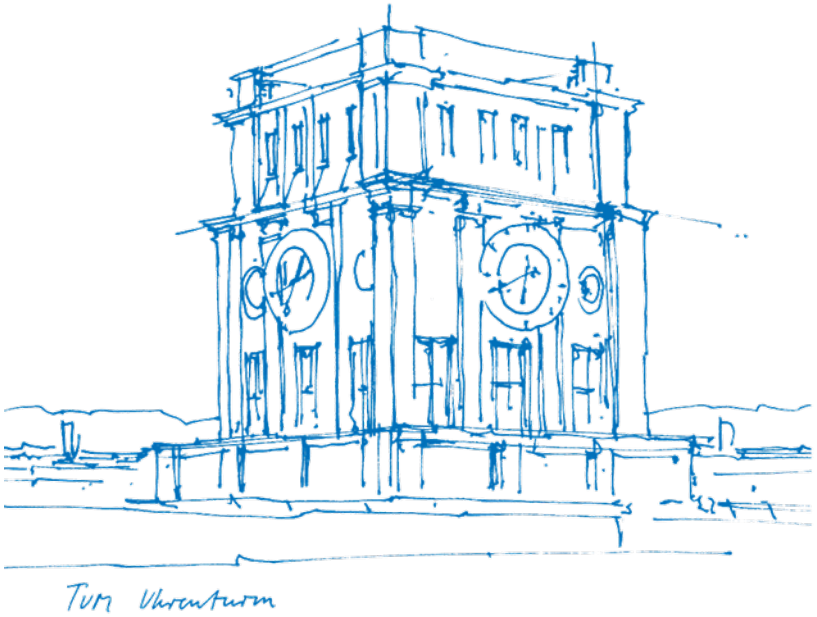
Variational Autoencoders

Robin Mittas
Technische Universität München
Computation, Information and Technology (MA)
Applied Mathematics
Munich, 25th of January 2023



Agenda

1. Motivation
2. Small Recap
3. Variational Bayesian
4. The Variational Bound
5. Gaussian VAE
6. β-VAE
7. σ-VAE
8. Own Coding Work
9. Limitations and Challenges
10. State of the Art
11. Appendix

Motivation

VAEs are used for:

- Anomaly Detection.
- Generating new Data.
- Dimensionality Reduction: For huge Datasets we can train Autoencoder and use encoded Data for further Processes. Often used in Finance or Bio-Processing
- ML and MAP estimation.
- Useful for data representation tasks. Useful in in computer vision applications such as image denoising, inpainting and super-resolution.
- De-Noising
- Predicting future frames for a video sequence.
- Image Segmentation.
- Music, Speech Generation ...

Recap: Dimensionality Reduction

PCA: Principal Component Analysis

Suppose we have a Data Matrix $A \in \mathbb{R}^{m \times n}$, then the k-th Principal Component is

$$v_k = \underset{\|v\|_2=1 \& v \perp v_1,\ldots,v_{k-1}}{\operatorname{argmax}} \|Av\|_2.$$

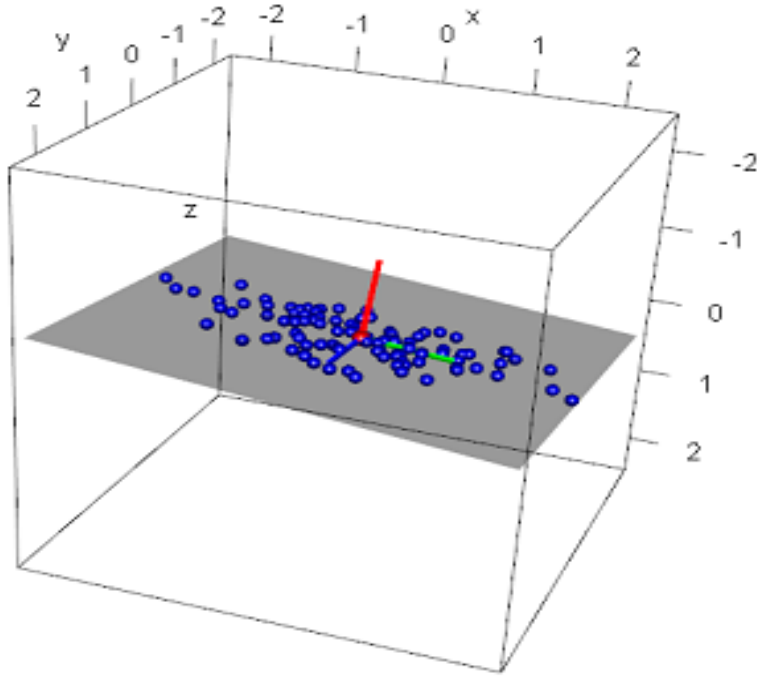


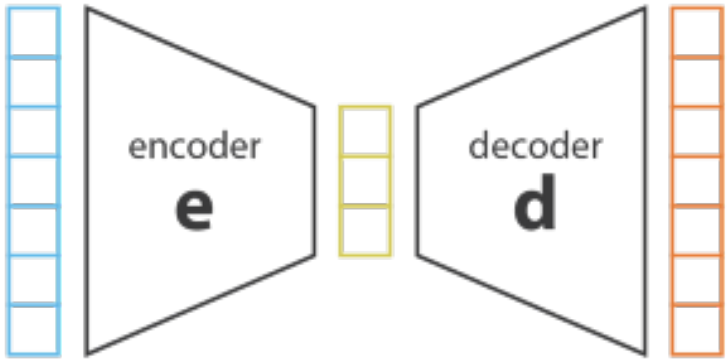
Figure: v_1, v_2, v_3, $span(v_1, v_2)$ [1]

Autoencoder: Encoder-Decoder Structure

Encoder = Dimensionality Reduction with latent space dimension $n \in \mathbb{N}$: Returns latent vector $x \in \mathbb{R}^n$. This is done by e.g. Concolutional or Linear layers.

Decoder = Decode Encoded Data back into initial Space. This is done by Transposed Convolutions are Linear mapping into higher dimensions.

Question: What happens if we plug a random vector into the Decoder?



Variational Autoencoder

Encoder returns two vectors for some latent dimension $n \in \mathbb{N}$: Mean $\mu \in \mathbb{R}^n$ and Variance $\sigma \in \mathbb{R}^n$. Afterwards we sample a vector $z \sim \mathcal{N}(\mu, \sigma)$.

