Chapter 5

Variational Autoencoders (VAE)

In this chapter we will first give an overview on generative models and next introduce the VAE as such as model. The VAE framework serves as backbone model for all our model implementations and variations with different encoder and decoder blocks but also with variations in latent spaces such as continuous or discrete ones. Moreover, we will show how disentanglement of VAEs can be evaluated. For that, we will explain two main strategies, one is of qualitative nature while the other is of quantitative meaning. Regarding the quantitative metrics we showcase two state of the art metrics which leverage different tools from statistics respectively information theory.

5.1 Overview generative models

Generative models are one one the most potential approaches towards the goal of understanding the underlying distribution of datasets from different domains such as image, text or audio.This class of models is able to create (generate) new data samples which are similar to the data samples the model has seen during training. The generative models are forced to discover and efficiently learn the underlying distributions and properties of the data in order to reconstruct or generate similar samples. We can think of generative models as a machine which is able to look at any object for example a car and by inspecting a huge amount of sample cars the model finally learns a construction manual plan of how to build new variations of cars with different colours, shapes, engine types, heights, number of doors and so on. If a model is truly able to generate new samples which follow the appearance of real world objects, we can truely say that it indeed has learned and understood a concept without teaching, known as supervised learning. The long term goal is to be able to automatically extract and learn these natural features occurring in the real world no matter what the data distribution might look like. There are two approaches to generative modeling which are shown in Figure 5.1 and described below

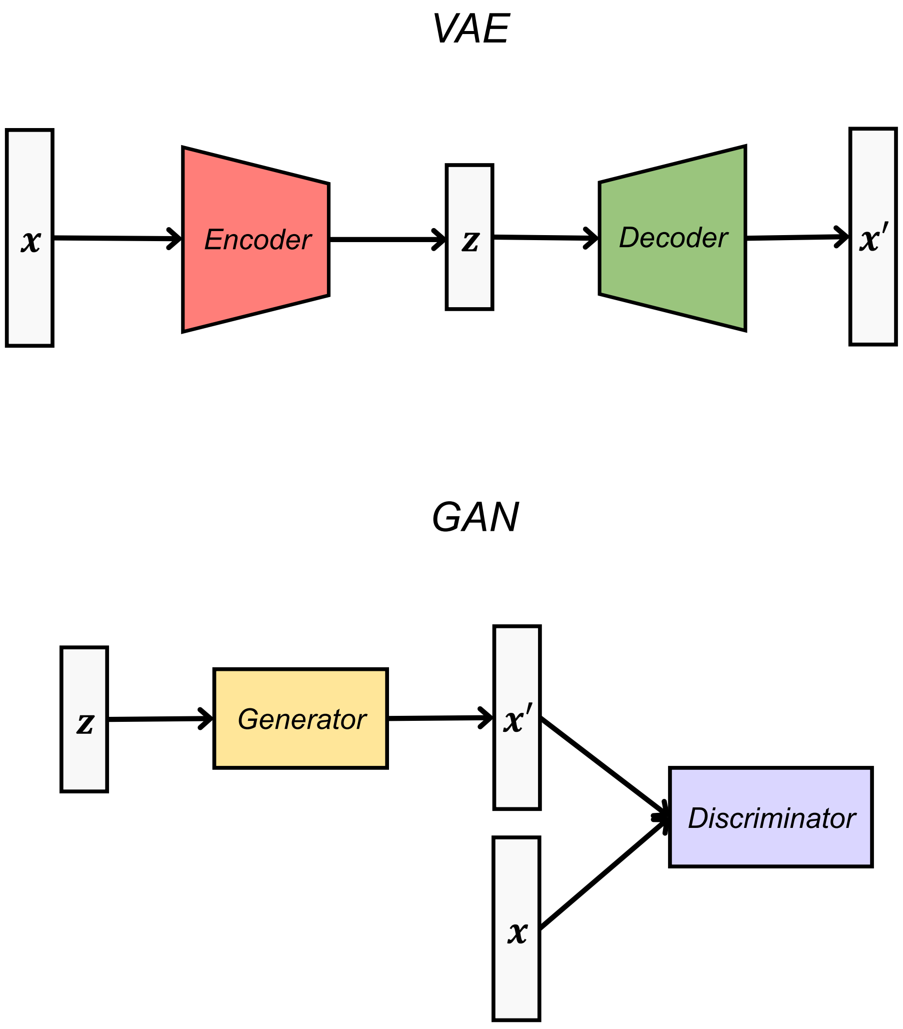


Figure 5.1: Architectural difference between GAN and VAE

- Generative Adverserial Networks (GAN):
  These models are trained in a competing game between two separate networks. One network is creating (generator) data samples, while the other network (discriminator) tries to distinguish data samples as either coming from the training data set or from the generator. The training is stopped as soon as the discriminator is randomly guessing between fake and real data samples, which means that its probability to correctly discriminate between images from the true distribution and those from the generator distribution is exactly $p = 0.5$. GANs are able to generate new data samples for the image domain with astonishing results; yet, it remains difficult to learn to generate discrete data such as text. Finally, when we want to learn a compressed representation of our data samples, in effect perform inference, GANs can not be used as they are a mapping from a latent code z with unit gaussian noise distribution $\mathcal{N}(0, 1)$ into the image domain x, but in effect we want the inverted functionality.

- Variational Autoencoders (VAE):
  VAEs can be used to approach and solve the two disadvantages of GANs, which is why we are using focusing on this class of generative models for our work with written language and NLP. VAEs give us the nice property to formalize the learning problem in a directed probabilistic graph model $p(x|z)$, where the encoder learns an approximation of the of the true data distribution $p(z|x)$ where we have to optimize our VAE objective in order to maximize the log likelihood of the data distribution. VAEs are an extension of vanilla autoencoders (AE) with constraints on the distribution of the latent variables z. VAEs similarly to GANs also consist of two neural networks. One of them is the inference network outputting the latent variable distribution of z and the second NN is the generation network, which maps back from latent variable space Z into the data domain X.

In order to understand how the architecture for our experiments in the following chapter works, we will start with the basic model contained in VAEs, the autoencoder AE. From there we will explain the differences between AE, VAE and β-VAE.

5.2 Autoencoder (AE)

Autoencoders first introduced in [69] are a concept of neural network models which try to learn a compressed representation $\mathbf{c} \in \mathbb{R}^m$ of the input data $\mathbf{x} \in \mathbb{R}^n$, which is of at least higher dimensionality. AEs learn these features in an unsupervised manner without any corresponding label because the goal of the autoencoder is to reconstruct the input at the output. Autoencoders consist of two neural network blocks:

- **Encoder:** The task of the encoder function is to map the input x from an n-dimensional space into a smaller m-dimensional code space, which is equivalent to the values within the hidden layer. The hidden layer if often referred to as bottleneck, as it is the layer with the lowest dimensional representation of the data from input layer up until the output layer. The input layer dimension is usually larger than the output layer dimension of the encoder.
- **Decoder:** The task of the decoder function is to re-map the compressed code representation from the m-dimensional code space back into the original n-dimensional input data space.

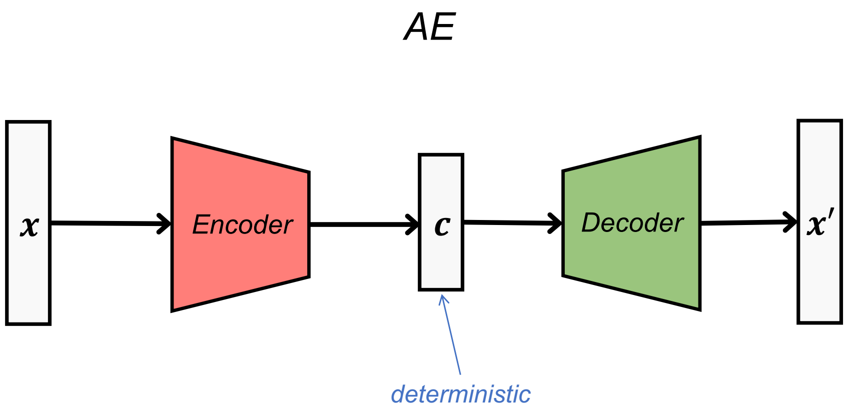


Figure 5.2: Simple autoencoder (AE) with non-stochastic latent code **c**

The basic structure of an AE is shown in Figure 5.2. AEs differ from other previously explained neural network models because they do not predict a certain ground truth information but rather try to reconstruct the output as accurately as possible to the input **x**. In effect, vanilla AEs come closest to models which learn the identity function, while at the same time extract a condensed, less sparse representation of the input data. Assuming that the encoder function is given by $\mathbf{c} = g_\phi(x)$ and the decoder function by $\mathbf{x}' = f_\theta(c)$, the objective function of the autoencoder is to minimize the sum of differences between every input and output, which represents the reconstruction error

$$\mathcal{L}_{AE}(\phi, \theta, \mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}^{(i)} - f_\theta(g_\phi(\mathbf{x}^{(i)})))^2 \tag{5.1}$$

The parameters of neural network parameters of the encoder ϕ and decoder θ are learned concurrently during training to reduce the reconstruction error. The ultimate goal of the enoder is $\mathbf{x} \approx f_\theta(g_\phi(\mathbf{x}))$, which can be achieved by the identity function.

