Speech Coding using Deep Neural Networks and the Information Bottleneck Principle

Barbara Martinovic
Mathematical Engineering, Aalborg University

Master's Thesis

AALBORG UNIVERSITY

STUDENT REPORT

Mathematical Engineering
Aalborg University

Department of Electronic Systems
Fredrik Bajers Vej 7
9220 Aalborg
http://es.aau.dk

Department of Mathematical Sciences
Skjernvej 4A
9220 Aalborg
http://math.aau.dk

Title:
Speech Coding using
Deep Neural Networks and
the Information Bottleneck Principle,

Project Period:
Long Master's Thesis E18-F19
(60 ECTS)

Project Group:
MATTEK9-10 5.213b

Participant(s):
Barbara Martinovic

Supervisor(s):
Jan Østergaard

Industry Partner(s):
Ricco Jensen (RTX A/S)
Peter Mariager (RTX A/S)

Copies: 1

Page Numbers: 104

Date of Completion:
7th of June 2019

Abstract:

In this project the possibility of using Deep Neural Networks (DNNs) and the Information Bottleneck (IB) principle to perform speech coding is explored. An end-to-end strategy using DNNs in form of autoencoders is developed and the DNNs are trained using both synthetic data and speech files from the TIMIT database. Signals are encoded using a b-bit scalar quantizer employed internally in the DNNs and the bit rate is easy controllable by parameters of the quantizer amongst others. It was found that the the developed speech autoencoders trained with the Mean Squared Error (MSE) as a objective function did not outperform the results obtained by encoding signals using the BroadVoice32 (BV32) codec in terms of both bit rate and Perceptual Evaluation of Speech Quality (PESQ) scores. The DNNs outperformed the BV32 codec in terms of PESQ scores for bit rates of 5 bit per sample or higher. By exploring the marginal entropies it was possible to achieve an average PESQ score of 4.46 and standard deviation of 0.03 for the DNN speech autoencoders and by using a bit rate less than half the bit rate used for standard 16-bit Pulse Code Modulation encoding.
A loss function involving the MSE and marginal entropies was proposed inspired by the IB principle. However it was not possible to find adequate weights such that the loss function was suitable for training DNN speech autoencoders.

Danish Summary

Dette kandidatspeciale omhandler brugen af såkaldte dybe neurale netværk samt informationsflaskehalsprincippet til at udføre digital talekodning i transmissionssammenhænge.
Kodning af signaler anvendes overalt i hverdagen, og højere efterspørgsel af digitale kommunikationssystemer samt hardware begrænsninger gør bl.a. talekodning til et profileret forskningsområde.

Digitale signaler er repræsenteret med binære cifre eller såkaldte bits, og formålet med talekodning er at komprimere digitale talesignaler således, at den komprimerede version indeholder færre bits end den originale version. Komprimeringen udføres af en såkaldt indkoder, og det er det indkodede signal der transmitteres. Efter endt transmission rekonstrueres signalet ved brug af en dekoder. Formålet med talekodning er således at komprimere talesignalet så meget som muligt, samtidig med at kvaliteten ønskes bevaret. I stort set alle anvendte talekodningsalgoritmer er komprimeringen af talesignalet til færre bits baseret på forudgående viden om signalet og specifikke antagelser. Da de dybe neurale netværk primært er data-drevne, er ideen i dette projekt at lade taledataet "tale" for sig selv. De neurale netværk findes ved løsning af optimeringsproblemer baseret på en masse dataeksempler. Denne proces kaldes også træning.

I projektet præsenteres først den nødvendige teori i form af en gennemgang af den generelle talekodningsprocess samt en forklaring af neurale netværk og typer heraf. Herefter præsenteres udformningen af det neurale netværk udviklet til talekodning i dette projekt. Eftersom al komprimering i en eller anden forstand indeholder spørgsmål om, hvad der er relevant information og dermed skal bibeholdes, og hvad der ikke skal, trækkes der i projektet også tråde til informationsteori og det såkaldte informationsflaskehals princip. I projektet foreslås et optimeringsproblem til træning af neurale netværk til brug i talekodning. Efterfølgende udføres en lang række praktiske eksperimenter både på tale og syntetisk data. Der undersøges først forskellige input strategiers indflydelse på kvaliteten af rekonstruerede syntetiske signaler samt bit raten. Efterfølgende udføres eksperimenter på rigtige talesignaler og den opnåede performance sammenlignes med en eksisterende talekoder. Resultaterne viser bl.a. at der ved brug af neurale netværk ikke er muligt at opnå resultater, der

slår den eksisterende talekoder – både mht. bit rate og det perceptuelle PESQ mål for talekvalitet. For bit rater højere en 5 bits per talesample er det dog muligt at slå den eksisterende koder i forhold til PESQ målet. Resultaterne viser også at det ikke er mulig at konfigurere det foreslåede optimeringsproblem, således det kan anvendes til træning af talekodnings neurale netværk.

Disse samt andre resultater diskuteres i slutningen af rapporten, hvortil der også diskuteres alternative fremgangsmåder og parametre der har haft muligt indflydelse på resultaterne. Til slut drages der en række konklusioner.

Preface

This Master's thesis (60 ECTS) is written by me, a Master student in Mathematical Engineering at Department of Mathematical Sciences and Department of Electronic Systems at Aalborg University.

This project was proposed by RTX A/S and I would like to thank Peter Mariager, Ricco Jensen and Jan Østergaard (with department of Electronic Systems at Aalborg Universtity) for their guidance, helpful insights and discussions.

The IEEE-method is used for references. The scripts in this report were developed in Python 3.6 and using Tensorflow 1.4.1 with the Keras module and are uploaded alongside the report. Unless otherwise stated, all figures in the report are made by me.

As for prerequisites, the reader is expected to be familiar with general digital signal processing, time series theory, probability theory and statistics. It is also an advantage to have knowledge of basic information theory, basic speech coding and the principles of machine learning.

Aalborg University, 7$^{\text{th}}$ of June 2019

Barbara Martinovic

Contents

1 Introduction 1
1.1 Objectives 2

I Part One - Theory

2 Digital Speech Coding 5
2.1 General Structure 6
2.1.1 Source Signal 6
2.1.2 Analysis Filter Bank 7
2.1.3 Quantization 8
2.1.4 Channel 9
2.2 Quantization 9
2.2.1 Scalar Quantization 10
2.2.2 Vector Quantization 15
2.3 Performance and Quality Assessment 17
2.4 The BroadVoice32 (BV32) Codec 18

3 Deep Neural Networks (DNNs) 19
3.1 General Concept 19
3.2 1D Convolution 22
3.3 Autoencoders 27
3.4 Activation Functions 28
3.4.1 Identity 28
3.4.2 Tanh 28
3.4.3 Exponential Linear Unit (ELU) 29
3.5 Training 31
3.5.1 Loss Functions 32

3.5.2 Learning Algorithms 32
3.5.3 Backpropagation 34
3.5.4 Under- and Overfitting 35

4 Our Speech Autoencoder 37
4.1 General Structure 37
4.2 The Quantization Layer 39

5 Information and Rate-distortion Theory 43
5.1 Entropy 43
5.2 Mutual Information 46
5.3 Rate-distortion Theory 47

6 Information Bottleneck 49
6.1 Information Bottleneck in DNNs 49
6.2 Loss Function for Speech Autoencoder Inspired by Information Bottleneck 52
6.2.1 Estimating the Entropy using a 1-bit Quantizer 53
6.2.2 Estimating the Entropy using a b-bit Quantizer 54

II Part Two - Application

7 The Data 61
7.1 Generating Synthetic Data 61
7.2 TIMIT Speech Data 62

8 Encoding and Decoding Synthetic Data using DNNs . . 63
8.1 Network Structure 63
8.1.1 Input and Output Types 65
8.2 Results 68
8.2.1 General Performance 69
8.2.2 Analysis of the Reconstruction 76
8.2.3 Dimension Versus Bits 78
8.3 Preliminary Discussion 79

9 Speech Coding using DNNs 81
9.1 Frame size of 16 samples 84
9.2 Frame size of 128 samples 87
9.3 Speech Coding using the Information Bottleneck Principle 90
9.4 Variable Rates 93

III Discussion and Conclusion

10 Discussion 99

11 Conclusion 103

IV Bibliography

Bibliography 107

V Appendix

A Results using Synthetic Data 115

A.1 Balanced Network 115

A.2 Undercomplete Network 121

B Spectrograms of Reconstructed Signals 127

C Information Plane Plots 131

1. Introduction

Digital coding is a central part of the digital age, we live in. Whether using mobile phones, playing DVDs or streaming tv-series and music, one encounters compressed digital signals. Digital *speech coding* is the process of obtaining a compressed representation of a digital speech signal in order to efficiently store or transmit it [49]. By using a digital speech coding algorithm the digital speech signal, which is represented using binary digits or *bits*, is transformed into a compact representation requiring fewer bits. This part is known as *encoding*. The compact representation is then either transmitted or stored. After ended transmission or storage a reconstructed version of the original signal is obtained from the compact representation by the *decoding* part of the speech coding algorithm. The need for speech coding techniques is governed by an increasing demand of digital speech communication systems [16] and/or limitations of the transmission and storage channels [59, pp. 4-5]. Furthermore by compressing signals, more bits can be allocated for other processes such as e.g. ensuring privacy of the transmitted speech signals [49]. Representing a speech signal using fewer bits is in general a lossy procedure [16], i.e. the original signal is in general not perfectly reconstructed. The overall goal in digital speech coding - or coding in general - is thus two-sided; to obtain a compressed version of a signal in terms of fewer bits meeting the requirements for transmission and/or storage, while at the same time maintaining a degree of quality in the reconstructed signal needed for its purpose [7, p. 5].

There exists a wide range of different speech coding algorithms and the nature of a coding algorithm is highly related to the application in which it is used. Most state-of-the-art speech coding algorithms can be distinguished into three classes [48][49]; waveform coding, parametric coding (vocoding) and hybrid coding involving both waveform and parametric coding. Waveform coding explores as the name suggests the waveform of speech signals, while vocoders model the speech production system and extract its features using a parametric model [16]. A well-known waveform coder is *Pulse Code Modulation (PCM)*, which is a standard coder used for representing digital speech signals [49].

Regardless of whether waveform, parametric or hybrid coding is used, all existing

speech coders rely to some extend on some prior knowledge about speech signals and their nature. However recently a new approach [41] paved the way for the possibility of using *Deep Neural Networks (DNNs)* for speech coding.
DNNs have obtained state-of-the art performances in various fields such as image recognition [32] [32], natural language processing [51] and speech recognition [25] [43]. DNNs process data inputs by a cascade of transformations. The suitable transformations needed for solving a task are found iteratively through optimization like problems using data examples and a adequate objective function [34]. Hence DNNs can be seen as data driven processing systems. Recently the authors in [41] were able to develop a compression algorithm for images using DNNs, which outperformed existing image compression algorithms such as JPEG. This project is thus inspired by their results and by the desire to develop a data driven speech coding algorithm. To our best knowledge there are no existing speech coding algorithms based on DNNs.

Despite DNNs being able to obtain state-of-the-art performances in various fields, there is still little knowledge as to why the networks are able obtain such performances [60]. Due to this, DNNs are often considered as *black boxes* [4]. One way to analyse DNNs is through the use of the *Information Bottleneck (IB)* method [53]. The IB method is a method for compressing random variables while at the same time maintaining relevant information w.r.t. some other variable. In [52] it was proposed that the objectives of a DNN could be explained by the IB principle. The authors in [47] furthermore showed via the IB principle that a network undergoes two phases during optimization; a phase of information increase and *compression phase*.

Regardless of what compression algorithm is considerd, compression involves determining what is relevant and what is not. It is now clear that when considering data driven speech coding, the fields of speech coding, DNNs and the IB principle are somehow related. This project explores these relations by examining the possibility of using DNNs and the IB principle for speech coding.

1.1 Objectives

The main objective of this master's thesis is to:

Examine and develop a speech coding algorithm using DNNs and the IB principle.

In order to do so, we will present the necessary theory. This project is written in collaboration with RTX A/S and some delimitations have been provided by the company. We will restrict our focus to lossy speech coding for transmission. The primary focus will be on fixed bit rate transmission and we will assume that the transmission channel is error-free or *lossless*. We limit the examination to DNN and IB principle aspects such as network architecture and objective functions, while speech coding aspects are limited to bit rate, distortion and performance. We do not require the developed speech coding algorithm to work in real time. Thus we do not examine aspects such as computation complexity, memory and power consumption and the time needed for encoding, transmitting and decoding a speech signal.

Part One - Theory

2 Digital Speech Coding 5
2.1 General Structure
2.2 Quantization
2.3 Performance and Quality Assessment
2.4 The BroadVoice32 (BV32) Codec

3 Deep Neural Networks (DNNs) .. 19
3.1 General Concept
3.2 1D Convolution
3.3 Autoencoders
3.4 Activation Functions
3.5 Training

4 Our Speech Autoencoder 37
4.1 General Structure
4.2 The Quantization Layer

5 Information and Rate-distortion Theory 43
5.1 Entropy
5.2 Mutual Information
5.3 Rate-distortion Theory

6 Information Bottleneck 49
6.1 Information Bottleneck in DNNs
6.2 Loss Function for Speech Autoencoder Inspired by Information Bottleneck

2. Digital Speech Coding

The purpose of this chapter is to provide a brief overview of the general digital coding process in the context of speech transmission. We emphasize that the overview is indeed general. There exists a wide range of different speech compression techniques ranging from simple to more complicated ones and the complexity of a coding algorithm is often highly related to the application in which it is used [48]. For instance one expect higher audio (and video) quality from DVDs than the conversations transmitted when using mobile phones. Furthermore it is worth mentioning that most state-of-the art speech coding algorithms consists of several processes [49], many of which are not described here. The processing elements and examples provided here are presented in order to give a general understanding. Some elements are described more comprehensively due to their importance in this project.

As described in the introduction, the overall goal in digital speech coding is two-sided; to obtain a compressed version of a speech signal in terms of fewer bits meeting the requirements for transmission, while at the same time maintaining a degree of quality of the speech signal needed for its purpose [7, p. 5]. The signal to be compressed is also referred to as the *source signal* and the general digital coding procedure for transmission is illustrated in Figure 2.1.

Figure 2.1: General digital coding process.

As previously described and as seen from Figure 2.1, the general coding process consists of two parts; an *encoding* and a *decoding* part [7, p. 4]. For this reason a coding algorithm is often abbreviated as *codec*. In the encoding part a compressed version of the source signal is obtained, meaning that the compressed signal is represented using fewer bits than the bits used to represent the source signal. The compressed version is then transmitted. In the decoder a reconstructed version of the source signal, $\hat{\boldsymbol{x}}$, is obtained from the compressed and transmitted version of the source

signal. If the signal is perfectly reconstructed, i.e. $\boldsymbol{x} = \hat{\boldsymbol{x}}$, the codec is referred to as *lossless* [59, p. 6]. On the other hand, if the codec is not able to perfectly reconstruct the source signal, i.e. some information is lost in the coding process, the codec is referred to as *lossy*. As described in the objectives, we focus on lossy codecs in this project.

In Section 2.1 the general coding process in Figure 2.1 is expanded and its elements are described. In Section 2.2 we describe one of the coding elements more comprehensively, due to its essential role in the speech codec developed in this project. Section 2.3 describes a objective measure for assessing speech quality, while Section 2.4 describes an existing speech codec.

2.1 General Structure

In Figure 2.2 the general coding process from Figure 2.1 is expanded into several parts. The different steps will be presented in the following subsections.

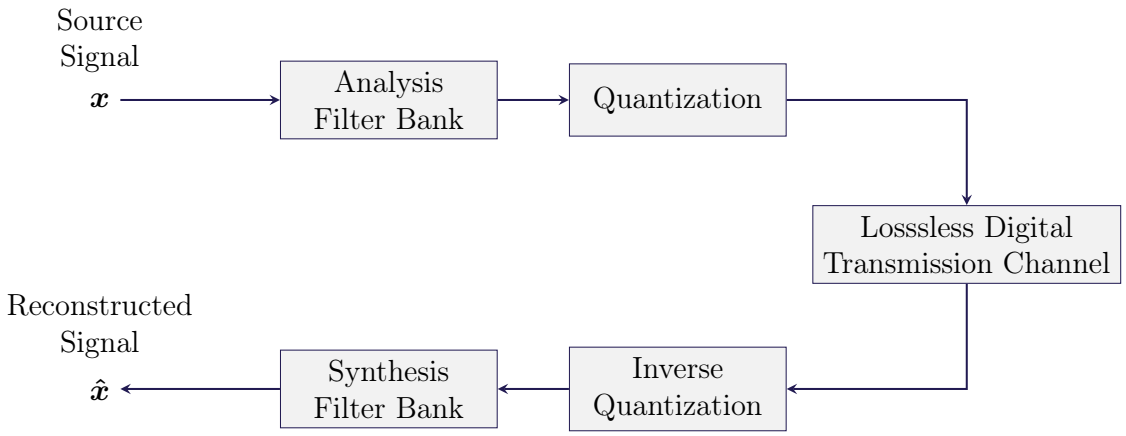


Figure 2.2: General digital speech coding process. Inspired from [59, p. 15].

In Figure 2.2, the processing elements of the source signal prior to the transmission channel constitute the encoding part, while processing elements after the transmission channel constitute the decoding part. The elements in the decoding part often involve simply performing the inverse procedure of the encoding part [59, p. 14] and are therefore not described in detail in the following sections.

2.1.1 Source Signal

Let $\boldsymbol{x}$ denote the speech signal to be compressed, i.e. the source signal. Speech itself is both continuous in time and amplitude, and is thus analogous by nature. By using a microphone speech can be transformed into a continuous electrical signal in form of voltage changes. In order to obtain a digital signal, the electrical signal is both sampled and quantized to obtain a discrete time and discrete amplitude valued (digital) signal. This is done by an *analogue-to-digital converter (ADC)*. Describing ADCs is out of the scope for this project and we refer the reader to [40] for more information about sampling theory and ADCs. However the conversion of an analogue signal to a digital signal is itself a compression procedure, since continuous signals with

infinite resolution are mapped to discrete signals with finite resolution. One standard representation of digital signals is through the use of *Pulse Code Modulation (PCM)* [59, p. 3].

In PCM the analogue signal is uniformly sampled with a sampling frequency f_s to obtain a discrete time signal. The discrete time signal is then *quantized* (see Section 2.1.3 and Section 2.2) to obtain a discrete amplitude valued signal with b bits per sample. The resulting signal is digital and the sampling frequency and number of bits used per sample influence the total amounts of bits used. For an N_{ch} channel PCM speech signal, the total number of bits per second (bps) R, also referred to as the *bit rate*, is given as [59, p. 4]:

$$R = b \cdot f_s \cdot N_{ch} \tag{2.1}$$

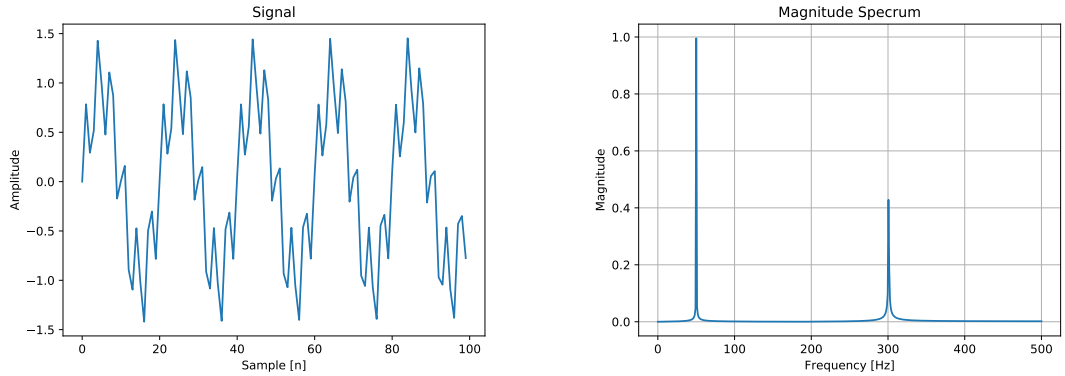
The term channel should not be confused with the term channel used in e.g. Figure 2.2. The term channel in (2.1) refers to the number of speech or audio tracks. For instance speech recorded by a single microphone consists of one audio track or one channel while CD recordings consists of two channels [59, p. 4]. Signals consisting of one and two channels are referred to as *mono* and *stereo* channel signals, respectively. In this project we will only consider mono channel signals, and thus for the rest of this report and unless otherwise stated we assume $N_{ch} = 1$.

