, using the location-scale transformation for the Gaussian distribution, as:

$$\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] = \mathbb{E}_{\mathcal{N}(\boldsymbol{\epsilon}|\mathbf{0},\mathbf{I})}\left[\log p_\theta(\mathbf{x}|\boldsymbol{\mu}_\phi(\mathbf{x}) + \boldsymbol{\sigma}_\phi(\mathbf{x}) \odot \boldsymbol{\epsilon})\right], \quad (3.11)$$

where $\odot$ indicates the element-wise product. While the expectation (3.11) still cannot be solved analytically, its gradients with respect to the generative parameters $\boldsymbol{\theta}$ and variational parameters $\boldsymbol{\phi}$ can be efficiently computed as expectations of simple gradients:

$$\nabla_{\{\theta,\phi\}}\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] = \mathbb{E}_{\mathcal{N}(\boldsymbol{\epsilon}|\mathbf{0},\mathbf{I})}\left[\nabla_{\{\theta,\phi\}} \log p_\theta(\mathbf{x}|\boldsymbol{\mu}_\phi(\mathbf{x}) + \boldsymbol{\sigma}_\phi(\mathbf{x}) \odot \boldsymbol{\epsilon})\right]. \quad (3.12)$$

This is also referred to as the reparameterization trick.

The gradients of the loss (3.10) for model M2 can be computed by a direct application of the chain rule and by noting that the conditional bound $\mathcal{L}(\mathbf{x}, y)$ contains the same type of terms as the loss (3.10). The gradients of the latter can then be efficiently estimated using (3.12) .

During optimization we use the estimated gradients in conjunction with standard stochastic gradient-based optimization methods such as SGD, RMSprop or AdaGrad [Duchi et al., 2010]. This results in parameter updates of the form: $(\boldsymbol{\theta}^{t+1}, \boldsymbol{\phi}^{t+1}) \leftarrow (\boldsymbol{\theta}^t, \boldsymbol{\phi}^t) + \boldsymbol{\Gamma}^t(\mathbf{g}^t_\theta, \mathbf{g}^t_\phi)$, where $\boldsymbol{\Gamma}$ is a diagonal preconditioning matrix that adaptively scales the gradients for faster minimization. The training procedure for models M1 and M2 are summarized in

algorithms 3 and 4, respectively. Our experimental results were obtained using AdaGrad.

3.3.3 *Computational Complexity*

The overall algorithmic complexity of a single joint update of the parameters $(\boldsymbol{\theta}, \boldsymbol{\phi})$ for M1 using the estimator (3.12) is $C_{\mathbf{M1}} = MSC_{\text{MLP}}$ where M is the minibatch size used, S is the number of samples of the random variate $\boldsymbol{\epsilon}$, and C_{MLP} is the cost of an evaluation of the MLPs in the conditional distributions $p_\theta(\mathbf{x}|\mathbf{z})$ and $q_\phi(\mathbf{z}|\mathbf{x})$. The cost C_{MLP} is of the form $O(KD^2)$ where K is the total number of layers and D is the average dimension of the layers of the MLPs in the model. Training M1 also requires training a supervised classifier, whose algorithmic complexity, if it is a neural net, it will have a complexity of the form C_{MLP} .

The algorithmic complexity for M2 is of the form $C_{\text{M2}} = LC_{\text{M1}}$, where L is the number of labels and C_{M1} is the cost of evaluating the gradients of each conditional bound, which is the same as for M1. The stacked generative semi-supervised model has an algorithmic complexity of the form $C_{\mathbf{M1}} + C_{\mathbf{M2}}$. But with the advantage that the cost $C_{\mathbf{M2}}$ is calculated in a low-dimensional space (formed by the latent variables of the model M1 that provides the embeddings).

These complexities make this approach extremely appealing, since they are no more expensive than alternative approaches based on auto-encoder or neural models, which have the lowest computational complexity amongst existing competitive approaches. In addition, our models are fully probabilistic, allowing for a wide range of inferential queries, which is not possible with many alternative approaches for semi-supervised learning.

3.4 EXPERIMENTAL RESULTS

Open source code, with which the most important results and figures can be reproduced, is available at `http://github.com/dpkingma/nips14-ssl`.

3.4.1 *Benchmark Classification*

We test performance on the standard MNIST digit classification benchmark. The data set for semi-supervised learning is created by splitting the 50,000 training points between a labeled and unlabeled set, and varying the size of the labeled from 100 to 3000. We ensure that all classes are balanced when

Algorithm 3: Learning in model M1

while *generativeTraining()* **do**
  $\mathcal{D} \leftarrow$ getRandomMiniBatch()
  $\mathbf{z}_i \sim q_\phi(\mathbf{z}_i|\mathbf{x}_i) \quad \forall \mathbf{x}_i \in \mathcal{D}$
  $\mathcal{J} \leftarrow \sum_n \mathcal{J}(\mathbf{x}_i)$
  $(\mathbf{g}_\theta, \mathbf{g}_\phi) \leftarrow (\frac{\partial \mathcal{J}}{\partial \theta}, \frac{\partial \mathcal{J}}{\partial \phi})$
  $(\boldsymbol{\theta}, \boldsymbol{\phi}) \leftarrow (\boldsymbol{\theta}, \boldsymbol{\phi}) + \boldsymbol{\Gamma}(\mathbf{g}_\theta, \mathbf{g}_\phi)$
end
while *discriminativeTraining()* **do**
  $\mathcal{D} \leftarrow$ getLabeledRandomMiniBatch()
  $\mathbf{z}_i \sim q_\phi(\mathbf{z}_i|\mathbf{x}_i) \quad \forall \{\mathbf{x}_i, y_i\} \in \mathcal{D}$
  trainClassifier($\{\mathbf{z}_i, y_i\}$)
end

Algorithm 4: Learning in model M2

while *training()* **do**
  $\mathcal{D} \leftarrow$ getRandomMiniBatch()
  $y_i \sim q_\phi(y_i|\mathbf{x}_i) \quad \forall \{\mathbf{x}_i, y_i\} \notin \mathcal{O}$
  $\mathbf{z}_i \sim q_\phi(\mathbf{z}_i|y_i, \mathbf{x}_i)$
  $\mathcal{J}^\alpha \leftarrow$ eq. (3.10)
  $(\mathbf{g}_\theta, \mathbf{g}_\phi) \leftarrow (\frac{\partial \mathcal{L}^\alpha}{\partial \theta}, \frac{\partial \mathcal{L}^\alpha}{\partial \phi})$
  $(\boldsymbol{\theta}, \boldsymbol{\phi}) \leftarrow (\boldsymbol{\theta}, \boldsymbol{\phi}) + \boldsymbol{\Gamma}(\mathbf{g}_\theta, \mathbf{g}_\phi)$
end

doing this, i.e. each class has the same number of labeled points. We create a number of data sets using randomized sampling to confidence bounds for the mean performance under repeated draws of data sets.

For model M1 we used a 50-dimensional latent variable $\mathbf{z}$. The MLPs that form part of the generative and inference models were constructed with two hidden layers, each with 600 hidden units, using softplus $\log(1 + e^x)$ activation functions. On top, a transductive SVM (TSVM) was learned on values of $\mathbf{z}$ inferred with $q_\phi(\mathbf{z}|\mathbf{x})$. For model M2 we also used 50-dimensional $\mathbf{z}$. In each experiment, the MLPs were constructed with one hidden layer, each with 500 hidden units and softplus activation functions. In case of SVHN and NORB, we found it helpful to pre-process the data with PCA.

Table 3.1: Benchmark results of semi-supervised classification on MNIST when trained with only N observed labels out of 50000. Shown are error rates (%) on the test-set of 10,000 images.

N	100	600	1000	3000
NN	25.81	11.44	10.7	6.04
CNN	22.98	7.68	6.45	3.35
TSVM	16.81	6.16	5.38	3.45
CAE	13.47	6.3	4.77	3.22
MTC	12.03	5.13	3.64	2.57
AtlasRBF	8.10 ($\pm$ 0.95)	–	3.68 ($\pm$ 0.12)	–
M1+TSVM (Ours)	11.82 ($\pm$ 0.25)	5.72 ($\pm$ 0.049	4.24 ($\pm$ 0.07)	3.49 ($\pm$ 0.04)
M2 (Ours)	11.97 ($\pm$ 1.71)	4.94 ($\pm$ 0.13)	3.60 ($\pm$ 0.56)	3.92 ($\pm$ 0.63)
M1+M2 (Ours)	**3.33** ($\pm$ 0.14)	**2.59** ($\pm$ 0.05)	**2.40** ($\pm$ 0.02)	**2.18** ($\pm$ 0.04)

This makes the model one level deeper, and still optimizes a lower bound on the likelihood of the unprocessed data.

Table 3.1 shows classification results. We compare to a broad range of existing solutions in semi-supervised learning, in particular to classification using nearest neighbors (NN), support vector machines on the labeled set (SVM), the transductive SVM (TSVM), and contractive auto-encoders (CAE). Some of the best results currently are obtained by the manifold tangent classifier (MTC) [Rifai et al., 2011] and the AtlasRBF method [Pitelis et al., 2014]. Unlike the other models in this comparison, our models are fully probabilistic but have a cost in the same order as these alternatives.

3.4.1.1 *Results*

The latent-feature discriminative model (M1) performs better than other models based on simple embeddings of the data, demonstrating the effectiveness of the latent space in providing robust features that allow for easier classification. By combining these features with a classification mechanism directly in the same model, as in the conditional generative model (M2), we are able to get similar results without a separate TSVM classifier.

However, by far the best results were obtained using the stack of models M1 and M2. This combined model provides accurate test-set predictions across all conditions, and easily outperforms the previously best methods. We also tested this deep generative model for supervised learning with all available labels, and obtain a test-set performance of 0.96%, which is among the best published results for this permutation-invariant MNIST classification task.

3.4.2 *Conditional Generation*

The conditional generative model can be used to explore the underlying structure of the data, which we demonstrate through two forms of analogical reasoning. Firstly, we demonstrate style and content separation by fixing the class label y, and then varying the latent variables $\mathbf{z}$ over a range of values. Figures 3.2 shows three MNIST classes in which, using a trained model with two latent variables, and the 2D latent variable varied over a range from -5 to 5. In all cases, we see that nearby regions of latent space correspond to similar writing styles, independent of the class; the left region represents upright writing styles, while the right-side represents slanted styles.

As a second approach, we use a test image and pass it through the inference network to infer a value of the latent variables corresponding to that image. We then fix the latent variables $\mathbf{z}$ to this value, vary the class label $\mathbf{y}$, and simulate images from the generative model corresponding to that combination of $\mathbf{z}$ and $\mathbf{y}$. This again demonstrate the disentanglement of style from class. Figure 3.1 shows these analogical fantasies for the MNIST and SVHN datasets [Netzer et al., 2011]. The SVHN data set is a far more complex data set than MNIST, but the model is able to fix the style of house number and vary the digit that appears in that style well. These generations represent the best current performance in simulation from generative models on these data sets.

The model used in this way also provides an alternative model to the stochastic feed-forward networks (SFNN) described by Tang and Salakhutdinov [2013]. The performance of our model significantly improves on SFNN, since instead of an inefficient Monte Carlo EM algorithm relying on importance sampling, we are able to perform efficient joint inference that is easy to scale.

3.4.3 *Image Classification*

We demonstrate the performance of image classification on the SVHN, and NORB image data sets. Since no comparative results in the semi-supervised setting exists, we perform nearest-neighbor and TSVM classification with RBF kernels and compare performance on features generated by our latent-feature discriminative model to the original features. The results are presented in tables 3.2 and 3.3, and we again demonstrate the effectiveness of our approach for semi-supervised classification.

Table 3.2: Semi-supervised classification on the SVHN dataset with 1000 labels. Shown are error rates (%) on the test set.

KNN	TSVM	M1+KNN	M1+TSVM	M1+M2
77.93 (± 0.08)	66.55 (± 0.10)	65.63 (± 0.15)	54.33 (± 0.11)	**36.02** (± 0.10)

Table 3.3: Semi-supervised classification on the NORB dataset with 1000 labels. Shown are error rates (%) on the test set.

KNN	TSVM	M1+KNN	M1+TSVM
78.71 (± 0.02)	26.00 (± 0.06)	65.39 (± 0.09)	**18.79** (± 0.05)

3.4.4 *Optimization details*

The parameters were initialized by sampling randomly from $\mathcal{N}(\mathbf{0}, 0.001^2\mathbf{I})$, except for the bias parameters which were initialized as 0. The objectives were optimized using minibatch gradient ascent until convergence, using a variant of RMSProp with momentum and initialization bias correction, a constant learning rate of 0.0003, first moment decay (momentum) of 0.1, and second moment decay of 0.001. For MNIST experiments, minibatches for training were generated by treating normalized pixel intensities of the images as Bernoulli probabilities and sampling binary images from this distribution. In the M2 model, a weight decay was used corresponding to a prior of $(\boldsymbol{\theta}, \boldsymbol{\phi}) \sim \mathcal{N}(0, I)$.

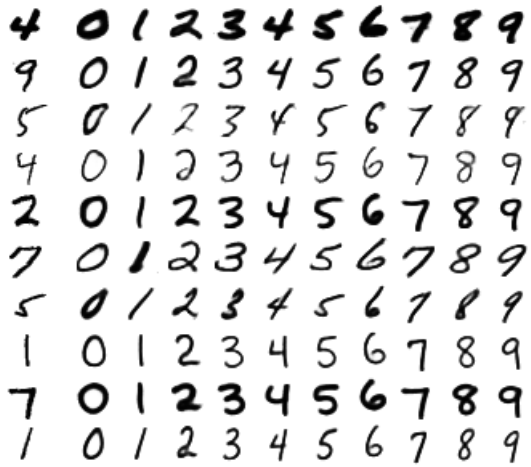
3.5 DISCUSSION AND CONCLUSION

The approximate inference methods introduced here can be easily extended to the model's parameters, harnessing the full power of variational learning. Such an extension also provides a principled ground for performing model selection. Efficient model selection is particularly important when the amount of available data is not large, such as in semi-supervised learning.

For image classification tasks, one area of interest is to combine such methods with convolutional neural networks that form the gold-standard for current supervised classification methods. Since all the components of

our model are parameterized by neural networks we can readily exploit convolutional or more general locally-connected architectures – and forms a promising avenue for future exploration.

We have developed new models for semi-supervised learning that allow us to improve the quality of prediction by exploiting information in the data density using generative models. We have developed an efficient variational optimization algorithm for approximate Bayesian inference in these models and demonstrated that they are amongst the most competitive models currently available for semi-supervised learning. We hope that these results stimulate the development of even more powerful semi-supervised classification methods based on generative models, of which there remains much scope.



(a) Synthetic analogies of MNIST digits.

(b) Synthetic analogies of SVHN images.

Figure 3.1: Analogical reasoning with generative semi-supervised models using a high-dimensional $\mathbf{z}$-space. The leftmost columns show images from the test set. The other columns show analogical fantasies of $\mathbf{x}$ by the generative model, where the latent variable $\mathbf{z}$ of each row is set to the value inferred from the test-set image on the left by the inference network. Each column corresponds to a class label $\mathbf{y}$.

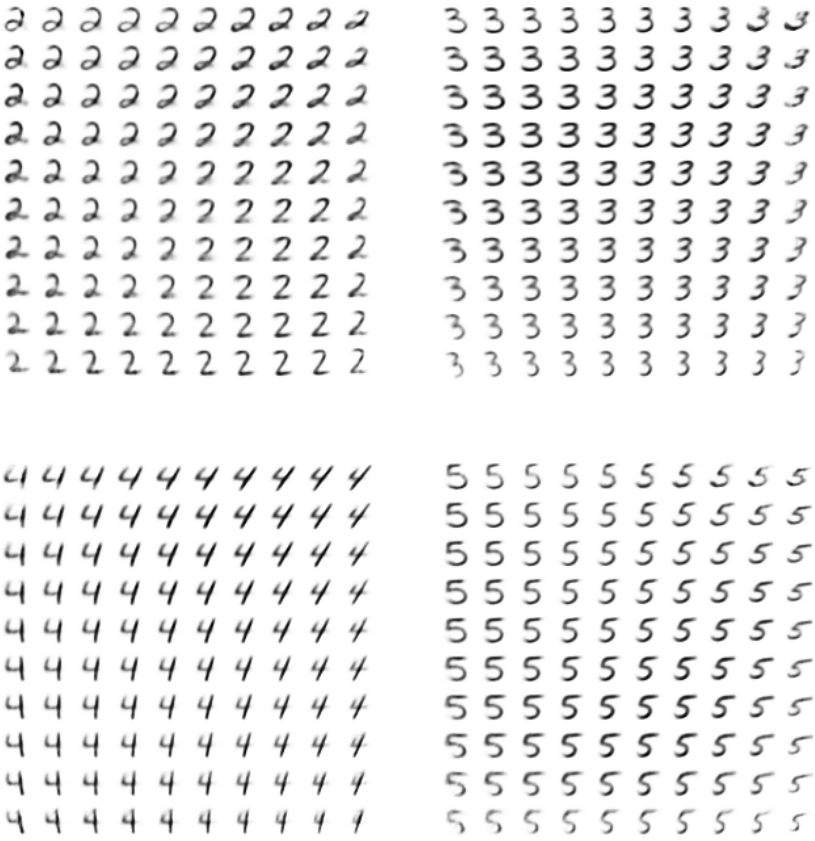


Figure 3.2: Visualization of handwriting styles learned by the model with 2D $\mathbf{z}$-space. The images are obtained by fixing the class label, varying the 2D latent variable $\mathbf{z}$, and generating the corresponding image $\mathbf{x}$ through the decoder.

4

DEEPER GENERATIVE MODELS

In this chapter, we review techniques for training deeper generative models, such as models with multiple layers of stochastic units.

4.1 INFERENCE AND LEARNING WITH MULTIPLE LATENT VARIABLES

The generative model $p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$, and corresponding inference model $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ can be parameterized as any directed graph. Both $\mathbf{x}$ and $\mathbf{z}$ can be composed of multiple variables with some topological ordering. It may not be immediately obvious how to optimize such models in the VAE framework; it is, however, quite straightforward, as we will now explain.

Let $\mathbf{z} = \{\mathbf{z}_1, ..., \mathbf{z}_K\}$, and $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = q_{\boldsymbol{\phi}}(\mathbf{z}_1, ..., \mathbf{z}_K|\mathbf{x})$ where subscript corresponds with the topological ordering of each variable. Given a datapoint $\mathbf{x}$, computation of the ELBO estimator consists of two steps:

1. Sampling $\mathbf{z} \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$. In case of multiple latent variables, this means *ancestral sampling* the latent variables one by one, in topological ordering defined by the inference model's directed graph. In pseudo-code, the ancestral sampling step looks like:

$$\text{for } i = 1...K: \tag{4.1}$$
$$\quad \mathbf{z}_i \sim q_{\boldsymbol{\phi}}(\mathbf{z}_i|Pa(\mathbf{z}_i)) \tag{4.2}$$

where $Pa(\mathbf{z}_i)$ are the parents of variable $\mathbf{z}_i$ in the inference model, which may include $\mathbf{x}$. In reparameterized (and differentiable) form, this is:

$$\text{for } i = 1...K: \tag{4.3}$$

$$\boldsymbol{\epsilon}_i \sim p(\boldsymbol{\epsilon}_i) \tag{4.4}$$

$$\mathbf{z}_i = \mathbf{g}_i(\boldsymbol{\epsilon}_i, Pa(\mathbf{z}_i), \boldsymbol{\phi}) \tag{4.5}$$

2. Evaluating the scalar value $(\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}))$ at the resulting sample $\mathbf{z}$ and datapoint $\mathbf{x}$. This scalar is the unbiased stochastic estimate lower bound on $\log p_{\boldsymbol{\theta}}(\mathbf{x})$. It is also differentiable and optimizable with SGD.

4.1.1 *Choice of ordering*

It should be noted that the choice of latent variables' topological ordering for the inference model, can be different from the choice of ordering for the generative model.

Since the inference model has the data as root node, while the generative model has the data as leaf node, one (in some sense) logical choice would be to let the topological ordering of the latent variables in the inference model, be the reverse of the ordering in the generative model.

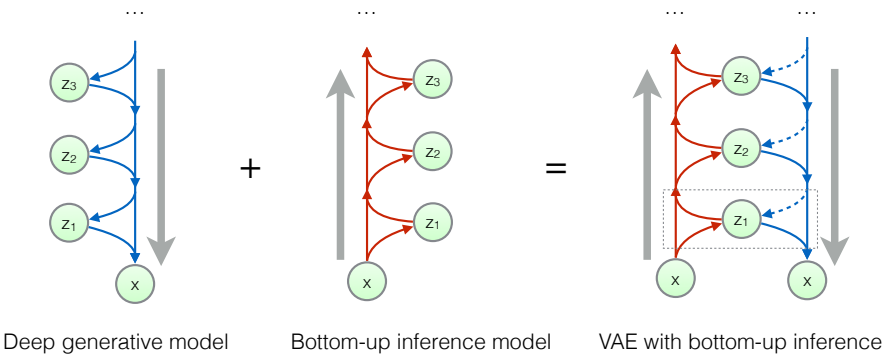
In multiple works [Salimans, 2016, Kaae Sønderby et al., 2016, Kingma et al., 2016] it has been shown that it can be advantageous to let the generative model and inference model *share* the topological ordering of latent variables. The two choices of ordering are illustrated in figure 4.1. One advantage of shared ordering, as explained in these works, is that this allows us to easily share parameters between the inference and generative models, leading to faster learning and better solutions.

To see why this might be a good idea, one should realize that the true posterior over the latent variables, is a function of the prior:

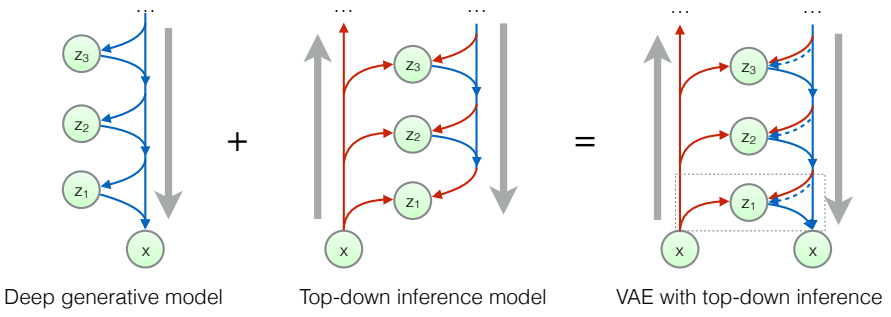
$$p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) \propto p_{\boldsymbol{\theta}}(\mathbf{z}) p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) \tag{4.6}$$

Likewise, the posterior of a latent variable given its parents (in the generative model), is:

$$p_{\boldsymbol{\theta}}(\mathbf{z}_i|\mathbf{x}, Pa(\mathbf{z}_i)) \propto p_{\boldsymbol{\theta}}(\mathbf{z}_i|Pa(\mathbf{z}_i)) p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}_i, Pa(\mathbf{z}_i)) \tag{4.7}$$



(a) VAE with bottom-up inference.



(b) VAE with top-down inference.

Figure 4.1: Illustration, taken from Kingma et al. [2016], of two choices of directionality of the inference model. Sharing directionality of inference, as in (**b**), has the benefit that it allows for straightforward sharing of parameters between the generative model and the inference model.

Optimization of the generative model changes both $p_{\boldsymbol{\theta}}(\mathbf{z}_i|Pa(\mathbf{z}_i))$ and $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}_i, Pa(\mathbf{z}_i))$. By coupling the inference model $q_{\boldsymbol{\phi}}(\mathbf{z}_i|\mathbf{x}, Pa(\mathbf{z}_i))$ and prior $p_{\boldsymbol{\theta}}(\mathbf{z}_i|Pa(\mathbf{z}_i))$, changes in $p_{\boldsymbol{\theta}}(\mathbf{z}_i|Pa(\mathbf{z}_i))$ can be directly reflected in changes in $q_{\boldsymbol{\phi}}(\mathbf{z}_i|Pa(\mathbf{z}_i))$.

This coupling is especially straightforward when $p_{\boldsymbol{\theta}}(\mathbf{z}_i|Pa(\mathbf{z}_i))$ is Gaussian distributed. The inference model can be directly specified as the product of this Gaussian distribution, with a learned quadratic pseudo-likelihood term: $q_{\boldsymbol{\phi}}(\mathbf{z}_i|Pa(\mathbf{z}_i), \mathbf{x}) = p_{\boldsymbol{\theta}}(\mathbf{z}_i|Pa(\mathbf{z}_i))\tilde{l}(\mathbf{z}_i; \mathbf{x}, Pa(\mathbf{z}_i))/Z$, where Z is tractable

to compute. This idea explored by [Salimans, 2016] and [Kaae Sønderby et al., 2016]. In principle this idea could be extended to a more general class of conjugate priors, by no work on this is known at the time of writing.

