Separation of Foreground and Background Signal in Variational Autoencoders

Albert-Ludwigs-Universität Freiburg

Florian Eichin

02.12.1995

Abstract

Abstract goes here

Dedication

To mum and dad

Declaration

I declare that..

Acknowledgements

I want to thank...

Contents

Chapter 1

Introduction

Chapter 2

Technical Part

For this purpose, many ideas of the following chapter are taken from (XX) and the notation mainly follows the same logic.

2.1 Inference Problem Setup

Before we dive into the technical questions of this thesis, we want to begin with a discussion of the problem we attempt to solve formally. Let $\mathbf{x}, \mathbf{z}$ be random variables with $\mathbf{x}$ observable and $\mathbf{z} \in \mathbb{R}^k$ hidden. Then we are interested in the *latent variable model* with model parameters θ^*

$$p_{\theta^*}(\mathbf{x}, \mathbf{z}) = p_{\theta^*}(\mathbf{x}|\mathbf{z})p_{\theta^*}(\mathbf{z}) \tag{2.1}$$

We further assume that prior $p_{\theta^*}(\mathbf{z})$ and $p_{\theta^*}(\mathbf{x}|\mathbf{z})$ are from parametric families of distributions $p_\theta(\mathbf{z})$ and $p_\theta(\mathbf{x}|\mathbf{z})$ and that they have probability density functions that are differentiable with respect to θ and $\mathbf{z}$ almost everywhere.

To make things clearer, we can look at it in a more practical way: Let $\mathbf{X} = \{\mathbf{x}^{(i)}\}_{i=1}^N$ be a dataset with N i.i.d. samples of our random variable $\mathbf{x}$. Note, that $\mathbf{x}$ can be a vector of arbitrary dimension encoding all kinds of data such as images, soundwaves etc. If we model our data with the above latent variable model, we suppose the datapoints to be generated with the involvement of $\mathbf{z}$ in the sense, that first a value $\mathbf{z}^{(i)}$ is generated from prior distribution $p_{\theta^*}(\mathbf{z})$ and in the second step, $\mathbf{x}^{(i)}$ is generated from $p_{\theta^*}(\mathbf{x}|\mathbf{z}^{(\mathbf{i})})$. Usually, $\mathbf{z}$ is assumed to have a much lower dimension and a much simpler distribution than $\mathbf{x}$. Therefore, the $\mathbf{z}$-space can be viewed as a space of encodings, where only relevant information for decoding datapoints into the high-dimensional $\mathbf{x}$-space is retained. This is why, from a machine learning perspective, we refer to

can identify this model with an dimensionality reduction problem: For our data, we want to learn functions $q_\phi : \mathcal{X} \to \mathcal{Z}$, $p_\theta : \mathcal{Z} \to \mathcal{X}$ such that for our data $\mathbf{X} \approx p_\theta(q_\phi(\mathbf{X}))$, constrained by the low dimension and regularization of (Z).

This is interesting for us, as we're not only interested in approximations of the posterior inference of $\mathbf{z}$ given a value of $\mathbf{x}$ but also the ...

For a given dataset, there is different approaches for the above scenario. However, we do make additional assumptions, that narrow the list of efficient algorithms significantly [XX]:

1 *Intractability*: the integral of the marginal likelihood $p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{z}) p_\theta(\mathbf{x}|\mathbf{z}) d\mathbf{z}$, as well as posterior distribution $p_\theta(\mathbf{z}|\mathbf{x}) = p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z}) / p_\theta(\mathbf{x})$ are intractable.

2 *Big dataset*: Batch optimization is too expensive and parameter updates on small minibatches preferable. Sampling-based solutions would be too inefficient [XX].

2.2 Variational Inference

In many probabilistic models inference is intractable and approximation methods are needed. One way of approximating solutions to inference problems is to describe it as an optimization problem. Algorithms involving this approach are called *variational.* Given an intractable probability distribution p^* and a set of tractable distributions $\mathcal{Q}$, the goal of a variational algorithm is to find $q \in \mathcal{Q}$ that is most 'similar' to p. Finding such a q usually involves a complex optimization procedure for an optimization target $J(q)$. Subsequently, we can use q instead of p in order to find approximate solutions to inference problems efficiently.
Of course, this rather informal description on variational techniques leaves us with questions. What is the similarity of two distributions q and p? How do we choose an according optimization objective $J(q)$? What are good ways of formulating tractable class of distributions $\mathcal{Q}$?
The (partial) answering to these four questions will be the main motivation for the following sections in order to lay the groundwork for the introduction of the Variational Autoencoder (VAE). Inheriting it's name from *Au-*

toencoders[XX], a VAE is a probabilistic model designed for learning latent representations with the help of Artificial Neuronal Networks (ANNs) and a variational optimization approach to the approximation of the encoder distribution.

