Foundations and Trends® in Machine Learning

An Introduction to Variational Autoencoders

Suggested Citation: Diederik P. Kingma and Max Welling (2019), "An Introduction to Variational Autoencoders", Foundations and Trends® in Machine Learning: Vol. xx, No. xx, pp 1–18. DOI: 10.1561/XXXXXXXXX.

Diederik P. Kingma
Google
durk@google.com

Max Welling
Universiteit van Amsterdam, Qualcomm
mwelling@qti.qualcomm.com

now
the essence of knowledge
Boston — Delft

Contents

An Introduction to Variational Autoencoders

Diederik P. Kingma[1] and Max Welling[2,3]

[1] *Google; durk@google.com*
[2] *Universiteit van Amsterdam*
[3] *Qualcomm; mwelling@qti.qualcomm.com*

ABSTRACT

Variational autoencoders provide a principled framework for learning deep latent-variable models and corresponding inference models. In this work, we provide an introduction to variational autoencoders and some important extensions.

Diederik P. Kingma and Max Welling (2019), "An Introduction to Variational Autoencoders", Foundations and Trends® in Machine Learning: Vol. xx, No. xx, pp 1–18. DOI: 10.1561/XXXXXXXXX.

1

Introduction

1.1 Motivation

One major division in machine learning is generative versus discriminative modeling. While in discriminative modeling one aims to learn a predictor given the observations, in generative modeling one aims to solve the more general problem of learning a joint distribution over all the variables. A generative model simulates how the data is generated in the real world. "Modeling" is understood in almost every science as unveiling this generating process by hypothesizing theories and testing these theories through observations. For instance, when meteorologists model the weather they use highly complex partial differential equations to express the underlying physics of the weather. Or when an astronomer models the formation of galaxies s/he encodes in his/her equations of motion the physical laws under which stellar bodies interact. The same is true for biologists, chemists, economists and so on. Modeling in the sciences is in fact almost always generative modeling.

There are many reasons why generative modeling is attractive. First, we can express physical laws and constraints into the generative process while details that we don't know or care about, i.e. nuisance variables, are treated as noise. The resulting models are usually highly intuitive

and interpretable and by testing them against observations we can confirm or reject our theories about how the world works.

Another reason for trying to understand the generative process of data is that it naturally expresses causal relations of the world. Causal relations have the great advantage that they generalize much better to new situations than mere correlations. For instance, once we understand the generative process of an earthquake, we can use that knowledge both in California and in Chile.

To turn a generative model into a discriminator, we need to use Bayes rule. For instance, we have a generative model for an earthquake of type A and another for type B, then seeing which of the two describes the data best we can compute a probability for whether earthquake A or B happened. Applying Bayes rule is however often computationally expensive.

In discriminative methods we directly learn a map in the same direction as we intend to make future predictions in. This is in the opposite direction than the generative model. For instance, one can argue that an image is generated in the world by first identifying the object, then generating the object in 3D and then projecting it onto an pixel grid. A discriminative model takes these pixel values directly as input and maps them to the labels. While generative models can learn efficiently from data, they also tend to make stronger assumptions on the data than their purely discriminative counterparts, often leading to higher asymptotic bias (Banerjee, 2007) when the model is wrong. For this reason, if the model is wrong (and it almost always is to some degree!), if one is solely interested in learning to discriminate, and one is in a regime with a sufficiently large amount of data, then purely discriminative models typically will lead to fewer errors in discriminative tasks. Nevertheless, depending on how much data is around, it may pay off to study the data generating process as a way to guide the training of the discriminator, such as a classifier. For instance, one may have few labeled examples and many more unlabeled examples. In this semi-supervised learning setting, one can use the generative model of the data to improve classification (Kingma *et al.*, 2014; Sønderby *et al.*, 2016a).

Generative modeling can be useful more generally. One can think of it as an auxiliary task. For instance, predicting the immediate future

may help us build useful abstractions of the world that can be used for multiple prediction tasks downstream. This quest for disentangled, semantically meaningful, statistically independent and causal factors of variation in data is generally known as unsupervised representation learning, and the variational autoencoder (VAE) has been extensively employed for that purpose. Alternatively, one may view this as an implicit form of regularization: by forcing the representations to be meaningful for data generation, we bias the inverse of that process, which maps from input to representation, into a certain mould. The auxiliary task of predicting the world is used to better understand the world at an abstract level and thus to better make downstream predictions.

The VAE can be viewed as two coupled, but independently parameterized models: the encoder or recognition model, and the decoder or generative model. These two models support each other. The recognition model delivers to the generative model an approximation to its posterior over latent random variables, which it needs to update its parameters inside an iteration of "expectation maximization" learning. Reversely, the generative model is a scaffolding of sorts for the recognition model to learn meaningful representations of the data, including possibly class-labels. The recognition model is the approximate inverse of the generative model according to Bayes rule.

One advantage of the VAE framework, relative to ordinary Variational Inference (VI), is that the recognition model (also called inference model) is now a (stochastic) function of the input variables. This in contrast to VI where each data-case has a separate variational distribution, which is inefficient for large data-sets. The recognition model uses one set of parameters to model the relation between input and latent variables and as such is called "amortized inference". This recognition model can be arbitrary complex but is still reasonably fast because by construction it can be done using a single feedforward pass from input to latent variables. However the price we pay is that this sampling induces sampling noise in the gradients required for learning. Perhaps the greatest contribution of the VAE framework is the realization that we can counteract this variance by using what is now known as the "reparameterization trick", a simple procedure to reorganize our gradient computation that reduces variance in the gradients.

The VAE is inspired by the Helmholtz Machine (Dayan *et al.*, 1995) which was perhaps the first model that employed a recognition model. However, its wake-sleep algorithm was inefficient and didn't optimize a single objective. The VAE learning rules instead follow from a single approximation to the maximum likelihood objective.

VAEs marry graphical models and deep learning. The generative model is a Bayesian network of the form $p(\mathbf{x}|\mathbf{z})p(\mathbf{z})$, or, if there are multiple stochastic latent layers, a hierarchy such as $p(\mathbf{x}|\mathbf{z}_L)p(\mathbf{z}_L|\mathbf{z}_{L-1})$ $...p(\mathbf{z}_1|\mathbf{z}_0)$. Similarly, the recognition model is also a conditional Bayesian network of the form $q(\mathbf{z}|\mathbf{x})$ or as a hierarchy, such as $q(\mathbf{z}_0|\mathbf{z}_1)...q(\mathbf{z}_L|X)$. But inside each conditional may hide a complex (deep) neural network, e.g. $\mathbf{z}|\mathbf{x} \sim f(\mathbf{x}, \boldsymbol{\epsilon})$, with f a neural network mapping and $\boldsymbol{\epsilon}$ a noise random variable. Its learning algorithm is a mix of classical (amortized, variational) expectation maximization but through the reparameterization trick ends up backpropagating through the many layers of the deep neural networks embedded inside of it.

Since its inception, the VAE framework has been extended in many directions, e.g. to dynamical models (Johnson *et al.*, 2016), models with attention (Gregor *et al.*, 2015), models with multiple levels of stochastic latent variables (Kingma *et al.*, 2016), and many more. It has proven itself as a fertile framework to build new models in. More recently, another generative modeling paradigm has gained significant attention: the generative adversarial network (GAN) (Goodfellow *et al.*, 2014). VAEs and GANs seem to have complementary properties: while GANs can generate images of high subjective perceptual quality, they tend to lack full support over the data (Grover *et al.*, 2018), as opposed to likelihood-based generative models. VAEs, like other likelihood-based models, generate more dispersed samples, but are better density models in terms of the likelihood criterion. As such many hybrid models have been proposed to try to represent the best of both worlds (Dumoulin *et al.*, 2017; Grover *et al.*, 2018; Rosca *et al.*, 2018).

As a community we seem to have embraced the fact that generative models and unsupervised learning play an important role in building intelligent machines. We hope that the VAE provides a useful piece of that puzzle.

1.2 Aim

The framework of *variational autoencoders* (VAEs) (Kingma and Welling, 2014; Rezende *et al.*, 2014) provides a principled method for jointly learning *deep latent-variable models* and corresponding inference models using stochastic gradient descent. The framework has a wide array of applications from generative modeling, semi-supervised learning to representation learning.

This work is meant as an expanded version of our earlier work (Kingma and Welling, 2014), allowing us to explain the topic in finer detail and to discuss a selection of important follow-up work. This is *not* aimed to be a comprehensive review of all related work. We assume that the reader has basic knowledge of algebra, calculus and probability theory.

In this chapter we discuss background material: probabilistic models, directed graphical models, the marriage of directed graphical models with neural networks, learning in fully observed models and deep latent-variable models (DLVMs). In chapter 2 we explain the basics of VAEs. In chapter 3 we explain advanced inference techniques, followed by an explanation of advanced generative models in chapter 4. Please refer to section A.1 for more information on mathematical notation.

1.3 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 understanding such phenomena, for prediction of unknowns in the future, and for various forms of assisted or automated decision making. As such, probabilistic models formalize the notion of knowledge and skill, and are central constructs in the field of machine learning and AI.

As probabilistic models contain unknowns and the data rarely paints a complete picture of the unknowns, we typically need to assume some level of uncertainty over aspects of the model. The degree and nature of this uncertainty is specified in terms of (conditional) probability dis-

tributions. Models may consist of both continuous-valued variables and discrete-valued variables. The, in some sense, most complete forms of probabilistic models specify all correlations and higher-order dependencies between the variables in the model, in the form of a joint probability distribution over those variables.

Let's use $\mathbf{x}$ as the vector representing the set of all observed variables whose joint distribution we would like to model. 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 A.1 for more on notation.

We assume the observed variable $\mathbf{x}$ is a random sample from an *unknown underlying process*, whose true (probability) 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.

1.3.1 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 the value of an observed 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 time series prediction, such as text or video prediction.

To avoid notational clutter we will often assume unconditional modeling, but one should always keep in mind that the methods introduced in this work are, in almost all cases, 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.

1.4 Parameterizing Conditional Distributions with Neural Networks

Differentiable feed-forward neural networks, from here just called *neural networks*, are a particularly flexible and computationally scalable type of function approximator. Learning of models based on neural networks with multiple 'hidden' layers of artificial neurons is often referred to as *deep learning* (Goodfellow *et al.*, 2016; LeCun *et al.*, 2015). A particularly interesting application is probabilistic models, i.e. the use of neural networks for probability density functions (PDFs) or probability mass functions (PMFs) in probabilistic models. Probabilistic models based on neural networks are computationally scalable since they allow for stochastic gradient-based optimization which, as we will explain,

allows scaling to large models and large datasets. We will denote a deep neural network as a vector function: NeuralNet(.).

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 case of neural-network based image classification LeCun *et al.*, 1998, for example, 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 NeuralNet(.) is typically a softmax() function such that $\sum_i p_i = 1$.

1.5 Directed Graphical Models and Neural Networks

We work with *directed* probabilistic models, also called directed *probabilistic graphical models* (PGMs), or *Bayesian 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, we condition on the parents. For root nodes, the set of parents is the empty set, such that the distribution is unconditional.

