ERASMUS UNIVERSITY OF ROTTERDAM

MASTER THESIS

# Improving Data Compression Based On Deep Learning

*Author:*
Friso H. KINGMA

*Supervisor:*
Associate Professor Michel VAN DE VELDEN

*A thesis submitted in fulfillment of the requirements for the degree of Master of Science*

*in the*

Operations Research and Quantitative Logistics
Econometrics and Management Science

November 6, 2019

ERASMUS UNIVERSITY OF ROTTERDAM

# *Abstract*

Erasmus School of Economics

Econometrics and Management Science

Master of Science

**Improving Data Compression**
**Based On**
**Deep Learning**

by Friso H. KINGMA

We generate and use an ever-increasing amount of data in digital form. Technological advancements in digital communication have allowed us to transmit data to almost anyone on the globe. However, storage and transmission capacities do not seem to keep up with the explosive growth of data. Due to data compression, we can represent data in compact form and therefore store and transmit more data with the same cost.

In this thesis, we focus on compression without loss of information, known as lossless compression, of high-dimensional data. Lossless compression can be achieved by finding structure that exists in the data through probabilistic modelling and exploiting that structure with compression algorithms. Probabilistic models based on deep learning have experienced a lot of progress in recent years. Efficiently using these models in compression algorithms, however, is still an open problem.

Earlier work has focused on designing a compression algorithm that uses a latent variable model, called the bits-back scheme. Latent variable models based on neural networks can be efficiently optimized for high-dimensional data. We extended both the latent variable model and the bits-back scheme, such that we achieve more effective lossless compression. We call the extensions *nested latent variable models* and the *recursive bits-back scheme* respectively.

Through experiments we verify that the recursive bits-back scheme using nested latent variable models results in lossless compression that is empirically superior to existing techniques. We also conduct an extensive analysis of how different versions of the method compare to earlier work on lossless compression using latent variable models.

# *Acknowledgements*

I thank my supervisor Professor Michel van de Velden at the Erasmus University of Rotterdam for supporting my choice to seek a research project abroad and having trust in me to develop my knowledge independently. Furthermore, the contribution could not have been accomplished without close supervision of Jonathan Ho at University of California, Berkeley. Jonathan has introduced me to most of the deep learning concepts and helped me write my first research paper. He kept challenging me and always pushed me to get out the best of myself. I also cannot express enough thanks to my advisor Professor Pieter Abbeel at University of California, Berkeley for accepting me to the Berkeley Artificial Intelligence Research lab, sponsoring me throughout my stay, introducing me to Jonathan and the many fruitful discussions. Lastly, I thank the board of the International Conference on Machine Learning for inviting me to give a plenary presentation on our contribution in Long Beach, California.

# Contents

# 1 Introduction

## 1.1 Motivation

In the last decade we have witnessed a transformation in the way we communicate. One of the largest factors that caused this transformation is the Internet. We are able to rapidly exchange information with other people around the globe without much effort. Not too long ago, people had to move heaven and earth to simply get a message to someone across the oceans, taking months to arrive. Right now everyone is part of "the grid" and communication is almost instant. Data compression is one of the technologies that enables fast and efficient transmission of information in the age of digital communication. It would not be practical to put images, let alone video, on websites if it were not for data compression algorithms. Mobile phones would not be able to provide increasingly clear communication were it not for compression. Where data compression was once the domain of a specialized group of engineers and scientists, it is now overwhelmingly ubiquitous.

What makes data compression so critical for communication of data? Data compression algorithms are used to reduce the number of bits required to represent data. Popular forms of compression are JPEG, PNG and MPEG. In brief, data compression is the science of representing data in compact form. The reason we need data compression is because we have an increasing amount of information in digital form. All kinds of sensors pick up progressively detailed information of our environment. Examples of such sensors are camera's and microphones, that have experienced great leaps in their ability to capture images, videos and audio of high quality. Furthermore, we humans are also producers of vast amounts of digital information. For example, through our ability to create large amounts of text, like this thesis. Without compression algorithms the resulting images, video, audio and text can take up large amounts of storage, not to mention being extremely costly to transmit.

One option is to focus on developing better transmission and storage technologies. However, development of these techniques is unable to keep up with the explosive growth of data. The need for mass storage and transmission seems to be increasing twice as fast as storage and transmission capacities improve (Sayood, 2017). There are even cases in which the transmission capacity is physically bounded. For example, the amount of information we can transmit over the airwaves is always limited by the properties of the atmosphere.

Therefore, it is important to keep improving compression algorithms such that we can store and transmit increasing amounts of data with the same cost. We focus on *lossless* compression of data. As the name implies, lossless compression involves no loss of information. If data has been losslessly compressed, the original data can be recovered exactly from the compressed data. Lossless compression techniques are important for cases in which any difference between the original and reconstructed data is not tolerated. For example, compressing text should ideally be without loss,

since small differences might lead to very different meanings. For example, "Do *not* start a war" compared to "Do *now* start a war". Furthermore, photographers might want to store their photographs without any quality loss, in that case lossless image compression should be used.

Lossless compression of high-dimensional data is done by finding structure in the data and exploiting that structure. Lossless compression algorithms generally make use of two main components: a probabilistic model that formalizes structure in the data and a coder that is able to assign short representations to data based on that structure. Deep learning, or machine learning with the use of large artificial neural networks, has experienced a lot of attention in recent years. Not without reason, because deep learning has among others accelerated research in probabilistic modelling of complex high-dimensional data, like images, video, audio and text. Probabilistic models whose design incorporates neural networks can be efficiently optimized using large datasets to accurately model complex relations in high-dimensional data.

The rapid developments in deep learning have resulted in a gap between the modelling capacity of probabilistic models and the exploitation of that capacity in terms of compression. It is not always immediately clear which probabilistic models can be efficiently used in fast compression algorithms. Therefore, both the probabilistic model and compression scheme have to be designed carefully.

## 1.2 Research Questions and Overview

In Section 2 we introduce the concept of lossless compression and probabilistic modelling. In Section 3 we state what is needed to compress complex high-dimensional data and what kind of resources we assume access to.

Afterwards, we dive deep into the design of both the model and the compression scheme. We do this on the basis of 3 sets of research questions. We formulate the research questions in this work as follows:

**Research Question Set 1**

What probabilistic model is capable of modelling complex high-dimensional data? What is a *scalable* and *effective* lossless compression algorithm for high-dimensional input data that makes efficient use of this probabilistic model?

In Section 4 we discuss a probabilistic latent variable model. Latent variable models parameterized with neural networks allow us to model complex relations in high-dimensional data. We also discuss how to optimize latent variable models using large datasets. Afterwards, we discuss a compression algorithm capable of using latent variable models for lossless compression, called the bits-back scheme.

**Research Question Set 2**

How do we improve the latent variable model? How do we extend the bits-back scheme such that it makes efficient use of the improved latent variable model?

In Section 5 we discuss nested latent variable models, an extension of latent variable models. Nested latent variable models parameterized with neural networks tend to be better in modelling complex high-dimensional data. We also discuss how to optimize nested latent variable models using large datasets. Afterwards, we introduce an extension of the bits-back scheme that is able to use nested latent variable models for lossless compression, called the extended bits-back scheme. Finally, we discuss an improved version of the bits-back scheme that exploits the unique structure of nested latent variable models, called the recursive bits-back scheme.

**Research Question Set 3**

How well are nested latent variable models able to model image data? How well does the recursive bits-back scheme perform on image data? How does it perform compared to other methods?

In Section 6 we discuss our experiments and results. First, we show that nested latent variable models are indeed better in modelling a broad variety of images than conventional latent variable models. After that, we perform lossless compression with the recursive bits-back scheme versus the extended bits-back scheme, and conduct an extensive analysis of the results. Finally, we compare the recursive bits-back scheme to established lossless compression techniques.

# 2 Background

We first give an introduction into lossless compression and probabilistic modelling. Section 2.1 introduces basic concepts and terminology. Section 2.2 lays out a simple baseline coding scheme. Section 2.3 explains the idea behind lossless compression. Section 2.4 limits of lossless compression.

## 2.1 Basic Concepts

Our goal is to store data such that the filesize is as small as possible, at high speed while retaining the ability to perfectly reconstruct the original data. The size of a file is determined by how the data it describes is represented on the system and how much information content that representation contains. Usually, but not always, data is represented by sequences of bits. One bit, portmanteau for binary digit, is a digit in the base-2 numeral system. For example, data represented by the binary number

$$\underbrace{1000111}_{7 \text{ bits}} \tag{2.1}$$

contains 7 bits of information. Therefore, this binary number takes 7 bits to store on a system. Attaining small filesizes without destroying content is done through lossless compression of the data contained in the file. **Lossless compression is converting the data into a representation that uses the least amount of information, while being able convert the representation back into the exact original data**. For simplicity, we always consider binary representations and measure information content in bits.

We define lossless compression on finite discrete data. We say that one-dimensional data $x$ comes from a pre-defined finite alphabet $\mathcal{X}$ consisting of symbols $s_i$.

$$\mathcal{X} = \{s_1, s_2, \ldots, s_{|\mathcal{X}|}\} \tag{2.2}$$

For example, images are often expressed through the RGB color model. Every pixel in the image is built up from 3 color channels: green, blue and red. Each color channel has an intensity value represented by an integer between 0 and 255. This boils down to a 256 symbol alphabet $\mathcal{X} = \{s_1, \ldots, s_{256}\} = \{0, \ldots, 255\}$ for every color channel. We interpret a data vector $\mathbf{x}$ as a finite sequence of symbols $(x_1, \ldots, x_P)$, where $x_i \in \mathcal{X}$. Therefore, RGB images are interpreted as a sequence of color channel values.

Before we can discuss compression of data, we first have to establish how to assign a representation to data. We define **coding** merely as linking symbols to a **unique representation**, without necessarily accounting for its length. Generally speaking, an encoding scheme takes in a sequence of symbols and **encodes** them into a representation. The resulting representations is a stream of bits, sometimes referred to as a **bitstream**. A decoding scheme takes in a bitstream and **decodes** (part of) it into

a sequence of symbols. We usually refer to a **coding scheme**, which includes the instructions for both encoding and decoding. A **compression scheme** has the additional goal of converting symbols into representations that use the least amount of information.

We present coding (or compression) schemes through the sender-receiver framework. A sender must encode the symbols into representations, the resulting bitstream is sent over a medium and a receiver must decode the bitstream to exactly retrieve the original symbol sequence. An example of where the sender-receiver framework would apply is a situation where person 1 (sender) wishes to send over an image over an e-mail network to person 2 (receiver), but the receiver requires a perfect copy of the image. Or a situation where you wish to store an image on a hard-drive, such that you can open the exact same content later. In the latter case "you" are both the sender and receiver, and the hard-drive is the medium.

## 2.2 Baseline Scheme

The most simple way to design a coding scheme for the 256 symbol alphabet of a color channel is by assigning the following binary number to every symbol,

$$\begin{aligned} C[0] &= 00000000 \\ C[1] &= 00000001 \\ C[2] &= 00000010 \\ &\vdots \\ C[253] &= 11111101 \\ C[254] &= 11111110 \\ C[255] &= 11111111 \end{aligned} \tag{2.3}$$

resulting in a *codelength* of $l(x) = 8$ bits for every $x \in \mathcal{X}$. In this case, a valid encoding scheme for data is merely a concatenation of the binary numbers corresponding to the symbol sequence:

$$C\left[(x_1, \ldots, x_P)\right] = C[x_1]C[x_2] \cdots C[x_P] \tag{2.4}$$

With this coding scheme we can construct a bitstream communicating an RGB image while still being able to distinguish between the different symbols. Suppose we wish to encode an RGB image

$$\mathbf{x}^\circ = (\underbrace{34, 55, 30}_{\text{pixel 1}}, \underbrace{2, 5, 200}_{\text{pixel 2}}) \tag{2.5}$$

The resulting bitstream, with codelength $l(\mathbf{x}^\circ) = 48$ bits, is then:

$$C\left[(134, 55, 30, 2, 5, 200)\right] = 100001100011001000011110000000100000010111001000 \tag{2.6}$$

The decoding scheme is simply defined as "read of 8 bits at the time and determine the corresponding symbols":

$$\underbrace{10000110}_{C[134]}\underbrace{00110010}_{C[55]}\underbrace{00011110}_{C[30]}\underbrace{00000010}_{C[2]}\underbrace{00000101}_{C[5]}\underbrace{11001000}_{C[200]} \tag{2.7}$$

Evidently, this coding scheme has an *expected* codelength $L$ of 8 bits per symbol and this strategy can be applied to any alphabet $\mathcal{X}$. Nonetheless, this approach is fairly trivial and assumes no prior knowledge over the data. In fact, the expected codelength equals the (rounded up) maximum rate of information that can be transmitted for an alphabet with cardinality $|\mathcal{X}|$, called the absolute rate:

$$L = \lceil \text{absolute rate} \rceil = \lceil \log_2(|\mathcal{X}|) \rceil \tag{2.8}$$

The number is rounded up because the absolute rate can be non-integer, whereas the amount of bits used must be integer. Whenever we design a compression scheme, we always refer to this coding scheme, called the *baseline scheme*, as the scheme we want to improve.

## 2.3 Lossless Compression

The idea behind lossless compression is to assign a short representation to symbols that are expected to be frequent and longer representations to symbols that are expected to be not-so-frequent. This is only possible if the compression scheme has some prior knowledge about the kind of data it is going to encounter. For example, suppose we know that we will only encounter images of nightlife. Then we can design a compression scheme that assigns short representations to dark color values ($x \in [0, 124]$) and longer representations to lighter color values ($x \in [125, 255]$). If we take images of nightlife as input data, this strategy will lead to short representations.

However, as we later see, whenever we choose short representations for some symbols, we essentially "sacrifice" a whole group of similar representations that can't be used by the other symbols. This is because assigning variable codelengths to symbols makes decoding more complicated than "read off $N$ bits for every symbol". If we do not pay close attention, ambiguities in the bitstream might occur that lead to more than one possible source sequence $\mathbf{x}$. This is against the requirement of having a lossless compression scheme, since we would like to retrieve the *exact* original data. This requirement is formalized by the *unique decodability* property (Cover and Thomas, 2012). A compression scheme satisfies unique decodability if we are able retrieve the exact original data sequence by reading of the *entire* bitstream. In case of RGB images, if we choose representations shorter than 8 bits for the dark color values ($x \in [0, 124]$), unique decodability *could* ensure that we assign representations longer than 8 bits to the lighter color values ($x \in [125, 255]$). This means that a compression scheme designed for nightlife images could perform *worse than the baseline scheme* on images of daylife, which contains generally lighter color values.

Consequently, we always consider building a compression scheme that improves upon the baseline scheme only for *one chosen population*. Examples of populations within the space of RGB images are nightlife images, daylife images, photos of grass, photos of ocean life, paintings or cartoons. Naturally, many populations can be part of one larger population. For example, photos of grass and ocean life can be part of the population of "natural images". Analogously, paintings and cartoons can be

part of the population of "unnatural images". A population of one-dimensional datapoints $x$ is assumed to have a univariate underlying probability distribution $x \sim p^*$. Likewise, a population of $P$-dimensional datapoints $\mathbf{x}$, like images, is assumed to have a multivariate (joint) underlying probability distribution $\mathbf{x} \sim p^*$. The underlying distribution of a population is considered to be highly complex and generally unknown.

In order to achieve lossless compression on a chosen population, a lossless **compression scheme** always makes use of two main components:

1. **Probabilistic model**: the probabilistic model $p_{\boldsymbol{\theta}}$ acts as an estimate to the underlying probability distribution $p^*$ of the chosen population.
2. **Coder**: the coder is able to convert symbols with a high probability into a short representation and symbols with low probability into longer representations without violating unique decodability.

If the two components are well-designed and our goal is to compress data from within the chosen population, the compression scheme then produces a shorter codelength on average compared to the baseline scheme. For one-dimensional data $x^\circ$, we write $l(x^\circ)$ for the codelength of representation $C[x^\circ]$. Similarly we write $l(\mathbf{x}^\circ)$ for the codelength of representation $C[\mathbf{x}^\circ]$ of high-dimensional data $\mathbf{x}^\circ$. The expected codelength, with respect to the underlying probability distribution is then:

$$\mathbb{E}_{p^*}\left[l(x)\right] = \sum_{x \in \mathcal{X}} p^*(x) l(x) \qquad \text{or} \qquad \mathbb{E}_{p^*}\left[l(\mathbf{x})\right] = \sum_{\mathbf{x} \in \mathcal{X}^P} p^*(\mathbf{x}) l(\mathbf{x}) \tag{2.9}$$

**Simple example of a compression scheme for one-dimensional data**

We present a simple and intuitive example of a compression scheme. Jan (the sender) from the Netherlands would like to digitally communicate the weather in the Netherlands, denoted by variable $x$, to Alejandro (the receiver) in Spain. The transmission medium only accepts binary representations. Alejandro only understands a limited number of English words that describe the weather. To avoid misunderstanding, Jan and Alejandro decide together to use only six words, as represented by the symbol alphabet in column 1 in Table 2.1. Assume that the underlying probability distribution of the weather on any given day during the year is given by the probabilities in column 2. Jan and Alejandro do not have knowledge of the underlying distribution, so they decide to collect historical data about the weather and group them into the six condition categories per day. The resulting dataset acts as a representative sample of the population "weather in the Netherlands". They decide to build a simple **probabilistic model** by counting the frequency of every condition, shown in column 3, which acts as estimate of the underlying distribution.

Jan always communicates the weather of the entire week **x**. They could use the baseline scheme, which assigns a representation of length $\lceil \log_2 (|\mathcal{X}|) \rceil = \lceil \log_2 (6) \rceil = 3$ bits to every weather condition, as shown in column 4. This results in an expected codelength of $L = 3$ bits/symbol. But instead, they decide to use a **Huffman coder** (Huffman, 1952) to assign a representation to every weather condition based on their probability. Their **compression scheme** is simply concatenating the representations of the entire week **x**. The representations are shown in column 5. The Huffman coder ensures that none of the representations is a prefix of one of the others. Therefore, if we see a particular representation, there is no other representation that is a longer version of that. This avoids ambiguity in the bitstream, that consists of concatenated representations. This makes the scheme uniquely decodable. These representations attain an expected codelength of $L = 2.61$ bits/symbol. So this saves Jan and Alejandro $3 - 2.61 = 0.39$ bits/symbol on average.

TABLE 2.1: Weather conditions and corresponding representations.

| $\mathcal{X}$ | **Underlying** $p^*$ | **Model** $p_{\boldsymbol{\theta}}$ | **Baseline** | **Huffman** |
|---|---|---|---|---|
| "rainy" | 1/3 | 35% | 000 | 11 |
| "cloudy" | 1/6 | 17% | 001 | 01 |
| "stormy" | 1/6 | 16% | 010 | 101 |
| "snowy" | 1/9 | 22% | 011 | 00 |
| "sunny" | 1/9 | 5% | 101 | 1001 |
| "windy" | 1/9 | 5% | 111 | 1000 |
| | | $\sum_{x \in \mathcal{X}} p^*(x) l(x)$ : | 3.00 | 2.61 |

However, because of the unique decodability property there are only a limited number of short representations. Consequently, some representations must be *longer than its baseline counterpart*. This is not a problem, since the probability of using such representations is fairly low. However, this compression scheme will likely be suboptimal for communicating the weather of other regions. For example, Alejandro (now the sender) decides to communicate an entire week of weather conditions of Spain with Jan (now the receiver). Since the climate in Spain is Mediterranean the weather conditions given in column 1 have underlying probabilities of $\{\frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{6}, \frac{1}{3}, \frac{1}{6}\}$ respectively. By using the same Huffman representations for a completely different weather system they attain an expected codelength of $\sum_{x \in \mathcal{X}} p^*(x) l(x) = 3.11$ bits/symbol, which is 0.11 bits worse than the baseline scheme on average.

## 2.4 Limits of Lossless Compression

The question remains whether a chosen representation $C[x^\circ]$ or $C[\mathbf{x}^\circ]$ has optimal length $l(x^\circ)$ or $l(\mathbf{x}^\circ)$ respectively. It turns out that every (sequence of) symbol(s) inherently contains a certain degree of information, depending on the underlying probability distribution $p^*$. By constructing codelengths close to this "inherent" information for every symbol, we can accomplish optimal compression.

To understand that, we have to establish a link between probability and information. Shannon (1948) managed to derive information content through an explicit function of probability $p$, called *self-information*:

$$\mathcal{I} = -\log_2 p \tag{2.10}$$

The degree of inherent information of symbols is then given by self-information given by the underlying distribution

$$\mathcal{I}(x) = -\log_2 p^*(x) \qquad \text{or} \qquad \mathcal{I}(\mathbf{x}) = -\log_2 p^*(\mathbf{x}) \tag{2.11}$$

The expected value of self-information with respect to the underlying distribution is called the *entropy* of that distribution:

$$H(p^*) = -\sum_{x\in\mathcal{X}} p^*(x)\log_2 p^*(x) \qquad \text{or} \qquad H(p^*) = -\sum_{x\in\mathcal{X}^p} p^*(\mathbf{x})\log_2 p^*(\mathbf{x}) \tag{2.12}$$

The entropy echoes the average amount of information contained in a distribution. It simply reflects a probability weighted average of the self-information of each symbol. Larger entropies represent larger average information, and perhaps counter-intuitively, the more random a sequence of symbols (the more even the probabilities) the more information they contain. Shannon (1948) shows that the entropy *lower bounds* the expected codelength of any set of uniquely decodable representations assigned to data:

$$H(p^*) \leq \mathbb{E}_{p^*}\left[l(x)\right] \qquad \text{or} \qquad H(p^*) \leq \mathbb{E}_{p^*}\left[l(\mathbf{x})\right] \tag{2.13}$$

In other words, it is impossible compress data to a smaller codelength than the entropy of the corresponding underlying distribution on average.

We express the difference between the **probabilistic model** and the underlying distribution in terms of the Kullback-Leibler divergence:

$$D_{\mathrm{KL}}\left(p^*(\mathbf{x}) \parallel p_{\boldsymbol{\theta}}(\mathbf{x})\right) = \mathbb{E}_{p^*}\left[\log_2 p^*(\mathbf{x}) - \log_2 p_{\boldsymbol{\theta}}(\mathbf{x})\right] \tag{2.14}$$

If a probabilistic model $p_{\boldsymbol{\theta}}$ closely resembles the underlying distribution $p^*$ of a population, or equivalently, if the Kullback-Leibler divergence approaches 0

$$D_{\mathrm{KL}}\left(p^*(x) \parallel p_{\boldsymbol{\theta}}(x)\right) \to 0 \qquad \text{or} \qquad D_{\mathrm{KL}}\left(p^*(\mathbf{x}) \parallel p_{\boldsymbol{\theta}}(\mathbf{x})\right) \to 0 \tag{2.15}$$

and the compression scheme is able to use this probabilistic model and a **coder** to construct representations $C[x]$ or $C[\mathbf{x}]$ that have corresponding codelengths close to the self-information given by the model

$$l(x) \to -\log_2 p_{\boldsymbol{\theta}}(x) \qquad \text{or} \qquad l(\mathbf{x}) \to -\log_2 p_{\boldsymbol{\theta}}(\mathbf{x}) \tag{2.16}$$

then the expected codelength comes close to the entropy of the underlying distribution:

$$\mathbb{E}_{p^*}\left[l(x)\right] \to H(p^*) \qquad \text{or} \qquad \mathbb{E}_{p^*}\left[l(\mathbf{x})\right] \to H(p^*). \tag{2.17}$$

**Simple example of a compression scheme for one-dimensional data continued**

Going back to Jan and Alejandro, the self-information of the underlying weather conditions gives us a notion of how good their compression scheme is. We calculate the self-information of every weather condition to get an idea of how much information, and therefore how many bits, every weather conditions contains. This can be found in column 3 of Table 2.2.