A less constraining variant, explored by Kingma et al. [2016], is to simply let the neural network that parameterizes $q_{\boldsymbol{\phi}}(\mathbf{z}_i|Pa(\mathbf{z}_i), \mathbf{x})$ be partially specified by a part of the neural network that parameterizes $p_{\boldsymbol{\theta}}(\mathbf{z}_i|Pa(\mathbf{z}_i))$. In general, we can let the two distributions share parameters. This allows for more complicated posteriors, like normalizing flows or IAF.

4.2 ALTERNATIVE METHODS FOR INCREASING EXPRESSIVITY OF GENERATIVE MODELS

Typically, especially with large data sets, we wish to choose an expressive class of directed models, such that it can feasibly approximate the true distribution. Popular strategies for specifying expressive models are:

- Introduction of latent variables into the directed models, and optimization through (amortized) variational inference, as explained in this work.

- Full autoregression: factorization of distributions into univariate (one-dimensional) conditionals, or at least very low-dimensional conditionals (section 4.3).

- Specification of distributions through *invertible transformations with tractable Jacobian determinant* (section 4.4).

Introduction of latent variables for improving expressivity is especially interesting when $\mathbf{x}$ is very high-dimensional. It is relatively straightforward and computationally attractive, due to parallelizability, to specify directed models over high-dimensional variables where each conditional factorizes into independent distributions. For example, if we let $p_{\boldsymbol{\theta}}(\mathbf{x}_j|Pa(\mathbf{x}_j)) = \prod_k p_{\boldsymbol{\theta}}(x_{j,k}|Pa(\mathbf{x}_j)$, where each factor is a univariate Gaussian whose means and variance are nonlinear functions (specified by a neural network) of the parents $Pa(\mathbf{x}_j)$, then computations for both synthesis and evaluation of log-likelihood can be fully parallelized across dimensions k.

The best models to date, in terms of attained log-likelihood on test data, employ a combination of the approaches listed here.

4.3 AUTOREGRESSIVE MODELS

A powerful strategy for modeling high-dimensional data is to divide up the high-dimensional observed variables into small constituents (often single dimensional parts, or otherwise just parts with a small number of dimensions), impose a certain ordering, and to model their dependencies as a directed graphical model. The resulting directed graphical model breaks up the joint distribution into a product of a factors:

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = p_{\boldsymbol{\theta}}(x_1, ..., x_D) = p_{\boldsymbol{\theta}}(x_1) \prod_{j=2}^{T} p_{\boldsymbol{\theta}}(x_j | Pa(\mathbf{x}_j)) \tag{4.8}$$

where D is the dimensionality of the data. This is known as an *autoregressive (AR) model*. In case of neural network based autoregressive models, we let the conditional distributions be parameterized with a neural network:

$$p_{\boldsymbol{\theta}}(x_j | \mathbf{x}_{<j}) = p_{\boldsymbol{\theta}}(x_j | \text{NeuralNet}^j_{\boldsymbol{\theta}}(Pa(\mathbf{x}_j))) \tag{4.9}$$

In case of continuous data, autoregressive models can be interpreted as a special case of a more general approach: learning an invertible transformation from the data to a simpler, known distribution such as a Gaussian or Uniform distribution; this approach with invertible transformations is discussed in section 4.4. The techniques of autoregressive models and invertible transformations can be naturally combined with variational autoencoders, and at the time of writing, the best systems use a combination Rezende and Mohamed [2015], Kingma et al. [2016], Gulrajani et al. [2016].

While full autoregression is arguably the simplest approach, it lacks the advantages of latent spaces. Moreover, a disadvantage of autoregressive models, compared to latent-variable models, is that ancestral sampling from autoregressive models is a sequential operation computation of $\mathcal{O}(D)$ length, i.e. proportional to the dimensionality of the data. This is not ideal when one needs to perform fast sampling on a parallel computer, leading to relatively slow synthesis from fully autoregressive models. The length of computation of the log-likelihood of fully autoregressive models does not necessarily scale with the dimensionality, such that optimization can still be made remarkably efficient on modern parallel hardware; see e.g. MADE [Germain et al., 2015] and convolutional autoregressive models [van den Oord et al., 2016b].

Autoregressive models requires one to chose a one-dimensional ordering of input elements (equation (4.8)). When no natural one-dimensional

ordering exists, like in two-dimensional images, this leads to a model with a somewhat awkward inductive bias.

4.4 INVERTIBLE TRANSFORMATIONS WITH TRACTABLE JACOBIAN DETERMINANT

In case of continuous data, autoregressive models can be interpreted as a special case of a more general approach: learning an invertible transformation with computationally tractable Jacobian determinant, from the data to a simpler, known distribution such as a Gaussian or Uniform distribution. If we use neural networks for such invertible mappings, this is a powerful and flexible approach towards probabilistic modeling of continuous data and nonlinear independent component analysis [Deco and Brauer, 1995].

Such transformations iteratively update a variable, which is constrained to be of the same dimensionality as the data, to a target distribution. This constraint on the dimensionality if intermediate states of the mapping, can make such transformations more challenging to optimize than methods without such constraint. An obvious advantage, on the other hand, is that the likelihood and its gradient are tractable. In [Dinh et al., 2014, 2016], particularly interesting flows (*NICE* and *Real NVP*) were introduced, with equal computational cost and depth in both directions, making it both relatively cheap to optimize and to sample from such models. At the time of writing, no such models has not yet been demonstrated to lead to the similar performance as purely autoregressive or VAE-based models in terms of data log-likelihood, but this remains an active area of research.

5

INVERSE AUTOREGRESSIVE FLOW

Increasing the flexibility of the inference model improves the tightness of the variational bound (ELBO), bringing it closer the true marginal likelihood objective.[1] In this chapter we introduce new techniques for improving the flexibility of the inference model $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$.

5.1 REQUIREMENTS FOR COMPUTATIONAL TRACTABILITY

Requirements for the inference model, in order to be able to efficiently optimize the ELBO, are that it is (1) computationally efficient to compute and differentiate its probability density $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$, and (2) computationally efficient to sample from, since both these operations need to be performed for each datapoint in a minibatch at every iteration of optimization. If $\mathbf{z}$ is high-dimensional and we want to make efficient use of parallel computational resources like GPUs, then parallelizability of these operations across dimensions of $\mathbf{z}$ is a large factor towards efficiency. This requirement restricts the class of approximate posteriors $q(\mathbf{z}|\mathbf{x})$ that are practical to use. In practice this often leads to the use of simple Gaussian posteriors. However, as explained, we also need the density $q(\mathbf{z}|\mathbf{x})$ to be sufficiently flexible to match the true posterior $p(\mathbf{z}|\mathbf{x})$, in order to arrive at a tight bound.

5.2 IMPROVING THE FLEXIBILITY OF INFERENCE MODELS

Here we will review two general techniques for improving the flexibility of approximate posteriors in the context of gradient-based variational inference: auxiliary latent variables, and normalizing flows.

[1] The main contributions in this chapter were first described in Kingma et al. [2016].

5.2.1 *Auxiliary Latent Variables*

One method for improving the flexibility of inference models, is through the introduction of *auxiliary latent variables*, as explored by Salimans et al. [2015], [Ranganath et al., 2015] and Maaløe et al. [2016].

The methods work by augmenting both the inference model and the generative model with a continuous auxiliary variable, here denoted with $\mathbf{u}$.

The inference model defines a distribution over both $\mathbf{u}$ and and $\mathbf{z}$, which can, for example, factorize as:

$$q_{\boldsymbol{\phi}}(\mathbf{u}, \mathbf{z}|\mathbf{x}) = q_{\boldsymbol{\phi}}(\mathbf{u}|\mathbf{x}) q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{u}, \mathbf{x}) \tag{5.1}$$

This inference model augmented with $\mathbf{u}$, implicitly defines a potentially powerful implicit marginal distribution:

$$q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = \int q_{\boldsymbol{\phi}}(\mathbf{u}, \mathbf{z}|\mathbf{x})\, d\mathbf{u} \tag{5.2}$$

Likewise, we introduce an additional distribution in the generative model: such that our generative model is now over the joint distribution $p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}, \mathbf{u})$. This can, for example, factorize as:

$$p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}, \mathbf{u}) = p_{\boldsymbol{\theta}}(\mathbf{u}|\mathbf{x}, \mathbf{z}) p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) \tag{5.3}$$

The ELBO objective with auxiliary variables, given empirical distribution $q_{\mathcal{D}}(\mathbf{x})$, is then (again) equivalent to minimization of a KL divergence:

$$\mathbb{E}_{q_{\mathcal{D}}(\mathbf{x})}\left[\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{u},\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}, \mathbf{u}) - \log q_{\boldsymbol{\phi}}(\mathbf{u}, \mathbf{z}|\mathbf{x})\right]\right] \tag{5.4}$$

$$= D_{KL}(q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{x}, \mathbf{z}, \mathbf{u})||p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}, \mathbf{u}) \tag{5.5}$$

Recall that maximization of the original ELBO objective, without auxiliary variables, is equivalent to minimization of $D_{KL}(q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{x}, \mathbf{z})||p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$, and that maximization of the expected marginal likelihood is equivalent to minimization of $D_{KL}(q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{x})$.

We can gain additional understanding into the relationship between the objectives, through the following equation:

$$
\begin{aligned}
& D_{KL}(q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{x},\mathbf{z},\mathbf{u})||p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z},\mathbf{u})) && (5.6)\\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(= ELBO objective with auxiliary variables)}\\
& = D_{KL}(q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{x},\mathbf{z})||p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})) + \mathbb{E}_{q_{\mathcal{D}}(\mathbf{x},\mathbf{z})}\left[D_{KL}(q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{u}|\mathbf{x},\mathbf{z})||p_{\boldsymbol{\theta}}(\mathbf{u}|\mathbf{x},\mathbf{z}))\right]\\
& \geq D_{KL}(q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{x},\mathbf{z})||p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})) && (5.7)\\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(= original ELBO objective))}\\
& = D_{KL}(q_{\mathcal{D}}(\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{x})) + \mathbb{E}_{q_{\mathcal{D}}(\mathbf{x})}\left[D_{KL}(q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}))\right] && (5.8)\\
& \geq D_{KL}(q_{\mathcal{D}}(\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{x})) && (5.9)\\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(= Marginal log-likelihood objective)}
\end{aligned}
$$

From this equation it can be seen that in principle, the ELBO gets worse by augmenting the VAE with an auxiliary variable $\mathbf{u}$:

$$D_{KL}(q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{x},\mathbf{z},\mathbf{u})||p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z},\mathbf{u})) \geq D_{KL}(q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{x},\mathbf{z})||p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}))$$

But because we now have access to a much more flexible class of distributions ($q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ and $p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})$, the original ELBO objective $D_{KL}(q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{x},\mathbf{z})||p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}))$ can improve, potentially outweighing the additional cost of $\mathbb{E}_{q_{\mathcal{D}}(\mathbf{x},\mathbf{z})}\left[D_{KL}(q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{u}|\mathbf{x},\mathbf{z})||p_{\boldsymbol{\theta}}(\mathbf{u}|\mathbf{x},\mathbf{z}))\right]$. In Salimans et al. [2015], [Ranganath et al., 2015] and Maaløe et al. [2016] it was shown that auxiliary variables can indeed lead to significant improvements in models.

The introduction of auxiliary latent variables in the graph, are a special case of VAEs with multiple layers of latent variables, which are discussed in the chapter 4. In our experiment with CIFAR-10, we make use of multiple layers of stochastic variables.

5.2.2 *Normalizing Flows*

An alternative approach towards flexible approximate posteriors is *Normalizing Flow* (NF), introduced by [Rezende and Mohamed, 2015] in the context of stochastic gradient variational inference. In normalizing flows, we build flexible posterior distributions through an iterative procedure. The general idea is to start off with an initial random variable with a relatively simple distribution with a known (and computationally cheap) probability density

function, and then apply a chain of invertible parameterized transformations $\mathbf{f}_t$, such that the last iterate $\mathbf{z}_T$ has a more flexible distribution[2]:

$$\boldsymbol{\epsilon}_0 \sim p(\boldsymbol{\epsilon}) \tag{5.10}$$

$$\text{for } t = 1...T: \tag{5.11}$$

$$\boldsymbol{\epsilon}_t = \mathbf{f}_t(\boldsymbol{\epsilon}_{t-1}, \mathbf{x}) \tag{5.12}$$

$$\mathbf{z} = \boldsymbol{\epsilon}_T \tag{5.13}$$

The Jacobian of the transformation factorizes:

$$\frac{d\mathbf{z}}{d\boldsymbol{\epsilon}_0} = \prod_{t=1}^{T} \frac{d\boldsymbol{\epsilon}_t}{d\boldsymbol{\epsilon}_{t-1}} \tag{5.14}$$

So its determinant also factorizes as well:

$$\log \left| \det \left(\frac{d\mathbf{z}}{d\boldsymbol{\epsilon}_0} \right) \right| = \sum_{t=1}^{T} \log \left| \det \left(\frac{d\boldsymbol{\epsilon}_t}{d\boldsymbol{\epsilon}_{t-1}} \right) \right| \tag{5.15}$$

As long as the Jacobian determinant of each of the transformations $\mathbf{f}_t$ can be computed, we can still compute the probability density function of the last iterate:

$$\log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = \log p(\boldsymbol{\epsilon}_0) - \sum_{t=1}^{T} \log \det \left| \frac{d\boldsymbol{\epsilon}_t}{d\boldsymbol{\epsilon}_{t-1}} \right| \tag{5.16}$$

Rezende and Mohamed [2015] experimented with a transformation of the form:

$$\mathbf{f}_t(\boldsymbol{\epsilon}_{t-1}) = \boldsymbol{\epsilon}_{t-1} + \mathbf{u} h(\mathbf{w}^T \boldsymbol{\epsilon}_{t-1} + b) \tag{5.17}$$

where $\mathbf{u}$ and $\mathbf{w}$ are vectors, $\mathbf{w}^T$ is $\mathbf{w}$ transposed, b is a scalar and $h(.)$ is a nonlinearity, such that $\mathbf{u}h(\mathbf{w}^T\mathbf{z}_{t-1} + b)$ can be interpreted as a MLP with a bottleneck hidden layer with a single unit. This flow does not scale well to a high-dimensional latent space: since information goes through the single bottleneck, a long chain of transformations is required to capture high-dimensional dependencies.

[2] where $\mathbf{x}$ is the context, such as the value of the datapoint. In case of models with multiple levels of latent variables, the context also includes the value of the previously sampled latent variables.

5.3 INVERSE AUTOREGRESSIVE TRANSFORMATIONS

In Kingma et al. [2016], we introduce a type of normalizing flow that scales well to a high-dimensional space. The technique is based on Gaussian versions of autoregressive autoencoders such as MADE [Germain et al., 2015] and the PixelCNN [van den Oord et al., 2016b]. Let $\mathbf{y}$ be a variable modeled by such a model, with some chosen ordering on its elements $\mathbf{y} = \{y_i\}_{i=1}^D$. We will use $[\boldsymbol{\mu}(\mathbf{y}), \boldsymbol{\sigma}(\mathbf{y})]$ to denote the function of the vector $\mathbf{y}$, to the vectors $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$. Due to the autoregressive structure, the Jacobian is triangular with zeros on the diagonal: $\partial[\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i]/\partial \mathbf{y}_j = [0, 0]$ for $j \geq i$. The elements $[\mu_i(\mathbf{y}_{1:i-1}), \sigma_i(\mathbf{y}_{1:i-1})]$ are the predicted mean and standard deviation of the i-th element of $\mathbf{y}$, which are functions of only the previous elements in $\mathbf{y}$.

Sampling from such a model is a sequential transformation from a noise vector $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$ to the corresponding vector $\mathbf{y}$: $y_0 = \mu_0 + \sigma_0 \odot \epsilon_0$, and for $i > 0$, $y_i = \mu_i(\mathbf{y}_{1:i-1}) + \sigma_i(\mathbf{y}_{1:i-1}) \cdot \epsilon_i$. The computation involved in this transformation is clearly proportional to the dimensionality D. Since variational inference requires sampling from the posterior, such models are not interesting for direct use in such applications. However, the inverse transformation is interesting for normalizing flows. As long as we have $\sigma_i > 0$ for all i, the sampling transformation above is a one-to-one transformation, and can be inverted:

$$\epsilon_i = \frac{y_i - \mu_i(\mathbf{y}_{1:i-1})}{\sigma_i(\mathbf{y}_{1:i-1})} \tag{5.18}$$

Kingma et al. [2016] make two key observations, important for normalizing flows. The first is that this inverse transformation can be parallelized, since (in case of autoregressive autoencoders) computations of the individual elements ϵ_i do not depend on each other The vectorized transformation is:

$$\boldsymbol{\epsilon} = (\mathbf{y} - \boldsymbol{\mu}(\mathbf{y}))/\boldsymbol{\sigma}(\mathbf{y}) \tag{5.19}$$

where the subtraction and division are element-wise

The second key observation, is that this inverse autoregressive operation has a simple Jacobian determinant. Note that due to the autoregressive structure, $\partial[\mu_i, \sigma_i]/\partial y_j = [0, 0]$ for $j \geq i$. As a result, the transformation has a lower triangular Jacobian ($\partial \epsilon_i / \partial y_j = 0$ for $j > i$), with a simple diagonal: $\partial \epsilon_i / \partial y_i = \frac{1}{\sigma_i}$. The determinant of a lower triangular matrix equals the prod-

uct of the diagonal terms. As a result, the log-determinant of the Jacobian of the transformation is remarkably simple and straightforward to compute:

$$\log \det \left| \frac{d\boldsymbol{\epsilon}}{d\mathbf{y}} \right| = \sum_{i=1}^{D} -\log \sigma_i(\mathbf{y}) \tag{5.20}$$

The combination of model flexibility, parallelizability across dimensions, and simple log-determinant, makes this transformation interesting for use as a normalizing flow over high-dimensional latent space.

For the following section we will use a slightly different, but equivalently flexible, transformation of the type:

$$\boldsymbol{\epsilon} = \boldsymbol{\sigma}(\mathbf{y}) \odot \mathbf{y} + \boldsymbol{\mu}(\mathbf{y}) \tag{5.21}$$

With corresponding log-determinant:

$$\log \det \left| \frac{d\boldsymbol{\epsilon}}{d\mathbf{y}} \right| = \sum_{i=1}^{D} \log \sigma_i(\mathbf{y}) \tag{5.22}$$

5.4 INVERSE AUTOREGRESSIVE FLOW (IAF)

In Kingma et al. [2016], we propose an inverse autoregressive flow (IAF) based on a chain of transformations that are each equivalent to an inverse autoregressive transformation of eq. (5.19) and eq. (5.21). See algorithm 5 for pseudo-code of an approximate posterior with the proposed flow. We let an initial encoder neural network output $\boldsymbol{\mu}_0$ and $\boldsymbol{\sigma}_0$, in addition to an extra output $\mathbf{h}$, which serves as an additional input to each subsequent step in the flow. The chain is initialized with a factorized Gaussian $q_{\boldsymbol{\phi}}(\mathbf{z}_0|\mathbf{x}) = \mathcal{N}(0, \text{diag}(\boldsymbol{\sigma})^2)$:

$$\boldsymbol{\epsilon}_0 \sim \mathcal{N}(0, I) \tag{5.23}$$

$$(\boldsymbol{\mu}_0, \log \boldsymbol{\sigma}_0, \mathbf{h}) = \text{EncoderNeuralNet}(\mathbf{x}; \boldsymbol{\theta}) \tag{5.24}$$

$$\mathbf{z}_0 = \boldsymbol{\mu}_0 + \boldsymbol{\sigma}_0 \odot \boldsymbol{\epsilon}_0 \tag{5.25}$$

IAF then consists of a chain of T of the following transformations:

$$(\boldsymbol{\mu}_t, \boldsymbol{\sigma}_t) = \text{AutoregressiveNeuralNet}_t(\boldsymbol{\epsilon}_{t-1}, \mathbf{h}; \boldsymbol{\theta}) \tag{5.26}$$

$$\boldsymbol{\epsilon}_t = \boldsymbol{\mu}_t + \boldsymbol{\sigma}_t \odot \boldsymbol{\epsilon}_{t-1} \tag{5.27}$$

Algorithm 5: Pseudo-code of an approximate posterior with Inverse Autoregressive Flow (IAF)

Data:
$\mathbf{x}$: a datapoint, and optionally other conditioning information
$\boldsymbol{\theta}$: neural network parameters
$\texttt{EncoderNN}(\mathbf{x}; \boldsymbol{\theta})$: encoder neural network, with additional output $\mathbf{h}$
$\texttt{AutoregressiveNN}[*](\mathbf{z}, \mathbf{h}; \boldsymbol{\theta})$: autoregressive neural networks, with additional input $\mathbf{h}$
$\texttt{sum}(.)$: sum over vector elements
$\texttt{sigmoid}(.)$: element-wise sigmoid function
Result:
$\mathbf{z}$: a random sample from $q(\mathbf{z}|\mathbf{x})$, the approximate posterior distribution
l: the scalar value of $\log q(\mathbf{z}|\mathbf{x})$, evaluated at sample '$\mathbf{z}$'

$[\boldsymbol{\mu}, \boldsymbol{\sigma}, \mathbf{h}] \leftarrow \texttt{EncoderNN}(\mathbf{x}; \boldsymbol{\theta})$
$\boldsymbol{\epsilon} \sim \mathcal{N}(0, I)$
$\mathbf{z} \leftarrow \boldsymbol{\sigma} \odot \boldsymbol{\epsilon} + \boldsymbol{\mu}$
$l \leftarrow -\texttt{sum}(\log \boldsymbol{\sigma} + \frac{1}{2}\boldsymbol{\epsilon}^2 + \frac{1}{2}\log(2\pi))$
for $t \leftarrow 1$ **to** T **do**
  $[\mathbf{m}, \mathbf{s}] \leftarrow \texttt{AutoregressiveNN}[t](\mathbf{z}, \mathbf{h}; \boldsymbol{\theta})$
  $\boldsymbol{\sigma} \leftarrow \texttt{sigmoid}(\mathbf{s})$
  $\mathbf{z} \leftarrow \boldsymbol{\sigma} \odot \mathbf{z} + (1 - \boldsymbol{\sigma}) \odot \mathbf{m}$
  $l \leftarrow l - \texttt{sum}(\log \boldsymbol{\sigma})$
end

Each step of this flow is an inverse autoregressive transformation of the type of eq. (5.19) and eq. (5.21), and each step uses a separate autoregressive neural network. Following eq. (5.16), the density under the final iterate is:

$$\mathbf{z} \equiv \boldsymbol{\epsilon}_T \tag{5.28}$$

$$\log q(\mathbf{z}|\mathbf{x}) = -\sum_{i=1}^{D} \left(\frac{1}{2}\epsilon_i^2 + \frac{1}{2}\log(2\pi) + \sum_{t=0}^{T} \log \sigma_{t,i} \right) \tag{5.29}$$

The flexibility of the distribution of the final iterate $\boldsymbol{\epsilon}_T$, and its ability to closely fit to the true posterior, increases with the expressivity of the autore-

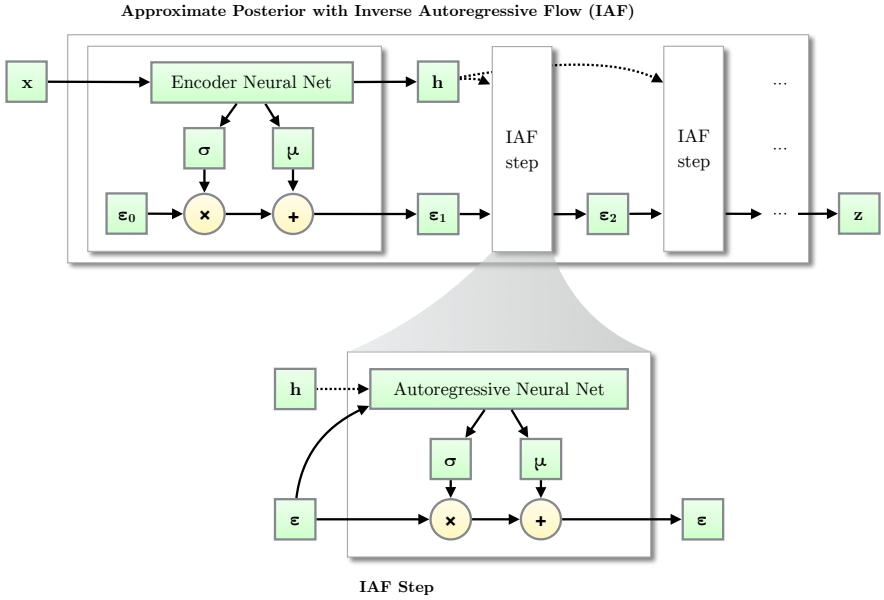


Figure 5.1: Like other normalizing flows, drawing samples from an approximate posterior with Inverse Autoregressive Flow (IAF) [Kingma et al., 2016] starts with a distribution with tractable density, such as a Gaussian with diagonal covariance, followed by a chain of nonlinear invertible transformations of $\mathbf{z}$, each with a simple Jacobian determinant. The final iterate has a flexible distribution.