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 (Koller and Friedman, 2009). As we explained above, a more flexible way to parameterize such conditional distributions is with neural networks. In this case, neural networks take as input the parents of a variable in a directed

graph, and produce the distributional parameters $\boldsymbol{\eta}$ over that variable:

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

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

We will now discuss how to learn the parameters of such models, if all the variables are observed in the data.

1.6 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.

1.6.1 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 $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$ 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. The log-probability assigned to the data by the model is therefore given by:

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

1.6.2 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 objective is to maximize the log-probability given by equation (1.10).

Using calculus' chain rule and automatic differentiation tools, we can efficiently compute gradients of this objective, i.e. the first derivatives of the objective w.r.t. its parameters $\boldsymbol{\theta}$. We can use such gradients 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 for large dataset size $N_{\mathcal{D}}$, since it scales linearly with $N_{\mathcal{D}}$.

A more efficient method for optimization is *stochastic gradient descent* (SGD) (section A.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 A.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.

1.6.3 Bayesian inference

From a Bayesian perspective, we can improve upon ML through *maximum a posteriori* (MAP) estimation (section section A.2.1), or, going

even further, inference of a full approximate posterior distribution over the parameters (see section A.1.4).

1.7 Learning and Inference in Deep Latent Variable Models

1.7.1 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 taken 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 $p_{\boldsymbol{\theta}}(\mathbf{x})$ is a mixture-of-Gaussians distribution. For continuous $\mathbf{z}$, $p_{\boldsymbol{\theta}}(\mathbf{x})$ can be seen as an infinite mixture, which are potentially more powerful than discrete mixtures. Such marginal distributions are also called compound probability distributions.

1.7.2 Deep Latent Variable Models

We use the term *deep latent variable model* (DLVM) to denote a latent variable model $p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$ whose distributions are parameterized by neural networks. Such a model can be conditioned on some context, like $p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}|\mathbf{y})$. 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 $p_{\boldsymbol{\theta}}(\mathbf{x})$ can be very complex, i.e. contain almost arbitrary dependen-

cies. This expressivity makes deep latent-variable models attractive for approximating complicated underlying distributions $p^*(\mathbf{x})$.

Perhaps the simplest, and most common, DLVM is one that is specified as factorization with the following structure:

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

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

1.7.3 Example DLVM for multivariate Bernoulli data

A simple example DLVM, used in (Kingma and Welling, 2014) for binary data $\mathbf{x}$, is 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.

1.8 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 A.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.

2

Variational Autoencoders

In this chapter we explain the basics of variational autoencoders (VAEs).

2.1 Encoder or Approximate Posterior

In the previous chapter, we introduced deep latent-variable models (DLVMs), and the problem of estimating the log-likelihood and posterior distributions in such models. The framework of variational autoencoders (VAEs) provides a computationally efficient way for optimizing DLVMs jointly with a corresponding inference model using SGD.

To turn the DLVM's 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* or *recognition model*. 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:

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

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:

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

where $Pa(\mathbf{z}_j)$ is the set of parent variables of variable $\mathbf{z}_j$ in the directed graph. 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.

2.2 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.

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

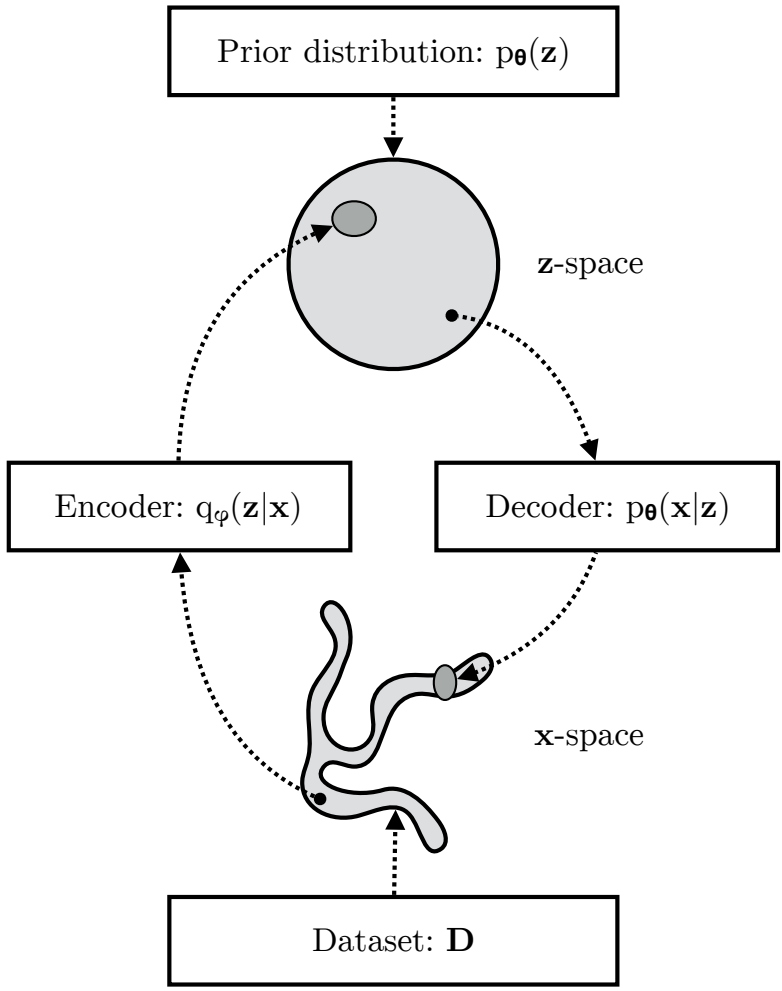


Figure 2.1: A VAE learns stochastic mappings between an observed $\mathbf{x}$-space, whose empirical distribution $q_{\mathcal{D}}(\mathbf{x})$ is typically complicated, and a latent $\mathbf{z}$-space, whose distribution can be relatively simple (such as spherical, as in this figure). The generative model learns a joint distribution $p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$ that is often (but not always) factorized as $p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) = p_{\boldsymbol{\theta}}(\mathbf{z})p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$, with 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.

variational parameters $\boldsymbol{\phi}$, we have:

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x})\right] \tag{2.5}$$

$$= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})}{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\right]\right] \tag{2.6}$$

$$= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\frac{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\right]\right] \tag{2.7}$$

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

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.

$$\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) = \log p_{\boldsymbol{\theta}}(\mathbf{x}) - D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})) \tag{2.11}$$

$$\leq \log p_{\boldsymbol{\theta}}(\mathbf{x}) \tag{2.12}$$

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.

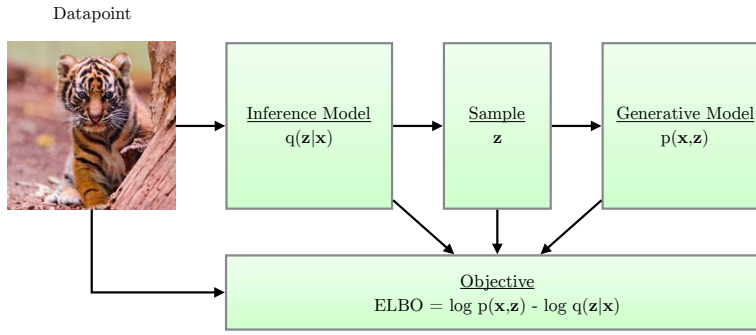


Figure 2.2: Simple schematic of computational flow in a variational autoencoder.

2.2.1 Two for One

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

1. It will approximately maximize the marginal likelihood $p_{\boldsymbol{\theta}}(\mathbf{x})$. This means that our generative model will become better.

2. It will minimize the KL divergence of the approximation $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ from the true posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$, so $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ becomes better.

2.3 Stochastic Gradient-Based Optimization of the ELBO

An important property of the ELBO, is that it allows *joint* optimization w.r.t. all parameters ($\boldsymbol{\phi}$ and $\boldsymbol{\theta}$) using stochastic gradient descent (SGD). We can start out with random initial values of $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$, 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:

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

The individual-datapoint ELBO, and its gradient $\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x})$ is, in general, intractable. However, good unbiased estimators $\tilde{\nabla}_{\boldsymbol{\theta},\boldsymbol{\phi}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x})$ 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 $\boldsymbol{\theta}$ are simple to obtain:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) = \nabla_{\boldsymbol{\theta}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\right] \tag{2.14}$$

$$= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \left[\nabla_{\boldsymbol{\theta}} (\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}))\right] \tag{2.15}$$

$$\simeq \nabla_{\boldsymbol{\theta}} (\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})) \tag{2.16}$$

$$= \nabla_{\boldsymbol{\theta}} (\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})) \tag{2.17}$$

The last line (eq. (2.17)) is a simple Monte Carlo estimator of the second line (eq. (2.15)), where $\mathbf{z}$ in the last two lines (eq. (2.16) and eq. (2.17)) is a random sample from $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$.

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

$$\nabla_{\boldsymbol{\phi}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) = \nabla_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\right] \tag{2.18}$$

$$\neq \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \left[\nabla_{\boldsymbol{\phi}} (\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}))\right] \tag{2.19}$$

In the case of continuous latent variables, we can use a reparameterization trick for computing unbiased estimates of $\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x})$, as we will now discuss. This stochastic estimate allows us to optimize the ELBO using SGD; see algorithm 1. See section 2.9.1 for a discussion of variational methods for discrete latent variables.

2.4 Reparameterization Trick

For continuous latent variables and a differentiable encoder and generative model, the ELBO can be straightforwardly differentiated w.r.t. both $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ through a change of variables, also called the *reparameterization trick* (Kingma and Welling, 2014 and Rezende *et al.*, 2014).

2.4.1 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 $\mathbf{x}$ 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 $\boldsymbol{\phi}$.

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

2.4.2 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:

$$\begin{aligned} \nabla_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[f(\mathbf{z})\right] &= \nabla_{\boldsymbol{\phi}} \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[f(\mathbf{z})\right] && (2.22)\\ &= \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\nabla_{\boldsymbol{\phi}} f(\mathbf{z})\right] && (2.23)\\ &\simeq \nabla_{\boldsymbol{\phi}} f(\mathbf{z}) && (2.24) \end{aligned}$$

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, and figure 3.2 for an illustration of the resulting posteriors for a 2D toy problem.

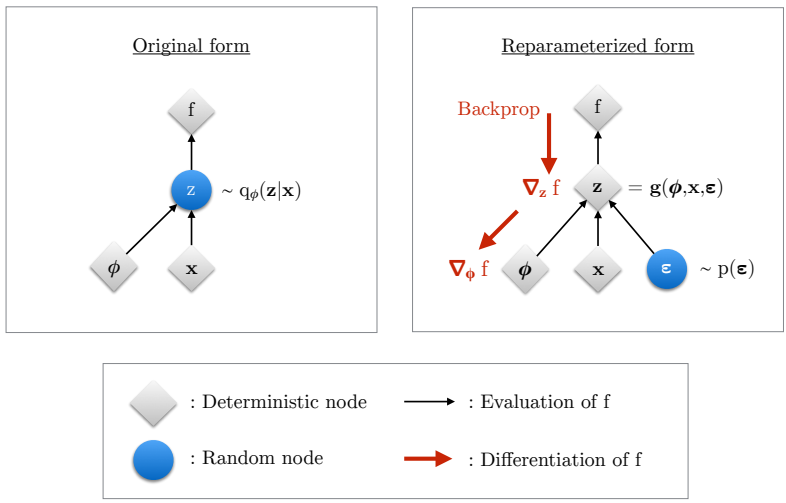


Figure 2.3: Illustration of the reparameterization trick. The variational parameters ϕ affect the objective f through the random variable $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$. We wish to compute gradients $\nabla_\phi f$ to optimize the objective with SGD. In the original form (left), we cannot differentiate f w.r.t. ϕ, 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 ϕ, $\mathbf{x}$, and a newly introduced random variable $\boldsymbol{\epsilon}$. This allows us to 'backprop through $\mathbf{z}$', and compute gradients $\nabla_\phi f$.

