Variational Autoencoders

Robin Mittas
M. Sc. in Mathematics in Data Science
TUM School of Computation, Information and Technology
Technical University Munich
robin.mittas@gmx.de

Abstract

Variational Autoencoders (VAEs), first introduced by Kingma et al. [4] is an efficient variational inference method in the presence of intractable posterior distributions and continuous latent variables belonging to the family of generative models. In recent years VAEs have emerged as one of the most popular approaches to unsupervised learning of complicated distributions. [2] In this paper we will first look at all tools and common assumptions to define a VAE, such as the variational lower bound (ELBO), which approximates the intractable log-marginal-likelihood or how to sample from a gaussian posterior distribution with known mean and variance vectors such that we are still able to backpropagate through the network w.r.t. these vectors. Based on this we will discuss limitations and challenges and introduce some further approaches either containing multiple stochastic latent layers or transforming an initial simple random variable into a more complex one using inverse autoregressive transformations to build more flexible approximate posterior distributions.

1 Introduction

A standard Autoencoder consists of an encoder and a decoder. Let $d \in \mathbb{N}$ be the dimension of the input data and $n \in \mathbb{N}$ the dimension of the latent space, where usually n is chosen much smaller then d. The Encoder is defined as a function $f : \mathbb{R}^d \to \mathbb{R}^n$, thus mapping the input data $\boldsymbol{x} \in \mathbb{R}^d$ to a lower dimensional representation $\boldsymbol{z} \in \mathbb{R}^n$. The decoder is then defined as $g : \mathbb{R}^n \to \mathbb{R}^d$, thus mapping the latent representation back into the initial data space. Summarizing we can define the Autoencoder as the composition $g \circ f : \mathbb{R}^d \to \mathbb{R}^d$. The general goal of Autoencoders is to compress the data into a lower dimensional representation, the latent code, while keeping as much information as possible of the original input. Thus we aim to minimize the reconstruction loss between the input $\boldsymbol{x}$ and the output of the Decoder $g(f(\boldsymbol{x}))$.
The idea of Variational Autoencoders is to also be able to sample from the latent space, for example to generate new data. VAEs are nowadays used for many different use-cases, such as data generation, anomaly detection, dimensionality reduction or image segmentation. This paper will give an overview over Variational Autoencoders, techniques and new approaches improving the performance.

2 Setting and Methods

We assume that the data is generated by a random process involving an unobserved continuous random variable $\boldsymbol{z}$, which we will refer to as the latent variable. The data generation process consists of two steps, first a latent variable $\boldsymbol{z} \sim p_{\boldsymbol{\theta}}(\boldsymbol{z})$ is sampled from the prior distribution and secondly the data conditioned on $\boldsymbol{z}$ is generated as $\boldsymbol{x} \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$. The prior and the likelihood both come from a parametric family over distributions defined as $\mathcal{P} = \{p_{\boldsymbol{\theta}}(\boldsymbol{z})|\boldsymbol{\theta} \in \mathbb{R}^n\}$ and $\mathcal{Q} = \{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})|\boldsymbol{\theta} \in \mathbb{R}^d\}$. In general we consider a dataset $\boldsymbol{X} = \{\boldsymbol{x}^{(i)}\}_{i=1}^N$ of N i.i.d. samples of a discrete or continuous

variable $\boldsymbol{x}$, where N can be arbitrarily large. Very importantly, VAEs work in the case of intractable true posterior distributions $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$, which is equivalent to the intractability of the marginal-likelihood

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{z}) p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) d\boldsymbol{z}.$$

This is very often the case when working with Neural Networks.
Let us now define the probabilistic encoder $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ as an approximation to the true intractable posterior distribution $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$, since given a datapoint $\boldsymbol{x}$, it produces a distribution over the possible values of the latent random variable $\boldsymbol{z}$. Similarly we will define the probabilistic decoder $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ since given a code $\boldsymbol{z}$ it produces a distribution over the possible corresponding values of $\boldsymbol{x}$. We refer to $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ as parameters from the probabilistic encoder and decoder respectively.

2.1 The Variational Lower Bound - ELBO

Generally we aim to maximize the marginal likelihood of the dataset. Due to its intractability we need to find a lower bound. For that we recall that the Kullback-Leibler-divergence (KL diverence) for some probability distributions $p_{\boldsymbol{\lambda}}(\boldsymbol{y})$ and $p_{\boldsymbol{\tau}}(\boldsymbol{y})$ is defined as

$$\mathcal{D}_{KL}[p_{\boldsymbol{\theta}}(\boldsymbol{y}) \parallel p_{\boldsymbol{\tau}}(\boldsymbol{y})] = \int \log\Big(\frac{p_{\boldsymbol{\theta}}(\boldsymbol{y})}{p_{\boldsymbol{\tau}}(\boldsymbol{y})}\Big) p_{\boldsymbol{\theta}}(\boldsymbol{y}) d\boldsymbol{y} = \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{y})}\left[\log\Big(\frac{p_{\boldsymbol{\theta}}(\boldsymbol{y})}{p_{\boldsymbol{\tau}}(\boldsymbol{y})}\Big)\right]. \tag{1}$$

We will use the i.i.d. assumption of the data

$$p_{\boldsymbol{\theta}}(\boldsymbol{X}) = p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(N)}) = \sum_{i=1}^{N} p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})$$

and thus we will just consider the log marginal likelihood of a single sample $\boldsymbol{x} \in \boldsymbol{X}$

$$\begin{aligned}
\log(p_{\boldsymbol{\theta}}(\boldsymbol{x})) &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log(p_{\boldsymbol{\theta}}(\boldsymbol{x}))] \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log\Big(\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\Big)\right] \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log\Big(\frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\Big)\right] + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log\Big(\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\Big)\right] \\
&= \mathcal{D}_{KL}[p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) \parallel q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})] + \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}|\boldsymbol{x}),
\end{aligned}$$

where in the first step, taking the expectation, we used the fact that there is no dependency on $\boldsymbol{z}$ for the log marginal and in the second step we used the identity $p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z}) = p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) p_{\boldsymbol{\theta}}(\boldsymbol{x})$. [1] Now we can further rewrite the equation

$$\begin{aligned}
\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}|\boldsymbol{x}) &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[-\log(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})) + \log(p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z}))] \\
&= \int q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})[-\log(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})) + \log(p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})) + \log(p_{\boldsymbol{\theta}}(\boldsymbol{z}))] d\boldsymbol{z} \\
&= \int q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \log\Big(\frac{p_{\boldsymbol{\theta}}(\boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\Big) d\boldsymbol{z} + \int q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \log(p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})) d\boldsymbol{z} \\
&= -\mathcal{D}_{KL}[p_{\boldsymbol{\theta}}(\boldsymbol{z}) \parallel q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})] + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log(p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}))].
\end{aligned} \tag{2}$$

And finally using the fact that the Kullback-Leibler-divergence term is none-negative we end up with the lower bound on the log marginal likelihood

$$\log(p_{\boldsymbol{\theta}}(\boldsymbol{x})) \geqslant \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}|\boldsymbol{x}) = -\mathcal{D}_{KL}[p_{\boldsymbol{\theta}}(\boldsymbol{z}) \parallel q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})] + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log(p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}))], \tag{3}$$

which defines the ELBO. The first term is often referred to as the regularization term while the second one is referred to as the reconstruction term.