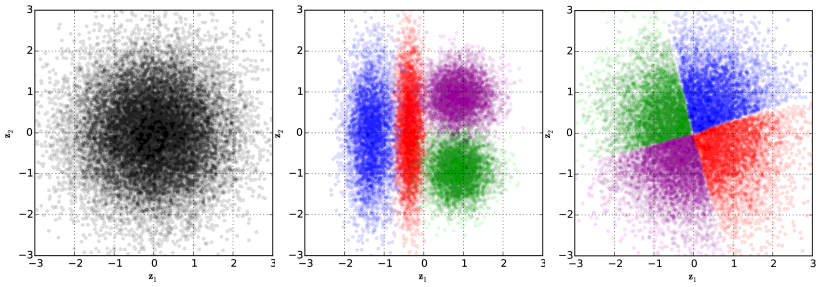
gressive models and the depth of the chain. See figure 5.1 for an illustration of the computation.

A numerically stable version, inspired by the LSTM-type update, is where we let the autoregressive network output $(\mathbf{m}_t, \mathbf{s}_t)$, two unconstrained real-valued vectors, and compute $\boldsymbol{\epsilon}_t$ as:

$$(\mathbf{m}_t, \mathbf{s}_t) = \text{AutoregressiveNeuralNet}_t(\boldsymbol{\epsilon}_{t-1}, \mathbf{h}; \boldsymbol{\theta}) \tag{5.30}$$

$$\boldsymbol{\sigma}_t = \text{sigmoid}(\mathbf{s}_t) \tag{5.31}$$

$$\boldsymbol{\epsilon}_t = \boldsymbol{\sigma}_t \odot \boldsymbol{\epsilon}_{t-1} + (1 - \boldsymbol{\sigma}_t) \odot \mathbf{m}_t \tag{5.32}$$



(a) Prior distribution (b) Factorized posteriors (c) IAF posteriors

Figure 5.2: Best viewed in color. We fitted a variational auto-encoder (VAE) with a spherical Gaussian prior, and with factorized Gaussian posteriors **(b)** or inverse autoregressive flow (IAF) posteriors **(c)** to a toy dataset with four datapoints. Each colored cluster corresponds to the posterior distribution of one datapoint. IAF greatly improves the flexibility of the posterior distributions, and allows for a much better fit between the posteriors and the prior.

This version is shown in algorithm 5. Note that this is just a particular version of the update of eq. (5.27), so the simple computation of the final log-density of eq. (5.29) still applies.

It was found beneficial for results to parameterize or initialize the parameters of each AutoregressiveNeuralNet$_t$ such that its outputs $\mathbf{s}_t$ are, before optimization, sufficiently positive, such as close to +1 or +2. This leads to an initial behavior that updates ϵ only slightly with each step of IAF. Such a parameterization is known as a 'forget gate bias' in LSTMs, as investigated by Jozefowicz et al. [2015].

It is straightforward to see that a special case of IAF with one step, and a linear autoregressive model, is the fully Gaussian posterior discussed earlier. This transforms a Gaussian variable with diagonal covariance, to one with linear dependencies, i.e. a Gaussian distribution with full covariance.

Autoregressive neural networks form a rich family of nonlinear transformations for IAF. For non-convolutional models, the family of masked autoregressive network introduced in [Germain et al., 2015] was used as the autoregressive neural networks. For CIFAR-10 experiments, which benefits more from scaling to high dimensional latent space, the family of convo-

lutional autoregressive autoencoders introduced by [van den Oord et al., 2016b,a] was used.

We found that results improved when reversing the ordering of the variables after each step in the IAF chain. This is a volume-preserving transformation, so the simple form of eq. (5.29) remains unchanged.

5.5 RELATED WORK

As we explained, inverse autoregressive flow (IAF) is a member of the family of normalizing flows, first discussed in [Rezende and Mohamed, 2015] in the context of stochastic variational inference. In [Rezende and Mohamed, 2015] two specific types of flows are introduced: planar flows and radial flows. These flows are shown to be effective to problems relatively low-dimensional latent space (at most a few hundred dimensions). It is not clear, however, how to scale such flows to much higher-dimensional latent spaces, such as latent spaces of generative models of /larger images, and how planar and radial flows can leverage the topology of latent space, as is possible with IAF. Volume-conserving neural architectures were first presented in in [Deco and Brauer, 1995], as a form of nonlinear independent component analysis.

Another type of normalizing flow, introduced by [Dinh et al., 2014] (*NICE*), uses similar transformations as IAF. In contrast with IAF, NICE was directly applied to the observed variables in a generative model. NICE is type of transformations that updates only half of the variables $\mathbf{z}_{1:D/2}$ per step, adding a vector $f(\mathbf{z}_{D/2+1:D})$ which is a neural network based function of the remaining latent variables $\mathbf{z}_{D/2+1:D}$. Such large blocks have the advantage of computationally cheap inverse transformation, and the disadvantage of typically requiring longer chains. In experiments, [Rezende and Mohamed, 2015] found that this type of transformation is generally less powerful than other types of normalizing flow, in experiments with a low-dimensional latent space. Concurrently to our work, NICE was extended to high-dimensional spaces in [Dinh et al., 2016] (*Real NVP*). An empirical comparison would be an interesting subject of future research.

A potentially powerful transformation is the *Hamiltonian flow* used in Hamiltonian Variational Inference [Salimans et al., 2015]. Here, a transformation is generated by simulating the flow of a Hamiltonian system consisting of the latent variables $\mathbf{z}$, and a set of auxiliary momentum variables. This type of transformation has the additional benefit that it is guided by the

exact posterior distribution, and that it leaves this distribution invariant for small step sizes. Such as transformation could thus take us arbitrarily close to the exact posterior distribution if we can apply it for a sufficient number of times. In practice, however, Hamiltonian Variational Inference is very demanding computationally. Also, it requires an auxiliary variational bound to account for the auxiliary variables, which can impede progress if the bound is not sufficiently tight.

An alternative method for increasing the flexibility of variational inference, is the introduction of auxiliary latent variables [Salimans et al., 2015, Ranganath et al., 2015, Tran et al., 2015], also discussed earlier, and corresponding auxiliary inference models. Latent variable models with multiple layers of stochastic variables, such as the one used in our experiments, are often equivalent to such auxiliary-variable methods. We combine deep latent variable models with IAF in our experiments, benefiting from both techniques.

5.6 EXPERIMENTS

We empirically evaluate IAF by applying the idea to improve variational autoencoders. Please see appendix 5.B for details on the architectures of the generative model and inference models. Code for reproducing key empirical results is available online[3].

5.6.1 *MNIST*

In this experiment we follow a similar implementation of the convolutional VAE as in [Salimans et al., 2015] with ResNet [He et al., 2015a] blocks. A single layer of Gaussian stochastic units of dimension 32 is used. To investigate how the expressiveness of approximate posterior affects performance, we report results of different IAF posteriors with varying degrees of expressiveness. We use a 2-layer MADE [Germain et al., 2015] to implement one IAF transformation, and we stack multiple IAF transformations with ordering reversed between every other transformation.

[3] `https://github.com/openai/iaf`

Table 5.1: Generative modeling results on the dynamically sampled binarized MNIST version used in [Burda et al., 2015]. Shown are averages; the number between brackets are standard deviations across 5 optimization runs. The right column shows an importance sampled estimate of the marginal likelihood for each model with 128 samples. Best previous results are reproduced in the first segment: [1]: [Salimans et al., 2015] [2]: [Burda et al., 2015] [3]: [Kaae Sønderby et al., 2016] [4]: [Tran et al., 2015]

Model	VLB	$\log p(\mathbf{x}) \approx$
Convolutional VAE + HVI [1]	-83.49	-81.94
DLGM 2hl + IWAE [2]		-82.90
LVAE [3]		-81.74
DRAW + VGP [4]	-79.88	
Diagonal covariance	-84.08 ($\pm$ 0.10)	-81.08 ($\pm$ 0.08)
IAF (Depth $= 2$, Width $= 320$)	-82.02 ($\pm$ 0.08)	-79.77 ($\pm$ 0.06)
IAF (Depth $= 2$, Width $= 1920$)	-81.17 ($\pm$ 0.08)	-79.30 ($\pm$ 0.08)
IAF (Depth $= 4$, Width $= 1920$)	-80.93 ($\pm$ 0.09)	-79.17 ($\pm$ 0.08)
IAF (Depth $= 8$, Width $= 1920$)	-80.80 ($\pm$ 0.07)	**-79.10** ($\pm$ 0.07)

5.6.1.1 *Results*

Table 5.1 shows results on MNIST for these types of posteriors. Results indicate that as approximate posterior becomes more expressive, generative modeling performance becomes better. Also worth noting is that an expressive approximate posterior also tightens variational lower bounds as expected, making the gap between variational lower bounds and marginal likelihoods smaller. By making IAF deep and wide enough, we can achieve best published log-likelihood on dynamically binarized MNIST: **-79.10**. On Hugo Larochelle's statically binarized MNIST, our VAE with deep IAF achieves a log-likelihood of **-79.88**, which is slightly worse than the best reported result, **-79.2**, using the PixelCNN [van den Oord et al., 2016b].

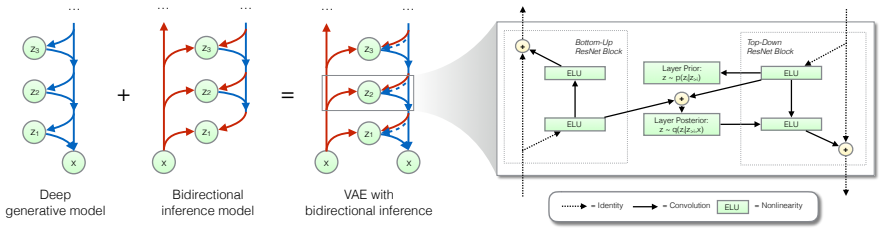


Figure 5.1: Overview of our ResNet VAE with bidirectional inference. The posterior of each layer is parameterized by its own IAF.

5.6.2 *CIFAR-10*

We also evaluated IAF on the CIFAR-10 dataset of natural images. Natural images contain a much greater variety of patterns and structure than MNIST images; in order to capture this structure well, we experiment with a novel architecture, ResNet VAE, with many layers of stochastic variables, and based on residual convolutional networks (ResNets) [He et al., 2015a, 2016]. Please see the appendix for architectural details.

5.6.2.1 *Log-likelihood*

See table 5.2 for a comparison to previously reported results. Our architecture with IAF achieves **3.11 bits per dimension**, which is better than other published latent-variable models, and almost on par with the best reported result using the PixelCNN. See the appendix for more experimental results. We suspect that the results can be further improved with more steps of flow, which we leave to future work.

5.6.2.2 *Synthesis speed*

Sampling took about **0.05 seconds/image** with the ResNet VAE model, versus **52.0 seconds/image** with the PixelCNN model, on a NVIDIA Titan X GPU. We sampled from the PixelCNN naïvely by sequentially generating a pixel at a time, using the full generative model at each iteration. With custom code that only evaluates the relevant part of the network, PixelCNN sampling could be sped up significantly; however the speedup will be limited on parallel hardware due to the sequential nature of the sampling operation.

Table 5.2: Our results with ResNet VAEs on CIFAR-10 images, compared to earlier results, in *average number of bits per data dimension* on the test set. The number for convolutional DRAW is an upper bound, while the ResNet VAE log-likelihood was estimated using importance sampling.

Method	bits/dim $\leq$
Results with tractable likelihood models:	
Uniform distribution [van den Oord et al., 2016b]	8.00
Multivariate Gaussian [van den Oord et al., 2016b]	4.70
NICE [Dinh et al., 2014]	4.48
Deep GMMs [van den Oord and Schrauwen, 2014]	4.00
Real NVP [Dinh et al., 2016]	3.49
PixelRNN [van den Oord et al., 2016b]	**3.00**
Gated PixelCNN [van den Oord et al., 2016a]	**3.03**
Results with variationally trained latent-variable models:	
Deep Diffusion [Sohl-Dickstein et al., 2015]	5.40
Convolutional DRAW [Gregor et al., 2016]	3.58
ResNet VAE with IAF (Ours)	**3.11**

Efficient sampling from the ResNet VAE is a parallel computation that does not require custom code.

5.7 CONCLUSION

We presented *inverse autoregressive flow* (IAF), a new type of normalizing flow that scales well to high-dimensional latent space. In experiments we demonstrated that autoregressive flow leads to significant performance gains compared to similar models with factorized Gaussian approximate posteriors, and we report close to state-of-the-art log-likelihood results on CIFAR-10, for a model that allows much faster sampling.

CHAPTER APPENDIX

5.A MNIST

In MNIST experiment we follow a similar implementation of the convolutional VAE as in [Salimans et al., 2015] with ResNet [He et al., 2015a] blocks. A single layer of Gaussian stochastic units of dimension 32 is used. The inference network has three 2-strided resnet blocks with 3x3 filters and [16,32,32] feature maps. Between every other strided convolution, there is another resnet block with stride 1 and same number feature maps. There is one more fully-connected layer after convolutional layers with 450 hidden units. The generation network has a symmetric structure with strided convolution replaced by transposed convolution[Zeiler et al., 2010]. When there is a change in dimensionality, we use strided convolution or transposed convolution to replace the identity connection in ResNet blocks. Exponential Linear Units [Clevert et al., 2015] are used as activation functions. The whole network is parametrized according to Weight Normalization[Salimans and Kingma, 2016] and data-dependent initialization is used.

5.B RESNET VAE

For details of the ResNet VAE architecture used for CIFAR-10, please see figure 5.1 and our provided source-code. The main benefits of this architecture is that it forms a flexible autoregressive prior latent space, while still being straightforward to sample from.

For CIFAR-10, we used a novel neural variational autoencoder (VAE) architecture with ResNet [He et al., 2015a, 2016] units and multiple stochastic layers. Our architecture consists of L stacked blocks, where each block $(l = 1..L)$ is a combination of a bottom-up residual unit for inference, producing a series of bottom-up activations $\mathbf{h}_l^{(q)}$, and a top-down residual unit used for both inference and generation, producing a series of top-down activations $\mathbf{h}_l^{(p)}$.

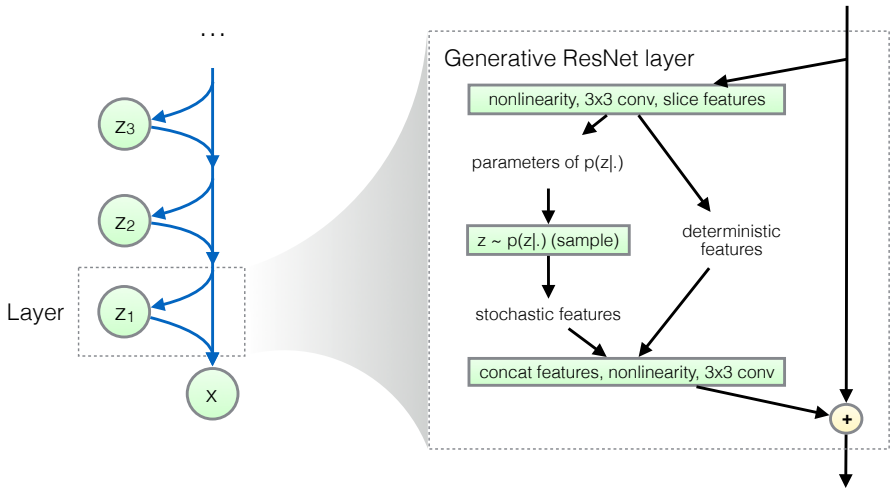


Figure 5.B.1: Generative ResNet and detail of layer. This is the generative component of our ResNet VAE.

The hidden layer of each residual function in the generative model contains a combination of the usual deterministic hidden units and a relatively small number of stochastic hidden units with a heteroscedastic diagonal Gaussian distribution $p(\mathbf{z}_l|\mathbf{h}_l^{(p)})$ given the unit's input $\mathbf{h}_l^{(p)}$, followed by a nonlinearity. We utilize wide [Zagoruyko and Komodakis, 2016] *pre-activation residual units* [He et al., 2015a] with single-hidden-layer residual functions.

See figure 5.B.2 for an illustration of the generative Resnet. Assuming L layers of latent variables, the generative model's density function is factorized as $p(\mathbf{x}, \mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3, ...) = p(\mathbf{x}, \mathbf{z}_{1:L}) = p(\mathbf{x}|\mathbf{z}_{1:L})p(\mathbf{z}_{1:L})$. The second part of this density, the prior over the latent variable, is autoregressive: $p(\mathbf{z}_{1:L}) = p(\mathbf{z}_L)\prod_{l=1}^{L-1} p(\mathbf{z}_l|\mathbf{z}_{l+1:L})$. This autoregressive nature of the prior increases the flexibility of the true posterior, leading to improved empirical results. This improved performance is easily explained: as the true posterior is largely a function of this prior, a flexible prior improves the flexibility of the true posterior, making it easier for the VAE to match the approximate and true posteriors, leading to a tighter bound, without sacrificing the flexibility of the generative model itself.

5.B.1 *Bottom-Up versus Bidirectional inference*

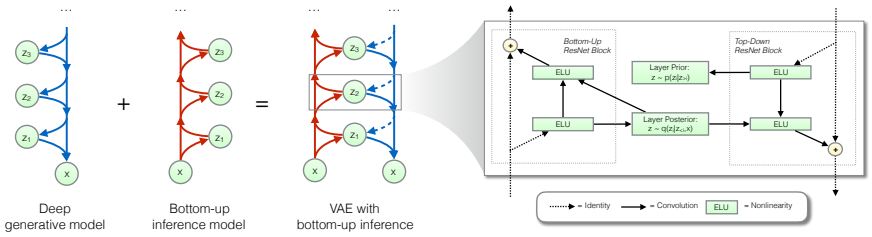
Figure 5.B.2 illustrates the difference between a *bidirectional* inference network, whose topological ordering over the latent variables equals that of the generative model, and a *bottom-up* inference network, whose topological ordering is reversed.

The top left shows a generative model with three levels of latent variables, with topological ordering $\mathbf{z}_1 \rightarrow \mathbf{z}_2 \rightarrow \mathbf{z}_3 \rightarrow \mathbf{x}$, and corresponding joint distribution that factorizes as $p(\mathbf{x}, \mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3) = p(\mathbf{x}|\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3)p(\mathbf{z}_1|\mathbf{z}_2, \mathbf{z}_3)p(\mathbf{z}_2|\mathbf{z}_3)$ $p(\mathbf{z}_3)$. The top middle shows a corresponding inference model with reversed topological ordering, and corresponding factorization $q(\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3|\mathbf{x}) = q(\mathbf{z}_1|\mathbf{x})$ $q(\mathbf{z}_2|\mathbf{z}_1, \mathbf{x})q(\mathbf{z}_3|\mathbf{z}_2, \mathbf{z}_1, \mathbf{x})$. We call this a bottom-up inference model. Right: the resulting variational autoencoder (VAE). In the VAE, the log-densities of the inference model and generative model, $\log p(\mathbf{x}, \mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3) -$ $\log q(\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3|\mathbf{x})$ respectively, are evaluated under a sample from the inference model, to produce an estimate of the variational bound. Computation of the density of the samples $(\mathbf{x}, \mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3)$ under the generative model, requires computation of the conditional distributions of the generative model, which requires bidirectional computation. Hence, evaluation of the model requires both bottom-up inference (for sampling $\mathbf{z}$ s, and evaluating the posterior density) and top-down generation.

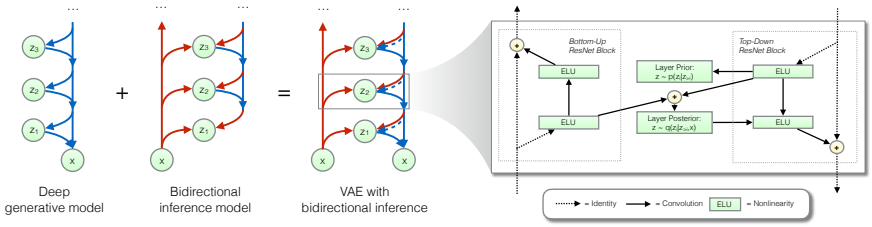
In case of bidirectional inference (see also [Salimans, 2016, Kaae Sønderby et al., 2016]), we first perform a fully deterministic bottom-up pass, before sampling from the posterior in top-down order in the topological ordering of the generative models.

5.B.2 *Inference with stochastic ResNet*

Like our generative Resnet, both our bottom-up and bidirectional inference models are implemented through ResNet blocks. See figure 5.B.3. As we explained, in case of bottom-up inference, latent variables are sampled in bottom-up order, i.e. the reverse of the topological ordering of the generative model. The residual functions in the bottom-up inference network compute a conditional approximate posterior, $q(\mathbf{z}_l|\mathbf{h}_{l+1}^{(p)})$, conditioned on the bottom-up residual unit's input. The sample from this distribution is then, after application of the nonlinearity, treated as part of the hidden layer of the bottom-up residual function and thus used upstream.



(a) Schematic overview of a ResNet VAE with bottom-up inference.



(b) Schematic overview of a ResNet VAE with bidirectional inference.

Figure 5.B.2: Schematic overview of topological orderings of bottom-up **(a)** versus bidirectional **(b)** inference networks, and their corresponding variational autoencoders (VAEs). See section 5.B.1

In the bidirectional inference case, the approximate posterior for each layer is conditioned on both the bottom-up input and top-down input: $q(\mathbf{z}_l | \mathbf{h}_l^{(q)}, \mathbf{h}_l^{(p)})$. The sample from this distribution is, again after application of the nonlinearity, treated as part of the hidden layer of the top-down residual function and thus used downstream.

5.B.3 *Approximate posterior*

The approximate posteriors $q(\mathbf{z}_l|.)$ are defined either through a diagonal Gaussian, or through an IAF posterior. In case of IAF, the context $\mathbf{c}$ is provided by either $\mathbf{h}_l^{(q)}$, or $\mathbf{h}_l^{(q)}$ and $\mathbf{h}_l^{(p)}$, dependent on inference direction.

Table 5.B.1: CIFAR-10 test-set bpp (bits per pixel), when training IAF with location-only perturbation versus full (location+scale) perturbation.

ResNet depth	Inference direction	IAF depth	Loc.	Loc.+scale
4	Bottom-up	0	3.67	3.68
4	Bottom-up	1	3.61	3.61
4	Bidirectional	0	3.66	3.67
4	Bidirectional	1	3.58	3.56
8	Bottom-up	0	3.54	3.55
8	Bottom-up	1	3.48	3.49
8	Bidirectional	0	3.51	3.52
8	Bidirectional	1	3.45	3.42

We use either diagonal Gaussian posteriors, or IAF with a single step of masked-based PixelCNN [van den Oord et al., 2016b] with zero, one or two layers of hidden layers with ELU nonlinearities [Clevert et al., 2015]. Note that IAF with zero hidden layers corresponds to a linear transformation; i.e. a Gaussian with off-diagonal covariance.

We investigate the importance of a full IAF transformation with learned (dynamic) $\sigma_t(.)$ rescaling term term (in eq.(5.27)), versus a fixed $\sigma_t(.) = 1$ term. See table 5.B.1 for a comparison of resulting test-set bpp (bits per pixel) performance, on the CIFAR-10 dataset. We found the difference in performance to be almost negligible.

5.B.4 *Bottom layer*

The first layer of the encoder is a convolutional layer with 2×2 spatial sub-sampling; the last layer of the decoder has a matching convolutional layer with 2×2 spatial up-sampling The ResNet layers could perform further up- and down-sampling, but we found that this did not improve empirical results.

5.B.5 *Discretized Logistic Likelihood*

The first layer of the encoder, and the last layer of the decoder, consist of convolutions that project from/to input space. The pixel data is scaled to the range $[0, 1]$, and the data likelihood of pixel values in the generative model is the probability mass of the pixel value under the logistic distribution. Noting that the CDF of the standard logistic distribution is simply the sigmoid function, we simply compute the probability mass per input pixel using $p(x_i|\mu_i, s_i) = \text{CDF}(x_i + \frac{1}{256}|\mu_i, s_i) - \text{CDF}(x_i|\mu_i, s_i)$, where the locations μ_i are output of the decoder, and the log-scales $\log s_i$ are learned scalar parameter per input channel.

5.B.6 *Weight initialization and normalization*

We also found that the noise introduced by batch normalization hurts performance; instead we use weight normalization [Salimans and Kingma, 2016] method. We initialized the parameters using the data-dependent technique described in [Salimans and Kingma, 2016].

5.B.7 *Nonlinearity*

We compared ReLU, softplus, and ELU [Clevert et al., 2015] nonlinearities; we found that ELU resulted in significantly better empirical results, and used the ELU nonlinearity for all reported experiments and both the inference model and the generative model.