2.4.3 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, 2014. 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, 2014).

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:

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

$$= \nabla_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\right]) \tag{2.31}$$

$$= \nabla_{\boldsymbol{\theta},\boldsymbol{\phi}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) \tag{2.32}$$

2.4.4 Computation of $\log q_{\phi}(\mathbf{z}|\mathbf{x})$

Computation of the (estimator of) the ELBO requires computation of the density $\log q_{\phi}(\mathbf{z}|\mathbf{x})$, 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 $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$.

2.5 Factorized Gaussian posteriors

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

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

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

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

2.5.1 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:

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

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

and then proceeding with $\mathbf{z} = \boldsymbol{\mu} + \mathbf{L}\boldsymbol{\epsilon}$ as described above. $\mathbf{L}_{mask}$ is a masking matrix with zeros on and above the diagonal, and ones below the diagonal. Note that due to the masking $\mathbf{L}$, the Jacobian matrix $(\partial \mathbf{z}/\partial \boldsymbol{\epsilon})$ is triangular with the values of $\boldsymbol{\sigma}$ on the diagonal. The log-determinant is therefore 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 3.

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.

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 $\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x})$

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

2.6 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 likelhood 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. We can also use the estimator of eq. (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 *et al.*, 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.

2.7 Marginal Likelihood and ELBO as KL Divergences

One way to improve the potential tightness of the ELBO, 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:

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

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

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:

$$D_{KL}(q_{\mathcal{D}}(\mathbf{x}) || p_{\boldsymbol{\theta}}(\mathbf{x})) = -\mathbb{E}_{q_{\mathcal{D}}(\mathbf{x})} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x}) \right] + \mathbb{E}_{q_{\mathcal{D}}(\mathbf{x})} \left[\log q_{\mathcal{D}}(\mathbf{x}) \right] \tag{2.61}$$

$$= -\log p_{\boldsymbol{\theta}}(\mathcal{D}) + \text{constant} \tag{2.62}$$

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 $q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{x}, \mathbf{z})$ from $p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$ can be written as the negative ELBO, plus a constant:

$$D_{KL}(q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{x}, \mathbf{z}) || p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})) \tag{2.63}$$

$$= -\mathbb{E}_{q_{\mathcal{D}}(\mathbf{x})} \left[\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \right] - \log q_{\mathcal{D}}(\mathbf{x}) \right] \tag{2.64}$$

$$= -\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathcal{D}) + \text{constant} \tag{2.65}$$

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 summa-

rized in the following simple equation:

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

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. See figure 2.4.

2.8 Challenges

2.8.1 Optimization issues

In our work, consistent with findings in (Bowman *et al.*, 2015) and (Sønderby *et al.*, 2016a), 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 $q(\mathbf{z}|\mathbf{x}) \approx p(\mathbf{z})$, resulting in a stable equilibrium from which it is difficult to escape. The solution proposed in (Bowman *et al.*, 2015) and (Sønderby *et al.*, 2016a) 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 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

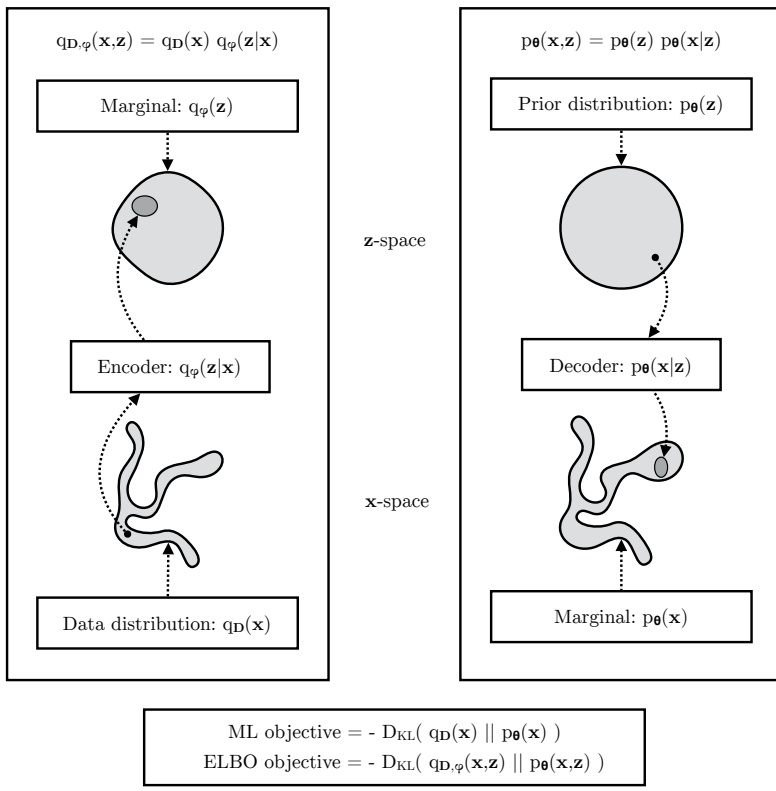


Figure 2.4: The maximum likelihood (ML) objective can be viewed as the minimization of $D_{KL}(q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{x}))$, while the ELBO objective can be viewed as the minimization of $D_{KL}(q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{x},\mathbf{z})||p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}))$, 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.

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 it was found that the method worked 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.

2.8.2 Blurriness of generative model

In section 2.7 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.

2.9 Related prior and concurrent work

Here we briefly discuss relevant literature prior to and concurrent with the work in (Kingma and Welling, 2014).

The wake-sleep algorithm (Hinton *et al.*, 1995) is another on-line learning method, 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.

Variational inference has a long history in the field of machine learning. We refer to (Wainwright and Jordan, 2008) for a comprehensive overview and synthesis of ideas around variational inference for exponential family graphical models. Among other connections, (Wainwright and Jordan, 2008) shows how various inference algorithms (such as expectation propagation, sum-product, max-product and many others) can be understood as exact or approximate forms of variational inference.

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 (Salimans 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 autoencoders. A connection between *linear* autoencoders 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 $p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}; \mathbf{Wz}, \epsilon\mathbf{I})$, 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:

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

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 *et al.*, 2014)

where noisy autoencoders 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.*, 2014), also learns a directed probabilistic model using an autoencoding structure, however their method applies to binary latent variables. In concurrent work, Rezende *et al.*, 2014 also make the connection between autoencoders, directed probabilistic models and stochastic variational inference using the reparameterization trick we describe in (Kingma and Welling, 2014). Their work was developed independently of ours and provides an additional perspective on the VAE.

2.9.1 Score function estimator

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

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

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

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

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 inference* (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.*, 2017), 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 $\nabla_{\mathbf{z}} \log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$.

3

Beyond Gaussian Posteriors

In this chapter we discuss techniques for improving the flexibility of the inference model $q_{\phi}(\mathbf{z}|\mathbf{x})$. Increasing the flexibility and accuracy of the inference model wel generally improve the tightness of the variational bound (ELBO), bringing it closer the true marginal likelihood objective.

3.1 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 $q_{\phi}(\mathbf{z}|\mathbf{x})$, 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.

3.2 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.

3.2.1 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.*, 2016) 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:

$$q_{\boldsymbol{\phi}}(\mathbf{u}, \mathbf{z}|\mathbf{x}) = q_{\boldsymbol{\phi}}(\mathbf{u}|\mathbf{x})q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{u}, \mathbf{x}) \tag{3.1}$$

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

$$q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = \int q_{\boldsymbol{\phi}}(\mathbf{u}, \mathbf{z}|\mathbf{x})\, d\mathbf{u} \tag{3.2}$$

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:

$$p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}, \mathbf{u}) = p_{\boldsymbol{\theta}}(\mathbf{u}|\mathbf{x}, \mathbf{z})p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) \tag{3.3}$$

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

$$\mathbb{E}_{q_{\mathcal{D}}(\mathbf{x})}\left[\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{u},\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}, \mathbf{u}) - \log q_{\boldsymbol{\phi}}(\mathbf{u}, \mathbf{z}|\mathbf{x})\right]\right] \tag{3.4}$$

$$= D_{KL}(q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{x}, \mathbf{z}, \mathbf{u})||p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}, \mathbf{u}) \tag{3.5}$$

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:

$$
\begin{aligned}
& D_{KL}(q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{x}, \mathbf{z}, \mathbf{u})||p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}, \mathbf{u})) && (3.6)\\
& \qquad (= \text{ELBO loss with auxiliary variables})\\
& = D_{KL}(q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{x}, \mathbf{z})||p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})) + \mathbb{E}_{q_{\mathcal{D}}(\mathbf{x},\mathbf{z})}\left[D_{KL}(q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{u}|\mathbf{x}, \mathbf{z})||p_{\boldsymbol{\theta}}(\mathbf{u}|\mathbf{x}, \mathbf{z}))\right]\\
& \geq D_{KL}(q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{x}, \mathbf{z})||p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})) && (3.7)\\
& \qquad (= \text{original ELBO objective}))\\
& = D_{KL}(q_{\mathcal{D}}(\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{x})) + \mathbb{E}_{q_{\mathcal{D}}(\mathbf{x})}\left[D_{KL}(q_{\mathcal{D},\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}))\right] && (3.8)\\
& \geq D_{KL}(q_{\mathcal{D}}(\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{x})) && (3.9)\\
& \qquad (= \text{Marginal log-likelihood objective})
\end{aligned}
$$

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 inference distributions, $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$, 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.*, 2016) 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 chapter 4. In our experiment with CIFAR-10, we make use of multiple layers of stochastic variables.

3.2.2 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[1]:

$$\boldsymbol{\epsilon}_0 \sim p(\boldsymbol{\epsilon}) \tag{3.10}$$

$$\text{for } t = 1...T: \tag{3.11}$$

$$\boldsymbol{\epsilon}_t = \mathbf{f}_t(\boldsymbol{\epsilon}_{t-1}, \mathbf{x}) \tag{3.12}$$

$$\mathbf{z} = \boldsymbol{\epsilon}_T \tag{3.13}$$

The Jacobian of the transformation factorizes:

$$\frac{d\mathbf{z}}{d\boldsymbol{\epsilon}_0} = \prod_{t=1}^{T} \frac{d\boldsymbol{\epsilon}_t}{d\boldsymbol{\epsilon}_{t-1}} \tag{3.14}$$

So its determinant also factorizes as well:

$$\log \left| \det \left(\frac{d\mathbf{z}}{d\boldsymbol{\epsilon}_0} \right) \right| = \sum_{t=1}^{T} \log \left| \det \left(\frac{d\boldsymbol{\epsilon}_t}{d\boldsymbol{\epsilon}_{t-1}} \right) \right| \tag{3.15}$$

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:

$$\log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = \log p(\boldsymbol{\epsilon}_0) - \sum_{t=1}^{T} \log \det \left| \frac{d\boldsymbol{\epsilon}_t}{d\boldsymbol{\epsilon}_{t-1}} \right| \tag{3.16}$$

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

$$\mathbf{f}_t(\boldsymbol{\epsilon}_{t-1}) = \boldsymbol{\epsilon}_{t-1} + \mathbf{u} h(\mathbf{w}^T \boldsymbol{\epsilon}_{t-1} + b) \tag{3.17}$$

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.