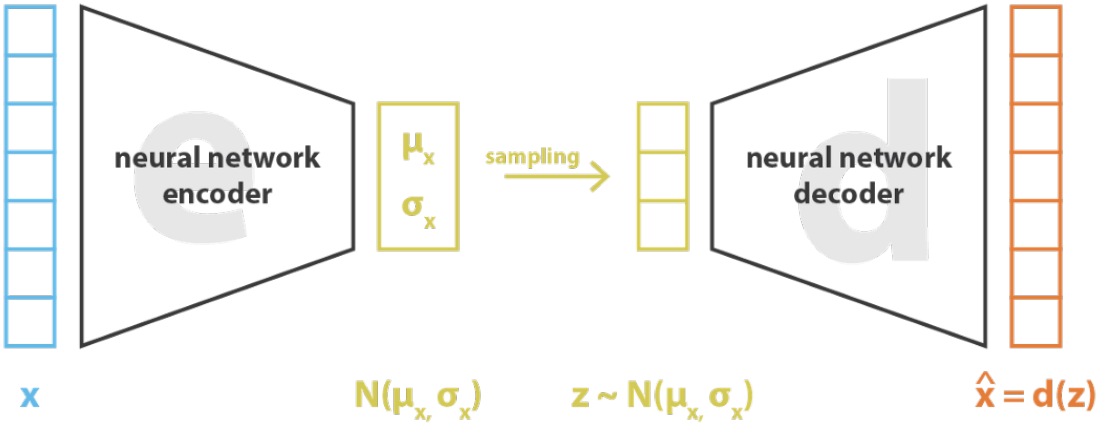


Figure: Variational Autoencoder Structure [8]

Variational Bayesian

Setting:

- Dataset $X = \{x_i\}_{i=1}^N$ of N i.i.d. samples of some continuous or discrete variable x. Assume we have given a large dataset, e.g. we make parameter updates using small minibatches or even single datapoints. [5]
- Data generated by random process, involving an unobserved continuous random variable z (the latent variable). [5]
- A Value $z^{(i)}$ is generated from the prior distribution $p_{\theta^*}(z)$ and $x^{(i)}$ is generated from conditional distribution $p_{\theta^*}(x|z)$ (likelihood) where θ^* are the true parameters. The prior and the likelihood come from parametric families of distributions $p_\theta(z)$ and $p_\theta(x|z)$. The prior represents our beliefs before we observe any data and in Variational Autoencoder Setting it is mostly assumed to be standard normal. [5]
- Intractability: The Integral of the marginal likelihood

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

is intractable. This is equivalent to the intractability of the true posterior $p_\theta(z|x) = p_\theta(z)p_\theta(x|z)/p_\theta(x)$. This is very common in Neural Networks due to None-Linearities. [5]

Recognition Model

Define Recognition model $q_\phi(z|x)$ as an approximation of the intractable true posterior $p_\theta(z|x)$, which encodes our beliefs after observing data. [5]

Refer to recognition model $q_\phi(z|x)$ as a **probabilistic encoder**, since given a datapoint x it produces a distribution (e.g. a Gaussian) over the possible values of the latent random variable z (or: „code“) from which the datapoint x could have been generated.

Similarly we will refer to $p_\theta(x|z)$ as a **probabilistic decoder**, since given a code z it produces a distribution over the possible corresponding values of x.

We refer to ϕ and θ as the parameters from the probabilistic encoder and decoder respectively. [5]

The Variational Bound

The ELBO is defined as lower bound for the Log-marginal likelihood and satisfies following inequality (see Appendix for derivation)

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

In general we want to maximize the (log-) marginal likelihood (integrate out all model parameters). It represents the probability of generating the observed sample from a prior and is therefore often referred to as model evidence.
Now think of x as an observed data point. The right-hand side consists of two terms, both involve taking a sample $z \sim q_\phi(z|x)$, which we can interpret as a code describing x or as a latent representation of x. The second term $\log(p_\theta(x|z))$ is the log-likelihood of the observed x given the code z that we have sampled. This term is maximized when $p_\theta(x|z)$ assigns high probability to the original x. It is trying to reconstruct x given the code z (proabilistic Decoder) and the term is called the reconstruction error.
The first term is the divergence between $q_\phi(z|x)$ and the prior $p_\theta(z)$, which we will fix to be a unit Normal in most cases. It encourages the codes z to look Gaussian and prevents the model to learn an identity mapping. We call it the regularization term. [5]

Sample from Latent Space, Re-parametrization

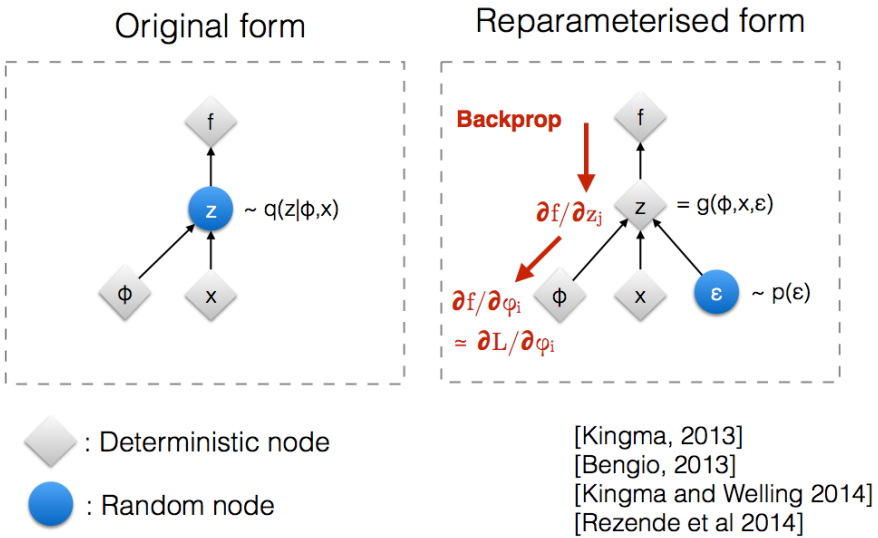


Figure: Illustration of the reparamtrization trick together with the gradient flow in order to be able to backpropagate through the Network [3]

Sample from Latent Space, Re-parametrization

In other words: The Encoder returns two vectors $\mu, \sigma \in \mathbb{R}^n$ with latent dimension $n \in \mathbb{N}$. Imagine μ to be the mean vector and σ the variance vector, e.g. the parameters for the Normal Distribution. We then sample $\varepsilon \sim \mathcal{N}(0, I)$ where $0 \in \mathbb{R}^n$ and $I \in \mathbb{R}^{n \times n}$ the identity matrix.
Then reparametrize z as

$$z = \mu + \sigma \odot \varepsilon,$$

where $\odot$ refers to the element-wise multiplication. We can now easily backpropagate through the Network.

SGVB: Stochastic Gradient Variational Bayesian

Bayesian VAE

In this section we'll give an example where we use a neural network for the probabilistic encoder for an n-dimensional latent space. Now we use a Neural Network to find the parameters of the probabilistic encoder/ the approximation to the posterior $q_\theta(z|x)$. The model parameters θ, ϕ are jointly optimized with the AEVB algorithm. [5]
We assume the prior $p_\theta(z) = \mathcal{N}(z; 0, I)$ to be a centered multivariate n-dimensional standard Gaussian. We let $p_\theta(x|z)$ be a multivariate Gaussian (in case of real-valued data) or Bernoulli (in case of binary data) whose distribution parameters are computed from z with a Neural Network including None-Linearities. Note the true posterior $p_\theta(z|x) = p_\theta(z)p_\theta(x|z)/p_\theta(x)$ is in this case intractable. While there is much freedom in the form $q_\theta(z|x)$, we assume the true posterior takes on an approximate Gaussian form with an approximately diagonal covariance. Thus the latent dimensions are not correlated with each other. Now we can let the variational approximate posterior be a multivariate Gaussian with a diagonal covariance structure. Let the encoder further output two vectors $\sigma \in \mathbb{R}^n$ and $\mu \in \mathbb{R}^n$ by e.g. Linear Layers. Then it holds for a single data point $x^{(i)}$ [5]

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

Bayesian VAE

Now we sample from the posterior $z^{(i,l)} \sim q_\theta(z|x^{(i)})$ using

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

where $\varepsilon^{(l)} \sim \mathcal{N}(0, I)$. Then with the Appendix the KL Divergence term between the prior (multivariate isotropic standard gaussian) and the posterior (multivariate isotropic gaussian) can be computed and the Loss function for the datapoint $x^{(i)}$ is defined as

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

where L corresponds to the Number of Samples of $\varepsilon^{(l)}$ to calculate the Expectation via Monte Carlo estimates

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

where $\varepsilon^{(l)} \sim p(\varepsilon) = \mathcal{N}(0, I)$ in our case, g_ϕ the reparametrization function and f the decoding function. In General we use $L = 1$ and $n \in \mathbb{N}$ corresponds to our latent space dimension.