5.B.8 *Objective with Free Bits*

To accelerate optimization and reach better optima, we optimize the bound using a slightly modified objective with *free bits*: a constraint on the minimum amount of information per group of latent variables. Consistent with findings in [Bowman et al., 2015] and [Kaae Sønderby et al., 2016], we found that stochastic optimization with the unmodified lower bound objective often gets stuck in an undesirable stable equilibrium. At the start of training, the likelihood term $\log p(\mathbf{x}|\mathbf{z})$ is relatively weak, such that an initially attractive state is where $q(\mathbf{z}|\mathbf{x}) \approx p(\mathbf{z})$. In this state, encoder gradients have a relatively low signal-to-noise ratio, resulting in a stable equilibrium from

which it is difficult to escape. The solution proposed in [Bowman et al., 2015] and [Kaae Sønderby et al., 2016] is to use an optimization schedule where the weight of the latent cost $D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))$ is slowly annealed from 0 to 1 over many epochs.

We propose a different solution that does not depend on an annealing schedule, but uses a modified objective function that is constant throughout training instead. We divide the latent dimensions into the K groups/subsets within which parameters are shared (e.g. the latent feature maps, or individual dimensions if no parameter are shared across dimensions). We then use the following objective, which ensures that using less than λ nats of information per subset j (on average per minibatch $\mathcal{M}$) is not advantageous:

$$\begin{aligned}\widetilde{\mathcal{L}}_\lambda &= \mathbb{E}_{\mathbf{x}\sim\mathcal{M}}\left[\mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\log p(\mathbf{x}|\mathbf{z})\right]\right] \\ &- \sum_{j=1}^{K} \text{maximum}(\lambda, \mathbb{E}_{\mathbf{x}\sim\mathcal{M}}\left[D_{KL}(q(\mathbf{z}_j|\mathbf{x})||p(\mathbf{z}_j))\right]) \end{aligned} \tag{5.33}$$

Since increasing the latent information is generally advantageous for the first (unaffected) term of the objective (often called the *negative reconstruction error*), this results in $\mathbb{E}_{\mathbf{x}\sim\mathcal{M}}\left[D_{KL}(q(\mathbf{z}_j|\mathbf{x})||p(\mathbf{z}_j))\right] \geq \lambda$ for all j, in practice.

We experimented with $\lambda \in [0, 0.125, 0.25, 0.5, 1, 2, 4, 8]$ and found that values in the range $\lambda \in [0.125, 0.25, 0.5, 1, 2]$ resulted in more than 0.1 nats improvement in bits/pixel on the CIFAR-10 benchmark.

5.B.9 *IAF architectural comparison*

We performed an empirical evaluation on CIFAR-10, comparing various choices of ResNet VAE combined with various approximate posteriors and model depths, and inference directions. See table 5.B.2 for results. Our best CIFAR-10 result, 3.11 bits/dim in table 5.2 was produced by a ResNet VAE with depth 20, bidirectional inference, nonlinear IAF with 2 hidden layers and 1 step. Please see our code for further details.

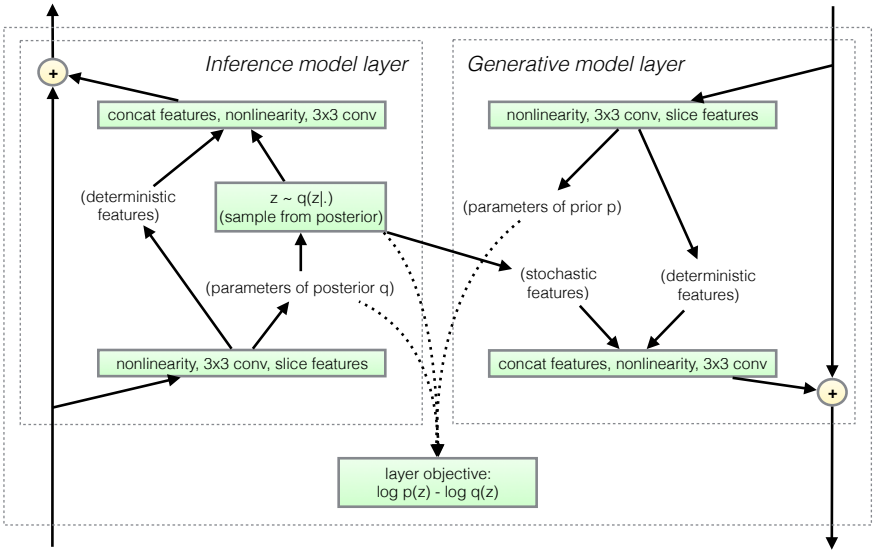
5.C EQUIVALENCE WITH AUTOREGRESSIVE PRIORS

Earlier work on improving variational auto-encoders has often focused on improving the prior $p(\mathbf{z})$ of the latent variables in our generative model. For example, [Gregor et al., 2013] use a variational auto-encoder where both the

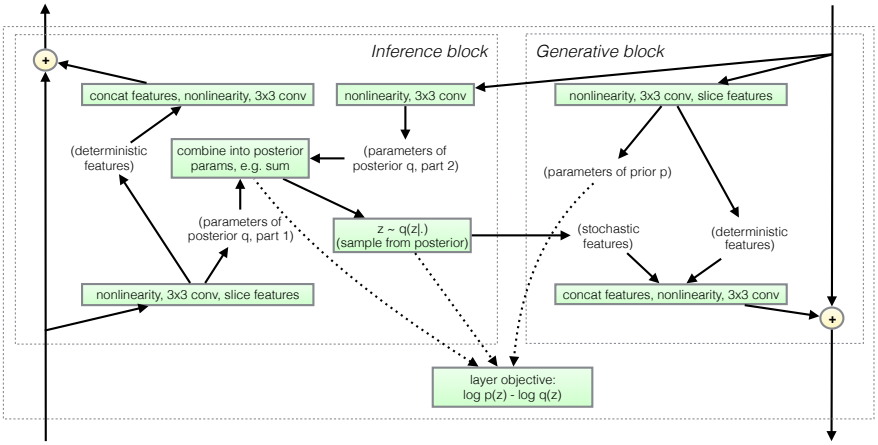
Table 5.B.2: Our results in *average number of bits per data dimension* on the test set with ResNet VAEs, for various choices of posterior ResNet depth, and IAF depth.

Posterior:	ResNet Depth:	4	8	12
Bottom-up, factorized Gaussians		3.71	3.55	3.44
Bottom-up, IAF, linear, 1 step)		3.68	3.55	3.41
Bottom-up, IAF, 1 hidden layer, 1 step)		3.61	3.49	3.38
Bidirectional, factorized Gaussians		3.74	3.60	3.46
Bidirectional, IAF, linear, 1 step)		3.67	3.52	3.40
Bidirectional, IAF, 1 hidden layer, 1 step)		3.56	3.42	3.28
Bidirectional, IAF, 2 hidden layers, 1 steps)		3.54	3.39	3.27
Bidirectional, IAF, 1 hidden layer, 2 steps)		3.53	3.36	3.26

prior and inference network have recursion. It is therefore worth noting that our method of improving the fully-factorized posterior approximation with inverse autoregressive flow, in combination with a factorized prior $p(\mathbf{z})$, is equivalent to estimating a model where the prior $p(\mathbf{z})$ is autoregressive and our posterior approximation is factorized. This result follows directly from the analysis of section 5.3: we can consider the latent variables $\mathbf{y}$ to be our target for inference, in which case our prior is autoregressive. Equivalently, we can consider the whitened representation $\mathbf{z}$ to be the variables of interest, in which case our prior is fully-factorized and our posterior approximation is formed through the inverse autoregressive flow that whitens the data (equation 5.19).



(a) Computational flow schematic of ResNet VAE with bottom-up inference



(b) Computational flow schematic of ResNet VAE with bidirectional inference

Figure 5.B.3: Detail of a single layer of the ResNet VAE, with the bottom-up inference (top) and bidirectional inference (bottom).

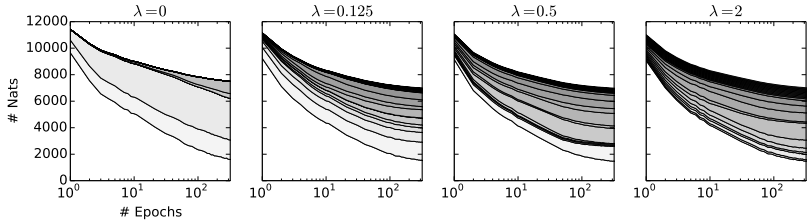


Figure 5.B.4: Shown are stack plots of the number of nats required to encode the CIFAR-10 set images, per stochastic layer of the 24-layer network, as a function of the number of training epochs, for different choices of minimum information constraint λ (see section 5.B.8). Enabling the constraint ($\lambda > 0$) results in avoidance of undesirable stable equilibria, and fuller use of the stochastic layers by the model. The bottom-most (white) area corresponds to the bottom-most (reconstruction) layer, the second area from the bottom denotes the first stochastic layer, the third area denotes the second stochastic layer, etc.

6

VARIATIONAL DROPOUT AND LOCAL REPARAMETERIZATION

We investigate a local reparameterization technique for reducing the variance of stochastic gradients for variational Bayesian inference (SGVB) [Kingma and Welling, 2013] of a posterior over model parameters, while retaining parallelizability. This local reparameterization translates uncertainty about global parameters into local noise that is independent across datapoints in the minibatch. Such parameterizations can be trivially parallelized and have variance that is inversely proportional to the minibatch size, generally leading to much faster convergence. Additionally, we explore a connection with dropout: Gaussian dropout objectives correspond to SGVB with local reparameterization, a scale-invariant prior and proportionally fixed posterior variance. Our method allows inference of more flexibly parameterized posteriors; specifically, we propose *variational dropout*, a generalization of Gaussian dropout where the dropout rates are learned, often leading to better models. The method is demonstrated through several experiments.[1]

6.1 INTRODUCTION

Deep neural networks are a flexible family of models that easily scale to millions of parameters and datapoints, but are still tractable to optimize using minibatch-based stochastic gradient ascent. Due to their high flexibility, neural networks have the capacity to fit a wide diversity of nonlinear patterns in the data. This flexibility often leads to *overfitting* when left unchecked: spurious patterns are found that happen to fit well to the training data, but are not predictive for new data. Various regularization tech-

[1] This chapter is adapted from our publication [Kingma et al., 2015].

niques for controlling this overfitting are used in practice; a currently popular and empirically effective technique being *dropout* Hinton et al. [2012]. In Wang and Manning [2013] it was shown that regular (binary) dropout has a Gaussian approximation called *Gaussian dropout* with virtually identical regularization performance but much faster convergence. In section 5 of Wang and Manning [2013] it is shown that Gaussian dropout optimizes a lower bound on the marginal likelihood of the data. In this work we show that a relationship between dropout and Bayesian inference can be extended and exploited to greatly improve the efficiency of variational Bayesian inference on the model parameters. This work has a direct interpretation as a generalization of Gaussian dropout, with the same fast convergence but now with the freedom to specify more flexibly parameterized posterior distributions.

Bayesian posterior inference over the neural network parameters is a theoretically attractive method for controlling overfitting; exact inference is computationally intractable, but efficient approximate schemes can be designed. Markov Chain Monte Carlo (MCMC) is a class of approximate inference methods with asymptotic guarantees, pioneered by Neal [1995] for the application of regularizing neural networks. Later useful refinements include Welling and Teh [2011] and Ahn et al. [2012].

An alternative to MCMC is variational inference Hinton and Van Camp [1993] or the equivalent *minimum description length* (MDL) framework. Modern variants of stochastic variational inference have been applied to neural networks with some success Graves [2011], but have been limited by high variance in the gradients. Despite their theoretical attractiveness, Bayesian methods for inferring a posterior distribution over neural network weights have not yet been shown to outperform simpler methods such as dropout. Even a new crop of efficient variational inference algorithms based on stochastic gradients with minibatches of data Salimans and Knowles [2013], Kingma and Welling [2013], Rezende et al. [2014] have not yet been shown to significantly improve upon simpler dropout-based regularization.

In section 6.2 we explore an as yet unexploited trick for improving the efficiency of stochastic gradient-based variational inference with minibatches of data, by translating uncertainty about global parameters into local noise that is independent across datapoints in the minibatch. The resulting method has an optimization speed on the same level as fast dropout Wang and Manning [2013], and indeed has the original Gaussian dropout method as a special case. An advantage of our method is that it allows for full Bayesian analysis of the model, and that it's significantly more flexible than standard dropout.

The approach presented here is closely related to several popular methods in the literature that regularize by adding random noise; these relationships are discussed in section 7.4.

6.2 EFFICIENT AND PRACTICAL BAYESIAN INFERENCE

We consider Bayesian analysis of a dataset $\mathcal{D}$, containing a set of N i.i.d. observations of tuples $(\mathbf{x}, \mathbf{y})$, where the goal is to learn a model with *parameters* or *weights* $\mathbf{w}$ of the conditional probability $p(\mathbf{y}|\mathbf{x}, \mathbf{w})$ (standard classification or regression)[2]. Bayesian inference in such a model consists of updating some initial belief over parameters $\mathbf{w}$ in the form of a prior distribution $p(\mathbf{w})$, after observing data $\mathcal{D}$, into an updated belief over these parameters in the form of (an approximation to) the posterior distribution $p(\mathbf{w}|\mathcal{D})$. Computing the true posterior distribution through Bayes' rule $p(\mathbf{w}|\mathcal{D}) = p(\mathbf{w})p(\mathcal{D}|\mathbf{w})/p(\mathcal{D})$ involves computationally intractable integrals, so good approximations are necessary. In *variational inference*, inference is cast as an optimization problem where we optimize the parameters $\boldsymbol{\phi}$ of some parameterized model $q_{\boldsymbol{\phi}}(\mathbf{w})$ such that $q_{\boldsymbol{\phi}}(\mathbf{w})$ is a close approximation to $p(\mathbf{w}|\mathcal{D})$ as measured by the Kullback-Leibler divergence $D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{w})||p(\mathbf{w}|\mathcal{D}))$. This divergence of our posterior $q_{\boldsymbol{\phi}}(\mathbf{w})$ to the true posterior is minimized in practice by maximizing the so-called variational lower bound $\mathcal{L}(\boldsymbol{\phi})$ of the marginal likelihood of the data:

$$\mathcal{L}(\boldsymbol{\phi}) = -D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{w})||p(\mathbf{w})) + L_{\mathcal{D}}(\boldsymbol{\phi}) \tag{6.1}$$

$$\text{where} \quad L_{\mathcal{D}}(\boldsymbol{\phi}) = \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{D}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{w})}\left[\log p(\mathbf{y}|\mathbf{x}, \mathbf{w})\right] \tag{6.2}$$

We'll call $L_{\mathcal{D}}(\boldsymbol{\phi})$ the *expected log-likelihood*. The bound $\mathcal{L}(\boldsymbol{\phi})$ plus $D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{w})||p(\mathbf{w}|\mathcal{D}))$ equals the (conditional) marginal log-likelihood $\sum_{(\mathbf{x},\mathbf{y})\in\mathcal{D}} \log p(\mathbf{y}|\mathbf{x})$. Since this marginal log-likelihood is constant w.r.t. $\boldsymbol{\phi}$, maximizing the bound w.r.t. $\boldsymbol{\phi}$ will minimize $D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{w})||p(\mathbf{w}|\mathcal{D}))$.

6.2.1 *Stochastic Gradient Variational Bayes (SGVB)*

Various algorithms for gradient-based optimization of the variational bound (eq. (6.1)) with differentiable q and p exist. See section 7.4 for an overview.

[2] Note that the described method is not limited to classification or regression and is straightforward to apply to other modeling settings like unsupervised models and temporal models.

A recently proposed efficient method for minibatch-based optimization with differentiable models is the *stochastic gradient variational Bayes* (SGVB) method introduced in Kingma and Welling [2013] (see appendix and previous chapters) and Rezende et al. [2014]. The basic trick in SGVB is to parameterize the random parameters $\mathbf{w} \sim q_{\boldsymbol{\phi}}(\mathbf{w})$ as: $\mathbf{w} = f(\boldsymbol{\epsilon}, \boldsymbol{\phi})$ where $f(.)$ is a differentiable function of variational parameter $\boldsymbol{\phi}$ and random noise $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$. In this new parameterization, an unbiased differentiable minibatch-based Monte Carlo estimator of the expected log-likelihood can be formed:

$$L_{\mathcal{D}}(\boldsymbol{\phi}) \simeq L_{\mathcal{D}}^{\text{SGVB}}(\boldsymbol{\phi}) = \frac{N}{M} \sum_{i=1}^{M} \log p(\mathbf{y}^{(i)}|\mathbf{x}^{(i)}, \mathbf{w} = f(\boldsymbol{\epsilon}, \boldsymbol{\phi})), \tag{6.3}$$

where $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})_{i=1}^{M}$ is a minibatch of data with M random datapoints $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) \sim \mathcal{D}$, and $\boldsymbol{\epsilon}$ is a noise vector drawn from the noise distribution $p(\boldsymbol{\epsilon})$. We'll assume that the remaining term in the variational lower bound, $D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{w})||p(\mathbf{w}))$, can be computed deterministically, but otherwise it may be approximated similarly. The estimator (6.3) is differentiable w.r.t. $\boldsymbol{\phi}$ and unbiased, so its gradient is also unbiased: $\nabla_{\phi} L_{\mathcal{D}}(\boldsymbol{\phi}) \simeq \nabla_{\boldsymbol{\phi}} L_{\mathcal{D}}^{\text{SGVB}}(\boldsymbol{\phi})$. We can proceed with variational Bayesian inference by randomly initializing $\boldsymbol{\phi}$ and performing stochastic gradient ascent on $\mathcal{L}(\boldsymbol{\phi})$ (6.1).

6.2.2 *Variance of the SGVB estimator*

The theory of stochastic approximation tells us that stochastic gradient ascent using (6.3) will asymptotically converge to a local optimum for an appropriately declining step size and sufficient weight updates Robbins and Monro [1951], but in practice the performance of stochastic gradient ascent crucially depends on the variance of the gradients. If this variance is too large, stochastic gradient descent will fail to make much progress in any reasonable amount of time. Our objective function consists of an expected log likelihood term that we approximate using Monte Carlo, and a KL divergence term $D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{w})||p(\mathbf{w}))$ that we assume can be calculated analytically and otherwise be approximated with Monte Carlo with similar reparameterization.

Assume that we draw minibatches of datapoints with replacement. Using L_i as shorthand for $\log p(\mathbf{y}^{(i)}|\mathbf{x}^{(i)}, \mathbf{w} = f(\boldsymbol{\epsilon}^{(i)}, \boldsymbol{\phi}))$, the contribution to the

likelihood for the i-th datapoint in the minibatch, the Monte Carlo estimator (6.3) may be rewritten as $L_{\mathcal{D}}^{\text{SGVB}}(\boldsymbol{\phi}) = \frac{N}{M}\sum_{i=1}^{M} L_i$, whose variance is given by

$$\text{Var}\left[L_{\mathcal{D}}^{\text{SGVB}}(\boldsymbol{\phi})\right] = \frac{N^2}{M^2}\Big(\sum_{i=1}^{M}\text{Var}\left[L_i\right] + 2\sum_{i=1}^{M}\sum_{j=i+1}^{M}\text{Cov}\left[L_i, L_j\right]\Big) \tag{6.4}$$

$$= N^2\Big(\frac{1}{M}\text{Var}\left[L_i\right] + \frac{M-1}{M}\text{Cov}\left[L_i, L_j\right]\Big), \tag{6.5}$$

where the variances and covariances are w.r.t. both the data distribution and $\boldsymbol{\epsilon}$ distribution, i.e. $\text{Var}\left[L_i\right] = \text{Var}_{\boldsymbol{\epsilon}, \mathbf{x}^{(i)}, \mathbf{y}^{(i)}}\Big[\log p(\mathbf{y}^{(i)}|\mathbf{x}^{(i)}, \mathbf{w} = f(\boldsymbol{\epsilon}, \boldsymbol{\phi}))\Big]$, with $\mathbf{x}^{(i)}, \mathbf{y}^{(i)}$ drawn from the empirical distribution defined by the training set. As can be seen from (6.5), the total contribution to the variance by $\text{Var}\left[L_i\right]$ is inversely proportional to the minibatch size M. However, the total contribution by the covariances does not decrease with M. In practice, this means that the variance of $L_{\mathcal{D}}^{\text{SGVB}}(\boldsymbol{\phi})$ can be dominated by the covariances for even moderately large M.

6.2.3 *Local Reparameterization Trick*

We therefore propose an alternative estimator for which we have $\text{Cov}\left[L_i, L_j\right] = 0$, so that the variance of our stochastic gradients scales as $1/M$. We then make this new estimator computationally efficient by not sampling $\boldsymbol{\epsilon}$ directly, but only sampling the intermediate variables $f(\boldsymbol{\epsilon})$ through which $\boldsymbol{\epsilon}$ influences $L_{\mathcal{D}}^{\text{SGVB}}(\boldsymbol{\phi})$. By doing so, the global uncertainty in the weights is translated into a form of local uncertainty that is independent across examples and easier to sample. We refer to such a reparameterization from global noise to local noise as the *local reparameterization trick*. Whenever a source of global noise can be translated to local noise in the intermediate states of computation ($\boldsymbol{\epsilon} \rightarrow f(\boldsymbol{\epsilon})$), a local reparameterization can be applied to yield a computationally and statistically efficient gradient estimator.

Such local reparameterization applies to a fairly large family of models, but is best explained through a simple example: Consider a standard fully connected neural network containing a hidden layer consisting of 1000 neurons. This layer receives an $M \times 1000$ input feature matrix $\mathbf{A}$ from the layer below, which is multiplied by a 1000×1000 weight matrix $\mathbf{W}$, before a nonlinearity is applied, i.e. $\mathbf{B} = \mathbf{AW}$. We then specify the posterior approximation on the weights to be a fully factorized Gaussian, i.e. $q_{\boldsymbol{\phi}}(w_{i,j}) = \mathcal{N}(\mu_{i,j}, \sigma_{i,j}^2)\ \forall w_{i,j} \in \mathbf{W}$, which means the weights are sampled as

$w_{i,j} = \mu_{i,j} + \sigma_{i,j}\epsilon_{i,j}$, with $\epsilon_{i,j} \sim \mathcal{N}(0,1)$. In this case we could make sure that Cov $\left[L_i, L_j\right] = 0$ by sampling a separate weight matrix $\mathbf{W}$ for each example in the minibatch, but this is not computationally efficient: we would need to sample M times a million random numbers for just a single layer of the neural network. Even if this could be done efficiently, the computation following this step would become much harder: Where we originally performed a simple matrix-matrix product of the form $\mathbf{B} = \mathbf{AW}$, this now turns into M separate local vector-matrix products. The theoretical complexity of this computation is higher, but, more importantly, such a computation can usually not be performed in parallel using fast device-optimized BLAS (Basic Linear Algebra Subprograms). This also happens with other neural network architectures such as convolutional neural networks, where optimized libraries for convolution cannot deal with separate filter matrices per example.

Fortunately, the weights (and therefore $\boldsymbol{\epsilon}$) only influence the expected log likelihood through the neuron activations $\mathbf{B}$, which are of much lower dimension. If we can therefore sample the random activations $\mathbf{B}$ directly, without sampling $\mathbf{W}$ or $\boldsymbol{\epsilon}$, we may obtain an efficient Monte Carlo estimator at a much lower cost. For a factorized Gaussian posterior on the weights, the posterior for the activations (conditional on the input $\mathbf{A}$) is also factorized Gaussian:

$$q_{\boldsymbol{\phi}}(w_{i,j}) = \mathcal{N}(\mu_{i,j}, \sigma_{i,j}^2)\ \forall w_{i,j} \in \mathbf{W} \implies q_{\boldsymbol{\phi}}(b_{m,j}|\mathbf{A}) = \mathcal{N}(\gamma_{m,j}, \delta_{m,j}), \text{ with}$$
$$\gamma_{m,j} = \sum_{i=1}^{1000} a_{m,i}\mu_{i,j}, \quad \text{and} \quad \delta_{m,j} = \sum_{i=1}^{1000} a_{m,i}^2\sigma_{i,j}^2. \tag{6.6}$$

Rather than sampling the Gaussian weights and then computing the resulting activations, we may thus sample the activations from their implied Gaussian distribution directly, using $b_{m,j} = \gamma_{m,j} + \sqrt{\delta_{m,j}}\zeta_{m,j}$, with $\zeta_{m,j} \sim \mathcal{N}(0,1)$. Here, ζ is an $M \times 1000$ matrix, so we only need to sample M times a thousand random variables instead of M times a million: a thousand fold savings.

6.3 VARIATIONAL DROPOUT

Dropout is a technique for regularization of neural network parameters, which works by adding multiplicative noise to the input of each layer of the neural

network during optimization. Using the notation of section 6.2.3, for a fully connected neural network dropout corresponds to:

$$\mathbf{B} = (\mathbf{A} \circ \xi)\theta, \quad \text{with} \quad \xi_{i,j} \sim p(\xi_{i,j}) \tag{6.7}$$

where $\mathbf{A}$ is the $M \times K$ matrix of input features for the current minibatch, θ is a $K \times L$ weight matrix, and $\mathbf{B}$ is the $M \times L$ output matrix for the current layer (before a nonlinearity is applied). The $\circ$ symbol denotes the element-wise (Hadamard) product of the input matrix with a $M \times K$ matrix of independent noise variables ξ. By adding noise to the input during training, the weight parameters θ are less likely to overfit to the training data, as shown empirically by previous publications. Originally, Hinton et al. [2012] proposed drawing the elements of ξ from a Bernoulli distribution with probability $1 - p$, with p the *dropout rate*. Later it was shown that using a continuous distribution with the same relative mean and variance, such as a Gaussian $\mathcal{N}(1, \alpha)$ with $\alpha = p/(1-p)$, works as well or better Srivastava et al. [2014].