[1] 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.

3.3 Inverse Autoregressive Transformations

In order to find a type of normalizing flow that scales well to a high-dimensional space, Kingma *et al.*, 2016 consider Gaussian versions of autoregressive autoencoders such as MADE (Germain *et al.*, 2015) and the PixelCNN (Van Oord *et al.*, 2016). 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 matrix 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:

$$\epsilon_i = \frac{y_i - \mu_i(\mathbf{y}_{1:i-1})}{\sigma_i(\mathbf{y}_{1:i-1})} \tag{3.18}$$

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:

$$\boldsymbol{\epsilon} = (\mathbf{y} - \boldsymbol{\mu}(\mathbf{y}))/\boldsymbol{\sigma}(\mathbf{y}) \tag{3.19}$$

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

$$\log\det\left|\frac{d\boldsymbol{\epsilon}}{d\mathbf{y}}\right| = \sum_{i=1}^{D} -\log\sigma_i(\mathbf{y}) \tag{3.20}$$

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:

$$\boldsymbol{\epsilon} = \boldsymbol{\sigma}(\mathbf{y}) \odot \mathbf{y} + \boldsymbol{\mu}(\mathbf{y}) \tag{3.21}$$

With corresponding log-determinant:

$$\log\det\left|\frac{d\boldsymbol{\epsilon}}{d\mathbf{y}}\right| = \sum_{i=1}^{D} \log\sigma_i(\mathbf{y}) \tag{3.22}$$

3.4 Inverse Autoregressive Flow (IAF)

Kingma *et al.*, 2016 propose inverse autoregressive flow (IAF) based on a chain of transformations that are each equivalent to an inverse autoregressive transformation of eq. (3.19) and eq. (3.21). See algorithm 3 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 $q_{\boldsymbol{\phi}}(\mathbf{z}_0|\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}_0, \mathrm{diag}(\boldsymbol{\sigma}_0)^2)$:

$$\boldsymbol{\epsilon}_0 \sim \mathcal{N}(0, I) \tag{3.23}$$

$$(\boldsymbol{\mu}_0, \log\boldsymbol{\sigma}_0, \mathbf{h}) = \mathrm{EncoderNeuralNet}(\mathbf{x}; \boldsymbol{\theta}) \tag{3.24}$$

$$\mathbf{z}_0 = \boldsymbol{\mu}_0 + \boldsymbol{\sigma}_0 \odot \boldsymbol{\epsilon}_0 \tag{3.25}$$

Algorithm 3: Pseudo-code of an approximate posterior with Inverse Autoregressive Flow (IAF).

Input: $\texttt{EncoderNN}(\mathbf{x}; \boldsymbol{\theta})$ is an encoder neural network, with additional output $\mathbf{h}$.
Input: $\texttt{AutoregressiveNN}(\mathbf{z}; \mathbf{h}, t, \boldsymbol{\theta})$ is a neural network that is autoregressive over $\mathbf{z}$, with additional inputs $\mathbf{h}$ and t.
Input: T signifies the number of steps of flow.
Data:
$\mathbf{x}$: a datapoint, and optionally other conditioning information
$\boldsymbol{\theta}$: neural network parameters
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 '$\mathbf{z}$'

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

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

$$(\boldsymbol{\mu}_t, \boldsymbol{\sigma}_t) = \text{AutoregressiveNeuralNet}_t(\boldsymbol{\epsilon}_{t-1}, \mathbf{h}; \boldsymbol{\theta}) \tag{3.26}$$

$$\boldsymbol{\epsilon}_t = \boldsymbol{\mu}_t + \boldsymbol{\sigma}_t \odot \boldsymbol{\epsilon}_{t-1} \tag{3.27}$$

Each step of this flow is an inverse autoregressive transformation of the type of eq. (3.19) and eq. (3.21), and each step uses a separate autoregressive neural network. Following eq. (3.16), the density under

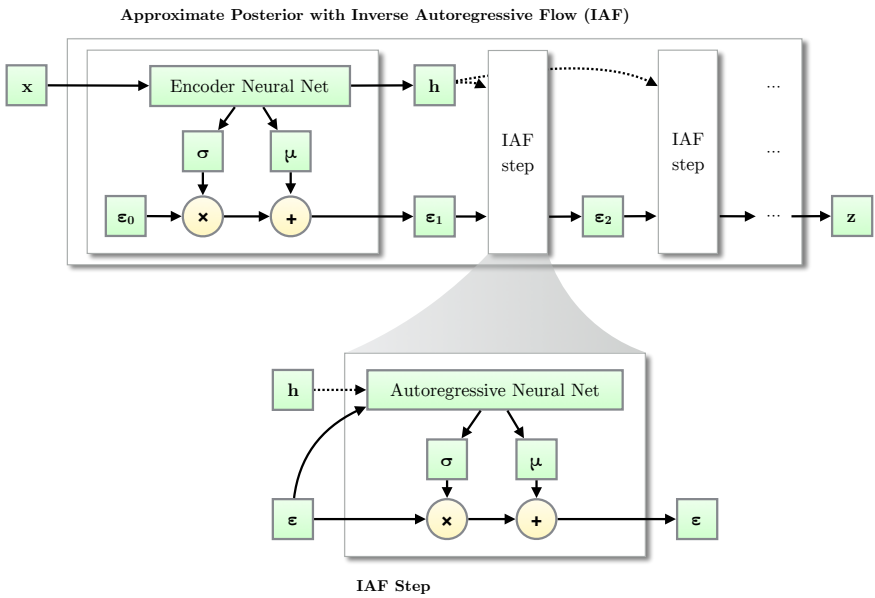


Figure 3.1: Like other normalizing flows, drawing samples from an approximate posterior with Inverse Autoregressive Flow (IAF) (Kingma *et al.*, 2016) starts with a distribution with tractable density, such as a Gaussian with diagonal covariance, followed by a chain of nonlinear invertible transformations of $\mathbf{z}$, each with a simple Jacobian determinant. The final iterate has a flexible distribution.

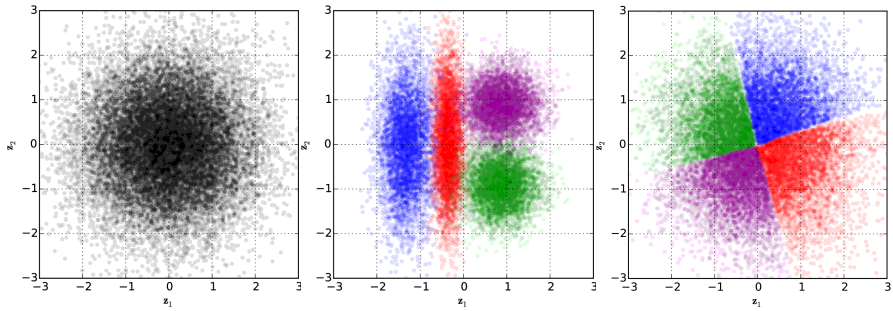
the final iterate is:

$$\mathbf{z} \equiv \boldsymbol{\epsilon}_T \tag{3.28}$$

$$\log q(\mathbf{z}|\mathbf{x}) = -\sum_{i=1}^{D} \left(\tfrac{1}{2}\epsilon_i^2 + \tfrac{1}{2}\log(2\pi) + \sum_{t=0}^{T} \log \sigma_{t,i} \right) \tag{3.29}$$

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 autoregressive models and the depth of the chain. See figure 3.1 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 uncon-



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

Figure 3.2: Best viewed in color. We fitted a variational autoencoder (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.

strained real-valued vectors, and compute $\boldsymbol{\epsilon}_t$ as:

$$(\mathbf{m}_t, \mathbf{s}_t) = \text{AutoregressiveNeuralNet}_t(\boldsymbol{\epsilon}_{t-1}, \mathbf{h}; \boldsymbol{\theta}) \tag{3.30}$$

$$\boldsymbol{\sigma}_t = \text{sigmoid}(\mathbf{s}_t) \tag{3.31}$$

$$\boldsymbol{\epsilon}_t = \boldsymbol{\sigma}_t \odot \boldsymbol{\epsilon}_{t-1} + (1 - \boldsymbol{\sigma}_t) \odot \mathbf{m}_t \tag{3.32}$$

This version is shown in algorithm 3. Note that this is just a particular version of the update of eq. (3.27), so the simple computation of the final log-density of eq. (3.29) still applies.

It was found beneficial for results to parameterize or initialize the parameters of each $\text{AutoregressiveNeuralNet}_t$ 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 $\boldsymbol{\epsilon}$ 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 convolutional autoregressive autoencoders introduced by (Van Oord *et al.*, 2016; Van den Oord *et al.*, 2016) was used.

It was 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. (3.29) remains unchanged.

3.5 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 flow (eq. (3.17)) and radial flow. These flows are shown to be effective to problems with a relatively low-dimensional latent space. 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*).

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 a transformation could thus take us arbitrarily close to the exact posterior distribution if we can apply it 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.*, 2016; Tran *et al.*, 2015), discussed in 3.2.1, 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.

4

Deeper Generative Models

In the previous chapter we explain advanced strategies for improving inference models. In this chapter, we review strategies for learning deeper generative models, such as inference and learning with multiple latent variables or observed variables, and techniques for improving the flexibility of the generative models $p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$.