2.3 Kullback-Leibler divergence

Beggining with our first question, there is a way of quantifying the 'similarity' of two distributions $p(\mathbf{x})$ and $q(\mathbf{x})$ in information theory known as the *Kullback-Leibler (KL) divergence.* For $p(\mathbf{x}), q(\mathbf{x})$ continuous, the KL divergence is defined as

$$KL(q(\mathbf{x})||p(\mathbf{x})) = \int_{-\infty}^{\infty} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{p(\mathbf{x})} d\mathbf{x} = \mathbb{E}_{q(\mathbf{x})} \left[\log \left(\frac{q(\mathbf{x})}{p(\mathbf{x})} \right) \right] \tag{2.2}$$

In the discrete case, it is analogously

$$KL(q(\mathbf{x})||p(\mathbf{x})) = \sum_{\mathbf{x}} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{p(\mathbf{x})} = \mathbb{E}_{q(\mathbf{x})} \left[\log \left(\frac{q(\mathbf{x})}{p(\mathbf{x})} \right) \right] \tag{2.3}$$

Here, log is an abbreviation for the logarithm to base e, the natural logarithm. Note, that for any $q(\mathbf{x}), p(\mathbf{x})$ (continuous or discrete) we can deduce the following properties:

- $KL(q||p) \geq 0$
- $KL(q||p) = 0$ if and only if $q = p$ (QUESTION: Do we need this???)

For a proof, we can consider $q(\mathbf{x}), p(\mathbf{x})$ to be continuous (the discrete case follows analofgously). With $1 - r \leq -\log(r)$ we have:

$$\begin{aligned} KL(q(\mathbf{x})||p(\mathbf{x})) &= \int_{\mathbf{x}} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{p(\mathbf{x})} d\mathbf{x} \\ &= \int_{\mathbf{x}} q(\mathbf{x})(-\log \frac{p(\mathbf{x})}{q(\mathbf{x})}) d\mathbf{x} \\ &\geq \int_{\mathbf{x}} q(\mathbf{x})(1 - \frac{p(\mathbf{x})}{q(\mathbf{x})}) d\mathbf{x} \\ &= \int_{\mathbf{x}} q(\mathbf{x}) d\mathbf{x} - \int_{\mathbf{x}} q(\mathbf{x}) \frac{p(\mathbf{x})}{q(\mathbf{x})} d\mathbf{x} = 0 \end{aligned} \tag{2.4}$$

And '=' $\Leftrightarrow \frac{p(\mathbf{x})}{q(\mathbf{x})} = 1 \Leftrightarrow p(\mathbf{x}) = q(\mathbf{x})$.

2.4 Variational Lower Bound (ELBO)

As we discussed previously, $p_\theta(\mathbf{x})$ as well as $p_\theta(\mathbf{z}|\mathbf{x})$ are supposed to be intractable in our problem setup. We have thus no way of retrieving either of the two out of the other. This is where the variational component from two sections before comes into play. In order to approximate $p_\theta(\mathbf{z}|\mathbf{x})$ we introduce a tractable *parametric inference model* $q_\phi(\mathbf{z}|\mathbf{x})$. We will optimize the so called *variational parameters* $\boldsymbol{\phi}$ of this model such that $p_\theta(\mathbf{z}|\mathbf{x}) \approx q_\phi(\mathbf{z}|\mathbf{x})$. Derived from Bayes' rule, we also have

$$p_\theta(\mathbf{x}) = \frac{p_\theta(\mathbf{z}|\mathbf{x})p_\theta(\mathbf{z})}{p_\theta(\mathbf{x}|\mathbf{z})} \approx \frac{p_\theta(\mathbf{z}|\mathbf{x})p_\theta(\mathbf{z})}{q_\phi(\mathbf{x}|\mathbf{z})} \tag{2.5}$$

It is clear, that for our model to fit the true distribution of our data well, we are interested in the following two things:

1. Maximization of the marginal likelihood $p_\theta(\mathbf{x})$ for our data to improve our generative model.
2. Minimization of the KL divergence between $p_\theta(\mathbf{x})$ and $q_\phi(\mathbf{x})$ to improve the approximation of $q_\phi(\mathbf{x})$.

Since log is monotonous, maximizing $p_\theta(\mathbf{x})$ is equivalent to maximizing $\log p_\theta(\mathbf{x})$. For an arbitrary choice of $q_\phi(\mathbf{z}|\mathbf{x})$ we can consider the following derivation:

$$\begin{aligned}
\log p_\theta(\mathbf{x}) &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x})\right] \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x},\mathbf{z})}{p_\theta(\mathbf{z}|\mathbf{x})}\right] \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \left(\frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})}\right)\right] \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \left(\frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right)\right] + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \left(\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})}\right)\right] \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \left(\frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right)\right] + KL(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x}))
\end{aligned} \tag{2.6}$$

Where the right term in the last row is the KL divergence of $q_\phi(\mathbf{z}|\mathbf{x})$ and $p_\theta(\mathbf{x},\mathbf{z})$. If we rearrange the equation, we have the following:

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

And since $KL(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{x},\mathbf{z})) \geq 0$, the right hand side is a lower bound for $\log p_\theta(\mathbf{x})$. It is also referred to as *variational lower bound* or *evidence lower bound* (ELBO):

$$\begin{aligned} \mathcal{L}_{\theta,\phi}(\mathbf{x}) &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log\left(\frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right)\right] \\ &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right] \end{aligned} \tag{2.8}$$

The ELBO can be rewritten in the following way:

$$\begin{aligned} &\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log\left(\frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right)\right] \\ &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log\left(\frac{p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right)\right] \\ &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z}) - \log\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z})}\right] \\ &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] - \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z})}\right] \\ &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] + KL(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})) \end{aligned} \tag{2.9}$$

With the above derivation in mind, we can identify another interpretation of $KL(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{x}|\mathbf{z}))$ besides being the KL divergence of approximate posterior $q_\phi(\mathbf{z}|\mathbf{x})$ and true posterior $p_\theta(\mathbf{x}|\mathbf{z})$): It is also the gap between ELBO $\mathcal{L}_{\theta,\phi}(\mathbf{x})$ and $\log p_\theta(\mathbf{x})$. If $q_\phi(\mathbf{z}|\mathbf{x})$ approximates the true $p_\theta(\mathbf{z}|\mathbf{x})$ 'better', the gap gets smaller.

Let $\mathbf{X}$ be the dataset of i.i.d. samples from before and $N_\mathbf{X} = |\mathbf{X}|$. If we want to fit our model on $\mathbf{X}$, the ELBO yields us an optimization objective we were asking for, namely the average of ELBOs of single datapoints $\mathbf{x} \in \mathbf{X}$:

$$\mathcal{L}_{\theta,\phi}(\mathbf{X}) = \sum_{\mathbf{x}\in\mathbf{X}} \frac{\mathcal{L}_{\theta,\phi}(\mathbf{x})}{N_\mathbf{X}} \tag{2.10}$$

If we maximize $\mathcal{L}_{\theta,\phi}(\mathbf{x})$ with respect to parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ for our data, we will approximately maximize $p_\theta(\mathbf{x})$ and minimize $KL(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x}))$ just like the goals we formulated in the beginning of this section.

2.5 Auto-encoding Variational Bayes

2.5.1 Batch Gradient Descent

With the means of the ELBO, we now have an objective to optimize the model parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ for. A naive solution, also known as *Batch Gradient Descent*, is to initialize the parameters randomly, to estimate the gradients $\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{X})$ and $\nabla_{\boldsymbol{\phi}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{X})$ and adjust $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ into their respective directions until convergence. With each step of adjusting the parameters for the gradients, also called an *epoch*, we expect $\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{X})$ to improve until we have reached a local maximum and the algorithm converges. It is up to implementation how to detect convergence of the algorithm. Typically, one can define criteria such as a threshold for change of loss after a certain number of epochs epoch. If the change of loss is lower than this set threshhold, we abort the procedure. Of course, there is other, more complex criteria. Sometimes the user decides theirselves, when to stop the algorithm due to the tradeoff between runtime and optimality.

Algorithm 1: Batch Gradient Descent

```
while not converged do
    Estimate gradients $\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{M})$, $\nabla_{\boldsymbol{\phi}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{M})$
    Update parameters $\boldsymbol{\theta} \to \boldsymbol{\theta} + \eta\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x})$, $\boldsymbol{\phi} \to \boldsymbol{\phi} + \eta\nabla_{\boldsymbol{\phi}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x})$
end
```

Note, that η is a hyperparameter, that determines the extent to which the gradients update the parameters for each epoch. It is therefore also called the *learning rate* and is an important parameter to choose as it highly affects the convergence of the algorithm: If we choose η too small, we will only move slowly in the 'preferred' direction of local maxima. However, if we choose η too big, our updating-steps get too large and the algorithm cannot converge in the worst case.

2.5.2 Estimation of the gradients and ELBO

Because in general analytical computations of the gradients of the ELBO are intractable, we have to estimate them. For $\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{X})$, it is sufficient to estimate $\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x})$ for each $\mathbf{x} \in \mathbf{X}$ because since (XX) we have

$$\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{X}) = \nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} \sum_{\mathbf{x}\in\mathbf{X}} \frac{\mathcal{L}_{\theta,\phi}(\mathbf{x})}{N_{\mathbf{X}}} = \frac{1}{N_{\mathbf{X}}} \sum_{\mathbf{x}\in\mathbf{X}} \nabla_{\boldsymbol{\theta},\boldsymbol{\phi}}\mathcal{L}_{\theta,\phi}(\mathbf{x}) \tag{2.11}$$

For gradients of the ELBO with respect to generative model parameters $\boldsymbol{\theta}$, we can obtain an unbiased Monte Carlo estimator (unbiased estimation will from now on be illustrated by '$\simeq$'):

$$\begin{aligned}\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) &\overset{(XX)}{=} \nabla_{\boldsymbol{\theta}}\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right] \\ &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\nabla_{\boldsymbol{\theta}}(\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x}))\right] \\ &\simeq \nabla_{\boldsymbol{\theta}}(\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})) \\ &= \nabla_{\boldsymbol{\theta}} \log p_\theta(\mathbf{x},\mathbf{z})\end{aligned} \tag{2.12}$$