Here, we re-interpret dropout with continuous noise as a variational method, and propose a generalization that we call *variational dropout*. In developing variational dropout we provide a firm Bayesian justification for dropout training by deriving its implicit prior distribution and variational objective. This new interpretation allows us to propose several useful extensions to dropout, such as a principled way of making the normally fixed dropout rates p *adaptive* to the data.

6.3.1 *Variational dropout with independent weight noise*

If the elements of the noise matrix ξ are drawn independently from a Gaussian $\mathcal{N}(1, \alpha)$, the marginal distributions of the activations $b_{m,j} \in \mathbf{B}$ are Gaussian as well:

$$q_{\boldsymbol{\phi}}(b_{m,j}|\mathbf{A}) = \mathcal{N}(\gamma_{m,j}, \delta_{m,j}), \text{ with } \gamma_{m,j} = \sum_{i=1}^{K} a_{m,i}\theta_{i,j}, \text{ and } \delta_{m,j} = \alpha \sum_{i=1}^{K} a_{m,i}^2 \theta_{i,j}^2. \tag{6.8}$$

Making use of this fact, Wang and Manning [2013] proposed *Gaussian dropout*, a regularization method where, instead of applying (6.7), the activations are directly drawn from their (approximate or exact) marginal distributions as given by (6.8). Wang and Manning [2013] argued that these marginal distributions are exact for Gaussian noise ξ, and for Bernoulli noise still approximately Gaussian because of the central limit theorem. This ignores the

dependencies between the different elements of $\mathbf{B}$, as present using (6.7), but Wang and Manning [2013] report good results nonetheless.

As noted by Wang and Manning [2013], and explained in appendix 6.B, this Gaussian dropout noise can also be interpreted as arising from a Bayesian treatment of a neural network with weights $\mathbf{W}$ that multiply the input to give $\mathbf{B} = \mathbf{AW}$, where the posterior distribution of the weights is given by a factorized Gaussian with $q_{\boldsymbol{\phi}}(w_{i,j}) = \mathcal{N}(\theta_{i,j}, \alpha\theta_{i,j}^2)$. From this perspective, the marginal distributions (6.8) then arise through the application of the local reparameterization trick, as introduced in section 6.2.3. The variational objective corresponding to this interpretation is discussed in section 6.3.3.

6.3.2 *Variational dropout with correlated weight noise*

Instead of ignoring the dependencies of the activation noise, as in section 6.3.1, we may retain the dependencies by interpreting dropout (6.7) as a form of *correlated weight noise*:

$$\begin{aligned} \mathbf{B} = (\mathbf{A} \circ \xi)\theta,\ \xi_{i,j} \sim \mathcal{N}(1, \alpha) \iff \mathbf{b}^m = \mathbf{a}^m\mathbf{W},\ \text{with} \\ \mathbf{W} = (\mathbf{w}_1', \mathbf{w}_2', \ldots, \mathbf{w}_K')',\ \text{and}\ \mathbf{w}_i = s_i\theta_i,\ \text{with}\ q_{\boldsymbol{\phi}}(s_i) = \mathcal{N}(1, \alpha), \end{aligned} \tag{6.9}$$

where $\mathbf{a}^m$ is a row of the input matrix and $\mathbf{b}^m$ a row of the output. The $\mathbf{w}_i$ are the rows of the weight matrix, each of which is constructed by multiplying a non-stochastic parameter vector θ_i by a stochastic scale variable s_i. The distribution on these scale variables we interpret as a Bayesian posterior distribution. The weight parameters θ_i (and the biases) are estimated using maximum likelihood. The original Gaussian dropout sampling procedure (6.7) can then be interpreted as arising from a local reparameterization of our posterior on the weights $\mathbf{W}$.

6.3.3 *Dropout's scale-invariant prior and variational objective*

The posterior distributions $q_{\boldsymbol{\phi}}(\mathbf{W})$ proposed in sections 6.3.1 and 6.3.2 have in common that they can be decomposed into a parameter vector θ that captures the mean, and a multiplicative noise term determined by parameters α. Any posterior distribution on $\mathbf{W}$ for which the noise enters this multiplicative way, we will call a *dropout posterior*. Note that many common distributions, such as univariate Gaussians (with nonzero mean), can be reparameterized to meet this requirement. During dropout training, θ is adapted to

maximize the expected log likelihood $\mathbb{E}_{q_\alpha}[L_\mathcal{D}(\theta)]$. For this to be consistent with the optimization of a variational lower bound of the form in (6.2), the prior on the weights $p(\mathbf{w})$ has to be such that $D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{w})||p(\mathbf{w}))$ does not depend on θ. In appendix 6.C we show that the only prior that meets this requirement is the scale invariant log-uniform prior:

$$p(\log(|w_{i,j}|)) \propto c,$$

i.e. a prior that is uniform on the log-scale of the weights (or the weight-scales s_i for section 6.3.2). As explained in appendix 6.A, this prior has an interesting connection with the floating point format for storing numbers: From an MDL perspective, the floating point format is optimal for communicating numbers drawn from this prior. Conversely, the KL divergence $D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{w})||p(\mathbf{w}))$ with this prior has a natural interpretation as regularizing the number of significant digits our posterior $q_{\boldsymbol{\phi}}$ stores for the weights $w_{i,j}$ in the floating-point format.

Putting the expected log likelihood and KL-divergence penalty together, we see that dropout training maximizes the following variatonal lower bound w.r.t. θ:

$$\mathbb{E}_{q_\alpha}[L_\mathcal{D}(\theta)] - D_{KL}(q_\alpha(\mathbf{w})||p(\mathbf{w})), \tag{6.10}$$

where we have made the dependence on the θ and α parameters explicit. The noise parameters α (e.g. the dropout rates) are commonly treated as hyperparameters that are kept fixed during training. For the log-uniform prior this then corresponds to a fixed limit on the number of significant digits we can learn for each of the weights $w_{i,j}$. In section 6.3.4 we discuss the possibility of making this limit adaptive by also maximizing the lower bound with respect to α.

For the choice of a factorized Gaussian approximate posterior with $q_{\boldsymbol{\phi}}(w_{i,j}) = \mathcal{N}(\theta_{i,j}, \alpha\theta_{i,j}^2)$, as discussed in section 6.3.1, the lower bound (6.10) is analyzed in detail in appendix 6.C. There, it is shown that for this particular choice of posterior the negative KL-divergence $-D_{KL}(q_\alpha(\mathbf{w})||p(\mathbf{w}))$ is not analytically tractable, but can be approximated extremely accurately using

$$-D_{KL}[q_{\boldsymbol{\phi}}(w_i)|p(w_i)] \approx \text{constant} + 0.5\log(\alpha) + c_1\alpha + c_2\alpha^2 + c_3\alpha^3,$$

with

$$c_1 = 1.16145124, \quad c_2 = -1.50204118, \quad c_3 = 0.58629921.$$

The same expression may be used to calculate the corresponding term $-D_{KL}($ $q_\alpha(\mathbf{s})||p(\mathbf{s}))$ for the posterior approximation of section 6.3.2.

6.3.4 *Adaptive regularization through optimizing the dropout rate*

The noise parameters α used in dropout training (e.g. the dropout rates) are usually treated as fixed hyperparameters, but now that we have derived dropout's variational objective (6.10), making these parameters adaptive is trivial: simply maximize the variational lower bound with respect to α. We can use this to learn a separate dropout rate per layer, per neuron, of even per separate weight. In section 6.5 we look at the predictive performance obtained by making α adaptive.

We found that very large values of α correspond to local optima from which it is hard to escape due to large-variance gradients. To avoid such local optima, we found it beneficial to set a constraint $\alpha \leq 1$ during training, i.e. we maximize the posterior variance at the square of the posterior mean, which corresponds to a dropout rate of 0.5.

6.4 RELATED WORK

Pioneering work in practical variational inference for neural networks was done in Graves [2011], where a (biased) variational lower bound estimator was introduced with good results on recurrent neural network models. In later work Kingma and Welling [2013], Rezende et al. [2014] it was shown that even more practical estimators can be formed for most types of continuous latent variables or parameters using a (non-local) reparameterization trick, leading to efficient and unbiased stochastic gradient-based variational inference. These works focused on an application to latent-variable inference; extensive empirical results on inference of global model parameters were reported in Blundell et al. [2015], including successful application to reinforcement learning. These earlier works used the relatively high-variance estimator (6.3), upon which we improve. Variable reparameterizations have a long history in the statistics literature, but have only recently found use for efficient *gradient-based* machine learning and inference Bengio [2013], Kingma [2013], Salimans and Knowles [2013]. Related is also *probabilistic backpropagation* Hernández-Lobato and Adams [2015], an algorithm for inferring marginal posterior probabilities; however, it requires certain tractabilities in the network making it unsuitable for the type of models under consideration in this work.

As we show here, regularization by dropout Srivastava et al. [2014], Wang and Manning [2013] can be interpreted as variational inference. DropConnect Wan et al. [2013] is similar to dropout, but with binary noise on the weights rather than hidden units. DropConnect thus has a similar interpretation as variational inference, with a uniform prior over the weights, and a mixture of two Dirac peaks as posterior. In Ba and Frey [2013], *standout* was introduced, a variation of dropout where a binary belief network is learned for producing dropout rates. Recently, Maeda [2014] proposed another Bayesian perspective on dropout. In recent work Bayer et al. [2015], a similar reparameterization is described and used for variational inference; their focus is on closed-form approximations of the variational bound, rather than unbiased Monte Carlo estimators. Maeda [2014] and Gal and Ghahramani [2015] also investigate a Bayesian perspective on dropout, but focus on the binary variant. Gal and Ghahramani [2015] reports various encouraging results on the utility of dropout's implied prediction uncertainty.

6.5 EXPERIMENTS

We compare our method to standard binary dropout and two popular versions of Gaussian dropout, which we'll denote with type A and type B. With Gaussian dropout type A we denote the pre-linear Gaussian dropout from Srivastava et al. [2014]; type B denotes the post-linear Gaussian dropout from Wang and Manning [2013]. This way, the method names correspond to the matrix names in section 2 (**A** or **B**) where noise is injected. Models were implemented in Theano Bergstra et al. [2010], and optimization was performed using Adam Kingma and Ba [2015] with default hyper-parameters and temporal averaging.

Two types of variational dropout were included. Type A is correlated weight noise as introduced in section 6.3.2: an adaptive version of Gaussian dropout type A. Variational dropout type B has independent weight uncertainty as introduced in section 6.3.1, and corresponds to Gaussian dropout type B.

A *de facto* standard benchmark for regularization methods is the task of MNIST hand-written digit classification. We choose the same architecture as Srivastava et al. [2014]: a fully connected neural network with 3 hidden layers and rectified linear units (ReLUs). We follow the dropout hyper-parameter recommendations from these earlier publications, which is

a dropout rate of $p = 0.5$ for the hidden layers and $p = 0.2$ for the input layer. We used early stopping with all methods, where the amount of epochs to run was determined based on performance on a validation set.

6.5.1 *Variance*

We start out by empirically comparing the variance of the different available stochastic estimators of the gradient of our variational objective. To do this we train the neural network described above for either 10 epochs (test error 3%) or 100 epochs (test error 1.3%), using variational dropout with independent weight noise. After training, we calculate the gradients for the weights of the top and bottom level of our network on the full training set, and compare against the gradient estimates per batch of $M = 1000$ training examples.

Table 6.1 shows that the local reparameterization trick yields the lowest variance among all variational dropout estimators for all conditions, although it is still substantially higher compared to not having any dropout regularization. The $1/M$ variance scaling achieved by our estimator is especially important early on in the optimization when it makes the largest difference (compare weight sample *per minibatch* and weight sample *per data point*). The additional variance reduction obtained by our estimator through drawing fewer random numbers (section 6.2.3) is about a factor of 2, and this remains relatively stable as training progresses (compare *local reparameterization* and *weight sample per data point*).

stochastic gradient estimator	top layer 10 epochs	top layer 100 epochs	bottom layer 10 epochs	bottom layer 100 epochs
local reparameterization (ours)	7.8×10^3	1.2×10^3	1.9×10^2	1.1×10^2
sample per data point (slow)	1.4×10^4	2.6×10^3	4.3×10^2	2.5×10^2
sample per minibatch (standard)	4.9×10^4	4.3×10^3	8.5×10^2	3.3×10^2
no dropout noise (minimal var.)	2.8×10^3	5.9×10^1	1.3×10^2	9.0×10^0

Table 6.1: Average empirical variance of minibatch stochastic gradient estimates (1000 examples) for a fully connected neural network, regularized by variational dropout with independent weight noise.

6.5.2 *Speed*

We compared the regular SGVB estimator, with separate weight samples per datapoint with the efficient estimator based on local reparameterization, in terms of wall-clock time efficiency. With our implementation on a modern GPU, optimization with the naïve estimator took **1635 seconds** per epoch, while the efficient estimator took **7.4 seconds**: an over 200 fold speedup.

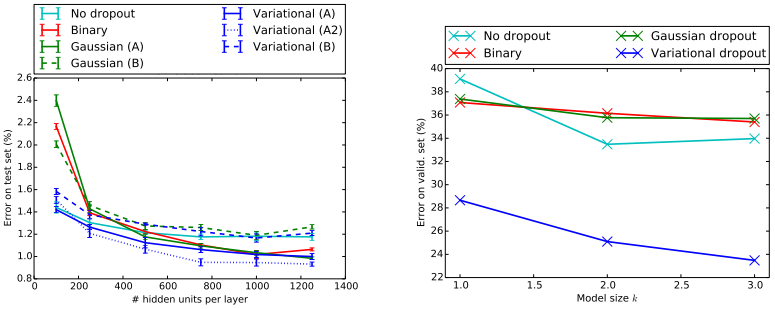
6.5.3 *Classification error*

Figure 6.1 shows test-set classification error for the tested regularization methods, for various choices of number of hidden units. Our adaptive variational versions of Gaussian dropout perform equal or better than their non-adaptive counterparts and standard dropout under all tested conditions. The difference is especially noticeable for the smaller networks. In these smaller networks, we observe that variational dropout infers dropout rates that are on average far lower than the dropout rates for larger networks. This adaptivity comes at negligible computational cost.

We found that slightly downscaling the KL divergence part of the variational objective can be beneficial. *Variational (A2)* in figure 6.1 denotes performance of type A variational dropout but with a KL-divergence downscaled with a factor of 3; this small modification seems to prevent under-fitting, and beats all other dropout methods in the tested models.

6.6 CONCLUSION

Efficiency of posterior inference using stochastic gradient-based variational Bayes (SGVB) can often be significantly improved through a *local reparameterization* where global parameter uncertainty is translated into local uncertainty per datapoint. By injecting noise locally, instead of globally at the model parameters, we obtain an efficient estimator that has low computational complexity, can be trivially parallelized and has low variance. We show how dropout is a special case of SGVB with local reparameterization, and suggest *variational dropout*, a straightforward extension of regular dropout where optimal dropout rates are inferred from the data, rather than fixed in advance. We report encouraging empirical results.



(a) Results on the MNIST dataset (b) Results on the CIFAR-10 dataset

Figure 6.1: Best viewed in color. **(a)** Comparison of various dropout methods, when applied to fully-connected neural networks for classification on the MNIST dataset. Shown is the classification error of networks with 3 hidden layers, averaged over 5 runs. he variational versions of Gaussian dropout perform equal or better than their non-adaptive counterparts; the difference is especially large with smaller models, where regular dropout often results in severe under-fitting **(b)** Comparison of dropout methods when applied to convolutional net a trained on the CIFAR-10 dataset, for different settings of network size k. The network has two convolutional layers with each $32k$ and $64k$ feature maps, respectively, each with stride 2 and followed by a softplus nonlinearity. This is followed by two fully connected layers with each $128k$ hidden units.

CHAPTER APPENDIX

6.A FLOATING-POINT NUMBERS AND COMPRESSION

The *floating-point* format is by far the most common number format used for model parameters in neural network type models. The absolute value of floating-point numbers in base 2 equals $sign \cdot (s/2^{p-1}) \cdot 2^e$ where s is the mantissa with p bits and e is the exponent, both non-negative integer-valued. For random bit patterns for s and e, floating point numbers closely follow a log-uniform distribution with finite range. The floating-point format is thus close to optimal, from a compression point of view, for communicating parameters for which the receiver has a log-uniform prior distribution, with finite range, over the magnitude of parameters. Specifically, this prior for each individual floating-point parameter w_j is $p(\log|w_j|) = \mathcal{U}(\text{interval})$, where the interval is approximately $(\log(2^{-8}), \log(2^8))$ for single-point precision and $(\log(2^{-11}), \log(2^{11}))$ for double-precision floating-point numbers in the most common IEEE specification.

The KL-divergence between a variational posterior $q_{\boldsymbol{\phi}}$ (eq. (6.1)) and this log-uniform prior is relatively simple: it equals the entropy under $q_{\boldsymbol{\phi}}$ of the log-transformed magnitude of $\mathbf{w}$ plus a constant:

$$-D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{w})||p(\mathbf{w})) = \mathcal{H}(q_{\boldsymbol{\phi}}(\log|\mathbf{w}|)) + \text{constant} \tag{6.11}$$

This divergence has a natural interpretation as controlling the number of significant digits under $q_{\boldsymbol{\phi}}$ of the corresponding floating-point number.

6.B DERIVATION OF DROPOUT'S IMPLICIT VARIATIONAL POSTERIOR

In the common version of dropout Srivastava et al. [2014], linear operations $\mathbf{b}^{(i)} = \mathbf{a}^{(i)}\mathbf{W}$ in a model, where $\mathbf{a}^{(i)}$ is a column vector that is the result of

previous operations on the input for datapoint i, are replaced by a stochastic operation:

$$\mathbf{b}^{(i)} = (\mathbf{a}^{(i)} \odot (\mathbf{d}^{(i)}/(1-p))\mathbf{W} \tag{6.12}$$

$$\text{i.e.:} \quad b_k^{(i)} = \sum_j W_{jk} a_j^{(i)} d_j^{(i)}/(1-p) \tag{6.13}$$

$$d_j^{(i)} \sim \text{Bernoulli}(1-p) \tag{6.14}$$

$$p = \text{dropout rate (hyper-parameter, e.g. 0.5)} \tag{6.15}$$

where column vector $\mathbf{a}^{(i)}$ is the result of previous operations on the input for datapoint i. Expected values and variance w.r.t. the dropout rate are:

$$\mathbb{E}\left[d_j^{(i)}\right] = (1-p) \tag{6.16}$$

$$\mathbb{E}\left[d_j^{(i)}/(1-p)\right] = 1 \tag{6.17}$$

$$\text{Var}\left[d_j^{(i)}/(1-p)\right] = \text{Var}\left[d_j^{(i)}\right]/(1-p)^2 = p/(1-p) \tag{6.18}$$

$$\tag{6.19}$$

The expected value and variance for the elements of the resulting vector $\mathbf{b}^{(i)}$ are:

$$\mathbb{E}\left[b_k^{(i)}\right] = \mathbb{E}\left[\sum_j W_{jk} a_j^{(i)} d_j^{(i)}/(1-p)\right] \tag{6.20}$$

$$= \sum_j W_{jk} a_j^{(i)} \mathbb{E}\left[d_j^{(i)}/(1-p)\right] \tag{6.21}$$

$$= \sum_j W_{jk} a_j^{(i)} \tag{6.22}$$

$$\text{Var}\left[b_k^{(i)}\right] = \text{Var}\left[\sum_j W_{jk} a_j^{(i)} d_j^{(i)}/(1-p)\right] \tag{6.23}$$

$$= \sum_j \text{Var}\left[W_{jk} a_j^{(i)} d_j^{(i)}/(1-p)\right] \tag{6.24}$$

$$= \sum_j W_{jk}^2 (a_j^{(i)})^2 \text{Var}\left[d_j^{(i)}/(1-p)\right] \tag{6.25}$$

$$= p/(1-p) \sum_j W_{jk}^2 (a_j^{(i)})^2 \tag{6.26}$$

6.B.1 *Gaussian dropout*

Thus, ignoring dependencies between elements of $\mathbf{b}^{(i)}$, dropout can be approximated by a Gaussian Wang and Manning [2013]:

$$p(\mathbf{b}^{(i)}|\mathbf{a}^{(i)}, W) = \mathcal{N}(\mathbf{a}^{(i)}W, p(1-p)^{-1}(\mathbf{a}^{(i)})^2(\mathbf{W})^2) \tag{6.27}$$

$$\text{i.e.:} \quad \mathbf{b}^{(i)} = \mathbf{a}^{(i)}W + \boldsymbol{\epsilon}^{(i)} \odot \sqrt{p(1-p)^{-1}(\mathbf{a}^{(i)})^2(\mathbf{W})^2} \tag{6.28}$$

$$\text{where:} \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, I) \tag{6.29}$$

where $(.)^2$, $\sqrt{(.)}$ and $\odot$ are again component-wise operations.

6.B.2 *Weight uncertainty*

Equivalently, the Gaussian dropout relationship $p(\mathbf{b}^{(i)}|\mathbf{a}^{(i)}, W)$ can be parameterized as weight uncertainty:

$$\mathbf{b}^{(i)} = \mathbf{a}^{(i)}\mathbf{V} \tag{6.30}$$

$$p(V_{jk}|W_{jk}, \alpha_{jk}) = \mathcal{N}(W_{jk}, \alpha_{jk}W_{jk}^2) \tag{6.31}$$

$$\alpha_{jk} = p/(1-p) \tag{6.32}$$

$$\Rightarrow \quad p(\mathbf{b}^{(i)}|\mathbf{a}^{(i)}, W) = \mathcal{N}(\mathbf{a}^{(i)}W, p(1-p)^{-1}(\mathbf{a}^{(i)})^2(\mathbf{W})^2) \tag{6.33}$$

In the main text, we show how we can treat the distribution over weights $p(V_{jk}|W_{jk}, \alpha_{jk})$ as a variational posterior $q_{\boldsymbol{\phi}}$, which we can optimize w.r.t. each α_{jk} (the dropout rate for individual parameters) to optimize a bound on the marginal likelihood of the data.

6.C NEGATIVE KL-DIVERGENCE FOR THE LOG-UNIFORM PRIOR

The variational lower bound (6.1) that we wish to optimize is given by

$$\mathcal{L}(\boldsymbol{\phi}) = -D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{w})||p(\mathbf{w})) + L_{\mathcal{D}}(\boldsymbol{\phi}),$$

where $L_{\mathcal{D}}(\boldsymbol{\phi})$ is the expected log likelihood term which we approximate using Monte Carlo, as discussed in section 6.2.3. Here we discuss how to deterministically compute the negative KL divergence term $-D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{w})||p(\mathbf{w}))$

for the *log-uniform* prior proposed in section 6.B. For simplicity we only discuss the case of factorizing approximate posterior $q_{\boldsymbol{\phi}}(\mathbf{w}) = \prod_{i=1}^{k} q_{\boldsymbol{\phi}}(w_i)$, although the extension to more involved posterior approximations is straight forward.

Our log-uniform prior $p(w_i)$ (appendix 6.A) consists of a Bernoulli distribution on the *sign bit* $s_i \in \{-1, +1\}$ which captures the sign of w_i, combined with a uniform prior on the log-scale of w_i:

$$\begin{aligned} w_i &= s_i|w_i| \\ p(s_i) &= \text{Bernoulli}(0.5) \text{ on } \{-1, +1\} \\ p(\log(|w_i|)) &\propto c \end{aligned} \tag{6.34}$$

For an approximate posterior $q_{\boldsymbol{\phi}}(w_i)$ with support on the real line, the KL divergence will thus consist of two parts: the KL-divergence between the posterior and prior on the sign bit s_i and the KL-divergence between the conditional posterior and prior on the log scale $\log(|w_i|)$.

The negative KL divergence between the posterior and prior on the sign bit is given by:

$$\begin{aligned} &- D_{KL}[q_{\boldsymbol{\phi}}(s_i)|p(s_i)] \\ &= \log(0.5) - Q_\phi(0)\log[Q_\phi(0)] - [1 - Q_\phi(0)]\log[1 - Q_\phi(0)], \end{aligned} \tag{6.35}$$

with $Q_\phi(0)$ the posterior CDF of w_i evaluated at 0, i.e. the probability of drawing a negative weight from our approximate posterior.