Autoencoders can be applied for problems, where we are interested in *dimensionality reduction* into a lower dimensional representation while at the same time learning a latent code mapping. Variations of the the autoencoder family are:

- **Denoising Autoencoders** [70] which prevent AEs from overfitting when there are more network parameters than the total amount training samples. Moreover, denoising AEs improve upon robustness of the latent representation in case of corruption or partial data loss.
- **Sparse Autoencoders** [71] which introduces a ‘sparse‘ constraint upon the activation of single units within the latent code **c** to prevent overfitting and improve robustness of the internal representation. The goal is to keep the number of simultaneously active hidden units c_i at a minimum, ideally at most one unit is activate

at any given time. The sparsity constraint is added as penalty term into the vanilla AE objective

- **Variational Autoencdoers** [1] (which will be explained in the next subsection)

5.3 Variational Autoencoder (VAE)

The term autoencoder in variational autoencoder is quite misleading [72], as it is more related to *Variational Bayesian* methods and *probabilistic graphical model* which describes the dependence structure between different random variables RV. The VAE‘s goal is to find an explicit distribution which is able explain most of the data samples of the input instead of learning a simple mapping onto a fixed vector (here fixed means deterministic as in the AE case). The VAE is a probabilistic approach to describe data samples $\mathbf{x}$ in the latent representation space $\mathbf{Z}$. Therefore, every latent variable in the latent code $\mathbf{z}$ is described by a probability distribution. The model architecture is shown in Figure 5.3 below.

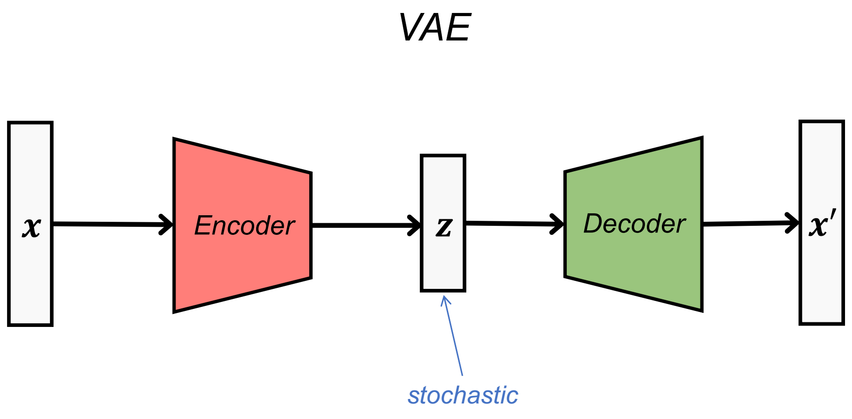


Figure 5.3: Architecture design of VAE

which can be further detailed and explained by opening up the decoder to have a look at the stochastic sampling process as shown below in Figure 5.4

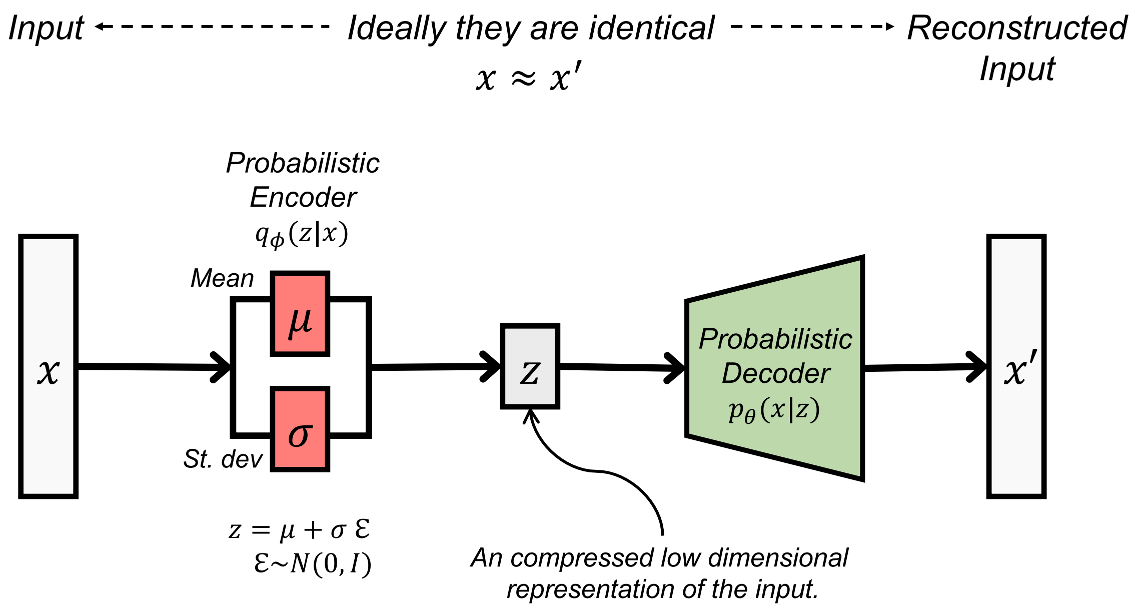
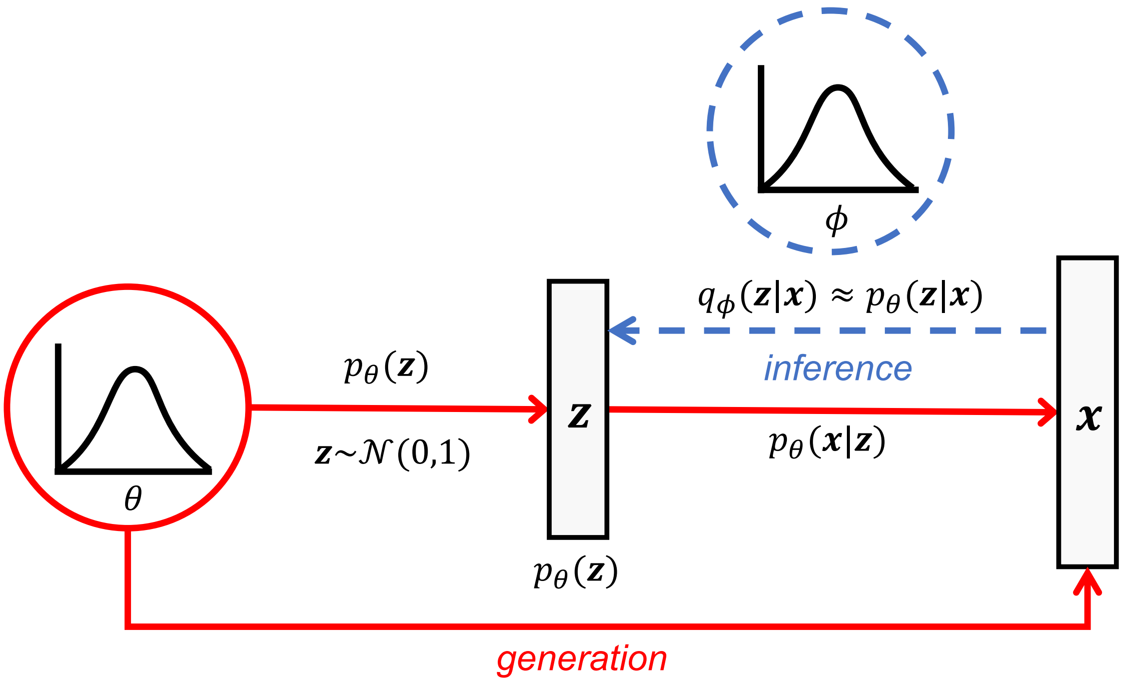


Figure 5.4: Architecture design of VAE - with detailed encoder process

In order to understand the principle of VAEs we need to differentiate between three different probability respectively conditional probability distributions which are

- *Prior* $p(\mathbf{z})$
- *Likelihood* $p_\theta(\mathbf{x}|\mathbf{z})$, which is defined by the decoder neural network
- *Posterior* $q_\phi(\mathbf{z}|\mathbf{x})$, which is defined by the encoder neural network

Knowing these terms, we can sketch our probabilistic graphical models as shown in Figure 5.5:



ϕ
$q_\phi(\mathbf{z}|\mathbf{x}) \approx p_\theta(\mathbf{z}|\mathbf{x})$
inference
$p_\theta(\mathbf{z})$
$\mathbf{z}$
$\mathbf{x}$
$\mathbf{z} \sim \mathcal{N}(0,1)$
$p_\theta(\mathbf{x}|\mathbf{z})$
θ
$p_\theta(\mathbf{z})$
generation

Figure 5.5: Graphical probabilistic model of the VAE process (inference + generation)

Where again ϕ is the collection of parameters belonging to the encoder function and θ for the parameters of the decoder function $\mathbf{z}$ denotes the latent space representation as output of the encoder. We can generate new data samples by doing the following two steps

1. Sampling a latent variables vector $\mathbf{z}^{(\mathbf{i})}$ from the posterior $p_\phi(\mathbf{z}|\mathbf{x})$ which is an approximation to the prior distribution $p(\mathbf{z})$

2. Using the decoder as generative network and reconstructing the sampled latent vector with the conditional likelihood function $p_\theta(\mathbf{x}|\mathbf{z} = \mathbf{z}^{(i)})$

The lower dimensional space $\mathbf{z}$ is stochastic with the encoder posterior distribution function $p_\phi(\mathbf{z}|\mathbf{x})$ being a Gaussian. Due to the sampling step 1, the latent representation $\mathbf{z}$ in the VAE is noisy and not deterministic. Therefore, we are actually enforcing the latent space representation to be continuous and smooth. Latent space vectors $\mathbf{z}$ which are close to each other, in effect only differ due to the added sampling noise, will thus be similar in their reconstructed appearance.