TABLE 2.2: Weather conditions and corresponding representations.

| $\mathcal{X}$ | **Underlying** $p^*$ | **Self-information** $\mathcal{I}$ | **Model** $p_\theta$ | **Baseline** | **Huffman** |
|---|---|---|---|---|---|
| "rainy" | 1/3 | 1.58 | 35% | 000 | 11 |
| "cloudy" | 1/6 | 2.58 | 17% | 001 | 01 |
| "stormy" | 1/6 | 2.58 | 16% | 010 | 101 |
| "snowy" | 1/9 | 3.17 | 22% | 011 | 00 |
| "sunny" | 1/9 | 3.17 | 5% | 101 | 1001 |
| "windy" | 1/9 | 3.17 | 5% | 111 | 1000 |
| | | | $\sum_{x\in\mathcal{X}} p^*(x)l(x)$ : | 3.00 | 2.61 |

Therefore, the corresponding entropy of the underlying distribution is:

$$H(p^*) \approx \frac{1}{3} \times 1.58 + \frac{2}{6} \times 2.58 + \frac{3}{9} \times 3.17 = 2.45 \tag{2.18}$$

The expected codelength using the Huffman representations in column 6 is 2.61 bits/symbol, which results in a discrepancy of $2.61 - 2.45 = 0.16$ bits/symbol. This means that the codelengths resulting from their compression scheme are still not optimal. This could be the result of the Huffman coder basing the representations on a slightly "wrong" model $p_\theta$. Therefore, assume that the model predicts probabilities that are equal to the probabilities of the underlying distribution. The corresponding Huffman representations and the expected codelength are shown in column 6 of Table 2.3.

TABLE 2.3: Weather conditions and corresponding representations.

| $\mathcal{X}$ | **Underlying** $p^*$ | **Self-information** $\mathcal{I}$ | **Model** $p_\theta$ | **Baseline** | **Huffman** |
|---|---|---|---|---|---|
| "rainy" | 1/3 | 1.58 | 1/3 | 000 | 11 |
| "cloudy" | 1/6 | 2.58 | 1/6 | 001 | 00 |
| "stormy" | 1/6 | 2.58 | 1/6 | 010 | 101 |
| "snowy" | 1/9 | 3.17 | 1/9 | 011 | 100 |
| "sunny" | 1/9 | 3.17 | 1/9 | 101 | 011 |
| "windy" | 1/9 | 3.17 | 1/9 | 111 | 010 |
| | | | $\sum_{x\in\mathcal{X}} p^*(x)l(x)$ : | 3.00 | 2.50 |

The expected codelength is still $2.55 - 2.45 = 0.10$ bits/symbol short of the entropy. This is because we chose a compression scheme that simply concatenates the representation of every condition. This way we restricted ourselves by having to construct a separate representation for every weather condition, which forces an integer number of bits per representation. In such a regime, it is impossible to reach the entropy limit. In order to actually reach the fractional self-information per symbol, modern coders use more complicated techniques, that are beyond the scope of this report.

# 3 Compression of High-Dimensional Data

In this thesis, we focus on lossless compression of $P$-dimensional data $\mathbf{x}$ coming from a population with an unknown and highly complex underlying distribution $p^*$. Building a good compression scheme for this type of data is much more complicated than described in the example in Section 2.3 and 2.4. The goal is to swiftly construct representations $C[\mathbf{x}^\circ]$ that have codelengths close to the entropy of the underlying distribution on average:

$$L = \mathbb{E}_{p^*}\left[l(\mathbf{x})\right] \to H(p^*) \tag{3.1}$$

Achieving that can be done by concurrently solving two problems:

1. Design a **probabilistic model** $p_{\boldsymbol{\theta}}$ that is able to be optimized to approximate the underlying distribution $p^*$ of the chosen population well
2. Design a **compression scheme** that is able to use the **probabilistic model** and a **coder** to construct codelengths $l(\mathbf{x})$ close to $-\log_2 p_{\boldsymbol{\theta}}(\mathbf{x})$ fast.

To solve those two problems, we assume access to the following resources:

1. **GPU-chips** that can perform a high number of operations in parallel. We exploit these GPU-chips by designing a compression scheme that is able to compress every dimension of the $P$-dimensional input data $\mathbf{x}$ in parallel. Since the dimensionality $P$ of images can be quite large, a parallelized compression scheme can be several orders of magnitude faster than a scheme that compresses the $P$ dimensions in sequence. For example, assume that compression of a color channel value takes 10 milliseconds. A relatively small RGB image of 200 by 200 pixels already has a dimensionality of $P = 200 \times 200 \times 3 = 120000$. This means that a parallelized compression scheme takes 10 milliseconds to compress this image, whereas a sequential compression scheme takes $120000 \times 10$ ms $= 2$ minutes for the same image.
2. A **coder** called Asymmetric Numeral Systems, (Duda, 2009), henceforth referred to as **ANS**, that is able to exploit GPU-chips to assign representations to $P$ symbols in parallel. The implementation details of ANS are beyond the scope of this thesis, but it is important to remember the following properties:
   - ANS is configured by a matrix where every entry $p_{ik}$ is the probability of the $k$th symbol (in the alphabet) of the $i$th dimension:

$$\mathbf{P} = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1|\mathcal{X}|} \\ p_{21} & p_{22} & \cdots & p_{2|\mathcal{X}|} \\ \vdots & \vdots & \ddots & \vdots \\ p_{P1} & p_{P2} & \cdots & p_{P|\mathcal{X}|} \end{bmatrix}, \tag{3.2}$$

- Using this configuration, ANS creates a bijective (one-to-one) mapping between symbols and representations, such that a sequence of symbols $\mathbf{x}^\circ$ gets assigned to a unique representation $C[\mathbf{x}^\circ]$, for all possible $\mathbf{x}^\circ \in \mathcal{X}^P$. The codelength of a representation $C[\mathbf{x}^\circ]$ is approximately equal to:

$$l(\mathbf{x}^\circ) \approx -\log_2 \left( \prod_{i=1}^{P} p(x_i^\circ) \right) = -\log_2 p(\mathbf{x}^\circ) \tag{3.3}$$

  where $p(x_i^\circ)$ is probability of symbol $x_i^\circ \in (x_1^\circ, \ldots, x_p^\circ)$ as given by matrix $\mathbf{P}$.

- **Encoding with ANS configured using P** can be interpreted as converting a sequence of symbols $\mathbf{x}^\circ$ into their assigned representation (and possibly appending it to a bitstream). As an illustration, consider an existing bitstream of "0100011111" and the representation corresponding to $\mathbf{x}^\circ$ is $C[\mathbf{x}^\circ] = 1011$, then ANS adds 1011 to the bitstream:

$$0100011111\boxed{1011} \leftarrow \mathbf{x}^\circ \tag{3.4}$$

- **Decoding with ANS configured using P** can be interpreted as searching from the right side of a bitstream to find a representation that uniquely corresponds to a sequence of symbols and removing the corresponding bits. As an illustration, consider an existing bitstream of "01000111111011". Assume that the first combination of bits (from the right side) that corresponds to a sequence of symbols is 1011. Then ANS removes 1011 and converts it into the corresponding $\mathbf{x}^\circ$:

$$0100011111\boxed{1011} \rightarrow \mathbf{x}^\circ \tag{3.5}$$

3. A **dataset** $\mathcal{D} = \{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^{N_\mathcal{D}}\}$ that acts as representative sample of the chosen population. For example, if we decide to compress cartoons, we aim to collect as much cartoon-images as possible. In practice, we aim for a compression scheme that is as general as possible, so we try to collect a large dataset that has a broad variety of images.

4. An optimization method called **stochastic gradient descent** (Robbins and Monro, 1951) that is able to find the probabilistic model parameters $\boldsymbol{\theta}$ that optimize the *maximum log-likelihood* (ML) criterion. Under the ML criterion, we wish to find the parameters $\boldsymbol{\theta}$ that maximize the average of the log-probabilities assigned to the datapoints in $\mathcal{D}$ by the model

$$\underset{\boldsymbol{\theta}}{\operatorname{argmax}} \; \frac{1}{N_\mathcal{D}} \sum_{i=1}^{N_\mathcal{D}} \log_2 p_{\boldsymbol{\theta}}(\mathbf{x}^i) \tag{3.6}$$

   such that under the probabilistic model the dataset $\mathcal{D}$ is most probable. This is otherwise referred to as *maximum likelihood estimation* (MLE), a common approach to estimate the parameters of a probability distribution. Note that the datapoints are representative for the chosen population, and therefore assumed to be independent and identically distributed according to the underlying distribution:

$$\mathbf{x}^i \overset{\text{i.i.d.}}{\sim} p^* \tag{3.7}$$

This means that the ML criterion is an unbiased (Monte Carlo) estimator of *expected* log-likelihood:

$$\frac{1}{N_{\mathcal{D}}} \sum_{i=1}^{N_{\mathcal{D}}} \log_2 p_{\boldsymbol{\theta}}(\mathbf{x}^i) \simeq \mathbb{E}_{p^*} \left[ \log_2 p_{\boldsymbol{\theta}}(\mathbf{x}) \right] \tag{3.8}$$

where we used "$\simeq$" to denote "is an unbiased estimate of". Note that we also use the notation "$\simeq$" if we mean "has the unbiased estimator", but this must be clear from the context. Maximizing the expected log-likihood is the same as minimizing the Kullback-Leibler divergence:

$$\begin{aligned} \operatorname*{argmax}_{\boldsymbol{\theta}} \ \mathbb{E}_{p^*} \left[ \log_2 p_{\boldsymbol{\theta}}(\mathbf{x}) \right] &= \operatorname*{argmin}_{\boldsymbol{\theta}} \ \mathbb{E}_{p^*} \left[ \log_2 p^*(\mathbf{x}) - \log_2 p_{\boldsymbol{\theta}}(\mathbf{x}) \right] \\ &= \operatorname*{argmin}_{\boldsymbol{\theta}} \ D_{\mathrm{KL}} \left( p^*(\mathbf{x}) \parallel p_{\boldsymbol{\theta}}(\mathbf{x}) \right) \end{aligned} \tag{3.9}$$

Therefore, MLE finds the parameters $\boldsymbol{\theta}$ of the probabilistic model $p_{\boldsymbol{\theta}}$ such that the Kullback-Leibler divergence is minimized.

# 4 Latent Variable Models & The Bits-Back Scheme

In this Section, we give an introduction to an existing approach to lossless compression of high-dimensional data. We first we introduce a latent variable model in Section 4.1, a probabilistic model capable of modelling complex high-dimensional data. Afterwards, we give a specification of the model distributions and explain how to optimize the model. Then, in Section 4.2, we explain how to use latent variable models for lossless compression. We lay out a naive compression scheme and the bits-back scheme, of which the latter is arguably better in compression. Finally, we analyze the expected codelength of the bits-back scheme.

## 4.1 Design and Optimization of the Probabilistic Model

### 4.1.1 Latent Variable Models

We are interested in a probabilistic model that is able to model complex patterns in high-dimensional data $\mathbf{x}$. We extend the variables $\mathbf{x}$, by assuming that there exist unobserved or *latent variables* $\mathbf{z} \in \mathbb{R}^M$ that the distribution of the data is conditioned on:

$$p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) \tag{4.1}$$

In contrast to the latent variables $\mathbf{z}$, the data specifying variables $\mathbf{x}$ are henceforth called *observed variables*. We also define a distribution that describes a prior belief about the values of the latent variables:

$$p(\mathbf{z}) \tag{4.2}$$

This density function does not change during optimization, hence the missing subscript. The latent variables can be interpreted as describing unobserved "high-level" features of the data.

The latent variable model is essentially a *Bayesian* model that draws the latent variables from a *prior distribution* $p(\mathbf{z})$ and then relates them to the observations through the *likelihood* distribution $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$.

The important advantage of latent variable models, is that even when the likelihood and prior can be simple distributions, the marginal distribution over the observed variables $\mathbf{x}$

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \int p(\mathbf{z}) p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) \, d\mathbf{z} \tag{4.3}$$

can contain almost arbitrary dependencies between the dimensions. Therefore, such an implicit distribution over the observed variables $\mathbf{x}$ can be very complex. This expressivity makes latent variable models suitable for approximating the complicated underlying distribution $p^*$. The distribution $p_{\boldsymbol{\theta}}(\mathbf{x})$ is called the *model evidence*.

The goal is to find the parameters $\boldsymbol{\theta}$ for distribution $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ that optimize the model evidence $p_{\boldsymbol{\theta}}(\mathbf{x})$. Using MLE, we need to compute the log of the model evidence $\log_2 p_{\boldsymbol{\theta}}(\mathbf{x})$. However, the model evidence $p_{\boldsymbol{\theta}}(\mathbf{x})$ is typically intractable to compute, since we need to evaluate all possible configurations of the latent variables $\mathbf{z}$. This makes it hard to optimize the model evidence explicitly.

Furthermore, we are interested in computing the posterior distribution $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$, which describes a posterior belief about the values of the latent variables *given* we have seen data $\mathbf{x}$. The posterior can be interpreted as describing a distribution over the high-level features given the data $\mathbf{x}$, essentially defining the relationship between the data and its corresponding high-level features. From Bayes' theorem the *posterior* over the latent variables $\mathbf{z}$ is equal to the likelihood, multiplied by our prior belief about the latent variables, scaled by a normalizing constant: the model evidence $p_{\boldsymbol{\theta}}(\mathbf{x})$.

$$p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{z})p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})}{p_{\boldsymbol{\theta}}(\mathbf{x})} \tag{4.4}$$

The intractability of the model evidence $p_{\boldsymbol{\theta}}(\mathbf{x})$ also makes it hard to calculate the posterior distribution $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$.

### 4.1.2 The Inference Model & The Evidence Lower Bound

We introduce a technique, called variational inference (Jordan et al., 1999), that introduces an additional model specifying the distribution $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ with parameters $\boldsymbol{\phi}$, that tries to approximate the "true" model posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$.

$$q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \approx p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) \tag{4.5}$$

The model $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ is called the *inference model*. As we show, next to approximating the "true" posterior, the inference model is also crucial in finding the parameters $\boldsymbol{\theta}$ that (approximately) optimize the model evidence.

Now for any choice of $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$, it holds that the log of the model evidence equals:

$$\begin{aligned}
\log_2 p_{\boldsymbol{\theta}}(\mathbf{x}) &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log_2 p_{\boldsymbol{\theta}}(\mathbf{x})\right] \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log_2\left[\frac{p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\right]\right] \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log_2\left[\frac{p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\frac{q_{\boldsymbol{\phi}}(\mathbf{x}|\mathbf{z})}{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\right]\right] \\
&= \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log_2\left[\frac{p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\right]\right]}_{\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x})\ \text{(ELBO)}} + \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log_2\left[\frac{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\right]\right]}_{D_{\mathrm{KL}}\left(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\ \|\ p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\right)}
\end{aligned} \tag{4.6}$$

Where the derivation in step 2 follows from Bayes theorem. The first term of the resulting equation is the called the *Evidence Lower BOund (ELBO)*

$$\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log_2 p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) + \log_2 p(\mathbf{z}) - \log_2 q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\right] \tag{4.7}$$

and the second term is the the Kullback-Leibler (KL) divergence between the inference model and the "true" posterior. The Kullback-Leibler divergence satisfies

$$D_{\mathrm{KL}}\left(q_{\boldsymbol{\phi}}(\mathbf{x},\mathbf{z}) \parallel p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\right) \geq 0 \tag{4.8}$$

and is equal to 0 if $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \equiv p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$. Due to the nonnegativity of the Kullback-Leibler divergence, the ELBO is a **lower bound** to the log of the model evidence $\log_2 p_{\boldsymbol{\theta}}(\mathbf{x})$. The tightness of the bound is determined by the KL-divergence:

$$\begin{aligned}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) &= \log_2 p_{\boldsymbol{\theta}}(\mathbf{x}) + D_{\mathrm{KL}}\left(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \parallel p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\right) \\ &\leq \log_2 p_{\boldsymbol{\theta}}(\mathbf{x})\end{aligned} \tag{4.9}$$

As we explain in Section 4.1.4, the ELBO can be maximized using MLE. By looking at equation 4.9 it can be understood that maximization of the ELBO with respect to the latent variable model parameters $\boldsymbol{\theta}$ and inference model parameters $\boldsymbol{\phi}$, will jointly optimize two entitites:

1. Approximately maximize the log of the model evidence $\log_2 p_{\boldsymbol{\theta}}(\mathbf{x})$. This means that we are finding the parameters $\boldsymbol{\theta}$ for the distribution $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ that approximately optimize the model evidence $p_{\boldsymbol{\theta}}(\mathbf{x})$.
2. Approximately minimize the Kullback-Leibler divergence of the inference model $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ from the true posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$, which means we are finding the parameters $\boldsymbol{\phi}$ such that $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ closely resembles the true posterior.

Therefore, by maximizing the ELBO using MLE, the ELBO comes closer to the log of the model evidence

$$\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) \rightarrow \log_2 p_{\boldsymbol{\theta}}(\mathbf{x}) \tag{4.10}$$

while concurrently minimizing Kullback-Leibler divergence between the model evidence and the underlying distribution:

$$D_{\mathrm{KL}}\left(p^*(\mathbf{x}) \parallel p_{\boldsymbol{\theta}}(\mathbf{x})\right) \rightarrow 0 \tag{4.11}$$

### 4.1.3 Specification of the Model Distributions

Before we can discuss optimization of the latent variable model, we must specify the prior, likelihood and inference model distributions:

$$\begin{aligned}\text{prior: } & p(\mathbf{z}) \\ \text{likelihood: } & p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) \\ \text{inference model: } & q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\end{aligned} \tag{4.12}$$

Assuming continuous latent variables $\mathbf{z}$ means that we have to choose the distributions of the prior $p(\mathbf{z})$ and inference model $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ accordingly. We choose a Logistic distribution, because of its heavier tails. This gives more numerical stability during optimization (Kingma, Abbeel, and Ho, 2019). The Logistic distribution has two distribution parameters: the location $\mu$ and scale $s$. The prior $p(\mathbf{z})$ density is chosen to be a product of independent Logistic distribution density functions whose location

and scale parameters are not changed during optimization:

$$\begin{aligned} p(z_j) &= \text{Logistic}\left(z_j;\ \{\mu = 0, s = 1\}\right) \\ p(\mathbf{z}) &= \prod_{j=1}^{M} p(z_j) \end{aligned} \tag{4.13}$$

The distribution parameters of the inference model $q_{\boldsymbol{\phi}}(\mathbf{x}|\mathbf{z})$ density are chosen to be a non-linear differentiable function $g_{\boldsymbol{\phi}}$ of $\mathbf{x}$ with changeable parameters $\boldsymbol{\phi}$. Note that the function parameters $\boldsymbol{\phi}$ are explicitly modified during optimization, whereas the distribution parameters are implicitly modified. The density $q_{\boldsymbol{\phi}}(\mathbf{x}|\mathbf{z})$ is written as a product of conditionally independent Logistic densities:

$$\begin{aligned} (\boldsymbol{\mu}, \mathbf{s}) &= g_{\boldsymbol{\phi}}(\mathbf{x}) \\ q_{\boldsymbol{\phi}}(z_j|\mathbf{x}) &= \text{Logistic}\left(z_j;\ \{\mu_j, s_j\}\right) \\ q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) &= \prod_{j=1}^{M} q_{\boldsymbol{\phi}}(z_j|\mathbf{x}) \end{aligned} \tag{4.14}$$

Finally, we have to define the distribution over the observed variables $\mathbf{x}$. Every dimension takes on a discrete value from an alphabet $\mathcal{X} = \{s_1, \ldots, s_{|\mathcal{X}|}\}$. The likelihood distribution needs to specify a probability for every symbol for every dimension. The probabilities are denoted by the following matrix $\mathbf{P}_{\mathbf{x}|\mathbf{z}}$, where every row represents a dimension:

$$\mathbf{P}_{\mathbf{x}|\mathbf{z}} = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1|\mathcal{X}|} \\ p_{21} & p_{22} & \cdots & p_{2|\mathcal{X}|} \\ \vdots & \vdots & \ddots & \vdots \\ p_{P1} & p_{P2} & \cdots & p_{P|\mathcal{X}|} \end{bmatrix} \tag{4.15}$$

The probabilities $\mathbf{P}_{\mathbf{x}|\mathbf{z}}$ are chosen to the output of a non-linear differentiable function $f_{\boldsymbol{\theta}}$ of $\mathbf{z}$ with changeable parameters $\boldsymbol{\theta}$. This means that the likelihood $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ is a product of conditionally independent Categorical probability mass functions:

$$\begin{aligned} \mathbf{P}_{\mathbf{x}|\mathbf{z}} &= f_{\boldsymbol{\theta}}(\mathbf{z}) \\ p_{\boldsymbol{\theta}}(x_i|\mathbf{z}) &= \text{Categorical}(x_i;\ \{p_{i1}, \ldots, p_{i|\mathcal{X}|}\}) \\ p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) &= \prod_{i=1}^{|X|} p_{\boldsymbol{\theta}}(x_i|\mathbf{z}) \end{aligned} \tag{4.16}$$

We use neural networks as non-linear differentiable functions $f_{\boldsymbol{\theta}}$ and $g_{\boldsymbol{\phi}}$, where $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ denote the parameters (sometimes called the *weights*) of the network. Optimizing the values of the parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ is done using stochastic gradient descent (see Section 4.1.4). Neural networks are arguably successful in a wide range of machine learning tasks (LeCun, Bengio, and Hinton, 2015; Goodfellow, Bengio, and Courville, 2016), and more specifically, as critical component in latent variable models (Kingma and Welling, 2019). But most importantly: neural network functions can be computed (and typically differentiated) in parallel on GPU chips.

This means that we can calculate the values of the outputs of a neural network function in parallel. The outputs corresponding to the inference model and likelihood are $(\boldsymbol{\mu}, \mathbf{s})$ and $\mathbf{P}_{\mathbf{x}|\mathbf{z}}$ respectively. Using GPU chips, we can also calculate the densities $q_{\boldsymbol{\phi}}(z_j|\mathbf{x})$ in parallel $\forall j \in \{1, \ldots, M\}$. As we explain in Section 4.2.2, the property of computing $\mathbf{P}_{\mathbf{x}|\mathbf{z}}$ and all $q_{\boldsymbol{\phi}}(z_j|\mathbf{x})$ in parallel makes this model design suitable for use in a fast compression scheme. Neural networks are explained in more depth in Appendix B. The specific neural network architectures we used can be found in (Kingma, Abbeel, and Ho, 2019).

### 4.1.4 Optimization of the Model

The ELBO $\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x})$ allows us to perform MLE with respect to the neural network parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ using a method called stochastic gradient descent. Given a dataset $\mathcal{D}$, our goal is to maximize the ELBO criterion (the average of the individual-datapoint ELBO's) with respect to $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$:

$$\underset{\boldsymbol{\theta},\boldsymbol{\phi}}{\operatorname{argmax}} \frac{1}{N_{\mathcal{D}}} \sum_{i=1}^{N_{\mathcal{D}}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i) \tag{4.17}$$

However, stochastic gradient descent relies on computing the ELBO per datapoint

$$\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^i)}\left[\log_2 p_{\boldsymbol{\theta}}(\mathbf{x}^i|\mathbf{z}) + \log_2 p(\mathbf{z}) - \log_2 q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^i)\right] \tag{4.18}$$

and its gradients $\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i)$ and $\nabla_{\boldsymbol{\phi}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i)$. All three functions of $\mathbf{x}^i$ are intracTable because of the expectation with respect to the inference model $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^i)$.