The negative KL divergence of the second part (the log-scale) is then

$$\begin{aligned} &-D_{KL}[q_{\boldsymbol{\phi}}(\log(|w_i|)|s_i)|p(\log(|w_i|))] \\ &= \log(c) - Q_\phi(0)\mathbb{E}_{q_{\boldsymbol{\phi}}(w_i|w_i<0)}\left[\log(q_{\boldsymbol{\phi}}(w_i)/Q_\phi(0)) + \log(|w_i|)\right] \\ &- \left(1 - Q_\phi(0)\right)\mathbb{E}_{q_{\boldsymbol{\phi}}(w_i|w_i>0)}\left[\log(q_{\boldsymbol{\phi}}(w_i)/(1 - Q_\phi(0))) + \log(|w_i|)\right], \end{aligned} \tag{6.36}$$

where $q_{\boldsymbol{\phi}}(w_i)/Q_\phi(0)$ is the truncation of our approximate posterior to the negative part of the real-line, and the $\log(|w_i|)$ term enters through transforming the uniform prior from the log-space to the normal space.

Putting both parts together the terms involving the CDF $Q_\phi(0)$ cancel, and we get:

$$-D_{KL}[q_{\boldsymbol{\phi}}(w_i)|p(w_i)] = \log(c) + \log(0.5) + \mathcal{H}(q_{\boldsymbol{\phi}}(w_i)) - \mathbb{E}_q \log(|w_i|), \tag{6.37}$$

where $\mathcal{H}(q_{\boldsymbol{\phi}}(w_i))$ is the entropy of our approximate posterior.

We now consider the special case of a *dropout posterior*, as proposed in section 6.3, which we defined as any approximate posterior distribution $q_{\boldsymbol{\phi}}(w_i)$ for which the weight noise is *multiplicative*:

$$w_i = \theta_i \epsilon_i, \text{ with } \epsilon_i \sim q_\alpha(\epsilon_i), \tag{6.38}$$

where we have divided the parameters $\phi_i = (\theta_i, \alpha)$ into parameters governing the mean of $q_{\boldsymbol{\phi}}(w_i)$ (assuming $\mathbb{E}_q \epsilon_i = 1$) and the multiplicative noise. Note that certain families of distributions, such as Gaussians, can always be reparameterized in this way.

Assuming $q_\alpha(\epsilon_i)$ is a continuous distribution, this choice of posterior gives us

$$\mathcal{H}(q_{\boldsymbol{\phi}}(w_i)) = \mathcal{H}(q_\alpha(\epsilon_i)) + \log(|\theta_i|), \tag{6.39}$$

where the last term can be understood as the log Jacobian determinant of a linear transformation of the random variable ϵ_i.

Furthermore, we can rewrite the $-\mathbb{E}_q \log(|w_i|)$ term of (6.37) as

$$-\mathbb{E}_q \log(|w_i|) = -\mathbb{E}_q \log(|\theta_i \epsilon_i|) = -\log(|\theta_i|) - \mathbb{E}_q \log(|\epsilon_i|). \tag{6.40}$$

Taking (6.39) and (6.40) together, the terms depending on θ thus cancel exactly, proving that the KL divergence between posterior and prior is indeed independent of these parameters:

$$\begin{aligned} -D_{KL}[q_{\boldsymbol{\phi}}(w_i)|p(w_i)] &= \log(c) + \log(0.5) + \mathcal{H}(q_{\boldsymbol{\phi}}(w_i)) - \mathbb{E}_q \log(|w_i|) \\ &= \log(c) + \log(0.5) + \mathcal{H}(q_\alpha(\epsilon_i)) - \mathbb{E}_{q_\alpha} \log(|\epsilon_i|). \end{aligned} \tag{6.41}$$

In fact, the log-uniform prior is the only prior consistent with dropout. If the prior had any additional terms involving w_i, their expectation w.r.t. $q_{\boldsymbol{\phi}}$ would not cancel with the entropy term in $-D_{KL}[q_{\boldsymbol{\phi}}(w_i)|p(w_i)]$.

For the specific case of *Gaussian dropout*, as proposed in section 6.3, we have $q_\alpha(\epsilon_i) = \mathcal{N}(1, \alpha)$. This then gives us

$$\begin{aligned} &- D_{KL}[q_{\boldsymbol{\phi}}(w_i)|p(w_i)] \\ &= \log(c) + \log(0.5) + 0.5(1 + \log(2\pi) + \log(\alpha)) - \mathbb{E}_{q_\alpha} \log(|\epsilon_i|). \end{aligned}$$

The final term $\mathbb{E}_{q_\alpha} \log(|\epsilon_i|)$ in this expression is not analytically tractable, but it can be pre-computed numerically for all values of α in the relevant range

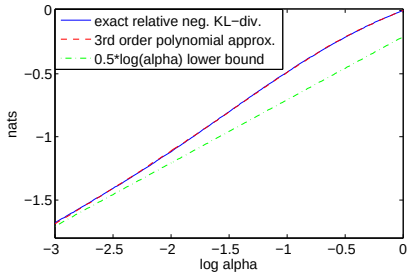


Figure 6.C.1: Exact and approximate negative KL divergence $-D_{KL}[q_{\boldsymbol{\phi}}(w_i)|p(w_i)]$ for the *log-uniform* prior and *Gaussian dropout* approximate posterior. Since we use an improper prior, the constant is arbitrary; we choose it so that the exact KL divergence is 0 at $\alpha = 1$, which is the maximum value we allow during optimization of the lower bound. The exact and approximate results are indistinguishable on the domain shown, which corresponds to dropout rates between $p = 0.05$ and $p = 0.5$. For $\log(\alpha) \to -\infty$ the approximated term vanishes, so that both our polynomial approximation and the lower bound become exact.

and then approximated using a 3rd degree polynomial in α. As shown in figure 6.C.1, this approximation is extremely close.

For Gaussian dropout this then gives us:

$$-D_{KL}[q_{\boldsymbol{\phi}}(w_i)|p(w_i)] \approx \text{constant} + 0.5\log(\alpha) + c_1\alpha + c_2\alpha^2 + c_3\alpha^3,$$

with

$$c_1 = 1.16145124, \qquad c_2 = -1.50204118, \qquad c_3 = 0.58629921.$$

Alternatively, we can make use of the fact that $-\mathbb{E}_{q_\alpha}\log(|\epsilon_i|) \geq 0$ for all values of α, and use the following lower bound:

$$-D_{KL}[q_{\boldsymbol{\phi}}(w_i)|p(w_i)] \geq \text{constant} + 0.5\log(\alpha).$$

7

ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION

We introduce *Adam*, an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. The method is straightforward to implement, is computationally efficient, has little memory requirements, is invariant to diagonal rescaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters. The method is also appropriate for non-stationary objectives and problems with very noisy and/or sparse gradients. The hyper-parameters have intuitive interpretations and typically require little tuning. Some connections to related algorithms, on which *Adam* was inspired, are discussed. Empirical results demonstrate that Adam works well in practice and compares favorably to other stochastic optimization methods. Finally, we discuss *AdaMax*, a variant of *Adam* based on the infinity norm.[1]

7.1 INTRODUCTION

Stochastic gradient-based optimization is of core practical importance in many fields of science and engineering. Many problems in these fields can be cast as the optimization of some scalar parameterized objective function requiring maximization or minimization with respect to its parameters. If the function is differentiable w.r.t. its parameters, gradient descent is a relatively efficient optimization method, since the computation of first-order partial derivatives w.r.t. all the parameters is of the same computational complexity as just evaluating the function. Often, objective functions are stochastic. For example, many objective functions are composed of a sum of sub-functions

[1] This chapter is adapted from our publication [Kingma and Ba, 2015].

evaluated at different subsamples of data; in this case optimization can be made more efficient by taking gradient steps w.r.t. individual sub-functions, i.e. stochastic gradient descent (SGD) or ascent. SGD proved itself as an efficient and effective optimization method that was central in many machine learning success stories, such as recent advances in deep learning. Objectives may also have other sources of noise than data sub-sampling, such as dropout [Srivastava et al., 2014] regularization. For all such noisy objectives, efficient stochastic optimization techniques are required. The focus of this work is on the optimization of stochastic objectives with high-dimensional parameters spaces. In these cases, higher-order optimization methods are ill-suited, and discussion in this work will be restricted to first-order methods.

We propose *Adam*, a method for efficient stochastic optimization that only requires first-order gradients with little memory requirement. The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients; the name *Adam* is derived from adaptive moment estimation. Our method is designed to combine the advantages of two recently popular methods: AdaGrad [Duchi et al., 2010], which works well with sparse gradients, and RMSProp [Tieleman and Hinton, 2012], which works well in on-line and non-stationary settings; important connections to these and other stochastic optimization methods are clarified in section 7.4. Some of Adam's advantages are that the magnitudes of parameter updates are invariant to rescaling of the gradient, its step-sizes are approximately bounded by the stepsize hyperparameter, it does not require a stationary objective, it works with sparse gradients, and it naturally performs a form of step size annealing.

In section 7.2 we describe the algorithm and the properties of its update rule. Section 7.3 explains our initialization bias correction technique. Empirically, our method consistently outperforms other methods for a variety of models and datasets, as shown in section 7.5. Overall, we show that Adam is a versatile algorithm that scales to large-scale high-dimensional machine learning problems.

7.2 ALGORITHM

See algorithm 6 for pseudo-code of our proposed algorithm *Adam*. Let $f(\theta)$ be a noisy objective function: a stochastic scalar function that is differentiable w.r.t. parameters θ. We are interested in minimizing the expected value of

Algorithm 6: *Adam*, our proposed algorithm for stochastic optimization. See section 7.2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the element-wise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require α: Step-size
Require $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require $f(\theta)$: Stochastic objective function with parameters θ
Require θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1$^{\text{st}}$ moment vector)
$v_0 \leftarrow 0$ (Initialize 2$^{\text{nd}}$ moment vector)
$t \leftarrow 0$ (Initialize time-step)
while θ_t *not converged* **do**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at time-step t)
  $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
  $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
  $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
  $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
  $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end
Return θ_t (Resulting parameters)

this function, $\mathbb{E}[f(\theta)]$ w.r.t. its parameters θ. With $f_1(\theta), ..., , f_T(\theta)$ we denote the realizations of the stochastic function at subsequent time-steps $1, ..., T$. The stochasticity might come from the evaluation at random subsamples (minibatches) of datapoints, or arise from inherent function noise. With $g_t = \nabla_\theta f_t(\theta)$ we denote the gradient, i.e. the vector of partial derivatives of f_t, w.r.t θ evaluated at time-step t.

The algorithm updates exponential moving averages of the gradient (m_t) and the squared gradient (v_t) where the hyper-parameters $\beta_1, \beta_2 \in [0, 1)$ control the exponential decay rates of these moving averages. The moving averages themselves are estimates of the 1$^{\text{st}}$ moment (the mean) and the 2$^{\text{nd}}$ raw moment (the uncentered variance) of the gradient. However, these moving averages are initialized as (vectors of) 0's, leading to moment estimates

that are biased towards zero, especially during the initial time-steps, and especially when the decay rates are small (i.e. the βs are close to 1). The good news is that this initialization bias can be easily counteracted, resulting in bias-corrected estimates $\widehat{m}_t$ and $\widehat{v}_t$. See section 7.3 for more details.

Note that the efficiency of algorithm 6 can, at the expense of clarity, be improved upon by changing the order of computation, e.g. by replacing the last three lines in the loop with the following lines: $\alpha_t = \alpha \cdot \sqrt{1-\beta_2^t}/(1-\beta_1^t)$ and $\theta_t \leftarrow \theta_{t-1} - \alpha_t \cdot m_t/(\sqrt{v_t} + \hat{\epsilon})$.

7.2.1 *Adam's update rule*

An important property of Adam's update rule is its careful choice of stepsizes. Assuming $\epsilon = 0$, the effective step taken in parameter space at timestep t is $\Delta_t = \alpha \cdot \widehat{m}_t/\sqrt{\widehat{v}_t}$. The effective stepsize has two upper bounds: $|\Delta_t| \leq \alpha \cdot (1-\beta_1)/\sqrt{1-\beta_2}$ in the case $(1-\beta_1) > \sqrt{1-\beta_2}$, and $|\Delta_t| \leq \alpha$ otherwise. The first case only happens in the most severe case of sparsity: when a gradient has been zero at all timesteps except at the current timestep. For less sparse cases, the effective stepsize will be smaller. When $(1-\beta_1) = \sqrt{1-\beta_2}$ we have that $|\widehat{m}_t/\sqrt{\widehat{v}_t}| < 1$ therefore $|\Delta_t| < \alpha$. In more common scenarios, we will have that $\widehat{m}_t/\sqrt{\widehat{v}_t} \approx \pm 1$ since $|\mathbb{E}[g]/\sqrt{\mathbb{E}[g^2]}| \leq 1$. The effective magnitude of the steps taken in parameter space at each timestep are approximately bounded by the stepsize setting α, i.e., $|\Delta_t| \lessapprox \alpha$. This can be understood as establishing a *trust region* around the current parameter value, beyond which the current gradient estimate does not provide sufficient information. This typically makes it relatively easy to know the right scale of α in advance. For many machine learning models, for instance, we often know in advance that good optima are with high probability within some set region in parameter space; it is not uncommon, for example, to have a prior distribution over the parameters. Since α sets (an upper bound of) the magnitude of steps in parameter space, we can often deduce the right order of magnitude of α such that optima can be reached from θ_0 within some number of iterations. With a slight abuse of terminology, we will call the ratio $\widehat{m}_t/\sqrt{\widehat{v}_t}$ the *signal-to-noise* ratio (*SNR*). With a smaller SNR the effective stepsize Δ_t will be closer to zero. This is a desirable property, since a smaller SNR means that there is greater uncertainty about whether the direction of $\widehat{m}_t$ corresponds to the direction of the true gradient. For example, the SNR value typically becomes closer to 0 towards an optimum, leading to smaller

effective steps in parameter space: a form of automatic annealing. The effective stepsize Δ_t is also invariant to the scale of the gradients; rescaling the gradients g with factor c will scale $\widehat{m}_t$ with a factor c and $\widehat{v}_t$ with a factor c^2, which cancel out: $(c \cdot \widehat{m}_t)/(\sqrt{c^2 \cdot \widehat{v}_t}) = \widehat{m}_t/\sqrt{\widehat{v}_t}$.

7.3 INITIALIZATION BIAS CORRECTION

As explained in section 7.2, Adam utilizes initialization bias correction terms. We will here derive the term for the second moment estimate; the derivation for the first moment estimate is completely analogous. Let g be the gradient of the stochastic objective f, and we wish to estimate its second raw moment (uncentered variance) using an exponential moving average of the squared gradient, with decay rate β_2. Let $g_1, ..., g_T$ be the gradients at subsequent timesteps, each a draw from an underlying gradient distribution $g_t \sim p(g_t)$. Let us initialize the exponential moving average as $v_0 = 0$ (a vector of zeros). First note that the update at timestep t of the exponential moving average $v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (where g_t^2 indicates the element-wise square $g_t \odot g_t$) can be written as a function of the gradients at all previous timesteps:

$$v_t = (1 - \beta_2) \sum_{i=1}^{t} \beta_2^{t-i} \cdot g_i^2 \tag{7.1}$$

We wish to know how $\mathbb{E}[v_t]$, the expected value of the exponential moving average at timestep t, relates to the true second moment $\mathbb{E}[g_t^2]$, so we can correct for the discrepancy between the two. Taking expectations of the left-hand and right-hand sides of eq. (7.1):

$$\mathbb{E}[v_t] = \mathbb{E}\left[(1 - \beta_2) \sum_{i=1}^{t} \beta_2^{t-i} \cdot g_i^2\right] \tag{7.2}$$

$$= \mathbb{E}[g_t^2] \cdot (1 - \beta_2) \sum_{i=1}^{t} \beta_2^{t-i} + \zeta \tag{7.3}$$

$$= \mathbb{E}[g_t^2] \cdot (1 - \beta_2^t) + \zeta \tag{7.4}$$

where $\zeta = 0$ if the true second moment $\mathbb{E}[g_i^2]$ is stationary; otherwise ζ can be kept small since the exponential decay rate β_1 can (and should) be chosen such that the exponential moving average assigns small weights to gradients too far in the past. What is left is the term $(1 - \beta_2^t)$ which is caused

by initializing the running average with zeros. In algorithm 6 we therefore divide by this term to correct the initialization bias.

In case of sparse gradients, for a reliable estimate of the second moment one needs to average over many gradients by choosing a small value of β_2; howevere it is exactly this case of small β_2 where a lack of initialization bias correction would lead to initial steps that are much larger.

7.4 RELATED WORK

Optimization methods bearing a direct relation to Adam are RMSProp [Tieleman and Hinton, 2012, Graves, 2013] and AdaGrad [Duchi et al., 2010]; these relationships are discussed below. Other stochastic optimization methods include vSGD [Schaul et al., 2012], AdaDelta [Zeiler, 2012] and the natural Newton method from Le Roux and Fitzgibbon [2010], all setting stepsizes by estimating curvature from first-order information. The Sum-of-Functions Optimizer (SFO) [Sohl-Dickstein et al., 2014] is a quasi-Newton method based on minibatches, but (unlike Adam) has memory requirements linear in the number of minibatch partitions of a dataset, which is often infeasible on memory-constrained systems such as a GPU. Like natural gradient descent (NGD) [Amari, 1998], Adam employs a preconditioner that adapts to the geometry of the data, since $\widehat{v}_t$ is an approximation to the diagonal of the Fisher information matrix [Pascanu and Bengio, 2013]; however, Adam's preconditioner (like AdaGrad's) is more conservative in its adaption than vanilla NGD by preconditioning with the square root of the inverse of the diagonal Fisher information matrix approximation.

7.4.1 *RMSProp*

An optimization method closely related to Adam is RMSProp [Tieleman and Hinton, 2012]. A version with momentum has sometimes been used [Graves, 2013]. There are a few important differences between RMSProp with momentum and Adam: RMSProp with momentum generates its parameter updates using a momentum on the rescaled gradient, whereas Adam updates are directly estimated using a running average of first and second moment of the gradient. RMSProp also lacks a bias-correction term; this matters most in case of a value of β_2 close to 1 (required in case of sparse gradients), since

in that case not correcting the bias leads to very large stepsizes and often divergence, as we also empirically demonstrate in section 7.4.3.

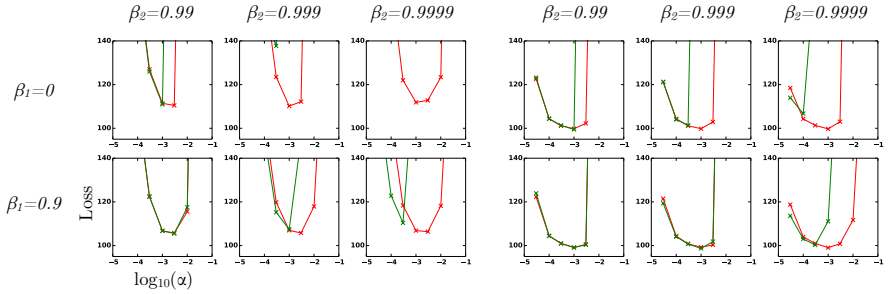
7.4.2 *AdaGrad*

An algorithm that works well for sparse gradients is AdaGrad [Duchi et al., 2010]. Its basic version updates parameters as $\theta_{t+1} = \theta_t - \alpha \cdot g_t / \sqrt{\sum_{i=1}^{t} g_t^2}$. Note that if we choose β_2 to be infinitesimally close to 1 from below, then $\lim_{\beta_2 \to 1} \widehat{v}_t = t^{-1} \cdot \sum_{i=1}^{t} g_t^2$. AdaGrad corresponds to a version of Adam with $\beta_1 = 0$, infinitesimal $(1 - \beta_2)$ and a replacement of α by an annealed version $\alpha_t = \alpha \cdot t^{-1/2}$, namely $\theta_t - \alpha \cdot t^{-1/2} \cdot \widehat{m}_t / \sqrt{\lim_{\beta_2 \to 1} \widehat{v}_t} = \theta_t - \alpha \cdot t^{-1/2} \cdot g_t / \sqrt{t^{-1} \cdot \sum_{i=1}^{t} g_t^2} = \theta_t - \alpha \cdot g_t / \sqrt{\sum_{i=1}^{t} g_t^2}$. Note that this direct correspondence between Adam and Adagrad does not hold when removing the bias-correction terms; without bias correction, like in RMSProp, a β_2 infinitesimally close to 1 would lead to infinitely large bias, and infinitely large parameter updates.

7.4.3 *Experiment: bias-correction term*

We also empirically evaluate the effect of the bias correction terms explained in sections 7.2 and 7.3. Discussed in section 7.4, removal of the bias correction terms results in a version of RMSProp [Tieleman and Hinton, 2012] with momentum. We vary the β_1 and β_2 when training a variational auto-encoder (VAE) with the same architecture as in [Kingma and Welling, 2013] with a single hidden layer with 500 hidden units with softplus nonlinearities and a 50-dimensional spherical Gaussian latent variable. We iterated over a broad range of hyper-parameter choices, i.e. $\beta_1 \in [0, 0.9]$ and $\beta_2 \in [0.99, 0.999, 0.9999]$, and $\log_{10}(\alpha) \in [-5, ..., -1]$. Values of β_2 close to 1, required for robustness to sparse gradients, results in larger initialization bias; therefore we expect the bias correction term is important in such cases of slow decay, preventing an adverse effect on optimization.

In Figure 7.1, values β_2 close to 1 indeed lead to instabilities in training when no bias correction term was present, especially at first few epochs of the training. The best results were achieved with small values of $(1 - \beta_2)$ and bias correction; this was more apparent towards the end of optimization when gradients tends to become sparser as hidden units specialize to spe-



(a) after 10 epochs (b) after 100 epochs

Figure 7.1: Effect of bias-correction terms (red line) versus no bias correction terms (green line) after 10 epochs (left) and 100 epochs (right) on the loss (y-axes) when learning a Variational Auto-Encoder (VAE) [Kingma and Welling, 2013], for different settings of stepsize α (x-axes) and hyper-parameters β_1 and β_2.

cific patterns. In summary, Adam performed equal or better than RMSProp, regardless of hyper-parameter setting.

7.5 EXPERIMENTS

To empirically evaluate the proposed method, we investigated different popular machine learning models, including logistic regression, multilayer fully connected neural networks and deep convolutional neural networks. Using large models and datasets, we demonstrate Adam can efficiently solve practical deep learning problems.

We use the same parameter initialization when comparing different optimization algorithms. The hyper-parameters, such as learning rate and momentum, are searched over a dense grid and the results are reported using the best hyper-parameter setting.

7.5.1 *Experiment: Logistic Regression*

We evaluate our proposed method on L2-regularized multi-class logistic regression using the MNIST dataset. Logistic regression has a well-studied convex objective, making it suitable for comparison of different optimizers

without worrying about local minimum issues. The logistic regression classifies the class label directly on the 784 dimension image vectors. We compare Adam to accelerated SGD with Nesterov momentum and Adagrad using minibatch size of 128. According to Figure 7.1, we found that the Adam yields similar convergence as SGD with momentum and both converge faster than Adagrad.

As discussed in [Duchi et al., 2010], Adagrad can efficiently deal with sparse features and gradients as one of its main theoretical results whereas SGD is low at learning rare features. Adam with $1/\sqrt{t}$ decay on its stepsize should theoretically match the performance of Adagrad. We examine the sparse feature problem using IMDB movie review dataset from [Maas et al., 2011]. We pre-process the IMDB movie reviews into bag-of-words (BoW) feature vectors including the first 10,000 most frequent words. The 10,000 dimension BoW feature vector for each review is highly sparse. As suggested in [Wang and Manning, 2013], 50% dropout noise can be applied to the BoW features during training to prevent over-fitting. In figure 7.1, Adagrad outperforms SGD with Nesterov momentum by a large margin both with and without dropout noise. Adam converges as fast as Adagrad. The empirical performance of Adam is consistent with our theoretical findings in sections 7.2 and our paper [Kingma and Ba, 2015]. Similar to Adagrad, Adam can take advantage of sparse features and obtain faster convergence rate than normal SGD with momentum.

7.5.2 *Experiment: Multi-layer Neural Networks*

Multi-layer neural network are powerful models with non-convex objective functions. Although our convergence analysis does not apply to non-convex problems, we empirically found that Adam often outperforms other methods in such cases. In our experiments, we made model choices that are consistent with previous publications in the area; a neural network model with two fully connected hidden layers with 1000 hidden units each and ReLU activation are used for this experiment with minibatch size of 128.

First, we study different optimizers using the standard deterministic cross-entropy objective function with L_2 weight decay on the parameters to prevent over-fitting. The sum-of-functions (SFO) method [Sohl-Dickstein et al., 2014] is a recently proposed quasi-Newton method that works with minibatches of data and has shown good performance on optimization of multi-layer

neural networks. We used their implementation and compared with Adam to train such models. Figure 7.2 shows that Adam makes faster progress in terms of both the number of iterations and wall-clock time. Due to the cost of updating curvature information, SFO is 5-10x slower per iteration compared to Adam, and has a memory requirement that is linear in the number minibatches.