The process of converting an analogue signal to a digital signal using e.g. PCM is by nature a lossy compression process. However PCM signals are often considered as uncompressed [49], since the conversion is merely a necessary step in order to obtain digital signals. In this project we will use PCM signals stored in the `wav`-file format and encoded with 16 bits per sample as source signals.

Consider a 16-bit mono channel PCM speech signal sampled at a sampling frequency of f_s. The objective of a speech coding algorithm is thus to reduce the bit rate in (2.1) by compressing the signal in some manner. Since we focus on lossy compression, a difference or *distortion* between the source signal and reconstructed signal is always present. Although it is obvious that one can always achieve an arbitrary low bit rate, throwing an arbitrary amount of bits away without care will most likely result in a significant distortion. Thus the compression must be done in such a way that the distortion is minimal or imperceptible while at the same time the rate is minimal. How this can be done is explored in the following sections.

2.1.2 Analysis Filter Bank

The purpose of the analysis filter bank is to explore the structure and correlations in the source signal, so that bits can be allocated to more relevant and descriptive parts of the signal [59, pp. 8-11]. For instance by considering the source signal from a perceptual point of view. By exploring the signal and allocating bits to the relevant parts, it is the hope that the bit rate can be reduced without the distortion being (too) perceptible. For instance consider the source signal in Figure 2.3a sampled at a sampling frequency of 1 kHz and consisting of a superposition of two sinusoidal waves with frequencies of 50 Hz and 300 Hz, respectively.



(a) Source signal in the time domain. **(b)** Source signal in the frequency domain.

Figure 2.3: A (a) source signal and its (b) magnitude response.

Transmitting such signal with 16 bits per sample requires a bit rate of 16 kbps. By transforming the signal into the frequency domain using the *Discrete Fourier Transform (DFT)* (see [40]), one obtains the magnitude spectrum in Figure 2.3b. As seen the magnitude spectrum consists of a few DFT coefficients with a large magnitude, while the remaining coefficients are close to zero. One approach of coding is thus to explore the frequency content of a signal and assign bits to the DFT coefficients based on their magnitudes [59, p. 11]. I.e. the coefficients with a large magnitude can be assigned bits, while fewer bits if any are assigned to the coefficients with a magnitude close to zero. Thus the coefficients are transmitted rather than the signal itself. Since the DFT has an invertible counterpart, the signal can be reconstructed using the inverse discrete Fourier transform in the synthesis filter bank and by the knowledge of the phase of the signal.

The term filter bank often refers to the source signal being transformed into the frequency domain, i.e. by (a bank of) filters as with e.g. the DFT, but need not to be. Another common approach is to use *Linear Predictive Coding (LPC)* (see e.g. [59]). The basic idea in LPC is to predict a current sample of a speech signal by using past samples and a mathematical model of the speech production system [59, pp. 9-10]. The difference between the original signal and predicted signal produces an error signal. Typically the magnitude of the error signal is significantly smaller than the source signal itself, resulting in a smaller range of values or *dynamic range* and hence fewer bits are needed to represent the error signal compared to the source signal [59, p. 10]. The original signal can be restored in the synthesis filter bank from the error signal and by the parameters used for the prediction.

2.1.3 Quantization

While the task for the analysis filter bank is to explore the structure of a signal, the actual bit reduction is handled by a quantization process or *quantizer* [59, p. 18]. In a quantization procedure the values of a signal or its features with a possible infinite range are mapped to a discrete set of values. A signal processed by a quantizer is said to be *quantized*. The set of values forming a quantizer is often of a cardinality much smaller than the set of possible values for the signal given as

input to the quantizer. Thus by quantizing a signal, the number of bits used for representing and describing the signal can be reduced and hence the bit-rate can be reduced. It is the output of the quantization process that is transformed to a bitsream and transmitted over the channel [59, p. 20]. Thus in the DFT and LPC schemes described in the previous section, both the features and parameters such as prediction parameters and phase are quantized and transmitted. As described in Section 2.1.1 quantization already happens when an analogue signal is converted to a digital signal. Thus the quantization step performed on the output from the analysis filter bank is essentially a re-quantization. Since there is no difference between quantization and re-quantization, the term quantization will be used ambiguously. There exists a wide range of quantization methods, and we will describe quantization more comprehensively in Section 2.2.

2.1.4 Channel

As explained in the previous section, it is the outputs from quantization process that is transformed into a bitstream and transmitted over the channel. From the transmitted encoded signal a reconstructed version of the source signal is obtained in the decoding part. However it is worth noting than in practise bit errors on the channel occur [7, p. 10]. Thus since we assume a lossless channel, mechanisms to handle and correct such possible errors must be employed. Furthermore a mechanism for obtaining bitstreams must also be employed. For this project it suffices to assume that such mechanism are handled by external systems, and we will not describe these any further.

Although not directly linked to the channel, it is also worth noting that a source signal is rarely encoded in its entire length [59, p. 16]. For instance consider a telephone conversation. A conversation puts demands on how much time the encoding and decoding processes can take before the conservation becomes intolerable. Furthermore encoding and decoding large signals require large hardware resources. Thus the entire coding process is often performed on smaller "blocks" or *frames* of duration of time, e.g. 20 ms. We will refer to the length or duration of a frame as the *frame size*. There exists a wide range of different techniques for obtaining frames, depending on the nature of the signal and the processes used in e.g. the analysis filter bank. However one must be careful by how the division and reconstruction of the blocks are made, since unwanted artefacts at the boundaries can be introduced [59, p. 11]. We will not describe this any further since, as will be seen in Chapter 9, considerations regarding boundary effects are not needed in the speech codec developed in this project.

2.2 Quantization

In this section the quantization process briefly presented in Section 2.1.3 is described more comprehensively. When the quantization is performed on one sample at a time, the quantization is referred to as *scalar* quantization and will be described in Section 2.2.1. It is also possible to quantize a collection of samples at a time. This is referred to as *vector* quantization and will be described in Section 2.2.2. In this section the term signal will refer to the input to a quantization procedure and could

for instance be the analogue electrical signal uniformly sampled in a PCM process or the signal features obtained in the analysis filter bank.

2.2.1 Scalar Quantization

To quantize a signal using scalar quantization, consider it being a realization of a continuous stochastic process X with probability density function p. Furthermore assume that the stochastic variable takes values in the range $\mathcal{X}$. For now we will assume that the range is finite and given as $\mathcal{X} = [X_{min}, X_{max}]$, i.e. the probability density function is bounded with left bound X_{min} and right bound X_{max}. The range of values that a quantizer covers is referred to as the *dynamic range* [59, p. 28], and for now we will assume that the dynamic range is $\mathcal{X}$.

Let $x \in \mathcal{X}$ denote a sample of the signal. In scalar quantization each sample is quantized at a time and to quantize a sample using scalar quantization the following is required [59, p. 21]:

- Partitioning of the range $\mathcal{X}$ into M *decision intervals* $\{\delta_q\}_{q=1}^{M}$ with lengths $\{\Delta_q\}_{q=1}^{M}$.
  The decision intervals are defined by the $M+1$ *decision boundaries* $\{b_q\}_{q=0}^{M}$ such that:
  $$\delta_q = [b_{q-1}, b_q) \quad , \quad q = 1, \ldots, M \tag{2.2}$$
  and where:
  $$\Delta_q = b_q - b_{q-1} \quad , \quad q = 1, \ldots, M \tag{2.3}$$
- Assigning of an unique index value to each decision interval. The index values forms an index set $\mathcal{I} = \{q_1, \ldots, q_M\}$.
- Deciding a unique *quantized value* for each decision interval. The quantized values form the set $\hat{\mathcal{X}} = \{\hat{x}_{q_1}, \ldots, \hat{x}_{q_M}\}$.

For simplicity we shall assume that the index values are chosen such that decision interval δ_q is assigned index value q, i.e. $I = \{1, \ldots, M\}$. Hence $\hat{x}_q$ refers to the quantized value associated with decision interval δ_q. Furthermore since the dynamic range covers the entire range $\mathcal{X}$ and the range is bounded, we assume that the partition is made such that $b_0 = X_{min}$ and $b_M = X_{max}$.

The goal is to map each sample x to one of the quantized values in $\hat{\mathcal{X}}$. I.e. the goal is to quantize each sample x such that:

$$\hat{x}_q = \texttt{Quantize}\,(x) \tag{2.4}$$

for some $\hat{x}_q \in \hat{\mathcal{X}}$. The quantization procedure in (2.4) is performed in two steps; a *forward quantization* step and a *backward quantization step.*

Definition 2.1 — Scalar Forward Quantization.
Let $x \in \mathcal{X}$ be a sample from a signal. The forward quantization is a mapping $Q : \mathcal{X} \to \mathcal{I}$ such that:

$$Q(x) = q \Leftrightarrow x \in \delta_q \tag{2.5}$$

where $\mathcal{I}$ is the index set, q is an index value and δ_q is the corresponding decision interval. [59, p. 21]

The forward quantization constitutes the quantization procedure of the encoding part in Figure 2.2 and it is the index value q that is converted to a bitstream and transmitted over the channel [59, p. 20]. Upon arrival at the decoder, a sample is reconstructed using backward quantization and the quantized values.

Definition 2.2 — Scalar Backward quantization.
Let $q \in \mathcal{I}$ denote the index value of a decision interval. The backward quantization is a mapping $Q^{-1} : \mathcal{I} \to \hat{\mathcal{X}}$ such that:

$$Q^{-1}(q) = \hat{x}_q \quad , \quad \hat{x}_q \in \hat{\mathcal{X}} \tag{2.6}$$

where $\mathcal{I}$ is the index set and $\hat{\mathcal{X}}$ is the set of quantized values. [59, p. 21]

Thus the backward quantization procedure constitutes the inverse quantization in the decoding part. We previously argued that the quantization process enables to compress a signal using fewer bits and that such process is by nature a lossy compression process. This is evident when considering Definitions 2.1-2.2 since the quantization procedure is a mapping from the set $\mathcal{X}$ with possible infinite amount of values to the finite set $\hat{\mathcal{X}}$ of M values. For a quantizer with M quantized values and where each index value is represented using a fixed amount of bits as considered in this project, the bit rate R in bits per sample is given as [59, p. 28]:

$$R = \lceil \log_2(M) \rceil \tag{2.7}$$

where $\lceil \cdot \rceil$ denotes the round to ceiling function.
In this project where such fixed bit rates are considered, we will refer to a quantizer with M decision intervals chosen such that $\lceil \log_2(M) \rceil = b$ as a b-bit quantizer.

By quantizing each sample using the above definitions and equations, we obtain a reconstructed signal. Since the quantization process is lossy, a distortion in terms of a *quantization error* is obtained. The quantization error can be quantified in different ways using different distortion measures. A commonly used measure is the

Mean Squared Quantization Error (MSQE) given as [59, p. 22]:

$$\sigma_e^2 = \int_{-\infty}^{\infty} \left(\hat{x}(x) - x\right)^2 p(x)dx \tag{2.8a}$$

$$= \sum_{q=1}^{M} \int_{\delta_q} \left(\hat{x}(x) - x\right)^2 p(x)dx \tag{2.8b}$$

$$= \sum_{q=1}^{M} \int_{\delta_q} \left(\hat{x}_q - x\right)^2 p(x)dx \tag{2.8c}$$

where we write $\hat{x}\,(x)$ for the quantized value of sample x to emphasize that the obtained quantized value is dependent on the input sample. Equation (2.8b) follows from the range $\mathcal{X}$ being partitioned into decision intervals and (2.8c) follows from $\hat{x}(x)$ being constant within an interval.

From (2.8c) it can be seen that the quantization error depends on both the quantized values and the probability density function. For a given probability density function p, the quantization error is minimized when the term $\left(\hat{x}_q - x\right)^2$ is minimized. This however implies a large amount of decision intervals M and from (2.7) it can be seen that this results in a higher bit rate. How the quantization can be implemented and how the quantization error is affected by the choice of decision intervals, their lengths and the corresponding quantization values is described further below.

Uniform Quantization

If the quantizer in (2.4) consists of decisions intervals with the same length, i.e. $\Delta_1 = \Delta_2 = \ldots = \Delta_M$, the quantizer is referred to as a *uniform quantizer* [59, p. 24]. Since the decision intervals are all of the same length, they can be described by a constant Δ which is referred to as the quantization *step size*. Considering once again the finite range $\mathcal{X} = [X_{min}, X_{max}]$, the step size for a given M number of decision intervals is given as [59, p. 25]:

$$\Delta = \frac{X_{max} - X_{min}}{M} \tag{2.9}$$

The decision intervals are then given by:

$$\delta_q = [X_{min} + \Delta \cdot (q-1), X_{min} + \Delta \cdot q) \qquad , \qquad q = 1, \ldots, M \tag{2.10}$$

As previously described, both the probability density function and chosen quantization values affect the MSQE. If we assume that X follows a uniform distribution it can be shown, that the MSQE for a uniform quantizer is minimized by letting the quantized values correspond to the mean value of each decision interval [7, p. 23]. Thus let the quantized values be given as:

$$\hat{x}_q = X_{min} + \Delta \cdot q - 0.5\Delta \qquad , \qquad q = 1, \ldots, M \tag{2.11}$$

Using (2.8c) it follows that the MSQE for a uniformly distributed signal quantized with a uniform quantizer is given as:

$$\sigma_e^2 = \sum_{q=1}^{M} \int_{X_{min}+\Delta\cdot(q-1)}^{X_{min}+\Delta\cdot q} (\hat{x}_q - x)^2 \, p(x) dx \tag{2.12}$$

$$= \sum_{q=1}^{M} \int_{X_{min}+\Delta\cdot(q-1)}^{X_{min}+\Delta\cdot q} (X_{min} + \Delta\cdot q - 0.5\Delta - x)^2 \, p(x) dx \tag{2.13}$$

$$= \sum_{q=1}^{M} \int_{-0.5\cdot\Delta}^{0.5\cdot\Delta} y^2 p\left(X_{min} + \Delta\cdot q - 0.5\Delta - y\right) dy \tag{2.14}$$

$$= \sum_{q=1}^{M} \int_{-0.5\cdot\Delta}^{0.5\cdot\Delta} x^2 p\left(X_{min} + \Delta\cdot q - 0.5\Delta - x\right) dx \tag{2.15}$$

$$= \sum_{q=1}^{M} \int_{-0.5\cdot\Delta}^{0.5\cdot\Delta} x^2 p\left(\hat{x}_q - x\right) dx \tag{2.16}$$

where (2.12) follows from using (2.10), (2.13) follows from using (2.11), (2.14) follows from integration by substitution with $y = X_{min} + \Delta\cdot q - 0.5\Delta - x$, (2.15) follows from change of variable name and finally (2.16) follows from using (2.11) once again.

For a uniform distribution on the interval $[X_{min}, X_{max}]$ the probability density function is given as:

$$p(x) = \frac{1}{X_{max} - X_{min}} \tag{2.17}$$

The MSQE in (2.16) can thus be reduced to:

$$\sigma_e^2 = \sum_{i=1}^{M} \int_{-0.5\cdot\Delta}^{0.5\cdot\Delta} x^2 \frac{1}{X_{max} - X_{min}} dx \tag{2.18}$$

$$= \frac{1}{X_{max} - X_{min}} \sum_{i=1}^{M} \frac{\Delta^3}{12} \tag{2.19}$$

$$= \frac{M}{X_{max} - X_{min}} \sum_{i=1}^{M} \frac{\Delta^3}{12} \tag{2.20}$$

$$= \frac{\Delta^2}{12} \tag{2.21}$$

where (2.21) follows from using (2.9). Thus for a uniform quantizer, decreasing the step size by a factor of two reduces the quantization error by a factor of four.

Two common types of the uniform quantizer, are the *midrise* and *midtread* quantizers [59, pp. 25-27] shown in Example 2.1.

Example 2.1 — Midrise and Midtread Uniform Quantizers.
The forward and backward quantization steps for the midtread quantizer can be implemented using the following:

$$q_{tread} = \left\lfloor \frac{x}{\Delta} + 0.5 \right\rfloor \tag{2.22}$$

$$\hat{x}_{q_{tread}} = \Delta \cdot q_{tread} \tag{2.23}$$

The steps for the midrise quantizer can obtained by:

$$q_{rise} = \left\lfloor \frac{x}{\Delta} \right\rfloor \tag{2.24}$$

$$\hat{x}_{q_{rise}} = \Delta \cdot (q_{rise} + 0.5) \tag{2.25}$$

where $\lfloor \cdot \rfloor$ denotes the rounding to floor function.

Using a dynamic range of $[-2, 2]$ and a step size of $\Delta = 0.5$, the relationship between the input value and the quantized value for the midtread and midrise quantizers above are shown in Figure 2.4 and Figure 2.5, respectively.

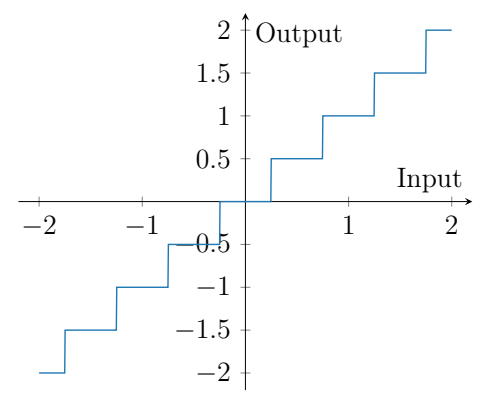


Figure 2.4: Example of midtread uniform quantizer.

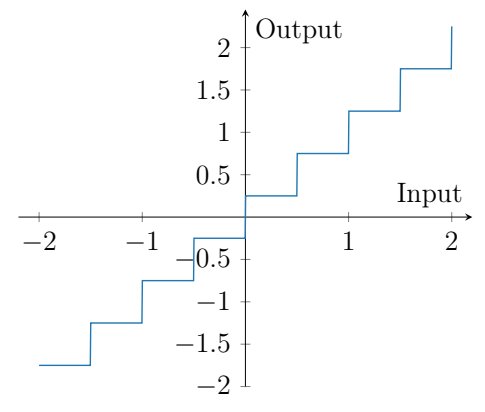


Figure 2.5: Example of midrise uniform quantizer.

The main difference between the two quantizers in Example 2.1 is how the value of zero is quantized. The midtread quantization scheme can be beneficial for signals where a value of zero is necessary, e.g. for speech signals with silent parts [59, p. 25].

There exists a wide range of different quantization schemes beside the uniform quantizer [59, Sec. 2.3.4-2.4.2]. No matter the quantization scheme used, it is worth mentioning that if the dynamic range of the quantizer is not able to cover the entire range $\mathcal{X}$ of the input signal, an additional quantization error is introduced [59, pp. 28-29]. This is for instance the case when $\mathcal{X}$ is unbounded. The MSQE in (2.8a) only accounts for the errors within the dynamic range of the quantizer. This error is in this context also referred to as the *granular error*. Quantization of signal samples falling outside the dynamic range provides an error referred to as the *overload error*. The total quantization error is thus a superposition of the granular error and overload error. One could argue that a solution is to simply increase or extend the dynamic range such that the overload error is minimized. However for a fixed M a large dynamic range implies large decision intervals which in turn implies a large granular error. Thus there exists a trade-off between the granular and overload error.