With the $\mathbf{z}$ in the last two lines a random sample from $q_\phi(\mathbf{z}|\mathbf{x})$. For unbiased gradients with respect to $\boldsymbol{\phi}$, things are a bit more difficult. This is due to the fact, that in general, we have:

$$\begin{aligned}\nabla_{\boldsymbol{\phi}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) &= \nabla_{\boldsymbol{\phi}}\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right] \\ &\neq \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\nabla_{\boldsymbol{\phi}}(\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}))\right]\end{aligned} \tag{2.13}$$

Moreover, we will thus assume $\mathbf{z}$ to be continuous and $q_\phi(\mathbf{z}|\mathbf{x})$ and $p_\theta(\mathbf{x}|\mathbf{z})$ differentiable. For the case of the Variational Autoencoder (and other instances of our problem), we are not constrained by this. However, we can now reparameterize $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$ to circumvent the problem of obtaining $\nabla_{\boldsymbol{\phi}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x})$. We can choose f as some invertible, differentiable transformation and introduce another random variable $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$ independent of $\boldsymbol{\phi}$, $\boldsymbol{\theta}$ and $\mathbf{x}$ such that

$$\mathbf{z} = f(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \mathbf{x}) \tag{2.14}$$

This is also referred to as *reparameterization trick*. Under reparameterization, the ELBO can be rewritten:

$$\begin{aligned}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right] \\ &= \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\log p_\theta(\mathbf{x}, f(\boldsymbol{\epsilon},\boldsymbol{\phi},\mathbf{x})) - \log q_\phi(f(\boldsymbol{\epsilon},\boldsymbol{\phi},\mathbf{x})|\mathbf{x})\right] \\ &\simeq \log p_\theta(\mathbf{x}, f(\boldsymbol{\epsilon},\boldsymbol{\phi},\mathbf{x})) - \log q_\phi(f(\boldsymbol{\epsilon},\boldsymbol{\phi},\mathbf{x})|\mathbf{x}) \\ &=: \tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x},\boldsymbol{\epsilon})\end{aligned} \tag{2.15}$$

Where we used a single sample $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$ to derive the Monte Carlo estimator $\tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x},\boldsymbol{\epsilon})$. We can rewrite our gradients and gain an estimator in a similar fashion:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) &= \nabla_{\boldsymbol{\theta},\boldsymbol{\phi}}\mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\log p_\theta(\mathbf{x}, f(\boldsymbol{\epsilon},\boldsymbol{\phi},\mathbf{x})) - \log q_\phi(f(\boldsymbol{\epsilon},\boldsymbol{\phi},\mathbf{x})|\mathbf{x})\right] \\
&= \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}}(\log p_\theta(\mathbf{x}, f(\boldsymbol{\epsilon},\boldsymbol{\phi},\mathbf{x})) - \log q_\phi(f(\boldsymbol{\epsilon},\boldsymbol{\phi},\mathbf{x})|\mathbf{x}))\right] \\
&\simeq \nabla_{\boldsymbol{\theta},\boldsymbol{\phi}}(\log p_\theta(\mathbf{x}, f(\boldsymbol{\epsilon},\boldsymbol{\phi},\mathbf{x})) - \log q_\phi(f(\boldsymbol{\epsilon},\boldsymbol{\phi},\mathbf{x})|\mathbf{x})) \\
&= \nabla_{\boldsymbol{\theta},\boldsymbol{\phi}}\tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x},\boldsymbol{\epsilon})
\end{aligned} \tag{2.16}$$

Again with the last two rows a Monte Carlo estimator for a single sample $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$. Both estimators are unbiased because:

$$\begin{aligned}
\mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}}\tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x},\boldsymbol{\epsilon})\right] &= \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\nabla_{\boldsymbol{\phi},\boldsymbol{\theta}}\log p_\theta(\mathbf{x}, f(\boldsymbol{\epsilon},\boldsymbol{\phi},\mathbf{x})) - \log q_\phi(f(\boldsymbol{\epsilon},\boldsymbol{\phi},\mathbf{x})|\mathbf{x})\right] \\
&= \nabla_{\boldsymbol{\phi},\boldsymbol{\theta}}\mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\log p_\theta(\mathbf{x}, f(\boldsymbol{\epsilon},\boldsymbol{\phi},\mathbf{x})) - \log q_\phi(f(\boldsymbol{\epsilon},\boldsymbol{\phi},\mathbf{x})|\mathbf{x})\right] \\
&= \nabla_{\boldsymbol{\phi},\boldsymbol{\theta}}\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right] \\
&= \nabla_{\boldsymbol{\theta},\boldsymbol{\phi}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x})
\end{aligned} \tag{2.17}$$