Bayesian VAE

Now with our posterior $p(z|x) = \mathcal{N}(\mu, \sigma^2)$ and the given reparametrization $z = \mu + \varepsilon \odot \sigma$ with $\varepsilon \sim \mathcal{N}(0, I)$ we can write the Expectation as

$$\mathbb{E}_{\mathcal{N}(z|\mu,\sigma^2)}[f(z)] = \mathbb{E}_{\mathcal{N}(\varepsilon 0,I)}[f(\mu + \varepsilon \odot \sigma)] \tag{1}$$

$$\simeq \frac{1}{L} \sum_{l=1}^{L} f(\mu + \varepsilon^{(l)} \odot \sigma) \tag{2}$$

$$= \frac{1}{L} \sum_{l=1}^{L} \log(p_\theta(x|z^{(l)})) \tag{3}$$

Classic-Gaussian-VAE

In general one assumes that the prior $p_\theta(z)$ is an n-dimensional Multivariate Standard Gaussian, meaning $z \sim \mathcal{N}(0, I)$ with $I \in \mathbb{R}^{n \times n}$ and $n \in \mathbb{N}$ the latent dimension, as the KLD Term of two Gaussians can be easily calculated as explained in the Appendix. Also we assume that $p_\theta(x|z) \sim \mathcal{N}(\hat{x}, I)$, where $\hat{x} \in \mathbb{R}^D$ is the output of the decoding function. We assume unit variance as we can then rewrite equation (3) with $L = 1$ and one single datapoint $x \in \mathbb{R}^D$ where each pixel x_j is conditionally independent of the others given z (but they become dependent if we marginalize out z)

$$\log(p_\theta(x|z)) = \log(\mathcal{N}(\mu, I)) = \prod_{i=1}^{D} \log(\mathcal{N}(\mu_i, 1)) \tag{4}$$

$$= \sum_{i=1}^{D} -\frac{1}{2}\log(2\pi) + \log(\exp(0.5(x_i - \hat{x}_i)^2)) \tag{5}$$

$$= \sum_{i=1}^{D} C + \frac{1}{2}(x_i - \hat{x}_i)^2 \tag{6}$$

$$= DC + \frac{1}{2} \parallel x - \hat{x} \parallel_2^2 \tag{7}$$

$$= DC + \frac{1}{2}\mathrm{MSE}(\hat{x}, x) \tag{8}$$

Classic-Gaussian-VAE

So we now ended up with the MSE Loss as our Reconstruction term between $\hat{x}$, the output of our VAE with a given input x! Note that if we had assume $p_\theta(x|z) \sim \text{Bernouli}(\lambda)$ we would end up with the BCE Loss. In Practice we thus mostly use MSE and BCE Loss as Reconstruction Error Term. These terms are then reduced by the sum over all Pixels (for image data).

$$\text{MSE}(\hat{x}, x)_n = (\hat{x}_n - x_n)^2$$
$$\text{BCE}(\hat{x}, x)_n = \hat{x}_n \log(x_n) + (1 - \hat{x}_n) \log(1 - x_n)$$

The complete objective Loss function with the Standard Normal Prior is

$$\begin{aligned}\mathcal{L} &= \text{Reconstruction}(\hat{x}, x) - \mathcal{D}_{KL}[q(z|x) \parallel p(z)] \\ &= \text{Reconstruction}(\hat{x}, x) - 0.5(1 + \log(\sigma) - \mu^2 - \sigma)\end{aligned}$$

Problem: We assumed Unit Variance for $p(x|z)$ and the loss might be dominated by either KLD Loss term or reconstruction loss term. In practice one chooses the losses that suits, meaning reducing MSE or BCE Loss by summing over the deviances or taking the mean over the deviances.

β-VAE/ Disentangled VAE

Introduction of weight $\beta \in \mathbb{R}$ for KL-Divergence Term. The objective Loss function is

$$\mathcal{L}_\beta(\hat{x}, x) = \text{Reconstruction}(\hat{x}, x) - \beta \mathcal{D}_{KL}[q(z|x) \,||\, p(z)].$$

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. [6]

Problem: The weight β of the KLD Loss is a hyperparameter which needs to be tuned.

β-VAE/ Disentangled VAE

Figure: Own trained β-VAE on CelebA Dataset with Latent dimension 12. 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. Ideas from [**?**]

Calibrated Decoder: σ-VAE

Let us now not assume Unit Variance, instead let $p_\theta(x|z) \sim \mathcal{N}(\mu(z), \sigma(z)^2)$. This parameterization of the decoding distribution outputs one variance value per each pixel and channel. While powerful, it has been observed, that this approach attains suboptimal performance, and is moreover prone to numerical instability. [9].

Instead, it has been found experimentally that a simpler parameterization, in which the covariance matrix is specified with a single shared parameter $\sigma \in \mathbb{R}$ has worked better in practice. This σ can either be a trainable parameter or batch dependent. For the first case the Neural Network learns the variance of the decoder as another parameter. [9]

Following the same calculation steps as in (8) we end up with

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

Calibrated Decoder: σ-VAE

The proposed optimal σ (often referred to as log-σ is calculataed per batch (e.g. per training step). Assume we have a batch B of size $k \in \mathbb{N}$ with $B = \{x_1, ..., x_k\}$, then we define

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

The Loss function would then look like for the specified σ^*

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

where D is the dimension of the input x. This result effectively scales the original reconstruction loss term and we do not require the hyperparameter β to balance out the KLD and the reconstruction loss terms. The parameter σ will not get too large due to the additional logarithmic Penalty in Loss Function. [9]

Own Coding Work I

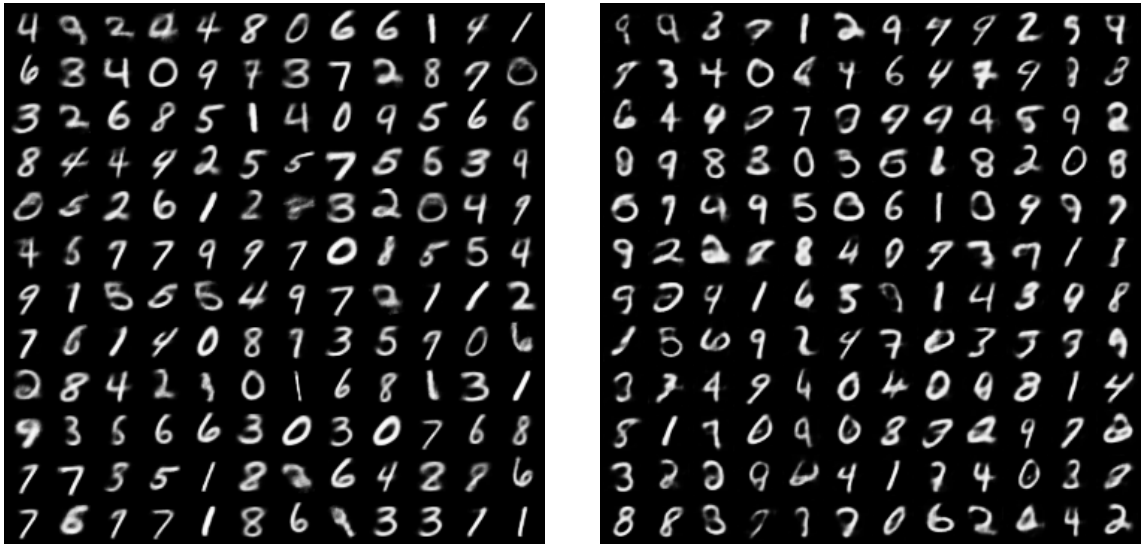


Figure: Samples from Linear VAE on the left and Convolutional VAE on the right with Latent Dimension 8 on MNIST Dataset.