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 the 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:

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

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:

$$\text{for } i = 1...K: \tag{4.3}$$
$$\boldsymbol{\epsilon}_i \sim p(\boldsymbol{\epsilon}_i) \tag{4.4}$$
$$\mathbf{z}_i = \mathbf{g}_i(\boldsymbol{\epsilon}_i, Pa(\mathbf{z}_i), \boldsymbol{\phi}) \tag{4.5}$$

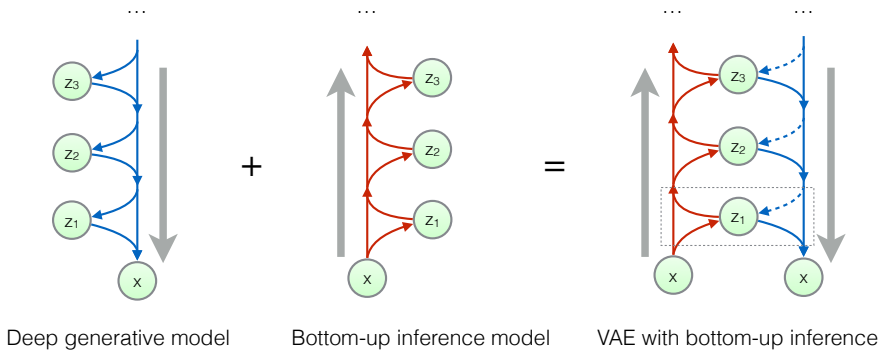
2. Evaluating the scalar value $(\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}))$ at the resulting sample $\mathbf{z}$ and datapoint $\mathbf{x}$. This scalar is the unbiased stochastic estimate lower bound on $\log p_{\boldsymbol{\theta}}(\mathbf{x})$.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

4.1.1 Choice of ordering

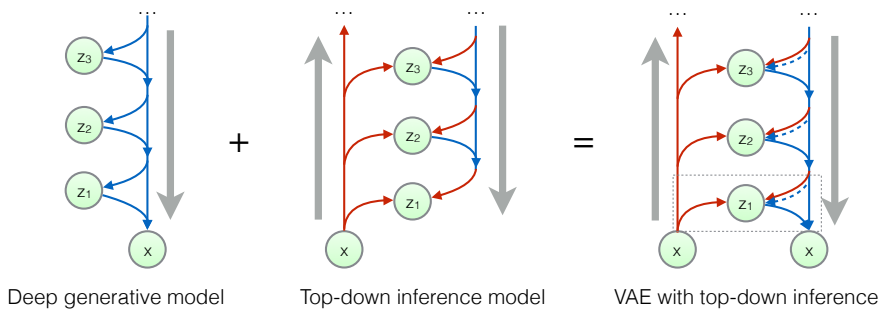
It should be noted that the choice of latent variables' topological ordering for the inference model can be different from the choice of ordering for the generative model.

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; Sønderby *et al.*, 2016a; 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.



(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.

To see why this might be a good idea, note 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:

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

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 is explored by (Salimans, 2016) and (Sønderby *et al.*, 2016a). In principle this idea could be extended to a more general class of conjugate priors, but 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

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).

Synthesis from fully autoregressive models models is relatively slow, since the *length of computation* for synthesis from such models is linear in the dimensionality of the data. The length of computation of

the log-likelihood of fully autoregressive models does not necesarilly scale with the dimensionality of the data. In this respect, 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. See (Kingma *et al.*, 2016) for experiments demonstrating a 100x improvement in speed of synthesis.

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

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.*, 2017.

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. Autoregressive models also require choosing a specific ordering of input elements (equation (4.8)). When no single 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 tractable Jacobian determinant (also called *normalizing flow*) 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 normalizing flows 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 of 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; Dinh *et al.*, 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 model has 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.

4.5 Follow-Up Work

Some important applications and motivations for deep generative models and variational autoencoders are:

- Representation learning: learning better representations of the data. Some uses of this are:
  - Data-efficient learning, such as semi-supervised learning
  - Visualisation of data as low-dimensional manifolds
- Artificial creativity: plausible interpolation between data and extrapolation from data.

Here we will now highlight some concrete applications to representation learning and artificial creativity.

4.5.1 Representation Learning

In the 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 competion (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.*, 2015; He *et al.*, 2016).

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.*, 2016a), 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.

4.5.2 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*.

Chemical Design

One example of a recent scientific application of artificial creativity, is shown in Gómez-Bombarelli *et al.*, 2018. 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

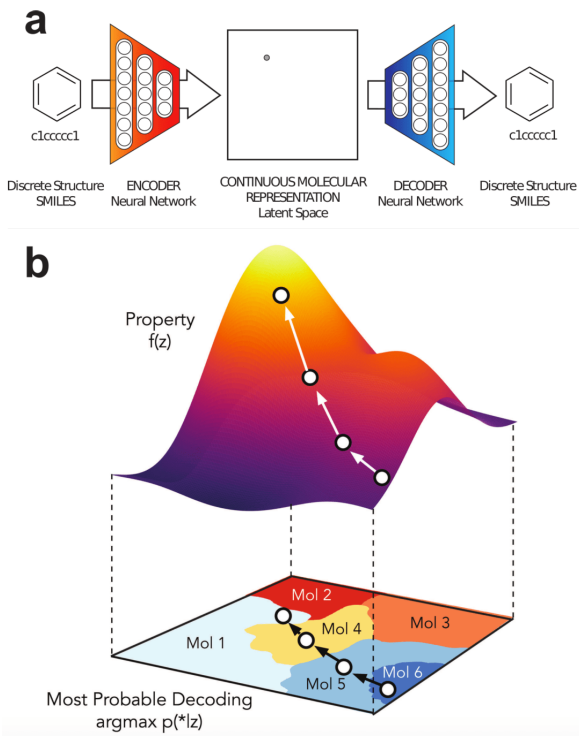


Figure 4.2: (a) Application of a VAE to chemical design in (Gómez-Bombarelli *et al.*, 2018). A latent continuous representation $\mathbf{z}$ of molecules is learned on a large dataset of molecules. (b) This continuous representation enables gradient-based search of new molecules that maximizes $f(\mathbf{z})$, a certain desired property.

towards certain properties; the method is demonstrated on the design of drug-like molecules and organic light-emitting diodes. See figure 4.2.

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 succesfully interpolate between sentences, and for imputation of missing words. See figure 4.3.

“ 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 4.3: 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.

Astronomy

In (Ravanbakhsh *et al.*, 2017), 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.

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 simple 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 of images in latent space along a "smile vector" in order to make them more happy, or more sad looking. See figure 4.4 for an example.

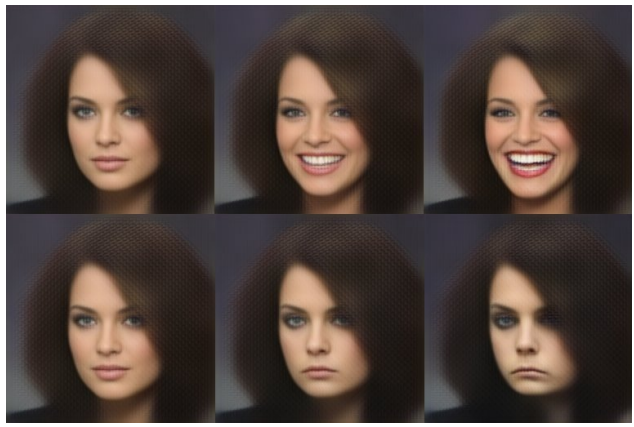


Figure 4.4: VAEs can be used for image resynthesis. In this example by White, 2016, an original image (left) is modified in a latent space in the direction of a *smile vector*, producing a range of versions of the original, from smiling to sadness.

4.5.3 Other relevant follow-up work

We unfortunately do not have space to discuss all follow-up work in depth, but will here highlight a selection of relevant recent work.

In addition to our original publication (Kingma and Welling, 2014), two later papers have proposed equivalent algorithms (Rezende *et al.*, 2014; Lázaro-Gredilla, 2014), where the latter work applies the same reparameterization gradient method to the estimation of parameter posteriors, rather than amortized latent-variable inference.

In the appendix of (Kingma and Welling, 2014) we proposed to apply the reparameterization gradients to estimation of parameter posteriors. In (Blundell *et al.*, 2015) this method, with a mixture-of-Gaussians prior and named *Bayes by Backprop*, was used in experiments with some promising early results. In (Kingma *et al.*, 2015) we describe a refined method, the *local reparameterization trick*, for further decreasing the variance of the gradient estimator, and applied it to estimation of Gaussian parameter posteriors. Further results were presented in (Louizos *et al.*, 2017; Louizos and Welling, 2017; Louizos and Welling, 2016) with increasingly sophisticated choices of priors and approximate posteriors. In (Kingma *et al.*, 2015; Gal and Ghahramani, 2016), a similar reparameterization was used to analyze Dropout as a Bayesian method, coined

Variational Dropout. In (Molchanov *et al.*, 2017) this method was further analyzed and refined. Various papers have applied reparameterization gradients for estimating parameter posteriors, including (Fortunato *et al.*, 2017) in the context of recurrent neural networks and (Kucukelbir *et al.*, 2017) more generally for Bayesian models and in (Tran *et al.*, 2017) for deep probabilistic programming. A Bayesian nonparametric variational family based in the Gaussian Process using reparameterization gradients was proposed in (Tran *et al.*, 2015).

Normalizing flows (Rezende and Mohamed, 2015) were proposed as a framework for improving the flexibility of inference models. In Kingma *et al.*, 2016, the first normalizing flow was proposed that scales well to high-dimensional latent spaces. The same principle was later applied in (Papamakarios *et al.*, 2017) for density estimation, and further refined in (Huang *et al.*, 2018). Various other flows were proposed in (Tomczak and Welling, 2016; Tomczak and Welling, n.d.) and (Berg *et al.*, 2017).

As an alternative to (or in conjunction with) normalizing flows, one can use auxiliary variables to improve posterior flexibility. This principle was, to the best of our knowledge, first proposed in Salimans *et al.*, 2015. In this paper, the principle was used in a combination of variational inference with Hamiltonian Monte Carlo (HMC), with the momentum variables of HMC as auxiliary variables. Auxiliary variables were more elaborately discussed in in (Maaløe *et al.*, 2016) as Auxiliary Deep Generative Models. Similarly, one can use deep models with multiple stochastic layers to improve the variational bound, as demonstrated in (Sønderby *et al.*, 2016a) and (Sønderby *et al.*, 2016b) as Ladder VAEs.

There has been plenty of follow-up work on gradient variance reduction for the variational parameters of discrete latent variables, as opposed to continuous latent variables for which reparameterization gradients apply. These proposals include NVIL (Mnih and Gregor, 2014), MuProp (Gu *et al.*, 2015), Variational inference for Monte Carlo objectives (Mnih and Rezende, 2016), the Concrete distribution (Maddison *et al.*, 2017) and Categorical Reparameterization with Gumbel-Softmax (Jang *et al.*, 2017).

The ELBO objective can be generalized into an importance-weighted objective, as proposed in (Burda *et al.*, 2015) (Importance-Weighted Autoencoders). This potentially reduces the variance in the gradient,

but has not been discussed in-depth here since (as often the case with importance-weighted estimators) it can be difficult to scale to high-dimensional latent spaces. Other objectives have been proposed such as Rényi divergence variational inference (Li and Turner, 2016), Generative Moment Matching Networks (Li *et al.*, 2015), objectives based on normalizing such as NICE and RealNVP flows (Sohl-Dickstein *et al.*, 2015; Dinh *et al.*, 2014), black-box α-divergence minimization (Hernández-Lobato *et al.*, 2016) and Bi-directional Helmholtz Machines (Bornschein *et al.*, 2016).

Various combinations with adversarial objectives have been proposed. In (Makhzani *et al.*, 2015), the "adversarial autoencoder" (AAE) was proposed, a probabilistic autoencoder that uses a generative adversarial network (GAN) (Goodfellow *et al.*, 2014) to perform variational inference. In (Dumoulin *et al.*, 2017) Adversarially Learned Inference (ALI) was proposed, which aims to minimize a GAN objective between the joint distributions $q_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{z})$ and $p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$. Other hybrids have been proposed as well (Larsen *et al.*, 2016; Brock *et al.*, 2017; Hsu *et al.*, 2017).