The optimal quantized values for a uniform quantizer and uniformly distributed signal was stated to be the mean values of the decision intervals. For non-uniform quantizers and non-uniformly distributed signals, the placement of the decision intervals and the corresponding quantized values is in general not entirely obvious. In general an iterative procedure for finding the decision intervals and quantized values is employed [59, pp. 35-39].

Last but not least it is worth mentioning that even though the scalar quantization and MSQE was presented for continuous random variables, the definitions and results are still valid for discrete random variables by replacing integration with summation and the probability density function with a probability mass function [59, p. 24].

2.2.2 Vector Quantization

In the previous section we described scalar quantization where each sample of a signal was quantized separately. If the quantization is done jointly for a collection of samples, i.e. if the quantization is done in higher dimensions, it is referred to as *vector quantization (VQ)* [59, p. 43].
From the previous section it was seen that the MSQE depended on both the probability density function and decision boundaries. A limitation of the scalar quantizer is that one is forced to use intervals, i.e. rectangular regions. By considering the signal in a higher dimensional space, one is not restricted to consider only rectangular regions. Thus the regions can be designed to match the distribution more properly, eventually resulting in a lower bit rate. This is the idea behind vector quantization [59, pp. 44-46].

In the following we will consider a signal as a continuous stochastic vector $\boldsymbol{X}$ with realizations elements $x \in \mathcal{X}$ such that $\boldsymbol{x} \in \mathcal{X}^n$ for realizations $\boldsymbol{x}$ of the vector $\boldsymbol{X}$. We will refer to $\boldsymbol{x}$ as a *sample vector*. For instance $\boldsymbol{x}$ could be a frame of n samples

from a speech signal, i.e. n could be the frame size.

Similarly to scalar quantization, the space $\mathcal{X}^n$ is partitioned into M *decision regions* $\Omega_1, \ldots, \Omega_M$ such that [59, p. 46]:

$$\mathcal{X}^n = \bigcup_{q=1}^{M} \Omega_q \quad , \quad \Omega_q \cap \Omega_{q'} = \emptyset \ , \forall q \neq q' \tag{2.26}$$

An index $q \in \mathcal{I}$ and a quantized vector or *representative vector* $\boldsymbol{x}_q \in \hat{\mathcal{X}}^n$ is assigned to each decision region. Similar to scalar quantization, the sets $\mathcal{I}$ and $\hat{\mathcal{X}}^n$ are referred to as the index set and set of quantized vectors, respectively. For simplicity we shall again assume that the index values are chosen such that index q is assigned to decision region Ω_q and representative vector $\boldsymbol{x}_q$.
Similar to the scalar quantization, a sample vector is quantized to a representative vector in two steps; a *forward vector quantization* step and a *backward vector quantization* step.

Definition 2.3 — Forward Vector Quantization.
Let $\boldsymbol{x} \in \mathcal{X}^n$ be a n-dimensional sample vector. The forward vector quantization is a mapping $Q : \mathcal{X}^n \to \mathcal{I}$ such that:

$$Q(\boldsymbol{x}) = q \Leftrightarrow \boldsymbol{x} \in \Omega_q \tag{2.27}$$

where $\mathcal{I}$ is the index set, q is an index value and Ω_q is the corresponding decision region. [59, p. 47]

In relation to the coding process in Figure 2.2, the forward vector quantization constitutes the quantization procedure in the encoding part. And similarly to the scalar quantization, it is the index value q that is converted to a bitsream and transmitted. Thus even though a multidimensional signal is quantized, it is still a single index value that is transmitted. In the decoding part the representative vector is obtained from the index value through inverse quantization obtained by the use of backward quantization.

Definition 2.4 — Backward Vector Quantization.
Let $q \in \mathcal{I}$ be an index value of a decision interval. The backward vector quantization is a mapping $Q^{-1} : \mathcal{I} \to \hat{\mathcal{X}}^n$ such that:

$$Q^{-1}(q) = \boldsymbol{x}_q \quad , \quad \boldsymbol{x}_q \in \hat{\mathcal{X}}^n \tag{2.28}$$

where $\mathcal{I}$ is the index set and $\hat{\mathcal{X}}^n$ is the set of quantized vectors. [59, p. 47]

The major difference between Definitions 2.3-2.4 and Definitions 2.1-2.2 is that the vector quantizer is performed on multidimensional inputs and hence the decision intervals are defined by multidimensional regions rather than intervals. Letting $n = 1$ it seen that the above definitions correspond to the definitions in the scalar case.

Similar to scalar quantization, the objective when designing a vector quantizer is to find the optimal decision regions and quantized vectors w.r.t. some distortion

measure. The decision regions and set of quantized vectors is also referred to as the *VQ codebook* and M is the codebook size [59, p. 47]. Thus the objective is to find the optimal VQ codebook such that the average distortion for some distortion measure is minimized. In the previous section when considering scalar quantization, we argued that the choice of decision intervals and quantized values is not necessarily obvious. Generalizing the quantization to higher dimensions does not make this task more obvious. Often a generalization of the iterative procedure used for scalar quantization is employed for finding the optimal codebook [59, pp. 48-50].

2.3 Performance and Quality Assessment

In the previous sections we stated that the overall goal for a coding process is to compress the signal using fewer bits while at the same time maintaining a degree of quality. In Section 2.2 the quality of a quantizer was quantified through the quantization error and a associated distortion measure. This section is concerned with the overall quality of the coding process, i.e. the quality of the codec when considering the reconstructed signals obtained at the end of a decoder. Since the coding process consists of several steps, the overall quality of the reconstructed signal is affected by every step in the coding process and not only the quantizer [59, p. 16]. For instance if the analysis filter bank is not able to obtain or detect the correct and relevant features, one is not able to reconstruct the signal properly - even if the quantization error is significantly small.

When considering speech signals it is common to asses the quality of the reconstructed speech in terms of how it is perceived by the human ear. A commonly used measure for perceptual quality of speech signals is the *Mean Opinion Score (MOS)*, where listeners are asked to rate the quality of reconstructed speech signals from a speech codec on some scale. The mean value of their ratings is a subjective measure of the perceptual quality of a speech codec. However such listening tests are extensive to perform, and often a objective measure is employed. [59, pp. 251-252]

In this project we will use the International Telecommunication Union (ITU) recommended *Perceptual Evaluation of Speech Quality (PESQ)* algorithm [1]. The PESQ algorithm is commonly used in various performance evaluation tasks regarding speech such as speech codecs and speech enhancement. The algorithm is an objective measure of speech quality, where the reconstructed speech signal obtained from a speech codec is compared to the original speech source signal. The output of the algorithm is a MOS-like score between -0.5 and 4.5, where a score of 4.5 indicates that the reconstructed signal is not perceptually different from the original signal. Thus a higher score indicates a higher speech quality.

It is worth noting that there exists different versions of the PESQ algorithm. In this project we will use version 1.2 - 2$^{\text{nd}}$ of August 2002 of the ITU-T Recommendation P.862 PESQ algorithm.

2.4 The BroadVoice32 (BV32) Codec

Apart from assessing the speech quality we will in this project also compare the performance of the developed speech codec to an existing codec as a baseline. The comparison will be made in terms of bit rate and PESQ scores.

The performance of our developed speech codec will be compared to the *BroadVoice32 (BV32)* [9] speech codec. The BV32 codec is a ITU-T recommended (recommendation J.361) standard and a standard codec in PacketCable 2.0 and American National Standards Institute, Society of Cable Telecommunications Engineers (ANSI/SCTE). The algorithm encodes speech sampled at a sampling frequency of 16 kHz using a frame size of 5 ms and various processing steps such as vector quantization. The codec outperforms several other ITU-T recommended speech codecs in various aspects such as PESQ and MOS scores on a variety of languages. The codec uses a bit rate of 2 bits per sample, which is a moderate bit rate compared to other codec [9].

3. Deep Neural Networks (DNNs)

The purpose of this chapter is to introduce the concept of Deep Neural Networks (DNNs) and describe their elements. In Section 3.1 the general concept of neural networks and its parameters are described. In Section 3.2 the concept is extended to so called convolutional neural networks, while Section 3.3 is concerned with a special type of neural networks relevant for speech coding. In Section 3.4 some essential functions used in the neural networks of this project are described. Section 3.5 describes how the parameters of a neural network are found.

3.1 General Concept

A neural network can essentially be seen as an universal approximating function [22, p. 167], which is found by solving an optimization problem based on some data examples. For simplicity we will first consider the case of classification in order to describe the concept of neural networks. Later in Section 3.3, we will present neural networks in the context of speech coding.

Consider a K-class classification problem with $K \in \mathbb{N}_+$. Let $\boldsymbol{x} \in \mathbb{R}^N$ be a data example. Let further $\boldsymbol{y} \in \mathbb{R}^K$ be the corresponding class label with elements such that $y_k = 1$ and all other elements equal to zero if $\boldsymbol{x}$ belongs to class k. For instance $\boldsymbol{x}$ could be the pixel values in a picture of either a cat or a dog. For this binary classification problem the corresponding class label could be chosen such that $\boldsymbol{y} = [1, 0]^T$ if $\boldsymbol{x}$ represents a picture of a cat and $\boldsymbol{y} = [0, 1]^T$ if $\boldsymbol{x}$ is a picture of a dog. Assuming that there is an underlying function f such that $f(\boldsymbol{x}) = \boldsymbol{y}$, the goal of a neural network is to obtain an approximation of - or *learn* - this function. Let $\hat{f}$ denote the approximation. the goal is to find $\hat{f}$ such that:

$$\hat{f}(\boldsymbol{x}) \approx \boldsymbol{y} \tag{3.1}$$

I.e. if (3.1) holds, the network is able to classify the input $\boldsymbol{x}$ correctly. Of course, the network should not only classify a single example correctly, but rather examples in general. For now whenever we refer to the relation in (3.1) it suffices to think of the relation being fulfilled for a collection of examples $\{\boldsymbol{x}_i\}_{i=1}^M$ and class labels $\{\boldsymbol{y}_i\}_{i=1}^M$.

While there are different types of neural networks, we will restrict our focus on so called *feedforward* neural networks.

As the name suggests, data can only be propagated forward in a feedforward neural network. The function f is approximated by a series of functional transformations. Let $\boldsymbol{x} \in \mathbb{R}^{N_0}$ be the input to the network. In the context of neural networks, each element of the input vector is also referred to as a neuron or *node* and the input is referred to as the *input layer*. Thus a layer consists of nodes. A new layer $\boldsymbol{h} \in \mathbb{R}^{N_1}$ is obtained by [6, p. 227]:

$$\boldsymbol{a} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \tag{3.2a}$$

$$\boldsymbol{h} = \phi(\boldsymbol{a}) \tag{3.2b}$$

where $\boldsymbol{b} \in \mathbb{R}^{N_1}$ and $\boldsymbol{W} \in \mathbb{R}^{N_1 \times N_0}$ are parameters referred to as the bias vector and weight matrix, respectively. The weight w_{ij} is the weight associated with node j in the input layer and node i in the succeeding layer. The function $\phi : \mathbb{R} \to \mathbb{R}$ is (often) a non-linear function referred to as an *activation function*, which acts on each node in a layer. Thus elements a of $\boldsymbol{a}$ are referred to as *activations*.

Remark Depending on the literature (see e.g. [22, Ch. 6] and [6, Sec. 5.1]), the term activation may also refer to the elements of $\boldsymbol{h}$ since the elements are obtained or "activated" by the activation function. Nevertheless we will refer to the elements a as activations and the elements h of $\boldsymbol{h}$ as *outputs* of the layer $\boldsymbol{h}$.

Equations (3.2a)-(3.2b) can be repeated a desired amount of times and the final layer is referred to as the *output layer*. I.e. the layers are given by:

$$\boldsymbol{a}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)} \tag{3.3a}$$

$$\boldsymbol{h}^{(l)} = \phi^{(l)}\left(\boldsymbol{a}^{(l)}\right) \tag{3.3b}$$

for layers $l = 1, \ldots, L$. We define $\boldsymbol{h}^{(0)} := \boldsymbol{x}$ and $\boldsymbol{h}^{L} := \hat{\boldsymbol{y}}$ to be the input and output of a network, respectively. We will furthermore let N_x and N_y denote the dimensions of the input and output, respectively. I.e. $\boldsymbol{x} \in \mathbb{R}^{N_x}$ and $\boldsymbol{y} \in \mathbb{R}^{N_y}$. The superscripts in (3.3a)-(3.3b) emphasize that different weights and biases are associated with each layer, and that the activation functions need no to be the same. Different activation functions are discussed in Section 3.4.

A layer that is neither an input nor output layer is referred to as a *hidden layer*. The value of L is referred to as the *depth* of the network. Hence deep neural networks and deep learning refer to the usage of several layers [22, p. 168]. We will refer to a layer of depth L as a L-layer network. An example of 4-layer network can be seen in Figure 3.1.

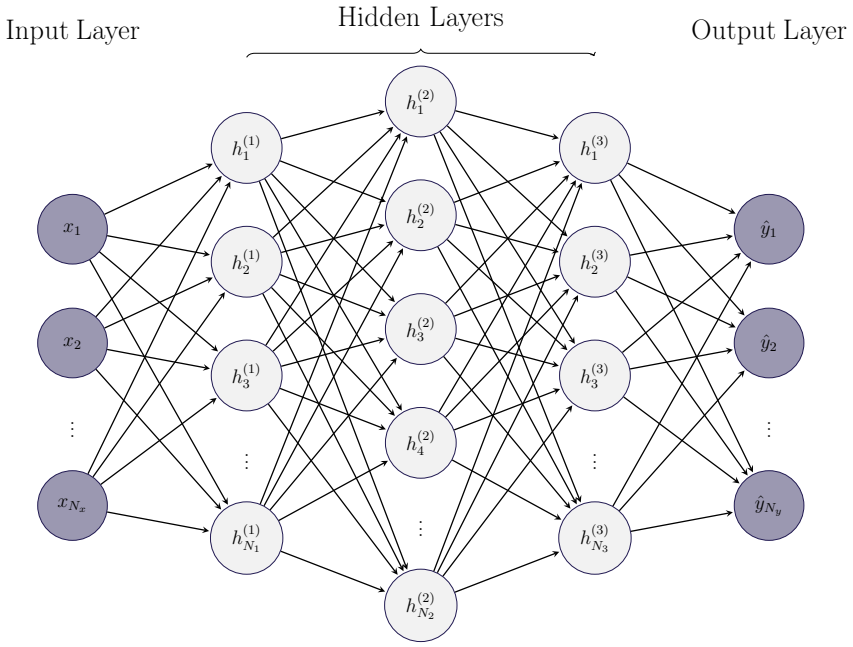


Figure 3.1: General 4-layer feedforward neural network. Weights are indicated by arrows. Biases are omitted for simplicity.

The network in Figure 3.1 is also referred to as a *fully connected* feedforward neural network, since each node in a layer is connected to every node in the succeeding layer.

We began this section by stating that the objective of a neural network is to approximate some underlying function, relating the input $\boldsymbol{x}$ to a desired output $\boldsymbol{y}$. In a feedforward neural network, the underlying function f is approximated by:

$$\hat{f}(\boldsymbol{x}) = \phi^{(L)}\left(\boldsymbol{W}^{(L)}\left(\cdots\left(\phi^{(1)}\left(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right)\right)\right) + \boldsymbol{b}^{(L)}\right) \tag{3.4}$$

The objective of a neural network thus reduces to finding the weights, biases - and possibly parameters for the activation functions - such that $\hat{f}(\boldsymbol{x}) = \hat{\boldsymbol{y}} \approx \boldsymbol{y}$. The process of finding these parameters is a iterative process referred to as *training* and will be discussed more comprehensively in Section 3.5.

It is not entirely obvious that a feedforward neural network is able to approximate an arbitrary underlying function f by (3.4). However the *Universal Approximation Theorem* [13][26] states, that a feedforward neural network with at least one hidden layer can approximate a broad class of functions satisfying some conditions. The conditions and the theorem itself is beyond the scope of this project. From a practical point of view it suffices to say that according to the theorem, a neural network is capable of approximating any underlying function between our data input $\boldsymbol{x}$ and desired output $\boldsymbol{y}$ [22, pp. 197-198]. However, the theorem does not provide the amount of nodes, layers and choice of activation functions needed in order for the approximation to be sufficient. A difficulty is thus to find a proper network architecture.

3.2 1D Convolution

In the previous section we explained neural networks in their basic form, i.e. consisting of feedforward fully connected layers. Another widely used layer is the *convolutional* layer [28][27][11][33][35].
A neural network consisting of at least one convolutional layer is referred to as a *Convolutional Neural Network (CNN)* [22, p. 330]. In this project we will be using 1-dimensional convolutional layers, and we will thus describe convolutional layers in this context.

Consider again the input $\boldsymbol{x} \in \mathbb{R}^{N_x}$ to a neural network. Instead of performing the matrix multiplication in (3.2a), the activation nodes in a convolutional layer are obtained by:

$$a_i = C_K\left(\boldsymbol{x}, \boldsymbol{w}\right)_i \qquad , \qquad i = 1, \ldots, N_1 \tag{3.5}$$

where $C_K\left(\cdot, \cdot\right)$ is the convolutional operator defined by [6, p. 333]:

$$C_K\left(\boldsymbol{x}, \boldsymbol{w}\right)_i := \sum_{k=1}^{K} x_{i+k-1} w_k \tag{3.6}$$

for input $\boldsymbol{x}$ and where $\boldsymbol{w} \in \mathbb{R}^K$ is a weight vector, which in the context of convolutional layers is referred to as a *kernel*.

Remark From a mathematical point of view,(3.6) is not a convolution but a *cross-correlation* [22, p. 333]. For (3.6) to be a convolution, the input should be reversed, i.e. the term x_{t+k-1} should be replaced by x_{t-k+1} (see [40]). Nevertheless we will use the term convolution.

A kernel of smaller dimension than the input is used, and typically $K \ll N_x$ [22, p. 350]. The dimension N_1 of the activation $\boldsymbol{a}$ is dependent on the nature of the convolution in (3.6). If the convolution in (3.6) is only performed where the indexation is valid in the summation, then:

$$N_1 = N_x - K + 1 \tag{3.7}$$

for an input of dimension N_x and a kernel of size K. Unless $K = 1$, dimensionality reduction is inevitable. Additionally one can use different zero-padding schemes to obtain different output dimensions of the convolution, e.g. one can zero-pad the input such that $N_1 = N_x$.

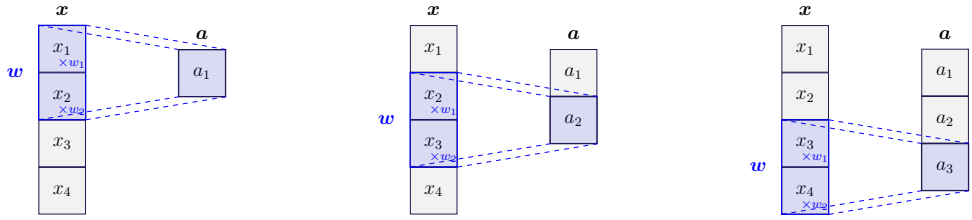
Another way to control the dimension of the output from the convolution is by the use of *strides*. In (3.6) the convolution can be seen as moving filter or kernel, that moves one step on the input to obtain each element in $\boldsymbol{a}$. Instead one can move the kernel s-steps. A 1-dimensional convolution with stride $s \in \mathbb{N}_+$ is given by [22, p. 348]:

$$C_K\left(\boldsymbol{x}, \boldsymbol{w}, s\right)_i = \sum_{k=1}^{K} x_{s(i-1)+k} w_k \tag{3.8}$$

By using strides of size s and without using zero-padding, the output dimension is given by:

$$N_1 = \frac{N_x - K + s}{s} \tag{3.9}$$

An example of the convolution in (3.8) with strides of 1 and 2 can be seen Figure 3.2 and Figure 3.3, respectively.