We circumvent computing the expectation by estimating $\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i)$, $\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i)$ and $\nabla_{\boldsymbol{\phi}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i)$ using Monte-Carlo estimates. The ELBO can be approximated by sampling $\mathbf{z}^i \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^i)$ for every datapoint $\mathbf{x}^i$:

$$\begin{aligned}
\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i) &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^i)}\left[\log_2 p_{\boldsymbol{\theta}}(\mathbf{x}^i|\mathbf{z}) + \log_2 p(\mathbf{z}) - \log_2 q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^i)\right] \\
&\simeq \log_2 p_{\boldsymbol{\theta}}(\mathbf{x}^i|\mathbf{z}^i) + \log_2 p_{\boldsymbol{\theta}}(\mathbf{z}^i) - \log_2 q_{\boldsymbol{\phi}}(\mathbf{z}^i|\mathbf{x}^i), \quad \text{with } \mathbf{z}^i \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^i) \\
&\equiv \overline{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i)
\end{aligned} \tag{4.19}$$

Similarly, we obtain an estimate $\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i)$ with respect to the parameters $\boldsymbol{\theta}$:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i) &= \nabla_{\boldsymbol{\theta}}\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^i)}\left[\log_2 p_{\boldsymbol{\theta}}(\mathbf{x}^i|\mathbf{z}) + \log_2 p(\mathbf{z}) - \log_2 q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^i)\right] \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^i)}\left[\nabla_{\boldsymbol{\theta}}\left(\log_2 p_{\boldsymbol{\theta}}(\mathbf{x}^i|\mathbf{z}) + \log_2 p(\mathbf{z}) - \log_2 q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^i)\right)\right] \\
&\simeq \nabla_{\boldsymbol{\theta}} \log_2 p_{\boldsymbol{\theta}}(\mathbf{x}^i|\mathbf{z}^i) + \log_2 p_{\boldsymbol{\theta}}(\mathbf{z}^i), \quad \text{with } \mathbf{z}^i \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^i) \\
&\equiv \nabla_{\boldsymbol{\theta}}\overline{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i)
\end{aligned} \tag{4.20}$$

Estimating (or calculating) $\nabla_{\boldsymbol{\phi}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i)$ with respect to the parameters $\boldsymbol{\phi}$, however, is not so straightforward:

$$\begin{aligned}\nabla_{\boldsymbol{\phi}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i) &= \nabla_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^i)} \left[\log_2 p_{\boldsymbol{\theta}}(\mathbf{x}^i|\mathbf{z}) + \log_2 p(\mathbf{z}) - \log_2 q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^i)\right] \\ &\neq \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^i)} \left[\nabla_{\boldsymbol{\phi}} \left(\log_2 p_{\boldsymbol{\theta}}(\mathbf{x}^i|\mathbf{z}) + \log_2 p(\mathbf{z}) - \log_2 q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^i)\right)\right]\end{aligned} \tag{4.21}$$

The VAE-framework (Kingma and Welling, 2013; Rezende, Mohamed, and Wierstra, 2014) demonstrates that for continuous latent variables $\mathbf{z}$, we can efficiently construct an estimate of $\nabla_{\boldsymbol{\phi}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i)$ by *change of variables*, denoted by $\nabla_{\boldsymbol{\phi}} \overline{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i)$. The necessary steps of this procedure and how the ELBO is differentiated with respect to the parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ is beyond the scope of this work. The general procedure of stochastic gradient descent applied to the ELBO is explained in Appendix C.

## 4.2 The Compression Scheme

The goal is to compress data $\mathbf{x}^\circ$. However, it is not straightforward to use a latent variable model for compression, but it is possible with the help of the inference model. The idea is the following: *we compress both the data* $\mathbf{x}^\circ$ ***and*** *a sample* $\mathbf{z}^\circ$ *of the "high-level features" that correspond to that data* $\mathbf{z}^\circ \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^\circ)$. Nonetheless, the latent variables $\mathbf{z}$ are continuous and we only defined lossless compression of finite discrete values. Therefore, we discretize the latent variables, explained in Section 4.2.1. After that, in Section 4.2.2, we describe an impractical *naive scheme* that uses the discretized latent variables. Finally, in Section 4.2.3, we describe the *bits-back scheme* that improves upon the naive scheme.

### 4.2.1 Discretization of the Latent Space

Every latent variable $z_j \in (z_1, \ldots, z_M)$ gets discretized into $D$ bins. Specifically, we tile the space $\mathbb{R}$ of every dimension into bins of width $\delta_{z_j}$. For $z_j \in \mathbb{R}$, let $B(z_j)$ be the unique bin that contains $z_j$, and let $\overline{z}_j$ be the center of the bin $B(z_j)$. We call $\overline{z}_j$ the discretized version of $z_j$. Every center $\overline{z}_j$ of bin $B(z_j)$ is interpreted as one symbol in the alphabet $\{\overline{z}_{j1}, \ldots, \overline{z}_{jD}\}$. Specification of the size of the bins is explained in (Kingma, Abbeel, and Ho, 2019).

The corresponding probability mass function and its multivariate counterpart of the inference model are defined as:

$$\begin{aligned}\overline{q}_{\boldsymbol{\phi}}(\overline{z}_j|\mathbf{x}) &= \int_{B(\overline{z}_j)} q_{\boldsymbol{\phi}}(z_j|\mathbf{x})\, dz_j \\ \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}|\mathbf{x}) &= \prod_{j=1}^{M} \overline{q}_{\boldsymbol{\phi}}(\overline{z}_j|\mathbf{x})\end{aligned} \tag{4.22}$$

where $\overline{\mathbf{z}}$ is the discretized version of $\mathbf{z}$. If the number of bins $E$ is large and the density function $q_{\boldsymbol{\phi}}(z_j|\mathbf{x})$ is sufficiently smooth, the probability mass function and its multivariate counterpart take on the form:

$$\begin{aligned}\overline{q}_{\boldsymbol{\phi}}(\overline{z}_j|\mathbf{x}) &\approx q_{\boldsymbol{\phi}}(\overline{z}_j|\mathbf{x})\delta_{z_j} \\ \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}|\mathbf{x}) &\approx q_{\boldsymbol{\phi}}(\overline{\mathbf{z}}|\mathbf{x})\delta_{\mathbf{z}}\end{aligned} \tag{4.23}$$

where $\delta_{\mathbf{z}} = \prod_{j=1}^{M} \delta_{z_j}$ (Ho, Lohn, and Abbeel, 2019). However, as we explain in Section 4.2.3 and show in Equation 4.30, the number of discretization bins cannot be

chosen too large. Furthermore, we define a probability matrix that contains the probability of every symbol of every latent variable $\overline{z}_{jk}$:

$$\mathbf{Q}_{\overline{\mathbf{z}}|\mathbf{x}} = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1D} \\ p_{21} & p_{22} & \cdots & p_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ p_{M1} & p_{M2} & \cdots & p_{MD} \end{bmatrix} \qquad \text{with} \qquad p_{jk} = \overline{q}_{\boldsymbol{\phi}}(\overline{z}_{jk}|\mathbf{x}) \tag{4.24}$$

The probability mass function $\overline{p}(\overline{\mathbf{z}})$ and probability matrix $\mathbf{P}_{\overline{\mathbf{z}}}$ corresponding to the prior $p(\mathbf{z})$ are defined analogously using the same discretization bins.

### 4.2.2 Naive Scheme

Now, assume we operate under the following conditions. Both the sender and receiver have access to an ANS coder and the model. Furthermore, both the sender and receiver are able to calculate the matrices $\mathbf{P}_{\mathbf{x}|\overline{\mathbf{z}}}$, $\mathbf{P}_{\overline{\mathbf{z}}}$ and $\mathbf{Q}_{\overline{\mathbf{z}}|\mathbf{x}}$ and the corresponding distributions $p_{\boldsymbol{\theta}}(\mathbf{x}|\overline{\mathbf{z}})$, $\overline{p}(\overline{\mathbf{z}})$ and $\overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}|\mathbf{x})$ fast using GPU-chips.

We first determine $\mathbf{P}_{\mathbf{x}|\overline{\mathbf{z}}^\circ}$ by conditioning on a sample from the inference model $\overline{\mathbf{z}}^\circ \sim \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}|\mathbf{x}^\circ)$. The latent sample $\overline{\mathbf{z}}^\circ$ can then be interpreted as probable "high-level features" that correspond to the data sample $\mathbf{x}^\circ$, likely making the distribution $p_{\boldsymbol{\theta}}(\mathbf{x}|\overline{\mathbf{z}}^\circ)$ accurate. Afterwards, the sender provides ANS both $\mathbf{x}^\circ$ and $\mathbf{P}_{\mathbf{x}|\overline{\mathbf{z}}^\circ}$. The ANS coder then encodes the data $\mathbf{x}^\circ$ into a bitstream using $\mathbf{P}_{\mathbf{x}|\overline{\mathbf{z}}^\circ}$, resulting in a codelength of

$$l(\mathbf{x}^\circ) = -\log_2 p_{\boldsymbol{\theta}}(\mathbf{x}^\circ|\overline{\mathbf{z}}^\circ) \tag{4.25}$$

However, if the receiver wishes to decode an exact reconstruction of the data $\mathbf{x}^\circ$ that the sender intended to communicate, he needs to use the exact same matrix $\mathbf{P}_{\mathbf{x}|\overline{\mathbf{z}}^\circ}$ as the sender did for encoding. This is only possible if the receiver *knows* the exact latent data $\overline{\mathbf{z}}^\circ$ the sender sampled from $q_{\boldsymbol{\phi}}(\overline{\mathbf{z}}|\mathbf{x}^\circ)$. As the sender and receiver only communicate through the bitstream, it is impossible for the receiver to know beforehand what the sampled latent data $\overline{\mathbf{z}}^\circ$ is. The only way the sender is able to communicate the latent sample is by encoding the values $\overline{\mathbf{z}}^\circ$ as well.

The corresponding scheme for the sender is described in Table 4.1. Every encode operation is done with the ANS coder. The third column represents the amount of bits that we add to the bitstream following from the operation. We included an illustration of a bitstream resulting from this scheme in the last column. In practice, the resulting bitstream will likely be much longer.

TABLE 4.1: Steps of the naive scheme at the sender's end. Every encode operation is done with the ANS coder.

| # | **Operation** | | | | **Δ Bits** | | | **Bitstream** (illustr.) |
|---|---|---|---|---|---|---|---|---|
| - | Sample | $\overline{\mathbf{z}}^\circ$ | from | $\overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}\|\mathbf{x}^\circ)$ | - | | | - |
| 1 | Encode | $\mathbf{x}^\circ$ | using | $\mathbf{P}_{\mathbf{x}\|\overline{\mathbf{z}}^\circ}$ | adds | $-\log_2 p_{\boldsymbol{\theta}}(\mathbf{x}\|\overline{\mathbf{z}}^\circ)$ | bits | 101111011 |
| 2 | Encode | $\overline{\mathbf{z}}^\circ$ | using | $\mathbf{P}_{\overline{\mathbf{z}}}$ | adds | $-\log_2 \overline{p}(\overline{\mathbf{z}}^\circ)$ | bits | 101111011001 |

The resulting codelength is then interpreted as the accumulated codelengths of $\mathbf{x}^\circ$ *and* $\overline{\mathbf{z}}^\circ$:

$$l_{\text{naive}}(\mathbf{x}^\circ, \overline{\mathbf{z}}^\circ) = -\log_2 p_{\boldsymbol{\theta}}(\mathbf{x}^\circ|\overline{\mathbf{z}}^\circ) - \log_2 \overline{p}(\overline{\mathbf{z}}^\circ) \tag{4.26}$$

This proces is clearly reversible, by reversing the order of operation and replacing encode with decode operations. Therefore, the corresponding scheme for the **receiver** is described in Table 4.2.

TABLE 4.2: Steps of the naive scheme at the receiver's end.

| # | **ANS Operation** | | | | **Δ Bits** | | | **Bitstream** (illustr.) |
|---|---|---|---|---|---|---|---|---|
| - | - | | | | - | | | 101111011001 |
| 2 | Decode | $\overline{\mathbf{z}}^\circ$ | using | $\mathbf{P}_{\overline{\mathbf{z}}}$ | takes | $-\log_2 \overline{p}(\overline{\mathbf{z}}^\circ)$ | bits | 101111011 |
| 1 | Decode | $\mathbf{x}^\circ$ | using | $\mathbf{P}_{\mathbf{x}\|\overline{\mathbf{z}}^\circ}$ | takes | $-\log_2 p_{\boldsymbol{\theta}}(\mathbf{x}^\circ|\overline{\mathbf{z}}^\circ)$ | bits | - |

Consequently, the data $\mathbf{x}^\circ$ and the sample $\overline{\mathbf{z}}^\circ$ are recovered in a lossless manner at the receiver's end.

### 4.2.3 Bits-Back Scheme

We can do better than the naive compression scheme described in Section 4.2.2 using the bits-back argument (Wallace, 1990; Hinton and Van Camp, 1993), which lets us achieve a codelength that is $-\log \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}^\circ|\mathbf{x}^\circ)$ shorter than the naive scheme. This way we are able to nearly "cancel" the number of bits we need to compress the latent sample $\overline{\mathbf{z}}^\circ$.

In order to achieve that, we have to loosen up our assumptions about the sender and receiver. In the introduction, we proposed using the sender-receiver framework for a coherent narrative. Up until now, we assumed that the sender was only able to perform *encoding* operations. Analogously, the receiver was only able to perform *decoding* operations. In order to improve upon the naive compression scheme, we have to assume that both the sender and receiver are able to manipulate a bitstream by performing both encoding *and* decoding operations on it.

Now, suppose the sender wishes to compress data $\mathbf{x}^\circ$. Assume that we operate under the same conditions as explained in Section 4.2.2. Additionally, we **must** assume that the sender wishes to communicate some **other data** as well. The sender has already encoded the other data onto the bitstream, resulting in a bitstream length of $N_{\text{init}}$ bits. These bits are called *initial bits*. Assume that the receiver knows how to decode the initial bits to retrieve the other data. As an illustration, assume that the initial bits are:

$$\underbrace{111010101000}_{N_{\text{init}} \text{ bits}} \tag{4.27}$$

In the naive scheme, the sender would begin with the step 'Sample $\overline{\mathbf{z}}^\circ \sim \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}|\mathbf{x}^\circ)$'. However, the sender can now use ANS and $\mathbf{Q}_{\overline{\mathbf{z}}|\mathbf{x}^\circ}$ to *decode* some bits of the initial bits to generate a sample $\overline{\mathbf{z}}^\circ$, shrinking the initial bits by $-\log_2 \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}^\circ|\mathbf{x}^\circ)$. For example, after decoding $\overline{\mathbf{z}}^\circ$ from the initial bits, we are left with the following bits:

$$11101010 \tag{4.28}$$

The values of the latent sample $\overline{\mathbf{z}}^\circ$ depend on what the initial bits are. To understand this, remember that ANS and a matrix of probabilities $\mathbf{Q}_{\overline{\mathbf{z}}|\mathbf{x}^\circ}$ together form a bijective (one-to-one) mapping between symbols $\overline{\mathbf{z}}^\circ$ and their representation $C[\overline{\mathbf{z}}^\circ]$. In our case, ANS (together with $\mathbf{Q}_{\overline{\mathbf{z}}|\mathbf{x}^\circ}$) can be interpreted as searching from the right side of the initial bits to find a part that matches a representation that maps to one of the the configurations of the latent variables $\overline{\mathbf{z}}$. In our example, the representation that ANS finds is 1000, which is then removed from the bitstream and converted into $\overline{\mathbf{z}}^\circ$:

$$11101010\,\boxed{1000} \rightarrow \overline{\mathbf{z}}^\circ \tag{4.29}$$

Since we do not "know" beforehand what kind of initial bits there are, we can interpret the generated latent sample $\overline{\mathbf{z}}^\circ$ as a *random sample* from the inference model $\overline{\mathbf{z}}^\circ \sim \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}|\mathbf{x}^\circ)$. The number of bits that we remove using ANS and $\mathbf{Q}_{\overline{\mathbf{z}}|\mathbf{x}^\circ}$ is equal to:

$$\begin{aligned} N_{\text{cost}} &= -\log_2 \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}^\circ|\mathbf{x}^\circ) \\ &\approx -\log_2 q_{\boldsymbol{\phi}}(\overline{\mathbf{z}}^\circ|\mathbf{x}^\circ) - \log_2 \delta_{\mathbf{z}} \end{aligned} \tag{4.30}$$

We call $N_{\text{cost}}$ the *initial cost* of the bits-back scheme. The initial cost grows with the number of discretization bins $D$. So we have to be careful when choosing the number of discretization bins. Furthermore, ANS crashes if it is not able to find a representation that corresponds to one of configurations of the latent variable $\overline{\mathbf{z}}^\circ$. Therefore, the length $N_{\text{init}}$ of the initial bits from which he decodes must be large enough to contain a representation that maps to one of the latent variables configurations. This means that the sender must have encoded enough other data before executing the bits-back scheme.

If we begin with an initial bitstream, the step 'Sample $\overline{\mathbf{z}}^\circ \sim \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}|\mathbf{x}^\circ)$' in Table 4.3 can be substituted by 'Decode $\overline{\mathbf{z}}^\circ$ using $\mathbf{Q}_{\overline{\mathbf{z}}|\mathbf{x}^\circ}$'. The corresponding scheme at the **sender**'s end is described in Table 4.3.

TABLE 4.3: Steps of the bits-back scheme at the sender's end.

| # | **ANS Operation** | | | | **Δ Bits** | | | **Bitstream** (illustr.) |
|---|---|---|---|---|---|---|---|---|
| - | - | | | | - | | | 111010101000 |
| 1 | Decode | $\overline{\mathbf{z}}^\circ$ | using | $\mathbf{Q}_{\overline{\mathbf{z}}\|\mathbf{x}^\circ}$ | takes | $-\log_2 \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}^\circ\|\mathbf{x}^\circ)$ | bits | 11101010 |
| 2 | Encode | $\mathbf{x}^\circ$ | using | $\mathbf{P}_{\mathbf{x}\|\overline{\mathbf{z}}^\circ}$ | adds | $-\log_2 p_{\boldsymbol{\theta}}(\mathbf{x}^\circ\|\overline{\mathbf{z}}^\circ)$ | bits | 1110101010100011 |
| 3 | Encode | $\overline{\mathbf{z}}^\circ$ | using | $\mathbf{P}_{\overline{\mathbf{z}}}$ | adds | $-\log_2 \overline{p}(\overline{\mathbf{z}}^\circ)$ | bits | 1110101010100011001 |

The codelength resulting from encoding $\mathbf{x}^\circ$, $\overline{\mathbf{z}}^\circ$ and the "other data" is equal to:

$$l_{\text{total}}(\text{"other data"}, \mathbf{x}^\circ, \overline{\mathbf{z}}^\circ) = N_{\text{init}} + \log \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}^\circ|\mathbf{x}^\circ) - \log p_{\boldsymbol{\theta}}(\mathbf{x}^\circ|\overline{\mathbf{z}}^\circ) - \log \overline{p}(\overline{\mathbf{z}}^\circ) \tag{4.31}$$

In order to exactly retrieve $\mathbf{x}^\circ$, $\overline{\mathbf{z}}^\circ$ and the "other data" the receiver reverses the order and replaces a decode with an encode operation and vice versa. The corresponding scheme can be found in Table 4.4.

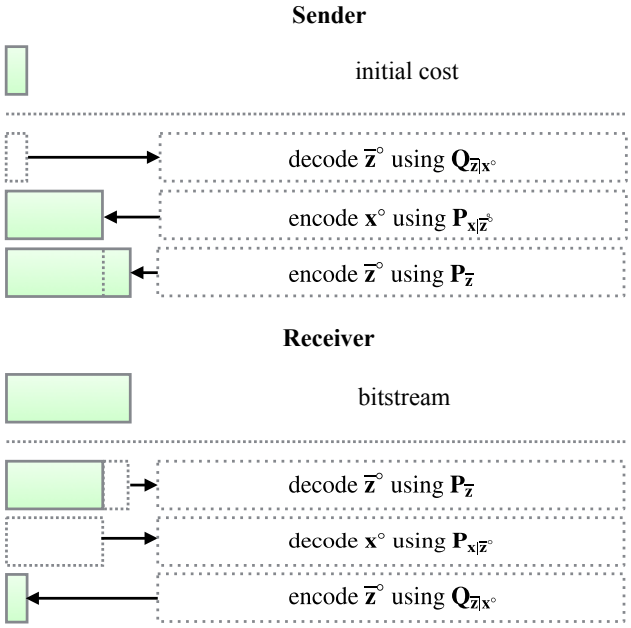



FIGURE 4.1: Lossless compression through bits-back coding with latent variable models.

TABLE 4.4: Steps of the bits-back scheme at the receiver's end

| # | **ANS Operation** | | | | **Δ Bits** | | | **Bitstream** (illustr.) |
|---|---|---|---|---|---|---|---|---|
| - | - | | | | - | | | 1110101010100011001 |
| 3 | Decode | $\overline{\mathbf{z}}^\circ$ | using | $\mathbf{P}_{\overline{\mathbf{z}}}$ | takes | $-\log_2 \overline{p}(\overline{\mathbf{z}}^\circ)$ | bits | 1110101010100011 |
| 2 | Decode | $\mathbf{x}^\circ$ | using | $\mathbf{P}_{\mathbf{x}\|\overline{\mathbf{z}}^\circ}$ | takes | $-\log_2 p_{\boldsymbol{\theta}}(\mathbf{x}^\circ\|\overline{\mathbf{z}}^\circ)$ | bits | 11101010 |
| 1 | Encode | $\overline{\mathbf{z}}^\circ$ | using | $\mathbf{Q}_{\overline{\mathbf{z}}\|\mathbf{x}^\circ}$ | adds | $-\log_2 \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}^\circ\|\mathbf{x}^\circ)$ | bits | 111010101000 |

Note that the last step at the receiver's side reverses the first step at the sender's side. Therefore, at the end of this scheme, the receiver has recovered the $N_{\text{init}}$ initial bits. In other words: the receiver gained the initial bits $N_{\text{init}}$ bits "back". After the last step, the receiver can start decoding the "other data" as well. The ordering of the encode and decode steps is called the *bits-back argument* (Wallace, 1990; Hinton and Van Camp, 1993). Townsend, Bird, and Barber, 2019 achieved bits-back compression using latent variable models for the first time with ANS. The procedure for the sender and receiver, including the size of the bitstream after every step, is illustrated in Figure 4.1.

Now, the *net* codelength used to compress the data vector $\mathbf{x}^\circ$ is calculated by "ignoring" the initial bits $N_{\text{init}}$:

$$\begin{aligned} l_{\text{bitsback}}(\mathbf{x}^\circ) &= l_{\text{total}}(\text{"other data"}, \mathbf{x}^\circ) - N_{\text{init}} \\ &= \log_2 \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}^\circ|\mathbf{x}^\circ) - \log_2 p_{\boldsymbol{\theta}}(\mathbf{x}^\circ|\overline{\mathbf{z}}^\circ) - \log_2 \overline{p}(\overline{\mathbf{z}}^\circ) \end{aligned} \tag{4.32}$$

Consequently, the cost to compress $\overline{\mathbf{z}}^\circ$ using the prior is now "cancelled" by first decoding $\overline{\mathbf{z}}^\circ$ from an initial bitstream using the inference model.

### 4.2.4 Bits-Back Scheme Repeated on Multiple Datapoints

As an example for "other data", Townsend, Bird, and Barber (2019) point out that it is possible to repeat the bits-back scheme on a sequence of datapoints $\{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$, where the scheme for datapoint $\mathbf{x}^i$ (except for the first one $\mathbf{x}^1$) uses the bitstream built up thus far as initial bits. Then, we repeat the bits-back scheme on the receiver's end on the datapoints in reverse order $\{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$ to retrieve the sequence. This way the receiver effectively gains the "bits back" after finishing the three steps in Table 4.4 of each datapoint $\mathbf{x}^i$, such that decoding of the next datapoint $\mathbf{x}^{i-1}$ can start. If the sender is only interested in communicating $\{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$ and not other data, the bits that are irrelevant or redundant are the initial bits needed for the bits-back scheme to compress the first datapoint $\mathbf{x}^1$. In practice, the initial bits needed for the bits-back scheme on the first datapoint $\mathbf{x}^1$ are sampled from a Binomial distribution (with $p = 0.5$) where the the length is set to be large $N_{\text{init}} \gg N_{\text{cost}}$. Then, we decode the latent sample $\overline{\mathbf{z}}^1$ using $\mathbf{Q}_{\overline{\mathbf{z}}|\mathbf{x}^\circ}$ from these initial bits, taking up some initial cost. Afterwards, we remove the bits that are left in the bitstream, since these bits do not represent anything. Then, we continue remaining 2 steps of the bits-back scheme for datapoint $\mathbf{x}^i$ and repeat the full bits-back scheme for every other datapoint $\{\mathbf{x}^2, \ldots, \mathbf{x}^N\}$. If we compress enough datapoints $\{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$, the initial cost to decode $\overline{\mathbf{z}}^1$ using $\mathbf{Q}_{\overline{\mathbf{z}}|\mathbf{x}^\circ}$ likely gets insignificant compared to the total length of the bitstream.