One of the most prominent, and most difficult, applications of generative models is image modeling. In (Kulkarni *et al.*, 2015) (Deep convolutional inverse graphics network), a convolutional VAE was applied to modeling images with some success, building on work by (Dosovitskiy *et al.*, 2015) proposing convolutional networks for image synthesis. In (Gregor *et al.*, 2015) (DRAW), an attention mechanism was combined with a recurrent inference model and recurrent generative model for image synthesis. This approach was further extended in (Gregor *et al.*, 2016) (Towards Conceptual Compression) with convolutional networks, scalable to larger images, and applied to image compression. In (Kingma *et al.*, 2016), deep convolutional inference models and generative models were also applied to images. Furthermore, (Gulrajani *et al.*, 2017) (PixelVAE) and (Chen *et al.*, 2017) (Variational Lossy Autoencoder) combined convolutional VAEs with the PixelCNN model (Van Oord *et al.*, 2016; Van den Oord *et al.*, 2016). Methods and VAE architectures for controlled image generation from attributes or text were studied in (Kingma *et al.*, 2014; Yan *et al.*, 2016; Mansimov *et al.*, 2015; Brock *et al.*, 2017; White, 2016). Predicting the color of pixels based on a

grayscale image is another promising application (Deshpande *et al.*, 2017). The application to semi-supervised learning has been studied in (Kingma *et al.*, 2014; Pu *et al.*, 2016; Xu *et al.*, 2017) among other work.

Another prominent application of VAEs is modeling of text and or sequential data (Bayer and Osendorfer, 2014; Bowman *et al.*, 2015; Serban *et al.*, 2017; Johnson *et al.*, 2016; Karl *et al.*, 2017; Fraccaro *et al.*, 2016; Miao *et al.*, 2016; Semeniuta *et al.*, 2017; Zhao *et al.*, 2017; Yang *et al.*, 2017; Hu *et al.*, 2017). VAEs have also been applied to speech and handwriting Chung *et al.*, 2015. Sequential models typically use recurrent neural networks, such as LSTMs (Hochreiter and Schmidhuber, 1997), as encoder and/or decoder. When modeling sequences, the validity of a sequence can sometimes be constrained by a context-free grammar. In this case, incorporation of the grammar in VAEs can lead to better models, as shown in (Kusner *et al.*, 2017) (Grammar VAEs), and applied to modeling molecules in textual representations.

Since VAEs can transform discrete observation spaces to continuous latent-variable spaces with approximately known marginals, they are interesting for use in model-based control (Watter *et al.*, 2015; Pritzel *et al.*, 2017). In (Heess *et al.*, 2015) (Stochastic Value Gradients) it was shown that the re-parameterization of the observed variables, together with an observation model, can be used to compute novel forms of policy gradients. Variational inference and reparameterization gradients have also been used for variational information maximisation for intrinsically motivated reinforcement learning (Mohamed and Rezende, 2015) and VIME (Houthooft *et al.*, 2016) for improved exploration. Variational autoencoders have also been used as components in models that perform iterative reasoning about objects in a scene (Eslami *et al.*, 2016).

In (Higgins *et al.*, 2017) (β-VAE) it was proposed to strengthen the contribution of $D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z}))$, thus restricting the information flow through the latent space, which was shown to improve disentanglement of latent factors, further studied in (Chen *et al.*, 2018).

Other applications include modeling of graphs (Kipf and Welling, 2016) (Variational Graph Autoencoders), learning of 3D structure from images (Rezende *et al.*, 2016b), one-shot learning (Rezende *et al.*, 2016a), learning nonlinear state space models (Krishnan *et al.*, 2017), voice

conversion from non-parallel corpora (Hsu *et al.*, 2016), discrimination-aware (fair) representations (Louizos *et al.*, 2015) and transfer learning (Edwards and Storkey, 2017).

The reparameterization gradient estimator discussed in this work has been extended in various directions (Ruiz *et al.*, 2016), including acceptance-rejection sampling algorithms (Naesseth *et al.*, 2017). The gradient variance can in some cases be reduced by 'carving up the ELBO' (Hoffman and Johnson, 2016; Roeder *et al.*, 2017) and using a modified gradient estimator. A second-order gradient estimator has also been proposed in (Fan *et al.*, 2015).

All in all, this remains an actively researched area with frequently exciting developments.

5

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. However, one is often more interested in flexible models with latent variables, such as deep latent-variable models, or Bayesian models with random parameters. In both cases one needs to perform approximate posterior estimation for which variational inference (VI) methods are suitable. In VI, 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 framework for efficient and scalable gradient-based variational posterior inference and approximate maximum likelihood learning.

In this work we describe the variational autoencoder (VAE) and some of its extensions. A VAE is a combination of a *deep latent-variable*

model (DLVM) with continuous latent variables, and an associated *inference model*. The DLVM is a type of generative model over the data. The inference model, also called *encoder* or *recognition model*, approximates the posterior distribution of the latent variables of the generative model. Both the generative model and the inference model are *directed graphical models* that are wholly or partially parameterized by deep neural networks. The parameters of the models, including the parameters of the neural networks such as the weights and biases, are jointly optimized by performing stochastic gradient ascent on the so-called *evidence lower bound* (ELBO). The ELBO is a lower bound on the marginal likelihood of the data, also called the *variational lower bound*. Stochastic gradients, necessary for performing SGD, are obtained through a basic *reparameterization trick*. 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.

For learning flexible inference models, we proposed inverse autoregressive flows (IAF), a type of normalizing flow that allows scaling to high-dimensional latent spaces. 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.*, 2016), 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.

Acknowledgements

We are grateful for the help of Tim Salimans, Alec Radford, Rif A. Saurous and others who have given us valuable feedback at various stages of writing.

Appendices

A

Appendix

A.1 Notation and definitions

A.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*.
$Pa(\mathbf{z})$	The parents of random variable $\mathbf{z}$ in a directed graph.
$\text{diag}(\mathbf{x})$	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 $\mathbf{x}$ 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.

A.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.

A.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$(\mathbf{x}; \mathbf{p}) = \prod_i \text{Bernoulli}(x_i; p_i)$ 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 $\boldsymbol{\Sigma}$.

Chain rule of probability

$$p(\mathbf{a}, \mathbf{b}) = p(\mathbf{a})p(\mathbf{b}|\mathbf{a}) \tag{A.1}$$

Bayes' Rule

$$p(\mathbf{a}|\mathbf{b}) = p(\mathbf{b}|\mathbf{a})p(\mathbf{a})/p(\mathbf{b}) \tag{A.2}$$

A.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:

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} \propto p(\mathcal{D}|\theta)p(\theta) \tag{A.3}$$

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:

$$p(\mathbf{x} = \mathbf{x}'|\mathcal{D}) = \int p_{\boldsymbol{\theta}}(\mathbf{x} = \mathbf{x}')p(\theta|\mathcal{D})d\theta$$

A.2 Alternative methods for learning in DLVMs

A.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:

$$L^{MAP}(\theta) = \log p(\theta|\mathcal{D}) \tag{A.4}$$

$$= \log p(\theta) + L^{ML}(\theta) + \text{constant} \tag{A.5}$$

The prior $p(\theta)$ in equation (A.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 $p(\theta) = \text{constant}$.

A.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 A.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:

$$\forall i = 1, ..., N: \quad \boldsymbol{\phi}^{(i)} \leftarrow \underset{\boldsymbol{\phi}}{\text{argmax}}\, \mathcal{L}(\mathbf{x}^{(i)}; \boldsymbol{\theta}, \boldsymbol{\phi}) \qquad \text{(E-step)} \qquad \text{(A.6)}$$

$$\boldsymbol{\theta} \leftarrow \underset{\boldsymbol{\theta}}{\text{argmax}} \sum_{i=1}^{N} \mathcal{L}(\mathbf{x}^{(i)}; \boldsymbol{\theta}, \boldsymbol{\phi}) \qquad \text{(M-step)} \qquad \text{(A.7)}$$

until convergence. Why does this work? Note that at the E-step:

$$\underset{\boldsymbol{\phi}}{\text{argmax}}\, \mathcal{L}(\mathbf{x}; \boldsymbol{\theta}, \boldsymbol{\phi}) \qquad \text{(A.8)}$$

$$= \underset{\boldsymbol{\phi}}{\text{argmax}}\, [\log p_{\boldsymbol{\theta}}(\mathbf{x}) - D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}))] \qquad \text{(A.9)}$$

$$= \underset{\boldsymbol{\phi}}{\text{argmin}}\, D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})) \qquad \text{(A.10)}$$

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 $D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}))$.

A.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. $\mathbf{z}$:

$$\begin{aligned}
\nabla_{\mathbf{z}} \log p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) &= \nabla_{\mathbf{z}} \log[p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})/p_{\boldsymbol{\theta}}(\mathbf{x})] && \text{(A.11)} \\
&= \nabla_{\mathbf{z}}[\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) - \log p_{\boldsymbol{\theta}}(\mathbf{x})] && \text{(A.12)} \\
&= \nabla_{\mathbf{z}} \log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) - \nabla_{\mathbf{z}} \log p_{\boldsymbol{\theta}}(\mathbf{x}) && \text{(A.13)} \\
&= \nabla_{\mathbf{z}} \log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) && \text{(A.14)}
\end{aligned}$$

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.

A.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 backpropagation 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:

$$\theta_{t+1} \leftarrow \theta_t + \alpha_t \cdot \nabla_\theta \tilde{L}(\theta, \xi) \tag{A.15}$$

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 have typically used the

Adam and Adamax optimization methods for choosing α_t (Kingma and Ba, 2015); 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. (A.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*.

References

Banerjee, A. (2007). "An analysis of logistic models: Exponential family connections and online performance". In: *Proceedings of the 2007 SIAM International Conference on Data Mining*. SIAM. 204–215.

Bayer, J. and C. Osendorfer (2014). "Learning stochastic recurrent networks". In: *NIPS 2014 Workshop on Advances in Variational Inference*.

Bengio, Y., A. Courville, and P. Vincent (2013). *Representation Learning: A Review and New Perspectives*. IEEE.

Bengio, Y., E. Laufer, G. Alain, and J. Yosinski (2014). "Deep generative stochastic networks trainable by backprop". In: *International Conference on Machine Learning*. 226–234.

Berg, R. v. d., L. Hasenclever, J. M. Tomczak, and M. Welling (2017). "Sylvester Normalizing Flows for Variational Inference". *Conference on Uncertainty in Artificial Intelligence*.

Blei, D. M., M. I. Jordan, and J. W. Paisley (2012). "Variational Bayesian inference with stochastic search". In: *International Conference on Machine Learning*. 1367–1374.

Blundell, C., J. Cornebise, K. Kavukcuoglu, and D. Wierstra (2015). "Weight Uncertainty in Neural Networks". In: *International Conference on Machine Learning*. 1613–1622.

Bornschein, J., S. Shabanian, A. Fischer, and Y. Bengio (2016). "Bidirectional Helmholtz machines". In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning*. 2511–2519.