2.2 Sample from Latent Space and the Reparameterization Trick

Let us now assume that the posterior distribution follows a Gaussian distribution with a diagonal Covariance matrix. Given the latent dimension $n \in \mathbb{N}$, we define the parameters for the distribution $\boldsymbol{\mu}, \boldsymbol{\sigma} \in \mathbb{R}^n$, thus $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2 \boldsymbol{I})$, where $\boldsymbol{I} \in \mathbb{R}^{n \times n}$ denotes the identity matrix. Both vectors

are returned by the probabilistic Encoder. In order to be able to backpropagate through the network w.r.t. $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$, we sample $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ where $\mathbf{0} \in \mathbb{R}^n$. Then we reparametrize $\boldsymbol{z}$ as

$$\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon},$$

where $\odot$ refers to the element-wise multiplication, allowing us to backpropagate and optimize the parameters $\boldsymbol{\mu}, \boldsymbol{\sigma}$. Under certain conditions one can find such reparameterization function for none-gaussian distributions. [4] In the following we will restrict ourselves to the gaussian case.

3 Common Approaches

In order to to find the parameters of the probabilistic encoder and decoder for an n-dimensional latent space, we will now use a Neural Network. We assume that the prior follows an n-dimensional standard normal distribution $p_{\boldsymbol{\theta}}(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and we let the Likelihood $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ be a multivariate Gaussian in case of real-valued/ continuous data, or a Bernoulli in case of binary data, whose distribution parameters are computed with a Neural Network including none-linearities. Further we assume the true posterior takes on an approximate Gaussian form with an approximately diagonal covariance, which is equivalent to the latent dimensions not being linearly correlated with each other. Now we can let the variational approximate posterior $q_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ be a multivariate Gaussian with a diagonal covariance structure. Let the encoder output two vectors $\boldsymbol{\mu}, \boldsymbol{\sigma} \in \mathbb{R}^n$, for example by two linear layers. Then it holds for a single data point $\boldsymbol{x}^{(i)}$ [4]

$$\log(q_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})) = \log(\mathcal{N}(\boldsymbol{\mu}^{(i)}, (\boldsymbol{\sigma}^{(i)})^2)).$$

Now we sample from the approximate posterior $\boldsymbol{z}^{(i,l)} \sim q_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})$ using the reparameterization trick defined as the function $g_{\boldsymbol{\phi}}$

$$\boldsymbol{z}^{(i,l)} = g_{\boldsymbol{\phi}}(\boldsymbol{x}^{(i)}, \boldsymbol{\epsilon}^{(l)}) = \boldsymbol{\mu}^{(i)} + \boldsymbol{\sigma}^{(i)} \odot \boldsymbol{\epsilon}^{(l)}$$

where $\boldsymbol{\epsilon}^{(l)} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$.
The first term of the ELBO, defined in equation (3), refers to the KL-divergence between the prior, which is a multivariate standard Gaussian, and the posterior, a multivariate Gaussian. This term can be easily evaluated with the Appendix.
With the Monte Carlo estimates of expectations we can approximate the second term of the ELBO (3). Let f denote the decoding function and L the number of generated samples of $\boldsymbol{\epsilon}^{(\boldsymbol{l})} \sim p(\boldsymbol{\epsilon}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, then it holds

$$\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}[f(\boldsymbol{z})] = \mathbb{E}_{p(\boldsymbol{\epsilon})}[f(g_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}, \boldsymbol{x}^{(i)}))] \simeq \frac{1}{L} \sum_{l=1}^{L} f(g_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}^{(l)}, \boldsymbol{x}^{(i)})).$$

Now with the posterior $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})$ and the given reparametrization we can write the expectation as

$$\begin{aligned}
\mathbb{E}_{\mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}^{(i)}, (\boldsymbol{\sigma}^{(i)})^2)}[f(\boldsymbol{z})] &= \mathbb{E}_{\mathcal{N}(\boldsymbol{\epsilon}|\mathbf{0}, \boldsymbol{I})}[f(\boldsymbol{\mu}^{(i)} + \boldsymbol{\sigma}^{(i)} \odot \boldsymbol{\epsilon})] && (4)\\
&\simeq \frac{1}{L} \sum_{l=1}^{L} f(\boldsymbol{\mu}^{(i)} + \boldsymbol{\sigma}^{(i)} \odot \boldsymbol{\epsilon}^{(l)}) && (5)\\
&= \frac{1}{L} \sum_{l=1}^{L} \log(p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z}^{(i,l)})) && (6)
\end{aligned}$$

Finally, we can rewrite the ELBO for a single datapoint $\boldsymbol{x}^{(i)}$ as

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}^{(i)}) = \underbrace{\frac{1}{2} \sum_{j=1}^{n} \Big(1 + \log((\sigma_j^{(i)})^2) - (\mu_j^{(i)})^2 - (\sigma_j^{(i)})^2\Big)}_{-\mathcal{D}_{KL}[p_{\boldsymbol{\theta}}(z) \| q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})]} + \underbrace{\frac{1}{L} \sum_{l=1}^{L} \log(p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z}^{(i,l)}))}_{\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log(p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}))]},$$

where in practice we often choose $L = 1$.