Within the VAE, information is lost because of two reasons

1. We squeeze our input $\mathbf{x}$ from a large dimensional space $\mathbb{R}^n$ through the encoder network into a lower dimensional space $\mathbb{R}^d$, which represents a bottleneck

2. In addition to the bottleneck, we additionally sample from the bottleneck which makes our reconstruction even more inaccurate to the original encoder input

The optimal parameter set θ' is the one, which maximizes the reconstruction likelihood for each data point $\mathbf{x^{(i)}}$. Therefore, our goal is

$$\theta' = \arg\max_{\theta} \prod_{i=1}^{N} p_\theta(\mathbf{x^{(i)}}) \tag{5.2}$$

which is equivalent to maximizing the sum of log probabilities. This is due to the fact that the logarithm *log* is a monotonically increasing function. Moreover, does it make computation numerically stable and allows subsequent simplifications

$$\theta' = \arg\max_{\theta} \sum_{i=1}^{N} log(p_\theta(\mathbf{x^{(i)}}) \tag{5.3}$$

If we want to compute the true posterior distribution of the latent space $p_\theta(\mathbf{z}|\mathbf{x})$ we would have to determine $p_\theta(\mathbf{x})$ given the *Bayes‘ Theorem*

$$p_\theta(\mathbf{z}|\mathbf{x}) = \frac{p_\theta(\mathbf{x}|\mathbf{z})\, p_\theta(\mathbf{z})}{p_\theta(\mathbf{x})} \tag{5.4}$$

Where $p_\theta(\mathbf{x})$ is the evidence which can be calculated as marginalization of all the possible latent space vector values $\mathbf{z}$ as follows

$$p_\theta(\mathbf{x^{(i)}}) = \int p_\theta(\mathbf{x^{(i)}}|\mathbf{z})\, p_\theta(\mathbf{z}) d\mathbf{z} \tag{5.5}$$

It is evident, that this approach is inconvenient because calculating the posterior $p_\theta(\mathbf{x^{(i)}})$ for each data sample $\mathbf{x^{(i)}}$ quickly becomes quite expensive and intractable. Therefore, VAEs use variational approximate inference of the intractable distribution of the true posterior which is represented by the encoder distribution function by $q_\phi(\mathbf{z}|\mathbf{x})$. Because $q_\phi(\mathbf{z}|\mathbf{x})$ is only an estimation of the true intractable posterior, we have to keep track of the difference of those two probabilities. This can be done by measuring the difference between those probability distributions with the help of the KL divergence which quantifies the distance. For two probability distributions the KL divergence is given by

$$D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p_\theta(\mathbf{z}|\mathbf{x})) = \int q_\phi(\mathbf{z}|\mathbf{x})\, log\left(\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})}\right) d\mathbf{z} \tag{5.6}$$

and by Jensen‘s inequality, the KL divergence is always non-negative $D_{KL}(p||q) \geq 0$. So we are able to determine how different the approximated posterior $q_\phi(\mathbf{z}|\mathbf{x})$ is from the true posterior $p_\theta(\mathbf{z}|\mathbf{x})$.

Now we will derive the objective function for the VAE by analyzing and decomposing the KL-divergence term above.

$$\begin{aligned}
D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p_\theta(\mathbf{z}|\mathbf{x})) &= \int q_\phi(\mathbf{z}|\mathbf{x}) \, log \left(\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})} \right) d\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \, log \left(\frac{q_\phi(\mathbf{z}|\mathbf{x}) \, p_\theta(\mathbf{x})}{p_\theta(\mathbf{z}, \mathbf{x})} \right) d\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \left(log \, p_\theta(\mathbf{x}) + log \left(\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}, \mathbf{x})} \right) \right) d\mathbf{z} \\
&= log \, p_\theta(\mathbf{x}) + \int q_\phi(\mathbf{z}|\mathbf{x}) \, log \left(\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}, \mathbf{x})} \right) d\mathbf{z} \\
&= log \, p_\theta(\mathbf{x}) + \int q_\phi(\mathbf{z}|\mathbf{x}) \, log \left(\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{x}|\mathbf{z}) \, p_\phi(\mathbf{z})} \right) d\mathbf{z} \\
&= log \, p_\theta(\mathbf{x}) + \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [\, log \left(\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z})} \right) - log \, p_\theta(\mathbf{x}|\mathbf{z}) \,] \\
&= log \, p_\theta(\mathbf{x}) + D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p_\theta(\mathbf{z}) - \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [\, log \, p_\theta(\mathbf{x}|\mathbf{z}) \,]
\end{aligned} \tag{5.7}$$

Line three to four uses the fact, that any probability distribution has the property $\int q(\mathbf{z}|\mathbf{x}) d\mathbf{z} = 1$. From line five to six, we used the definition of KL divergence provided above. Now, when we reorder the equation above, we get the following identity:

$$\begin{aligned}
log \, p_\theta(\mathbf{x}) - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p_\theta(\mathbf{z}|\mathbf{x})) &= \\
\mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [\, log \, p_\theta(\mathbf{x}|\mathbf{z}) \,] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p_\theta(\mathbf{z})) &= \\
-ELBO &
\end{aligned} \tag{5.8}$$

The left-hand side is our objective for the VAE which we want to maximize because the first term represents the reconstruction likelihood (we want to increase this term as much as possible) and the second term ensures that our learned approximate posterior distribution is similar to the true prior distribution of the data samples (we want to keep this term as small as possible). This works because the KL divergence term has the effect of a regularizer. Given the fact, that many optimization algorithms such as the stochastic gradient descent (SGD) work by minimzing the objective function, we have to flip the signs of this term. The final objective function for the VAE is thus given by

$$\begin{aligned}
\mathcal{L}_{VAE}(\phi, \theta, \mathbf{x}, \mathbf{z}) &= -ELBO \\
&= \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [\, log \, p_\theta(\mathbf{x}|\mathbf{z}) \,] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p_\theta(\mathbf{z}))
\end{aligned} \tag{5.9}$$

ELBO is an expression defined for Variational Bayesian methods. This loss expression is known as the *variational lower bound* respectively *evidence lower bound.* ELBO is the negative of the objective function for VAEs. The term is using ‘lower‘ because the KL divergence is non-negative. Therefore, the VAE objective will always be smaller than the ‘true‘ log likelihood $log\, p_\theta(\mathbf{x})$ and therefore underestimating it as follows

$$log\, p_\theta(\mathbf{x}) - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\,||\,p_\theta(\mathbf{z}|\mathbf{x})) = \mathcal{L}_{VAE} = -ELBO \leq log\, p_\theta(\mathbf{x}) \tag{5.10}$$

Further, we now see that we are optimizing by minimizing the negative log-likelihood (NLL), which is the first term in the equation above. Thus we retrieve the best set of parameters by minimizing the loss function $\mathcal{L}_{VAE}$ respectively maximizing the evidence lower bound (ELBO) using gradient descent methods to solve

$$\theta', \phi' = \arg\min_{\theta,\phi} \mathcal{L}_{VAE} \tag{5.11}$$

We can see, that by minimizing the VAE loss, we are simultaneously maximizing the ELBO which is proportionally behaving to the likelihood and thus we increase the reconstruction likelihood. To sum up how the VAE works here a sequence of steps from input to output:

1. Pass input $\mathbf{x}$ into the encoder network
2. A latent representation vector $\mathbf{z}$ can be sampled from the posterior distribution $q_\phi(\mathbf{z}|\mathbf{x})$
3. Pass the latent vector $\mathbf{z}$ into the decoder to retrieve a close reconstruction $\mathbf{x}'$ of the initial input $\mathbf{x}$ in step 1.

5.3.1 Reparameterization trick - normally distributed prior

The only missing part in fully explaining the VAE is the process within the bottleneck which generates latent variable samples $\mathbf{z}$ which are non-deterministic by sampling from $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$. This method is called *reparameterization trick.* This is needed because back-propagation (BP) in neural networks for determining the encoder and decoder parameters θ, ϕ only works for deterministic nodes and operations. This problem is shown in the Figure 5.6 where we clearly have to pass through a deterministic sampling node (yellow colored).