### 4.2.5 Analysis of the Expected Codelength

We now analyze the codelength of the bits-back scheme. The goal was to construct representations $C[\mathbf{x}^\circ]$ that have codelength close to the entropy of the underlying distribution on average:

$$L = \mathbb{E}_{p^*}\left[l(\mathbf{x})\right] \to H(p^*) \tag{4.33}$$

In order to calculate the *expected* value of the codelengths $l_{\text{bitsback}}(\mathbf{x})$ of the bits-back scheme we first need to take the expected value of the codelength with respect to the inference model $\overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}|\mathbf{x})$, because we interpret the latent samples $\overline{\mathbf{z}}^\circ$ as random samples $\overline{\mathbf{z}}^\circ \sim \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}|\mathbf{x}^\circ)$. Afterwards, we take the expected value with respect to the underlying distribution $p^*$.

$$L_{\text{bitsback}} = \mathbb{E}_{p^*}\left[\mathbb{E}_{\overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}|\mathbf{x})}\left[\log_2 \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}|\mathbf{x}) - \log_2 p_{\boldsymbol{\theta}}(\mathbf{x}|\overline{\mathbf{z}}) - \log_2 \overline{p}(\overline{\mathbf{z}})\right]\right] \tag{4.34}$$

From Equation 4.23 we see that the expected codelength is approximately equal the the expected value of the negative ELBO with respect to the underlying distribution:

$$\begin{aligned} L_{\text{bitsback}} &\approx \mathbb{E}_{p^*}\left[\mathbb{E}_{\overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}|\mathbf{x})}\left[\log_2 q_{\boldsymbol{\phi}}(\overline{\mathbf{z}}|\mathbf{x}) + \log_2 \delta_{\mathbf{z}} - \log_2 p_{\boldsymbol{\theta}}(\mathbf{x}|\overline{\mathbf{z}}) - \log_2 p(\overline{\mathbf{z}}) - \log_2 \delta_{\mathbf{z}}\right]\right] \\ &\approx \mathbb{E}_{p^*}\left[\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log_2 q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) - \log_2 p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) - \log_2 p(\mathbf{z})\right]\right] \\ &= \mathbb{E}_{p^*}\left[-\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x})\right] \end{aligned} \tag{4.35}$$

From Equations 4.10 and 4.11 we also know that if we maximize the ELBO using MLE, the ELBO comes closer to the log of the model evidence $\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) \to \log_2 p_{\boldsymbol{\theta}}(\mathbf{x})$ while concurrently minimizing the KL-divergence between the model evidence and

the underlying distribution $D_{\mathrm{KL}}\left(p^*(\mathbf{x}) \parallel p_{\boldsymbol{\theta}}(\mathbf{x})\right) \to 0$. Therefore, the expected codelength of the bits-back scheme approaches the entropy:

$$L_{\text{bitsback}} \approx \mathbb{E}_{p^*}\left[-\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x})\right] \to H(p^*) \tag{4.36}$$

Furthermore, from Equations 4.19 and 4.35 we see that we can construct a Monte-Carlo estimate of the expected codelength given a dataset $\mathcal{D}$:

$$-\frac{1}{N_{\mathcal{D}}}\sum_{i=1}^{N_{\mathcal{D}}}\overline{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i) \simeq \mathbb{E}_{p^*}\left[-\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x})\right] \approx L_{\text{bitsback}} \tag{4.37}$$

This comes in convenient whenever we wish to measure the discrepancy between what codelength the model "predicts" and what the compression scheme actually outputs.

# 5 Nested Latent Variable Models & The Recursive Bits-Back Scheme

In this Section, we explain how to improve upon the compression scheme described in Section 4. In Section 5.1, we extend latent variable models by introducing nested latent variable models, a probabilistic model design that tends to be better in modelling complex high-dimensional data. Afterwards, we specify the model distributions and explain how to optimize the model. In Section 5.2, we explain how to extend the bits-back compression scheme to nested latent variable models. We explain two versions: the extended bits-back scheme and the recursive bits-back scheme. Afterwards, we explain how the recursive bits-back scheme improves upon the extended bits-back scheme. Finally, we analyze the expected codelength of both the extended and the recursive bits-back scheme.

## 5.1 Extending the Probabilistic Model

### 5.1.1 Nested Latent Variable Models

We extend the latent variable model described in Section 4.1.1, henceforth referred to as the *conventional* latent variable model, by introducing additional *layers* (or levels) of latent variables.

Assume that $\mathbf{z}_1 \equiv \mathbf{z}$. In Section 4.1.1 we explained that the important advantage of using latent variables $\mathbf{z}_1$ is that even though the prior $p(\mathbf{z}_1)$ and likelihood $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}_1)$ distributions can be relatively simple, the marginal distribution $p_{\boldsymbol{\theta}}(\mathbf{x})$ can model complex dependencies between the variables $\mathbf{x}$:

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \int p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}_1)p(\mathbf{z}_1)\, d\mathbf{z}_1 \tag{5.1}$$

We say that the distribution $p_{\boldsymbol{\theta}}(\mathbf{x})$ is *implicitly* defined through an *explicit* prior $p(\mathbf{z}_1)$ and likelihood $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}_1)$. Instead of explicitly defining a distribution $p(\mathbf{z}_1)$, we introduce additional latent variables $\mathbf{z}_2 = (z_{21}, \ldots, z_{2M})$ that help define a distribution $p_{\boldsymbol{\theta}}(\mathbf{z}_2|\mathbf{z}_1)$ over the first latent variables $\mathbf{z}_1$, along with a distribution $p_{\boldsymbol{\theta}}(\mathbf{z}_2)$. The distribution $p_{\boldsymbol{\theta}}(\mathbf{z}_1)$ is then *implicitly* defined as:

$$p_{\boldsymbol{\theta}}(\mathbf{z}_1) = \int p_{\boldsymbol{\theta}}(\mathbf{z}_1|\mathbf{z}_2)p_{\boldsymbol{\theta}}(\mathbf{z}_2)\, d\mathbf{z}_2 \tag{5.2}$$

The marginal distribution $p_{\boldsymbol{\theta}}(\mathbf{z}_1)$ can now model complex dependencies between the latent variables $\mathbf{z}_1$. Again, instead of explicitly defining a distribution $p_{\boldsymbol{\theta}}(\mathbf{z}_2)$, we introduce a third set of latent variables $\mathbf{z}_3 = (z_{31}, \ldots, z_{3M})$ that help define a distribution $p_{\boldsymbol{\theta}}(\mathbf{z}_2|\mathbf{z}_3)$ over the second latent variables $\mathbf{z}_2$, along with a distribution

$p_\theta(\mathbf{z}_3)$. The distribution $p_\theta(\mathbf{z}_2)$ is then implicitly defined as:

$$p_\theta(\mathbf{z}_2) = \int p_\theta(\mathbf{z}_2|\mathbf{z}_3)p_\theta(\mathbf{z}_3)\, d\mathbf{z}_3 \tag{5.3}$$

We repeat this argument for $L$ sets of latent variables. Every set $\mathbf{z}_l$ of latent variables helps define a distribution of the variables $\mathbf{z}_{l-1}$, where $l \in \{1, \ldots, L\}$ and $\mathbf{z}_0 \equiv \mathbf{x}$. Consequently, we explicitly specify $L$ likelihood distributions and an unconditional prior distribution:

$$p_\theta(\mathbf{x}|\mathbf{z}_1), p_\theta(\mathbf{z}_1|\mathbf{z}_2), \ldots, p_\theta(\mathbf{z}_{L-1}|\mathbf{z}_L), p(\mathbf{z}_L) \tag{5.4}$$

This way we effectively define $L$ latent variable models where every model implicitly models its own set of variables:

$$\begin{aligned} p_\theta(\mathbf{x}) &= \int p_\theta(\mathbf{x}|\mathbf{z}_1)p_\theta(\mathbf{z}_1)\mathrm{d}\mathbf{z}_1 \\ p_\theta(\mathbf{z}_1) &= \int p_\theta(\mathbf{z}_1|\mathbf{z}_2)p_\theta(\mathbf{z}_2)\mathrm{d}\mathbf{z}_2 \\ &\vdots \\ p_\theta(\mathbf{z}_{L-1}) &= \int p_\theta(\mathbf{z}_{L-1}|\mathbf{z}_L)p(\mathbf{z}_L)\mathrm{d}\mathbf{z}_L \end{aligned} \tag{5.5}$$

Every model implicitly defines a marginal distribution in Equation 5.5 that is "nested" in the Equation above that, hence the name *nested latent variable models*.

The latent variable set $\mathbf{z}_l$ is called *latent layer* $l$. We sometimes write $\mathbf{z}_{1:L} = \{\mathbf{z}_1, \ldots, \mathbf{z}_L\}$ if we refer to all latent layers. The model evidence $p_\theta(\mathbf{x})$ can be written in terms of the explicitly defined likelihood distributions and prior:

$$\begin{aligned} p_\theta(\mathbf{x}) &= \int p_\theta(\mathbf{x}|\mathbf{z}_1) \left[ \int p_\theta(\mathbf{z}_1|\mathbf{z}_2) \left[ \int p_\theta(\mathbf{z}_1|\mathbf{z}_2) \left[ \cdots \right] d\mathbf{z}_3 \right] d\mathbf{z}_2 \right] d\mathbf{z}_1 \\ &= \int p_\theta(\mathbf{x}|\mathbf{z}_1)p_\theta(\mathbf{z}_1|\mathbf{z}_2) \cdots p(\mathbf{z}_L)\mathrm{d}\mathbf{z}_{1:L}. \end{aligned} \tag{5.6}$$

Every latent layer can be interpreted as describing an additional layer of "high-level features" of the data, where the features are assumed to have a naturally occurring hierarchical structure. As an illustration: the composition of images may be dependent on the locations of edges and textures, which may be dependent on the locations of objects, which may be dependent on the scene composition, etc. Latent variable models that have a hierarchy defined between the latent variables are more commonly called *hierarchical latent variable models*. Increasing the number of latent layers of a hierarchical latent variable model tends to make them better in modelling complex high-dimensional data (Rezende, Mohamed, and Wierstra, 2014; Gregor et al., 2015; Sønderby et al., 2016; Kingma et al., 2016; Maaløe et al., 2019).

### 5.1.2 The Inference Model & The Evidence Lower Bound

We now explain how to approximate the "true" posteriors of a nested latent variable model and how to derive the corresponding Evidence Lower BOund (ELBO).

According to Bayes theorem, the model posteriors are defined as:

$$\begin{aligned} p_{\boldsymbol{\theta}}(\mathbf{z}_1|\mathbf{x}) &= \frac{p_{\boldsymbol{\theta}}(\mathbf{z}_1)p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}_1)}{p_{\boldsymbol{\theta}}(\mathbf{x})} \\ p_{\boldsymbol{\theta}}(\mathbf{z}_2|\mathbf{z}_1) &= \frac{p_{\boldsymbol{\theta}}(\mathbf{z}_2)p_{\boldsymbol{\theta}}(\mathbf{z}_1|\mathbf{z}_2)}{p_{\boldsymbol{\theta}}(\mathbf{z}_1)} \\ &\vdots \\ p_{\boldsymbol{\theta}}(\mathbf{z}_L|\mathbf{z}_{L-1}) &= \frac{p(\mathbf{z}_L)p_{\boldsymbol{\theta}}(\mathbf{z}_{L-1}|\mathbf{z}_L)}{p_{\boldsymbol{\theta}}(\mathbf{z}_{L-1})} \end{aligned} \tag{5.7}$$

The inference model is defined by $L$ distributions, each approximating one of the $L$ posterior distributions:

$$\begin{aligned} q_{\boldsymbol{\phi}}(\mathbf{z}_1|\mathbf{x}) &\approx p_{\boldsymbol{\theta}}(\mathbf{z}_1|\mathbf{x}) \\ q_{\boldsymbol{\phi}}(\mathbf{z}_2|\mathbf{z}_1) &\approx p_{\boldsymbol{\theta}}(\mathbf{z}_2|\mathbf{z}_1) \\ &\vdots \\ q_{\boldsymbol{\phi}}(\mathbf{z}_L|\mathbf{z}_{L-1}) &\approx p_{\boldsymbol{\theta}}(\mathbf{z}_L|\mathbf{z}_{L-1}) \end{aligned} \tag{5.8}$$

For conciseness, we sometimes refer to the posterior, inference model and likelihood distributions as

$$\begin{aligned} p_{\boldsymbol{\theta}}(\mathbf{z}_{1:L}|\mathbf{x}) &= \prod_{l=1}^{L} p_{\boldsymbol{\theta}}(\mathbf{z}_l|\mathbf{z}_{l-1}) \\ q_{\boldsymbol{\phi}}(\mathbf{z}_{1:L}|\mathbf{x}) &= \prod_{l=1}^{L} q_{\boldsymbol{\phi}}(\mathbf{z}_l|\mathbf{z}_{l-1}) \\ p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}_{1:L-1}|\mathbf{z}_L) &= \prod_{l=0}^{L-1} p_{\boldsymbol{\theta}}(\mathbf{z}_l|\mathbf{z}_{l+1}) \end{aligned} \tag{5.9}$$

respectively, where $\mathbf{z}_0 \equiv \mathbf{x}$. Furthermore, whenever we say "sample $\mathbf{z}_{1:L}$ from $q_{\boldsymbol{\phi}}(\mathbf{z}_{1:L}|\mathbf{x})$", we mean "perform ancestral sampling from the inference model distributions given $\mathbf{x}$ to obtain samples $\{\mathbf{z}_1, \ldots, \mathbf{x}_L\}$". Using the notation defined in Equation 5.9, we can derive the ELBO following the same steps as in Equation 4.6:

$$\begin{aligned} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}_{1:L}|\mathbf{x})}\left[\log_2 p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}_{1:L-1}|\mathbf{z}_L) + \log_2 p(\mathbf{z}_L) - \log_2 q_{\boldsymbol{\phi}}(\mathbf{z}_{1:L}|\mathbf{x})\right] \\ &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}_{1:L}|\mathbf{x})}\left[\sum_{l=0}^{L-1} \log_2 p_{\boldsymbol{\theta}}(\mathbf{z}_l|\mathbf{z}_{l+1}) + \log_2 p(\mathbf{z}_L) - \sum_{l=1}^{L} \log_2 q_{\boldsymbol{\phi}}(\mathbf{z}_l|\mathbf{z}_{l-1})\right] \end{aligned} \tag{5.10}$$

where $\mathbf{z}_0 \equiv \mathbf{x}$. Furthermore, analogous to the conventional latent variable model case, it holds that:

$$\begin{aligned} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) &= \log_2 p_{\boldsymbol{\theta}}(\mathbf{x}) + D_{\mathrm{KL}}\left(q_{\boldsymbol{\phi}}(\mathbf{z}_{1:L}|\mathbf{x}) \parallel p_{\boldsymbol{\theta}}(\mathbf{z}_{1:L}|\mathbf{x})\right) \\ &\leq \log_2 p_{\boldsymbol{\theta}}(\mathbf{x}) \end{aligned} \tag{5.11}$$

Therefore, optimizing the ELBO using MLE enjoys the same properties as the optimizing the ELBO of the conventional latent variable model.

### 5.1.3 Specification of the Model Distributions

We now specify the nested latent variable model distributions. Assume that we have $L$ latent layers. The prior over the top layer $\mathbf{z}_L$ has the same specification as the prior of the conventional latent variable model:

$$\begin{aligned} p(z_{Lj}) &= \text{Logistic}\left(z_{Lj};\ \{\mu = 0, s = 1\}\right) \\ p(\mathbf{z}_L) &= \prod_{j=1}^{M} p(z_{Lj}) \end{aligned} \tag{5.12}$$

The distribution parameters of every inference model distribution over latent layer $\mathbf{z}_l$ with $l \in \{1, \ldots, L\}$ are chosen to be a non-linear differentiable function $g_{\boldsymbol{\phi}_l}$ of $\mathbf{z}_{l-1}$ with changeable parameters $\boldsymbol{\phi}_l$. Consequently, every density $q_{\boldsymbol{\phi}}(\mathbf{z}_l|\mathbf{z}_{l-1})$ is written as a product of conditionally independent Logistic densities:

$$\begin{aligned} (\boldsymbol{\mu}, \mathbf{s}) &= g_{\boldsymbol{\phi}_l}(\mathbf{z}_{l-1}) \\ q_{\boldsymbol{\phi}}(z_{lj}|\mathbf{z}_{l-1}) &= \text{Logistic}\left(z_{lj};\ \{\mu_j, s_j\}\right), \\ q_{\boldsymbol{\phi}}(\mathbf{z}_l|\mathbf{z}_{l-1}) &= \prod_{j=1}^{M} q_{\boldsymbol{\phi}}(z_{lj}|\mathbf{z}_{l-1}), \\ & \quad \forall l \in \{1, \ldots, L\} \text{ and where } \mathbf{z}_0 \equiv \mathbf{x} \end{aligned} \tag{5.13}$$

The distribution parameters of the likelihood distribution over latent layer $\mathbf{z}_l$ with $l \in \{1, \ldots, L-1\}$ are chosen to be a non-linear differentiable function $f_{\boldsymbol{\theta}_l}$ of $\mathbf{z}_{l+1}$ with changeable parameters $\boldsymbol{\theta}_l$. Consequently, every density $p_{\boldsymbol{\theta}}(\mathbf{z}_l|\mathbf{z}_{l+1})$, except for $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}_1)$, is written as a product of Logistic densities:

$$\begin{aligned} (\boldsymbol{\mu}, \mathbf{s}) &= f_{\boldsymbol{\theta},l}(\mathbf{z}_{l+1}) \\ p_{\boldsymbol{\theta}}(z_{lj}|\mathbf{z}_{l+1}) &= \text{Logistic}\left(z_{lj};\ \{\mu_j, s_j\}\right) \\ p_{\boldsymbol{\theta}}(\mathbf{z}_l|\mathbf{z}_{l+1}) &= \prod_{j=1}^{M} p_{\boldsymbol{\theta}}(z_{lj}|\mathbf{z}_{l+1}), \\ & \quad \forall l \in \{1, \ldots, L-1\} \end{aligned} \tag{5.14}$$

The distribution parameters of the likelihood distribution $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}_1)$ are chosen to be a non-linear differentiable function $f_{\boldsymbol{\theta}_0}$ of $\mathbf{z}_1$ with changeable parameters $\boldsymbol{\theta}_0$. The likelihood $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}_1)$ is again written as a product of conditionally independent Categorical probability mass functions:

$$\begin{aligned} \mathbf{P}_{\mathbf{x}|\mathbf{z}_1} &= f_{\boldsymbol{\theta},0}(\mathbf{z}_1) \\ p_{\boldsymbol{\theta}}(x_i) &= \text{Categorical}(x_i;\ \{p_{i1}, \ldots, p_{i|\mathcal{X}|}\}) \\ p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}_1) &= \prod_{i=1}^{|\mathcal{X}|} p_{\boldsymbol{\theta}}(x_i) \end{aligned} \tag{5.15}$$

For conciseness, we omitted the subscript of the parameter vectors $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ in the notation of the densities and probability mass function, since this is clear from the context. The functions $g_{\boldsymbol{\phi},l}$ and $f_{\boldsymbol{\theta},l}$ are again chosen to be neural network functions for all $l \in \{1, \ldots, L\}$, because of the advantages described in Section 4.1.3.

### 5.1.4 Optimization of the Model

Given a dataset $\mathcal{D}$, the goal is to maximize the ELBO criterion $\frac{1}{N_\mathcal{D}} \sum_{i=1}^{N_\mathcal{D}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i)$ with respect to parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ of a nested latent variable model. In order to perform stochastic gradient descent, we need to compute the ELBO $\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i)$ and its gradients $\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i)$ and $\nabla_{\boldsymbol{\phi}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i)$. These functions are intracTable because they depend on computing the expectation with respect to $q_{\boldsymbol{\phi}}(\mathbf{z}_{1:L}|\mathbf{x})$.

The procedure to construct Monte-Carlo estimates of these quantities is almost the same as described in Section 4.1.4, only now we need to sample $\mathbf{z}_{1:L} \sim q_{\boldsymbol{\phi}}(\mathbf{z}_{1:L}|\mathbf{x})$ (instead of $\mathbf{z} \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$). Consequently, the ELBO $\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i)$ is estimated as:

$$\begin{aligned} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i) &\simeq \sum_{l=0}^{L-1} \log_2 p_{\boldsymbol{\theta}}(\mathbf{z}_l^i|\mathbf{z}_{l+1}^i) + \log_2 p(\mathbf{z}_L^i) - \sum_{l=1}^{L} \log_2 q_{\boldsymbol{\phi}}(\mathbf{z}_l^i|\mathbf{z}_{l-1}^i), \\ &\quad \text{with } \mathbf{z}_{1:L}^i \sim q_{\boldsymbol{\phi}}(\mathbf{z}_{1:L}|\mathbf{x}^i) \\ &\equiv \overline{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i) \end{aligned} \tag{5.16}$$

where $\mathbf{z}_0^i \equiv \mathbf{x}^i$. The procedure for $\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i)$ is analogous. Again, estimating $\nabla_{\boldsymbol{\phi}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i)$ is not straightforward and done using the VAE-framework.

## 5.2 Extending and Improving the Compression Scheme

We extend the bits-back scheme described in Section 4.2.3 to make it able to use the nested latent variable model to compress data $\mathbf{x}^\circ$. The general idea is analogous as described in Section 4.2: *we compress both the data* $\mathbf{x}^\circ$ ***and*** *samples of* ***all*** *"high-level features"* $\mathbf{z}_{1:L}^\circ$ *that correspond to that data* $\mathbf{z}_{1:L}^\circ \sim q_{\boldsymbol{\phi}}(\mathbf{z}_{1:L}|\mathbf{x}^\circ)$. However, all latent layers have continuous latent variables. Therefore, we discretize the latent space $\mathbf{z}_{1:L}$ as described in Section 5.2.1. After that, we describe how to extend the bits-back scheme for nested latent variable models in Section 5.2.2, called the extended bits-back scheme. Finally, we describe the recursive bits-back scheme that improves upon the extended bits-back scheme by exploiting the structure of nested latent variable models in Section 5.2.3.

### 5.2.1 Discretization of the Latent Space

Every latent variable $z_{lj} \in (z_{l1}, \ldots, z_{lM}) = \mathbf{z}_l$ gets discretized into $D$ bins for all $l \in \{1, \ldots, M\}$. Analogously to the single latent layer case described in Section 4.2.1, we tile the space $\mathbb{R}$ of every dimension into bins of width $\delta_{z_{lj}}$. For $z_{lj} \in \mathbb{R}$, let $B(z_{lj})$ be the unique bin that contains $z_{lj}$, and let $\overline{z}_{lj}$ be the center of the bin $B(z_{lj})$. We call $\overline{z}_{lj}$ the discretized version of $z_{lj}$. Every center $\overline{z}_{lj}$ of bin $B(z_{lj})$ is interpreted as one symbol in the alphabet $\{\overline{z}_{lj1}, \ldots, \overline{z}_{ljD}\}$. Furthermore, define $\overline{\mathbf{z}}_l$ as the discretized version version of $\mathbf{z}_l$. Specification of the size of the bins is explained in (Kingma, Abbeel, and Ho, 2019).