3.1 Gaussian VAE

Assume that the prior $p_{\boldsymbol{\theta}}(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ is an n-dimensional Multivariate Standard Gaussian and let the likelihood $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) = \mathcal{N}(\hat{\boldsymbol{x}}, \boldsymbol{I})$ be a Gaussian, where $\hat{\boldsymbol{x}} \in \mathbb{R}^d$ is the output of the decoding function, with $\boldsymbol{I} \in \mathbb{R}^{n \times n}$ and $n \in \mathbb{N}$ the latent dimension. Then we can rewrite equation (6) with $L = 1$ and for one single datapoint $\boldsymbol{x} \in \mathbb{R}^d$ and a Constant $C \in \mathbb{R}$, where each pixel x_j is conditionally independent of the others given $\boldsymbol{z}$, as

$$
\begin{aligned}
-\log(p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})) &= -\log(\mathcal{N}(\hat{\boldsymbol{x}}, \boldsymbol{I})) = -\prod_{i=1}^{D} \log(\mathcal{N}(\hat{x}_i, 1)) && (7)\\
&= \sum_{i=1}^{D} \frac{1}{2}\log(2\pi) - \log(\exp(-\frac{1}{2}(x_i - \hat{x}_i)^2)) && (8)\\
&= \sum_{i=1}^{D} C + \frac{1}{2}(x_i - \hat{x}_i)^2 = DC + \frac{1}{2}\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|_2^2 && (9)\\
&= DC + \frac{1}{2}\,\mathrm{MSE}(\hat{\boldsymbol{x}}, \boldsymbol{x}). && (10)
\end{aligned}
$$

So we ended up with the MSE Loss as a Reconstruction term between $\boldsymbol{x}$ and $\hat{\boldsymbol{x}}$, the output of the VAE, now finally having all parts to easily calculate and optimize the ELBO. Note that if we assumed $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) = \text{Bernouli}(\boldsymbol{\lambda})$, e.g. for Binary Images, we would end up with the BCE Loss. In practice one chooses the losses that suits, BCE is also often used for none-Binary data. These terms are reduced by the sum over all dimensions - so for every dimension $i \in \{1, ..., d\}$ we have following reconstruction terms

$$
\begin{aligned}
\mathrm{MSE}(\hat{\boldsymbol{x}}, \boldsymbol{x})_i &= (\hat{x}_i - x_i)^2\\
\mathrm{BCE}(\hat{\boldsymbol{x}}, \boldsymbol{x})_i &= x_i \log(\hat{x}_i) + (1 - x_i)\log(1 - \hat{x}_i)
\end{aligned}
$$

The complete objective Loss function with the Standard Normal Prior is

$$
\begin{aligned}
\mathcal{L}(\hat{\boldsymbol{x}}, \boldsymbol{x}) &= \mathrm{Reconstruction}(\hat{\boldsymbol{x}}, \boldsymbol{x}) + \mathcal{D}_{KL}[q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z})]\\
&= \mathrm{Reconstruction}(\hat{\boldsymbol{x}}, \boldsymbol{x}) - 0.5 \sum_{i=0}^{n} (1 + \log(\sigma_i^2) - \mu_i^2 - \sigma_i^2).
\end{aligned}
$$

Potential limitations are that we assumed Unit Variance for the likelihood $p(\boldsymbol{x}|\boldsymbol{z})$ and that the loss might be dominated by either the KL divergence term or the reconstruction loss term.

3.2 β-VAE

In order to balance out both loss terms we introduce the weight $\beta \in \mathbb{R}$ for the KL-Divergence Term. The objective Loss function is then defined as

$$
\mathcal{L}_\beta(\hat{\boldsymbol{x}}, \boldsymbol{x}) = \mathrm{Reconstruction}(\hat{\boldsymbol{x}}, \boldsymbol{x}) + \beta \mathcal{D}_{KL}[q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z})].
$$

The goal is for some $\beta >> 1$ to have a disentangled representation, such that each latent dimension learns its own features as we force the posterior to be closer to the prior. [8] This can be seen in the section of experimental results on Figure 6.
Note that the weight β of the KLD Loss is a hyperparameter which needs to be tuned.

3.3 Calibrated Decoder: σ-VAE

For this type of VAE we assume that the likelihood is defined as $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{\mu}(\boldsymbol{z}), \boldsymbol{\sigma}(\boldsymbol{z})^2)$, with a given covariance matrix, instead of the Unit variance. This parameterization of the decoding distribution outputs one variance value for each dimension, e.g. for each pixel and channel in case of image data. While this of course is more powerful, it led to suboptimal performance and numerical instability. [13]. Instead the authors [13] proposed to use a simpler parameterization, in which the covariance matrix is specified with a single shared parameter $\sigma \in \mathbb{R}$. This parameter σ can either be

trainable or batch dependent. For the first case we would let the Neural Network learn the variance of the decoder as another parameter. [13] As in equation (10) we end up with the negative log-likelihood as

$$\begin{aligned} -\log(p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})) &= \frac{d}{2\sigma^2}\,\mathrm{MSE}(\hat{\boldsymbol{x}},\boldsymbol{x}) + d\log(\sigma 2\pi) \\ &= \frac{d}{2\sigma^2}\,\mathrm{MSE}(\hat{\boldsymbol{x}},\boldsymbol{x}) + d\log(\sigma) + C. \end{aligned}$$

The proposed optimal σ is calculated per batch, e.g. per training step. For a batch B of size $k \in \mathbb{N}$ with $B = \{\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(k)}\}$, the proposed optimal σ^* is defined as [13]

$$\sigma^* = \log\Big(\sqrt{\frac{1}{k}\sum_{i=1}^{k}(\mathrm{MSE}(\hat{\boldsymbol{x}}^{(i)}, \boldsymbol{x}^{(i)}, reduction = mean))}\Big).$$

The loss function given σ^* is

$$\mathcal{L}(\hat{\boldsymbol{x}},\boldsymbol{x}) = \frac{d}{2}\frac{\mathrm{MSE}(\hat{\boldsymbol{x}},\boldsymbol{x})}{exp((\sigma^*)^2)} + d\sigma^* + \mathcal{D}_{KL}[q(\boldsymbol{z}|\boldsymbol{x}) \parallel p(\boldsymbol{z})],$$

where d is the dimension of the input $\boldsymbol{x}$.
This effectively scales the original reconstruction loss term and we do not require the hyperparameter β to balance out the KL divergence and the reconstruction loss terms. Also we note that the parameter σ will not get too large due to the additional logarithmic penalty in the loss function. [13]