Which works with an analog argument for $\tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x},\boldsymbol{\epsilon})$. In order to compute this estimation of the ELBO and its gradients, we will need to compute $\log q_\phi(\mathbf{z}|\mathbf{x}) = \log q_\phi(f(\boldsymbol{\epsilon},\boldsymbol{\phi},\mathbf{x})|\mathbf{x})$. We will use a result from Statistics, called the *Change-of-Variables Technique* [XX], which yields us

$$q_\phi(f(\boldsymbol{\epsilon},\boldsymbol{\phi},\mathbf{x})|\mathbf{x})\left|\det\left(\frac{\partial}{\partial\boldsymbol{\epsilon}}f(\boldsymbol{\epsilon},\boldsymbol{\phi},\mathbf{x})\right)\right| = p(\boldsymbol{\epsilon}) \tag{2.18}$$

Where $\left|\det\left(\frac{\partial}{\partial\boldsymbol{\epsilon}}f(\boldsymbol{\epsilon},\boldsymbol{\phi},\mathbf{x})\right)\right|$ is the absolute value of the determinant of the Jacobian. Putting it together, we thus have:

$$\log q_\phi(f(\boldsymbol{\epsilon},\boldsymbol{\phi},\mathbf{x})|\mathbf{x}) = \log p(\boldsymbol{\epsilon}) - \log\left|\det\left(\frac{\partial}{\partial\boldsymbol{\epsilon}}f(\boldsymbol{\epsilon},\boldsymbol{\phi},\mathbf{x})\right)\right| \tag{2.19}$$

For many choices of reparameterizations, this is simple to compute, as we will see in the following chapters.

2.5.3 Stochastic Gradient Descent

Vanilla Gradient Descent has several shortcomings, that we still have to address. For one, it is usually very expensive to estimate $\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{X})$ on big datasets $\mathbf{X}$, violating our goal for efficiency with large data. Also, the algorithm usually only finds the closest local maximum and has no means to escape before convergence. There is several solutions to those problems. One that tackles both is called *Minibatch Stochastic Gradient Descent (SGD).* The idea behind Minibatch SGD is again intuitive: Instead of estimating the gradients on the whole dataset $\mathbf{X}$, we randomly draw a subset $\mathbf{M} \subset \mathbf{X}$ of size $N_{\mathbf{M}}$. We call $\mathbf{M}$ a *minibatch*. With $\boldsymbol{\alpha} \in \{\boldsymbol{\theta}, \boldsymbol{\phi}\}$ we have:

$$\nabla_{\boldsymbol{\alpha}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{X}) = \frac{1}{N_{\mathbf{X}}} \sum_{\mathbf{x} \in \mathbf{X}} \nabla_{\boldsymbol{\alpha}} \mathcal{L}_{\theta,\phi}(\mathbf{x}) \simeq \frac{1}{N_{\mathbf{M}}} \sum_{\mathbf{x} \in \mathbf{M}} \nabla_{\boldsymbol{\alpha}} \mathcal{L}_{\theta,\phi}(\mathbf{x}) = \nabla_{\boldsymbol{\alpha}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{M}) \tag{2.20}$$

Assume $\mathbf{M'} = \{(\mathbf{x}, \boldsymbol{\epsilon}) | \mathbf{x} \in \mathbf{M}, \boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})\}$ to be a set of tuples of each datapoint in $\mathbf{M}$ and an according sample of $\boldsymbol{\epsilon}$. For $\boldsymbol{\alpha} = \boldsymbol{\theta}$, we can get the following unbiased gradient estimator from combining (2.10) with (2.12):

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{M}) \simeq \frac{1}{N_{\mathbf{M}}} \sum_{(\mathbf{x}^{(i)}, \boldsymbol{\epsilon}^{(i)}) \in \mathbf{M'}} \nabla_{\boldsymbol{\theta}} \log p_\theta(\mathbf{x}, f(\boldsymbol{\epsilon}^{(i)}, \boldsymbol{\phi}, \mathbf{x})) \tag{2.21}$$

And for $\boldsymbol{\alpha} = \boldsymbol{\phi}$ with (2.13) we can derive:

$$\nabla_{\boldsymbol{\phi}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{M}) \simeq \frac{1}{N_{\mathbf{M}}} \sum_{(\mathbf{x}^{(i)}, \boldsymbol{\epsilon}^{(i)}) \in \mathbf{M'}} \nabla_{\boldsymbol{\phi}} (\log p_\theta(\mathbf{x}^{(i)}, f(\boldsymbol{\epsilon}^{(i)}, \boldsymbol{\phi}, \mathbf{x})) - \log q_\phi(f(\boldsymbol{\epsilon}^{(i)}, \boldsymbol{\phi}, \mathbf{x}^{(i)}) | \mathbf{x}^{(i)})) \tag{2.22}$$

