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 differantiable 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 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. Subsequently, we can use q instead of p 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), a probabilistic model designed for learning latent representations with the help of Deep Neuronal Networks (DNNs).

2.3 Kullback-Leibler divergence

Continuing 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

$$KL(q(\mathbf{x})||p(\mathbf{x})) = \int_{\mathbf{x}} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{p(\mathbf{x})} d\mathbf{x} \tag{2.2}$$

In the discrete case, it is analogously

$$KL(q(\mathbf{x})||p(\mathbf{x})) = \sum_{\mathbf{x}} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{p(\mathbf{x})} \tag{2.3}$$

Here, log is an abbreviation for the logarithm to base 2. 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 '=' *iff* $\frac{p(\mathbf{x})}{q(\mathbf{x})} = 1 \Leftrightarrow p(\mathbf{x}) = q(\mathbf{x})$.

Before we dive deeper into how to utilize the KL divergence for our problem, let us gain a better understanding of why it is a sound choice for measuring 'similarity'.

2.4 Variational Lower Bound (ELBO)

As we discussed previously, $p(\mathbf{x})$ as well as$p(\mathbf{z}|\mathbf{x})$ are supposed to be intractable. 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* ϕ 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 the 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:

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

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)

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

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 $\mathbf{x} \in \mathbf{X}$:

$$\mathcal{L}_{\theta,\phi}(\mathbf{X}) = \sum_{\mathbf{x}\in\mathbf{X}} \frac{\mathcal{L}_{\theta,\phi}(\mathbf{x})}{N_\mathbf{X}} \tag{2.9}$$

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 $KL(p_\theta(\mathbf{x})||q_\phi(\mathbf{x}))$ 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 and then 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 each 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, how η is a hyperparameter, that determines the amount of influence the gradients have on the parameters when updating. It is therefore also called the *learning rate*. It is an important to choose η properly as it highly affects the convergence of the algorithm: If we choose η too small, we will only move slowly in the direction of local maxima. However, if we choose it to big, our steps get too big and the algorithm can not converge.

2.5.2 Estimation of the gradients

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 $\mathbf{x} \in \mathbf{X}$ because since (XX) we have

$$\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{X}) = \nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} \sum_{\mathbf{x} \in \mathbf{X}} \frac{\mathcal{L}_{\theta,\phi}(\mathbf{x})}{N_{\mathbf{X}}} = \frac{1}{N_{\mathbf{X}}} \sum_{\mathbf{x} \in \mathbf{X}} \nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} \mathcal{L}_{\theta,\phi}(\mathbf{x}) \tag{2.10}$$

For gradients of the ELBO with respect to generative model parameters $\boldsymbol{\theta}$, it is simple to obtain an unbiased Monte Carlo estimator:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) &= \nabla_{\boldsymbol{\theta}} \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\log p_\theta(\mathbf{x}, \mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x}) \right] \\ &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\nabla_{\boldsymbol{\theta}} (\log p_\theta(\mathbf{x}, \mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})) \right] \\ &\simeq \nabla_{\boldsymbol{\theta}} (\log p_\theta(\mathbf{x}, \mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})) \\ &= \nabla_{\boldsymbol{\theta}} \log p_\theta(\mathbf{x}, \mathbf{z}) \end{aligned} \tag{2.11}$$

With the $\mathbf{z}$ in the last two lines being 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:

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

For $\mathbf{z}$ continuous and $q_\phi(\mathbf{z}|\mathbf{x})$ and $p_\theta(\mathbf{x}|\mathbf{z})$ differentiable (the case we're focussing on), we can reparameterize $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$ to circumvent this problem.

We can choose f as some differentiable transformation and introduce another random variable $\boldsymbol{\epsilon}$ independent of $\boldsymbol{\phi}$, $\boldsymbol{\theta}$ and $\mathbf{x}$ such that

$$\mathbf{z} = f(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \mathbf{x}) \tag{2.13}$$

This is also referred to as *reparameterization trick*. With (2.12) we can rewrite our gradient as follows:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) &= \nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\log p_\theta(\mathbf{x}, \mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right] \\
&= \nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} \mathbb{E}_{p(\boldsymbol{\epsilon})} \left[\log p_\theta(\mathbf{x}, f(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \mathbf{x})) - \log q_\phi(f(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \mathbf{x})|\mathbf{x})\right] \\
&= \mathbb{E}_{p(\boldsymbol{\epsilon})} \left[\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} (\log p_\theta(\mathbf{x}, f(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \mathbf{x})) - \log q_\phi(f(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \mathbf{x})|\mathbf{x}))\right] \\
&\simeq \nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} (\log p_\theta(\mathbf{x}, f(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \mathbf{x})) - \log q_\phi(f(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \mathbf{x})|\mathbf{x})) \\
&=: \nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} \tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}, \boldsymbol{\epsilon})
\end{aligned} \tag{2.14}$$

And with $\tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}, \boldsymbol{\epsilon}) := \log p_\theta(\mathbf{x}, f(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \mathbf{x})) - \log q_\phi(f(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \mathbf{x})|\mathbf{x})$ the last row is a Monte Carlo estimator for a single sample $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$. It is unbiased because:

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

Also, $\tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}, \boldsymbol{\epsilon})$ is an unbiased estimator for $\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x})$ with an analog argument.