3.4 Experimental Results

This section will introduce own experimental results based on classic gaussian VAEs, β- and σ-VAEs, presented in the previous sections. Figure 1 shows the output of a trained decoder on the MNIST dataset for 8-dimensional standard normal random samples, e.g. for some $\boldsymbol{\epsilon} \in \mathbb{R}^8$ with $\epsilon_i \sim \mathcal{N}(0,1)$ for all $i \in \{1, ..., 8\}$ is a standard normal entry, while Figure 2 shows reconstructed input images from the test set. Figure 3 in turn depicts the decoded vectors for 2-dimensional vectors which lie within the L_∞ unit ball. On the right plot we can clearly see that the posterior $p_\theta(\boldsymbol{Z}|\boldsymbol{X})$ for the test set $\boldsymbol{X}$ is very close to the standard normal gaussian prior, as we try to minimize the KL-divergence term between them. With increasing latent dimension on the CelebA dataset we can see in the Figures 4-10 how the reconstruction get much better, but the random samples seem to be less realistic. An explanation of this might be, that some latent dimensions of the posterior are in fact correlated. This might happen if some latent dimensions e.g. control the hair region, and an other one the beard region, thus making specific combinations less probable. We can see that the latent dimension is an important hyperparameter which needs to be tuned and chosen carefully. It might depend on what we want to achieve with the VAE, e.g. drastically decreasing the dimension of the data to be able to further process it or to loose as less information as possible, which might be of crucial interest in some applications. In Figure 6 we can see the latent traversals of a trained β-VAE on the CelebA dataset. It shows that for example the first and last latent dimension, represented in the first and last row respectivley, control some part of the hair region, whereas the 5-th row controls the roation of the upper body. From here we can conclude that each latent dimension learns its own features controlling specific parts of the images.

Figure 1: Decoded standard normal random samples from Linear VAE. [9]

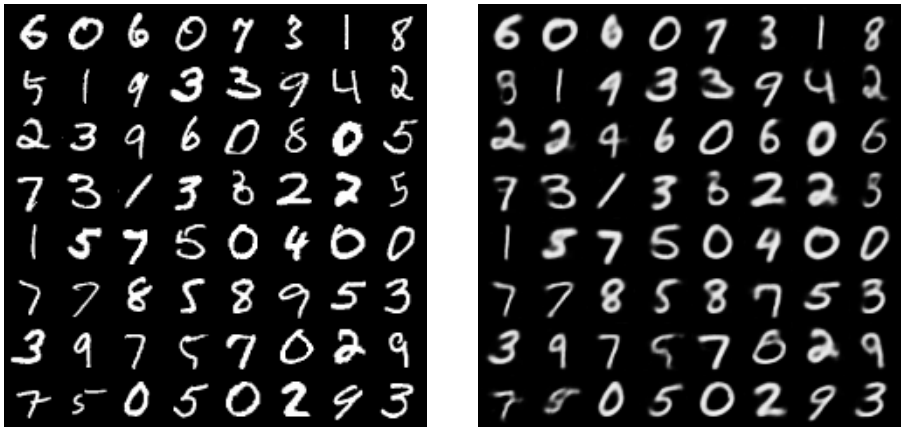

Figure 2: Reconstructions from Convolutional VAE with 8 latent dimensions. [9]

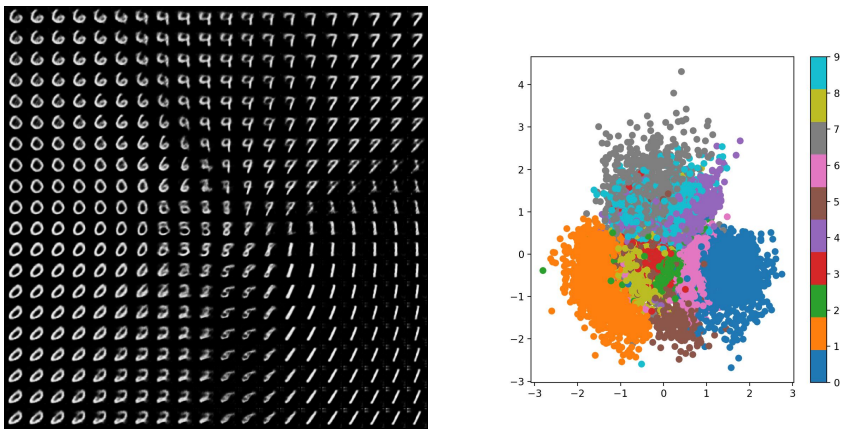

Figure 3: Left: Illustration of the Latent Space distribution for a σ-VAE with $2-$dimensional latent space. The image on the top right is a decoded Image of the vector $(-1, 1)$, the top right displays the decoded Image of the latent vector $(1, 1)$. Right: Class-dependent Latent Space Representation of MNIST Data. [9]

Figure 4: Samples from β-VAE with 10 latent dimensions. [9]

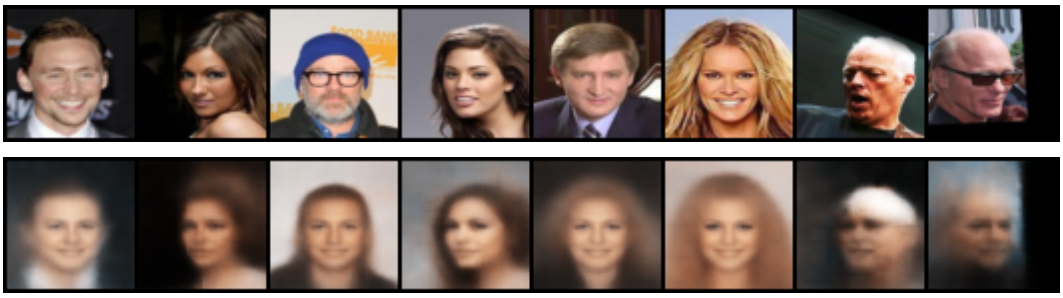

Figure 5: Reconstructions from β-VAE with 10 latent dimensions. [9]

Figure 6: Latent Traversals from β-VAE with 10 latent dimensions. Each image represents one decoded Vector $z \in \mathbb{R}^{10}$ where each entry is standard normal, $z_i \sim \mathcal{N}(0, 1)\ \forall i \in \{1, ...10\}$. Each row represents one latent dimension $i \in \{1, ...10\}$ when adding small deltas to one latent dimension at a time. [9] Ideas from [3]. (Best seen in Adobe Acrobat or at [9].)