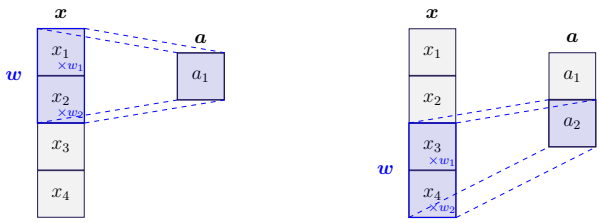


(a) The elements x_1 and x_2 are convolved with the kernel $\boldsymbol{w} = [w_1, w_2]^T$ to obtain node a_1.

(b) The elements x_2 and x_3 are convolved with the kernel $\boldsymbol{w} = [w_1, w_2]^T$ to obtain node a_2.

(c) The elements x_3 and x_4 are convolved with the kernel $\boldsymbol{w} = [w_1, w_2]^T$ to obtain node a_3.

Figure 3.2: An example of the convolution in (3.8) without zero-padding and with stride $s = 1$ and $K = 2$ for an input $\boldsymbol{x} = [x_1, x_2, x_3, x_4]^T$ and kernel $\boldsymbol{w} = [w_1, w_2]^T$. In each figure, the kernel is moved 1 step to obtain an activation. Since there is no zero-padding, a layer of size $N_1 = 3$ is obtained.



(a) The elements x_1 and x_2 are convolved with the kernel $\boldsymbol{w} = [w_1, w_2]^T$ to obtain node a_1.

(b) The elements x_3 and x_4 are convolved with the kernel $\boldsymbol{w} = [w_1, w_2]^T$ to obtain node a_2.

Figure 3.3: An example of the convolution in (3.8) without zero-padding and with stride $s = 2$ and $K = 2$ for an input $\boldsymbol{x} = [x_1, x_2, x_3, x_4]^T$ and kernel $\boldsymbol{w} = [w_1, w_2]^T$. In each figure, the kernel is moved 2 steps to obtain an activation. Since there is no zero-padding, a layer of size $N_1 = 2$ is obtained.

The convolution can be seen as some sort of feature extraction [22, pp. 337-340], where a feature vector $\boldsymbol{a}$ is extracted from $\boldsymbol{x}$ by the use of the kernel $\boldsymbol{w}$. Thus one feature vector is obtained by one kernel. One can extract several different feature vectors from the input by using different kernels and obtain a *feature map* [22, Fp. 332].

Let $\boldsymbol{W} \in \mathbb{R}^{K \times K_{N_1}}$ be a kernel matrix consisting of K_{N_1} kernels each of size K. Let $w_k^{(j)}$ denote the element in row k and column j of $\boldsymbol{W}$, i.e. the kth element of kernel j. Element (i, j) of the feature map in a convolutional layer with stride s is obtained by:

$$a_{i,j} = C_K\left(\boldsymbol{x}, \boldsymbol{W}, s\right)_{i,j} \qquad , \qquad i = 1, \ldots, N_1, j = 1, \ldots, K_{N_1} \tag{3.10}$$

where:

$$C_K\left(\boldsymbol{x}, \boldsymbol{W}, s\right)_{i,j} = \sum_{k=1}^{K} x_{s(i-1)+k} w_k^{(j)} \tag{3.11}$$

As with the feedforward fully connected layers in (3.2a)-(3.2b), an activation function is applied to each element in the feature map:

$$h_{i,j} = \phi\left(a_{i,j}\right) \tag{3.12}$$

Let $\boldsymbol{H}^{(1)} \in \mathbb{R}^{N_1 \times K_{N_1}}$ denote the output from a convolutional layer with elements $h_{i,j}^{(1)}$. A new convolutional layer $\boldsymbol{H}^{(2)} \in \mathbb{R}^{N_2 \times K_{N_2}}$ can be obtained by:

$$a_{i,j}^{(2)} = C_{K^{(2)}}\left(\boldsymbol{H}^{(1)}, \boldsymbol{W}^{(2)}, s^{(2)}\right)_{i,j} \tag{3.13a}$$

$$h_{i,j}^{(2)} = \phi^{(2)}\left(a_{i,j}^{(2)}\right) \tag{3.13b}$$

for $i = 1, \ldots, N_2, j = 1, \ldots, K_{N_2}$ and where [22, p. 348]:

$$C_{K^{(2)}}\left(\boldsymbol{H}^{(1)}, \boldsymbol{W}^{(2)}, s^{(2)}\right)_{i,j} = \sum_{\substack{k=1,\ldots,K_{N_2} \\ m=1,\ldots,N_1}} h_{s^{(2)}(i-1)+k,m}^{(1)} w_{k,m}^{(2,j)} \tag{3.14}$$

for kernel matrix $\boldsymbol{W}^{(2)} \in \mathbb{R}^{K^{(2)} \times K_{N_1} \times K_{N_2}}$. The kernel matrix consists of K_{N_2} kernels each of size $K^{(2)} \times K_{N_1}$ such that $w_{k,m}^{(2,j)}$ is element (k, m) of kernel j. Even though the kernel in this case is 2-dimensional, we will still refer to $K^{(2)}$ as the kernel size, since the second dimension is implicitly give by the previous layer. Similar to the fully connected layers, we use the superscripts to indicate that the kernel matrix is different for each layer and that the kernel size, stride and activation functions need not to be the same.

Even though $\boldsymbol{H}^{(1)}$ is 2-dimensional, (3.14) is still referred to as a 1-dimensional convolution since the kernel spans the entire second dimension and is only moved across the first dimension. An example of the convolution in (3.14) with a stride of 2 can be seen in Figure 3.4.

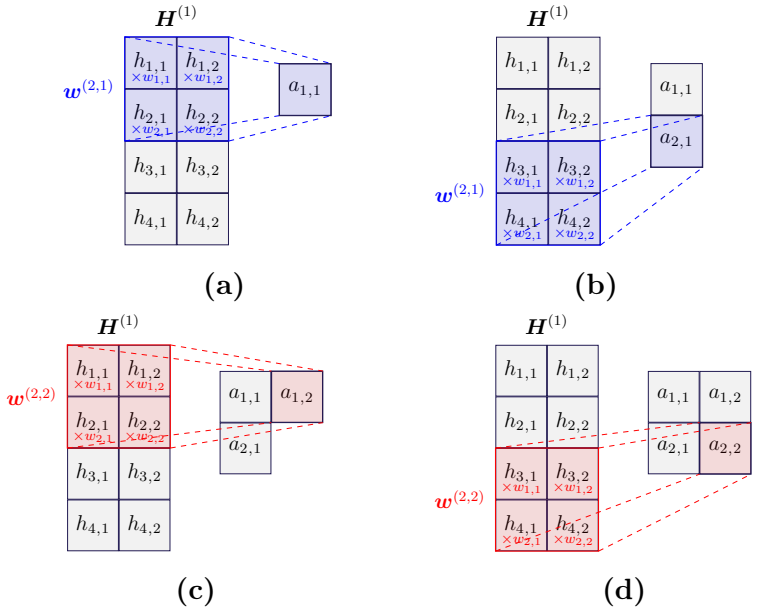


Figure 3.4: An example of the convolution in (3.14) without zero-padding and with stride $s = 2$ and $K^{(2)} = 2$. In (a)-(b) $\boldsymbol{H}^{(1)}$ is first convolved with kernel $w^{(2,1)}$ to obtain $a_{1,1}$ and $a_{2,1}$. In (c)-(d) $\boldsymbol{H}^{(1)}$ is then convolved with kernel $w^{(2,2)}$ to obtain $a_{1,2}$ and $a_{2,2}$.

As with the fully connected layers, one can use (3.13a)-(3.13b) a desired amount of times to obtain L layers. I.e. layer l is given by:

$$a_{i,j}^{(l)} = C_{K^{(l)}}\left(\boldsymbol{H}^{(l-1)}, \boldsymbol{W}^{(l)}, s^{(l)}\right)_{i,j} \tag{3.15a}$$

$$h_{i,j}^{(l)} = \phi^{(l)}\left(a_{i,j}^{(l)}\right) \tag{3.15b}$$

with:

$$C_{K^{(l)}}\left(\boldsymbol{H}^{(l-1)}, \boldsymbol{W}^{(l)}, s^{(l)}\right)_{i,j} = \sum_{k,m} h_{s^{(l)}(i-1)+k,m}^{(l-1)} w_{k,m}^{(l,j)} \tag{3.16}$$

The objective of a DNN does not change when using convolutional layers. For a CNN the objective is to find the kernels rather than the weights. Furthermore a CNN can be regarded as a special case of a feedforward DNN. To see this, the convolution shown in Figure 3.2 is shown as a special case feedforward DNN in Figure 3.5.

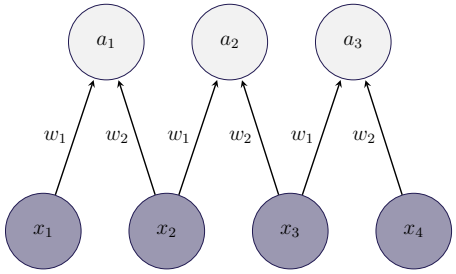


Figure 3.5: The convolution from Figure 3.2 as a special case of feedforward neural networks.

From Figure 3.5 it can be seen that the weights are shared across the nodes, and that the connectivity is sparse in the sense that not all nodes from the input layer are connected to all nodes in succeeding layer. The weight sharing and sparse connectivity properties are some advantages and motivations behind CNNs [22, p. 335-338]. This also reduces the storage and computation complexity of a network, since the amount of weights needed to be stored is dependent on the kernel sizes and amount of kernels used and not the dimension of the layers. This further enables CNNs to be able to process inputs of variable size, e.g. images of different resolutions [22, Sec. 9.7].

The parameter sharing means that the networks learn one type of feature extraction for every kernel, rather than learning one type of feature extraction per output node. Thus in a convolutional layer a kernel is not tied to a specific node as the weights are in a fully connected layer. This means that shifting an input to a fully connected layer by m samples produces an entirely different activation that has not relation to the activation obtained by the non-shifted input. However for a convolutional layer there is a relation between the activation obtained by a shifted input and the activation for the corresponding non-shifted input.

Let T_m denote the function that translates an input by m samples. For a convolution it follows that:

$$C_K\left(T_m\left(\boldsymbol{x}\right), \boldsymbol{w}\right)_i = \sum_k T_m\left(x_{s(i-1)+k}\right) w_k \tag{3.17a}$$

$$= \sum_k x_{s(i+m-1)+k} w_k \tag{3.17b}$$

$$= C_K\left(\boldsymbol{x}, \boldsymbol{w}\right)_{i+m} \tag{3.17c}$$

$$= T_m\left(C_K\left(\boldsymbol{x}, \boldsymbol{w}\right)_i\right) \tag{3.17d}$$

for a kernel $\boldsymbol{w} \in \mathbb{R}^K$ and stride s. Thus shifting the input by m samples results in an output also shifted by m samples. This property is known as *equivariance to transition* [22, p. 338-339]. This property is desirable for various tasks. For instance if we consider classifications tasks, then a network should be able to classify a picture of a cat or dog regardless if they are translated by some samples. Likewise if we consider a time series as input to the network, as will be done in this project, then a feature present at some samples will still be detected by a convolutional network even though the time series is shifted.

In this project we will use both fully connected and convolutional layers. The idea of using convolutional layers is further inspired by its use in the WaveNet [56] CNN used for synthesizing and generating raw audio signals. Since reconstructing speech by a compressed version can be seen as a sort of synthesis, e.g. by the use of synthesis filter banks, it is the hope that a neural network will benefit from using convolutional layers as done in [55] in its attempt to perform speech coding.

3.3 Autoencoders

In Section 3.1 we presented DNNs in the context of classification, where the goal of a neural network was to obtain an approximating function $\hat{f}$ such that $\hat{f}(\boldsymbol{x}) \approx \boldsymbol{y}$ for input example $\boldsymbol{x}$ and corresponding class label $\boldsymbol{y}$. In this section we will present neural networks in the context of speech coding.

As explained in Chapter 2 a speech coder consists of two parts; an encoding part and a decoding part. Furthermore it was explained that the objective of speech coding is to obtain a compressed representation of the speech signal $\boldsymbol{x}$ (encoding part) while maintaining a degree of quality in the reconstructed signal $\hat{\boldsymbol{x}}$ obtained by the decoding part, such that $\boldsymbol{x} \approx \hat{\boldsymbol{x}}$.
Let $\boldsymbol{x} \in \mathbb{R}^{N_x}$ denote the input signal to the autoencoder and let $\hat{\boldsymbol{x}} \in \mathbb{R}^{N_x}$ denote the output of the autoencoder, i.e. the reconstructed signal. Similar to the general speech coding process described in Section 2.1, an autoencoder $\hat{f}$ can be seen as consisting of two DNNs, an encoding DNN $\hat{f}_e$ and a decoding DNN $\hat{f}_d$ such that:

$$\hat{f}(\boldsymbol{x}) = \hat{f}_d\left(\hat{f}_e(\boldsymbol{x})\right) = \hat{f}_d(\boldsymbol{x}_q) = \hat{\boldsymbol{x}} \approx \boldsymbol{x} \tag{3.18}$$

where $\hat{f}_e(\boldsymbol{x}) = \boldsymbol{x}_q \in \mathbb{R}^{N_q}$ is the encoded input. Notice that the dimensions N_x and N_q need not to be the same. An autoencoder with $N_q < N_x$ is referred to as *undercomplete* while an autoencoder with $N_q > N_x$ is referred to as *overcomplete* [22, pp. 505-506].

It may seem comprehensive to utilize neural networks to simply approximate the identity mapping in (3.18). However often and in the context of speech coding, one seeks to obtain features in $\boldsymbol{x}_q$ present in the input $\boldsymbol{x}$. The features should be relevant such that $\boldsymbol{x}$ can be reconstructed from $\boldsymbol{x}_q$ in the decoding part, which is why one requires $\hat{\boldsymbol{x}} \approx \boldsymbol{x}$. Different strategies involving the measurement of similarity between the input and output, guiding the network into obtaining relevant features and prevent the network from simply memorizing the input give rise to different autoencoder structures and types. Often the layer consisting of $\boldsymbol{x}_q$ is restricted in some sense [22, p. 504]. For a more detailed overview of different autoencoders we refer the reader to [22, Chapter 14] and [54].

In this project we will guide the network to achieve meaningful representations by restricting the layer consisting of $\boldsymbol{x}_q$ by use of quantizers and undercomplete autoencoders. The reader may notice that the use of autoencoders as is done in this project requires inclusion of the transmission channel from Section 2.1.4 in some manner. All these aspects will be described more comprehensively in Chapter 4. We will furthermore guide the network to achieve meaningful representations by using the information bottleneck principle in form of guided training. This will be described in Chapter 6.

3.4 Activation Functions

The role of the activation function $\phi : \mathbb{R} \to \mathbb{R}$ introduced in Sections 3.1-3.2 is to introduce non-linearity in the network, as a network consisting of only linear functions such as the matrix multiplication in (3.3a) will be a linear function itself [22, p. 171]. In such cases it can further be shown that no hidden layers are needed [6, p. 229]. In this section we will present the activation functions used in this project. For notational convenience, we will drop the superscripts of the activation function and activations.

3.4.1 Identity

Even though a neural network consisting of only linear activation functions is depreciated, linear activation functions can still be used in some layers. For instance one can view the input layer of a neural network as being processed by an identity function. In this project we will use the identity functions as an activation function in the output layer, too. The identity function of an activation $a \in \mathbb{R}$ is defined as:

$$\phi_{ID}(a) := a \tag{3.19}$$

The choice of this function is solely based on our desire to obtain a reconstructed version, $\hat{\boldsymbol{x}}$, of the input $\boldsymbol{x}$. Hence using the identity function as an activation function in the output layer impose no restrictions on the values the output layer can take.

For reasons that will become clear in Section 3.5, the derivative of the activation function is needed in order to train neural networks. The derivative of the identity function w.r.t. the activation a is given as:

$$\frac{\partial \phi_{ID}(a)}{\partial a} = \frac{\partial a}{\partial a} = 1 \tag{3.20}$$

3.4.2 Tanh

As mentioned the use of the identity function in the output layer impose no restrictions regarding the range of the output. In this project we will in some cases perform preprocessing of the inputs to a network by normalizing the input to be in the range $[-1, 1]$. In such cases it can be beneficial to also restrict the range of the output of the network to the same range as the input. For this reason we will use the hyperbolic tangent or *tanh* function as an activation function in the output layer for networks with normalized inputs. The tanh function is defined as [6, p. 245]:

$$\phi_{tanh}(a) := \frac{e^a - e^{-a}}{e^a + e^{-a}} \tag{3.21}$$

for activation $a \in \mathbb{R}$. The tanh function is shown in Figure 3.6.

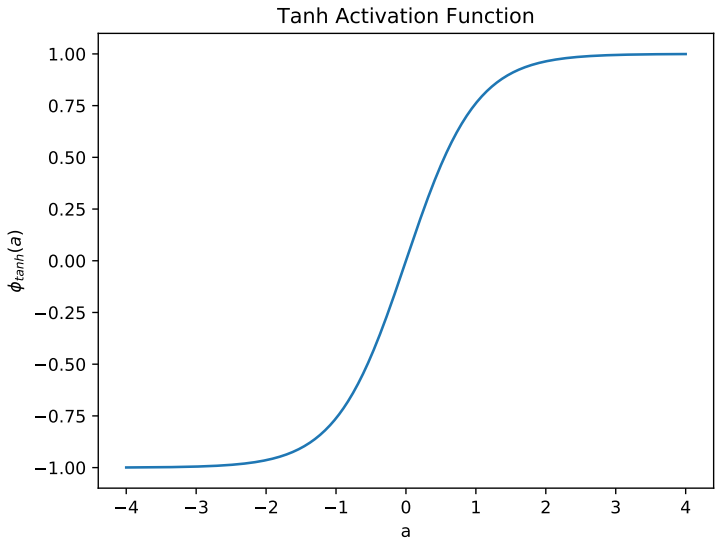


Figure 3.6: The tanh activation function.

The derivative of the tanh activation function w.r.t. to the activation a is given as:

$$\frac{\partial \phi_{tanh}(a)}{\partial a} = \frac{\left(e^a - (-1)\, e^{-a}\right)\left(e^a + e^{-a}\right) - \left(e^a - e^{-a}\right)\left(e^a - e^{-a}\right)}{\left(e^a + e^{-a}\right)^2} \tag{3.22a}$$

$$= \frac{\left(e^a + e^{-a}\right)^2 - \left(e^a - e^{-a}\right)^2}{\left(e^a + e^{-a}\right)^2} \tag{3.22b}$$

$$= 1 - \phi_{tanh}(a)^2 \tag{3.22c}$$

where (3.22a) follows from using the quotient rule for derivatives and (3.22c) follows from (3.21).

3.4.3 Exponential Linear Unit (ELU)

For the hidden layers, we will in this project use the *Exponential Linear Unit (ELU)* as an activation function. Before we present this function, we will first present some similar activation functions.

The most widely used activation function is the *Rectified Linear Unit (ReLU)* [34]. The ReLU function is defined as [22, p. 192]:

$$\phi_{ReLU}(a) := \max(0, a) \tag{3.23}$$

for activation $a \in \mathbb{R}$. The ReLU activation function is similar to the identity function in (3.19) but with outputs equal to zero for inputs $a \leq 0$. Since both the outputs and derivatives are zero for inputs $a \leq 0$, the usage of ReLU can lead to *dead nodes* . As we shall soon see, the derivatives are essential for finding the weights, biases and kernels for a neural network. The term dead nodes refer to nodes that output zero and are never changed due to associated weights never being changed because of the zero-valued gradient [22, p. 192].