Bourlard, H. and Y. Kamp (1988). "Auto-association by multilayer perceptrons and singular value decomposition". *Biological Cybernetics*. 59(4-5): 291–294.

Bowman, S. R., L. Vilnis, O. Vinyals, A. M. Dai, R. Jozefowicz, and S. Bengio (2015). "Generating sentences from a continuous space". *arXiv preprint arXiv:1511.06349*.

Brock, A., T. Lim, J. M. Ritchie, and N. J. Weston (2017). "Neural photo editing with introspective adversarial networks". In: *International Conference on Learning Representations*.

Burda, Y., R. Grosse, and R. Salakhutdinov (2015). "Importance weighted autoencoders". *arXiv preprint arXiv:1509.00519*.

Chen, R. T., X. Li, R. Grosse, and D. Duvenaud (2018). "Isolating sources of disentanglement in VAEs". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. Curran Associates Inc. 2615–2625.

Chen, X., D. P. Kingma, T. Salimans, Y. Duan, P. Dhariwal, J. Schulman, I. Sutskever, and P. Abbeel (2017). "Variational lossy autoencoder". *International Conference on Learning Representations*.

Chung, J., K. Kastner, L. Dinh, K. Goel, A. C. Courville, and Y. Bengio (2015). "A recurrent latent variable model for sequential data". In: *Advances in neural information processing systems*. 2980–2988.

Cremer, C., Q. Morris, and D. Duvenaud (2017). "Re-interpreting importance weighted autoencoders". *International Conference on Learning Representations*.

Dayan, P., G. E. Hinton, R. M. Neal, and R. S. Zemel (1995). "The Helmholtz machine". *Neural computation*. 7(5): 889–904.

Deco, G. and W. Brauer (1995). "Higher order statistical decorrelation without information loss". *Advances in Neural Information Processing Systems*: 247–254.

Dempster, A. P., N. M. Laird, and D. B. Rubin (1977). "Maximum likelihood from incomplete data via the EM algorithm". *Journal of the Royal Statistical Society. Series B (Methodological)*: 1–38.

Deshpande, A., J. Lu, M.-C. Yeh, M. Jin Chong, and D. Forsyth (2017). "Learning diverse image colorization". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 6837–6845.

Dinh, L., D. Krueger, and Y. Bengio (2014). "NICE: Non-linear independent components estimation". *arXiv preprint arXiv:1410.8516*.

Dinh, L., J. Sohl-Dickstein, and S. Bengio (2016). "Density estimation using Real NVP". *arXiv preprint arXiv:1605.08803*.

Dosovitskiy, A., J. Tobias Springenberg, and T. Brox (2015). "Learning to generate chairs with convolutional neural networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1538–1546.

Dumoulin, V., I. Belghazi, B. Poole, A. Lamb, M. Arjovsky, O. Mastropietro, and A. Courville (2017). "Adversarially learned inference". *International Conference on Learning Representations*.

Edwards, H. and A. Storkey (2017). "Towards a neural statistician". *International Conference on Learning Representations*.

Eslami, S. A., N. Heess, T. Weber, Y. Tassa, D. Szepesvari, G. E. Hinton, *et al.* (2016). "Attend, infer, repeat: Fast scene understanding with generative models". In: *Advances In Neural Information Processing Systems*. 3225–3233.

Fan, K., Z. Wang, J. Beck, J. Kwok, and K. A. Heller (2015). "Fast second order stochastic backpropagation for variational inference". In: *Advances in Neural Information Processing Systems*. 1387–1395.

Fortunato, M., C. Blundell, and O. Vinyals (2017). "Bayesian recurrent neural networks". *arXiv preprint arXiv:1704.02798*.

Fraccaro, M., S. K. Sønderby, U. Paquet, and O. Winther (2016). "Sequential neural models with stochastic layers". In: *Advances in Neural Information Processing Systems*. 2199–2207.

Fu, M. C. (2006). "Gradient estimation". *Handbooks in Operations Research and Management Science*. 13: 575–616.

Gal, Y. and Z. Ghahramani (2016). "A theoretically grounded application of dropout in recurrent neural networks". In: *Advances in neural information processing systems*. 1019–1027.

Germain, M., K. Gregor, I. Murray, and H. Larochelle (2015). "Made: Masked autoencoder for distribution estimation". In: *International Conference on Machine Learning*. 881–889.

Gershman, S. and N. Goodman (2014). "Amortized inference in probabilistic reasoning." In: *CogSci*.

Glasserman, P. (2013). *Monte Carlo methods in financial engineering*. Vol. 53. Springer Science & Business Media.

Glynn, P. W. (1990). "Likelihood ratio gradient estimation for stochastic systems". *Communications of the ACM*. 33(10): 75–84.

Gómez-Bombarelli, R., J. N. Wei, D. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams, and A. Aspuru-Guzik (2018). "Automatic chemical design using a data-driven continuous representation of molecules". *ACS central science*. 4(2): 268–276.

Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep learning*. MIT press.

Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2014). "Generative adversarial nets". In: *Advances in Neural Information Processing Systems*. 2672–2680.

Graves, A. (2011). "Practical variational inference for neural networks". In: *Advances in Neural Information Processing Systems*. 2348–2356.

Gregor, K., F. Besse, D. J. Rezende, I. Danihelka, and D. Wierstra (2016). "Towards conceptual compression". In: *Advances In Neural Information Processing Systems*. 3549–3557.

Gregor, K., I. Danihelka, A. Graves, D. Rezende, and D. Wierstra (2015). "DRAW: A Recurrent Neural Network For Image Generation". In: *International Conference on Machine Learning*. 1462–1471.

Gregor, K., I. Danihelka, A. Mnih, C. Blundell, and D. Wierstra (2014). "Deep AutoRegressive Networks". In: *International Conference on Machine Learning*. 1242–1250.

Grover, A., M. Dhar, and S. Ermon (2018). "Flow-GAN: Combining maximum likelihood and adversarial learning in generative models". In: *AAAI Conference on Artificial Intelligence*.

Gu, S., S. Levine, I. Sutskever, and A. Mnih (2015). "MuProp: Unbiased backpropagation for stochastic neural networks". *arXiv preprint arXiv:1511.05176*.

Gulrajani, I., K. Kumar, F. Ahmed, A. A. Taiga, F. Visin, D. Vazquez, and A. Courville (2017). "PixelVAE: A latent variable model for natural images". *International Conference on Learning Representations*.

He, K., X. Zhang, S. Ren, and J. Sun (2015). "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE International Conference on Computer Vision*. 1026–1034.

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

Heess, N., G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa (2015). "Learning continuous control policies by stochastic value gradients". In: *Advances in Neural Information Processing Systems*. 2944–2952.

Hernández-Lobato, J. M., Y. Li, M. Rowland, D. Hernández-Lobato, T. Bui, and R. E. Turner (2016). "Black-box α-divergence minimization".

Higgins, I., L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner (2017). "beta-vae: Learning basic visual concepts with a constrained variational framework". *International Conference on Learning Representations*.

Hinton, G. E., P. Dayan, B. J. Frey, and R. M. Neal (1995). "The "Wake-Sleep" algorithm for unsupervised neural networks". *Science*: 1158–1158.

Hochreiter, S. and J. Schmidhuber (1997). "Long Short-Term Memory". *Neural Computation*. 9(8): 1735–1780.

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

Hoffman, M. D. and M. J. Johnson (2016). "Elbo surgery: yet another way to carve up the variational evidence lower bound". In: *Workshop in Advances in Approximate Bayesian Inference, NIPS*.

Houthooft, R., X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel (2016). "Vime: Variational information maximizing exploration". In: *Advances in Neural Information Processing Systems*. 1109–1117.

Hsu, C.-C., H.-T. Hwang, Y.-C. Wu, Y. Tsao, and H.-M. Wang (2016). "Voice conversion from non-parallel corpora using variational auto-encoder". In: *Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2016 Asia-Pacific*. IEEE. 1–6.

Hsu, C.-C., H.-T. Hwang, Y.-C. Wu, Y. Tsao, and H.-M. Wang (2017). "Voice conversion from unaligned corpora using variational autoencoding wasserstein generative adversarial networks". *arXiv preprint arXiv:1704.00849*.

Hu, Z., Z. Yang, X. Liang, R. Salakhutdinov, and E. P. Xing (2017). "Controllable text generation". *arXiv preprint arXiv:1703.00955*.

Huang, C.-W., D. Krueger, A. Lacoste, and A. Courville (2018). "Neural Autoregressive Flows". In: *International Conference on Machine Learning*. 2083–2092.

Jang, E., S. Gu, and B. Poole (2017). "Categorical Reparameterization with Gumbel-Softmax". *International Conference on Learning Representations*.

Johnson, M., D. K. Duvenaud, A. Wiltschko, R. P. Adams, and S. R. Datta (2016). "Composing graphical models with neural networks for structured representations and fast inference". In: *Advances in Neural Information Processing Systems*. 2946–2954.

Jozefowicz, R., W. Zaremba, and I. Sutskever (2015). "An empirical exploration of recurrent network architectures". In: *International Conference on Machine Learning*. 2342–2350.

Karl, M., M. Soelch, J. Bayer, and P. van der Smagt (2017). "Deep variational bayes filters: Unsupervised learning of state space models from raw data". *International Conference on Learning Representations*.

Kavukcuoglu, K., M. Ranzato, and Y. LeCun (2008). "Fast inference in sparse coding algorithms with applications to object recognition". *Tech. rep.* No. CBLL-TR-2008-12-01. Computational and Biological Learning Lab, Courant Institute, NYU.

Kingma, D. P., S. Mohamed, D. J. Rezende, and M. Welling (2014). "Semi-supervised learning with deep generative models". In: *Advances in Neural Information Processing Systems*. 3581–3589.

Kingma, D. P., T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling (2016). "Improved variational inference with inverse autoregressive flow". In: *Advances in Neural Information Processing Systems*. 4743–4751.

Kingma, D. P., T. Salimans, and M. Welling (2015). "Variational dropout and the local reparameterization trick". In: *Advances in Neural Information Processing Systems*. 2575–2583.

Kingma, D. P. and M. Welling (2014). "Auto-Encoding Variational Bayes". *International Conference on Learning Representations*.

Kingma, D. and J. Ba (2015). "Adam: A Method for Stochastic Optimization". *International Conference on Learning Representations*.

Kipf, T. N. and M. Welling (2016). "Variational graph auto-encoders". *arXiv preprint arXiv:1611.07308*.

Kleijnen, J. P. and R. Y. Rubinstein (1996). "Optimization and sensitivity analysis of computer simulation models by the score function method". *European Journal of Operational Research*. 88(3): 413–427.

Koller, D. and N. Friedman (2009). *Probabilistic graphical models: Principles and techniques*. MIT press.

Krishnan, R. G., U. Shalit, and D. Sontag (2017). "Structured Inference Networks for Nonlinear State Space Models." In: *AAAI*. 2101–2109.

Kucukelbir, A., D. Tran, R. Ranganath, A. Gelman, and D. M. Blei (2017). "Automatic differentiation variational inference". *The Journal of Machine Learning Research*. 18(1): 430–474.