The probability mass function and the probability matrices corresponding to the inference model, likelihood and prior distributions (except for $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}_1)$ are defined analogous to the steps described in Section 4.2.1 and are denoted as:

$$\begin{aligned} q_{\boldsymbol{\phi}}(\overline{\mathbf{z}}_l|\overline{\mathbf{z}}_{l-1}) &\Rightarrow \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}_l|\overline{\mathbf{z}}_{l-1}) \text{ and } \mathbf{Q}_{\overline{\mathbf{z}}_l|\overline{\mathbf{z}}_{l-1}} \quad \forall l \in \{1,\ldots,L\} \\ p_{\boldsymbol{\theta}}(\overline{\mathbf{z}}_l|\overline{\mathbf{z}}_{l+1}) &\Rightarrow \overline{p}_{\boldsymbol{\theta}}(\overline{\mathbf{z}}_l|\overline{\mathbf{z}}_{l+1}) \text{ and } \mathbf{P}_{\overline{\mathbf{z}}_l|\overline{\mathbf{z}}_{l+1}} \quad \forall l \in \{1,\ldots,L-1\} \\ p(\overline{\mathbf{z}}_L) &\Rightarrow \overline{p}(\overline{\mathbf{z}}_L) \text{ and } \mathbf{P}_{\overline{\mathbf{z}}_L} \end{aligned} \tag{5.17}$$

respectively, where $\overline{\mathbf{z}}_0 \equiv \mathbf{x}$. Consequently, if the number of discretization bins is large and the density functions of the inference model, likelihood and prior are sufficiently smooth, the probability mass functions take on the form:

$$\begin{aligned} \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}_l|\overline{\mathbf{z}}_{l-1}) &\approx q_{\boldsymbol{\phi}}(\overline{\mathbf{z}}_l|\overline{\mathbf{z}}_{l-1})\delta_{\mathbf{z}_l} \quad \forall l \in \{1,\ldots,L\} \\ \overline{p}_{\boldsymbol{\theta}}(\overline{\mathbf{z}}_l|\overline{\mathbf{z}}_{l+1}) &\approx p_{\boldsymbol{\theta}}(\overline{\mathbf{z}}_l|\overline{\mathbf{z}}_{l+1})\delta_{\mathbf{z}_l} \quad \forall l \in \{1,\ldots,L-1\} \\ p(\overline{\mathbf{z}}_L) &\approx p(\overline{\mathbf{z}}_L)\delta_{\mathbf{z}_L} \end{aligned} \tag{5.18}$$

respectively, where $\overline{\mathbf{z}}_0 \equiv \mathbf{x}$.

### 5.2.2 Extended Bits-Back Scheme

Now, assume we operate under the following conditions. Both the sender and receiver have access to an ANS coder and the model. Furthermore, both the sender and receiver are able to calculate the matrices $\mathbf{Q}_{\overline{\mathbf{z}}_l|\overline{\mathbf{z}}_{l-1}}$, $\mathbf{P}_{\overline{\mathbf{z}}_l|\overline{\mathbf{z}}_{l+1}}$, $\mathbf{P}_{\overline{\mathbf{z}}_L}$ and $\mathbf{P}_{\mathbf{x}|\overline{\mathbf{z}}_1}$ and the corresponding distributions $\overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}_l|\overline{\mathbf{z}}_{l-1})$, $\overline{p}_{\boldsymbol{\theta}}(\overline{\mathbf{z}}_l|\overline{\mathbf{z}}_{l+1})$, $\overline{p}(\overline{\mathbf{z}}_L)$ and $p_{\boldsymbol{\theta}}(\mathbf{x}|\overline{\mathbf{z}}_1)$ for all $l \in \{1,\ldots,L\}$ fast using GPU-chips, where $\overline{\mathbf{z}}_0 \equiv 0$. Finally, assume that the sender already encoded some other data, resulting in a bitstream that we again call the *initial bits*. The receiver knows how to decode the initial bits to retrieve the other data. As an illustration, assume that the initial bits are:

$$\underbrace{111010101000}_{N_{\text{init}} \text{ bits}} \tag{5.19}$$

In order to communicate $\mathbf{x}^\circ$, the first thing the sender must do is to decode the latent samples $\{\overline{\mathbf{z}}_1^\circ,\ldots,\overline{\mathbf{z}}_L^\circ\}$ from the initial bits. We described the procedure to decode a latent sample from a bitstream in Section 4.2.3. The same procedure is repeated to generate a sample of every latent layer $l \in \{1,\ldots,L\}$. Using the same argument as given in Section 4.2.3, these latent samples $\overline{\mathbf{z}}_{1:L}^\circ$ can be interpreted as random samples from the inference model distributions $\overline{\mathbf{z}}_{1:L}^\circ \sim \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}_{1:L}|\mathbf{x}^\circ)$. After generating the samples, the sender first encodes the data vector $\mathbf{x}^\circ$ and afterwards encodes all the latent samples $\{\overline{\mathbf{z}}_1^\circ,\ldots,\overline{\mathbf{z}}_L^\circ\}$. The resulting scheme at the **sender**'s end is described in Table 5.1.

TABLE 5.1: Steps of the extended bits-back scheme for nested latent variable models at the sender's end.

| # | ANS Operation | | | | Δ Bits | | | Bitstream (illustr.) |
|---|---|---|---|---|---|---|---|---|
| - | - | | | | - | | | 111010101000 |
| 1 | Decode | $\overline{\mathbf{z}}_1^\circ$ | using | $\mathbf{Q}_{\overline{\mathbf{z}}_1\|\mathbf{x}^\circ}$ | takes | $-\log_2 \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}_1^\circ\|\mathbf{x}^\circ)$ | bits | 11101010 |
| 2 | Decode | $\overline{\mathbf{z}}_2^\circ$ | using | $\mathbf{Q}_{\overline{\mathbf{z}}_2\|\overline{\mathbf{z}}_1^\circ}$ | takes | $-\log_2 \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}_2^\circ\|\overline{\mathbf{z}}_1^\circ)$ | bits | 111010 |
| ⋮ | | ⋮ | | | | ⋮ | | ⋮ |
| $L$ | Decode | $\overline{\mathbf{z}}_L^\circ$ | using | $\mathbf{Q}_{\overline{\mathbf{z}}_L\|\overline{\mathbf{z}}_{L-1}^\circ}$ | takes | $-\log_2 \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}_L^\circ\|\overline{\mathbf{z}}_{L-1}^\circ)$ | bits | 111 |
| $L+1$ | Encode | $\mathbf{x}^\circ$ | using | $\mathbf{P}_{\mathbf{x}\|\overline{\mathbf{z}}_1^\circ}$ | adds | $-\log_2 p_{\boldsymbol{\theta}}(\mathbf{x}^\circ\|\overline{\mathbf{z}}_1^\circ)$ | bits | 111110000 |
| $L+2$ | Encode | $\overline{\mathbf{z}}_1^\circ$ | using | $\mathbf{P}_{\overline{\mathbf{z}}_1\|\overline{\mathbf{z}}_2^\circ}$ | adds | $-\log_2 \overline{p}_{\boldsymbol{\theta}}(\overline{\mathbf{z}}_1^\circ\|\overline{\mathbf{z}}_2^\circ)$ | bits | 11111000010 |
| ⋮ | | ⋮ | | | | ⋮ | | ⋮ |
| $2L$ | Encode | $\overline{\mathbf{z}}_{L-1}^\circ$ | using | $\mathbf{P}_{\overline{\mathbf{z}}_{L-1}\|\overline{\mathbf{z}}_L^\circ}$ | adds | $-\log_2 \overline{p}_{\boldsymbol{\theta}}(\overline{\mathbf{z}}_{L-1}^\circ\|\overline{\mathbf{z}}_L^\circ)$ | bits | 1111100001011000 |
| $2L+1$ | Encode | $\overline{\mathbf{z}}_L^\circ$ | using | $\mathbf{P}_{\overline{\mathbf{z}}_L}$ | adds | $-\log_2 \overline{p}(\overline{\mathbf{z}}_L^\circ)$ | bits | 1111100001011000111 |

The codelength resulting from encoding $\{\mathbf{x}^\circ, \overline{\mathbf{z}}_1^\circ, \ldots, \overline{\mathbf{z}}_L^\circ\}$ and the "other data" is equal to

$$\begin{aligned} l_{\text{total}}(\text{"other data"}, \mathbf{x}^\circ, \overline{\mathbf{z}}_1^\circ, \ldots, \overline{\mathbf{z}}_L^\circ) = N_{\text{init}} &+ \log_2 \sum_{l=1}^{L} \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}_l^\circ|\overline{\mathbf{z}}_{l-1}^\circ) \\ &- \log_2 p_{\boldsymbol{\theta}}(\mathbf{x}^\circ|\overline{\mathbf{z}}_1^\circ) \\ &- \sum_{l=l}^{L-1} \log_2 \overline{p}_{\boldsymbol{\theta}}(\overline{\mathbf{z}}_l^\circ|\overline{\mathbf{z}}_{l+1}^\circ) \\ &- \log_2 \overline{p}(\overline{\mathbf{z}}_L^\circ) \end{aligned} \tag{5.20}$$

where $\overline{\mathbf{z}}_0^\circ \equiv \mathbf{x}^\circ$. The initial cost of this scheme is:

$$N_{\text{cost}}^{\text{bitsback}} := \sum_{l=l}^{L} -\log \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}_l^\circ|\overline{\mathbf{z}}_{l-1}^\circ) \tag{5.21}$$

where $\overline{\mathbf{z}}_0^\circ \equiv \mathbf{x}^\circ$. Notice that the initial cost $N_{\text{cost}}^{\text{bitsback}}$ grows with the depth $L$ of the nested latent variable model. It also grows with the number of discretization bins (see Equation 4.30). **This can make the extended bits-back scheme impractical whenever we have a large number of latent layers.**

In order to exactly retrieve $\{\mathbf{x}^\circ, \overline{\mathbf{z}}_1^\circ, \ldots, \overline{\mathbf{z}}_L^\circ\}$ and the "other data", the receiver reverses the order and replaces decode with encode operations and vice versa. The corresponding scheme can be found in Table 5.4.

TABLE 5.2: Steps of the extended bits-back scheme for nested latent variable models at the receiver's end.

| # | ANS Operation | | | | Δ Bits | | | Bitstream (illustr.) |
|---|---|---|---|---|---|---|---|---|
| - | - | | | | - | | | 1111100001011000111 |
| $2L+1$ | Decode | $\overline{\mathbf{z}}_L^\circ$ | using | $\mathbf{P}_{\overline{\mathbf{z}}}$ | takes | $-\log_2 \overline{p}(\overline{\mathbf{z}}_L^\circ)$ | bits | 1111100001011000 |
| $2L$ | Decode | $\overline{\mathbf{z}}_{L-1}^\circ$ | using | $\mathbf{P}_{\overline{\mathbf{z}}_{L-1}\mid\overline{\mathbf{z}}_L^\circ}$ | takes | $-\log_2 \overline{p}_{\boldsymbol{\theta}}(\overline{\mathbf{z}}_{L-1}^\circ\mid\overline{\mathbf{z}}_L^\circ)$ | bits | 11111000010 |
| ⋮ | | ⋮ | | | | ⋮ | | ⋮ |
| $L+2$ | Decode | $\overline{\mathbf{z}}_1^\circ$ | using | $\mathbf{P}_{\overline{\mathbf{z}}_1\mid\overline{\mathbf{z}}_2^\circ}$ | takes | $-\log_2 \overline{p}_{\boldsymbol{\theta}}(\overline{\mathbf{z}}_1^\circ\mid\overline{\mathbf{z}}_2^\circ)$ | bits | 111110000 |
| $L+1$ | Decode | $\mathbf{x}^\circ$ | using | $\mathbf{P}_{\mathbf{x}\mid\overline{\mathbf{z}}_1^\circ}$ | takes | $-\log_2 p_{\boldsymbol{\theta}}(\mathbf{x}^\circ\mid\overline{\mathbf{z}}_1^\circ)$ | bits | 111 |
| $L$ | Encode | $\overline{\mathbf{z}}_L^\circ$ | using | $\mathbf{Q}_{\overline{\mathbf{z}}_L\mid\overline{\mathbf{z}}_{L-1}^\circ}$ | adds | $-\log_2 \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}_L^\circ\mid\overline{\mathbf{z}}_{L-1}^\circ)$ | bits | 111010 |
| ⋮ | | ⋮ | | | | ⋮ | | ⋮ |
| 2 | Encode | $\overline{\mathbf{z}}_2^\circ$ | using | $\mathbf{Q}_{\overline{\mathbf{z}}_2\mid\overline{\mathbf{z}}_1^\circ}$ | adds | $-\log_2 \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}_2^\circ\mid\overline{\mathbf{z}}_1^\circ)$ | bits | 11101010 |
| 1 | Encode | $\overline{\mathbf{z}}_1^\circ$ | using | $\mathbf{Q}_{\overline{\mathbf{z}}_1\mid\mathbf{x}^\circ}$ | adds | $-\log_2 \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}_1^\circ\mid\mathbf{x}^\circ)$ | bits | 111010101000 |

After executing the last $L$ steps, the receiver reconstructed the initial bits the sender started with, thus effectively gaining these bits "back". Therefore, the sender can start decoding the initial bits to retrieve the other data. The procedure for a nested latent variable model with 3 latent layers $\{\overline{\mathbf{z}}_1, \overline{\mathbf{z}}_2, \overline{\mathbf{z}}_3\}$ is illustrated in Figure 5.1(a).

Again, the *net* codelength used to compress $\mathbf{x}^\circ$ is calculated by "ignoring" the initial bits $N_{\text{init}}$:

$$\begin{aligned} l_{\text{bitsback}}(\mathbf{x}^\circ) &= l_{\text{total}}(\text{"other data"}, \mathbf{x}^\circ, \overline{\mathbf{z}}_1^\circ, \dots, \overline{\mathbf{z}}_L^\circ) - N_{\text{init}} \\ &= \sum_{l=1}^{L} \log_2 \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}_l^\circ\mid\overline{\mathbf{z}}_{l-1}^\circ) - \log_2 p_{\boldsymbol{\theta}}(\mathbf{x}^\circ\mid\overline{\mathbf{z}}_1^\circ) \\ &\quad - \sum_{l=l}^{L-1} \log_2 \overline{p}_{\boldsymbol{\theta}}(\overline{\mathbf{z}}_l^\circ\mid\overline{\mathbf{z}}_{l+1}^\circ) - \log_2 \overline{p}(\overline{\mathbf{z}}_L^\circ) \end{aligned} \tag{5.22}$$

Consequently, the cost to encode the latent samples $\{\overline{\mathbf{z}}_1^\circ, \dots, \overline{\mathbf{z}}_L^\circ\}$ is "cancelled" by first decoding them from an initial bitstream using the inference model distributions.

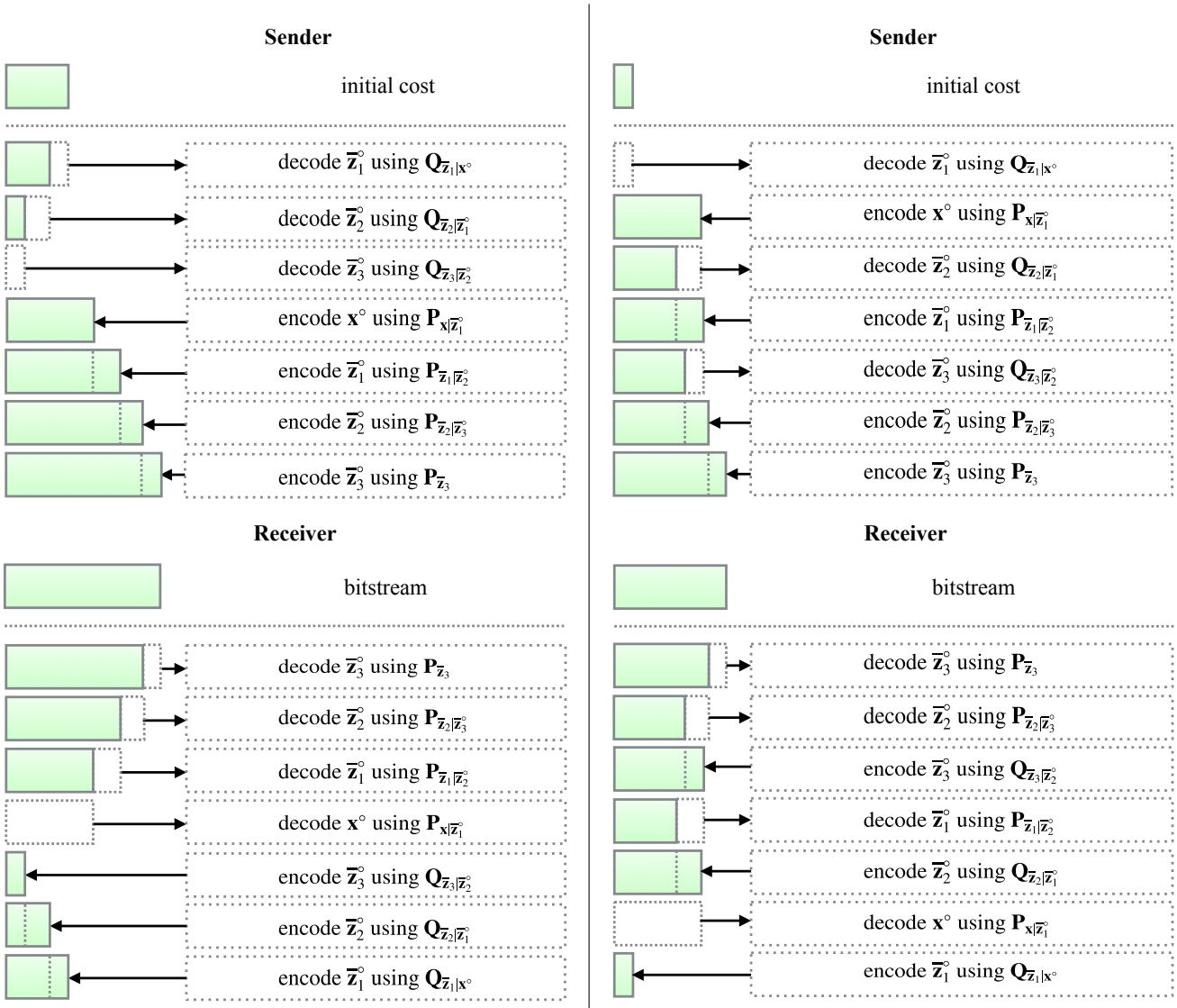
### 5.2.3 Recursive Bits-Back Scheme

We show how to *reduce the initial cost* of the extended bits-back scheme. The idea is to exploit the structure of a nested latent variable model by permuting the order of encode and decode operations.

Assume that we operate under the same conditions as explained in Section 5.2.2. In the extended bits-back scheme, the sender begins by decoding latent samples $\{\overline{\mathbf{z}}_1^\circ, \dots, \overline{\mathbf{z}}_L^\circ\}$, which incurs a large initial cost. In the recursive bits-back scheme, we notice that we can apply the first two steps of the bits-back argument on the first latent layer $\overline{\mathbf{z}}_1$:

$$\begin{aligned} &\text{Decode } \overline{\mathbf{z}}_1^\circ \text{ using } \mathbf{Q}_{\overline{\mathbf{z}}_1\mid\mathbf{x}^\circ} \\ &\text{Encode } \mathbf{x}^\circ \text{ using } \mathbf{P}_{\mathbf{x}\mid\overline{\mathbf{z}}_1^\circ} \end{aligned} \tag{5.23}$$

This adds bits to the bitstream, which means that further decoding operations for samples $\overline{\mathbf{z}}_{2:L}^\circ$ will need fewer initial bits to proceed. Now, we recursively apply the




(a) Extended Bits-Back Scheme

(b) Recursive Bits-Back Scheme

FIGURE 5.1: Lossless compression using the bits-back scheme (left) vs. the recursive bits-back scheme (right) on a nested latent variable model with 3 latent layers $\{\overline{\mathbf{z}}_1, \overline{\mathbf{z}}_2, \overline{\mathbf{z}}_3\}$. Notice that the bits-back scheme has a bigger initial cost compared to the recursive bits-back scheme.

bits-back argument for the second latent layer $\overline{\mathbf{z}}_2$ in a similar fashion:

$$\begin{aligned} &\text{Decode } \overline{\mathbf{z}}_2^\circ \text{ using } \mathbf{Q}_{\overline{\mathbf{z}}_2|\overline{\mathbf{z}}_1^\circ} \\ &\text{Encode } \overline{\mathbf{z}}_1^\circ \text{ using } \mathbf{P}_{\overline{\mathbf{z}}_1|\overline{\mathbf{z}}_2^\circ} \end{aligned} \tag{5.24}$$

Similar decoding and encoding operations can be performed for the remaining latent layers $\overline{\mathbf{z}}_{3:L}^\circ$: right before decoding $\overline{\mathbf{z}}_{l+1}^\circ$, we always encode $\overline{\mathbf{z}}_{l-1}^\circ$, and hence at least $-\log \overline{q}_{\boldsymbol{\theta}}(\overline{\mathbf{z}}_{l-1}^\circ|\overline{\mathbf{z}}_l^\circ)$ are available to decode $\overline{\mathbf{z}}_{l+1}^\circ$ using $\mathbf{Q}_{\overline{\mathbf{z}}_{l+1}|\overline{\mathbf{z}}_l^\circ}$ without an extra cost of initial bits.

The resulting scheme at the **sender**'s side is described in Table 5.3.

TABLE 5.3: Steps of the recursive bits-back scheme for nested latent variable models at the sender's end.

| # | ANS Operation | | | | Δ Bits | | | Bitstream (illustr.) |
|---|---|---|---|---|---|---|---|---|
| - | - | | | | - | | | 111010101000 |
| 1 | Decode | $\overline{\mathbf{z}}_1^\circ$ | using | $\mathbf{Q}_{\overline{\mathbf{z}}_1\|\mathbf{x}^\circ}$ | takes | $-\log_2 \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}_1^\circ\|\mathbf{x}^\circ)$ | bits | 11101010 |
| 2 | Encode | $\mathbf{x}^\circ$ | using | $\mathbf{P}_{\mathbf{x}\|\overline{\mathbf{z}}_1^\circ}$ | adds | $-\log_2 p_{\boldsymbol{\theta}}(\mathbf{x}^\circ\|\overline{\mathbf{z}}_1^\circ)$ | bits | 11101010101011111 |
| 3 | Decode | $\overline{\mathbf{z}}_2^\circ$ | using | $\mathbf{Q}_{\overline{\mathbf{z}}_2\|\overline{\mathbf{z}}_1^\circ}$ | takes | $-\log_2 \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}_2^\circ\|\overline{\mathbf{z}}_1^\circ)$ | bits | 111010101010111 |
| 4 | Encode | $\overline{\mathbf{z}}_1^\circ$ | using | $\mathbf{P}_{\overline{\mathbf{z}}_1\|\overline{\mathbf{z}}_2^\circ}$ | adds | $-\log_2 \overline{p}_{\boldsymbol{\theta}}(\overline{\mathbf{z}}_1^\circ\|\overline{\mathbf{z}}_2^\circ)$ | bits | 1110101010101111 |
| ⋮ | | ⋮ | | | | ⋮ | | ⋮ |
| $2L-1$ | Decode | $\overline{\mathbf{z}}_L^\circ$ | using | $\mathbf{Q}_{\overline{\mathbf{z}}_L\|\overline{\mathbf{z}}_{L-1}^\circ}$ | takes | $-\log_2 \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}_L^\circ\|\overline{\mathbf{z}}_{L-1}^\circ)$ | bits | 11101010101 |
| $2L$ | Encode | $\overline{\mathbf{z}}_{L-1}^\circ$ | using | $\mathbf{P}_{\overline{\mathbf{z}}_{L-1}\|\overline{\mathbf{z}}_L^\circ}$ | adds | $-\log_2 \overline{p}_{\boldsymbol{\theta}}(\overline{\mathbf{z}}_{L-1}^\circ\|\overline{\mathbf{z}}_L^\circ)$ | bits | 111010101011001 |
| $2L+1$ | Encode | $\overline{\mathbf{z}}_L^\circ$ | using | $\mathbf{P}_{\overline{\mathbf{z}}_L}$ | adds | $-\log_2 \overline{p}(\overline{\mathbf{z}}_L^\circ)$ | bits | 1110101010011001010 |

The codelength resulting from encoding $\{\mathbf{x}^\circ, \overline{\mathbf{z}}_1^\circ, \ldots, \overline{\mathbf{z}}_L^\circ\}$ and the "other data" is *the same* as the codelength resulting from the extended bits-back scheme:

$$\begin{aligned} l_{\text{total}}(\text{"other data"}, \mathbf{x}^\circ, \overline{\mathbf{z}}_1^\circ, \ldots, \overline{\mathbf{z}}_L^\circ) = N_{\text{init}} &+ \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}_1|\mathbf{x}^\circ) \\ &+ \log_2 \sum_{l=2}^{L} \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}_l^\circ|\overline{\mathbf{z}}_{l-1}^\circ) \\ &- \log_2 p_{\boldsymbol{\theta}}(\mathbf{x}^\circ|\overline{\mathbf{z}}_1^\circ) \\ &- \sum_{l=l}^{L-1} \log_2 \overline{p}_{\boldsymbol{\theta}}(\overline{\mathbf{z}}_l^\circ|\overline{\mathbf{z}}_{l+1}^\circ) \\ &- \log_2 \overline{p}(\overline{\mathbf{z}}_L^\circ) \end{aligned} \tag{5.25}$$

where $\overline{\mathbf{z}}_0^\circ \equiv \mathbf{x}^\circ$. Therefore, the *net* codelength $l_{\text{bitsback}}(\mathbf{x}^\circ)$ is also the same (see Equation 5.22). However, the recursive bits-back scheme has a lower initial cost than the extended bits-back scheme. We derive this property in Section 5.2.4.