To circumvent the problem of dead nodes several generalizations of the ReLU activation function exist. In the *Leaky Rectified Linear Unit (LReLU)* activation function the problem is circumvented by introducing a small value $\beta > 0$. The LReLU is defined as [22, p. 192]:

$$\phi_{LReLU}(a) := \max(\beta a, a) \tag{3.24}$$

The (L)ReLU activation function can be seen as consisting of two piecewise linear functions. In the ELU activation function the problems regarding the gradient of the ReLU function are circumvented by using a non-linear function for inputs $a \leq 0$. The ELU activation function is defined as [14]:

$$\phi_{ELU}(a) := \begin{cases} a & , a > 0 \\ \beta(\exp(a) - 1) & , a \leq 0 \end{cases} \tag{3.25}$$

for activation $a \in \mathbb{R}$ and with $\beta > 0$.

It has been shown that networks with the ELU activation function obtains both faster training and better performances on a broad class of tasks including autoencoding, compared to networks with ReLU or LReLU activation functions [14]. This coincides with our own initial experiments. We will thus be using the ELU activation function in the hidden layers of our autoencoder. The ReLU, LReLU and ELU activation functions are shown in Figure 3.7.

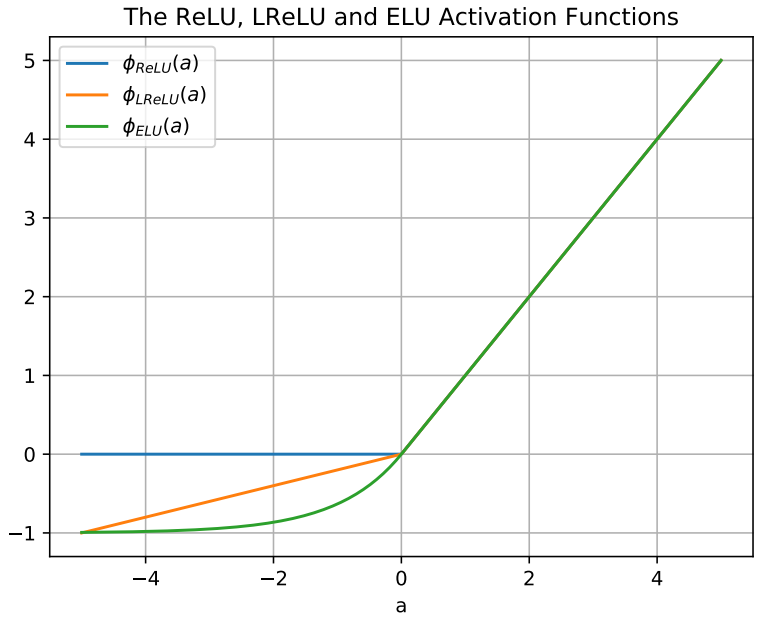


Figure 3.7: The activation functions ReLU, LReLU with $\beta = 0.2$ and ELU with $\beta = 1$.

For $a > 0$ the ELU activation is equal to the identity function. Thus for such activations, the derivative of the ELU activation function is equal to the derivative in (3.20) of the identity function in. For $a \leq 0$ the derivative is given as:

$$\frac{\partial \phi_{ELU}(a)}{\partial a} = \beta \exp(a) \tag{3.26}$$

3.5 Training

In the previous sections we explained neural networks as an universal approximating function $\hat{f}$ of an underlying function f. The approximation is found through M implicit examples of f [22, p. 151]. The set of examples:

$$\mathcal{T} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i), \ldots, (\boldsymbol{x}_M, \boldsymbol{y}_M)\} \quad , \quad f(\boldsymbol{x}_i) = \boldsymbol{y}_i, \forall i \tag{3.27}$$

is referred to as *training set*. In (3.27) $\boldsymbol{x}_i \in \mathbb{R}^{N_x}$ is the i'th data example, i.e. the i'th example input to the network, and $\boldsymbol{y}_i \in \mathbb{R}^{N_y}$ is the corresponding desired output such as a class label in case of classification or the input itself in case of autoencoding. Thus the objective of a neural network is to configure or *learn* the weights, biases and kernels such that:

$$\hat{f}(\boldsymbol{x}_i) \approx \boldsymbol{y}_i \quad , \quad i = 1, \ldots, M \tag{3.28}$$

That is to find the weights, biases and kernels such that the output of the network matches the desired output for all M training examples. As previously explained the process of obtaining the weights, biases and kernels is referred to as *training* and we will in this section describe this process. Since the presented training procedure applies to general problems where one seeks to obtain a desired output for an input, we will use $\boldsymbol{y}$ to denote a desired output even though $\boldsymbol{y} = \boldsymbol{x}$ in an autoencoding setup. In aspects related directly to autoencoders and this project, we may use $\boldsymbol{x}$ to denote the desired output, too. This will be clear from context.

In neural network training, the objective of finding the weights, biases and kernels such that (3.28) is fulfilled is considered as an optimization problem [6, Sec. 5.2]. Let $\mathcal{L}$ denote the function that quantifies the error between the desired output and obtained output. For notational convenience we will for the remainder of this section include the biases and kernels in the weight vector $\boldsymbol{w} \in \mathbb{R}^K$ such that K is the total amount of weight, biases and kernel values. The objective of a neural network can be restated as:

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \mathcal{L}(\mathcal{T}; \boldsymbol{w}) \tag{3.29}$$

where we write $\mathcal{L}$ as a function of $\boldsymbol{w}$ to emphasize that the error or *loss* is dependent on the value of the weights, biases and kernels. The function $\mathcal{L}$ is referred to as the *loss function* and the value obtained by evaluating it on the training set is often referred to as the *training loss*. For notational convenience we will drop the set $\mathcal{T}$ in the argument of $\mathcal{L}$ as it should be clear from context that the loss function is evaluated on the training set.

In Section 3.5.1 we will describe the loss function used in this project, while we in Sections 3.5.2-3.5.3 describe the ideas behind the iterative procedures used for solving the optimization problem in (3.29). In Section 3.5.4 we briefly explain some considerations regarding the training of neural networks and how it differs from standard optimization problems.

3.5.1 Loss Functions

The loss function in (3.29) is typically evaluated as an average across the M training examples [22, p. 275]:

$$\mathcal{L}(\boldsymbol{w}) = \frac{1}{M}\sum_{i=1}^{M}\mathcal{L}_i(\boldsymbol{w}) \tag{3.30}$$

where $\mathcal{L}_i$ is the per examples loss, i.e. the loss between the output from the network for the i'th input example and the corresponding desired i'th output example. In this project we will use the *Mean Squared Error (MSE)* as a loss function.
In Tensorflow the MSE is evaluated node-wise, i.e. the mean is first taken across the number of samples in each example and then averaged across all examples.

Let $\boldsymbol{x}_i \in \mathbb{R}^{N_x}$ denote the i'th input example to a DNN autoencoder and let $\hat{\boldsymbol{x}}_i$ denote the corresponding output. The MSE of the i'th example is given as [22, p. 108]:

$$\mathcal{L}_i(\boldsymbol{w}) = \frac{1}{N_x}\|\boldsymbol{x}_i - \hat{\boldsymbol{x}}_i\|_2^2 \tag{3.31}$$

where $\|\cdot\|_2$ is the ℓ_2-norm.
Thus the MSE loss function as it is used in Tensorflow is given as:

$$\mathcal{L}_{MSE}(\boldsymbol{w}) = \frac{1}{M}\sum_{i=1}^{M}\mathcal{L}_i(\boldsymbol{w}) \tag{3.32a}$$

$$= \frac{1}{N_x M}\sum_{i=1}^{M}\|\boldsymbol{x}_i - \hat{\boldsymbol{x}}_i\|_2^2 \tag{3.32b}$$

From (3.32b) it is clear that the loss function is minimized when $\hat{\boldsymbol{x}}_i = \boldsymbol{x}_i\,, \forall i$. This coincides with our desire to obtain a reconstruction $\hat{\boldsymbol{x}}$ of an input $\boldsymbol{x}$ such that $\hat{\boldsymbol{x}} \approx \boldsymbol{x}$. in our autoencoder.

The MSE is in general used as a loss function for various DNN regression tasks [6, p. 236]. For DNNs involving speech, the MSE is often used in speech separation [37][17][30] and speech enhancement [58][10].

3.5.2 Learning Algorithms

The complex structure of neural networks makes it difficult to solve the optimization problem in (3.29) analytically [22, p. 176]. Instead the optimal set of weights $\boldsymbol{w}^*$ is found iteratively using an optimization algorithm which in the context of neural networks is referred to as a *learning algorithm*. We will not describe the learning algorithms used in this project in detail. However since most learning algorithms are machine learning variants of the *gradient descent* algorithm used for standard optimization problems [22, pp. 176-177], we will briefly present the idea behind gradient descent.

Let $\mathcal{J}$ denote an arbitrary and differentiable objective function and let $\boldsymbol{\theta} \in \mathbb{R}^K$ denote its K parameters. Assume we wish to find the parameters $\boldsymbol{\theta}^*$ such that the

objective function is minimized. I.e. we wish to find $\boldsymbol{\theta}^*$ such that:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}) \tag{3.33}$$

Let $\nabla\mathcal{J}(\boldsymbol{\theta})$ denote the gradient of the objective function w.r.t. its parameters. The gradient points in the direction of greatest increase of the objective function $\mathcal{J}$ and recall that our goal is to minimize the objective function. The idea in gradient descent is thus to iteratively obtain an optimal value of $\boldsymbol{\theta}$ by taking a small step of size $\alpha > 0$ in the opposite direction of the gradient. [8, Ch. 9]

Let $\boldsymbol{\theta}^{(n)}$ denote the value of the parameters at time n. In gradient descent a new value $\boldsymbol{\theta}^{(n+1)}$ of the parameters at time $n+1$ is obtained by [8, p. 466]:

$$\boldsymbol{\theta}^{(n+1)} = \boldsymbol{\theta}^{(n)} - \alpha\nabla\mathcal{J}\left(\boldsymbol{\theta}^{(n)}\right) \tag{3.34}$$

Equation (3.34) is repeated until a stopping criterion is met. Under some conditions such as convexity of the objective function, the algorithm converges to a global minimum (see e.g. [8, Chapter 9]). Letting $\boldsymbol{\theta} = \boldsymbol{w}$ and $\mathcal{J} = \mathcal{L}$ it is seen that (3.33) coincides with (3.29). However the use of non-linear activation functions in neural networks generally results in non-convex loss functions [22, p. 176]. Nevertheless the idea behind gradient descent is used in machine learning algorithms to iteratively obtain the weights by minimization of the loss function. The stepsize α in (3.34) is referred to as the *learning rate* in the context of neural network training.

Equation (3.34) requires evaluation of the gradient of the objective function. In the context of neural networks where the loss function $\mathcal{L}$ is written as an average of per example loss as in (3.30), the gradient is given as:

$$\nabla\mathcal{L}(\boldsymbol{w}) = \frac{1}{M}\sum_{i=1}^{M}\nabla\mathcal{L}_i(\boldsymbol{w}) \tag{3.35}$$

for M training examples. However as the number of training examples grow, the computation of the gradient cam become computationally heavy. Thus in neural network training one in practise estimates the gradient using a subset $\mathcal{B} \subset \mathcal{T}$ of examples:

$$\mathcal{B} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{M'} \quad , \quad (\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathcal{T} \tag{3.36}$$

The set $\mathcal{B}$ is referred to as a *batch* and M' is the referred to as the *batch size* . [22, p. 151]

In this project we will use two learning algorithms; the *Stochastic Gradient Descent (SGD)* algorithm [22, Sec. 8.3.1] and the *Adam* algorithm [29]. Both algorithms utilize the gradient descent update rule in (3.34) by estimating the gradient using a batch of examples, and similar to gradient descent both algorithms stop when a stopping criterion is met. The SGD algorithm is more or less a direct implementation of (3.34). The "stochastic" term in the name simply refer to the use of batches and should not be confused with e.g. stochastic random variables. In the Adam

algorithm both the gradient of the loss function and some second order moments of the gradient are used for updating the weights [29]. Alongside these algorithms we will also in some cases utilize a so called *momentum*, which is an extension of the algorithms. When using momentum, an additional term involving moving averages with an exponential decay of the past gradients is used in the gradient descent update rule. Using such term has been shown to speed up the training process [22, p. 296], in the sense that fewer epochs are needed to achieve a desired value of the loss function. For more information about momentum, we refer the reader to [22, Sec. 8.3.2].

It is worth noting that in practise one partitions the entire training set into disjoint batches each of size M' and cycles through these batches. One cycle through the entire set of batches, i.e. the entire training set, is referred to as an *epoch* and a stopping criterion could be to stop the training when a desired amount of epochs is reached.

3.5.3 Backpropagation

As seen in the previous section, training neural networks involves computing the gradient of the loss function w.r.t. the weights. The structure of neural networks enables one to evalute the gradient using the *chain rule of calculus* [22, Sec. 6.5].

Consider $\boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{y} \in \mathbb{R}^m$ and $z \in \mathbb{R}$ such that $\boldsymbol{y} = f(\boldsymbol{x})$ and $z = g(\boldsymbol{y})$ for differentiable functions $f : \mathbb{R}^n \to \mathbb{R}^m$ and $g : \mathbb{R}^m \to \mathbb{R}$. By the chain rule of calculus [18, p.953][22, p. 206]:

$$\frac{\partial z}{\partial x_i} = \sum_{j=1}^{m} \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i} \tag{3.37}$$

where x_i and y_j are element i and j of $\boldsymbol{x}$ and $\boldsymbol{y}$, respectively. The chain rule of calculus enables one to evaluate the gradient of a composite function. Since a neural network can be seen as a composition of several functions, e.g. each layer can be seen as function of the preceding layer(s), it follows that the chain rule of calculus can be used to evaluate the derivative of the loss function w.r.t. any weight and thus to evaluate the gradient in (3.34) of the gradient descent update rule.

However since a neural network often consists of several composite functions and thousands or millions of parameters, evaluating the gradient using the chain rule can become computational heavy. In general a node in a layer of a neural network directly or indirectly affects all nodes in all succeeding layers and thus the value of the loss function. When calculating the gradient of the loss function w.r.t weights connected to the same node using the chain rule of calculus, one could imagine that some terms or gradients are shared. This is the idea behind the *backpropagation* algorithm, which is used in neural network training to compute the gradient in an efficient manner using the chain rule [6, pp. 241-244]. We will not derive nor describe the backpropagation algorithm any further as it is out of the scope for this project. However we emphasize that for a neural network to be trainable, all computations handled in a neural network must be differentiable or at least have

a custom gradient defined. For instance the derivative of the (L)ReLU activation function (see Section 3.4.3) at 0 is not well-defined. Nevertheless the (L)ReLU activation function is still used in neural networks by defining the gradient using e.g. subgradients.

For further details on the backpropagation algorithm, we refer the reader to [22, Section 6.5] and [6, Section 5.3].

3.5.4 Under- and Overfitting

Even though neural network training can be seen as an optimization problem, it is important to emphasize that it differs from standard optimization problems. A key difference is that in neural network training one seeks a network that also performs well on *unseen* data, i.e. one seeks a network that generalizes well [22, p. 110]. For instance if the task of a neural network is to classify between pictures of dogs and cats, a proper trained network should be able to not only classify the pictures in the training set $\mathcal{T}$ correctly, but also any picture of a dog or cat. Likewise, the speech autoencoder developed in this project should be able to encode and decode unseen speech examples. The expected value of the loss function on unseen data is referred to as the generalization loss and is typically estimated using another set of examples $\mathcal{T}'$ referred to as the *test set* [22, p. 110]. Hence the generalization loss if often referred to as the *test loss*. When training a neural network using e.g. SGD to minimize the training loss, one hopes that the test loss will be minimized too [22, p. 275]. Typically the training and testing sets are obtained by dividing a single dataset into two disjoint sets. It is important to emphasize that only the training set is used for adjusting the weights, kernels and biases while the testing set is used for getting an general idea of how the neural network will perform on unseen data.

Even though a neural network is trained to minimize a training loss iteratively, one can not train a network for an infinite amount of epochs since this will affect the test loss [22, p. 111]. If the training loss is small while the test loss is significantly larger, the network has most likely learned features which are not present in the test set. This is referred to as *overfitting*. On the other hand if the network is not able to obtain a small training loss, the network is not able to learn the features and parameters needed for its task. Hence the network will also underperform on unseen data. This is referred to as *underfitting* . Thus the training of a neural network is a balance between obtaining a small training loss while maintaining a small test loss. The concept of under- and overfitting is illustrated in Figure 3.8 where polynomials of different degrees are fitted to a set of datapoints.

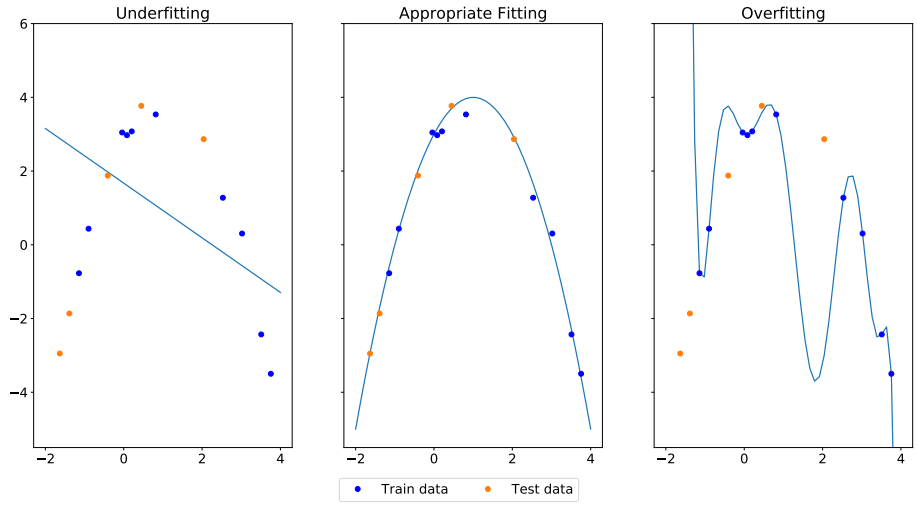


Figure 3.8: Underfitting, appropriate fitting and overfitting illustrated with curve fitting. In each plot a polynomial is fitted to the 10 blue data points representing the training data. The 2^{nd} degree polynomial (middle) best fits both the train and test datapoints, while the 1^{st} (left) and 10^{th} (right) degree polynomials under- and overfit, respectively.

As mentioned previously, a stopping criterion for the learning algorithm could be to stop the training when a specified amount of epochs is reached. In this project we will use this as a stopping criterion for the training, where the specified amount of epochs will be found through initial experiments. However in order to prevent overfitting, the training will be stopped earlier if the test loss has not improved for a another specified amount of epochs. The weights, biases and kernels obtained at the epoch with the best test loss will be restored. The process of stopping the training early is referred to as *early stopping* [22, p. 246-247].

4. Our Speech Autoencoder

As described in Section 3.3 we utilize DNN autoencoders to perform speech coding in this project. The purpose of this chapter is to present the general coding process and structure of the autoencoder employed in this project alongside some of its elements.

4.1 General Structure

The overall speech coding process utilized in this project is illustrated in Figure 4.1.

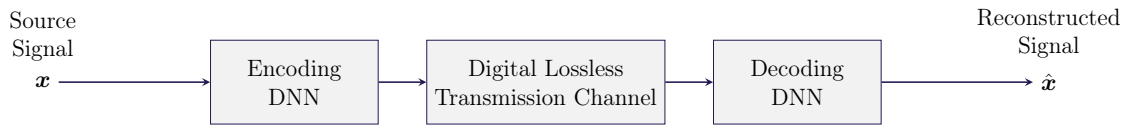


Figure 4.1: The overall speech coding process utilized in this project.