Own Coding Work II

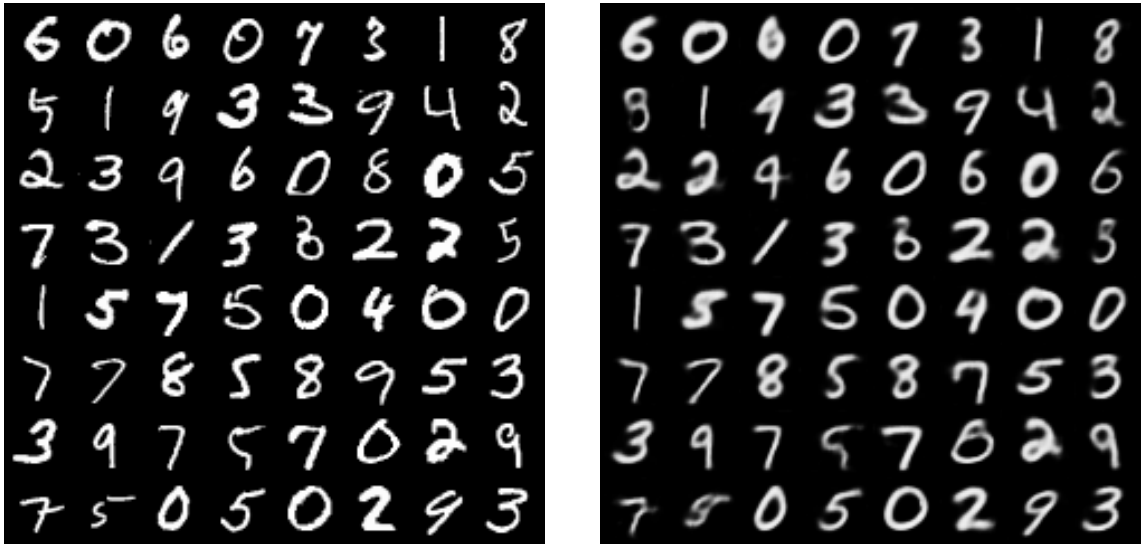


Figure: Reconstructions from Convolutional VAE with 8 latent dimensions.

Own Coding Work III: Latent Space Representation

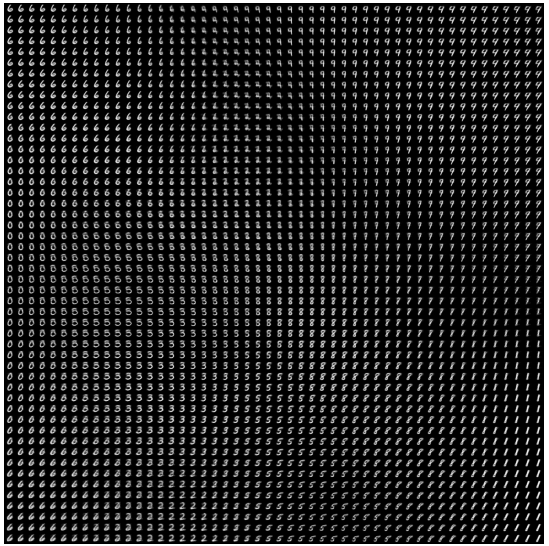


Figure: The following image illustrates the Latent Space distribution for a σ-VAE with $2-$dimensional latent space. The image on the top right is a decoded Image of the vector $(-1,1)$, the top right displays the decoded Image of the latent vector $(1,1)$.

Own Coding Work IV

Figure: Samples from β-VAE with 12 latent dimensions.

Own Coding Work V

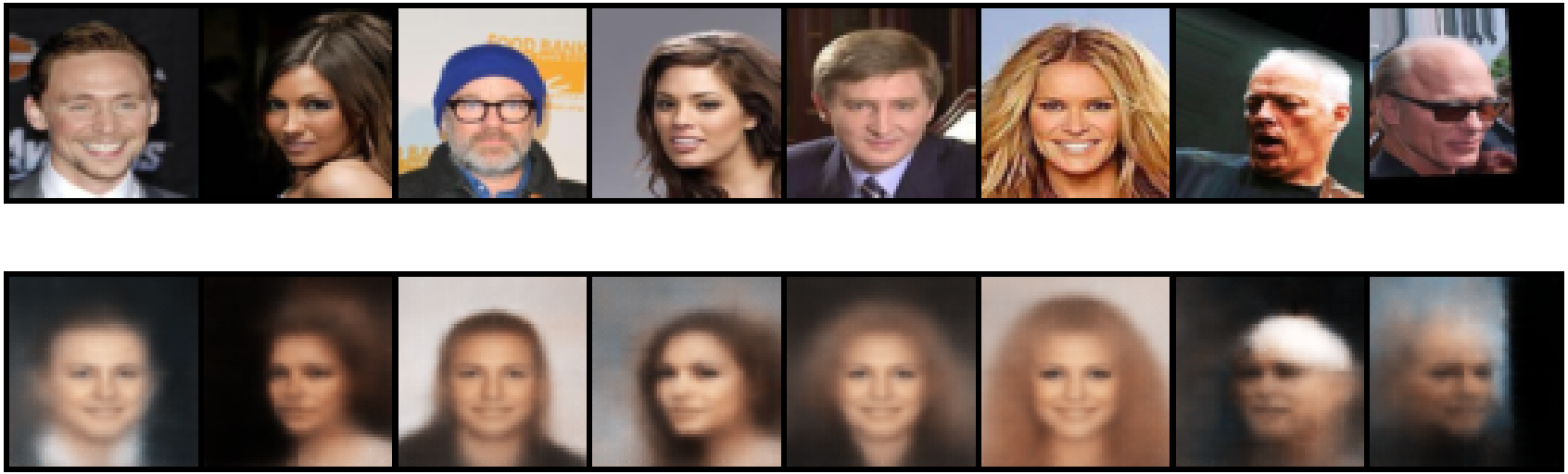


Figure: Reconstructions from β-VAE with 12 latent dimensions.

Own Coding Work VI: β-VAE/ Disentangled VAE

Figure: Latent Traversals from β-VAE with 12 latent dimensions.

Own Coding Work VII

Figure: Samples from β-VAE with 128 latent dimensions.