In order to exactly retrieve $\{\mathbf{x}^\circ, \overline{\mathbf{z}}_1^\circ, \ldots, \overline{\mathbf{z}}_L^\circ\}$ and the "other data", the receiver reverses the order and replaces decode with encode operations and vice versa. The corresponding scheme can be found in Table 5.2.

TABLE 5.4: Steps of the recursive bits-back scheme for nested latent variable models at the receiver's end.

| # | ANS Operation | | | | Δ Bits | | | Bitstream (illustr.) |
|---|---|---|---|---|---|---|---|---|
| - | - | | | | - | | | 1110101010011001010 |
| $2L+1$ | Decode | $\overline{\mathbf{z}}_L^\circ$ | using | $\mathbf{P}_{\overline{\mathbf{z}}_L}$ | takes | $-\log_2 \overline{p}(\overline{\mathbf{z}}_L^\circ)$ | bits | 111010101011001 |
| $2L$ | Decode | $\overline{\mathbf{z}}_{L-1}^\circ$ | using | $\mathbf{P}_{\overline{\mathbf{z}}_{L-1}\|\overline{\mathbf{z}}_L^\circ}$ | takes | $-\log_2 \overline{p}_{\boldsymbol{\theta}}(\overline{\mathbf{z}}_{L-1}^\circ\|\overline{\mathbf{z}}_L^\circ)$ | bits | 11101010101 |
| $2L-1$ | Encode | $\overline{\mathbf{z}}_L^\circ$ | using | $\mathbf{Q}_{\overline{\mathbf{z}}_L\|\overline{\mathbf{z}}_{L-1}^\circ}$ | adds | $-\log_2 \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}_L^\circ\|\overline{\mathbf{z}}_{L-1}^\circ)$ | bits | 1110101010101111 |
| ⋮ | | ⋮ | | | | ⋮ | | ⋮ |
| 4 | Decode | $\overline{\mathbf{z}}_1^\circ$ | using | $\mathbf{P}_{\overline{\mathbf{z}}_1\|\overline{\mathbf{z}}_2^\circ}$ | takes | $-\log_2 \overline{p}_{\boldsymbol{\theta}}(\overline{\mathbf{z}}_1^\circ\|\overline{\mathbf{z}}_2^\circ)$ | bits | 111010101010111 |
| 3 | Encode | $\overline{\mathbf{z}}_2^\circ$ | using | $\mathbf{Q}_{\overline{\mathbf{z}}_2\|\overline{\mathbf{z}}_1^\circ}$ | adds | $-\log_2 \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}_2^\circ\|\overline{\mathbf{z}}_1^\circ)$ | bits | 11101010101011111 |
| 2 | Decode | $\mathbf{x}^\circ$ | using | $\mathbf{P}_{\mathbf{x}\|\overline{\mathbf{z}}_1^\circ}$ | takes | $-\log_2 p_{\boldsymbol{\theta}}(\mathbf{x}^\circ\|\overline{\mathbf{z}}_1^\circ)$ | bits | 11101010 |
| 1 | Encode | $\overline{\mathbf{z}}_1^\circ$ | using | $\mathbf{Q}_{\overline{\mathbf{z}}_1\|\mathbf{x}^\circ}$ | adds | $-\log_2 \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}_1^\circ\|\mathbf{x}^\circ)$ | bits | 111010101000 |

The procedure for the sender and receiver for 3 latent variables is displayed in Figure 5.1(b).

### 5.2.4 Advantage of using the Recursive Bits-Back Scheme

We show that the recursive bits-back scheme has a lower initial cost than the extended bits-back scheme, by deriving a bound for initial cost of the recursive bits-back scheme. Let $N^l_{\text{cost}}$ denote the initial cost to decode $\overline{\mathbf{z}}^\circ_l$ using $\mathbf{Q}_{\overline{\mathbf{z}}_l|\overline{\mathbf{z}}^\circ_{l-1}}$. For brevity, we use the convention $\overline{\mathbf{z}}^\circ_0 \equiv \mathbf{x}^\circ$ and $\overline{p}_{\boldsymbol{\theta}}(\mathbf{x}^\circ|\overline{\mathbf{z}}^\circ_1) \equiv p_{\boldsymbol{\theta}}(\mathbf{x}^\circ|\overline{\mathbf{z}}^\circ_1)$.

For the first latent layer, we need the following number of bits:

$$N^1_{\text{cost}} = -\log_2 \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}^\circ_1|\mathbf{x}^\circ) \tag{5.26}$$

Then, with recursive bits-back coding, for every remaining latent layer $l \in \{2, .., L\}$, we always *first* encode $-\log_2 \overline{p}_{\boldsymbol{\theta}}(\overline{\mathbf{z}}^\circ_{l-2}|\overline{\mathbf{z}}^\circ_{l-1})$ bits and afterwards decode $-\log_2 \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}^\circ_l|\overline{\mathbf{z}}^\circ_{l-1})$ bits. We only need initial bits for latent layer $\overline{\mathbf{z}}_l$ if $-\log_2 \overline{p}_{\boldsymbol{\theta}}(\overline{\mathbf{z}}^\circ_{l-2}|\overline{\mathbf{z}}^\circ_{l-1})$ is less than $-\log_2 \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}^\circ_l|\overline{\mathbf{z}}^\circ_{l-1})$, or equivalently:

$$-\log_2 \overline{p}_{\boldsymbol{\theta}}(\overline{\mathbf{z}}^\circ_{l-2}|\overline{\mathbf{z}}^\circ_{l-1}) + \log_2 \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}^\circ_l|\overline{\mathbf{z}}^\circ_{l-1}) \leq 0 \implies N^l_{\text{cost}} = \log_2 \frac{\overline{p}_{\boldsymbol{\theta}}(\overline{\mathbf{z}}^\circ_{l-2}|\overline{\mathbf{z}}^\circ_{l-1})}{\overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}^\circ_l|\overline{\mathbf{z}}^\circ_{l-1})} \tag{5.27}$$

Therefore, for every layer $l \in \{2, .., L\}$ it holds that the initial cost is equal to:

$$N^l_{\text{cost}} = \max\left(0, \log_2 \frac{\overline{p}_{\boldsymbol{\theta}}(\overline{\mathbf{z}}^\circ_{l-2}|\overline{\mathbf{z}}^\circ_{l-1})}{\overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}^\circ_l|\overline{\mathbf{z}}^\circ_{l-1})}\right) \tag{5.28}$$

The total initial cost is equal to the sum of the initial cost of every latent layer $\overline{\mathbf{z}}^\circ_l$:

$$\begin{aligned} N^{\text{recursive}}_{\text{cost}} &:= \sum_{i=1}^{L} N^l_{\text{cost}} \\ &= -\log_2 \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}^\circ_1|\mathbf{x}) + \max\left(0, \log_2 \frac{\overline{p}_{\boldsymbol{\theta}}(\overline{\mathbf{z}}^\circ_{l-2}|\overline{\mathbf{z}}^\circ_{l-1})}{\overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}^\circ_l|\overline{\mathbf{z}}^\circ_{l-1})}\right) \\ &= \sum_{l=0}^{L-1} \max\left(0, \log \frac{\overline{p}_{\boldsymbol{\theta}}(\overline{\mathbf{z}}^\circ_{l-1}|\overline{\mathbf{z}}^\circ_l)}{\overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}^\circ_{l+1}|\overline{\mathbf{z}}^\circ_l)}\right) \end{aligned} \tag{5.29}$$

where we used the convention $\overline{p}_{\boldsymbol{\theta}}(\overline{\mathbf{z}}^\circ_{-1}|\overline{\mathbf{z}}^\circ_0) = 1$. Thus, we can guarantee that the recursive bits-back scheme has a lower initial cost than the extended bits-back scheme:

$$\begin{aligned} \sum_{l=0}^{L-1} \max\left(0, \log \frac{\overline{p}_{\boldsymbol{\theta}}(\overline{\mathbf{z}}^\circ_{l-1}|\overline{\mathbf{z}}^\circ_l)}{\overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}^\circ_{l+1}|\overline{\mathbf{z}}^\circ_l)}\right) &\leq \sum_{l=0}^{L-1} -\log \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}^\circ_{l+1}|\overline{\mathbf{z}}^\circ_l) \\ &= N^{\text{bitsback}}_{\text{cost}} \end{aligned} \tag{5.30}$$

### 5.2.5 Analysis of the Expected Codelength

We now analyze the codelength of both the extended and recursive bits-back scheme. The goal was to construct representations $C[\mathbf{x}^\circ]$ that have codelength close to the entropy of the underlying distribution on average:

$$L = \mathbb{E}_{p^*}\left[l(\mathbf{x})\right] \to H(p^*) \tag{5.31}$$

In order to fit the equations on the page, we rewrite the codelength of both the recursive and extended bits-back scheme as follows:

$$l_{\text{bitsback}}(\mathbf{x}^\circ) = \log_2 \left( \frac{\prod_{l=1}^{L} \overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}_l^\circ | \overline{\mathbf{z}}_{l-1}^\circ)}{\overline{p}(\overline{\mathbf{z}}_L^\circ) \cdot p_{\boldsymbol{\theta}}(\mathbf{x}^\circ | \overline{\mathbf{z}}_1^\circ) \cdot \prod_{l=l}^{L-1} \overline{p}_{\boldsymbol{\theta}}(\overline{\mathbf{z}}_l^\circ | \overline{\mathbf{z}}_{l+1}^\circ)} \right) \tag{5.32}$$

where $\overline{\mathbf{z}}^\circ \equiv \mathbf{x}$. Consequently, from Equation 5.18 we see that the expected codelength of both the recursive and extended bits-back scheme using nested latent variable models is:

$$\begin{aligned}
L_{\text{bitsback}}^{\text{nested}} &= \mathbb{E}_{p^*}\left[\mathbb{E}_{\overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}_{1:L}|\mathbf{x})}\left[\log_2\left(\frac{\prod_{l=1}^{L}\overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}_l|\overline{\mathbf{z}}_{l-1})}{\overline{p}(\overline{\mathbf{z}}_L)\cdot p_{\boldsymbol{\theta}}(\mathbf{x}|\overline{\mathbf{z}}_1)\cdot\prod_{l=l}^{L-1}\overline{p}_{\boldsymbol{\theta}}(\overline{\mathbf{z}}_l|\overline{\mathbf{z}}_{l+1})}\right)\right]\right] \\
&\approx \mathbb{E}_{p^*}\left[\mathbb{E}_{\overline{q}_{\boldsymbol{\phi}}(\overline{\mathbf{z}}_{1:L}|\mathbf{x})}\left[\log_2\left(\frac{\prod_{l=1}^{L} q_{\boldsymbol{\phi}}(\overline{\mathbf{z}}_l|\overline{\mathbf{z}}_{l-1})}{p(\overline{\mathbf{z}}_L)\cdot p_{\boldsymbol{\theta}}(\mathbf{x}|\overline{\mathbf{z}}_1)\cdot\prod_{l=l}^{L-1} p_{\boldsymbol{\theta}}(\overline{\mathbf{z}}_l|\overline{\mathbf{z}}_{l+1})}\cdot\frac{\delta_{\mathbf{z}_1}\delta_{\mathbf{z}_2}\cdots\delta_{\mathbf{z}_L}}{\delta_{\mathbf{z}_1}\delta_{\mathbf{z}_2}\cdots\delta_{\mathbf{z}_L}}\right)\right]\right] \\
&\approx \mathbb{E}_{p^*}\left[\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}_{1:L}|\mathbf{x})}\left[\log_2\left(\frac{\prod_{l=1}^{L} q_{\boldsymbol{\phi}}(\mathbf{z}_l|\mathbf{z}_{l-1})}{p(\mathbf{z}_L)\cdot\prod_{l=0}^{L-1} p_{\boldsymbol{\theta}}(\mathbf{z}_l|\mathbf{z}_{l+1})}\right)\right]\right] \\
&= \mathbb{E}_{p^*}\left[\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}_{1:L}|\mathbf{x})}\left[\sum_{l=1}^{L}\log_2 q_{\boldsymbol{\phi}}(\mathbf{z}_l|\mathbf{z}_{l-1}) - \sum_{l=0}^{L-1}\log_2 p_{\boldsymbol{\theta}}(\mathbf{z}_l|\mathbf{z}_{l+1}) - \log_2 p(\mathbf{z}_L)\right]\right] \\
&= \mathbb{E}_{p^*}\left[-\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x})\right]
\end{aligned} \tag{5.33}$$

where $\overline{\mathbf{z}}^\circ \equiv \mathbf{z}^\circ \equiv \mathbf{x}^\circ$. As we explained at the end of Section 5.1.2, optimizing the ELBO with MLE enjoys the same properties as the conventional latent variable model case. Therefore, the analysis of the expected codelength is analogous to the analysis for conventional latent variable model case, described in Section 4.2.5. The expected codelength of both the recursive and extended bits-back scheme is:

$$L_{\text{bitsback}}^{\text{nested}} \approx \mathbb{E}_{p^*}\left[-\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x})\right] \to H(p^*) \tag{5.34}$$

Again, from Equations 4.35 and 5.16 we see that we can construct a Monte-Carlo estimate of the expected codelength given a dataset $\mathcal{D}$:

$$-\frac{1}{N_{\mathcal{D}}}\sum_{i=1}^{N_{\mathcal{D}}}\overline{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i) \simeq \mathbb{E}_{p^*}\left[-\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x})\right] \approx L_{\text{bitsback}}^{\text{nested}} \tag{5.35}$$

# 6 Compression Performance on Image Data

We put content into the described methods by compressing actual images. We claimed in Section 5 that adding more latent layers to a nested latent variable model tends to make them better in modelling complex high-dimensional data. We also claimed in Section 5.2.4 that the recursive bits-back scheme has a lower initial cost compared to the extended bits-back scheme.

We conducted an extensive empirical analysis of these two claims using three toy image databases consisting of a large number of small images. Every database is split up in a *train-set* and *test-set*. The train-set is used for optimization and the test-set is used for evaluation of the model and the compression schemes.

We optimized four different nested latent variable model designs: with 1, 2, 4 and 8 latent layers $L$. The result can be found in Section 6.2. After optimization, the models are used for lossless compression with the recursive bits-back scheme versus the extended bits-back scheme on the test-sets. The analysis of the resulting initial bitstreams and compression rates can be found in Section 6.3.

Furthermore, we compared our method with established lossless compression schemes by compressing 100 images of normal size sampled from an additional database. As a comparison, we also compressed the three toy test-sets with these methods. The recursive bits-back scheme with nested latent variable models outperforms all other compression schemes on all datasets. The methodology and results can be found in Section 6.4.

## 6.1 Used Image Databases

### 6.1.1 Toy Images

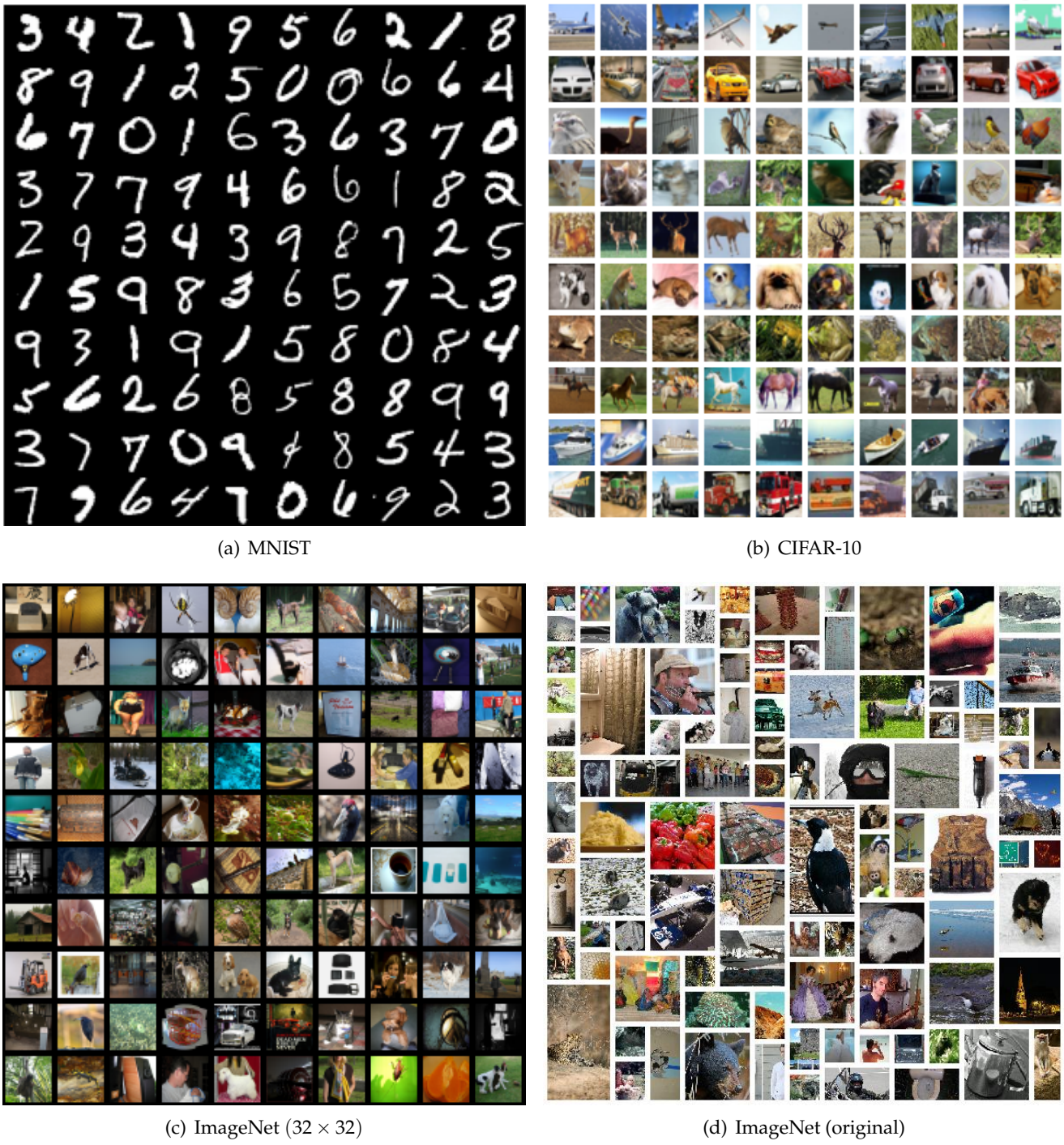
In order to optimize the models and analyze the compression performance of the compression schemes, we need large databases with a broad variety of images. We consider three different image databases:

1. MNIST (LeCun, 1998)
2. CIFAR-10 (Krizhevsky, Nair, and Hinton, 2014)
3. ImageNet ($32 \times 32$) (Deng et al., 2009)

The MNIST database consists of 70000 $28 \times 28$ images of handwritten digits. The CIFAR-10 database consists of 60000 $32 \times 32$ images divided into 10 different classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. The ImageNet ($32 \times 32$) database consists of 1.2 million $32 \times 32$ images that are downsized (and cropped) from the original ImageNet database (Krizhevsky, Sutskever, and Hinton, 2012). Sample images from the datasets are shown in Figures 6.1(a) (MNIST), 6.1(b) (CIFAR-10) and 6.1(c) (ImageNet ($32 \times 32$)). We split up every dataset into a *test set* $\mathcal{T}$ ($N_{\mathcal{T}} = 10000$ images) and *train set* $\mathcal{D}$ (the remaining $N_{\mathcal{D}}$ images). The train set is used to optimize the models. The test sets are used to evaluate the model, investigate the initial bitstreams and compression performance.

### 6.1.2 Normal Images

In order to make a fair comparison of the recursive bits-back scheme with established techniques, we need larger images. The original ImageNet database (Krizhevsky, Sutskever, and Hinton, 2012) consists of 1.2 million relatively large images with differing sizes of an extremely broad variety of things. Because of the size, we only consider a subset of 150000 images from the original ImageNet database. These 150000 images are split up in a test-set $\mathcal{T}$ (100 images) and train-set $\mathcal{D}$ (the remaining images). The train-set is used to optimize the model. The test-set is used to compare the compression performance of existing techniques with the recursive bits-back scheme. Sample images from the original ImageNet database are shown in Figure 6.1(d).



(a) MNIST

(b) CIFAR-10

(c) ImageNet ($32 \times 32$)

(d) ImageNet (original)

FIGURE 6.1: 100 sample images from MNIST (top-left), CIFAR-10 (top-right), ImageNet ($32 \times 32$) (bottom-left) and original ImageNet (bottom-right).

## 6.2 Model Performance

We optimized every model using every toy train-set $\mathcal{D}$. After optimization, we estimated the expected codelength with every corresponding toy test-set $\mathcal{T}$ using the Monte-Carlo estimates described in Equations 4.37 and 5.35.

We scale these estimates by the number of color channels $P$ per image to get an

estimate of the expected codelength in *bits per color channel* (bits/channel). Since every database has images of a different size, this gives a fair comparison between the test-sets. The models have an estimated expected codelength of approximately 8.00 bits/channel before optimization. The results for every model and for every test-set after optimization can be found in Table 6.1. From the results we can see that, as we add more latent layers, the estimated expected codelength of the nested latent variable model decreases. Therefore, deeper models are are better in modelling the image data.

TABLE 6.1: The estimated expected codelengths of every toy test-set using every model after optimization. In order to make a fair comparison, we made sure that the parameter counts of the corresponding neural networks regress if we use more latent layers $L$.

| | **MNIST** | | **CIFAR-10** | | **ImageNet** ($32 \times 32$) | |
|---|---|---|---|---|---|---|
| Depth ($L$) | # Parameters | bits/channel | # Parameters | bits/channel | # Parameters | bits/channel |
| 1 | 2.84M | 1.35 | 45.3M | 4.57 | 45.3M | 4.94 |
| 2 | 2.75M | 1.28 | 45.0M | 3.83 | 45.0M | 4.53 |
| 4 | 2.67M | 1.27 | 44.9M | 3.81 | 44.9M | 4.48 |
| 8 | 2.60M | 1.27 | 44.7M | 3.78 | - | - |

## 6.3 Compression Performance

After optimization of the models on the three toy train-sets $\mathcal{D}$, we discretized the latent space with $D = 2^{10}$ discretization bins per latent variable. Afterwards, we applied the recursive bits-back scheme and the extended bits-back scheme with these models on the corresponding toy test-sets $\mathcal{T}$.