Kulkarni, T. D., W. F. Whitney, P. Kohli, and J. Tenenbaum (2015). "Deep convolutional inverse graphics network". In: *Advances in Neural Information Processing Systems*. 2539–2547.

Kusner, M. J., B. Paige, and J. M. Hernández-Lobato (2017). "Grammar variational autoencoder". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. 1945–1954.

Larsen, A. B. L., S. K. Sønderby, H. Larochelle, and O. Winther (2016). "Autoencoding beyond pixels using a learned similarity metric". In: *International Conference on Machine Learning*. 1558–1566.

Lázaro-Gredilla, M. (2014). "Doubly stochastic variational Bayes for non-conjugate inference". In: International Conference on Machine Learning.

LeCun, Y., Y. Bengio, and G. Hinton (2015). "Deep Learning". *Nature*. 521(7553): 436–444.

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

Li, Y. and R. E. Turner (2016). "Rényi divergence variational inference". In: *Advances in Neural Information Processing Systems*. 1073–1081.

Li, Y., K. Swersky, and R. S. Zemel (2015). "Generative moment matching networks". In: *International Conference on Machine Learning*. 1718–1727.

Linsker, R. (1989). *An Application of the Principle of Maximum Information Preservation to Linear Systems*. Morgan Kaufmann Publishers Inc.

Louizos, C., K. Swersky, Y. Li, M. Welling, and R. Zemel (2015). "The variational fair autoencoder". *arXiv preprint arXiv:1511.00830*.

Louizos, C., K. Ullrich, and M. Welling (2017). "Bayesian compression for deep learning". In: *Advances in Neural Information Processing Systems*. 3288–3298.

Louizos, C. and M. Welling (2016). "Structured and efficient variational deep learning with matrix gaussian posteriors". In: *International Conference on Machine Learning*. 1708–1716.

Louizos, C. and M. Welling (2017). "Multiplicative normalizing flows for variational Bayesian neural networks". In: *International Conference on Machine Learning*. 2218–2227.

Maaløe, L., C. K. Sønderby, S. K. Sønderby, and O. Winther (2016). "Auxiliary deep generative models". In: *International Conference on Machine Learning*.

Maddison, C. J., A. Mnih, and Y. W. Teh (2017). "The concrete distribution: A continuous relaxation of discrete random variables". *International Conference on Learning Representations*.

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

Mansimov, E., E. Parisotto, J. L. Ba, and R. Salakhutdinov (2015). "Generating images from captions with attention". *arXiv preprint arXiv:1511.02793*.

Miao, Y., L. Yu, and P. Blunsom (2016). "Neural variational inference for text processing". In: *International Conference on Machine Learning*. 1727–1736.

Mnih, A. and K. Gregor (2014). "Neural variational inference and learning in belief networks". In: *International Conference on Machine Learning*.

Mnih, A. and D. Rezende (2016). "Variational Inference for Monte Carlo Objectives". In: *International Conference on Machine Learning*. 2188–2196.

Mohamed, S. and D. J. Rezende (2015). "Variational information maximisation for intrinsically motivated reinforcement learning". In: *Advances in Neural Information Processing Systems*. 2125–2133.

Molchanov, D., A. Ashukha, and D. Vetrov (2017). "Variational dropout sparsifies deep neural networks". In: *International Conference on Machine Learning*. 2498–2507.

Naesseth, C., F. Ruiz, S. Linderman, and D. Blei (2017). "Reparameterization gradients through acceptance-rejection sampling algorithms". In: *Artificial Intelligence and Statistics*. 489–498.

Neal, R. (2011). "MCMC Using Hamiltonian Dynamics". *Handbook of Markov Chain Monte Carlo*: 113–162.

Neal, R. M. and G. E. Hinton (1998). "A view of the EM algorithm that justifies incremental, sparse, and other variants". In: *Learning in Graphical Models*. Springer. 355–368.

Paisley, J., D. Blei, and M. Jordan (2012). "Variational Bayesian Inference with stochastic search". In: *International Conference on Machine Learning*. 1367–1374.

Papamakarios, G., I. Murray, and T. Pavlakou (2017). "Masked autoregressive flow for density estimation". In: *Advances in Neural Information Processing Systems*. 2335–2344.

Pritzel, A., B. Uria, S. Srinivasan, A. P. Badia, O. Vinyals, D. Hassabis, D. Wierstra, and C. Blundell (2017). "Neural episodic control". In: *International Conference on Machine Learning*. 2827–2836.

Pu, Y., Z. Gan, R. Henao, X. Yuan, C. Li, A. Stevens, and L. Carin (2016). "Variational autoencoder for deep learning of images, labels and captions". In: *Advances in Neural Information Processing Systems*. 2352–2360.

Ranganath, R., S. Gerrish, and D. Blei (2014). "Black Box Variational Inference". In: *International Conference on Artificial Intelligence and Statistics*. 814–822.

Ranganath, R., D. Tran, and D. Blei (2016). "Hierarchical variational models". In: *International Conference on Machine Learning*. 324–333.

Ravanbakhsh, S., F. Lanusse, R. Mandelbaum, J. Schneider, and B. Poczos (2017). "Enabling dark energy science with deep generative models of galaxy images". In: *AAAI Conference on Artificial Intelligence*.

Rezende, D. J., S. Mohamed, I. Danihelka, K. Gregor, and D. Wierstra (2016a). "One-shot generalization in deep generative models". In: *International Conference on International Conference on Machine Learning*. 1521–1529.

Rezende, D. J., S. Mohamed, and D. Wierstra (2014). "Stochastic backpropagation and approximate inference in deep generative models". In: *International Conference on Machine Learning*. 1278–1286.

Rezende, D. J., S. A. Eslami, S. Mohamed, P. Battaglia, M. Jaderberg, and N. Heess (2016b). "Unsupervised learning of 3d structure from images". In: *Advances In Neural Information Processing Systems*. 4997–5005.

Rezende, D. and S. Mohamed (2015). "Variational inference with normalizing flows". In: *International Conference on Machine Learning*. 1530–1538.

Roeder, G., Y. Wu, and D. K. Duvenaud (2017). "Sticking the landing: Simple, lower-variance gradient estimators for variational inference". In: *Advances in Neural Information Processing Systems*. 6925–6934.

Rosca, M., B. Lakshminarayanan, and S. Mohamed (2018). "Distribution matching in variational inference". *arXiv preprint arXiv:1802.06847*.

Roweis, S. (1998). "EM algorithms for PCA and SPCA". *Advances in Neural Information Processing Systems*: 626–632.

Ruiz, F. R., M. T. R. AUEB, and D. Blei (2016). "The generalized reparameterization gradient". In: *Advances in Neural Information Processing Systems*. 460–468.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1988). "Learning representations by back-propagating errors". *Cognitive Modeling*. 5(3): 1.

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

Salakhutdinov, R. and H. Larochelle (2010). "Efficient learning of deep Boltzmann machines". In: *International Conference on Artificial Intelligence and Statistics*. 693–700.

Salimans, T. (2016). "A structured variational auto-encoder for learning deep hierarchies of sparse features". *arXiv preprint arXiv:1602.08734*.

Salimans, T., D. P. Kingma, and M. Welling (2015). "Markov Chain Monte Carlo and Variational Inference: Bridging the Gap." In: *International Conference on Machine Learning*. Vol. 37. 1218–1226.

Salimans, T. and D. A. Knowles (2013). "Fixed-Form variational posterior approximation through stochastic linear regression". *Bayesian Analysis*. 8(4).

Semeniuta, S., A. Severyn, and E. Barth (2017). "A hybrid convolutional variational autoencoder for text generation". *arXiv preprint arXiv:1702.02390*.

Serban, I. V., A. Sordoni, R. Lowe, L. Charlin, J. Pineau, A. Courville, and Y. Bengio (2017). "A hierarchical latent variable encoder-decoder model for generating dialogues". In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*. AAAI Press. 3295–3301.

Sohl-Dickstein, J., E. Weiss, N. Maheswaranathan, and S. Ganguli (2015). "Deep unsupervised learning using nonequilibrium thermodynamics". In: *International Conference on Machine Learning*. 2256–2265.

Sønderby, C. K., T. Raiko, L. Maaløe, S. K. Sønderby, and O. Winther (2016a). "How to train deep variational autoencoders and probabilistic ladder networks". In: *International Conference on Machine Learning*.

Sønderby, C. K., T. Raiko, L. Maaløe, S. K. Sønderby, and O. Winther (2016b). "Ladder variational autoencoders". In: *Advances in Neural Information Processing Systems*. 3738–3746.

Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich (2015). "Going deeper with convolutions". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1–9.

Tomczak, J. M. and M. Welling. "Improving variational auto-encoders using convex combination linear inverse autoregressive flow". In: *Benelearn 2017: Proceedings of the Twenty-Sixth Benelux Conference on Machine Learning, Technische Universiteit Eindhoven, 9-10 June 2017*. 162.

Tomczak, J. M. and M. Welling (2016). "Improving variational auto-encoders using householder flow". *arXiv preprint arXiv:1611.09630*.

Tran, D., M. D. Hoffman, R. A. Saurous, E. Brevdo, K. Murphy, and D. M. Blei (2017). "Deep probabilistic programming". *International Conference on Learning Representations*.

Tran, D., R. Ranganath, and D. M. Blei (2015). "The variational Gaussian process". *arXiv preprint arXiv:1511.06499*.

Van den Oord, A., N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, *et al.* (2016). "Conditional image generation with PixelCNN decoders". In: *Advances in neural information processing systems*. 4790–4798.

Van Oord, A., N. Kalchbrenner, and K. Kavukcuoglu (2016). "Pixel Recurrent Neural Networks". In: *International Conference on Machine Learning*. 1747–1756.

Vincent, P., H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol (2010). "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion". *Journal of Machine Learning Research*. 11(Dec): 3371–3408.

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

Watter, M., J. Springenberg, J. Boedecker, and M. Riedmiller (2015). "Embed to control: A locally linear latent dynamics model for control from raw images". In: *Advances in Neural Information Processing Systems*. 2746–2754.

White, T. (2016). "Sampling Generative Networks: Notes on a Few Effective Techniques". *arXiv preprint arXiv:1609.04468*.

Williams, R. J. (1992). "Simple statistical gradient-following algorithms for connectionist reinforcement learning". *Machine Learning*. 8(3-4): 229–256.

Wingate, D. and T. Weber (2013). "Automated variational inference in probabilistic programming". *arXiv preprint arXiv:1301.1299*.

Xu, W., H. Sun, C. Deng, and Y. Tan (2017). "Variational autoencoder for semi-supervised text classification". In: *AAAI*. 3358–3364.

Yan, X., J. Yang, K. Sohn, and H. Lee (2016). "Attribute2image: Conditional image generation from visual attributes". In: *European Conference on Computer Vision*. Springer. 776–791.

Yang, Z., Z. Hu, R. Salakhutdinov, and T. Berg-Kirkpatrick (2017). "Improved variational autoencoders for text modeling using dilated convolutions". In: *International Conference on Machine Learning*. 3881–3890.

Zhao, T., R. Zhao, and M. Eskenazi (2017). "Learning discourse-level diversity for neural dialog models using conditional variational autoencoders". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 654–664.