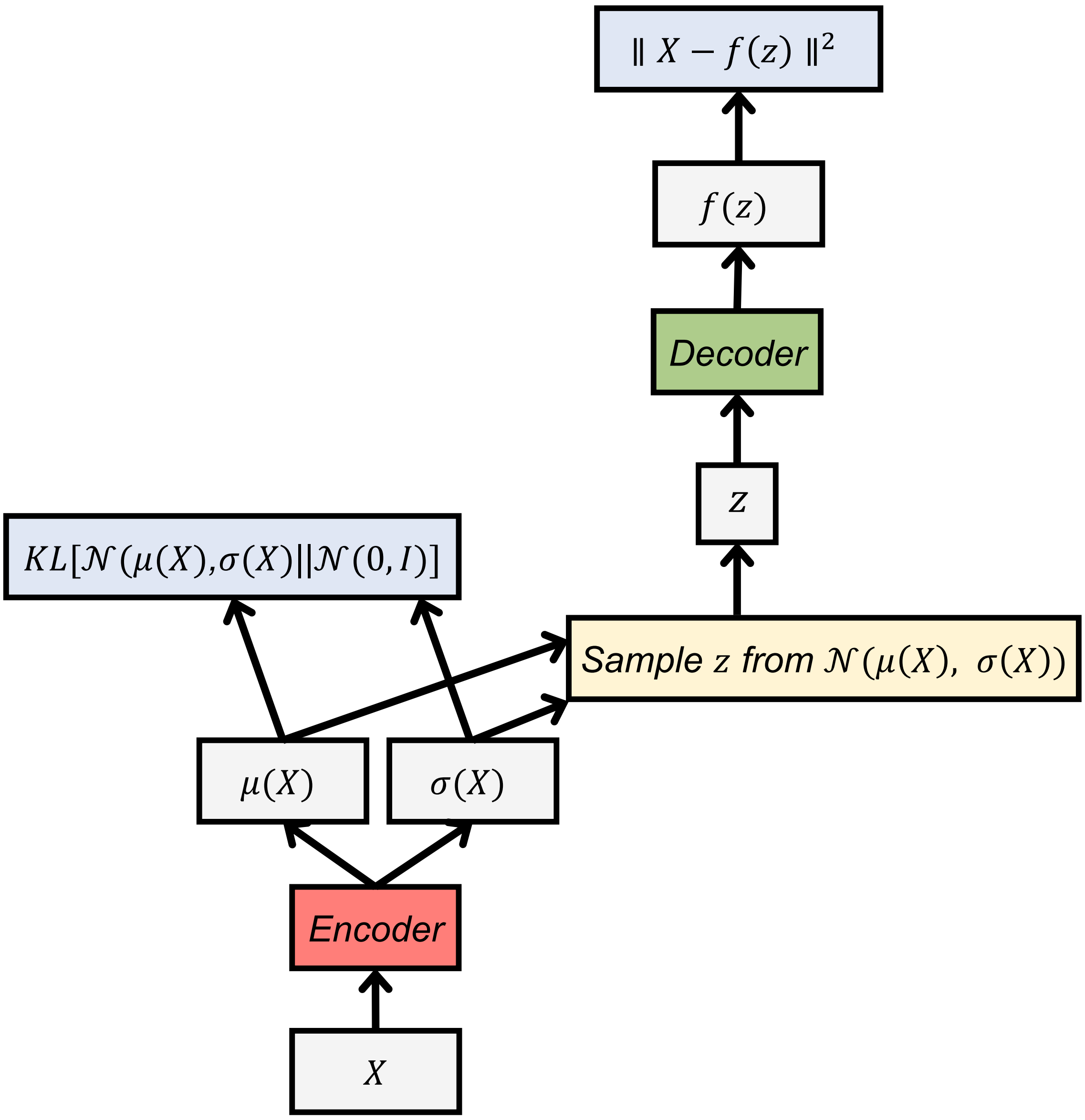

$\| X - f(z) \|^2$
$f(z)$
Decoder
z
$KL[\mathcal{N}(\mu(X),\sigma(X)||\mathcal{N}(0,I)]$
Sample z from $\mathcal{N}(\mu(X),\ \sigma(X))$
$\mu(X)$
$\sigma(X)$
Encoder
X

Figure 5.6: Backpropagation and the need for a reparameterization trick

This excludes the option of a simple random sampling in the bottleneck because sampling generates stochastic values, whereas backpropagation expects deterministic values. The reparamterization trick is our remedy for solving this issue. It assumes, that we can sample a random latent variable vector $\mathbf{z}$ by employing a deterministic construction function which is determined by the data input $\mathbf{x}$ and some small value of randomness ϵ sampled independently and injected into this construction function as follows

$$\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^{\mathbf{2}} * \mathbf{I}) \tag{5.12}$$

In that way, the stochasticity from the sampling process is independent of the parameters of the encoder and decoder network. The values will be sampled in the following way

$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon} \tag{5.13}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are deterministic outputs generated by two different FC layers. Here, $\odot$ refers to the *Hadamard product* which denotes the element-wise product. A standard choice for the approximated posterior is a multivariate Gaussian distribution with a diagonal covariance matrix. Therefore, in order to get such a multivariate Gaussian distribution, our sampled noise must also follow a Gaussian distribution. A common way to achieve this is by simply sampling $\boldsymbol{\epsilon}$ from the normal distribution; thus we have

$$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{5.14}$$

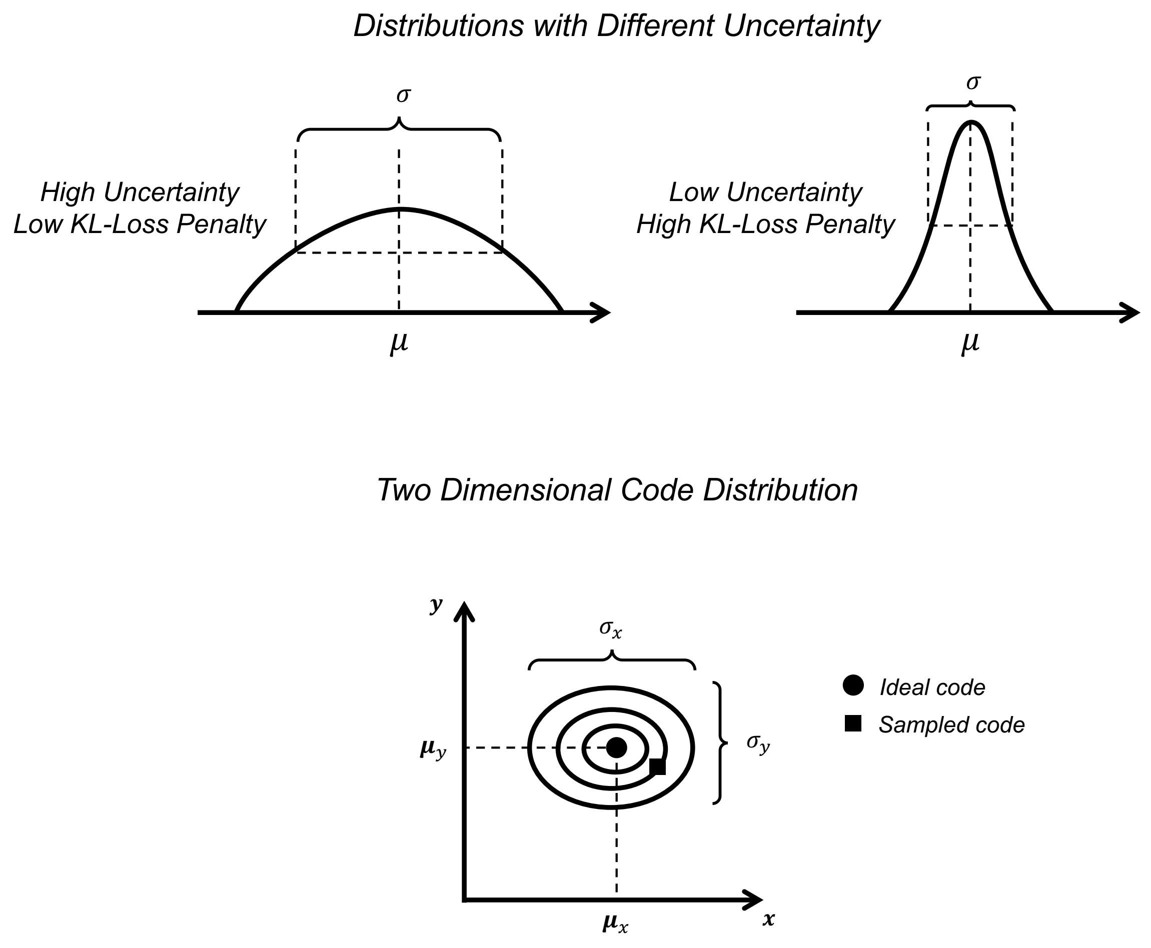


Figure 5.7: The interpretation behind the variance & mean for a latent code variable z

Therefore, the circumvented random sampling process using the reparameterization trick looks as shown in the Figure 5.8 below