We first show that the recursive bits-back scheme indeed reduces the initial cost and therefore outperforms the extended bits-back scheme on nested latent variable models in terms of actual compression rates. In Section 4.2.4 we argued that the initial cost compared to the total length of the bitstream gets insignificant if we repeat the schemes on an entire sequence of images. In other words, the initial cost *amortizes* over a sequence of images.

We now describe the experiments we did for one test-set $\mathcal{T}$, which is the same for all test-sets. In order to analyse the amount of amortization of the initial cost over a sequence of images, we applied the same methodology for every model (depth $L = \{2, 4, 8\}$) for every compression scheme (recursive and extended). We split the toy test-set in subsets of 100 images $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_{100}\}$. Since the test-set $\mathcal{T}$ contains $N_{\mathcal{T}} = 10000$ images, every subset $\mathcal{T}_k$ contains 100 images. Afterwards, we compressed the 100 images in subset $\mathcal{T}_k$ in sequence. After compressing each image, we calculated the cumulative moving average (CMA) of the codelength by dividing the length of the bitstream by the number of color channels $P$ and the number of images $i$ compressed so far:

$$\text{Rate}(\mathcal{T}_k, i) = \frac{\text{\# bits after compressing } i \text{ images from } \mathcal{T}_k}{i \cdot P}$$

This quantity depicts the *compression rate* after each image, expressed in bits/channel. Note that this quantity *includes* the initial cost. We also calculated the CMA of the codelength but then *without* the initial cost:

$$\text{Net Rate}(\mathcal{T}_k, i) = \frac{\text{\# bits after compressing } i \text{ images from } \mathcal{T}_k - \text{initial cost of } \mathcal{T}_k}{i \cdot P}$$

This quantity depicts the *net compression rate*. This is interpreted as a lower bound of the average compression rate. Both compression rate and the *net* compression rate are averaged over the subsets $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_{100}\}$:

$$\begin{aligned} \overline{\text{Rate}(i)} &= \frac{1}{100} \sum_{k=1}^{100} \text{Rate}(\mathcal{T}_k, i) \\ \overline{\text{Net Rate}(i)} &= \frac{1}{100} \sum_{k=1}^{100} \text{Net Rate}(\mathcal{T}_k, i) \end{aligned} \tag{6.1}$$

The average compression rate (with the corresponding average net compression rate) over 100 experiments for every model and dataset is shown in Figure 6.2. The results of the recursive bits-back scheme is depicted in blue and the extended bits-back scheme in orange. These graphs show two properties: the difference in initial cost and the fact that the average compression rate of both schemes amortize towards the average net compression rate.

The initial cost is clearly amortized as the amount of images compressed grows. The relatively high initial cost of both compression schemes comes from the fact that the

initial cost increases with the number of discretization bins $D$. When using the extended bits-back scheme, remember that this initial cost also grows linearly with the number of latent layers $L$. The recursive bits-back scheme does not have this problem. This results in a performance gap in terms of the average compression rate that grows with the amount of latent layers $L$. The efficiency of the recursive compared to the extended bits-back scheme results in much faster amortization, which makes the recursive bits-back scheme a more practical algorithm.

Furthermore, by compressing 100 images in sequence with the recursive and the extended bits-back scheme we save a certain amount of space on a system compared to the baseline scheme. We calculated the percentage that we would save after compressing 100 images from a subset $\mathcal{T}_k$ by the following formula:

$$\text{Savings} = 100.00 - \frac{\overline{\text{Rate}(100)}}{8.00 \text{ (= absolute rate)}} \times 100\% \tag{6.2}$$

The results for every model and for every test-set can be found in Table 6.2.

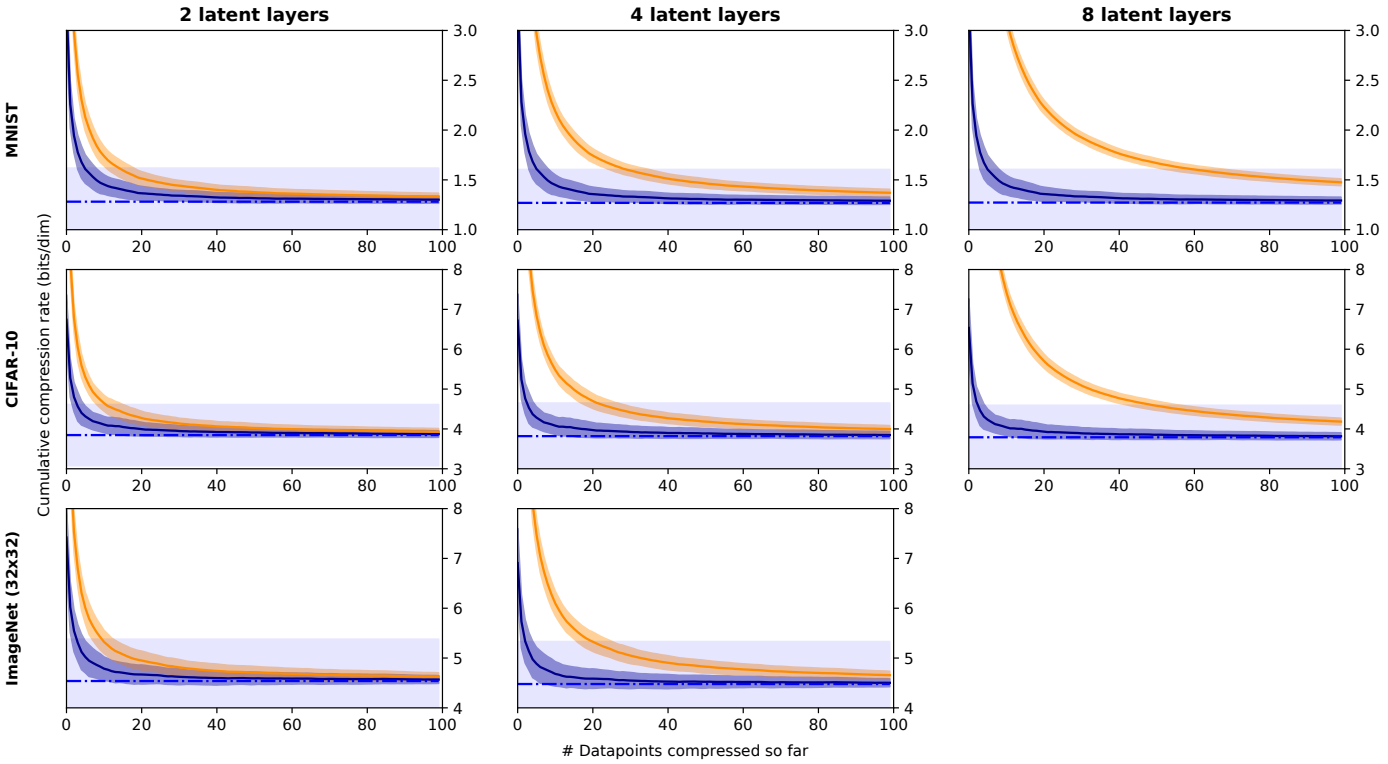



FIGURE 6.2: Cumulative moving average of the compression rate over time for the recursive bits-back scheme (blue) and the extended bits-back scheme (orange) for sequences of 100 images, averaged over 100 subsets of the toy test-sets. The blue dotted line and region represent the average and standard deviation of the net compression rate across the entire toy test-set.

TABLE 6.2: Compression rates and savings of every model using the recursive versus the extended bits-back scheme on every toy test-set.

| | | **MNIST** | | **CIFAR-10** | | **ImageNet** ($32 \times 32$) | |
|---|---|---|---|---|---|---|---|
| Depth ($L$) | Compression Scheme | Rate | Savings | Rate | Savings | Rate | Savings |
| 2 | Extended | $1.33 \pm 0.03$ | 83.37% | $3.93 \pm 0.08$ | 50.87% | $4.63 \pm 0.08$ | 42.14% |
| | **Recursive** | $1.30 \pm 0.03$ | 83.75% | $3.87 \pm 0.08$ | 51.62% | $4.57 \pm 0.08$ | 42.84% |
| 4 | Extended | $1.37 \pm 0.04$ | 82.87% | $4.00 \pm 0.09$ | 50.00% | $4.66 \pm 0.08$ | 41.75% |
| | **Recursive** | $1.29 \pm 0.04$ | 83.87% | $3.85 \pm 0.09$ | 51.87% | $4.50 \pm 0.08$ | 43.75% |
| 8 | Extended | $1.48 \pm 0.03$ | 81.50% | $4.19 \pm 0.09$ | 47.62% | - | - |
| | **Recursive** | $1.29 \pm 0.03$ | 83.87% | $3.82 \pm 0.09$ | 52.25% | - | - |

## 6.4 Comparison with Other Methods

Finally, we compared both the recursive and the extended bits-back scheme against a number of benchmark lossless compression schemes on normal images. First, we optimized the same model as used for Imagenet ($32 \times 32$) with $L = 4$ latent layers on random $32 \times 32$ patches of the corresponding train-set $\mathcal{D}$ of normal ImageNet. Afterwards we compressed the 100 images of the test-set $\mathcal{T} = \{\mathbf{x}^1, \ldots, \mathbf{x}^{100}\}$ with every benchmark scheme with the following strategy. We cropped every image $\mathbf{x}^i$ to multiples of 32 pixels on each side, resulting in a cropped image $\mathbf{x}^i_*$ with $P^*$ color channels and $K$ pixel blocks. Then we repeated the recursive and extended bits-back scheme on one $32 \times 32$ pixel block of $\mathbf{x}^i_*$ at the time, essentially compressing a sequence of $K$ smaller "images". The resulting bitstream is divided by $P_*$ to obtain the compression rate of the cropped image $\mathbf{x}^i_*$, expressed in bits/channel. We averaged the compression rates of every image using every benchmark scheme and calculated the percentage that we would save compared to using the baseline scheme. The results can be found in the second column of Table 6.1.

As a comparison, we also compressed the three toy test-sets using the benchmark compression schemes. The results can be found in the third, fourth and fifth column of Table 6.1. For the extended and recursive bits-back scheme, we copied the best results on these test-sets from Table 6.2. That is, the results on MNIST with $L = 8$ latent layers, the results for CIFAR-10 with $L = 8$ latent layers and the results for ImageNet $(32 \times 32)$ with $L = 4$ latent layers.

TABLE 6.3: The percentage of savings after compressing the test-sets using different lossless compression schemes. We averaged the results over 100 experiments per test-set. For the original ImageNet test-set (second column) we compressed 1 large image per experiment by dividing it into $32 \times 32$ pixel blocks and compressing the blocks in sequence. For the three toy test-sets (third, fourth and fifth column) we compressed 100 small images in sequence per experiment.

| | **ImageNet** | **MNIST** | **CIFAR-10** | **ImageNet** $(32 \times 32)$ |
|---|---|---|---|---|
| # Images per experiment | 1 | 100 | 100 | 100 |
| Fixed-Length Coding | 0.00 % | 0.00 % | 0.00 % | 0.00 % |
| GNU Gzip | 25.50 % | 79.37 % | 7.87 % | 8.62 % |
| bzip2 | 36.62 % | 80.12 % | 12.75 % | 12.50 % |
| LZMA | 36.37 % | 81.37 % | 24.14 % | 23.12 % |
| PNG | 41.14 % | 65.00 % | 26.62 % | 20.12 % |
| WebP | 54.25 % | 73.75 % | 42.37 % | 33.87 % |
| Extended Bits-Back Scheme | 54.75 % | 81.50 % | 47.62 % | 41.75 % |
| **Recursive Bits-Back Scheme** | **56.12%** | **83.87%** | **52.25%** | **43.75%** |

# 7 Conclusion

Data compression has become an indispensable part of our lives. Due to the ever-increasing amount of data in digital form, we wish to transmit and store data in compact form. We focused on compression without loss of information, known as lossless compression, of high-dimensional data. Probabilistic models based on deep learning are highly effective in modelling complex dependencies in high-dimensional data. In this thesis, we aimed to design a fast and effective compression algorithm for high-dimensional data, that uses a probabilistic model based on deep learning. Hence, we formulated three sets of research questions in Section 1.2. The answers are given in this thesis, and are now be summarized.

*(Question Set 1) What probabilistic model is capable of modelling complex high-dimensional data? What is a scalable and effective lossless compression algorithm for high-dimensional input data that makes efficient use of this probabilistic model?* In Section 4 we explained a probabilistic model, called a latent variable model, whose design incorporates neural networks. This model can be optimized using large datasets to model complex high-dimensional data. We also explained the bits-back scheme, a fast and effective lossless compression algorithm that makes efficient use of latent variable models to achieve compression of high-dimensional input data. We derived that compression can be near-optimal if the latent variable model precisely resembles the underlying distribution of the input data.

*(Question Set 2) How do we improve the latent variable model? How do we extend the bits-back scheme such that it makes efficient use of the improved latent variable model?* In Section 5 we introduced the nested latent variable model, a hierarchical extension of a latent variable model whose design incorporates neural networks in an analogous matter. Nested latent variable models can also be optimized using large datasets, but they tend to be better in modelling complex high-dimensional data compared to conventional latent variable models. We also introduced the extended bits-back scheme, an extension of the bits-back scheme that is able to use nested latent variable models for compression. Finally, we introduced the recursive bits-back scheme, which we show improves upon the extend bits-back scheme by exploiting the design structure of nested latent variable models.

*(Question Set 3) How well are nested latent variable models able to model image data? How well does the recursive bits-back scheme perform on image data? How does it perform compared to other methods?* In Section 6 we conducted modelling and compression experiments on image databases to support our claims. We have shown that increasing the depth of the nested latent variable model improves the modelling capacity. Furthermore, lossless compression of image data with the recursive bits-back scheme improves upon the extended bits-back scheme. The performance discrepancy becomes increasingly apparent as we use deeper nested latent variable models. Finally, we have shown that the recursive bits-back scheme results in lossless compression that is empirically superior to established methods.

# Bibliography


Bowman, Samuel R et al. (2015). "Generating Sentences from a Continuous Space". In: *arXiv preprint arXiv:1511.06349*.

Chen, Xi et al. (2016). "Variational lossy autoencoder". In: *arXiv preprint arXiv:1611.02731*.

Cover, Thomas M and Joy A Thomas (2012). *Elements of information theory*. John Wiley & Sons.

Deng, Jia et al. (2009). "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee, pp. 248–255.

Duda, Jarek (2009). *Asymmetric numeral systems*. arXiv: 0902.0271 [cs.IT].

Giesen, Fabian (2014). "Interleaved entropy coders". In: *arXiv preprint arXiv:1402.3392*.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). "Deep Learning". Book in preparation for MIT Press. URL: http://www.deeplearningbook.org.

Gregor, Karol et al. (2015). "DRAW: A recurrent neural network for image generation". In: *arXiv preprint arXiv:1502.04623*.

He, Kaiming et al. (2015). "Deep Residual Learning for Image Recognition". In: *arXiv preprint arXiv:1512.03385*.

Hinton, Geoffrey E and Drew Van Camp (1993). "Keeping the neural networks simple by minimizing the description length of the weights". In: *Proceedings of the sixth annual conference on Computational learning theory*. ACM, pp. 5–13.

Ho, Jonathan, Evan Lohn, and Pieter Abbeel (2019). "Compression with Flows via Local Bits-Back Coding". In: *arXiv preprint arXiv:1905.08500*.

Hochreiter, Sepp et al. (2001). *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies*.

Huffman, David A (1952). "A method for the construction of minimum-redundancy codes". In: *Proceedings of the IRE* 40.9, pp. 1098–1101.

Hutson, Matthew (2018). *Has artificial intelligence become alchemy?*

Jordan, M.I. et al. (1999). "An introduction to variational methods for graphical models". In: *Machine learning* 37.2, pp. 183–233.

Judd, J Stephen (1990). *Neural network design and the complexity of learning*. MIT press.

Kaae Sønderby, Casper et al. (2016). "How to Train Deep Variational Autoencoders and Probabilistic Ladder Networks". In: *arXiv preprint arXiv:1602.02282*.

Kingma, Diederik P and Max Welling (2013). "Auto-Encoding Variational Bayes". In: *Proceedings of the 2nd International Conference on Learning Representations*.

Kingma, Diederik P. and Max Welling (2019). *An Introduction to Variational Autoencoders*. arXiv: 1906.02691 [cs.LG].

Kingma, Diederik P et al. (2016). "Improved Variational Inference with Inverse Autoregressive Flow". In: *Advances in Neural Information Processing Systems*, pp. 4743–4751.

Kingma, Friso H, Pieter Abbeel, and Jonathan Ho (2019). "Bit-Swap: Recursive Bits-Back Coding for Lossless Compression with Hierarchical Latent Variables". In: *International Conference on Machine Learning*.

Krizhevsky, Alex, Vinod Nair, and Geoffrey Hinton (2014). "The CIFAR-10 dataset". In: *online: http://www. cs. toronto. edu/kriz/cifar. html* 55.

Krizhevsky, Alex, Ilya Sutskever, and Geoff Hinton (2012). "Imagenet classification with deep convolutional neural networks". In: *Advances in Neural Information Processing Systems 25*, pp. 1106–1114.

LeCun, Yann (1998). "The MNIST database of handwritten digits". In: *http://yann. lecun. com/exdb/mnist/*.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep learning". In: *Nature* 521.7553, pp. 436–444.

LeCun, Yann et al. (1989). "Backpropagation applied to handwritten zip code recognition". In: *Neural computation* 1.4, pp. 541–551.

Maaløe, Lars et al. (2019). "Biva: A very deep hierarchy of latent variables for generative modeling". In: *arXiv preprint arXiv:1902.02102*.

Martens, James (2010). "Deep learning via hessian-free optimization." In: *ICML*. Vol. 27, pp. 735–742.

Polyak, Boris T and Anatoli B Juditsky (1992). "Acceleration of stochastic approximation by averaging". In: *SIAM Journal on Control and Optimization* 30.4, pp. 838–855.

Prechelt, Lutz (1998). "Early stopping-but when?" In: *Neural Networks: Tricks of the trade*. Springer, pp. 55–69.

Rezende, Danilo J, Shakir Mohamed, and Daan Wierstra (2014). "Stochastic backpropagation and approximate inference in deep generative models". In: *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 1278–1286.

Robbins, Herbert and Sutton Monro (1951). "A stochastic approximation method". In: *The annals of mathematical statistics*, pp. 400–407.

Salimans, Tim and Diederik P Kingma (2016). "Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks". In: *arXiv preprint arXiv:1602.07868*.

Sayood, Khalid (2017). *Introduction to data compression*. Morgan Kaufmann.

Serban, Iulian Vlad et al. (2016). "A Hierarchical Latent Variable Encoder-Decoder Model for Generating Dialogues". In: *arXiv preprint arXiv:1605.06069*.

Shannon, Claude Elwood (1948). "A mathematical theory of communication". In: *Bell system technical journal* 27.3, pp. 379–423.

Sønderby, Casper Kaae et al. (2016). "Ladder variational autoencoders". In: *Advances in neural information processing systems*, pp. 3738–3746.

Srivastava, Nitish et al. (2014). "Dropout: A simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1, pp. 1929–1958.

Sutskever, Ilya et al. (2013). "On the importance of initialization and momentum in deep learning". In: *International conference on machine learning*, pp. 1139–1147.

Townsend, James, Thomas Bird, and David Barber (2019). "Practical lossless compression with latent variables using bits back coding". In:

Wallace, Chris S (1990). "Classification by minimum-message-length inference". In: *International Conference on Computing and Information*. Springer, pp. 72–81.

# Appendices

# A Asymmetric Numeral Systems

Asymmetric Numeral Systems is a coder that is scalable over dimensionality of the input $P$ and is compatible with the bits-back scheme, without any tricks or modifications. To understand why the bits-back scheme is an important factor regarding the choice of the coder, it is critical to note that the bits-back scheme requires a certain order in which symbols are encoded and decoded.

## A.1 LIFO & rANS

Suppose the sender wishes to communicate multiple datapoints $\{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$ to the receiver. The compression scheme, assuming we have access to an initial bitstream of length $N_{\text{init}}$, for the sender in pseudo-code is:

$$\begin{aligned} &\text{for } i = 1 : N \\ &\quad \text{Decode } \overline{\mathbf{z}}^i \text{ using } \mathbf{Q}_{\overline{\mathbf{z}}|\mathbf{x}^i} \\ &\quad \text{Encode } \mathbf{x}^i \text{ using } \mathbf{P}_{\mathbf{x}|\overline{\mathbf{z}}^i} \\ &\quad \text{Encode } \overline{\mathbf{z}}^i \text{ using } \mathbf{P}_{\overline{\mathbf{z}}} \end{aligned} \tag{A.1}$$

Now in order to reverse this process and gain "bits back" after decompressing every $\mathbf{x}^i$, the receiver has to execute the following steps in pseudo-code:

$$\begin{aligned} &\text{for } i = N : 1 \\ &\quad \text{Decode } \overline{\mathbf{z}}^i \text{ using } \mathbf{P}_{\overline{\mathbf{z}}} \\ &\quad \text{Decode } \mathbf{x}^i \text{ using } \mathbf{P}_{\mathbf{x}|\overline{\mathbf{z}}^i} \\ &\quad \text{Encode } \overline{\mathbf{z}}^i \text{ using } \mathbf{Q}_{\overline{\mathbf{z}}|\mathbf{x}^i} \end{aligned} \tag{A.2}$$

Evidently, the receiver's steps are the exact reverse and opposite of the sender's steps. This means that the compression scheme only works if we are able to decode the symbols in opposite order as they were encoded to the bitstream. This is what we call the *LIFO (Last-In-First-Out) property*, displayed in Figure A.1.

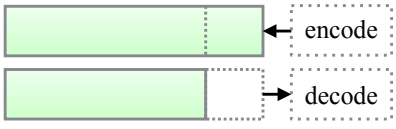



FIGURE A.1: The bits-back scheme works best with a coder that operates in LIFO order: symbols are decoded in opposite order as they were encoded. Range Asymmetric Numeral Systems (rANS) operates on a bitstream in this manner.

Range ANS (rANS), an application of a family of coders introduced by Duda (2009), turned out to be a natural fit for both the scalability requirement and the LIFO property of the bits-back scheme (Townsend, Bird, and Barber, 2019). We explain in general how rANS operates and how it is able to decode symbols that were last encoded. The possibility to integrate rANS with parallel-computing architectures for the purpose of scalability is explained by Giesen (2014).

rANS encodes a (sequence of) data point(s) into a natural number $\nu \in \mathbb{N}$, which is called the *state*. We will use unconventional notation, yet consistent with our work: $x$ to denote a single symbol and $\nu$ to denote the state. The goal is to obtain a state $\nu$ whose length, if converted to binary using standard decimal-binary conversion, grows with a rate that closely matches $-\log_2 p(x^\circ)$ per symbol $x^\circ$ encoded using $p(x)$.