Own Coding Work VIII

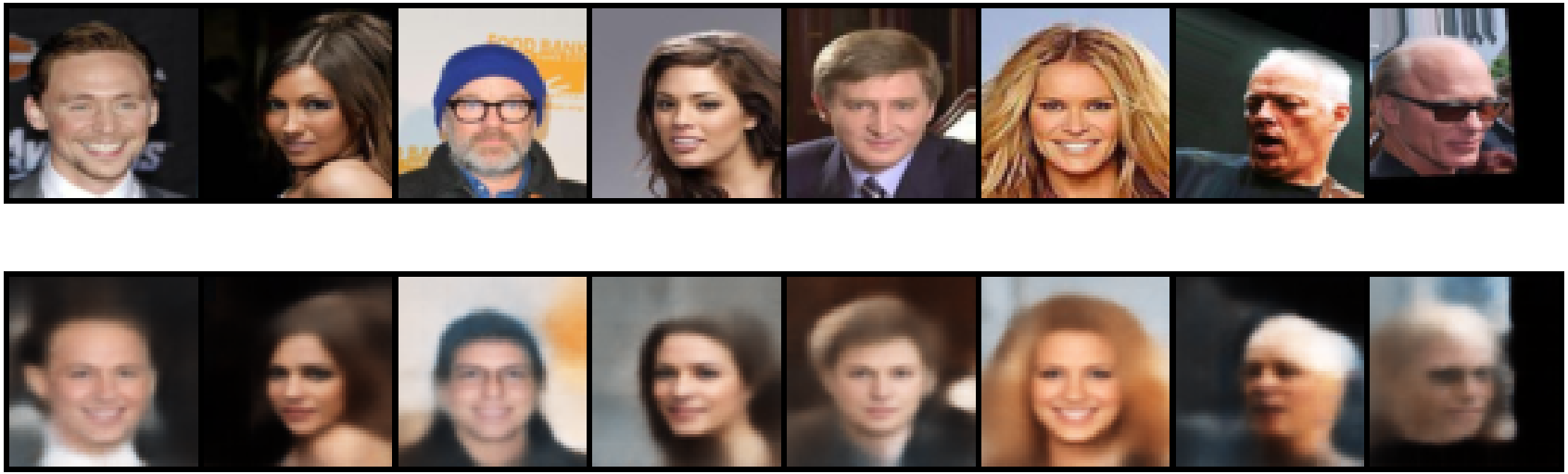


Figure: Reconstructions from β-VAE with 128 latent dimensions.

Own Coding Work IX: β-VAE/ Disentangled VAE

Figure: Latent Traversals from β-VAE with 128 latent dimensions.

Own Coding Work X: Question

What was the approximate latent dimension here? Also have a look at the next Slide!

Figure: Samples

Own Coding Work XI: Question

What was the approximate latent dimension here?

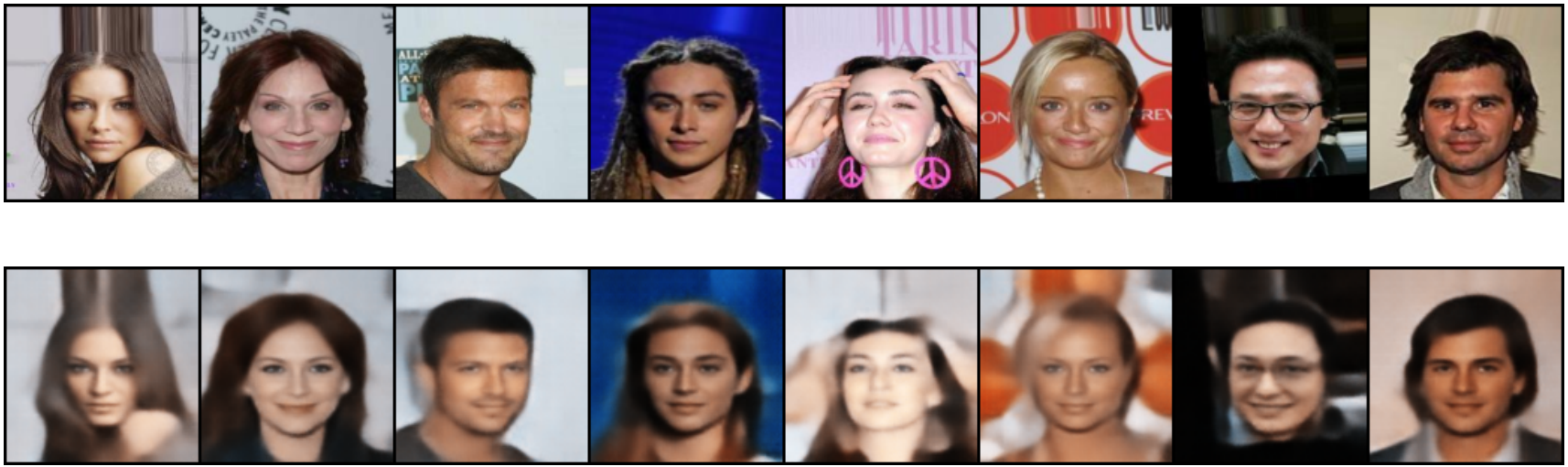


Figure: Reconstructions.

Limitations and Challenges

So far:

- The dimension of the Latent Space is a hyperparameter which needs to be tuned carefully: If the dimension is too low, we will have too much compression and a rather bad reconstruction, a dimension chosen too high, results in too little compression and often in bad decoded samples from the latent space.
- Tendency to generate blurry, unrealistic outputs. This is related to the manner in which VAEs recover data distributions and calculate loss functions.
- KLD Loss Term and Reconstruction loss Term on different Scales and the Neural Network thus just optimizes one of them
- Assumption that the variational posterior distribution $q_\theta(z|x)$ follows an isotropic Gaussian. VAE cannot guarantee that the inference algorithm will converge onto the standard Gaussian prior if the k latent neurons $z = (z_1, ..., z_k)$ are in fact correlated.
- check 2.2. here https://arxiv.org/pdf/2007.10389.pdf

Inverse Autoregressive Flow

Goal: Building flexible posterior distributions $q_\theta(z|x)$ through an iterative procedure. Think of the „simplest“ special case of IAF where we transform a Gaussian variable with diagonal covariance to one with linear dependencies, thus the Covariance between latent dimension is not necessarily 0. (see Appendix in [4])

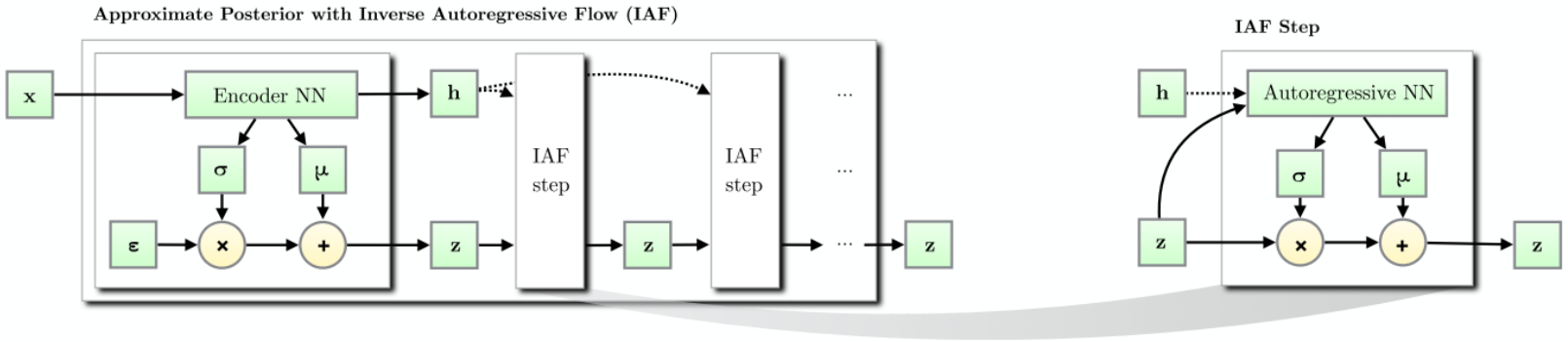


Figure: (Left) An Inverse Autoregressive Flow (IAF) transforming the basic posterior of a variational encoder to a more complex posterior through multiple IAF transforms. (Right) A single IAF transform. [4]