Usually the minibatch size $N_{\mathbf{M}}$ is set to be much lower than the number of datapoints in our dataset. Depending on application domain and optimization target, different sizes are optimal. For $N_{\mathbf{M}} = 1$ we also have 'normal' *Stochastic Gradient Descent* as an instance of Minibatch SGD. It is easy to see, why Minibatch SGD is a more efficient way of optimizing our model parameters: The cost of estimating our gradients on smaller minibatches is of course much cheaper than on the whole dataset. In the same runtime, Minibatch SGD updates parameters much more often than vanilla Gradient Descent, usually leading to much faster convergence. Besides bare runtime, it also saves memory as we only have to load a small fraction of the dataset in order for the algorithm to work. Also, because we update our parameters on random subsets, the optimization process becomes more 'noisy', allowing the algorithm to escape local maxima by taking globally non-optimal steps. However, this does not solve all the problems with local maxima. Theres more techniques to tackle this problem, but an in-depth discussion would by far escape the scope of this thesis. The interested reader is advised to continue their research in [XX].

2.5.4 Stochastic Optimization of the ELBO

Putting things together, we can now introduce the *Auto-Encoding Variational Bayes* procedure, that utilizes Minibatch SGD and the gradient estimators we have derived.

Algorithm 2: Auto-Encoding Variational Bayes (AEVB)

```
Initialize parameters $\boldsymbol{\theta}$, $\boldsymbol{\phi}$ randomly
while not converged do
    Randomly draw a minibatch $\mathbf{M} \subset \mathbf{X}$
    Sample $\boldsymbol{\epsilon}_1, ..., \boldsymbol{\epsilon}_{N_{\mathbf{M}}} \sim p(\boldsymbol{\epsilon})$
    Compute $\tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{M})$ and estimate gradient $\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} \tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{M})$
    Update parameters $\boldsymbol{\theta}$, $\boldsymbol{\phi}$
end
```

Note, how not only the minibatch drawing of SGD, but also the sampling of $\epsilon \sim p(\epsilon)$ introduces noise to the procedure. This makes it even more robust to getting stuck in local maxima, for the reasosns discussed above. Also, we have replaced the parameter updating in order to account for more complex strategies of different instances of this algorithm. While simply updating the parameters in the direction of the gradients with a set learning rate η is still a viable approach, there is other variations that can lead to better results for different application domains. A more detailed discussion can be found in [XX].

2.6 Artificial Neural Networks

Artificial Neural Networks (ANN) have gained a lot of attention due to their stellar performance in many different domains of machine learning lately. An ANN is a function composed by *layers* $\alpha^{(i)}$:

$$NeuralNet = \alpha^{(L)} \circ \alpha^{(L-1)} \circ ... \circ \alpha^{(1)} \tag{2.23}$$

With L the number of layers. For each $i \in \{2, ..., L\}$ we define *weight matrix* $W^{(i)} \in Mat(M^{(i)} \times N^{(i)}, \mathbb{R})$, *bias vector* $b^{(i)} \in \mathbb{R}^{M^{(i)}}$ and *activation function* $g^{(i)} : \mathbb{R} \to \mathbb{R}$. Layer $\alpha^{(i)}$ maps a vector of *input dimension* $N^{(i)}$ to a vector of *output dimension* $N^{(i+1)}$ as follows:

$$\begin{aligned} \alpha^{(i)} : \mathbb{R}^{N^{(i)}} &\to \mathbb{R}^{N^{(i+1)}} \\ x &\mapsto g^{(i)}(W^{(i)}x + b^{(i)}) \end{aligned} \tag{2.24}$$

These are also called the *hidden layers* of our network and $\alpha^{(L)}$ is the *output layer*. By convention, we define *input layer* $\alpha^{(1)}$ as the identity of the input vector of the network and $N^{(1)}$ as its dimension. Usually the activation functions $g^{(i)}$, which are applied element-wise, are chosen to be non-linear, or else α would just be a linear function and the ANN would deflate into a single linear transformation as well. In order to enable an ANN to 'learn' $\boldsymbol{\phi} = \{W^{(2)}, b^{(2)}, W^{(3)}, b^{(3)}, ..., W^{(L)}, b^{(L)}\}$ are interpreted as parameters, that we can update with respect to their gradients. Therefore, $g^{(i)}$ has to be differentiable (almost) everywhere (we have to define assumed values for the derivation at points where $g^{(i)}$ is non-differntiable). Common activation functions include many others:

- sigmoid function $Sig(x) = \frac{1}{1-\exp(-kx)}$ with $k \in \mathbb{R}$
- rectifier function $ReLu(x) = \max(0, x)$ (with $ReLu'(0) := 0$)
- tangens hyperbolicus $tanh$

And while each has it's own perks, some are better suited for different tasks. For example, in practice, a big advantage of $ReLu$ is the faster convergence of optimization tasks. On the other hand, $tanh$ and historically important Sig sometimes offer better interpretability of the results as they have values in $(0, 1)$.
The gradients with respect to the parameters $\boldsymbol{\phi}$ of a NeuralNetwork can be calculated using a procedure called *Backpropagation*, which exploits the chain rule of differentiation. Another approach that is currently evolving is *differentiable programming*, where the key idea is to compute derivatives for defined functions automatically at compiling time. In the case of neural networks, there is only matrix multiplications, vector additions and the usually simple activation functions which makes them a promising application for differential programming.