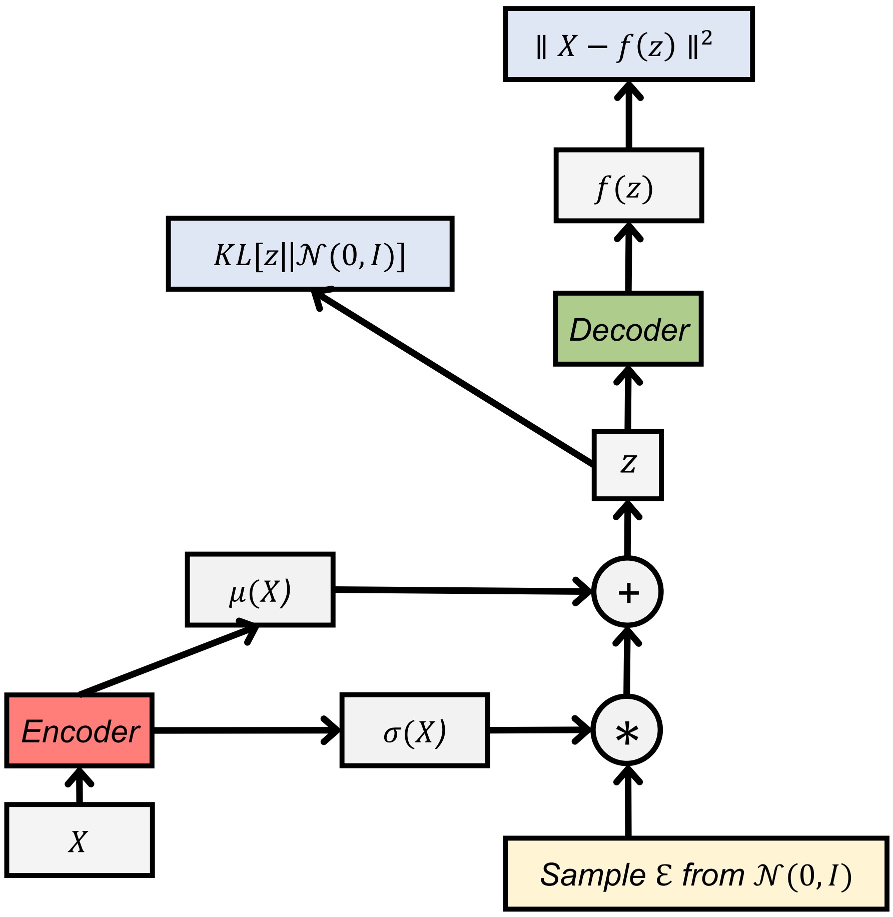


Figure 5.8: Reparameterization without any stochastic node on the way from output to input feasible to backpropagation (BP)

Changing the sampling process in the bottleneck and getting from '$\sim$' which is equivalent to drawing from a random distribution to '$=$' is the most important step in the VAE and allow us to leverage the tools of deep learning. This reparameterization function only depends on the deterministic parameters from the inference network. We can therefore calculate the gradients of the decoder output $f(\mathbf{z})$ with respect to the parameters of the latent variable distribution $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ and backpropagate this information to the encoder. Finally, this allows us to train the VAE in an end-to-end fashion while using a graphical model and variational bayesian methods. The parameter σ can be understood as a measure of uncertainty. This is shown in Figure 5.7. The autoencoder is a distribution, in which the sentences of our dataset are encoded without any uncertainty as shown in the Figure 5.9.

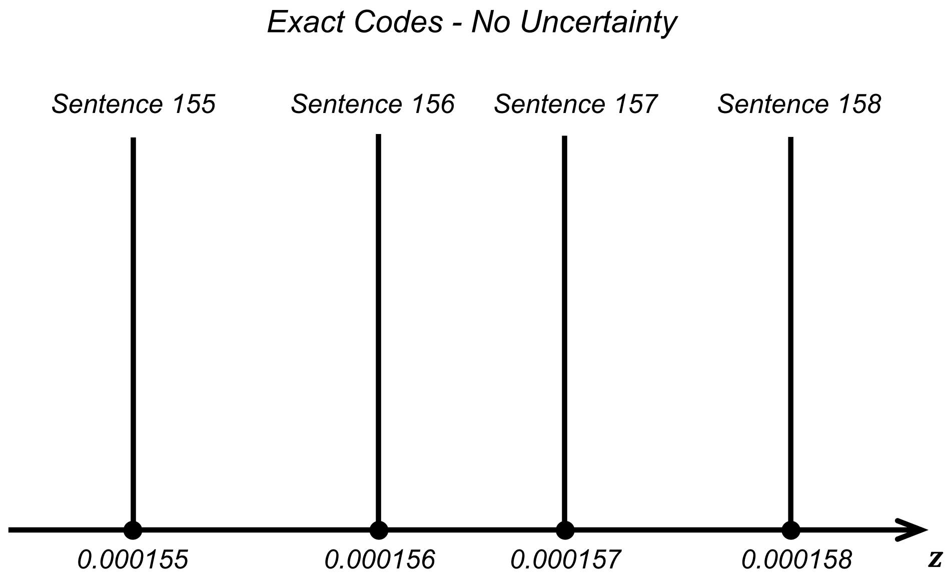


Figure 5.9: What the distribution of an autoencoder (AE) looks like

5.4 Regularized Variational Autoencoder (β-VAE)

Now, that we have in detail explained how the VAE works, understanding what β-VAE is doing is quite straightforward because the β-VAE model is a modification of the VAE framework presented in previous section. The emphasis of introducing β-VAE lies in the discovery of independent generative factors in separate latent dimensions z_i of within the whole latent vector $\mathbf{z}$, thus disentangled representation learning. The key essence is the introduction of a *Lagrangian multiplier* β which is considered as an additional hyperparameter to the original VAE objective which is affecting the KL divergence between our approximated posterior $q_\phi(\mathbf{z}|\mathbf{x})$ and the imposed prior distribution $p_\theta(\mathbf{z})$. From now on, in order to prevent confusion between the likelihood and the prior, we will denote the prior as: $p(\mathbf{z})$. β-VAE focuses on the latent space $\mathbf{Z}$ and understanding its disentanglement. One can think of the latent space $\mathbb{R}^d$ as a space, in which every dimension d contains certain information which might be helpful in reconstructing the input $\mathbf{x}$, for example when talking about NLP, one dimension could save information about the sentiment of the text document, whereas the second dimension would describe the topic of the text document and so on. With that regard, the latent space contains the reconstruction plan the decoder should make use of in order to optimize the data likelihood. Often this latent space is entangled, which means, that attributes such as style, topic, tense and so on are contained in several latent variables z_i and therefore single latent variables will have a high correlation or contain redundancy such as information or attributes already sufficiently described by another latent variable z_i. The main benefit of a β-VAE is that is is capable of learning a smooth latent space representations $\mathbf{z}$ while at the same time forcing the model to encode different factors of variation into different latent variable units z_i due to the constraint it gets in the KL divergence term over β. Whereas for standard AEs (where $\beta = 0$), we only need to learn a 'hard' in discontinuous representation of the input which only is capable of exactly reproduce the input without any safety margin in case our latent space information is corrupted by a small amount of noise or shifted on purpose.

A disentangled representation means, that each latent dimension in the latent representation vector $\mathbf{z}$ is only sensitive to a single ground truth generative factor and therefore invariant to variations of other generative factors. Therefore, we can clearly inspect and denote the attribute each latent dimension has learned which helps us in clearly interpreting the learned latent representation. β-VAE is able to learn such attributes in a very disentangled manner such that it separates information about the x and y coordinates into two mutually exclusive latent variable dimensions. Because VAE and β-VAE are directly related to each other, we also have to maximize the log likelihood of the generated data samples while keeping the KL divergence loss between prior and posterior fairly small. As the only difference is the introduction of the new KL divergence hyperparameter β, the objective function looks as follows

$$\begin{aligned}\mathcal{L}_{\beta-VAE}(\phi, \theta, \beta, \mathbf{x}, \mathbf{z}) &= -ELBO \\ &= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\,log\, p_\theta(\mathbf{x}|\mathbf{z})\,] - \beta D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\,||\,p_\theta(\mathbf{z}))\end{aligned} \tag{5.15}$$

Having introduced the β-VAE, the relationship between all presented variations of bottleneck models can be described as follows:

- $\beta = 0$: vanilla AE with standard maximum likelihood objective function
- $\beta = 1$: vanilla VAE with the standard Bayes solution
- $\beta \notin \{0, 1\}$: regularized VAE, β-VAE

A variation of the β-VAE is the capacity controlled β-VAE by introducing a target capacity value C which can be increased during the training process and allow a higher KL divergence loss as training progresses. The capacity C corresponds to the amount of information encoded in the latent and it is measured in the *nats* which is a unit of information originating from ideas in information theory. The capacity C is increased from zero to a value which is provides enough capacity in order to generate output with lower reconstruction error.

The objective function for the capacity controlled γ-VAE is described as follows

$$\begin{aligned}\mathcal{L}_{\gamma-VAE}(\phi,\theta,\gamma,C,\mathbf{x},\mathbf{z}) &= -ELBO \\ &= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\,log\, p_\theta(\mathbf{x}|\mathbf{z})\,] - \gamma\,|D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\,||\,p_\theta(\mathbf{z})) - C|\end{aligned} \tag{5.16}$$

5.4.1 Effect of β

β in the objective function acts as a regularizer on the KL divergence between the approximated posterior and the imposed prior.

We want to explain this given the following three visualizations.