Normalizing Flow

Idea: Start with initial random variable with a simple distribution with known density function and then apply a chain of invertible parameterized transformations f_t, where $t \in \mathbb{N}$ represents a time-step/ iteration, such that the last iterate z_T has a more flexible distribution. [4]

$$z_0 \sim q(z_0|x)$$
$$z_t = f_t(z_{t-1}, x) \quad \forall t = 1, ..., T$$

As long as 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 as [4]

$$\log q(z_T|x) = \log(q(z_0|x)) - \sum_{t=1}^{T} \log \det \left| \frac{dz_t}{dz_{t-1}} \right|. \tag{9}$$

A single IAF layer takes as input a point in the latent space and produces as output a transformed point in the latent space. Note that the term „Normalizing“ means that after each iterate t the probability distribution needs to fulfill

$$\int q(z_t|x) dz_t = 1.$$

Inverse Autoregressive Flow

The Encoder Neural Network outputs μ_0, σ_0 as usual in addition to an extra output h. Again as usual we sample $\varepsilon \sim \mathcal{N}(0, I)$ and initialize the chain of length T with z_0 and define the iterate z_t

$$z_0 = \mu_0 + \sigma_0 \odot \varepsilon$$
$$z_t = \mu_t + \sigma_t \odot z_{t-1},$$

where for every step $t \in \{1, ..., T\}$ we use a different Autoregressive Neural Network with input z_{t-1}. We receive a triangular Jacobian with σ_t on the diagonal (note that the Jacobian $d\mu_t/dz_{t-1}$ and $d\sigma_t/dz_{t-1}$ is triangular with 0 on the diagonal) and thus the determinant

$$\det \left| \frac{dz_t}{dz_{t-1}} \right| = \prod_{i=1}^{n} \sigma_{t,i}.$$

With equation (9) we receive following density for the last iterate (note that all exp terms are removed by log) [4]

$$\log q(z_T|x) = -\sum_{i=1}^{n} \left(\frac{1}{2}\varepsilon_i^2 + \frac{1}{2}\log(2\pi) + \sum_{t=0}^{T} \log \sigma_{t,i} \right). \tag{10}$$

IAF numerically stable Transformation

A numerically more stable variant lets the Autoregressive Step outputs $[m_t, s_t]$ and and reparameterize z_t as

$$
\begin{aligned}
[m_t, s_t] &= \text{AutoregressiveNN}[t](z_t, h; \theta) \\
\sigma_t &= \text{sigmoid}(s_t) \\
z_t &= \sigma_t \odot z_{t-1} + (1 - \sigma_t) \odot m_t,
\end{aligned}
$$

where such parameterization is known as a forget gate bias in LSTMs.

Algorithm IAF

The flexibility of the distribution of the final iterate z_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. [4]

Algorithm 1: Pseudo-code of an approximate posterior with Inverse Autoregressive Flow (IAF)

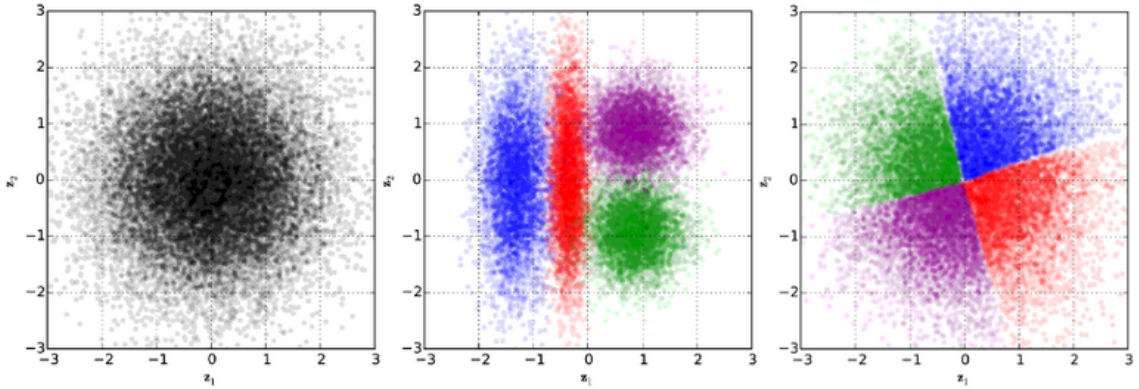
Data:
- $\mathbf{x}$: a datapoint, and optionally other conditioning information
- $\boldsymbol{\theta}$: neural network parameters
- $\texttt{EncoderNN}(\mathbf{x}; \boldsymbol{\theta})$: encoder neural network, with additional output $\mathbf{h}$
- $\texttt{AutoregressiveNN}[*](\mathbf{z}, \mathbf{h}; \boldsymbol{\theta})$: autoregressive neural networks, with additional input $\mathbf{h}$
- $\texttt{sum}(.)$: sum over vector elements
- $\texttt{sigmoid}(.)$: element-wise sigmoid function

Result:
- $\mathbf{z}$: a random sample from $q(\mathbf{z}|\mathbf{x})$, the approximate posterior distribution
- l: the scalar value of $\log q(\mathbf{z}|\mathbf{x})$, evaluated at sample '$\mathbf{z}$'

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

Figure: Proposed Pseudo-Algorithm [4]

Posterior IAF



(a) Prior distribution (b) Posteriors in standard VAE (c) Posteriors in VAE with IAF

Figure: Fitted VAE with a spherical Gaussian prior, and with factorized Gaussian (class distributions are independent - covariance matrix is diagonal) posteriors (b) or inverse autoregressive flow (IAF) posteriors (c) to a toy dataset with four classes. Each colored cluster corresponds to the posterior distribution of one class. IAF greatly improves the flexibility of the posterior distributions, and allows for a much better fit between the posteriors and the prior. [4]