## A.2 rANS for Two Symbols

Suppose we wish to encode a variable $x$ that can take on two symbols $\{s_1, s_2\}$ that have equal probability. Starting with $\nu = 1$. A valid scheme for this distribution is

$$\begin{aligned} s_1 &: \nu \to 2\nu \\ s_2 &: \nu \to 2\nu + 1. \end{aligned} \tag{A.3}$$

This simply assigns 0 to $s_1$ and 1 to $s_2$ in binary. Therefore, it appends 1 or 0 to the right of the binary representation of the state $\nu$. Note that this scheme is uniquely decodable: if the current state is $\nu'$, we can read of the last encoded symbol by telling if the state $\nu'$ is even (last symbol was $s_1$) or odd (last symbol was $s_2$). Consequently, after figuring out which symbol $x$ was last encoded, the state $\nu$ before encoding that symbol is obtained by

$$\begin{aligned} s_1 \ (\nu' \text{ even}) &: \nu \to \frac{\nu'}{2} \\ s_2 \ (\nu' \text{ odd}) &: \nu \to \frac{\nu' - 1}{2}. \end{aligned} \tag{A.4}$$

## A.3 rANS for an Arbitrary Alphabet

Now, for the general case, suppose that the variable $x$ can take on a multitude of symbols $x \in \{s_1 = 1, s_2 = 2, \ldots, s_{|\mathcal{X}|} = |\mathcal{X}|\}$ with probability $\{p_1, p_2, \ldots, p_{|\mathcal{X}|}\}$. In order to obtain a scheme analogous to the case with two symbols $\{s_1, s_2\}$, we have to assign every possible symbol $s_i$ to a specific *subset of the natural numbers* $\mathcal{S}_i \subset \mathbb{N}$, that *partitions* the natural numbers. Consequently, $\mathbb{N}$ is a disjoint union of the subsets $S_i$. Also, the elements in the subset $\nu \in S_i$ corresponding to $s_i$ must be chosen such that they occur in $\mathbb{N}$ with probability $p_i$.

This is accomplished by choosing a multiplier $M$, called the precision of rANS, that scales up the probabilities $\{p_1, p_2, ..., p_{|\mathcal{X}|}\}$. The scaled up probability $p_i$ is denoted by $F[i]$ and the $F$'s are chosen such that $\sum_{i=1}^{I} F[i] = M$. We also choose subsets $\{K_1, K_2, ...\}$ that form intervals of length $M$ and partition the natural numbers. That means, the first $M$ numbers belong to $K_1$, the second $M$ numbers belong to $K_2$, and so on. Then, in every partition $K_n$, the first $Mp_1$ numbers are assigned to symbol

$s_1$ and form the subset $S_{n1}$, the second $Mp_2$ numbers are assigned to symbol $s_2$ and form the subset $S_{n2}$, and so on.

Now, we define $\mathcal{S}_i = \cup_{n=1}^{\infty} S_{ni}$. The resulting subsets $\mathcal{S}_i$ partition the natural numbers $\mathbb{N}$. Furthermore, the elements of $\mathcal{S}_i$ occur with probability approximately equal to $p_i$ in $\mathbb{N}$.

Now, suppose we are given an initial state $\nu$. The scheme rANS can be interpreted as follows. **Encoding** a symbol $s_i$ is done by converting the state $\nu$ to a new state $\nu'$ that equals the $\nu^{\text{th}}$ occurrence in the set $\mathcal{S}_i$. This operation is made concrete in the following coding formula:

$$\begin{aligned} C(x, \nu) &= M\left\lfloor \frac{\nu}{F[x]} \right\rfloor + B[x] + \nu \pmod{F[x]} \\ &= \nu' \end{aligned} \tag{A.5}$$

where $B[x] = \sum_{i=1}^{x-1} F[i]$, $R = \nu' \pmod{M}$ and $\lfloor \, \rfloor$ denotes the floor function.

Furthermore, suppose we are given a state $\nu'$ and we wish to know which number was last encoded (or in other words, we wish to **decode** from $\nu'$). Note that the union of the subsets $\mathcal{S}_i$ partitions the the natural numbers $\mathbb{N}$, so every number can be uniquely identified with one of the symbols $s_i$. Afterwards, if we know what the last encoded symbol $s_i$ was, we can figure out the state $\nu$ that *preceded* that symbol by doing a look-up for $s_i$ in the set $\mathcal{S}_i$. The index of $s_i$ in $\mathcal{S}_i$ equals the state $\nu$ that preceded $\nu'$. This operation is made concrete in the following decoding formula, which returns a symbol-state $(x, \nu)$ pair.

$$\begin{aligned} D(\nu') &= \left( \text{argmax}\{B[x] < R\}, F[x]\lfloor \frac{\nu'}{M} \rfloor + R - B[x] \right) \\ &= (x, \nu) \end{aligned} \tag{A.6}$$

## A.4 Sampling is Equivalent to Decoding

The probabilities introduced in Section A together comprise the following probability mass function:

$$p(x = s_i) = p_i \tag{A.7}$$

We show that decoding a symbol $x^\circ$ using the distribution $p(x)$ from random bits is equivalent to sampling $x^\circ \sim p(x)$. Random bits are a bitstream of arbitrary length $N$ whose bits are sampled from a Binomial distributed with $p = 0.5$. Every bitstream uniquely corresponds to a natural number $\nu \in \mathbb{N}$. Consequently, sampling random bits is equivalent to randomly picking a number $\nu \in \mathbb{N}$. Now, the natural numbers $\mathbb{N}$ is a disjoint union of the subsets $\mathcal{S}_i$ and every subset $\mathcal{S}_i$ uniquely corresponds to a symbol $s_i$. This means that if we randomly pick a number $\nu \in \mathbb{N}$, we will randomly end up in one of the subsets $\mathcal{S}_i$. The elements in the subset $\mathcal{S}_i$ occur with a probability approximately equal to $p_i$. So if we *decode* from this random number $\nu$ using rANS, it will return symbol $s_i$ with probability $p_i$. Therefore, decoding from a

random number $\nu$ is equivalent to sampling a symbol $x^\circ \sim p(x)$.

## A.5 Codelength of rANS

The new state $\nu'$ after encoding $s_i$ using this scheme is approximately equal to $\frac{\nu}{p_i}$. The total length of the bitstream after encoding $s_i$ is:

$$N_{\text{total}} \approx \log_2 \nu + \log_2 \left( \frac{1}{p_i} \right) \tag{A.8}$$

Consequently, the bitstream approximately grows with the correct codelength, namely the self-information:

$$-\log_2 p_i \text{ bits} \tag{A.9}$$

# B Parameterizing Distributions with Neural Networks

A neural network is a non-linear function, denoted by $\text{Net}(\cdot)$, that is typically built up from a series of non-linear vector function compositions. Suppose we have an input vector $\mathbf{x}$, a neural network with parameters $\boldsymbol{\theta}$ can be written as:

$$\text{Net}_{\boldsymbol{\theta}}(\mathbf{x}) = (h_{\boldsymbol{\theta}^{(1)}} \circ h_{\boldsymbol{\theta}^{(2)}} \circ \cdots \circ h_{\boldsymbol{\theta}^{(H)}} \circ \text{out})(\mathbf{x}) \tag{B.1}$$

The input vector $\mathbf{x}$ is called the input layer, the output vector of every function $h_{\boldsymbol{\theta}^{(i)}}(\cdot)$ is called a hidden layer and the output vector after applying the last function $\text{out}(\cdot)$ is called the output layer. Every function $h_{\boldsymbol{\theta}^{(i)}}$ is built up from a subset of the parameters $\boldsymbol{\theta}^{(i)} \subset \boldsymbol{\theta}$. Let $\mathbf{v}$ be the input vector of function $h_{\boldsymbol{\theta}^{(i)}}$. The most basic neural network consists of functions $h_{\boldsymbol{\theta}^{(i)}}(\cdot)$ specified in the following way:

$$h_{\boldsymbol{\theta}^{(i)}}(\mathbf{v}) = \sigma_{\text{hidden}}\left(\mathbf{b}^{(i)} + \begin{bmatrix} \mathbf{v} \cdot \mathbf{w}_1^{(i)} \\ \mathbf{v} \cdot \mathbf{w}_2^{(i)} \\ \vdots \\ \mathbf{v} \cdot \mathbf{w}_K^{(i)} \end{bmatrix}\right) \tag{B.2}$$

where $\mathbf{b}^{(i)} = \left(b_1^{(i)}, \ldots, b_K^{(i)}\right)^\mathsf{T} \subset \boldsymbol{\theta}^{(i)}$ are called the *biases*, $\mathbf{w}_k^{(i)} \subset \boldsymbol{\theta}^{(i)}$ are called the *weights* and $\sigma_{\text{hidden}}(\cdot)$ is a simple differentiable non-linear function, such as the sigmoid function. The functional form of the output function $\text{out}(\cdot)$ depends on the chosen codomain of the neural network function $\text{Net}_{\boldsymbol{\theta}}$. If the neural network is meant to output the distribution parameters of the Logistic distribution, then the outputs are real-valued. The output function consists of two functions: the first $\text{out}_{\boldsymbol{\mu}}(\cdot)$ outputs the values of the $\boldsymbol{\mu}$ parameters and the second $\text{out}_{\mathbf{s}}(\cdot)$ outputs the values of the $\mathbf{s}$ parameters. The two functions have parameters $\boldsymbol{\theta}^{(\text{out})} \subset \boldsymbol{\theta}$ and are typically defined in the following form:

$$\text{out}_{\boldsymbol{\mu}}(\mathbf{v}) = \left(\begin{bmatrix} \mathbf{v} \cdot \mathbf{w}_1^{\text{out}} \\ \mathbf{v} \cdot \mathbf{w}_2^{\text{out}} \\ \vdots \\ \mathbf{v} \cdot \mathbf{w}_M^{\text{out}} \end{bmatrix}\right), \qquad \text{out}_{\mathbf{s}}(\mathbf{v}) = \sigma_{\mathbf{s}}\left(\begin{bmatrix} \mathbf{v} \cdot \mathbf{w}_1^{\text{out}} \\ \mathbf{v} \cdot \mathbf{w}_2^{\text{out}} \\ \vdots \\ \mathbf{v} \cdot \mathbf{w}_M^{\text{out}} \end{bmatrix}\right) \tag{B.3}$$

with $\mathbf{w}_m^{\text{out}} \subset \boldsymbol{\theta}^{\text{out}}$ and $\sigma_{\mathbf{s}}$ is a function that outputs strictly positive values, such as the exponential function. If the neural network is meant to output probabilities $\mathbf{P}$ as distribution parameters for the Categorical distribution, then the outputs must be real-valued between in the interval $(0, 1)$. Now, let the matrix $\mathbf{v}$ be the input of the

function:

$$\mathbf{v} = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1|\mathcal{X}|} \\ v_{21} & v_{22} & \cdots & v_{2|\mathcal{X}|} \\ \vdots & \vdots & \ddots & \vdots \\ v_{P1} & v_{P2} & \cdots & v_{P|\mathcal{X}|} \end{bmatrix} \tag{B.4}$$

The output function $\text{out}_{\mathbf{P}}(\cdot)$ consists of softmax functions that causes every value to be between 0 and 1 and every row to add up to 1:

$$\text{out}_{\mathbf{P}}(\mathbf{v}) = \begin{bmatrix} \frac{\exp(v_{11})}{\sum_{i=1}^{|\mathcal{X}|}\exp(v_{1i})} & \frac{\exp(v_{12})}{\sum_{i=1}^{|\mathcal{X}|}\exp(v_{1i})} & \cdots & \frac{\exp(v_{1|\mathcal{X}|})}{\sum_{i=1}^{|\mathcal{X}|}\exp(v_{1i})} \\ \frac{\exp(v_{21})}{\sum_{i=1}^{|\mathcal{X}|}\exp(v_{1i})} & \frac{\exp(v_{22})}{\sum_{i=1}^{|\mathcal{X}|}\exp(v_{1i})} & \cdots & \frac{\exp(v_{2|\mathcal{X}|})}{\sum_{i=1}^{|\mathcal{X}|}\exp(v_{1i})} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\exp(v_{P1})}{\sum_{i=1}^{|\mathcal{X}|}\exp(v_{1i})} & \frac{\exp(v_{P2})}{\sum_{i=1}^{|\mathcal{X}|}\exp(v_{1i})} & \cdots & \frac{\exp(v_{P|\mathcal{X}|})}{\sum_{i=1}^{|\mathcal{X}|}\exp(v_{1i})} \end{bmatrix} \tag{B.5}$$

Neural networks are typically represented with a graph structure. An example with 1 hidden layer can be seen in Figure B.1. Every node in the graph represents a vector entry of a layer.

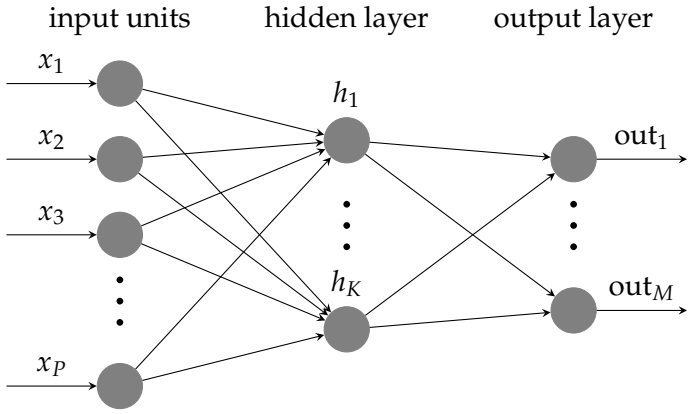



FIGURE B.1: Graph diagram neural network function $\text{Net}_{\boldsymbol{\theta}}(\mathbf{x})$ with one hidden layer.

## B.1 Challenges and Tricks for Neural Networks

A latent variable model based on neural networks has the advantage of being very flexible. The flexibility strongly relies on the number of hidden layers and the number of hidden units per layer. Using a large number of hidden layers/units we hope it learns the general patterns of the train-set $\mathcal{D}$ very well. However, optimization of neural networks using gradient descent is unstable unless certain guidelines are followed. There are many optimization challenges that have to be conquered, but describing them distract from the main point of this report. So we shed some light on a few and mention what we did to overcome them.

Perhaps the biggest disadvantage of a neural network is that it may also learn the characteristics of each datapoint's unique noise, called *overfitting*. We try to prevent overfitting using Dropout (Srivastava et al., 2014) and early stopping (Prechelt, 1998).

Also, parameterization of a model with neural networks results in a highly *non-convex* ELBO $\mathcal{L}_{\theta,\phi}(\mathbf{x})$. This makes the optimization problem NP-complete (Judd, 1990) and makes it almost impossible to find a global optimum. Therefore, we are bound to optimization towards a "good" local optimum. The loss-surface contains many local optima. The trick lies in balancing between optimization run-time and the quality of a local optimum. If we do not pay close attention then it could happen that gradient descent gets stuck in an unsatisfactory local optimum. We employ stochastic gradient descent (Robbins and Monro, 1951) to prevent becoming stuck in a local optimum. Furthermore, we use Polyak-averaging of the parameters (Polyak and Juditsky, 1992) and learning rate annealing during optimization to get better solutions.

Furthermore, in case of a large amount of hidden layers, the gradient of the ELBO $\mathcal{L}_{\theta,\phi}(\mathcal{D})$ with respect to some parameters could be vanishingly small (Hochreiter et al., 2001), called the *vanishing gradient problem*. This effectively prevents the parameters from changing its value after one step of gradient descent. We use residual connections between the layers (He et al., 2015) to prevent the gradients from vanishing. This stands in contrast to the *exploding gradient problem*. As the name suggests, this describes the difficulty in which the gradients get exceedingly large, often in case of a large amount of hidden layers. We use a technique called gradient clipping, which sets an upper bound to the magnitude of the gradient vector. This prevents the gradient from getting too large.

Before we start the gradient descent procedure, we have to set the initial values of the neural network parameters. The *initialization* of neural networks greatly determnines how fast the model converges. Bad initialization can significantly slow down optimization (Sutskever et al., 2013). We use an initialization method called Data-Dependent Initialization (Salimans and Kingma, 2016). Furthermore, optimization of neural networks with gradient descent is often *ill-conditioned*. This is a common cause of slow and inaccurate results (Martens, 2010). We counteract the conditioning problem by using Weight Normalization (Salimans and Kingma, 2016). There are also a few challenges that are specific to optimization with the VAE-framework. Perhaps the biggest challenge is *posterior collapse* (Bowman et al., 2015; Serban et al., 2016; Kaae Sønderby et al., 2016), which can be effectively prevented using the *free-bits technique* (Chen et al., 2016).

Last, but not least, we employ *convolutional* neural networks (LeCun et al., 1989). These neural networks have a particular functional form that is specialized for data $\mathbf{x}$ whose dimensions are naturally organized into a grid, like images. With convolutional neural networks it is easier to optimize the latent variable model for image data.

All the other details regarding implementation can be found in the paper written by Kingma, Abbeel, and Ho, 2019 or in the code at `https://github.com/fhkingma/bitswap`.

# C Stochastic Gradient Descent

We must find the neural network parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ that maximize the ELBO criterion:

$$\underset{\boldsymbol{\theta},\boldsymbol{\phi}}{\operatorname{argmax}} \frac{1}{N_{\mathcal{D}}} \sum_{i=1}^{N_{\mathcal{M}}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i) \tag{C.1}$$

In Section 4.1.4, we constructed estimates of the ELBO and its two gradients:

$$\begin{aligned} \overline{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i) &\simeq \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i) \\ \nabla_{\boldsymbol{\theta}} \overline{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i) &\simeq \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i) \\ \nabla_{\boldsymbol{\phi}} \overline{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i) &\simeq \nabla_{\boldsymbol{\phi}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i) \end{aligned} \tag{C.2}$$

During optimization, the ELBO is a function of its parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$. For conciseness, we denote the criterion as:

$$\overline{\mathcal{L}}_{\mathcal{D}}(\boldsymbol{\theta},\boldsymbol{\phi}) \equiv \frac{1}{N_{\mathcal{D}}} \sum_{i=1}^{N_{\mathcal{D}}} \overline{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i) \tag{C.3}$$

The non-linearity of the functions $f_{\boldsymbol{\theta}}$ and $g_{\boldsymbol{\phi}}$ may cause the objective to be non-convex. The non-convexity could make it hard to obtain an exact global maximum value of the objective, especially if $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ are high-dimensional vectors. Therefore, we use a first-order numerical optimization method called *gradient ascent*, in which we utilize the gradients (first-order derivatives) of the ELBO $\overline{\mathcal{L}}_{\mathcal{D}}(\boldsymbol{\theta},\boldsymbol{\phi})$ with respect to the parameter vectors $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ to find a local maximum.

We describe the gradient ascent procedure to jointly update the parameter vectors

$$\Theta = \{\boldsymbol{\theta},\boldsymbol{\phi}\}. \tag{C.4}$$

Suppose we start from an initial parameter setting $\Theta^{(0)}$, then the ELBO $\overline{\mathcal{L}}_{\mathcal{D}}(\Theta)$ increases *fastest* with respect to $\Theta$ if one goes from $\Theta^{(0)}$ in the direction of the positive gradient $\nabla_{\Theta}\overline{\mathcal{L}}_{\mathcal{D}}\left(\Theta^{(0)}\right)$. It follows that, if

$$\Theta^{(1)} \leftarrow \Theta^{(0)} + \alpha \nabla_{\Theta}\overline{\mathcal{L}}_{\mathcal{D}}\left(\Theta^{(0)}\right) \tag{C.5}$$

for $\alpha$ small enough, then

$$\overline{\mathcal{L}}_{\mathcal{D}}\left(\Theta^{(0)}\right) \leq \overline{\mathcal{L}}_{\mathcal{D}}\left(\Theta^{(1)}\right). \tag{C.6}$$

In other words, the term $\alpha \nabla_{\Theta}\overline{\mathcal{L}}_{\mathcal{D}}\left(\Theta^{(0)}\right)$ is added to $\Theta^{(0)}$ because we want to move towards the gradient, towards the maximum. With this observation in mind, one

considers a sequence $\Theta^{(0)}, \Theta^{(1)}, \Theta^{(2)}, \ldots$ such that

$$\Theta^{(t+1)} \leftarrow \Theta^{(t)} + \alpha \nabla_{\Theta} \overline{\mathcal{L}}_{\mathcal{D}} \left( \Theta^{(t)} \right) \tag{C.7}$$

We have a monotonic sequence

$$\overline{\mathcal{L}}_{\mathcal{D}} \left( \Theta^{(0)} \right) \leq \overline{\mathcal{L}}_{\mathcal{D}} \left( \Theta^{(1)} \right) \leq \overline{\mathcal{L}}_{\mathcal{D}} \left( \Theta^{(2)} \right) \leq \cdots \tag{C.8}$$

so hopefully the sequence $\left( \Theta^{(t)} \right)_{t=0}^{\infty}$ converges to a desired local maximum.

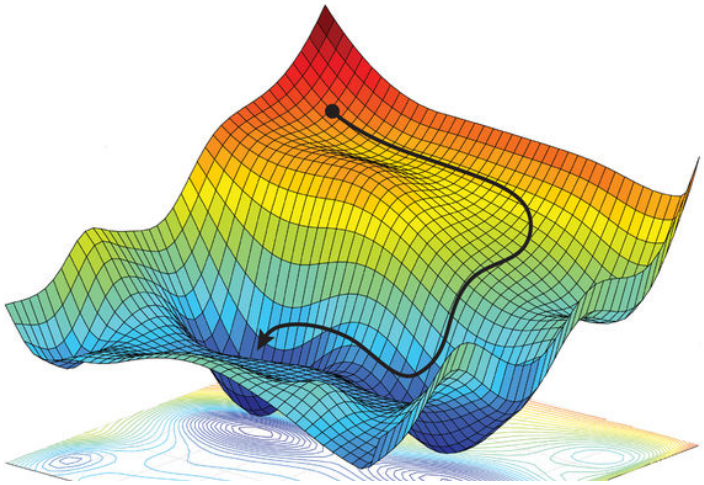


FIGURE C.1: The *negative* ELBO can be imagined as a hilly landscape, where every valley corresponds to a local minimum of the negative ELBO. Finding a valley is done by walking towards the direction that lets you decrease the fastest in every step. Image taken from (Hutson, 2018).

More specifically, gradient ascent updates both parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ to obtain an an ELBO closer to a local maximum, with step-size $\alpha \in \mathbb{R}_+$:

$$\begin{aligned} \boldsymbol{\theta}^{(t+1)} &\leftarrow \boldsymbol{\theta}^{(t)} + \alpha \nabla_{\boldsymbol{\theta}} \overline{\mathcal{L}}_{\mathcal{D}} \left( \boldsymbol{\theta}^{(t)}, \boldsymbol{\phi}^{(t)} \right) \\ \boldsymbol{\phi}^{(t+1)} &\leftarrow \boldsymbol{\phi}^{(t)} + \alpha \nabla_{\boldsymbol{\phi}} \overline{\mathcal{L}}_{\mathcal{D}} \left( \boldsymbol{\theta}^{(t)}, \boldsymbol{\phi}^{(t)} \right) \end{aligned} \tag{C.9}$$

An easy way to understand this procedure is by imagining that the *negative* ELBO is a hilly landscape, where every valley corresponds to a local minimum. The goal is to find a valley starting from random place in the landscape, but only by looking at the steepness of the ground directly under your feet. A simple and intuitive strategy is to be to walk in the direction that brings you most far down within one step. Do this for every step and you will find a valley. The process is visualized in Figure C.1.

Most likely, we have to do many updates of the parameters to obtain an adequate maximum value of the average of ELBO's. We divide the train-set up into smaller subsets $\mathcal{M} \subset \mathcal{D}$ of equal size, called *mini-batches*. The ELBO evaluated on the mini-batch $\mathcal{M}$ is an unbiased Monte-Carle estimator of the ELBO evaluated on the full

dataset $\mathcal{D}$:

$$\frac{1}{N_{\mathcal{M}}} \sum_{i=1}^{N_{\mathcal{M}}} \overline{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i) \simeq \frac{1}{N_{\mathcal{D}}} \sum_{i=1}^{N_{\mathcal{D}}} \overline{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^i) \tag{C.10}$$

The same holds for its gradients. Every iteration we update the gradient and the parameters using a different mini-batch, called *stochastic gradient descent* (Robbins and Monro, 1951). An *epoch* refers to the point after which we have iterated through the entire train-set $\mathcal{D}$.