2.7 The Variational Auto-encoder (VAE)

2.7.1 Choice of model

The definition of the AEVB algorithm left some options regarding the exact nature of the final model. Namely, we still have to choose:

1) the parameterization of decoder $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$

2) the distribution of prior $p_{\boldsymbol{\theta}}(\mathbf{z})$ and its reparameterization $\mathbf{z} = f(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \mathbf{x})$

3) the parameterization for the variational encoder $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$

We want 1) to be complex enough to model the data sufficiently well. 3) brings us back to our last question from the beginning of this chapter, where we asked for a good way to formulate a the variational class of tractable distributions.

2.7.2 Neural Networks for parameterizing distributions

For the VAE, we let $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ be a multivariate normal Distribution (for continuous data) or Bernoulli (binary data). We can model the parameters $\boldsymbol{\mu}, \boldsymbol{\Sigma}$ or $\mathbf{p}$ for both cases as the outputs of an ANN with parameters $\boldsymbol{\theta}$:

$$\begin{aligned} p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) &= \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})(\mathbf{z}) \text{ or } \mathcal{B}(\mathbf{p})(\mathbf{z}) \\ (\boldsymbol{\mu}, \boldsymbol{\Sigma}) \text{ or } \mathbf{p} &= DecoderANN_{\boldsymbol{\theta}}(\mathbf{x}) \end{aligned} \tag{2.25}$$

Furthermore, we let $p_{\boldsymbol{\theta}}(\mathbf{z}) = \mathcal{N}(0, E)(\mathbf{x})$ with $E = \operatorname{diag}(1)$ be the centered isotropic multivariate normal Distribution. In this case, $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ is intractable and we choose $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2 E)(\mathbf{x})$ assuming the true posterior to approximately be a multivariate normal distribution with diagonal covariance. Again, we can use an ANN to parameterize mean $\boldsymbol{\mu}$ and log-deviation $\log \boldsymbol{\sigma}$ of the variational posterior in the following sense:

$$\begin{aligned} q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) &= \mathcal{N}(\boldsymbol{\mu}, \operatorname{diag}(\boldsymbol{\sigma})^2)(\mathbf{z}) \\ (\boldsymbol{\mu}, \log \boldsymbol{\sigma}) &= NeuralNet_{\boldsymbol{\phi}}(\mathbf{x}) \end{aligned} \tag{2.26}$$

Reparameterization in this case is simple:

$$\begin{aligned} \boldsymbol{\epsilon} &\sim \mathcal{N}(0, E) \\ (\boldsymbol{\mu}, \log \boldsymbol{\sigma}) &= NeuralNet_{\boldsymbol{\phi}}(\mathbf{x}) \\ \mathbf{z} &= f(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \boldsymbol{x}) = \boldsymbol{\mu} + \boldsymbol{\sigma} \cdot \boldsymbol{\epsilon} \end{aligned} \tag{2.27}$$

with $\cdot$ the element-wise multiplication. And the Jacobian from $\boldsymbol{\epsilon}$ to $\mathbf{z} = f(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \boldsymbol{x})$ is:

$$\frac{\partial}{\partial \boldsymbol{\epsilon}} f(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \mathbf{x}) = \frac{\partial \mathbf{z}}{\partial \boldsymbol{\epsilon}} = \operatorname{diag}(\boldsymbol{\sigma}) \tag{2.28}$$

With (XX), we can thus compute the log-posterior density by:

$$\begin{aligned}
\log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) &= \log p(\boldsymbol{\epsilon}) - \log \left| \det \left(\frac{\partial}{\partial \boldsymbol{\epsilon}} f(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \mathbf{x}) \right) \right| \\
&= \log p(\boldsymbol{\epsilon}) - \sum_i \log \boldsymbol{\sigma}_i \\
&= \sum_i (\log \mathcal{N}(0,1)(\boldsymbol{\epsilon}_i) - \log \boldsymbol{\sigma}_i)
\end{aligned} \tag{2.29}$$

As for the decoder, we can just insert the respective densities to compute $\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$. For a normal distribution like above and N the dimension of our input datapoints, we have:

$$\begin{aligned}
\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) &= \log \left((2\pi)^{-\frac{N}{2}} det(\boldsymbol{\sigma}^2 E)^{-\frac{1}{2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T(\boldsymbol{\sigma}^2 E)^{-1}(\mathbf{x}-\boldsymbol{\mu})} \right) \\
&= -\frac{1}{2} \left(N \log 2\pi + \sum_{i=1}^{N} \log \boldsymbol{\sigma}_i^2 + \sum_{j=1}^{N} \frac{(\mathbf{x}_j - \boldsymbol{\mu}_j)^2}{\boldsymbol{\sigma}_j^2} \right) \\
&= -\frac{1}{2} \sum_{i=1}^{N} \left(\log(2\pi \boldsymbol{\sigma}_i^2) + \frac{(\mathbf{x}_i - \boldsymbol{\mu}_i)^2}{\boldsymbol{\sigma}_i^2} \right)
\end{aligned} \tag{2.30}$$