Figure 7: Samples from β-VAE with 128 latent dimensions. [9]

Figure 8: Reconstructions from β-VAE with 128 latent dimensions. [9]

Figure 9: Samples from β-VAE with 256 latent dimensions.. [9]

Figure 10: Reconstructions from β-VAE with 256 latent dimensions. [9]

3.5 Interim Conclusion

So far we have seen some approaches which face some limitations. For most latent variable models the dimension of the latent space is a hyperparameter which needs to be chosen carefully. Wrong choices can lead to over-/ underfitting. Choosing some $n \in \mathbb{N}$ which is too small, the model will probably have poor reconstruction, while n chosen too large, will lead to little compression and thus resulting in a poor generalization, as seen in the previous section. The so far presented types of VAEs tend to generate blurry and unrealistic outputs. Also the KLD Loss Term and Reconstruction loss Term might be on different scales such that the Neural Network will just optimize the dominating loss term.
The Assumption that the variational posterior distribution $q_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ follows an isotropic Gaussian might be limiting. VAEs cannot guarantee that the inference algorithm will converge onto the standard Gaussian prior if the k latent neurons $\boldsymbol{z} = (z_1, ..., z_k)$ are in fact correlated. Also the assumption that prior is standard Gaussian might be restraining as we try to approximate the posterior to also be standard Gaussian, leading to potential limitations in the flexibility of the posterior approximation.
One further problem is the so called Posterior collapse. It occurs if each datapoint has the exact same value in one latent dimension, thus if there exists an $i \in \{1, ..., n\}$ s.t. for all $\boldsymbol{x}$ it holds that $q_{\boldsymbol{\phi}}(z_i|\boldsymbol{x}) \approx p(z_i)$. Here the variational distribution collapses towards the prior. It reduces the capacity of the generative model, resulting in not using the information of all latent dimensions.

4 Ladder-VAE

The idea of Ladder-VAE is to reduce the gap between the approximate and the true posterior by adding multiple stochastic layers where the parameters of the distribution of the generative model are shared with the inference model.
In the generative model $p_{\boldsymbol{\theta}}$ we split the latent variable into $L \in \mathbb{N}$ stochastic layers. [14] For a dataset $\boldsymbol{X}$ we train a generative model $p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) = p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1)p_{\boldsymbol{\theta}}(\boldsymbol{z})$, where we define the prior and the likelihood for $i = 1, ..., L$ as

$$
\begin{aligned}
p_{\boldsymbol{\theta}}(z) &= p_{\boldsymbol{\theta}}(\boldsymbol{z}_L) \prod_{i=1}^{L-1} p_{\boldsymbol{\theta}}(\boldsymbol{z}_i|\boldsymbol{z}_{i+1}) \\
p_{\boldsymbol{\theta}}(\boldsymbol{z}_i|\boldsymbol{z}_{i+1}) &= \mathcal{N}(\boldsymbol{z}_i|\boldsymbol{\mu}_{p,i}(\boldsymbol{z}_{i+1}), \boldsymbol{\sigma}^2_{p,i}(\boldsymbol{z}_{i+1})), \quad p_{\boldsymbol{\theta}}(\boldsymbol{z}_L) = \mathcal{N}(\boldsymbol{z}_L|\boldsymbol{0}, \boldsymbol{I}) \\
p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1) &= \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_{p,0}(\boldsymbol{z}_1), \boldsymbol{\sigma}^2_{p,0}(\boldsymbol{z}_1)),
\end{aligned}
$$

where $\boldsymbol{\mu}_{p,i}(\boldsymbol{z}_{i+1})$ and $\boldsymbol{\sigma}^2_{p,i}(\boldsymbol{z}_{i+1}))$ denote functions, e.g. linear layers with input $\boldsymbol{z}_{i+1}$. As before, in case of binary data we can also model $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1)$ as a Bernoulli distribution. For the generative model we use the subscript p, and for the inference model we use q. We note that the parameters of $p_{\boldsymbol{\theta}}(\boldsymbol{z}_i|\boldsymbol{z}_{i+1})$ for all $i \in \{1, ..., L-1\}$ are learned, and we just define the topmost stochastic layer, the prior, as a standard gaussian. The hierarchical specification allows the lower layers of the latent variables to be highly correlated while still maintaining the computational efficiency of fully factorized models. [14] The inference model is parameterized as a bottom-up process. Each layer is

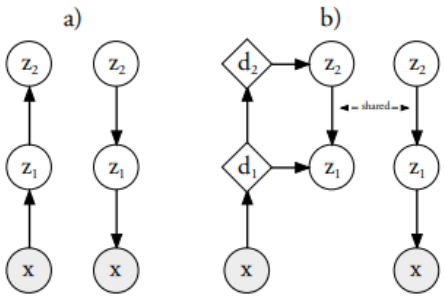

Figure 11: Inference and generative models for a) VAE with two stochastic layers and b) LVAE. Circles are stochastic variables and diamonds are deterministic variables. In LVAE we first go from x to d_L and z_L. Then we can have a none-linear mapping from z_L (standard normal prioi) to z_{L-1} (normal prior with mean and covariance matrix learned). [14]

defined as a fully factorized gaussian distribution [14]

$$
\begin{aligned}
q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) &= q_{\boldsymbol{\phi}}(\boldsymbol{z}_L|\boldsymbol{x}) \prod_{i=1}^{L-1} q_{\boldsymbol{\phi}}(\boldsymbol{z}_i|\boldsymbol{z}_{i+1}) \\
\boldsymbol{\sigma}_{q,i} &= \frac{1}{\hat{\boldsymbol{\sigma}}_{q,i}^{-2} + \boldsymbol{\sigma}_{p,i}^{-2}} \\
\boldsymbol{\mu}_{q,i} &= \frac{\hat{\boldsymbol{\mu}}_{q,i}\hat{\boldsymbol{\sigma}}_{q,i}^{-2} + \boldsymbol{\mu}_{p,i}\boldsymbol{\sigma}_{p,i}^{-2}}{\hat{\boldsymbol{\sigma}}_{\boldsymbol{q},\boldsymbol{i}}{}^{-2} + \boldsymbol{\sigma}_{p,i}^{-2}} \\
q_{\boldsymbol{\phi}}(\boldsymbol{z}_i|\cdot) &= \mathcal{N}(\boldsymbol{z}_i|\boldsymbol{\mu}_{q,i}, \boldsymbol{\sigma}_{q,i}^2).
\end{aligned}
$$