Stochastic regularization methods, such as dropout, are an effective way to prevent over-fitting and often used in practice due to their simplicity. SFO assumes deterministic sub-functions, and indeed failed to converge on cost functions with stochastic regularization. We compare the effectiveness of Adam to other stochastic first order methods on multi-layer neural networks trained with dropout noise. Figure 7.2 shows our results; Adam shows better convergence than other methods.

7.5.3 *Experiment: Convolutional Neural Networks*

Convolutional neural networks (CNNs) with several layers of convolution, pooling and non-linear units have shown considerable success in computer vision tasks. Unlike most fully connected neural nets, weight sharing in CNNs results in vastly different gradients in different layers. A smaller learning rate for the convolution layers is often used in practice when applying SGD. We show the effectiveness of Adam in deep CNNs. Our CNN architecture has three alternating stages of 5x5 convolution filters and 3x3 max pooling with stride of 2 that are followed by a fully connected layer of 1000 rectified linear hidden units (ReLU's). The input image are preprocessed by whitening, and dropout noise is applied to the input layer and fully connected layer. The minibatch size is also set to 128 similar to previous experiments.

Interestingly, although both Adam and Adagrad make rapid progress lowering the cost in the initial stage of the training, shown in Figure 7.3 (left), Adam and SGD eventually converge considerably faster than Adagrad for CNNs shown in Figure 7.3 (right). We notice the second moment estimate $\widehat{v}_t$ vanishes to zeros after a few epochs and is dominated by the ϵ in algorithm 6. The second moment estimate is therefore a poor approximation to the geometry of the cost function in CNNs comparing to fully connected network from Section 7.5.2. Whereas, reducing the minibatch variance through the first moment is more important in CNNs and contributes to the speed-up.

Algorithm 7: *AdaMax*, a variant of Adam based on the infinity norm. See section 7.6.1 for details. Good default settings for the tested machine learning problems are $\alpha = 0.002$, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. With β_1^t we denote β_1 to the power t. Here, $(\alpha/(1-\beta_1^t))$ is the learning rate with the bias-correction term for the first moment. All operations on vectors are element-wise.

Require α: Stepsize
Require $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates
Require $f(\theta)$: Stochastic objective function with parameters θ
Require θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1^{st} moment vector)
$u_0 \leftarrow 0$ (Initialize the exponentially weighted infinity norm)
$t \leftarrow 0$ (Initialize timestep)
while θ_t *not converged* **do**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
  $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
  $u_t \leftarrow \max(\beta_2 \cdot u_{t-1}, |g_t|)$ (Update the exponentially weighted infinity norm)
  $\theta_t \leftarrow \theta_{t-1} - (\alpha/(1-\beta_1^t)) \cdot m_t/u_t$ (Update parameters)
end
Return θ_t (Resulting parameters)

As a result, Adagrad converges much slower than others in this particular experiment. Though Adam shows marginal improvement over SGD with momentum, it adapts learning rate scale for different layers instead of hand picking manually as in SGD.

7.6 EXTENSIONS

7.6.1 *AdaMax*

In Adam, the update rule for individual weights is to scale their gradients inversely proportional to a (scaled) L^2 norm of their individual current and past gradients. We can generalize the L^2 norm based update rule to a L^p

norm based update rule. Such variants become numerically unstable for large p. However, in the special case where we let $p \to \infty$, a surprisingly simple and stable algorithm emerges; see algorithm 7. We'll now derive the algorithm. Let, in case of the L^p norm, the stepsize at time t be inversely proportional to $v_t^{1/p}$, where:

$$v_t = \beta_2^p v_{t-1} + (1 - \beta_2^p)|g_t|^p \tag{7.5}$$

$$= (1 - \beta_2^p) \sum_{i=1}^{t} \beta_2^{p(t-i)} \cdot |g_i|^p \tag{7.6}$$

Note that the decay term is here equivalently parameterized as β_2^p instead of β_2. Now let $p \to \infty$, and define $u_t = \lim_{p\to\infty}(v_t)^{1/p}$, then:

$$u_t = \lim_{p\to\infty}(v_t)^{1/p} = \lim_{p\to\infty}\left((1 - \beta_2^p) \sum_{i=1}^{t} \beta_2^{p(t-i)} \cdot |g_i|^p\right)^{1/p} \tag{7.7}$$

$$= \lim_{p\to\infty}(1 - \beta_2^p)^{1/p}\left(\sum_{i=1}^{t} \beta_2^{p(t-i)} \cdot |g_i|^p\right)^{1/p} \tag{7.8}$$

$$= \lim_{p\to\infty}\left(\sum_{i=1}^{t} \left(\beta_2^{(t-i)} \cdot |g_i|\right)^p\right)^{1/p} \tag{7.9}$$

$$= \max\left(\beta_2^{t-1}|g_1|, \beta_2^{t-2}|g_2|, \ldots, \beta_2|g_{t-1}|, |g_t|\right) \tag{7.10}$$

Which corresponds to the remarkably simple recursive formula:

$$u_t = \max(\beta_2 \cdot u_{t-1}, |g_t|) \tag{7.11}$$

with initial value $u_0 = 0$. Note that, conveniently enough, we don't need to correct for initialization bias in this case. Also note that the magnitude of parameter updates has a simpler bound with AdaMax than Adam, namely: $|\Delta_t| \leq \alpha$.

7.6.2 *Temporal averaging*

Since the last iterate is noisy due to stochastic approximation, better generalization performance is often achieved by averaging. Previously in Moulines and Bach [2011], Polyak-Ruppert averaging [Polyak and Juditsky, 1992, Ruppert, 1988] has been shown to improve the convergence of standard SGD,

where $\bar{\theta}_t = \frac{1}{t}\sum_{k=1}^{n} \theta_k$. Alternatively, an exponential moving average over the parameters can be used, giving higher weight to more recent parameter values. This can be trivially implemented by adding one line to the inner loop of algorithms 6 and 7: $\bar{\theta}_t \leftarrow \beta_2 \cdot \bar{\theta}_{t-1} + (1 - \beta_2)\theta_t$, with $\bar{\theta}_0 = 0$. Initialization bias can again be corrected by the estimator $\widehat{\theta}_t = \bar{\theta}_t / (1 - \beta_2^t)$.

7.7 CONCLUSION

We have introduced a simple and computationally efficient algorithm for gradient-based optimization of stochastic objective functions. Our method is aimed towards machine learning problems with large datasets and/or high-dimensional parameter spaces. The method combines the advantages of two recently popular optimization methods: the ability of AdaGrad to deal with sparse gradients, and the ability of RMSProp to deal with non-stationary objectives. The method is straightforward to implement and requires little memory. Overall, we found Adam to be robust and well-suited to a wide range of non-convex optimization problems in the field machine learning.

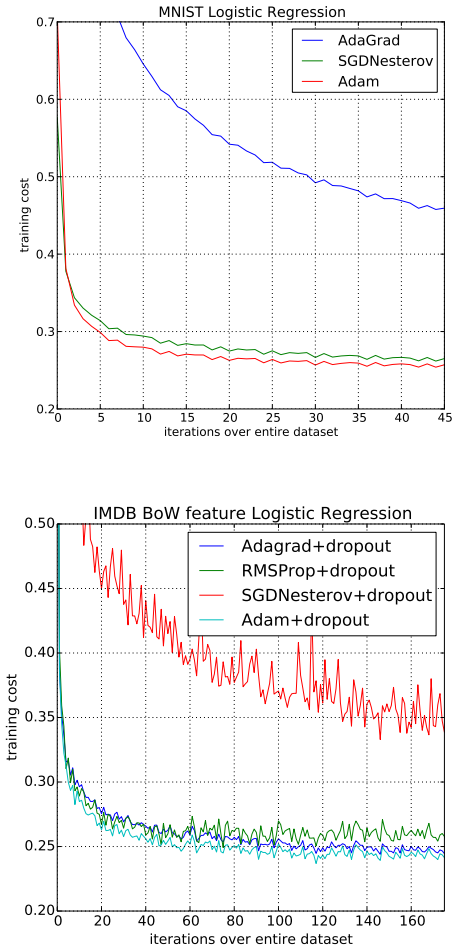


Figure 7.1: Logistic regression training negative log likelihood on MNIST images and IMDB movie reviews with 10,000 bag-of-words (BoW) feature vectors.

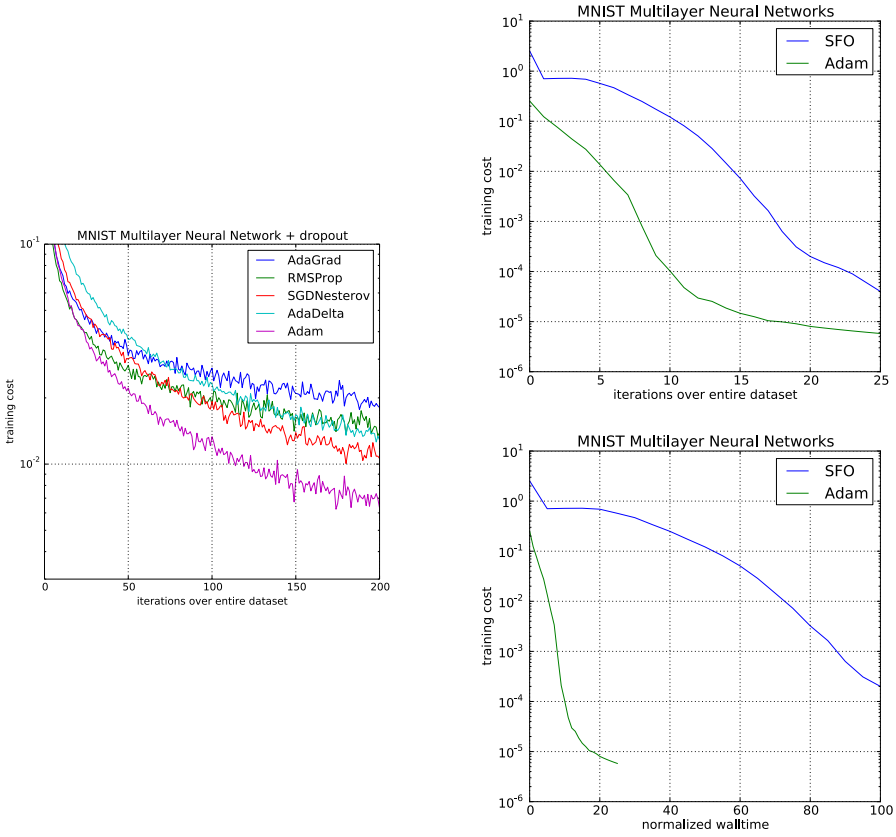


Figure 7.2: Training of multilayer neural networks on MNIST images. (a) Neural networks using dropout stochastic regularization. (b) Neural networks with deterministic cost function. We compare with the sum-of-functions (SFO) optimizer [Sohl-Dickstein et al., 2014]

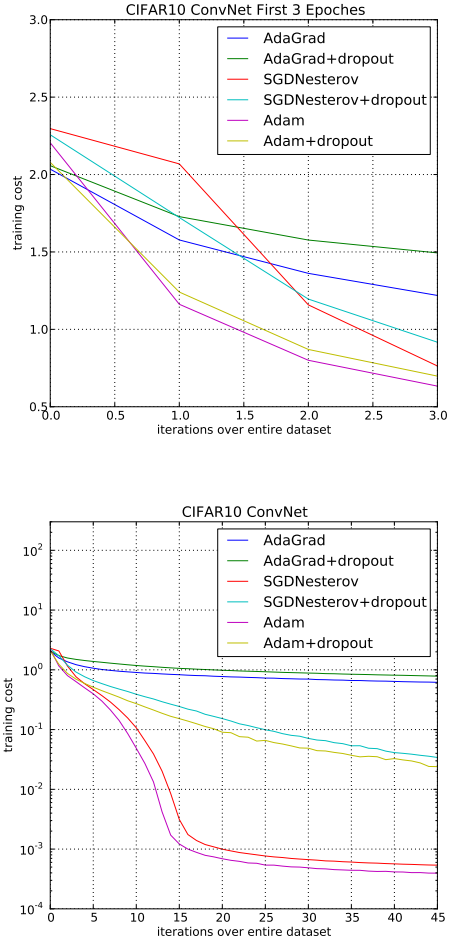


Figure 7.3: Convolutional neural networks training cost. (left) Training cost for the first three epochs. (right) Training cost over 45 epochs. CIFAR-10 with c64-c64-c128-1000 architecture.

8

CONCLUSION

Directed probabilistic models form an important aspect of modern artificial intelligence. Such models can be made incredibly flexible by parameterizing the conditional distributions with differentiable deep neural networks.

Optimization of such models towards the maximum likelihood objective is straightforward in the fully-observed case. Often we are interested in a more sophisticated (approximate) Bayesian posterior inference in such models. Two common cases are (1) maximum likelihood estimation in the partially observed case, such as deep latent-variable models (DLVMs), and (2) Bayesian posterior inference over the parameters. In case of *variational* inference, inference is cast as an optimization problem over newly introduced variational parameters, typically optimized towards the ELBO, a lower bound on the model evidence, or marginal likelihood of the data. Existing methods for such posterior inference were either relatively inefficient, or not applicable to models with neural networks as components.

Our main contribution is a set of efficient methods for efficient and scalable gradient-based variational posterior inference and approximate maximum likelihood learning. A common component in our proposed methods is a reparameterization of the latent variables or parameters as a differentiable function of independent noise and the variational parameters. This allows for straightforward computation of gradients using automatic differentiation software. For optimization of latent-variable models w.r.t. the ELBO, we proposed the variational autoencoder (VAE) framework, combining reparameterization gradients with inference networks, avoiding local parameters and enabling scalability to large datasets through a doubly stochastic optimization procedure. We showed that this framework allows for learning flexible DLVMs that are in principle able to model complicated distributions

from large datasets. This answers research question 2 in the affirmative. The VAE framework is now a commonly used tool for various applications of probabilistic modeling and artificial creativity, and basic implementations are available in most major deep learning software libraries.

We also show how this framework allowed for a large jump in semi-supervised classification performance compared to the previous state of the art at time of publication. This answers research question 2 in the affirmative. The result has now been surpassed, such as by GAN-based methods [Salimans et al., 2016]. Novel estimators for discrete latent variables [Maddison et al., 2016, Jang et al., 2016] make it possible to scale architectures similar to ours to a much larger number of classes, and are an interesting avenue for further exploration of semi-supervised learning with deep generative models.

For learning flexible inference models, we proposed inverse autoregressive flows (IAF), a type of normalizing flow that allows scaling to high-dimensional latent spaces. This answered research question 3 in the affirmative. The VAE framework is compatible with almost arbitrary differentiable neural networks. We showed that a VAE with an IAF posterior and ResNets, a novel type of neural network, can learn models of natural images that are close to state-of-the-art in terms of log-likelihood, while allowing for orders of magnitude faster sampling. An interesting direction for further exploration is comparison with transformations with computationally cheap inverses, such as NICE [Dinh et al., 2014] and Real NVP [Dinh et al., 2016]. Application of such transformations in the VAE framework can potentially lead to relatively simple VAEs with a combination of powerful posteriors, priors and decoders. Such architectures can potentially rival or surpass purely autoregressive architectures [van den Oord et al., 2016a], while allowing much faster synthesis.

The proposed VAE framework remains the only framework in the literature that allows for both discrete and continuous observed variables, allows for efficient amortized latent-variable inference and fast synthesis, and which can produce close to state-of-the-art performance in terms of the log-likelihood of data.

We performed work to further improve the efficiency of gradient-based variational inference of factorized Gaussian posteriors over neural network parameters. We developed a local reparameterization technique that parameterizes the uncertainty over the global weights as uncertainty over local activations, leading to a gradient estimator whose precision scales linearly with

the minibatch size. This answered research question 4 in the affirmative. Our experiments did not show enormous improvements upon early stopping or binary dropout in terms of accuracy or log-likelihood. Follow-up work such as [Molchanov et al., 2017] and [Louizos et al., 2017] shows how similar parameterizations can be used for model sparsification and compression of models. We belief this is an interesting avenue for future research.

Finally, we proposed Adam, a first-order gradient-based optimization algorithm based on adaptive moments. We showed that the method is relatively easy to tune, and that hyper-parameter choices exist that produce acceptable results across a large range of problems. Our results answered research question 5 in the affirmative. The optimization method has been implemented in all major deep learning software libraries and used in thousands of publications. We suspect that further improvement is feasible through incorporation of curvature information.

9

APPENDIX

9.1 NOTATION AND DEFINITIONS

9.1.1 *Notation*

Example(s)	Description
$\mathbf{x}, \mathbf{y}\, \mathbf{z}$	With characters in bold we typically denote random *vectors*. We also use this notation for collections of random variables variables.
x, y, z	With characters in italic we typically denote random *scalars*, i.e. single real-valued numbers.
$\mathbf{X}, \mathbf{Y}, \mathbf{Z}$	With bold and capitalized letters we typically denote random *matrices*.
$\text{diag}(\mathbf{x})$	Diagonal matrix, with the values of vector $\mathbf{x}$ on the diagonal.
$\mathbf{x} \odot \mathbf{y}$	Element-wise multiplication of two vectors. The resulting vector is $(x_1 y_1, ..., x_K y_K)^T$.
θ	Parameters of a (generative) model are typically denoted with the Greek lowercase letter θ (theta).
$\boldsymbol{\phi}$	Variational parameters are typically denoted with the bold Greek letter $\boldsymbol{\phi}$ (phi).

$p(\mathbf{x}), p(\mathbf{z})$	Probability density functions (PDFs) and probability mass functions (PMFs), also simply called *distributions*, are denoted by $p(.)$, $q(.)$ or $r(.)$.
$p(\mathbf{x}, \mathbf{y}, \mathbf{z})$	Joint distributions are denoted by $p(.,.)$
$p(\mathbf{x}\|\mathbf{z})$	Conditional distributions are denoted by $p(.\|.)$
$p(.;\theta), p_\theta(\mathbf{x})$	The parameters of a distribution are denoted with $p(.;\theta)$ or equivalently with subscript $p_\theta(.)$.
$p(\mathbf{x} = \mathbf{a}), p(\mathbf{x} \leq \mathbf{a})$	We may use an (in-)equality sign within a probability distribution to distinguish between function arguments and value at which to evaluate. So $p(\mathbf{x} = \mathbf{a})$ denotes a PDF or PMF over variable $\mathbf{a}$ evaluated at the value of variable $\mathbf{a}$. Likewise, $p(\mathbf{x} \leq \mathbf{a})$ denotes a CDF evaluated at the value of $\mathbf{a}$.
$p(.), q(.)$	We use different letters to refer to different probabilistic models, such as $p(.)$ or $q(.)$. Conversely, we use the *same* letter across different marginals/conditionals to indicate they relate to the same probabilistic model.

9.1.2 *Definitions*

Term	**Description**
Probability density function (PDF)	A function that assigns a probability *density* to each possible value of given *continuous* random variables.
Cumulative distribution function (CDF)	A function that assigns a cumulative probability density to each possible value of given univariate *continuous* random variables.
Probability mass function (PMF)	A function that assigns a probability *mass* to given *discrete* random variable.

9.1.3 *Distributions*

We overload the notation of distributions (e.g. $p(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$) with two meanings: (1) a distribution from which we can sample, and (2) the probability density function (PDF) of that distribution.

Term	**Description**
Categorical$(x; \mathbf{p})$	Categorical distribution, with parameter $\mathbf{p}$ such that $\sum_i p_i = 1$.
Bernoulli$(\mathbf{x}; \mathbf{p})$	Multivariate distribution of independent Bernoulli. Bernoulli$(\mathbf{x}; \mathbf{p}) = \prod_i$ Bernoulli$(x_i; p_i)$ with $\forall i : 0 \leq p_i \leq 1$.
Normal$(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$	Multivariate Normal distribution with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$.

9.1.3.1 *Chain rule of probability*

$$p(\mathbf{a}, \mathbf{b}) = p(\mathbf{a})p(\mathbf{b}|\mathbf{a}) \tag{9.1}$$

9.1.3.2 *Bayes' Rule*

$$p(\mathbf{a}|\mathbf{b}) = p(\mathbf{a}|\mathbf{b})p(\mathbf{a})/p(\mathbf{b}) \tag{9.2}$$

9.1.4 *Bayesian Inference*

Let $p(\theta)$ be a chosen marginal distribution over its parameters θ, called a *prior distribution*. Let $\mathcal{D}$ be observed data, $p(\mathcal{D}|\theta) \equiv p_{\boldsymbol{\theta}}(\mathcal{D})$ be the probability assigned to the data under the model with parameters θ. Recall the chain rule in probability:

$$p(\theta, \mathcal{D}) = p(\theta|\mathcal{D})p(\mathcal{D}) = p(\theta)p(\mathcal{D}|\theta)$$

Simply re-arranging terms above, the posterior distribution over the parameters θ, taking into account the data $\mathcal{D}$, is:

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} \propto p(\mathcal{D}|\theta)p(\theta) \tag{9.3}$$

where the proportionality ($\propto$) holds since $p(\mathcal{D})$ is a constant that is not dependent on parameters θ. The formula above is known as *Bayes' rule*, a fundamental formula in machine learning and statistics, and is of special importance to this work.

A principal application of Bayes' rule is that it allows us to make predictions about future data $\mathbf{x}'$, that are optimal as long as the prior $p(\theta)$ and model class $p_{\boldsymbol{\theta}}(\mathbf{x})$ are correct:

$$p(\mathbf{x} = \mathbf{x}'|\mathcal{D}) = \int p_{\boldsymbol{\theta}}(\mathbf{x} = \mathbf{x}')p(\theta|\mathcal{D})d\theta$$

9.2 ALTERNATIVE METHODS FOR LEARNING IN DLVMS

9.2.1 *Maximum A Posteriori*

From a Bayesian perspective, we can improve upon the maximum likelihood objective through *maximum a posteriori* (MAP) estimation, which maximizes the log-posterior w.r.t. θ. With i.i.d. data $\mathcal{D}$, this is:

$$\begin{aligned} L^{MAP}(\theta) &= \log p(\theta|\mathcal{D}) && (9.4)\\ &= \log p(\theta) + L^{ML}(\theta) + \text{constant} && (9.5) \end{aligned}$$

The prior $p(\theta)$ in equation (9.5) has diminishing effect for increasingly large N. For this reason, in case of optimization with large datasets, we often choose to simply use the maximum likelihood criterion by omitting the prior from the objective, which is numerically equivalent to setting $p(\theta) =$ constant.

9.2.2 *Variational EM with local variational parameters*

Expectation Maximization (EM) is a general strategy for learning parameters in partially observed models [Dempster et al., 1977]. See section 9.2.3 for a discussion of EM using MCMC. The method can be explained as coordinate ascent on the ELBO [Neal and Hinton, 1998]. In case of of i.i.d. data, traditional variational EM methods estimate **local variational parameters** $\boldsymbol{\phi}^{(i)}$, i.e. a separate set of variational parameters per datapoint i in the dataset. In contrast, VAEs employ a strategy with **global variational parameters**.

EM starts out with some (random) initial choice of $\boldsymbol{\theta}$ and $\boldsymbol{\phi}^{(1:N)}$. It then iteratively applies updates:

$$\forall i = 1, ..., N: \quad \boldsymbol{\phi}^{(i)} \leftarrow \underset{\boldsymbol{\phi}}{\operatorname{argmax}} \mathcal{L}(\mathbf{x}^{(i)}; \boldsymbol{\theta}, \boldsymbol{\phi}) \qquad \text{(E-step)} \tag{9.6}$$

$$\boldsymbol{\theta} \leftarrow \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{i=1}^{N} \mathcal{L}(\mathbf{x}^{(i)}; \boldsymbol{\theta}, \boldsymbol{\phi}) \qquad \text{(M-step)} \tag{9.7}$$

until convergence. Why does this work? Note that at the E-step:

$$\underset{\boldsymbol{\phi}}{\operatorname{argmax}} \mathcal{L}(\mathbf{x}; \boldsymbol{\theta}, \boldsymbol{\phi}) \tag{9.8}$$

$$= \underset{\boldsymbol{\phi}}{\operatorname{argmax}} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x}) - D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}))\right] \tag{9.9}$$

$$= \underset{\boldsymbol{\phi}}{\operatorname{argmin}} D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})) \tag{9.10}$$

so the *E*-step, sensibly, minimizes the KL divergence of $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ from the true posterior.

Secondly, note that if $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ equals $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$, the ELBO equals the marginal likelihood, but that for any choice of $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$, the *M*-step optimizes a bound on the marginal likelihood. The tightness of this bound is defined by $D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}))$.