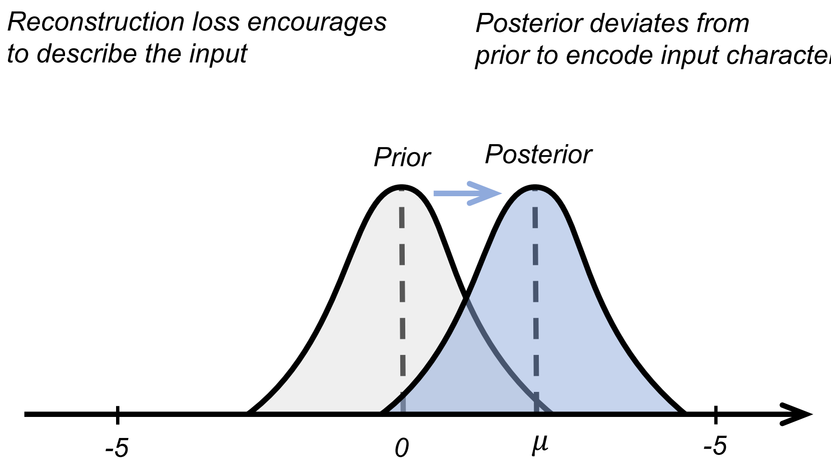

Reconstruction loss encourages to describe the input
Posterior deviates from prior to encode input characteristic
Prior
Posterior
-5
0
μ
-5

Figure 5.10: Active posterior

In the case of 5.10, the posterior is moving away from the posterior and therefore tries to find a space in the latent space, which might help in reducing the reconstruction error. Whereas in 5.11 the opposite effect is happening. As the KL penalty is larger the the benefit the posterior distribution is contributing to the increase the likelihood, the posterior is being drawn towards the distribution of the prior in order to reduce the penalty introduced by the KL divergence term.

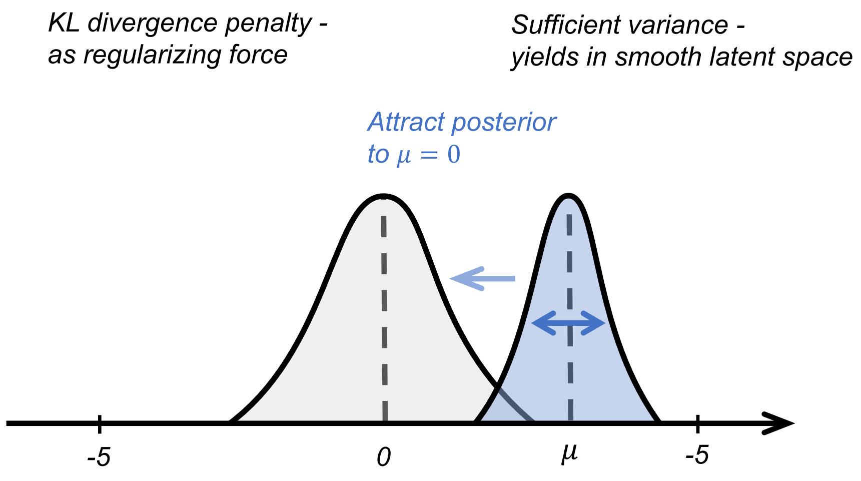


Figure 5.11: Collapsing posterior

Finally, in 5.12, the case without a KL divergence term, the posterior decides to only encode a very tiny space in the latent space, which causes the latent space to be very unsmooth and filled with a lot of ‘air‘ instead of sample representations

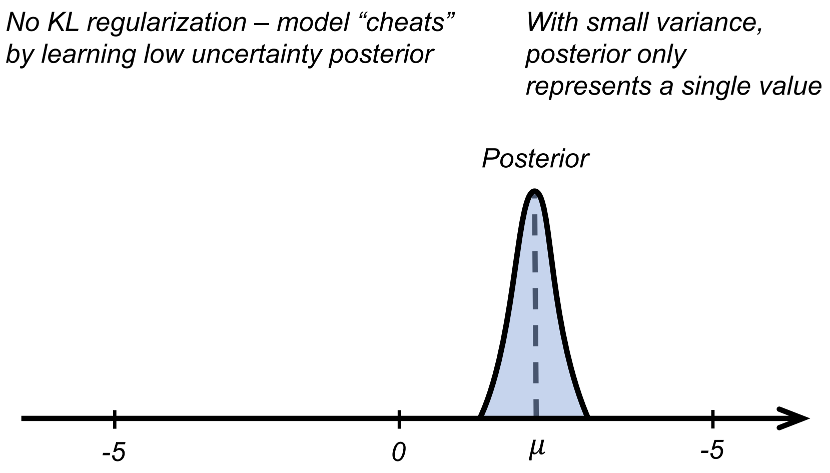

No KL regularization – model “cheats”
by learning low uncertainty posterior
With small variance,
posterior only
represents a single value
Posterior
-5
0
μ
-5

Figure 5.12: Posterior distribution in case: no KL divergence applied

It can be viewed as a coefficient which balances out the magnitude of the gradients [4] during backpropagation (BP) from the two terms in the loss function, which are the log-likelihood expectation (by sampling from the posterior distribution) and the KL-term in order to match prior and posterior. Thus, When $\beta > 1$, the VAE model enforces a stronger constraint on the latent representation bottleneck to keep the distributions similar. Moreover, the effect of β also depends on the dimensionality d of our latent representation $\mathbf{z}$. Larger d also results in the need for larger β values. β limits the ‘representation capacity‘ $\mathbf{z}$ of the bottleneck without changing the capacity in terms of the number of existing latent variable units z_i and thus the VAE architecture remains physically same. When β is to low (AE case) or too high the model‘s learned latent representation becomes entangled and uninterpretable. In one case, when β is very low, there is just no incentive to use the capacity effectively because enough capacity is given, In the other case of high β, there is just too little capacity available to learn anything and the model decides to create an entangled latent representation $\mathbf{z}$. An appropriate sized and tuned β value encourages the model to more efficiently encode and structure necessary information by making better use of the available capacity in the bottleneck. This encourages disentanglement, where latent variables are factorized and therefore uncorrelated and independent from each other.

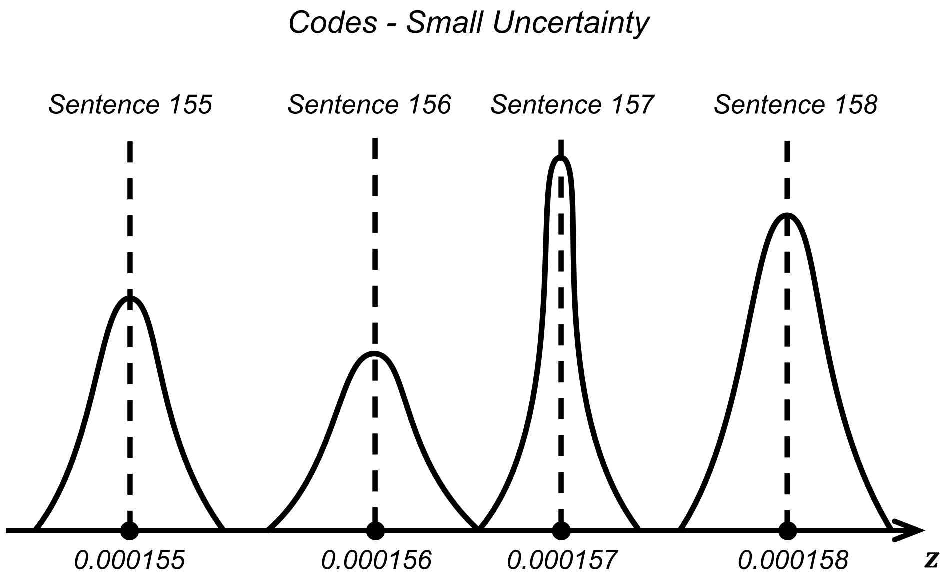


Figure 5.13: Latent code of β-VAE with not too large regularizer, small uncertainty

This is effect is shown in Figure 5.13. The only drawback, that comes with too high β values is that we run into a trade-off between reconstruction quality (in effect the NLL term in the ELBO) and successfully disentangling the latent variables (in effect the KL divergence term in the ELBO). We will later show, that this is not necessarily always the case and might well depend on the type and features of the latent space. The higher the bottleneck capacity C, the less blurry the output of an image is. Still, it remains open, what a ‘blurry‘ reconstruction might mean in the case of other data domains such as text and audio. When the hyperparamter gets even higher, which leads to larger variances and thus higher code uncertainty, the structure of the latent variables must be well organized, such that neighbouring samples are similar to each other. This results in a smooth latent space, which is shown in the Figure 5.14 below. In this case, we really want that sentences which are similar or closely related to each other also be located in that property within the latent space.