Posterior IAF

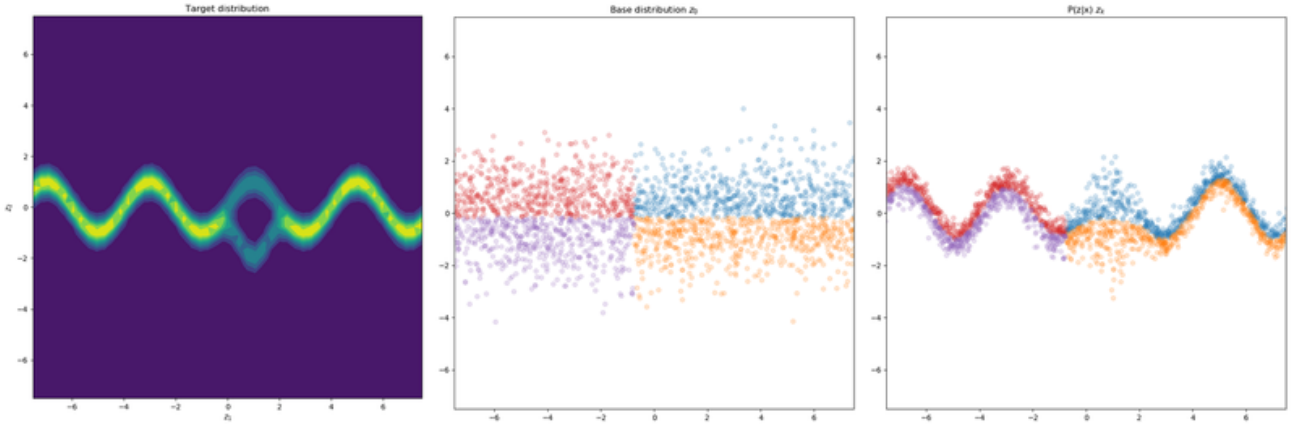


Figure: Approximate Posterior with IAF Layers with Base Distribution $z_0 \sim \mathcal{N}(0, I)$ and uniform Priors with 4 IAF Layers [12]

Drawpack: Sampling from posterior takes much longer as we need to iterate through T flow layers, in comparison to just sample $\varepsilon \sim \mathcal{N}(0, I)$ and reparameterize in standard VAE. Not very straightforward to implement.

Ladder VAE

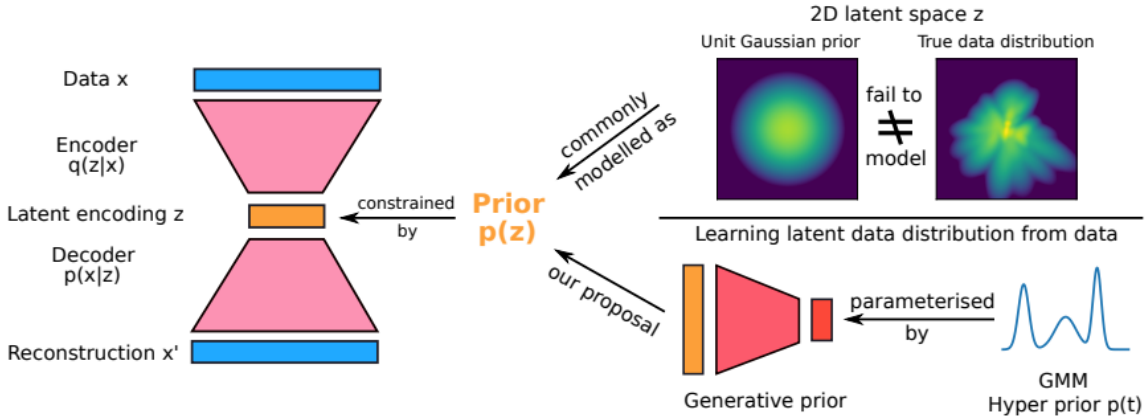


Figure: Ladder VAE: Modelling more complex Priors though multiple stochastic layers which are shared between Encoder and Decoder.

Ladder VAE: Generative Model

Same Setting as in Variational Bayes: Train a generative model $p_\theta(x,z) = p_\theta(x|z)p_\theta(z)$ for some data x using latent variables z, and an inference model q_ϕ optimizing the ELBO.
In generative model p_θ we split the latent variable into L layers and we receive for $i = 1, ..., L$

$$p_\theta(z) = p_\theta(z_L) \prod_{i=1}^{L-1} p_\theta(z_i|z_{i+1})$$
$$p_\theta(z_i|z_{i+1}) = \mathcal{N}(z_i|\mu_{p,i}(z_{i+1}), \sigma^2_{p,i}(z_{i+1})), \quad p_\theta(z_L) = \mathcal{N}(z_L|0, I)$$
$$p_\theta(x|z_1) = \mathcal{N}(x|\mu_{p,0}(z_1), \sigma^2_{p,0}(z_1)),$$

where we can also model $p_\theta(x|z_1)$ as Bernoulli in case of Binary data. We use p for the generative model, and q for the inference model. The hierarchical specification allows the lower layers of the latent variables to be highly correlated but still maintain the computational efficiency of fully factorized mode

Ladder VAE: Inference Model

VAE inference models are parameterized as a bottom-up process. Each stochastic layer is defined as a fully factorized gaussian distribution

$$
\begin{aligned}
q_\phi(z|x) &= q_\phi(z_L|x) \prod_{i=1}^{L-1} q_\phi(z_i|z_{i+1}) \\
\sigma_{q,i} &= \frac{1}{\hat{\sigma}_{q,i}^{-2} + \sigma_{p,i}^{-2}} \\
\mu_{q,i} &= \frac{\hat{\mu}_{q,i}\hat{\sigma}_{q,i}^{-2} + \mu_{p,i}\sigma_{p,i}^{-2}}{\hat{\sigma_{q,i}}^{-2} + \sigma_{p,i}^{-2}} \\
q_\phi(z_i|\cdot) &= \mathcal{N}(z_i|\mu_{q,i}, \sigma_{q,i}^2),
\end{aligned}
$$

The inference model is a precision-weighted combination of $\hat{\mu}_q$ and $\hat{\sigma}_q^2$ carrying bottom-up information and μ_p and σ_p^2 from the generative distribution carrying top-down prior information.

Ladder VAE

Define Layers with $d_0 = x$ for $i = 1, ..., L$ as

$$d_i = \text{MLP}(d_{i-1})$$
$$\hat{\mu}_{q,i} = \text{Linear1}(d_i)$$
$$\hat{\sigma}^2_{q,i} = \text{Softplus}(\text{Linear2}(d_i))$$

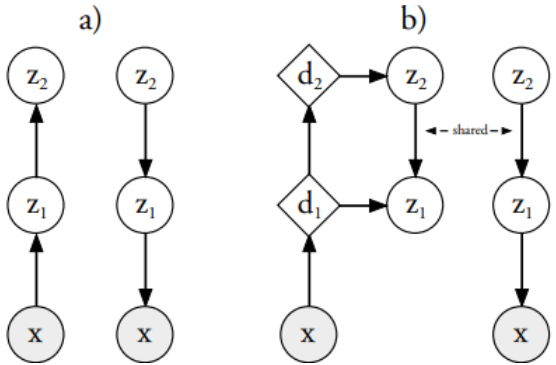


Figure: Inference (or encoder/recognition) and generative (or decoder) models for a) VAE and b) LVAE. Circles are stochastic variables and diamonds are deterministic variables. [10]

Ladder VAE

Sample from prior $p_\theta(z) = p_\theta(z_L) \prod_{i=1}^{L-1} p_\theta(z_i|z_{i+1})$ of course takes longer as we need to iterate through L stochastic layers, but it results in less blurrier samples.

Figure: Conditional Samples from Prior $p_\theta(z)$. [2]

Outlook: NVAE

Downside: Training takes 92 hours on a machine with 8 16-GB V100 GPUs! [7]

Figure: Current Achievements: Sample from Latent Space with NVAE results in very clear and sharp images. [7]

Outlook

BIVA: A Very Deep Hierarchy of Latent Variables for Generative Modeling

Flow Based Algorithms

Glow: Generative Flow with Invertible 1×1 Convolutions

VAE for Reference based Image Super Resolution: Combination of VAEs and GANs: Transformation of Low Resolution Images to High Resolution

Generating Diverse High-Fidelity Images with VQ-VAE-2

Thanks For your Attention! Questions?