9.2.3 *MCMC-EM*

Another Bayesian approach towards optimizing the likelihood $p_{\boldsymbol{\theta}}(\mathbf{x})$ with DLVMs is Expectation Maximization (EM) with Markov Chain Monte Carlo (MCMC). In case of MCMC, the posterior is approximated by a mixture of a set of approximately i.i.d. samples from the posterior, acquired by running a Markov chain. Note that posterior gradients in DLVMs are relatively affordable to compute by differentiating the log-joint distribution w.r.t. $\mathbf{z}$:

$$\nabla_{\mathbf{z}} \log p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) = \nabla_{\mathbf{z}} \log[p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) / p_{\boldsymbol{\theta}}(\mathbf{x})] \tag{9.11}$$

$$= \nabla_{\mathbf{z}} [\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) - \log p_{\boldsymbol{\theta}}(\mathbf{x})] \tag{9.12}$$

$$= \nabla_{\mathbf{z}} \log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) - \nabla_{\mathbf{z}} \log p_{\boldsymbol{\theta}}(\mathbf{x}) \tag{9.13}$$

$$= \nabla_{\mathbf{z}} \log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) \tag{9.14}$$

One version of MCMC which uses such posterior for relatively fast convergence, is Hamiltonian MCMC [Neal, 2011]. A disadvantage of this approach is the requirement for running an independent MCMC chain per datapoint.

9.3 STOCHASTIC GRADIENT DESCENT

We work with directed models where the objective per datapoint is scalar, and due to the differentiability of neural networks that compose them, the objective is differentiable w.r.t. its parameters θ. Due to the remarkable efficiency of reverse-mode automatic differentiation (also known as the back-propagation algorithm [Rumelhart et al., 1988]), the value and gradient (i.e. the vector of partial derivatives) of differentiable scalar objectives can be computed with equal time complexity. In SGD, we iteratively update parameters θ:

$$\theta_{t+1} \leftarrow \theta_t + \alpha_t \cdot \nabla_\theta \tilde{L}(\theta, \xi) \tag{9.15}$$

where α_t is a learning rate or preconditioner, and $\tilde{L}(\theta, \xi)$ is an unbiased estimate of the objective $L(\theta)$, i.e. $\mathbb{E}_{\xi \sim p(\xi)}\left[\tilde{L}(\theta, \xi)\right] = L(\theta)$. The random variable ξ could e.g. be a datapoint index, uniformly sampled from $\{1, ..., N\}$, but can also include different types of noise such posterior sampling noise in VAEs. In experiments, we use the Adam and Adamax optimization methods for choosing α_t; these methods are invariant to constant rescaling of the objective, and invariant to constant re-scalings of the individual gradients. As a result, $\tilde{L}(\theta, \xi)$ only needs to be unbiased up to proportionality. We iteratively apply eq. (9.15) until a stopping criterion is met. A simple but effective criterion is to stop optimization as soon as the probability of a holdout set of data starts decreasing; this criterion is called *early stopping*.

BIBLIOGRAPHY

Ryan Prescott Adams and Zoubin Ghahramani. Archipelago: nonparametric Bayesian semi-supervised learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2009.

Sungjin Ahn, Anoop Korattikara, and Max Welling. Bayesian posterior sampling via stochastic gradient Fisher scoring. *arXiv preprint arXiv:1206.6380*, 2012.

Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

Jimmy Ba and Brendan Frey. Adaptive dropout for training deep neural networks. In *Advances in Neural Information Processing Systems*, pages 3084–3092, 2013.

Justin Bayer, Maximilian Karol, Daniela Korhammer, and Patrick Van der Smagt. Fast adaptive weight noise. *arXiv preprint arXiv:1507.05331*, 2015.

Mikhail Belkin and Partha Adviser-Niyogi. Problems of learning on manifolds. 2003.

Yoshua Bengio. Estimating or propagating gradients through stochastic neurons. *arXiv preprint arXiv:1305.2982*, 2013.

Yoshua Bengio and Éric Thibodeau-Laufer. Deep generative stochastic networks trainable by backprop. *arXiv preprint arXiv:1306.1091*, 2013.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. *Representation learning: A review and new perspectives*. IEEE, 2013.

James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, volume 4, 2010.

David M Blei, Michael I Jordan, and John W Paisley. Variational Bayesian inference with Stochastic Search. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 1367–1374, 2012.

Avrim Blum, John Lafferty, Mugizi Robert Rwebangira, and Rajashekar Reddy. Semi-supervised learning using randomized mincuts. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2004.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

Hervé Bourlard and Yves Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 59(4-5):291–294, 1988.

Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, 2015.

Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by Exponential Linear Units (ELUs). *arXiv preprint arXiv:1511.07289*, 2015.

Quaid Cremer, Chris ad Morris and David Duvenaud. Re-interpreting importance weighted autoencoders. *ICLR 2017*, 2017.

Peter Dayan. Helmholtz machines and wake-sleep learning. *Handbook of Brain Theory and Neural Network. MIT Press, Cambridge, MA*, 44(0), 2000. URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.105.6986&rep=rep1&type=pdfhttp://cns-course.org/~dayan/papers/d2000a.pdf`.

Gustavo Deco and Wilfried Brauer. Higher order statistical decorrelation without information loss. *Advances in Neural Information Processing Systems*, pages 247–254, 1995.

Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38, 1977.

Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using Real NVP. *arXiv preprint arXiv:1605.08803*, 2016.

Simon Duane, Anthony D Kennedy, Brian J Pendleton, and Duncan Roweth. Hybrid Monte Carlo. *Physics letters B*, 195(2):216–222, 1987.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2010.

Rob Fergus, Yair Weiss, and Antonio Torralba. Semi-supervised learning in gigantic image collections. In *Advances in Neural Information Processing Systems (NIPS)*, 2009.

Michael C Fu. Gradient estimation. *Handbooks in operations research and management science*, 13:575–616, 2006.

Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. *arXiv preprint arXiv:1506.02142*, 2015.

Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. *arXiv preprint arXiv:1502.03509*, 2015.

Samuel Gershman and Noah Goodman. Amortized inference in probabilistic reasoning. In *CogSci*, 2014.

Paul Glasserman. *Monte Carlo methods in financial engineering*, volume 53. Springer Science & Business Media, 2013.

Peter W Glynn. Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM*, 33(10):75–84, 1990.

Rafael Gómez-Bombarelli, David Duvenaud, José Miguel Hernández-Lobato, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *arXiv preprint arXiv:1610.02415*, 2016.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016. URL `http://www.deeplearningbook.org`.

Alex Graves. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, pages 2348–2356, 2011.

Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

Karol Gregor, Andriy Mnih, and Daan Wierstra. Deep AutoRegressive Networks. *arXiv preprint arXiv:1310.8499*, 2013.

Karol Gregor, Frederic Besse, Danilo Jimenez Rezende, Ivo Danihelka, and Daan Wierstra. Towards conceptual compression. In *Advances In Neural Information Processing Systems*, pages 3549–3557, 2016.

Ishaan Gulrajani, Kundan Kumar, Faruk Ahmed, Adrien Ali Taiga, Francesco Visin, David Vazquez, and Aaron Courville. PixelVAE: A latent variable model for natural images. *arXiv preprint arXiv:1611.05013*, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015b.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. *arXiv preprint arXiv:1603.05027*, 2016.

José Miguel Hernández-Lobato and Ryan P Adams. Probabilistic backpropagation for scalable learning of Bayesian neural networks. *arXiv preprint arXiv:1502.05336*, 2015.

Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13. ACM, 1993.

Geoffrey E Hinton, Peter Dayan, Brendan J Frey, and Radford M Neal. The "wake-sleep" algorithm for unsupervised neural networks. *SCIENCE*, pages 1158–1158, 1995.

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural computation*, 9(8):1735–1780, 1997.

Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *The Journal of Machine Learning Research*, 14 (1):1303–1347, 2013.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with Gumbel-Softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Thorsten Joachims. Transductive inference for text classification using support vector machines. In *Proceeding of the International Conference on Machine Learning (ICML)*, volume 99, pages 200–209, 1999.

Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 2342–2350, 2015.

Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. How to train deep variational autoencoders and probabilistic ladder networks. *arXiv preprint arXiv:1602.02282*, 2016.

Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. Fast inference in sparse coding algorithms with applications to object recognition. Technical Report CBLL-TR-2008-12-01, Computational and Biological Learning Lab, Courant Institute, NYU, 2008.

Charles Kemp, Thomas L Griffiths, Sean Stromsten, and Joshua B Tenenbaum. Semi-supervised learning with trees. In *Advances in Neural Information Processing Systems (NIPS)*, 2003.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *Proceedings of the International Conference on Learning Representations 2015*, 2015.

Diederik P Kingma. Fast gradient-based inference with continuous latent variable models in auxiliary form. *arXiv preprint arXiv:1306.0733*, 2013.

Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. *Proceedings of the 2nd International Conference on Learning Representations*, 2013.

Diederik P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, pages 3581–3589, 2014.

Diederik P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pages 2575–2583, 2015.

Diederik P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *Advances in Neural Information Processing Systems*, pages 4743–4751, 2016.

Jack PC Kleijnen and Reuven Y Rubinstein. Optimization and sensitivity analysis of computer simulation models by the score function method. *European Journal of Operational Research*, 88(3):413–427, 1996.

Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M Blei. Automatic differentiation variational inference. *arXiv preprint arXiv:1603.00788*, 2016.

Nicolas Le Roux and Andrew W Fitzgibbon. A fast natural newton method. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 623–630, 2010.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

Peng Li, Yiming Ying, and Colin Campbell. A variational approach to semi-supervised clustering. In *Proceedings of the European Symposium on Artificial Neural Networks (ESANN)*, pages 11 – 16, 2009.

Percy Liang. *Semi-supervised learning for natural language*. PhD thesis, Massachusetts Institute of Technology, 2005.

Ralph Linsker. *An application of the principle of maximum information preservation to linear systems*. Morgan Kaufmann Publishers Inc., 1989.

Yuzong Liu and Katrin Kirchhoff. Graph-based semi-supervised learning for phone and segment classification. In *Proceedings of Interspeech*, 2013.

Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. *arXiv preprint arXiv:1705.08665*, 2017.

Lars Maaløe, Casper Kaae Sønderby, Søren Kaae Sønderby, and Ole Winther. Auxiliary deep generative models. *arXiv preprint arXiv:1602.05473*, 2016.

Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 142–150. Association for Computational Linguistics, 2011.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

Shin-ichi Maeda. A Bayesian encourages dropout. *arXiv preprint arXiv:1412.7003*, 2014.

Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. In *The 31st International Conference on Machine Learning (ICML)*, 2014.

Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. *arXiv preprint arXiv:1701.05369*, 2017.

Eric Moulines and Francis R Bach. Non-asymptotic analysis of stochastic approximation algorithms for machine learning. In *Advances in Neural Information Processing Systems*, pages 451–459, 2011.

R Neal. MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, pages 113–162, 2011.

Radford M Neal. *Bayesian learning for neural networks*. PhD thesis, University of Toronto, 1995.

Radford M Neal and Geoffrey E Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pages 355–368. Springer, 1998.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, 2011.

John Paisley, David Blei, and Michael Jordan. Variational Bayesian inference with stochastic search. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 1367–1374, 2012.

Razvan Pascanu and Yoshua Bengio. Revisiting natural gradient for deep networks. *arXiv preprint arXiv:1301.3584*, 2013.

Nikolaos Pitelis, Chris Russell, and Lourdes Agapito. Semi-supervised learning using an unsupervised atlas. In *Proceddings of the European Conference on Machine Learning (ECML)*, volume LNCS 8725, pages 565 – 580, 2014.

Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.

Yunchen Pu, Zhe Gan, Ricardo Henao, Xin Yuan, Chunyuan Li, Andrew Stevens, and Lawrence Carin. Variational autoencoder for deep learning of images, labels and captions. In *Advances in Neural Information Processing Systems*, pages 2352–2360, 2016.

Rajesh Ranganath, Sean Gerrish, and David Blei. Black box variational inference. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, pages 814–822, 2014.

Rajesh Ranganath, Dustin Tran, and David M Blei. Hierarchical variational models. *arXiv preprint arXiv:1511.02386*, 2015.

Marc'Aurelio Ranzato and Martin Szummer. Semi-supervised learning of compact document representations with deep networks. In *Proceedings of the 25th International Conference on Machine Learning (ICML)*, pages 792–799, 2008.

Siamak Ravanbakhsh, Francois Lanusse, Rachel Mandelbaum, Jeff Schneider, and Barnabas Poczos. Enabling dark energy science with deep generative models of galaxy images. *arXiv preprint arXiv:1609.05796*, 2016.

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *Proceedings of The 32nd International Conference on Machine Learning*, pages 1530–1538, 2015.

Danilo J Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1278–1286, 2014.

Danilo Jimenez Rezende, Shakir Mohamed, Ivo Danihelka, Karol Gregor, and Daan Wierstra. One-shot generalization in deep generative models. *arXiv preprint arXiv:1603.05106*, 2016.

Salah Rifai, Yann Dauphin, Pascal Vincent, Yoshua Bengio, and Xavier Muller. The manifold tangent classifier. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2294–2302, 2011.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

Chuck Rosenberg, Martial Hebert, and Henry Schneiderman. Semi-supervised self-training of object detection models. In *Proceedings of the Seventh IEEE Workshops on Application of Computer Vision (WACV/MOTION'05)*, 2005.

Sam Roweis. EM algorithms for PCA and SPCA. *Advances in neural information processing systems*, pages 626–632, 1998.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

David Ruppert. Efficient estimations from a slowly convergent robbins-monro process. Technical report, Cornell University Operations Research and Industrial Engineering, 1988.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

Ruslan Salakhutdinov and Hugo Larochelle. Efficient learning of deep boltzmann machines. In *International Conference on Artificial Intelligence and Statistics*, pages 693–700, 2010.

Tim Salimans. A structured variational auto-encoder for learning deep hierarchies of sparse features. *arXiv preprint arXiv:1602.08734*, 2016.

Tim Salimans and Diederik P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *arXiv preprint arXiv:1602.07868*, 2016.

Tim Salimans and David A Knowles. Fixed-form variational posterior approximation through stochastic linear regression. *Bayesian Analysis*, 8(4), 2013.

Tim Salimans, Diederik P Kingma, and Max Welling. Markov Chain Monte Carlo and variational inference: Bridging the gap. In *ICML*, volume 37, pages 1218–1226, 2015.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2234–2242, 2016.

Tom Schaul, Sixin Zhang, and Yann LeCun. No more pesky learning rates. *arXiv preprint arXiv:1206.1106*, 2012.

Mingguang Shi and Bing Zhang. Semi-supervised learning improves gene expression-based prediction of cancer recurrence. *Bioinformatics*, 27(21):3017–3023, 2011. doi: 10.1093/bioinformatics/btr502. URL `http://bioinformatics.oxfordjournals.org/content/early/2011/09/04/bioinformatics.btr502.abstract`.

Jascha Sohl-Dickstein, Ben Poole, and Surya Ganguli. Fast large-scale optimization by unifying stochastic gradient and quasi-newton methods. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 604–612, 2014.

Jascha Sohl-Dickstein, Eric A Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. *arXiv preprint arXiv:1503.03585*, 2015.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Andreas Stuhlmüller, Jacob Taylor, and Noah Goodman. Learning stochastic inverses. In *Advances in neural information processing systems*, pages 3048–3056, 2013.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

Yichuan Tang and Ruslan Salakhutdinov. Learning stochastic feedforward neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 530–538, 2013.

T. Tieleman and G. Hinton. COURSERA: Neural networks for machine learning. 2012.

Dustin Tran, Rajesh Ranganath, and David M Blei. The variational Gaussian process. *arXiv preprint arXiv:1511.06499*, 2015.

Aaron van den Oord and Benjamin Schrauwen. Factoring variations in natural images with deep gaussian mixture models. In *Advances in Neural Information Processing Systems*, pages 3518–3526, 2014.

Aaron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with PixelCNN decoders. In *Advances in Neural Information Processing Systems*, pages 4790–4798, 2016a.

Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016b.

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(Dec):3371–3408, 2010.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th*

International Conference on Machine Learning (ICML-13), pages 1058–1066, 2013.

Sida Wang and Christopher Manning. Fast dropout training. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 118–126, 2013.

Yang Wang, Gholamreza Haffari, Shaojun Wang, and Greg Mori. A rate distortion approach for semi-supervised conditional random fields. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2008–2016, 2009.

Max Welling and Yee W Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 681–688, 2011.

Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*, pages 639–655. Springer, 2012.

Tom White. Sampling generative networks: Notes on a few effective techniques. *arXiv preprint arXiv:1609.04468*, 2016.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

David Wingate and Theophane Weber. Automated variational inference in probabilistic programming. *arXiv preprint arXiv:1301.1299*, 2013.

Zhirong Yang, Tele Hao, Onur Dikmen, Xi Chen, and Erkki Oja. Clustering by nonnegative matrix factorization using graph random walk. In *Advances in Neural Information Processing Systems*, pages 1079–1087, 2012.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Matthew D Zeiler. ADADELTA: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Matthew D Zeiler, Dilip Krishnan, Graham W Taylor, and Rob Fergus. Deconvolutional networks. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2528–2535. IEEE, 2010.

Xiaojin Zhu. Semi-supervised learning literature survey. Technical report, Computer Science, University of Wisconsin-Madison, 2006.

Xiaojin Zhu, Zoubin Ghahramani, John Lafferty, et al. Semi-supervised learning using Gaussian fields and harmonic functions. In *Proceddings of the International Conference on Machine Learning (ICML)*, volume 3, pages 912–919, 2003.

LIST OF PUBLICATIONS

As required, we provide a list of publications whose content was used in this thesis, and provide the contributions of co-authors.

- Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014.
- Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.
- Diederik P. Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-Supervised Learning with Deep Generative Models. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.
- Diederik P. Kingma, Tim Salimans, and Max Welling. Variational Dropout and the Local Reparameterization Trick. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.
- Diederik P. Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever and Max Welling. Improving Variational Inference with Inverse Autoregressive Flow. In *Advances in Neural Information Processing Systems (NIPS)*, 2016

The majority of ideas, text, figures and experiments originated from the first author, with the exception of the following cases. In case of the Adam paper, both authors contributed equally to all material. In case of the *inverse autoregressive flow* paper, the first two authors contributed equally to the main ideas. Max Welling had an important advisory role over all performed research.

SAMENVATTING — SUMMARY IN DUTCH

Here follows a summary of this thesis in the Dutch language. Hieronder volgt een Nederlandstalige samenvatting van dit proefschrift, "*Variational Inference and Deep Learning: A New Synthesis*".

Mensen hebben een relatief bijzondere mentale eigenschap genaamd *intelligentie*. Intelligentie is lastig te definiëren, maar kan worden omschreven als onze natuurlijke vaardigheid om te *leren* uit ervaring en van andere mensen en derhalve nieuwe kennis en vaardigheden te verwerven. Onze (gezamenlijke) intelligentie is grotendeels verantwoordelijk voor de ontwikkeling van taal, cultuur, wetenschap en technologie. Ontwikkelingen zoals landbouw, industrie en medicijnen, hebben grote vooruitgang gebracht in de lengte en kwaliteit van het menselijk leven. Ontwikkelingen in computertechnologie en computer-wetenschap (informatica) hebben daar in grote mate aan bijgedragen. Echter, net als onze fysieke vaardigheid is onze cognitieve vaardigheid beperkt: op de evolutionaire tijdschaal zijn wij immers niet ver verwijderd van andere primaten, zoals de chimpansees. Hoeveel meer zouden wij als mensheid kunnen bereiken als wij toegang kunnen verkrijgen tot een hogere mate van intelligentie? Wat als wij onze intelligentie gebruiken om kennis te vergaren over intelligentie zelf, en dit gebruiken om technologie te ontwikkelen die ons kan helpen om onze eigen cognitieve beperkingen te overwinnen?

Deze laatste vraag staat centraal in het veld van *artificial intelligence* (AI). *Machine learning* is een tak van de AI met als doel *de ontwikkeling van machines (computers) die zelf kunnen leren uit ervaring (data), en zo zelf nieuwe kennis en vaardigheden kunnen verwerven*.

Het belangrijkste doel van dit werk is de verdere ontwikkeling van het raakvlak tussen twee belangrijke gebieden binnen de machine learning: *deep learning* en *variational inference*. Deep learning is een moderne benaming voor machine learning met *neurale netwerken*, een type model dat efficiënt kan leren uit grote hoeveelheden hoog-dimensionele data. *Variational inference* is een type van Bayesiaanse inferentie, waarmee op een principiële wijze de onzekerheid over parameters en niet-geobserveerde variabelen kan worden

geschat. Wij onderzoeken een nieuwe synthese van ideeën uit deep learning en variational inference.

Hoofdstuk 1 bevat een introductie van de stof. In hoofdstukken 2, 3, 4 en 5 onderzoeken wij methodes voor het leren van *diepe generatieve modellen* (deep generative models), een krachtig type neuraal netwerk. Diepe generatieve modellen hebben de mogelijkheid om te leren wat *het verborgen (nonlineaire) proces* is waarmee de data tot stand zijn gekomen, en hebben diverse toepassingen. Tot op heden bestonden er echter nog geen praktische methodes voor het leren van diepe generatieve modellen uit grote datasets. In hoofdstuk 2 stellen wij een oplossing voor: het raamwerk van *variational autoencoders* (VAEs), gebruik makend van een *reparameterization trick* voor efficiënte variational inference. De methode heeft als bijkomend voordeel dat er een bijbehorend zogeheten *inference model* wordt getraind, die kan worden gebruikt voor *representation learning*. In hoofdstuk 2 demonstreren wij de toepassing van VAEs op generatief modelleren en representation learning. In hoofdstuk 3 demonsteren wij de toepassing van VAEs op *semi-supervised learning*, oftewel het leren uit zowel gelabelde als ongelabelde data. In hoofdstuk 5 behandelen wij technieken voor het leren van diepe generatieve modellen. In hoofdstuk 4 introduceren en onderzoeken wij een nieuwe methode voor het trainen van flexibelere *inference models*, namenlijk *inverse autoregressive flows* (IAF). Door middel van experimenten laten wij zien dat het met IAF mogelijk is generatieve modellen te leren die zowel accuraat zijn (in termen van de log-likelihood), als efficient om uit te genereren. Variational inference op basis van de *reparameterization trick*, geintroduceerd in hoofdstuk 2, kan ook worden gebruikt voor *Bayesiaanse inferentie*: het berekenen, na de observatie van nieuwe data, van de correcte onzekerheid over de waarde van de model-parameters. In hoofdstuk 6 introduceren wij een verbetering van deze methode voor neurale netwerken met Gaussian posteriors over de parameters, genaamd de *local reparameterization trick*, en demonstreren de effectiviteit van de methode. Neurale netwerken, inclusief de modellen gebruikt in ons onderzoek, worden doorgaans getraind aan de hand van *stochastic gradient descent* (SGD) algoritmes. In hoofdstuk 7 introduceren wij *Adam*, een nieuwe methode voor SGD. In diverse experimenten wordt onze methode effectiever bevonden dan de alternatieven.

ACKNOWLEDGMENTS

I would like to acknowledge a number of people that supported, in various ways, the creation of this work.

First and foremost I'd like to express gratitude towards my advisor Prof. Max Welling for accepting me as his first Ph.D. student in the Netherlands. Max gave me the opportunity to work on my research in relative peace, and our frequent discussions and brainstorming sessions were of key importance. I could not have hoped for an advisor with a greater overlap in academic interests, a greater intelligence or a greater clarity of thought. I also greatly admire that Max manages to combine his qualities with a sense of humility. I would also like to thank Max for providing the opportunity to spend a month performing research at UC Irvine; I thoroughly enjoyed our discussions at the university campus while enjoying Southern California's perfect weather.

Before starting my PhD, I was given the opportunity by prof. Yann LeCun to spend two semesters at his lab at New York University. My time in NYC was formative, reinforcing my belief that this was the right field in which to pursue a doctoral degree.

The colleagues I interacted most with during my time in Amsterdam were Ted Meeds and Taco Cohen. I should thank Ted for weathering my frequent requests for feedback on various ideas. I enjoyed our discussions while walking through the all desolate and wind-filled lands of Science Park. Taco was Max's second Ph.D. candidate in Amsterdam, with whom I shared a number of travels: to many conferences, a summer school in Canada, Detroit and later traveling around the world via China and the US. It was always a pleasure to poke at Taco's mind with new ideas, and his insights on group-theoretic machine learning were always thought-provoking.

I would also like to thank Daan Wierstra, Shakir Mohamed and Danilo Rezende for hosting me at DeepMind in London during the summers of 2014 and 2015. It was a true privilege collaborating on the paper on semi-supervised learning, and it was a true pleasure to brainstorm about deep generative models and their role in the future of AI. I also had the pleasure to collaborate with Jimmy Ba after we left London, resulting in our optimization paper.

During my Ph.D. I joined OpenAI, a new AI research lab in San Francisco. I would like to credit the organization for providing me with the time to finish this work.

A large role in this work was played by my collaborator and colleague Tim Salimans. By sheer coincidence, we both wrote papers proposing nearly the same idea, at nearly the same time while being at nearly the same location on this globe. This, in addition to a general shared research interest, subsequently led us to extensively collaborate, leading to six joint research papers. Much gratitude goes to Tim for his role in this work.

I'd like to thank Charlotte for her continuing support and for willing to move to San Francisco with me. In addition, I'd like to recognize my parents for providing a great example and for providing the best upbringing one could wish for. Lastly, I'd like to give credit to all the unnamed colleagues, friends and family who often endured my absence, and at other times provided me with the much-needed moral support and distraction.

Sincerely,

Durk Kingma

San Francisco
USA