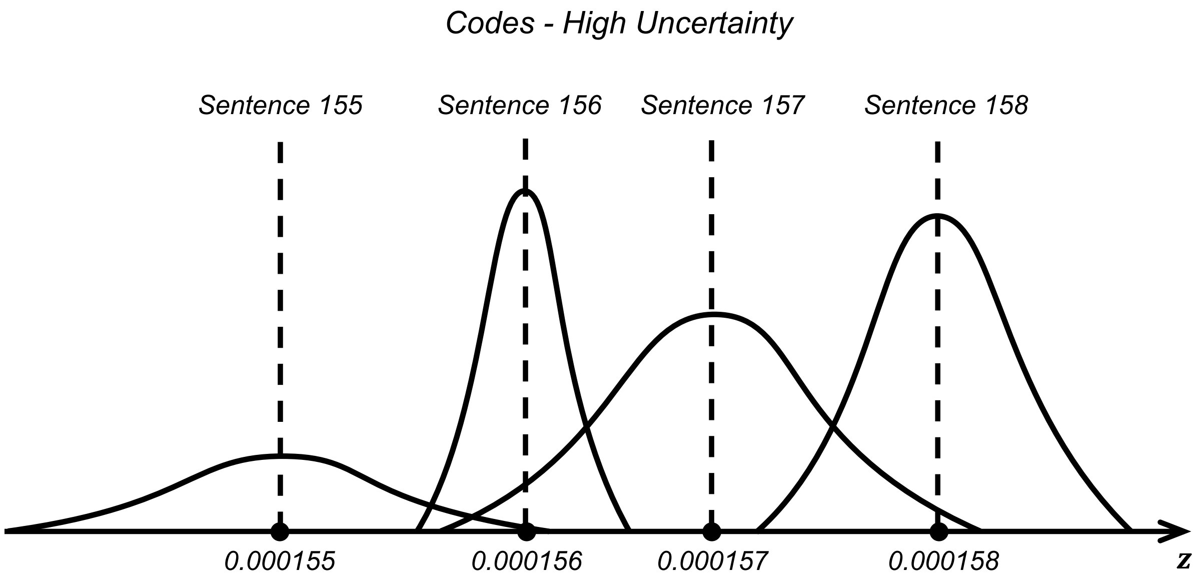


Figure 5.14: Latent code of β-VAE with not large regularizer, high uncertainty

Finally, good reconstruction results are not an indicator for a highly disentangled latent representation. Perfect reconstruction can be achieved with an AE, as its objective is to learn the identity function.

5.5 Discrete β-VAE with flexible latent sizes

For some data distributions, categorical variables, in effect a discrete latent representation, would be better suited. This might be for the identity of a digit in the MNIST dataset but also for text in natural language which consists of a sequence of discrete tokens.

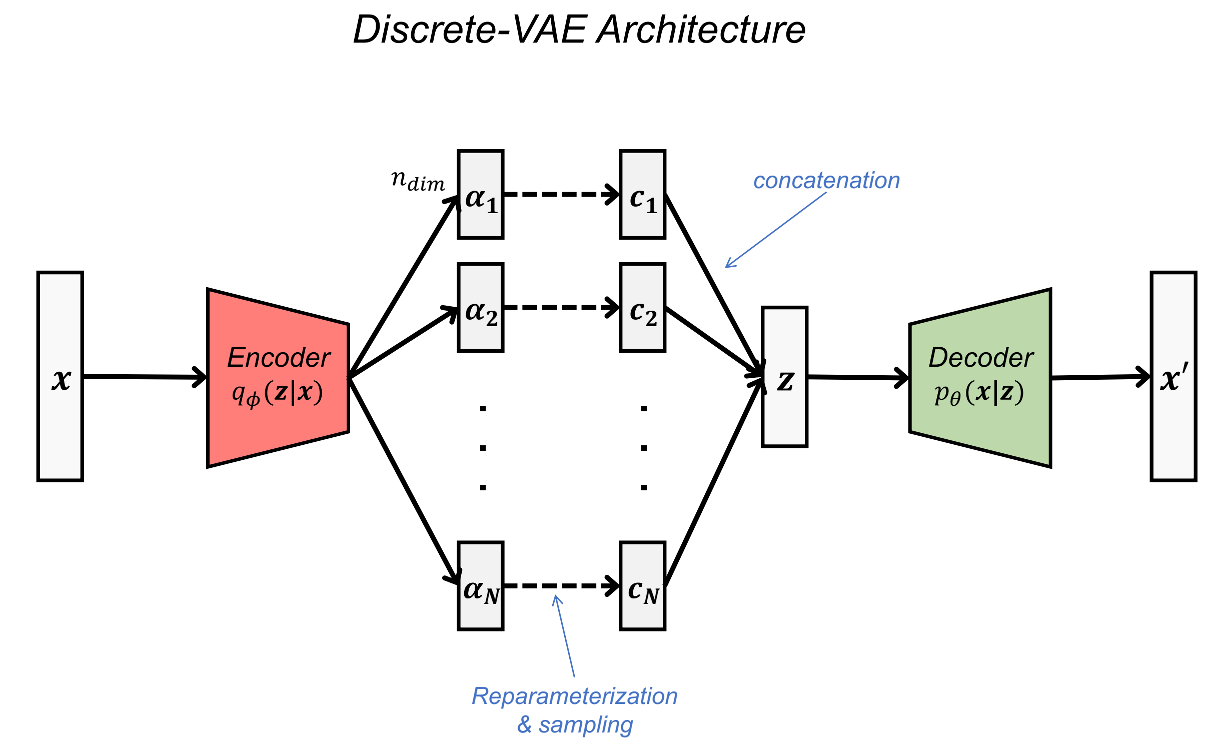


Figure 5.15: Our implemented architecture of the fully discrete β-VAE

Figure 5.15 gives an shows how we have constructed the fully discrete VAE. If we want to apply the VAE with a discrete latent space in a similar way as the continuous latent space VAE models, we clearly need to solve the problem of being able to backpropagate through the discrete samples in the bottleneck $\mathbf{c}$. We will explain how this is done using the Gumbel-Softmax trick for sampling and reparameterization of the latent representation $\mathbf{c}$. In the discrete-β-VAE categorical variables with n categories are represented as one-hot vectors in an n-dimensional space. In this section we switch the notation of our bottleneck from $\mathbf{z}$ to $\mathbf{c}$ because the presented VAE architecture consist of a fully discrete latent vector with flexible dimension size in each latent variable c_i. Looking at what source in VAEs enforces the probability distribution of latent variables, it becomes quite clear, that in order to train a VAE with discrete latent space we have to enforce that with a discrete prior distribution. For that, we are choosing the prior to be a *categorical distribution* denoted with $p_\theta(\mathbf{c})$. The β-VAE objective thus becomes

$$\begin{aligned}\mathcal{L}_{discrete-\beta-VAE}(\phi,\theta,\beta,\mathbf{x},\mathbf{c}) &= -ELBO \\ &= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{c}|\mathbf{x})}[\,log\, p_\theta(\mathbf{x}|\mathbf{c})\,] - \beta D_{KL}(q_\phi(\mathbf{c}|\mathbf{x})\,||\,p_\theta(\mathbf{c}))\end{aligned} \tag{5.17}$$

The KL divergence term between the categorical latent variable distribution $q_\phi(\mathbf{c}|\mathbf{x})$ of the inference network and a uniform categorical prior $p_\theta(\mathbf{c})$ is bounded. Therefore, we have the nice property of a limited and not diverging KL term for the discrete β-VAE setup as opposed to the KL-term for the vanilla VAE setup which has no limit and can cause issues in getting a VAE to initially train. We denote the distribution of the discrete posterior as $P := q_\phi(\mathbf{c}|\mathbf{x})$ and similarly denoting the discrete categorical prior distribution $Q := p_\theta(\mathbf{c})$. We can use the analytical expression of the KL divergence and evaluate the KL divergence for the discrete latent space $\boldsymbol{C}$ as follows

$$\begin{aligned} D_{KL}(P\,||\,Q) &= \sum_{i=1}^{n} p_i \, log \, \frac{p_i}{q_i} \\ &= \sum_{i=1}^{n} p_i \, log \, \frac{p_i}{1/n} \\ &= -H(P) + log(n) \leq log(n) \end{aligned} \tag{5.18}$$

Where n denotes the number of categories a single latent variable vector code can take on. We can think of it as the dimension a *one-hot* vector would need to encode n different categories or classes. Line to follows from the fact, that any uniform categorical distribution Q with n different elements has a limited entropy of $H(Q) = log(n)$. This is because if Q has n different classes, then the entropy is calculated as follows

$$\begin{aligned} H(Q) &= -\sum_{i=1}^{n} Q_i \, log \, Q_i \\ &= -\sum_{i=1}^{n} \frac{1}{n} \, log \, \frac{1}{n} \\ &= -n \, \frac{1}{n} \, (log(1) - log(n)) \\ &= log(n) \end{aligned} \tag{5.19}$$

Line two follows due to the fact, that for any uniform categorical distribution with n different classes the probability of any class is given by $Q_i = 1/n$. Given the case, that we have several discrete latent variable vectors $\mathbf{c^{(i)}}$, which can be of different dimensions $d^{(i)}$ (in effect take on a different amount of categorical values), then we can retrieve the final latent representation by concatenating all single categorical latent vectors into a single latent vector $\mathbf{c}$.