In this project we consider frame sizes of N_x as input to the DNNs. Thus let $\boldsymbol{x} \in \mathbb{R}^{N_x}$ be a frame of size N_x of a source signal. From Figure 4.1 it is seen that the frame is encoded using a DNN. The output of the encoding DNN is then transmitted losslessly over a digital transmission channel. The encoded and transmitted version of the frame is then reconstructed by a decoding DNN. Our assumption of a lossless digital transmission channel enables us to train the encoding DNN and decoding DNN as a single DNN, i.e. an autoencoder. Comparing Figure 4.1 with Figure 2.2 on page 6 it is seen that the speech coding approach considered in this project is an end-to-end approach, meaning that we will employ DNNs to perform the entire coding process except from external processes related to the channel such as e.g. bitstream conversion.

The general structure of the autoencoder employed in this project is shown in Figure 4.2.

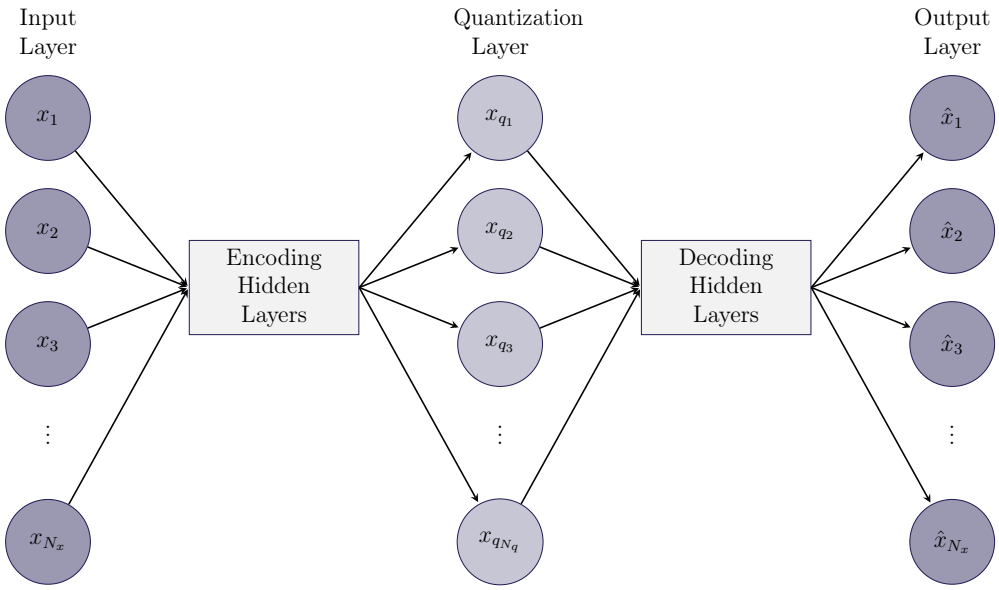


Figure 4.2: The general structure of the autoencoder utilized in this project.

From the Figure 4.2 it can be seen that the autoencoder consists of an input layer of dimension N_x, encoding hidden layers, a quantization layer of dimension N_q, decoding hidden layers and a output layer of dimension N_x. In this project we restrict the input, quantization layer and output layer to be fully connected layers. The type of layers used in the encoding and decoding hidden layers will be specified in Chapter 8 and Chapter 9. However we restrict our focus to feed forward neural networks. In the quantization layer a b-bit quantizer is employed and the quantizer acts as an activation function (see Section 3.3) for this layer.
Even though we here consider the input $\boldsymbol{x} \in \mathbb{R}^{N_x}$ as a frame of size N_x of the source signal, the input to the autoencoder does not necessarily (only) have to be a frame of the source signal itself. The coding process in Figure 4.1 implies great flexibility in the sense that one can employ an arbitrary amount of encoding DNN pre-processing and decoding DNN post-processing steps. In Chapter 8 we will consider different input/output strategies. Furthermore we will in this project consider quantization dimensions of $N_q = N_x$ alongside undercomplete autoencoders, i.e. $N_q < N_x$. The idea of using uncerdomplete autoencoders arises from dimensionality reduction. For instance if correlations are present in a high dimensional input it might be possible to (help the network) to describe the input using fewer dimensions. By employing the quantization as an internal part of the autoencoder great controllability of the bit rate is also obtained, as we shall see soon.

The remainder of this chapter is concerned with presenting and describing the quantization layer and the associated quantizer.

4.2 The Quantization Layer

As previously described the quantization layer consists of a b-bit quantizer acting as the activation function for the layer.

Let N_e denote the dimension of the layer prior to the quantization layer, i.e. the last encoding hidden layer of the autoencoder. Let further $\boldsymbol{h} \in \mathbb{R}^{N_e}$ be the output of this layer. i.e. the output of the activation function associated with this layer. The output, $\boldsymbol{x}_q$, of the quantization layer is implemented in the following way:

$$\boldsymbol{a}_q = \boldsymbol{W}_q \boldsymbol{h} + \boldsymbol{b}_q \tag{4.1a}$$

$$\boldsymbol{x}_q = \texttt{Quantize}_b\left(\boldsymbol{a}_q\right) \quad , \quad b \in \mathbb{N} \setminus \{0\} \tag{4.1b}$$

where $\boldsymbol{W}_q$ and $\boldsymbol{b}_q$ are the weights and biases (see Section 3.1) associated with the quantization layer and $\texttt{Quantize}_b$ is a b-bit quantizer (see Section 2.2).

The quantizer acts element wise on $\boldsymbol{a}_q$ and is thus a scalar quantizer (see Section 2.2.1). Let $a_q \in \mathbb{R}$ be an element of $\boldsymbol{a}_q$. The quantizer $\texttt{Quantize}_\texttt{b}$ is implemented as:

$$a_d = \max\left(\alpha, \min\left(\beta, a_q\right)\right) \tag{4.2a}$$

$$\texttt{Quantize}_b(a_d) := \text{round}\left(\frac{a_d}{\Delta}\right)\Delta \tag{4.2b}$$

where $\alpha, \beta \in \mathbb{R}$ and $\beta > \alpha$.

The quantizer is implemented with 2^b quantization levels using a step size Δ of:

$$\Delta = \frac{\beta - \alpha}{2^b - 1} \tag{4.3}$$

Thus an input to the quantizer, a_q, is first mapped to a value $a_d \in [\alpha, \beta]$ using (4.2a). The value a_d is then quantized with b-bits using the quantizer in (4.2b), i.e. the output of the quantizer $z_q \in [\alpha, \beta]$ where the number of distinct elements in $[\alpha, \beta]$ is 2^b. As seen from (4.2b) and (4.3), the quantizer is a uniform midtread quantizer with midpoint reconstruction, i.e. the quantization values are chosen such that they correspond to the midpoint of the decision intervals. The values of α and β are fixed and is not changed during training. In this project we chose $\alpha = 0$ and $\beta = 1$.

The need to restrict a_q to the range $[\alpha, \beta]$ prior to the quantizer is due our restriction to fixed-rate quantization. Thus it is needed to decide upon the number and size of intervals. It should be noted that whether the range is $[0, 1]$ or e.g. $[1, 5]$ should not make a difference, due to the network being capable of adjusting the weights. For the same reason, a range including negative values is not needed even though the input to the network may have negative values.
The quantizer in (4.2b) with the range $[0, 1]$ has already been implemented in successful networks such as DoReFa-Net [61], where weights and the outputs from the activation functions are quantized to speed up training and where comparable results with well known non-quantized networks are achieved.

An example of the quantizer in (4.2b) for inputs $a_q \in [-1, 2]$ and with $\alpha = 0, \beta = 1$ and $b = 3$ is seen in Figure 4.3.

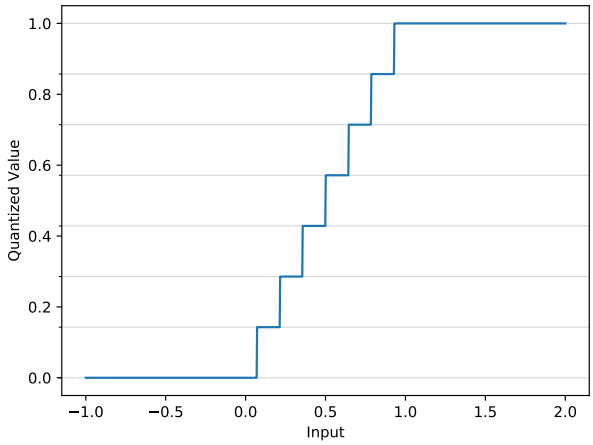


Figure 4.3: Inputs and their quantized values using the quantizer in (4.2b) with $b = 3$ and inputs $a_q \in [-1, 2]$.

In Section 2.2 we presented quantization as a procedure involving forward and backward quantization. However looking at (4.2b), we see that the implemented quantizer employs both the forward and backward quantization in one step. It is important to emphasize that the quantization can easily be split up into a forward and backward procedure. Thus the autoencoder can be split into two DNNs as in Figure 4.1 and where the encoding DNN consists of the input layer, the encoding hidden layers and a layer containing the forward quantization. The decoding DNN will then consist of a layer containing the backward quantization, the decoding hidden layers and the output layer. The reason for not employing the forward and backward quantization procedures as separate layers during training is due to practical aspects in Tensorflow regarding the gradient of the quantizer.

From (4.2b), it is seen that the quantizer is a mapping from $\mathbb{R}$ to a discrete set of 2^b values. Thus the quantization function is not differentiable. To comprehend this we notice that for a large b the stepsize Δ in (4.3) is small and thus:

$$\texttt{Quantize}_b\left(a_d\right) \approx a_d = \max\left(\alpha, \min\left(\beta, a_q\right)\right) \tag{4.4}$$

I.e. the quantizer approximates the identity function for inputs a_d. The approximation is shown in Figure 4.4 for $a_q \in [-1, 2]$, $\alpha = 0$ and $\beta = 1$.

Using this approximation the quantizer can be seen as a mixture of three piecewise linear functions. We define the derivative of the quantizer w.r.t. the input a_q as:

$$\frac{\partial \texttt{Quantize}_b}{\partial a_q} := \frac{\partial a_d}{\partial a_q} = \begin{cases} 0 & a_q \leq \alpha \\ 1 & \alpha < a_q < \beta \\ 0 & a_q \geq \beta \end{cases} \tag{4.5}$$

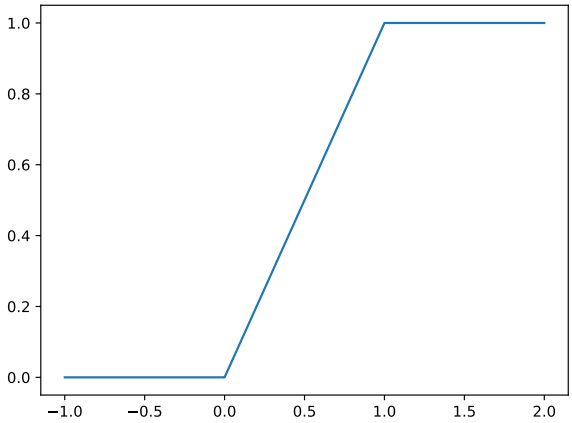


Figure 4.4: Approximation of the quantizer from Figure 4.3 for inputs $a_q \in [-1, 2], \alpha = 0$ and $\beta = 1$.

The approach of approximating the quantizer with linear functions to obtain a gradient is also used in DoRaFe-Net [61] and furthermore in Tensorflow built-in quantization functions used for quantizing already trained models.

It is important to emphasize that the approximation in (4.4) is only used to obtain the gradient. Thus the activations in the quantization layer are still quantized using equations (4.2a)-(4.2b). Furthermore it is worth noting that even though a layer is quantized node wise with the same uniform scalar quantizer, the network sees the whole quantization layer as a vector. Thus the quantization layer can be seen as a vector quantizer (see Section 2.2.2). For a quantization dimension of N_q and a b-bit quantizer the codebook size is $2^{b \cdot N_x}$.

We previously stated that employing quantization as an interval part of the autoencoder provides great controllability of the bit rate. To see this, let N_q denote the dimension of the quantization layer and b denote the number of bits in (4.2b). For a source signal with a frame size N_x of samples, the bit-rate R in bits per sample is given as:

$$R = \frac{N_q \cdot b}{N_x} \tag{4.6}$$

Thus for a fixed frame size, one can adjust the bit-rate by adjusting either N_q and/or b. From (4.6) it follows that if $N_q < N_x$ one can obtain bit rates $R < 1$. Since the bit-rate is a function of the frame size and not the dimension of the input layer, the bit-rate is not affected even though we provide additional data at the input layer as will be done in Chapter 8.

5. Information and Rate-distortion Theory

The purpose of this chapter is to introduce some definitions and quantities regarding the measure of information, All the presented quantities belong to the field of *information theory*, and are essential elements to the speech codec developed in this project. Using these quantities one can describe the objectives of speech coding (see Chapter 2) from an information theoretic perspective known as *rate-distortion theory*. This will be described at the end of this chapter in Section 5.3.

In information theory, information is quantified using probability distributions and for reasons that will become clear in Chapter 6, we will unless otherwise stated restrict our focus to discrete random variables.

5.1 Entropy

One way to think of and quantify information is in terms of *uncertainty* [46, pp. 10-11]. To see this consider a coin tossing experiment and assume one wants to predict the outcome of the coin toss. If the coin is fair, knowledge of the probabilities will not provide information useful for predicting the outcome. On the other hand if the probability of e.g. tails is 0.99, the knowledge of this probability results in less uncertainty about the outcome. The use of uncertainty as a measure of information leads to the concept of *entropy*.

Definition 5.1 — Entropy.
Let X be a discrete random variable with probability mass function $p(x)$. The entropy of X is defined as:

$$H(X) := -\sum_{x \in S_X} p(x) \log(p(x)) \tag{5.1}$$

where S_X is the sample space. [12, p. 14]

Remark We will use the convention $p(x) \log p(x) = 0$ for $p(x) = 0$ since $\lim_{x \to 0} x \log(x) = 0$ [12, p. 14]. Furthermore the logarithm in Definition 5.1 is not restricted to a specific base. In this project and unless otherwise stated, we will use the logarithm to the base of 2. In such cases the entropy is expressed in bits.

Since the definition of entropy relies on the probability mass function rather than the outcomes of the random variable, it follows that $H(X) \geq 0$ [12, p. 15]. From the Definition it can further be seen that the entropy of a random variable is zero if and only if $p(x) = 1$ for some event $x \in S_X$. If we consider the entropy as a measure of uncertainty about a random variable, this intuitively makes sense. If $p(x) = 1$ for some event $x \in S_X$ there is no uncertainty involved since the outcome is purely deterministic.

The entropy and thus the uncertainty as a function of the probability mass function for a Bernoulli random variable is shown in Example 5.1.

Example 5.1 — Entropy of Bernoulli Random Variable.
Let $X \sim \text{Bernoulli}(p)$, i.e.:

$$P(X = x) = \begin{cases} p & , x = 1 \\ 1 - p & , x = 0 \end{cases} \tag{5.2}$$

For instance X could represent the coin tossing experiment at the beginning of this section. Using Definition 5.1, the entropy of X is given as:

$$H(X) = -p \log(p) - (1 - p) \log(1 - p) \tag{5.3}$$

The entropy as a function of p can be seen in Figure 5.1.

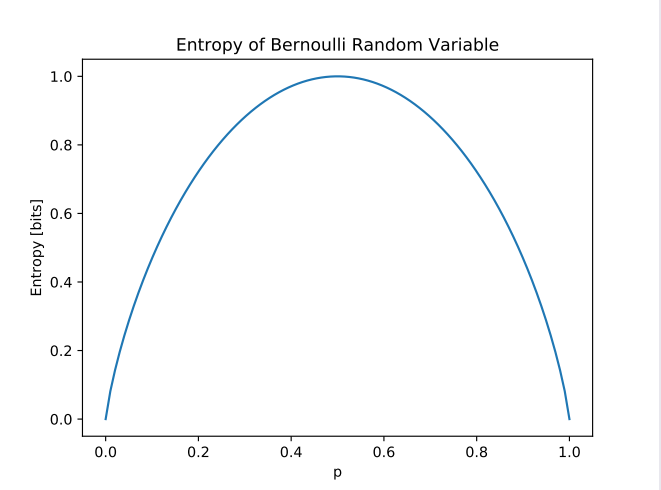


Figure 5.1: The entropy of a Bernoulli random variable as a function of p.

From Figure 5.1 in Example 5.1 it can be seen that the entropy is maximized when $p = 0.5$. In general it can be shown that for a discrete random variable X with cardinality $|S_X| = n$ of the sample space S_X, the entropy is maximised with value

$\log(n)$ when X is uniformly distributed [46, p. 11]. This intuitively makes sense since one is most uncertain about an outcome if the events are equal probable. E.g. one is most uncertain about the outcome of a coin tossing experiment when the coin is fair.

Instead of viewing the entropy as a measure of uncertainty of a random variable X, one can also view the entropy as a measure of the average amount of information required to describe X [12, p. 19]. Thus when using the logarithm to the base of 2, the entropy is a measure of the average amount of bits required to describe a random variable. Indeed if there is no uncertainty regarding a random variable, i.e. $p(x) = 1$ for some x, no bits are needed to describe it. Returning to Example 5.1 it can seen that when $p = 0.5$ an average amount of 1 bit is required to describe the Bernoulli random variable - which we in fact already did [1].

The entropy can be extended to several random variables by e.g. considering the random variable pair (X, Y) as a random vector variable [12, p. 16]. This is referred to as the *joint entropy*. The joint entropy for two random variables is defined below.

Definition 5.2 — Joint Entropy.
Let X and Y be two discrete random variables with joint probability mass function $p(x, y)$. The joint entropy is defined as:

$$H(X, Y) := -\sum_{x \in S_X} \sum_{y \in S_Y} p(x, y) \log\left(p(x, y)\right) \tag{5.4}$$

where S_X and S_Y are the sample spaces of X and Y, respectively. [12, p. 16]

When considering the entropy as a measure of uncertainty it seems reasonable that the entropy of one random variable is somehow affected by the knowledge of another variable. This leads to the definition of *conditional entropy*.

Definition 5.3 — Conditional Entropy.
Let X and Y be discrete random variables with joint probability mass function $p(x, y)$. The conditional entropy of Y given X is defined as:

$$H(Y|X) := -\sum_{x \in S_X} \sum_{y \in S_Y} p(x, y) \log\left(p(y|x)\right) \tag{5.5}$$

where $p(y|x)$ is the conditional probability mass function and S_X and S_Y are the samples spaces of X and Y, respectively. [12, p. 17]

An important property used in this project, which we will state without proof, is that $H(Y|X) = 0$ if and only if Y is a function of X [12, p. 37].

[1] bad joke intended :)

5.2 Mutual Information

In the previous section we defined the entropy as the average amount of information required to describe one or more random variables. Another relevant measure of information is the amount of information one random variable contains about another. This is referred to as the *mutual information* [12, p. 13].

Definition 5.4 — Mutual Information.
Let X and Y be discrete random variables with joint probability mass function $p(x, y)$ and marginal probability mass functions $p(x)$ and $p(y)$, respectively. The mutual information between X and Y is defined as:

$$I(X;Y) := \sum_{x \in S_X} \sum_{y \in S_Y} p\left(x, y\right) \log \left(\frac{p\left(x, y\right)}{p\left(x\right) p\left(y\right)} \right) \tag{5.6}$$

where S_X and S_Y are the sample spaces of X and Y, respectively. [12, p. 20]

From Definition 5.4 it can be seen that $I\left(X;Y\right) = I\left(Y;X\right)$. Furthermore an important property which we will not derive is that $I(X;Y) \geq 0$ with equality if and only if X and Y are independent [12, p. 19]. If we think of mutual information as the amount of information one random variable contains about another, it intuitively makes sense that the mutual information is non-negative. Similarly if two random variables are independent, they can not contain information about one another. Hence the mutual information is zero in such cases. For further details we refer the reader to [12, Section 2.3].