The inference model is a precision-weighted combination of $\hat{\boldsymbol{\mu}}_q$ and $\hat{\boldsymbol{\sigma}}_q^2$ carrying bottom-up information and $\boldsymbol{\mu}_p$ and $\boldsymbol{\sigma}_p^2$ from the generative distribution carrying top-down prior information. [14] By sharing the parameters of the generative distribution with the inference model, the inference model has information of the current state of the generative model in each layer and the „top down pass recursively corrects the generative distribution with the data dependent approximate log-likelihood“. [14]
The layers and the learned parameters can be defined for $i = 1, ..., L$ as [14]

$$
\begin{aligned}
\boldsymbol{d}_0 &= \boldsymbol{x} \\
\boldsymbol{d}_i &= \text{MLP}(\boldsymbol{d}_{i-1}) \\
\hat{\boldsymbol{\mu}}_{q,i} &= \text{Linear1}(\boldsymbol{d}_i) \\
\hat{\boldsymbol{\sigma}}_{q,i}^2 &= \text{Softplus}(\text{Linear2}(\boldsymbol{d}_i)).
\end{aligned}
$$

The structure of the Ladder-VAE is visualized in Figure 11. In practice one uses the ELBO as the objective loss function with the parameter β, similar to the case of β-VAEs, but in addition the authors propose to have a warm-up step for β to not get stuck at local minima or saddlepoints by linearly increasing β from 0 to 1 during the first N_t training epochs. [14] This trick is often seen in practice.

5 Improving Variational Inference with Inverse Autoregressive Flows

Inverse autoregressive flows (IAF) are used to build flexible approximate posterior distributions $q_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ through an iterative procedure. But in order to be able to efficiently optimize the ELBO we need the inference model to be computationally efficient to compute and to sample from, since we need to perform both operations for every single datapoint in each epoch. Also to make use of parallel computational resources in case of high dimensional latent representations $\boldsymbol{z}$, parallelizability of operations across dimensions of $\boldsymbol{z}$ is a large factor towards efficiency. This in fact restricts the class of approximate posteriors that are practical to use and mostly results in the use of simple gaussian

posteriors. However we also require the approximate posterior to be sufficiently flexible to match the true posterior distribution. [6]
One can think of the „simplest“ special case of an IAF where we transform a Gaussian variable with diagonal covariance to one with linear dependencies, thus the Covariance between latent dimension is not necessarily zero (see Appendix in [5]).

5.1 Normalizing Flows

The basic rule for transformations of densities considers an invertible, smooth mapping $f : \mathbb{R}^n \to \mathbb{R}^n$ with inverse $f^{-1} = g$, s.t. for the composition $g(f(\boldsymbol{z})) = \boldsymbol{x}$ holds. If we use this mapping to transform a random variable $\boldsymbol{z}$ with distribution $q(\boldsymbol{x})$, the resulting random variable $\boldsymbol{z}^* = f(\boldsymbol{z})$ has following distribution [12]

$$q(\boldsymbol{z}^*) = q(\boldsymbol{z}) \left| \det \frac{\partial f^{-1}}{\partial \boldsymbol{z}^*} \right| = q(\boldsymbol{z}) \left| \det \frac{\partial f}{\partial \boldsymbol{z}} \right|^{-1}.$$

Now with this rule we receive for $\boldsymbol{z}_0 = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$ and the Jacobian $\partial z_0 / \partial \boldsymbol{\epsilon} = \text{diag}(\sigma_1, ..., \sigma_n)$ for some $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$, $\boldsymbol{\mu}, \boldsymbol{\sigma} \in \mathbb{R}^n$ and $C \in \mathbb{R}$ the density

$$q_{\boldsymbol{\phi}}(\boldsymbol{z}_0|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{\epsilon}|\boldsymbol{0}, \boldsymbol{I}) \prod_{i=1}^{n} \sigma_i^{-1}$$
$$\log(q_{\boldsymbol{\phi}}(\boldsymbol{z}_0|\boldsymbol{x})) = C - \sum_{i=1}^{n} \frac{\epsilon_i^2}{2} + \sigma_i.$$

Normalizing flows start with an initial random variable $\boldsymbol{z}_0$ with a simple distribution with a known density function to then apply a chain of invertible parameterized transformations $f_t : \mathbb{R}^n \to \mathbb{R}^n$, where $t \in \mathbb{N}$ represents a time-step/ iteration, such that the last iterate $\boldsymbol{z}_T$ has a more flexible distribution. [12]

$$\boldsymbol{z}_0 \sim q(\boldsymbol{z}_0|\boldsymbol{x})$$
$$\boldsymbol{z}_t = f_t(\boldsymbol{z}_{t-1}, \boldsymbol{x}) \;\; \forall t = 1, ..., T$$

If the Jacobian determinant of each of the transformations f_t can be computed, the probability density function of the last iterate can still be computed with the basic rule for transformations of densities as [5]

$$\log q(\boldsymbol{z}_T|\boldsymbol{x}) = \log(q(\boldsymbol{z}_0|\boldsymbol{x})) - \sum_{t=1}^{T} \log \det \left| \frac{d\boldsymbol{z}_t}{d\boldsymbol{z}_{t-1}} \right|. \tag{11}$$

The authors of [12] used the simple transformation

$$f_t(\boldsymbol{z}_{t-1}) = \boldsymbol{z}_{t-1} + \boldsymbol{u} h(\boldsymbol{w}^T \boldsymbol{z}_{t-1} + b),$$

where $\boldsymbol{u}, \boldsymbol{w} \in \mathbb{R}^n$ are vectors, $b \in \mathbb{R}$ a scalar and $h(\cdot)$ a none-linearity. The second term can be interpreted as a MLP with a bottleneck hidden layer with a single unit. But to capture high-dimensional dependencies we require a long chain of transformations since the information passes the bottleneck with a single unit, thus this flow does not scale well to high-dimensional latent spaces. [5]