And for a Bernoulli distribution we can derive in the same way:

$$\begin{aligned}
\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) &= \log \left(\prod_{i=1}^{N} \mathbf{p}_i^{\mathbf{x}_i} (1 - \mathbf{p}_i)^{1-\mathbf{x}_i} \right) \\
&= \sum_{i=1}^{N} \mathbf{x}_i \log(\mathbf{p}_i) + (1 - \mathbf{x}_i) \log(1 - \mathbf{p}_i)
\end{aligned} \tag{2.31}$$

And with $p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}|\mathbf{z})p(\mathbf{z})$, we have now all the tools to compute the estimator of the ELBO $\tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}, \boldsymbol{\epsilon})$ and its gradients and thus all the results to make AEVB work. With this setup of a Gaussian prior and variational encoder, as well as a Gaussian/Bernoulli generative model parametereized by ANNs, we call this instance of AEVB algorithm the *Variational Autencoder* (VAE). Before we dive into the implementation and results of this thesis, we will discuss some further theoretical results that are more specific to the application of the VAE on the problem dataset.

2.8 Convolutional Neural Networks (CNN)

While standard, Fully Connected Neuronal networks offer a great way to realize the encoder and decoder of our VAE, there are approaches that work better on certain kinds of data. *Convolutional Neural Networks* (CNN) are an ANN structure, that offers good results when applied on high-dimensional image data. The main idea with CNNs is, that images consist of recurring combinations of shapes that can be learned by lower dimensional *filters*, which are basically smaller neural nets applied to multiple positions in the image.

2.9 Dual VAE setup

Let $\tilde{\mathbf{z}}_1 = i_1(\mathbf{z}_1, \mathbf{z}_2)$ and $\tilde{\mathbf{z}}_2 = i_2(\mathbf{z}_1, \mathbf{z}_2)$ be the interactions of $\mathbf{z}_1$ and $\mathbf{z}_2$ and g be a scaling function, that maps high-dimensional ....

$$\begin{aligned}
&\log p_\theta(\mathbf{x}) + \log r_\iota(g(\mathbf{x})) \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x})\right] + \mathbb{E}_{s_\chi(\mathbf{z}|\mathbf{x})}\left[\log r_\iota(g(\mathbf{x}))\right] \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x}, \tilde{\mathbf{z}}_1)}{p_\theta(\tilde{\mathbf{z}}_1|\mathbf{x})}\right] + \mathbb{E}_{s_\chi(\mathbf{z}|\mathbf{x})}\left[\log \frac{r_\iota(g(\mathbf{x}), \tilde{\mathbf{z}}_2)}{r_\iota(\tilde{\mathbf{z}}_2|g(\mathbf{x}))}\right] \\
&= \ldots \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log\left(\frac{p_\theta(\mathbf{x}, \tilde{\mathbf{z}}_1)}{q_\phi(\mathbf{z}_1|\mathbf{x})}\right)\right] + KL(q_\phi(\mathbf{z}_1|\mathbf{x})||p_\theta(\tilde{\mathbf{z}}_2|\mathbf{x})) \\
&\quad + \mathbb{E}_{s_\chi(\mathbf{z}|\mathbf{x})}\left[\log\left(\frac{r_\iota(g(\mathbf{x}), \tilde{\mathbf{z}}_2)}{s_\chi(\mathbf{z}_2|\mathbf{x})}\right)\right] + KL(s_\chi(\mathbf{z}_2|\mathbf{x})||r_\iota(\tilde{\mathbf{z}}_2|g(\mathbf{x})))
\end{aligned} \tag{2.32}$$

Chapter 3

Application

For this thesis, a dataset was provided to evaluate the performance of different variations of VAEs for the problem of separating foreground and background signal in images of neuronal activity inside of a mouse brain. The images were obtained using a technique called *2-photon BLBKL*, where a special camera used to film multiple layers of brain cells. Each image has a very tight spatial focus on one layer of cells, making it vulnerable to background signal emmited by cells in layers beneath. Their activity usually pans out across large sectors of the image making it harder to gain information about foreground signal by the cells located in the focussed layer. The real world data in this scenario suffers from many other problems that have to be addressed as well by preprocessing steps and more complex models. Also dimensionality is usually very high, requiring expensive training. Therefore, for the experiments of this thesis, a dummy dataset has been provided to simulate the basic problem setting and allow for more efficient evaluation of the different scenarios to be discussed in the following. The dataset consists of $N_{\mathbf{X}} = 2000$ images with 60x60 pixels. Within each image, there are different 6x6 sectors, that independently are either on or off

In the following, details of the implementation of a VAE and results