Using the entropy definitions, the mutual information can be expressed as [12, pp. 20-21]:

$$\begin{aligned}
I(X;Y) &= \sum_{x \in S_X} \sum_{y \in S_Y} p(x,y) \log \left(\frac{p(x,y)}{p(x)p(y)} \right) && \text{(5.7a)} \\
&= \sum_{x \in S_X} \sum_{y \in S_Y} p(x,y) \log \left(\frac{p(x|y)}{p(x)} \right) && \text{(5.7b)} \\
&= -\sum_{x \in S_X} \sum_{y \in S_Y} p(x,y) \log(p(x)) + \sum_{x \in S_X} \sum_{y \in S_Y} p(x,y) \log(p(x|y))) && \text{(5.7c)} \\
&= -\sum_{x \in S_X} p(x) \log(p(x)) - \left(-\sum_{x \in S_X} \sum_{y \in S_Y} p(x,y) \log(p(x|y)) \right) && \text{(5.7d)} \\
&= H(X) - H(X|Y) && \text{(5.7e)}
\end{aligned}$$

where (5.7b) follows from the definition of conditional probability distribution, (5.7d) follows from calculating the marginal distribution and (5.7e) follows from using Definitions 5.1 and 5.3. From (5.7e) it can be seen that the mutual information can also be interpreted as the reduction in uncertainty of the random variable X by having knowledge of Y [12, p. 19]. Since $I\left(X;Y\right) = I\left(Y;X\right)$ this reduction in uncertainty is equal to the reduction in uncertainty of the random variable Y by having knowledge of X.

The different presented quantities and their relations are illustrated in the Venn diagrams of Figure 5.2.

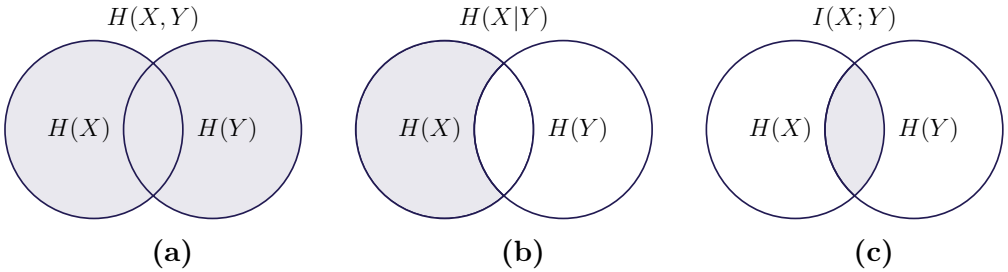


Figure 5.2: Venn diagram of the presented quantities. The circles represent the entropies $H(X)$ and $H(Y)$. The coloured area shows (a) the joint entropy $H(X,Y)$, (b) the conditional entropy $H(X|Y)$ and (c) the mutual information $I(X;Y)$.

5.3 Rate-distortion Theory

In Chapter 2 it was described that lossy digital coding involves minimization of both the bit rate R and some distortion D between the source signal and reconstructed signal. This is illustrated in Figure 5.3.

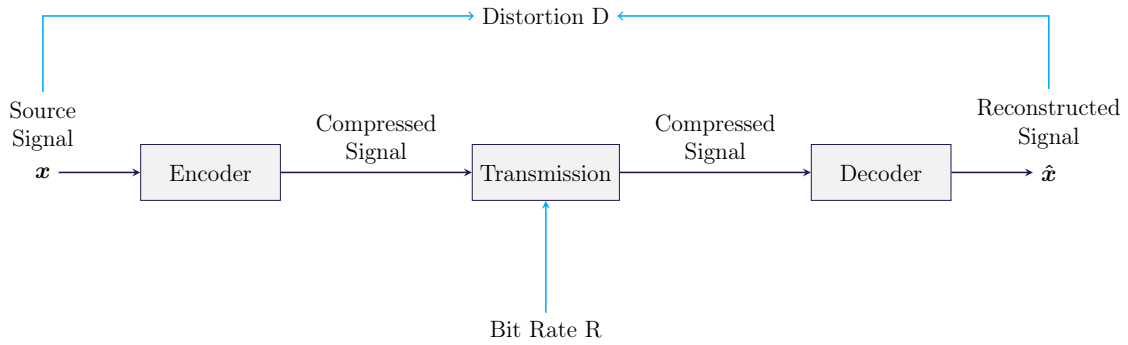


Figure 5.3: General digital coding process from Chapter 2 with bit rate R and distortion D.

The coding problem can be viewed in two ways [12, p. 301]; Given a maximum rate R, minimize the distortion D or given a distortion D, minimize the rate R. If we assume that a maximum distortion of D^* is given, the question remains on how well one can do. I.e. what is the minimal possible rate such that the distortion does not exceed D^*?. This leads to the definition of the *rate-distortion function*.

In the following we will consider a source signal as a random variable X and let $\hat{X}$ denote the corresponding reconstructed signal. The rate-distortion function is defined below

Definition 5.5 — Rate-distortion Function.
Let X and $\hat{X}$ be a source signal and the corresponding reconstructed signal. The rate-distortion function for distortion measure d is defined as:

$$R\left(D^*\right) = \min_{D=\mathrm{E}\left[d(X,\hat{X})\right]\leq D^*} I\left(X;\hat{X}\right) \tag{5.8}$$

where $\mathrm{E}\left[\cdot\right]$ denotes the expectation operator. [12, p. 307]

The rate-distortion function gives the minimal possible rate R for distortion D^* [12, pp. 306-307]. We will not go into further details about this, the assumptions about X, $\hat{X}$ and D nor whether such function is achievable. We refer the reader to [12, Chapter 10] for further details.

An important rate-distortion function which is used in Chapter 8 is stated below without proof.

Theorem 5.1 — Rate-distortion function for Gaussian Source with Memory.
Let $\{X_t\}$ be a stationary stochastic process given as:

$$X_{t+1} = \alpha X_t + \epsilon_t \quad , \quad t = 1, \ldots \tag{5.9}$$

where $\epsilon_t \overset{i.i.d.}{\sim} \mathrm{N}\left(0, \sigma_\epsilon^2\right)$ and $X_0 \sim \mathrm{N}\left(0, \sigma_X^2\right)$ with $\sigma_X^t = \sigma_\epsilon^2 / \left(1 - \alpha^2\right)$.
The rate-distortion function is given as:

$$R(D) = \frac{1}{2} \log\left(\alpha^2 + \frac{\sigma_\epsilon^2}{D}\right) \tag{5.10}$$

for per sample MSE distortion D. [15]

6. Information Bottleneck

The Information Bottleneck method [53] was originally proposed in 1999 as a pure information theoretic method for compressing a random variable as much as possible while maintaining the relevant information w.r.t some other variable. In 2015 Tishby and Zaslavsky [52] proposed that the training and objectives of a DNN could be explained by the IB principle. The purpose of this chapter is to introduce both the IB principle and its relation to DNNs as it was proposed in [52] and the results empirically shown in [47]. This will be done in Section 6.1. In Section 6.2 we relate the IB principle to the DNN speech autoencoder developed in this project, by proposing a loss function inspired by the IB principle.

6.1 Information Bottleneck in DNNs

The IB principle in [52] applies the knowledge of rate-distortion theory (see Section 5.3) to DNNs (see Chapter 3) for classification. However in order to explain the intuition behind the IB principle, we will first present the IB problem from a general point of view.

Let X be a random variable that is to be compressed in a lossy way such that some information is lost. Assume that the compression is to be made such that most of the relevant information w.r.t. some other variable is preserved. Let Y be a random variable representing the relevant information such that the mutual information (see Section 5.2) $I(X;Y)$ quantifies the relevant information X contains about Y. Let T denote the compressed version of X, and consider the *Markov chain* (see e.g. [39]):

$$Y \to X \to T \tag{6.1}$$

We will not go into further details about Markov chains. For this project it suffices to say that the random variables A, B and C form a Markov chain $A \to B \to C$ if and only if A and C are conditionally independent given B [12, p. 34]. Thus the Markov chain in (6.1) should be seen as the compressed version T being a function of X and that the information about Y in T is only achievable through X. In [52] the authors state that an compression algorithm should aim to compress X in such

a manner that only the relevant information w.r.t. Y is preserved. This is stated formally below.

Definition 6.1 — Information Bottleneck (IB) Problem.
Let Y, X and T be random variables such that:

$$Y \to X \to T \tag{6.2}$$

The Information Bottleneck (IB) problem is defined as:

$$\min_{p(t|x)} I(X;T) - \beta I(T;Y) \tag{6.3}$$

for $\beta > 0$ and where $I(\cdot,\cdot)$ and $p(t|x)$ denotes the mutual information and encoding distribution, respectively. [52]

In Definition 6.1 it is assumed that the compressed representation T of X is given as a (possibly stochastic) mapping of X characterized by the distribution $p(t|x)$ [47]. Hence the distribution is referred to as the encoding distribution and the minimization is taken across all possible mappings.

From Section 5.3 it is seen that the term $I(X;T)$ in Definition 6.1 is the rate of compression. The term $I(T;Y)$ is the mutual information between T and Y, and since the information that Y contains is of interest, the mutual information quantifies the amount of preserved (relevant) information. Thus the IB problem describes a trade-off between minimizing the rate of compression and maximising the amount of preserved information. The trade-off is controlled by β. When $\beta = 0$ only compression is relevant, while a large value of β implies that the preservation of information is more important.
Due to the markov chain assumption and the *Data Processing Inequality (DPI)* (see [12]) given as:

$$I(X;T) \geq I(T;Y) \tag{6.4}$$

it follows that the optimization problem in Definition 6.1 forms a bottleneck [52].

In order to relate the IB problem in Definition 6.1 to neural networks as done in [52], consider a feedforward classification DNN where each layer is considered as a random variable. Let X, T_i, $\hat{Y}$ and Y denote the random variables representing the input layer, the i'th hidden layer, the output layer and corresponding desired output (class label), respectively. Since each layer is obtained by processing of the preceding layer, a layer T_{i+1} is conditionally independent of T_{i-1} given T_i. Thus due to the feedforward structure, a DNN consisting of N hidden layers forms the markov chain [47]:

$$Y \to X \to T_1 \to T_2 \to \cdots \to T_N \to \hat{Y} \tag{6.5}$$

Using the Markov chain in (6.5) it is seen that the IB problem in Definition 6.1 applies to each layer and with $Y \to X \to \hat{Y}$ also to the network as a whole.

Previously in Chapter 3 we presented neural networks as a function from the input layer to the output layer, and that the objective was to obtain an approximating

function $\hat{f}$ such that $\hat{f}(x) = \hat{y} \approx y$ for input and desired output examples x and y, respectively. However as seen in (6.5) the Markov chain starts from Y. Similarly to the Markov chain in (6.2) this should be seen as the information regarding Y, i.e. the labels, being only implicitly accessible through X. This is indeed the case in a neural network setup.

The interpretation of the IB problem in context of classification DNNs is as follows [52]: Since the input X is often of higher dimension than the label Y, then in general most of the information in X is not informative about Y. For instance consider classification of pictures of cats and dogs. A picture of is often high dimensional, i.e. it consists of many pixels, while the label on the other hand is low dimensional. Following the example given in Section 3.1 the label could be $y = [1, 0]^T$ for a cat and $y = [0, 1]^T$ for a dog. When using DNNs one implies that there is a relation between the input and class label. Thus one can view X as arisen through some processing of the label Y. Since most of the information in X is not informative about Y, the goal is to compress X in such a way that only the relevant information is preserved. I.e. such that the relevant information in order to predict Y is preserved.

In [52] it was claimed that the pair $(I(X;T), I(T;X))$ forms a point on the so called *information plane*. This and the trade-off in Definition 6.1 was empirically verified in [47] for classification DNNs. The authors showed that a classification DNN trained with SGD (see Section 3.5.2) implicitly achieves a compression of the input X while maximising the amount of preserved information between the label Y and a layer. They showed that this is done in two stages. This is illustrated in Figure 6.1.

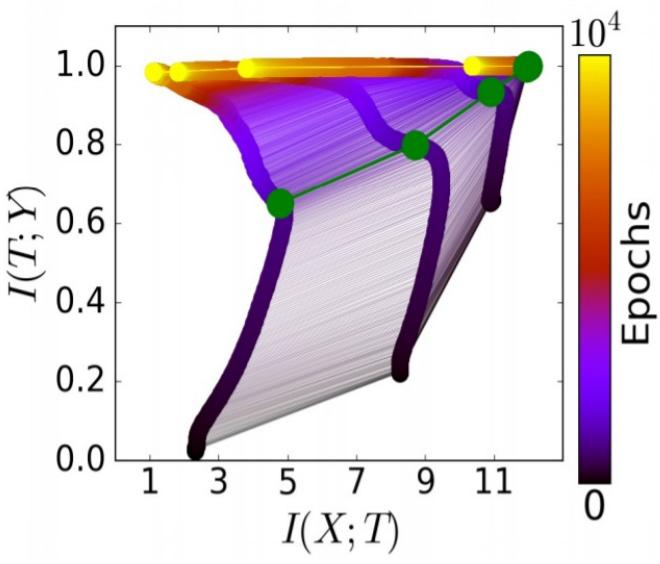


Figure 6.1: Information plane. From [47]

In Figure 6.1 the pair $((X;T), I(T;X))$ for different layers T is illustrated as a function of epochs. The color intensity is determined by the epoch number. Points connected with a line correspond to mutual information values for layers of the same network in the same epoch. It is seen that the points form similar paths, each path

corresponding to a specific layer. The leftmost path correspond to the hidden layer closest to the output layer, while the rightmost path correspond to the hidden layer closest to the input layer. For each path it can be seen that during training both mutual informations generally increase and that the mutual informations are higher for layers closest to the input layer. However at some point a phase change (green points) is seen. At this phase change the mutual information $I(X;T)$ decreases while the mutual information $I(T;Y)$ increases. This is referred to as the *compression phase* [47].

The claims proposed in [52] and empricially showed in [47] have been widely discussed [44][5][38]. For instance in [38] it is shown, that the compression phase is not achieved for all activation functions. The authors argue that the compression phase seen in [47] is due to the saturating activation functions used. In [5] it is also argued that the IB problem in Defintion 6.1 is only suitable if the layers of a network is quantized in some manner. Several proposals have been made inspired by the information bottleneck [31][19][2]. For a detailed overview of the IB principle in the context of DNNs, we refer the reader to [24].

The IB principle in [52] was stated in the context of classification DNNs only. However in this project where autoencoders are considered, i.e. $X = Y$, (6.3) is not suitable. Nevertheless the intuition behind the IB problem is still relevant in the context of speech coding, as we aim to compress the input X at some rate while maintaining some relevant information in the compressed variable T. Another immediate problem when considering the IB in relation to neural network training, is that the mutual informations rely on knowledge of the probability distributions which are often unknown. Thus estimates are needed. However as explained in Section 3.5 the gradients of each computation related to network training is needed. Thus the estimators must either be differentiable or at least have a custom gradient defined. In the following section we propose a loss function circumventing these problems.

6.2 Loss Function for Speech Autoencoder Inspired by Information Bottleneck

In section we propose a loss function inspired by IB.
Consider each layer of a DNN autoencoder as a random variable. Thus let X denote the input to a DNN autoencoder and let $\hat{X}$ denote the output of the DNN. As stated in Section 3.3, a DNN autoencoder can be seen as consisting of two DNNs; an encoding DNN f_e and a decoding DNN f_d such that:

$$f_d(f_e(X)) = f_d(X_q) = \hat{X} \tag{6.6}$$

where X_q denotes the random variable representing the encoded input, i.e. the compressed version of X. From Chapter 4 it follows that X_q is the random variable representing the quantization layer of our speech autoencoder.

The goal of our autoencoder is to encode the input X at a minimal rate while maintaining the relevant information of the input useful for decoding the signal. We

argue that the mutual information quantities in Definition 6.1 can not stand alone, if the IB problem is wished to be used as a training function in this context. The mutual informations must be seen in relation to some context. To see this consider $\hat{X}$ being a permutation of X. Then $I\left(X;\hat{X}\right)$ is maximized but the output does not resemble the input - at least not sample-wise. Instead we will use the MSE loss function from Section 3.5.1 as an implicit measure of relevance, since a low MSE will most likely imply that relevant information is preserved. Regarding the rate term in Definition 6.1, recall from (5.7e) in Section 5.2 that $I(X;Y) = H(X) - H(X|Y)$. Thus:

$$I(X;X_q) = H(X_q) - H(X_q|X) = H(X_q) \tag{6.7}$$

where the last equation follows from X_q being a deterministic function of X for fixed weights. In this project we will consider the marginal entropies $H(X_{q_i}), \ldots, H(X_{q_{N_q}})$ of the quantization nodes $X_{q_i}, \ldots, X_{q_{N_q}}$ for quantization layer with dimension N_x.

In this project we propose the loss function:

$$\min \mathcal{L}_{MSE}(\boldsymbol{w}) + \beta \sum_{i=1}^{N_q} H(X_q) \tag{6.8}$$

where $\mathcal{L}_{MSE}(\boldsymbol{w})$ is the MSE from Section 3.5.1 and $H(\cdot)$ is the entropy (see Section 5.1).

In (6.8) the second term is only dependent on the encoding part of the autoencoder. We minimize the entropy in the hope that this will force the network to extract only relevant parameters. For instance if we consider a quantization layer of size N_q, then even if the quantization layer is smaller than the dimension of the input, the total number of possible quantized vectors is $2^{N_q \cdot b}$ for a b-bit quantizer. Thus even for quantizers with a small value of b, the total number of possible vectors is large. By minimizing the entropy of the quantization layer it is the hope, that the network is forced to perform a many-to-one mapping which in turn could help the network to discover relevant features of the input. The first term on the other hand is dependent on the network as a whole. A poor encoding, i.e. an encoding that does not contain relevant features about the input, will likely effect the decoding and thus the output $\hat{X}$. Thus minimizing the mean squared error implicitly implies not only maximization of the amount of preserved information in the encoding part but also maximization of the resemblance between the input and output.

By considering the entropy we do not avoid the problems of estimators and non-differentiability. However the nature of our autoencoder provides means to estimate the entropy and where the operations are directly linked to the nature of the quantizer employed in the quantization layer. Thus no further definitions of gradients are needed.

6.2.1 Estimating the Entropy using a 1-bit Quantizer

We will first estimate the entropies in (6.8) for a 1-bit quantizer. The estimation of the entropies is based on J examples, i.e. realizations, of X_q.

Let $\boldsymbol{x}_q \in \mathbb{R}^{N_q}$ denote an output example of the quantization layer with elements x_{q_i}. Since we consider a 1-bit quantizer it follows from Section 4.2 that $x_{q_i} \in \{0, 1\} \, \forall i$. Let further $x_{q_i}^{(j)}$ denote node i in the quantization layer of data example j. We define:

$$p_i := \frac{1}{J} \sum_{j=1}^{J} x_{q_i}^{(j)} \tag{6.9}$$

to be the mean value of node i in the quantization layer across J examples. Since $x_{q_i} \in \{0, 1\}$ it follows that $p_i \in [0, 1] \, \forall i$. Furthermore if we consider node i as a random variable X_{q_i}, then due to the 1-bit quantizer we can regard p_i as a probability:

$$p_i = P(X_{q_i} = 1) \tag{6.10}$$

Thus the pair $(p_i, 1 - p_i)$ forms an estimate of the pmf of the discrete random variable X_{q_i}. An estimate $\hat{H}$ of the entropy of node i can then be obtained by:

$$\hat{H}\left(X_{q_i}\right) = -p_i \log\left(p_i\right) - \left(1 - p_i\right) \log\left(1 - p_i\right) \tag{6.11}$$

If $p_i = 0$, then all examples of node i in the quantization layer are 0. Thus the entropy should be zero. By looking at (6.11) we see that this is indeed the case. This is also true for $p_i = 1$ where all examples of node i have a value of 1. If $p_i = 0.5$ an equal amount of examples of node i has obtained the values 0 and 1. The pmf then forms a uniform distribution and it follows from Section 5.1 that the entropy in (6.11) is maximized.