Appendix: Derivation of Variational Bound I [11]

Assume the true posterior $p_\theta(z|x)$ is too difficult to compute directly (i.e., via Bayes rule), so we instead approximate it with $q_\phi(z|x)$. We then optimize ϕ instead of working with probabilities directly. Mathematically we have that

$$p_\theta(z|x) = \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(x)} = \frac{p_\theta(x|z)p_\theta(z)}{\int p_\theta(x|z)p(z)dz} \approx q_\phi(z|x).$$

The Log-marginal likelihood can then be expressed as

$$\log(p_\theta(x)) = \mathcal{D}_{KL}[p_\theta(z|x) \parallel q_\phi(z|x)] + \mathcal{L}(\theta, \phi|x), \tag{11}$$

where the first term is the non-negative KL-Divergence of the approximate to the true posterior. The second term is called the variational lower bound (or: evidence lower bound (ELBO)) on the marginal likelihood of datapoint i. We can rewrite this as

$$\begin{aligned}
\mathcal{L}(\theta, \phi|x) &= \mathbb{E}_{q_\phi(z|x)}[-\log(q_\phi(z|x)) + \log(p_\theta(x,z))] \\
&= \int q_\phi(z|x)[-\log(q_\phi(z|x)) + \log(p_\theta(x|z)) + \log(p_\theta(z))]dz \\
&= \int q_\phi(z|x)\log\Big(\frac{p_\theta(z)}{q_\phi(z|x)}\Big)dz + \int q_\phi(z|x)\log(p_\theta(x|z))dz \\
&= -\mathcal{D}_{KL}[p_\theta(z) \parallel q_\phi(z|x)] + \mathbb{E}_{q_\phi(z|x)}[\log(p_\theta(x|z))]
\end{aligned} \tag{12}$$

Appendix: Derivation of Variational Bound II [11]

In the last step we have used the fact that the KL-Divergence loss for some probability distributions p_λ and p_τ is defined as

$$\mathcal{D}_{KL}[p_\lambda(y) \parallel p_\tau(y)] = \int \log\Big(\frac{p_\lambda(y)}{p_\tau(y)}\Big) p_\lambda(y) dy = \mathbb{E}_{p_\lambda(y)}\Big[\log\Big(\frac{p_\lambda(y)}{p_\tau(y)}\Big)\Big]. \tag{13}$$

Now let us derive equation (11):

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

where in the first step, taking the expectation, we used the fact that there is no dependence on z in the marginal and the second used the identity $p_\theta(x,z) = p_\theta(z|x)p_\theta(x)$. Now using the fact that the KL-Divergence is none-negative in combination with equation (12) we will recieve the following bound for the marginal likelihood

$$\log(p_\theta(x)) \geqslant \mathcal{L}(\theta,\phi|x) = -\mathcal{D}_{KL}[p_\theta(z) \parallel q_\phi(z|x)] + \mathbb{E}_{q_\phi(z|x)}[\log(p_\theta(x|z))]. \tag{14}$$

Appendix: Derivation of Variational Bound III [11]

We note that maximizing equation (12), with respect to θ will concurrently maximize the expectation (right term) and minimize the KL-Divergence Term (left term). In general we want to maximize the Log-marginal likelihood $\log(p_\theta(x))$ which is then equivalent to maximizing equation 14.

Appendix: KL-Divergence of two Gaussians

Let $n \in \mathbb{N}$ be the dimensionality of the latent space. Assume that the prior $p_\theta(z) \sim \mathcal{N}(0, I)$ and the posterior approximation $q_\phi(z|x) \sim \mathcal{N}(\mu, \sigma)$ are Gaussian, with μ and σ as the variational mean and standard deviation evaluated at datapoint k, and let μ_i and σ_i simply denote the i-th element of these vectors. Then with the definition of the KLD term given in Equation (13) we split the log into two parts and calculate two integrals:

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

For the second integral we get:

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

Putting now both together, it follows with a constant $C \in \mathbb{R}$

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

Appendix: Loss Function for Inverse Autoregressive Flows

We can use the Derivation of the Variational Bound.

$$\begin{aligned}\mathcal{L}(\theta,\phi|x) &= \mathbb{E}_{q_\phi(z_T|x)}[-\log(q_\phi(z_T|x)) + \log(p_\theta(x,z_T)] \\ &= \mathbb{E}_{q_\phi(z_T|x)}[-\log(q_\phi(z_T|x)) + \log(p_\theta(x|z_T)) + \log(p_\theta(z_T))],\end{aligned}$$

where the first term refers to equation (10), the second term to the distribution of our output variable (e.g. Gaussian for continuous, Bernoulli for discrete values) and the last term to our standard diagonal Gaussian prior.

$$\log q_\phi(z_T|x) = -\sum_{i=1}^{n}\left(\frac{1}{2}\varepsilon_i^2 + \frac{1}{2}\log(2\pi) + \sum_{t=0}^{T}\log\sigma_{t,i}\right)$$

$$\log p_\theta(z_T) = -\sum_{i=1}^{n}\frac{1}{2}(z_T)_i^2 + \frac{1}{2}\log(2\pi).$$

The paper proposes that each IAF layer has an objective $\log(p(z_t|z_{<t})) - \log(q(z_t|x,z_{<t}))$ where the priors can be flexible. This would result in an autoregressive prior for the generative model

$$\log p_\theta(z_{1:T}) = p_\theta(z_T)\prod_{t=1}^{T-1} p_\theta(z_t|z_{t+1:T})$$

References

- Pca for 3-dimensional point cloud.
  URL:https://www.algosome.com/articles/pca-three-dimensions-point-cloud.html.
- ermongroup.
  Variational-ladder-autoencoder.
  https://github.com/ermongroup/Variational-Ladder-Autoencoder, 2017.
- David Dao (https://stats.stackexchange.com/users/98120/david dao).
  How does the reparameterization trick for vaes work and why is it important?
  Cross Validated.
  URL:https://stats.stackexchange.com/q/205336 (version: 2018-01-16).
- Diederik P. Kingma, Tim Salimans, and Max Welling.
  Improving variational inference with inverse autoregressive flow.
  CoRR, abs/1606.04934, 2016.
- Diederik P Kingma and Max Welling.
  Auto-encoding variational bayes, 2013.

References

- Emile Mathieu, Tom Rainforth, N. Siddharth, and Yee Whye Teh.
  Disentangling disentanglement in variational autoencoders, 2018.
- NVLABS.
  Nvae.
  https://github.com/NVlabs/NVAE, 2021.
- Joseph Rocca.
  Understanding variational autoencoders (vaes).
  URL:https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73.
- Oleh Rybkin, Kostas Daniilidis, and Sergey Levine.
  Simple and effective VAE training with calibrated decoders.
  CoRR, abs/2006.13202, 2020.
- Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther.
  Ladder variational autoencoders, 2016.

References

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).

Ritchie Vink.
Another normalizing flow: Inverse autoregressive flows.
URL:https://www.ritchievink.com/blog/2019/11/12/another-normalizing-flow-inverse-autoregressive-flows/.