5.2 Inverse Autoregressive Transformations

Consider some autoregressive Autoencoders such as MADE or PixelCNN where the basic idea is that each pixel is dependent on the previously indexed pixel - which is an autoregressive structure, e.g. for $\boldsymbol{x} \in \mathbb{R}^{d \times d}$ a $d \times d$ image we have

$$p(\boldsymbol{x}) = \prod_{i=1}^{d^2} p(x_i|x_1, ..., x_{i-1}).$$

Assuming a specific ordering of the Pixels, e.g. row by row and pixel by pixel within every row, every conditional distribution in a PixelCNN is modelled by a convolutional neural networks. To satisfy the

requirement that the CNN can only use information about pixels above and to the left of the current pixel, the filters/ weights of the convolutional layer are masked. [11]
Now let the variable $\boldsymbol{y} \in \mathbb{R}^{d+1}$ be modelled by such an autoregressive model with a given ordering of its elements $\{y_i\}_{i=0}^{d}$. We use $[\boldsymbol{\mu}(\boldsymbol{y}), \boldsymbol{\sigma}(\boldsymbol{y})]$ to denote a function of the vector $\boldsymbol{y}$ to the vectors $\boldsymbol{\mu}, \boldsymbol{\sigma}$ and the elements $[\mu_i(\boldsymbol{y}_{0:i-1}), \sigma_i(\boldsymbol{y}_{0:i-1})]$ denote the predicted mean and variance of the i-th element of $\boldsymbol{y}$.
We further sample $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and define the first entry of $\boldsymbol{y}$ as $y_0 = \mu_0 + \sigma_0\epsilon_0$ and for $i > 0$ we get the autoregressive structure $y_i = \mu_i(y_{0:i-1}) + \sigma_i(y_{0:i-1})\epsilon_i$. Given the autoregressive structure of $\boldsymbol{y}$, the Jacobian matrix is a lower triangular matrix with 0 on the diagonal as $\partial[\mu_i, \sigma_i]/\partial y_j = 0$ for all $j \geq i$. This computation is clearly proportional to the dimension d and inefficient to compute for high-dimensional d since we need to sample from this distribution. [5]
However the inverse transformation is of interest as long as we have $\sigma_i > 0$ for all $i = \{0, ..., d\}$ as we then have a one-to-one transformation

$$\epsilon_i = \frac{y_i - \mu_i(y_{1:i-1})}{\sigma_i(y_{1:i-1})} = y_i s_i(y_{1:i-1}) + m_i(y_{1:i-1}), \tag{12}$$

where $s_i = 1/\sigma_i$ and $m_i = -\mu_i/\sigma_i$. Now one key observation is that the entries ϵ_i do not depend on each other and thus the transformation can be parallelized. An other important observation is that the inverse transformation has a simple Jacobian which again is a lower triangular matrix as $\partial\epsilon_i/\partial y_j = 0$ for $j > i$ and $\partial\epsilon_i/\partial y_i = 1/\sigma_i$. As for triangular matrices the determinant is just the product of its diagonal we receive [5]

$$\log\det\left|\frac{d\boldsymbol{\epsilon}}{d\boldsymbol{y}}\right| = \sum_{i=0}^{d} -\log(\sigma_i(\boldsymbol{y})).$$

5.3 Inverse Autoregressive Flow

A single IAF layer takes as input a point in the latent space and produces as output a transformed point in the latent space.

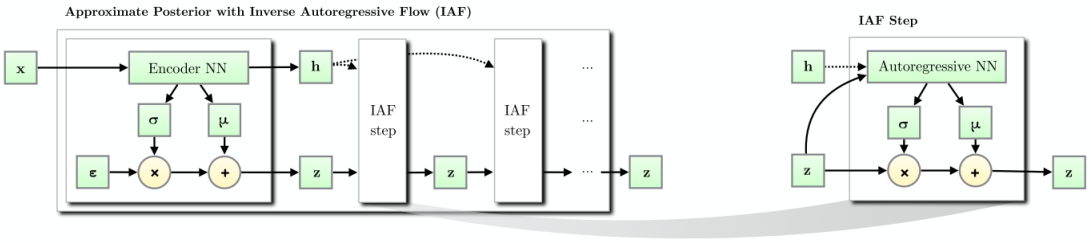

Figure 12: (Left) An Inverse Autoregressive Flow transforming the basic posterior of a variational encoder to a more complex posterior through multiple IAF transforms. (Right) A single IAF transform. [5]

In this section we will use the notation of equation (12) where we replace $[\boldsymbol{s}, \boldsymbol{m}]$ with $[\boldsymbol{\sigma}, \boldsymbol{\mu}]$ respectively, thus resulting in $\epsilon_i = \mu_i(y_{0:i-1}) + \sigma_i(y_{0:i-1})y_i$.
We let the encoder return two vectors $\boldsymbol{\mu}_0, \boldsymbol{\sigma}_0 \in \mathbb{R}^n$. We initialize the chain with a factorized gaussian $q_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}_0|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{\mu}_0, \text{diag}(\boldsymbol{\sigma}_0^2))$. As usual we first sample from a standard gaussian to then use the reparametrization trick to initialize the chain as

$$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$$
$$\boldsymbol{\epsilon}_0 = \boldsymbol{\mu}_0 + \boldsymbol{\sigma}_0 \odot \boldsymbol{\epsilon}.$$

The inverse autoregressive flow consists of a chain of T transformations where for every step t we use a different autoregressive neural network, which can be expressed for all $t \in \{1, ..., T\}$ as

$$(\boldsymbol{\mu}_t, \boldsymbol{\sigma}_t) = \text{AutoRegressiveNeuralNetwork}_t(\boldsymbol{\epsilon}_{t-1})$$
$$\boldsymbol{\epsilon}_t = \boldsymbol{\mu}_{t-1} + \boldsymbol{\sigma}_{t-1} \odot \boldsymbol{\epsilon}_{t-1}.$$

With equation (11) we can now compute the density of the final iterate for $\boldsymbol{z}_T = \boldsymbol{\epsilon}_T$ as

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

where ϵ_i corresponds to the i-th dimension of the initial standard normal noise vector. The expressivity of the autoregressive networks and the length of the chain increases the flexibility of the last iterate $\boldsymbol{z}_T$ and thus its ability to closely match to the true posterior distribution as illustrated in Figure 13 [5]

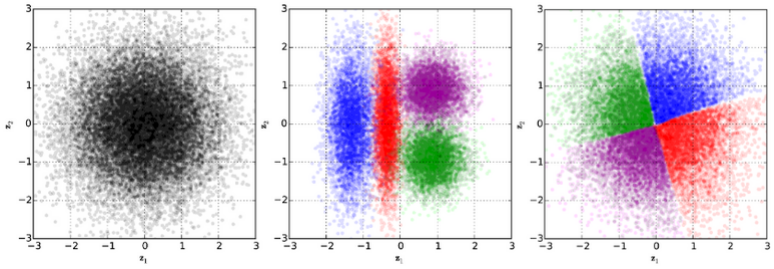

Figure 13: (Left) Prior Distribution (Middle) Approximate posterior in standard VAE (Right) Approximate posterior with IAF. IAF improves the flexibility of the posterior distributions, and allows for a better fit between the posteriors and the prior. [5]