5.5.1 Reparameterization trick - categorically distributed prior

The novelty in the discrete-β-VAE is the fact, that our bottleneck is discrete. This means, as we have another prior distribution we definitely must sample from another noise source than a Gaussian normal distribution as we have done in for the vanilla VAE models. Moreover, we need to come up with a different way of sampling one-hot vectors for the latent space representation $\mathbf{c}$. We cannot parameterize the posterior with a categorical distribution because we need to consider that we want to train the VAE which employs backpropagation which generally only works for continuous valued nodes in the computation graph. The remedy in this case is to make use of the Gumbel-Softmax trick during reparameterization. It solves the difficulty in sampling from the categorical probability distribution $q_\phi(\mathbf{c}|\mathbf{x})$. We can think of $\mathbf{c}$ as a set categorical variables c_i where each category has a specific class probability α_i. The Gumbel-Softmax distribution is reparameterizable and therefore it allows to circumvent the stochastic node which we would have in a normal and direct sampling process. Therefore, we can again apply backpropagation solely through deterministic computation nodes. This reparameterization trick can smoothely deform the posterior during training into a categorical distribution. Drawing samples from the categorical posterior distribution $q_\phi(\mathbf{c}|\mathbf{x})$ with class probabilities α_i can be achieved as follows

$$\mathbf{c} = OneHot\left(\arg\max_i \left[g_i + \alpha_i\right]\right) \tag{5.20}$$

Where $\mathbf{g}$ is a vector of i.i.d sampled Gumbel noise values g_i from the Gumbel distribution $G(0, 1)$ which is given by

$$g = -\,log(-log(u)) \tag{5.21}$$

where u is sampled from

$$u \sim Uniform(0, 1) \tag{5.22}$$

Now, ones we have our class probabilities α_i and a set of corresponding Gumbel noise values g_i, we can use the softmax function as an alternative to $arg\,max$ in order to retrieve a continuous and differentiable estimation for the categorically distributed latent vector sample $\mathbf{c}$. Therefore, we can instead generate such 'soft' one-hot samples using the following equation

$$c_i = \frac{exp((log(\alpha_i) + g_i)/\tau)}{\sum_{j=1}^{n} exp((log(\alpha_j) + g_j)/\tau)} \tag{5.23}$$

Where n denotes the dimension of the categorical sample $\mathbf{c}$, which is a one-hot vector and thus can represent exactly n different categories.

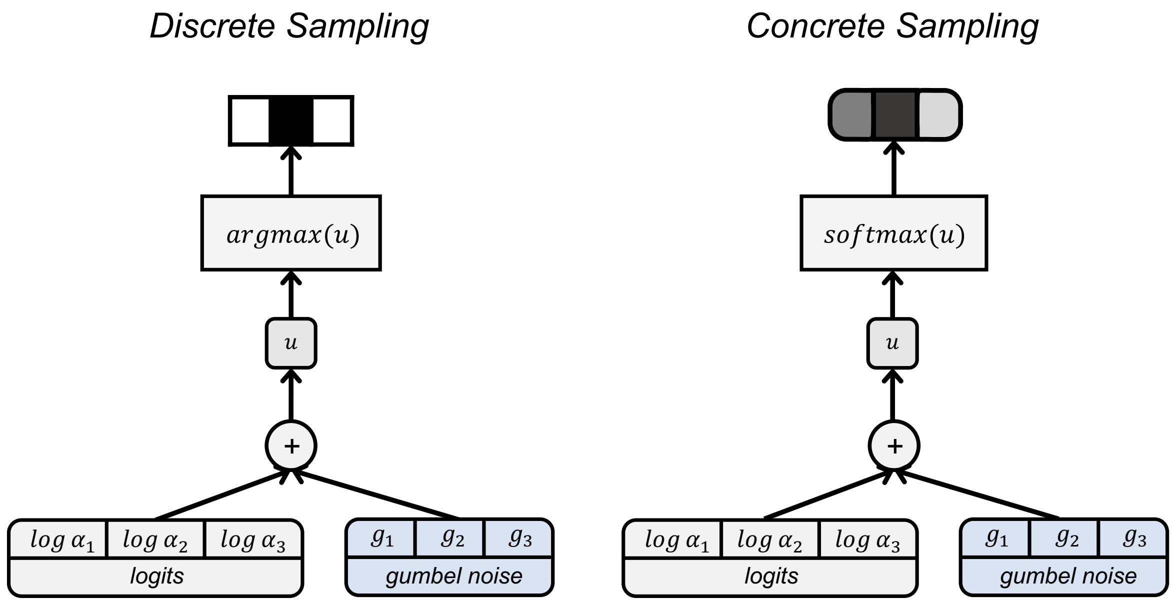


Figure 5.16: Discrete sampling for testting mode and conrecte sampling for training mode. Purple colour represents stochastic nodes and square boxes denotes discrete valued where as round denotes continuous valued

This process is shown in 5.16. The distribution from which we can sample these categorical latent vectors is termed as *Gumbel-Softmax* distribution. Moreover, τ denotes a temperature value, an additional parameter which can be used to control how close these 'soft' one-hot sampled vectors get to the actual categorical distribution, which would then be a 'hard' one-hot sampled vector. When $\tau \to 0$, the outputs of this Gumbel-Softmax distribution become real one-hot vectors and therefore there is no difference between applying the argmax operator directly. The temperature value during training phase starts from a larger value, in order to allow the flow of gradients and thus information for updating weights and biases past the sampling process back to the encoder network. Ones the training progresses, we decrease the temperature parameter τ to make the 'soft' one-hot samples from Gumbel-Softmax distribution become more and more distinct. Still, the temperature value is never annealed down to zero, as this would result in not being able to apply backpropagation and on the other hand the temperature annealing must be limited as too small values do lead to exploding gradient values.

This categorical but continuous distribution is called as Gumbel-Softmax distribution.

The prior in the discrete VAE model $p_\theta(\mathbf{c})$ is equal to the product of uniform Gumbel-Softmax distributions and the categorical posterior distribution $q_\phi(\mathbf{c}|\mathbf{x})$ is calculated as the product of independent Gumbel-Softmax distributions such that $q_\phi(\mathbf{c}|\mathbf{x}) = \prod_i q_\phi(\mathbf{c_i}|\mathbf{x})$. Therefore, each dimension in the posterior $q_\phi(\mathbf{c_i}|\mathbf{x})$ is a Gumbel-Softmax distribution itself with $q_\phi(\mathbf{c_i}|\mathbf{x}) = g(\boldsymbol{\alpha^{(i)}})$

To sum it up, the reparameterization trick in case of categorically distributed prior works as follows

1. Sample Gumbel noise $\mathbf{g}$ from the Gumbel distribution $G(0, 1)$
2. Apply the softmax function onto the logits vector in order to obtain a probability vector $\boldsymbol{\alpha}$ of class probabilities α_i
3. Finally add the terms from step 1. and step 2. and apply the softmax function to retrieve a 'soft' one-hot latent representation vector $\mathbf{c_i}$

To make the comparison clear we want to show the process of sampling using the VAE architecture as presented in 5.17.

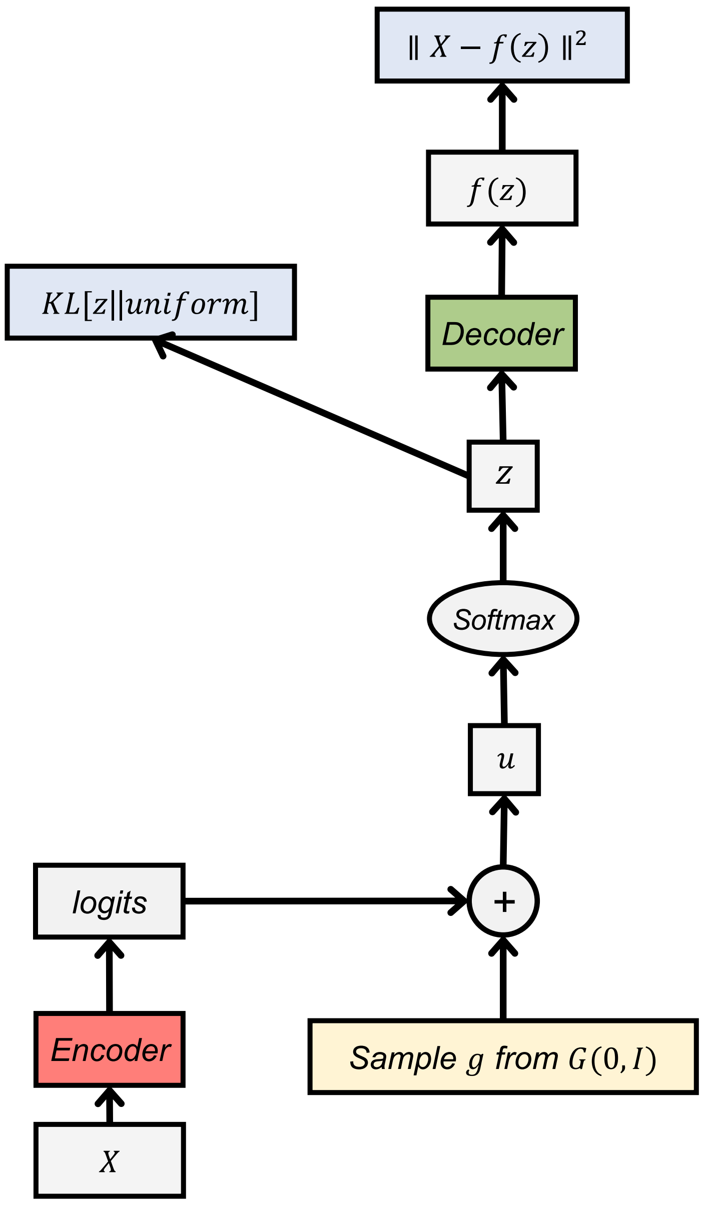


Figure 5.17: Gumbel Softmax trick within the VAE

For the vanilla VAE with continuous latent space we sample latent variables z_i as follows

$$z_i \sim \mathcal{N}(\mu_i, \sigma_i^2) \tag{5.24}$$

whereas for the discrete VAE with a categorical latent space we sample latent variables c_i