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:

$$\nabla_{\boldsymbol{\alpha}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{X}) = \frac{1}{N_\mathbf{X}} \sum_{\mathbf{x} \in \mathbf{X}} \nabla_{\boldsymbol{\alpha}} \mathcal{L}_{\theta,\phi}(\mathbf{x}) \simeq \frac{1}{N_\mathbf{M}} \sum_{\mathbf{x} \in \mathbf{M}} \nabla_{\boldsymbol{\alpha}} \mathcal{L}_{\theta,\phi}(\mathbf{x}) = \nabla_{\boldsymbol{\alpha}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{M}) \tag{2.16}$$

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):

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{M}) \simeq \frac{1}{N_{\mathbf{M}}} \sum_{(\mathbf{x}^{(i)}, \boldsymbol{\epsilon}^{(i)}) \in \mathbf{M}'} \nabla_{\boldsymbol{\theta}} \log p_\theta(\mathbf{x}, f(\boldsymbol{\epsilon}^{(i)}, \boldsymbol{\phi}, \mathbf{x})) \tag{2.17}$$

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

$$\nabla_{\boldsymbol{\phi}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{M}) \simeq \frac{1}{N_{\mathbf{M}}} \sum_{(\mathbf{x}^{(i)}, \boldsymbol{\epsilon}^{(i)}) \in \mathbf{M}'} \nabla_{\boldsymbol{\phi}} (\log p_\theta(\mathbf{x}^{(i)}, f(\boldsymbol{\epsilon}^{(i)}, \boldsymbol{\phi}, \mathbf{x})) - \log q_\phi(f(\boldsymbol{\epsilon}^{(i)}, \boldsymbol{\phi}, \mathbf{x}^{(i)}) | \mathbf{x}^{(i)})) \tag{2.18}$$

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: Batch Gradient Descent

Initialize parameters $\boldsymbol{\theta}$, $\boldsymbol{\phi}$ randomly
while *not converged* **do**
  Randomly draw a minibatch $\mathbf{M} \subset \mathbf{X}$
  Sample $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$
  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, 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 The Variational Auto-encoder (VAE)

2.6.1 Choice of posteriors

The definition of the AEVB algorithm left some choices regarding the exact nature of the final model. Namely:

- the parameterization of decoder model $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$
- the parameterization for the variational encoder model $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$
- the distribution of $\mathbf{z}$ and it's reparameterization f

The definition of the AEVB algorithm left some more choices regarding the exact nature of the final model. We also need to specify the nature of $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$, which has to be complex enough to model the data sufficiently well.

2.6.2 Neuronal Networks

An Artificial Neural Network (ANN) is a composition of *layers* $\alpha^{(i)}$ with L the number of layers.

$$NeuralNet = \alpha^{(L)} \circ \alpha^{(L-1)} \circ ... \circ \alpha^{(1)} \tag{2.19}$$

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* $M^{(i)}$ as follows:

$$\begin{aligned} \alpha^{(i)} : \mathbb{R}^{N^{(i)}} &\to \mathbb{R}^{M^{(i)}} \\ x &\mapsto g^{(i)}(W^{(i)}x + b^{(i)}) \end{aligned} \tag{2.20}$$

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. To match neighbouring input and output dimensions of the layers, we further require $M^{(i-1)} = N^{(i)}$ for $i \in \{2, ..., L\}$ with $M^{(1)}$ the dimension of the input of the network. 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 function as well. In order for an ANN to 'learn' we interpret $\boldsymbol{\phi} = \{W^{(2)}, b^{(2)}, W^{(3)}, b^{(3)}, ..., W^{(L)}, b^{(L)}\}$ as parameters, that we can update with respect to their gradients while optimizing. Therefore, $g^{(i)}$ has to be differentiable. Common activation functions include many others:

- sigmoid function $Sig(x) = \frac{1}{1-\exp(-kx)}$ with $k \in \mathbb{R}$
- rectifier function $ReLu(x) = \max(0, x)$
- 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 Sig sometimes offer better interpretability of the results as they have values in $(0, 1)$.
They 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 to compute the gradients. Another approach that is currently evolving is *differentiable programming*, where the key idea is to compute derivatives for defined functions automatically. 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.

2.6.3 Neural Networks for parameterizing encoder

We can use ANNs to parameterize the encoder as a simple factorized Gaussian as follows:

$$\begin{aligned} q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) &= \mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma})^2)(\mathbf{z}) \\ (\boldsymbol{\mu}, \log \boldsymbol{\sigma}) &= NeuralNet_{\boldsymbol{\phi}}(\mathbf{x}) \\ q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) &= \prod_i q_{\boldsymbol{\phi}}(\mathbf{z}_i|\mathbf{x}) = \prod_i \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i^2)(\mathbf{z}_i) \end{aligned} \tag{2.21}$$

Together with the reparameterization trick (XX)

2.7 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 a NN structure, that offers good results when applied on high-dimensional image data. The main idea with CNNs is, that images consists of recurring combinations of shapes that can be learned by lower dimensional filters.

Appendix A

Appendix Title