6 State of the Art Results

To end this paper we want to give a short outlook into modern approaches as Bidirectional VAE (BIVA) [7] and Noveau VAE (NVAE) [10] as these models achieve state of the art test results. BIVA is a model formed by a deep hierarchy of stochastic variables. To enhance the flow of information and to avoid inactive units, it uses skip connections. The inference network is constructed to be bidirectional using stochastic variables in a bottom-up and a top-down inference path, which resembles the idea of Ladder VAE with the addition that the bottom-up pass is now also stochastic. The overall structure resulted in better test likelihoods. [7]
NVAE is related to VAEs with inverse autoregressive flows, as seen in chapter 5, as it borrows the statistical models from IAF-VAEs (see Appendix of [5]). But, it differs from IAF-VAEs in terms of neural network architecture, the parameterization of the approximate posteriors, and in terms of scaling up the training to large images. [15] The authors rather concentrated on improving the architecture of VAEs by carefully designing the architecture for hierarchical VAEs, instead of e.g. trying to reduce the gap between the approximate and the true posterior, formulating tighter bounds on the marginal likelihood or tackling posterior collapse. [15] In NVAE the latent variable is partitioned into L disjoint groups $\boldsymbol{z} = \{\boldsymbol{z_1}, ..., \boldsymbol{z_L}\}$ where the latent vectors can lie in different dimensions. The generative model then starts from a small spatially arranged latent variable, and then samples from the hierarchy group-by-group while gradually doubling the spatial dimensions. The model used to generate the samples shown in Figure 14 consists of 5 latent groups, starting with $\boldsymbol{z_1} \in \mathbb{R}^{8\times 8}$ and then doubling its way up to $\boldsymbol{z_5} \in \mathbb{R}^{128\times 128}$. Due to this multi-scale approach, NVAE is able to capture „global long-range correlations at the top of the hierarchy and local fine-grained dependencies at the lower groups“. [15] In contrast to normal VAEs which tend to often generate blurry outputs/ images, NVAE is able to generate very sharp high-resolution samples as depicted in Figure 14.

Figure 14: Decoded random samples from the latent space. [10]

References

[1] user3658307 (https://math.stackexchange.com/users/346641/user3658307). *marginal likelihood in variational bayes*. Mathematics Stack Exchange. URL:https://math.stackexchange.com/q/3341523 (version: 2022-09-13). eprint: https://math.stackexchange.com/q/3341523. URL: https://math.stackexchange.com/q/3341523.

[2] Carl Doersch. *Tutorial on Variational Autoencoders*. 2021. arXiv: 1606.05908 [stat.ML].

[3] Irina Higgins et al. "beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework". In: *International Conference on Learning Representations*. 2017. URL: https://openreview.net/forum?id=Sy2fzU9gl.

[4] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2013. DOI: 10.48550/ARXIV.1312.6114. URL: https://arxiv.org/abs/1312.6114.

[5] Diederik P. Kingma, Tim Salimans, and Max Welling. "Improving Variational Inference with Inverse Autoregressive Flow". In: *CoRR* abs/1606.04934 (2016). arXiv: 1606.04934. URL: http://arxiv.org/abs/1606.04934.

[6] Diederik P. Kingma and Max Welling. "An Introduction to Variational Autoencoders". In: *CoRR* abs/1906.02691 (2019). arXiv: 1906.02691. URL: http://arxiv.org/abs/1906.02691.

[7] Lars Maaløe et al. *BIVA: A Very Deep Hierarchy of Latent Variables for Generative Modeling*. 2019. DOI: 10.48550/ARXIV.1902.02102. URL: https://arxiv.org/abs/1902.02102.

[8] Emile Mathieu et al. *Disentangling Disentanglement in Variational Autoencoders*. 2018. DOI: 10.48550/ARXIV.1812.02833. URL: https://arxiv.org/abs/1812.02833.

[9] Robin Mittas. *variational-autoencoder*. https://github.com/robinmittas/variational-autoencoder. 2023. URL: https://github.com/robinmittas/variational-autoencoder.

[10] NVLABS. *NVAE*. https://github.com/NVlabs/NVAE. 2021.

[11] Aäron van den Oord et al. "Conditional Image Generation with PixelCNN Decoders". In: *CoRR* abs/1606.05328 (2016). arXiv: 1606.05328. URL: http://arxiv.org/abs/1606.05328.

[12] Danilo Jimenez Rezende and Shakir Mohamed. *Variational Inference with Normalizing Flows*. 2015. DOI: 10.48550/ARXIV.1505.05770. URL: https://arxiv.org/abs/1505.05770.

[13] Oleh Rybkin, Kostas Daniilidis, and Sergey Levine. "Simple and Effective VAE Training with Calibrated Decoders". In: *CoRR* abs/2006.13202 (2020). arXiv: 2006.13202. URL: https://arxiv.org/abs/2006.13202.

[14] Casper Kaae Sønderby et al. *Ladder Variational Autoencoders*. 2016. DOI: 10.48550/ARXIV.1602.02282. URL: https://arxiv.org/abs/1602.02282.

[15] Arash Vahdat and Jan Kautz. *NVAE: A Deep Hierarchical Variational Autoencoder*. 2020. DOI: 10.48550/ARXIV.2007.03898. URL: https://arxiv.org/abs/2007.03898.

A Appendix

A.1 KL divergence of two Gaussians

Let $n \in \mathbb{N}$ be the dimension of the latent space. Assume that the prior $p_{\boldsymbol{\theta}}(\boldsymbol{z}) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and the posterior approximation $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})$ are Gaussian, with $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ as the variational mean and standard deviation evaluated at the datapoint x, and let μ_i and σ_i simply denote the i-th element of these vectors. Then with the definition of the KL divergence term given in Equation (1) we split the $\log$ into two parts and calculate the following two integrals.

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

For the second integral we get:

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

Putting now both together, it follows

$$-\mathcal{D}_{KL}[q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z})] = \frac{1}{2}\sum_{j=1}^{n}(1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2)$$

B Notations

Table 1: Notations

$n \in \mathbb{N}$	Latent dimension
$d \in \mathbb{N}$	Data dimension
$\boldsymbol{X} \in \mathbb{R}^{N \times d}$	The data matrix containing all N datapoints
$\boldsymbol{x} \in \mathbb{R}^d$	With bold symbols we notate vectors
$x_i \in \mathbb{R}$	None bold symbols with indices indicate the i-th element of the vector $\boldsymbol{x}$
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$	Normal distribution with a diagonal Covariance matrix with σ_i on the diagonal