By using a 1-bit quantizer one can thus replace (6.8) with:

$$\min \boldsymbol{\mathcal{L}}_{MSE}\left(\boldsymbol{w}\right) + \beta \sum_{i=1}^{N_q} \hat{H}\left(x_{q_i}\right) \tag{6.12}$$

where $\hat{H}$ is given by equations (6.9) and (6.11).

6.2.2 Estimating the Entropy using a b-bit Quantizer

We now generalize the loss function in (6.12) to a b-bit quantizer with 2^b quantization levels.

Let again $\boldsymbol{x}_q \in \mathbb{R}^{N_q}$ denote the output from the quantization layer with elements x_{q_i}. Let further $\tilde{q}_1 < \ldots < \tilde{q}_{2^b}$ denote the quantized values for a b-bit quantizer, i.e. $x_{q_i} \in \{\tilde{q}_1, \ldots, \tilde{q}_{2^b}\} \, \forall i$. Again we consider node i as a random varaible X_{q_i} and we wish to compute the probabilities:

$$p_{i,l} = P\left(X_{q_i} = q_l\right) \quad , \quad i = 1, \ldots, N_q, l = 1, \ldots, 2^b \tag{6.13}$$

in order to obtain an estimate of the pmf of each node and hence obtain an estimate of the entropy. However when using a b-bit quantizer with $b \neq 1$ one can not rely on

the node values alone to compute these probabilities as in (6.9). To comprehend this we first define:

$$\Upsilon_l\left(x_{q_i}^{(j)}\right) := \prod_{m \neq l} \left|\tilde{q}_m - x_{q_i}^{(j)}\right| \tag{6.14}$$

where $x_{q_i}^{(j)}$ is node i in the quantization layer of data example j. If $x_{q_i}^{(j)} = \tilde{q}_l$, the product in (6.14) equals the product of relative distances between the quantization value q_l and all other quantization values. If on the other hand $x_{q_i}^{(j)} \neq q_l$, equation (6.14) equals zero. We now define a normalization parameter:

$$\epsilon_l := \Upsilon_l\left(\tilde{q}_l\right) = \prod_{m \neq l} \left|\tilde{q}_m - \tilde{q}_l\right| \tag{6.15}$$

Using equations (6.14)-(6.15) we can estimate the pmf of each node by the probabilities:

$$p_{i,l} = \frac{1}{J} \sum_{j=1}^{J} \frac{\Upsilon_l\left(x_{q_i}^{(j)}\right)}{\epsilon_l} \tag{6.16}$$

for J examples. Each term in the summation equals 1 if $x_{q_i}^{(j)} = \tilde{q}_l$ and zero otherwise. Thus similar to a histogram used for e.g. estimating probability mass functions, each term in the summation can be seen as a indicator function $I : \{\tilde{q}_1, \ldots, \tilde{q}_{2^b}\} \to \{0, 1\}$:

$$\frac{\Upsilon_l\left(x_{q_i}^{(j)}\right)}{\epsilon_l} = I_l\left(x_{q_i}^{(j)}\right) = \begin{cases} 1 & , x_{q_i}^{(j)} = \tilde{q}_l \\ 0 & , x_{q_i}^{(j)} \neq \tilde{q}_l \end{cases} \tag{6.17}$$

The entropy of node i can then be estimated by:

$$\hat{H}\left(X_{q_i}\right) = -\sum_{l=1}^{2^b} p_{i,l} \log\left(p_{i,l}\right) \tag{6.18}$$

As described in Section 4.2 the quantizer in the quantization layer is an uniform quantizer with stepsize Δ. In this setup, the normalization parameter can easily be calculated as a function of the stepsize.

To see this, consider first a b-bit quantizer with $b = 3$ and thus $2^b = 8$ quantization

levels. It follows that:

$$\epsilon_1 = \prod_{i=2}^{8} |\tilde{q}_i - \tilde{q}_1| = \prod_{i=1}^{7} |\tilde{q}_{i+1} - \tilde{q}_1| = \prod_{i=1}^{7} \Delta \cdot i = 7!\Delta^7 \tag{6.19}$$

$$\epsilon_2 = |\tilde{q}_1 - \tilde{q}_2| \prod_{i=3}^{8} |\tilde{q}_i - \tilde{q}_2| = \Delta \prod_{i=1}^{6} |\tilde{q}_{i+2} - \tilde{q}_2| = \Delta \prod_{i=1}^{6} \Delta \cdot i = \Delta 6!\Delta^6 \tag{6.20}$$

$$\vdots$$

$$\epsilon_4 = \prod_{i=1}^{3} |\tilde{q}_i - \tilde{q}_4| \prod_{i=1}^{4} |\tilde{q}_{i+4} - \tilde{q}_4| = \prod_{i=1}^{3} \Delta \cdot i \prod_{i=1}^{4} \Delta \cdot i = 3!\Delta^3 4!\Delta^4 \tag{6.21}$$

$$\epsilon_5 = \prod_{i=1}^{4} |\tilde{q}_i - \tilde{q}_5| \prod_{i=1}^{3} |\tilde{q}_{i+5} - \tilde{q}_5| = \prod_{i=1}^{4} \Delta \cdot i \prod_{i=1}^{3} \Delta \cdot i = 4!\Delta^4 3!\Delta^3 \tag{6.22}$$

$$\vdots$$

$$\epsilon_8 = \prod_{i=1}^{7} |\tilde{q}_i - \tilde{q}_1| = \prod_{i=1}^{7} \Delta \cdot i = 7!\Delta^7 \tag{6.23}$$

This can be generalized such that the normalization parameter for quantization level l in a b-bit quantizer is given as:

$$\epsilon_l = (l-1)!\Delta^{l-1} \left(2^b - l\right)!\Delta^{2^b - l} \tag{6.24}$$

As seen from the above equations the normalization parameters are not unique since $\epsilon_l = \epsilon_{2^b - l + 1}$. However the uniqueness is not needed since the key in equation (6.16) is that each term *only* equals 1 if $x_{q_i}^{(j)} = \tilde{q}_l$ and zero otherwise. Nevertheless the normalization parameter in (6.24) will be used.

The pmf estimates given by (6.16) coincides with the pmf estimates given by (6.9) for a 1-bit quantizer. To see this let $\tilde{q}_1 = 0$ and $\tilde{q}_2 = 1$ denote the quantization values for the 1-bit quantizer. It follows:

$$\epsilon_1 = |\tilde{q}_2 - \tilde{q}_1| = |1 - 0| = 1 \tag{6.25}$$

$$\epsilon_2 = |\tilde{q}_1 - \tilde{q}_2| = |0 - 1| = 1 \tag{6.26}$$

and further:

$$p_{i,2} = \frac{1}{J} \sum_{j=1}^{J} \frac{\Upsilon_2 \left(x_{q_i}^{(j)}\right)}{\epsilon_2} \tag{6.27a}$$

$$= \frac{1}{J} \sum_{j=1}^{J} \left|\tilde{q}_1 - x_{q_i}^{(j)}\right| \tag{6.27b}$$

$$= \frac{1}{J} \sum_{j=1}^{J} x_{q_i}^{(j)} \tag{6.27c}$$

$$p_{i,1} = \frac{1}{J}\sum_{j=1}^{J}\frac{\Upsilon_1\left(x_{q_i}^{(j)}\right)}{\epsilon_1} \tag{6.28a}$$

$$= \frac{1}{J}\sum_{j=1}^{J}\left|\tilde{q}_2 - x_{q_i}^{(j)}\right| \tag{6.28b}$$

$$= \frac{1}{J}\sum_{j=1}^{J}\left|1 - x_{q_i}^{(j)}\right| \tag{6.28c}$$

$$= \frac{1}{J}\sum_{j=1}^{J}1 - \frac{1}{J}\sum_{j=1}^{J}x_{q_i}^{(j)} \tag{6.28d}$$

$$= 1 - p_{i,2} \tag{6.28e}$$

Thus the pmf estimates given by (6.16) can be used to estimate the entropy of all nodes and for all b-bit quantizers. Hence the loss function in (6.8) can be replaced by:

$$\min \mathcal{L}_{b-bit} = \mathcal{L}_{MSE}\left(X, \hat{X}\right) + \beta\sum_{i=1}^{N_q}\hat{H}\left(x_{q_i}\right) \tag{6.29}$$

where $\hat{H}$ is given by equations (6.18).

In Figure 6.2 the estimated entropy of a node X_q quantized with different bits b and using our *relative distance estimator* is compared to the corresponding estimated entropy using histograms. From the figure it can be seen that for $b = 1, \ldots, 5$ the entropy estimate in (6.18) corresponds to the estimate obtained by using a histogram. For $b > 5$ the estimator experiences numerical issues due to the factorial and hence an approximation is needed in such cases.

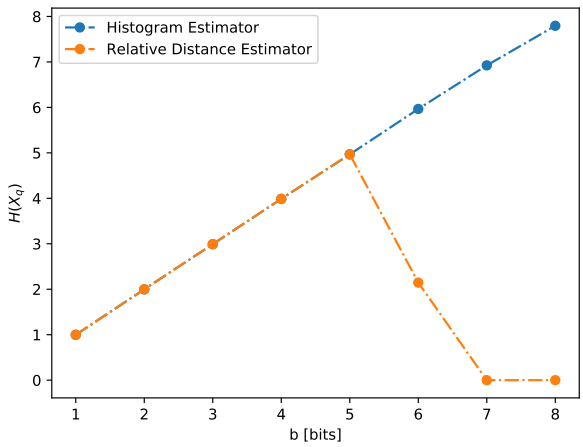


Figure 6.2: The estimated entropy of node X_q using our estimator (orange) and a histogram estimator (blue) for different b-bit quantizers.

II Part Two - Application

7 The Data 61
7.1 Generating Synthetic Data
7.2 TIMIT Speech Data

8 Encoding and Decoding Synthetic Data using DNNs 63
8.1 Network Structure
8.2 Results
8.3 Preliminary Discussion

9 Speech Coding using DNNs 81
9.1 Frame size of 16 samples
9.2 Frame size of 128 samples
9.3 Speech Coding using the Information Bottleneck Principle
9.4 Variable Rates

7. The Data

The purpose of this chapter is to present the data used as input for the autoencoders in Chapter 8 and Chapter 9.

7.1 Generating Synthetic Data

In Chapter 8 we will use synthetic data as input to DNN autoencoders. The synthetic data consist of realizations of a stochastic process $\{X_t\}_{t=1}^{\infty}$ where:

$$X_t = \rho X_{t-1} + W_t \quad , \quad t = 1, \ldots \tag{7.1}$$

and where $W_t \overset{i.i.d.}{\sim} \mathrm{N}(0, \sigma_w^2)$ is a white noise process, $X_1 \sim \mathrm{N}(0, \sigma_x^2)$ and $|\rho| < 1$. For $|\rho| < 1$ it follows that the process $\{X_t\}_{t=1}^{\infty}$ is a stationary *autoregressive (AR)* process of order 1 and that the variance is given as $\sigma_x^2 = \sigma_w^2/(1-\rho^2)$ (see e.g. [36]). We will refer to this process as an AR(1) process. The variance σ_w^2 is chosen such that $\sigma_x^2 = 1$, i.e. $\sigma_w^2 = 1 - \rho^2$.

In Chapter 8 we will consider different input/output strategies. Thus the following datasets are created:

- A dataset $\boldsymbol{X}_{AR} \in \mathbb{R}^{N \times T}$ containing N realizations of the AR(1) process with $t = 1, \ldots, T$.
- A dataset $\boldsymbol{W} \in \mathbb{R}^{N \times T}$ containing N realizations of the white noise process with $t = 1, \ldots, T$.
- A dataset $\boldsymbol{X}_{Res} \in \mathbb{R}^{N \times T}$ containing the residual version of $\boldsymbol{X}_{AR}$ obtained by predicting the AR(1) process. The residuals were obtained by subtracting the prediction $\hat{X}_t = \rho X_{t-1}$ from elements in $\boldsymbol{X}_{AR}$.
- A dataset $\boldsymbol{X}_{Mixed} \in \mathbb{R}^{N \times 2T}$ containing both $\boldsymbol{X}_{AR}$ and $\boldsymbol{X}_{Res}$. I.e. rows of $\boldsymbol{X}_{Res}$ concatenated with corresponding rows of $\boldsymbol{X}_{AR}$, such that each row contains realizations of the AR(1) process for $t = 1, \ldots, T$ and the corresponding residual.

An AR parameter of $\rho = 0.95$ was chosen and with $N = 100,000$ and $T = 10,000$, respectively. To exclude the transient part, only the last N_x columns of $\boldsymbol{X}_{AR}$ and

$\boldsymbol{W}$ are used and for creating $\boldsymbol{X}_{Res}$ and $\boldsymbol{X}_{Mixed}$. Thus the resulting dimensions of the datasets are as given above but with $T = N_x$. In relation to previous chapters and definitions we will refer to N_x as the frame size. The value of N_x is presented in Section 8.2.

The datasets contains all the data used for both training and testing, where 70% of the examples are used for training and the remaining 30% are used for testing.

An example of the AR(1) process in (7.1) with $\rho = 0.95$ and $\sigma_x = 1$, its prediction and residual can be seen in Figure 7.1.

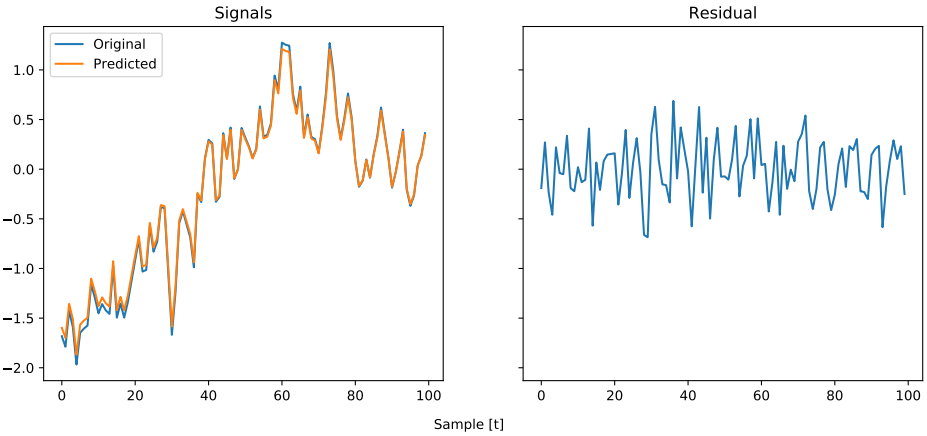


Figure 7.1: An example of the AR(1) process with $\rho = 0.95$ and $\sigma_x = 1$, its prediction and the corresponding residual.

7.2 TIMIT Speech Data

In this project we use frames of speech audio files from the TIMIT database [20] as inputs for our DNN speech autoencoder in Chapter 9.

The speech data in the TIMIT database was designed for acoustic-phonetic studies and for the development of automatic speech recognition [20]. The database was developed by the Defense Advanced Research Projects Agency - Information Science and Technology Office (DARPA-ISTO), Massachusetts Institute of Technology (MIT), Stanford Research Institute (SRI), Texas Instruments (TI) and the National Institute of Standards and Technology (NIST). TIMIT has been used in several DNNs involving speech such as speech recognition [23] [3] [45] [25] and speech enhancement [58] alongside development of new neural network schemes [50]. It is considered as a benchmark speech dataset [22, p. 461].

The database contains in total 6300 speech files of read sentences in American English from 630 different speakers. The speech files are sampled with a sampling frequency of 16 kHz, stored in the `wav`-file format and encoded using 16-bit PCM (see Section 2.1.1). Approximately 27% of the speech files are used for testing, while the remaining 73% are used for training.

8. Encoding and Decoding Synthetic Data using DNNs

The purpose of this chapter is to perform coding of the synthetic data described in Section 7.1 using DNN autoencoders. This is done in order to investigate how a neural network perform in a encoding and decoding setup. It is reasonable to assume that if DNNs do not work at all in an encoding and decoding setup for synthetic data, DNNs will not work for more complicated data such as speech. This chapter and its results should thus be seen as a proof of concept.

In Section 8.1 the structures of the DNN autoencoders examined in this chapter are described alongside different input/output strategies. Using the synthetic data from Section 7.1 enables one to compare the results with the rate-distortion function from Theorem 5.1 (see Section 5.3). This alongside other results are presented in Section 8.2.

8.1 Network Structure

The encoding and decoding hidden layers of the autoencoder in Figure 4.2 from Section 4.1 can be designed in several ways. In this chapter we utilize two different autoencoder structures; one with the frame size equal to the quantization dimension, i.e. $N_x = N_q$, and one where $N_x > N_q$, i.e. an undercomplete structure. Both network structures are fully connected feed forward DNNs (see Section 3.1) and are illustrated in Figure 8.1 and Figure 8.2.

As seen from the figures, there are 7 hidden layers in each of the two networks; 3 hidden layers in the encoding part, 1 hidden layer constituting the quantization layer and 3 hidden layers in the decoding part. The choice of hidden layers was based on initial experiments. As described in Section 4.2 a b-bit quantizer acting as an activation function is employed in the quantization layer. We will refer to networks with the structures in Figure 8.1 and Figure 8.2 as *balanced* and undercomplete networks, respectively. In the undercomplete network an increasing and decreasing amount of nodes is employed in the encoding and decoding hidden layers, respectively. This is done in order to help guide the network and to avoid an abrupt dimensionality reduction.

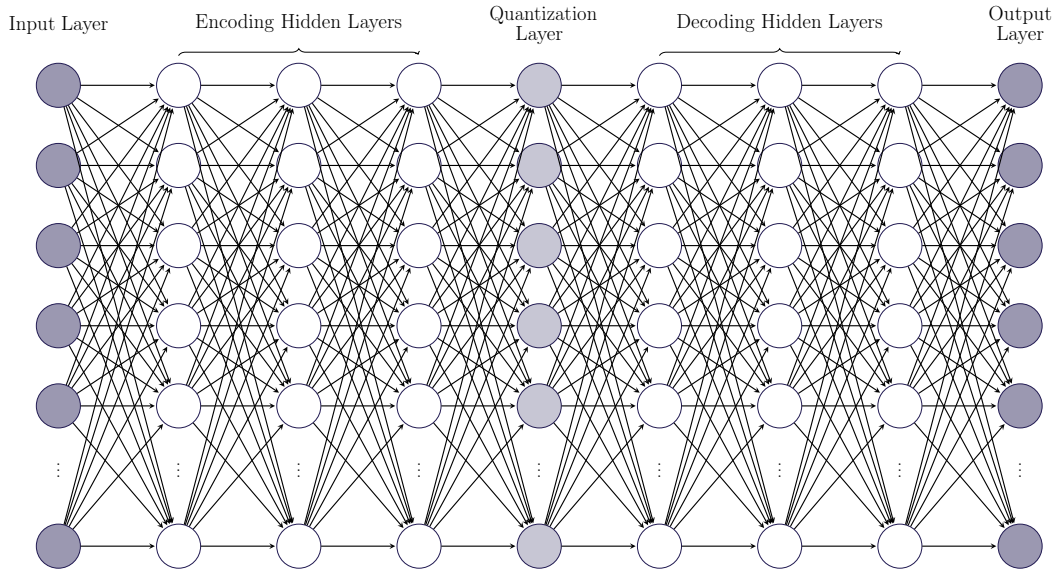


Figure 8.1: Network with a balanced structure, i.e. $N_x = N_q$. For simplicity the biases are here omittied.

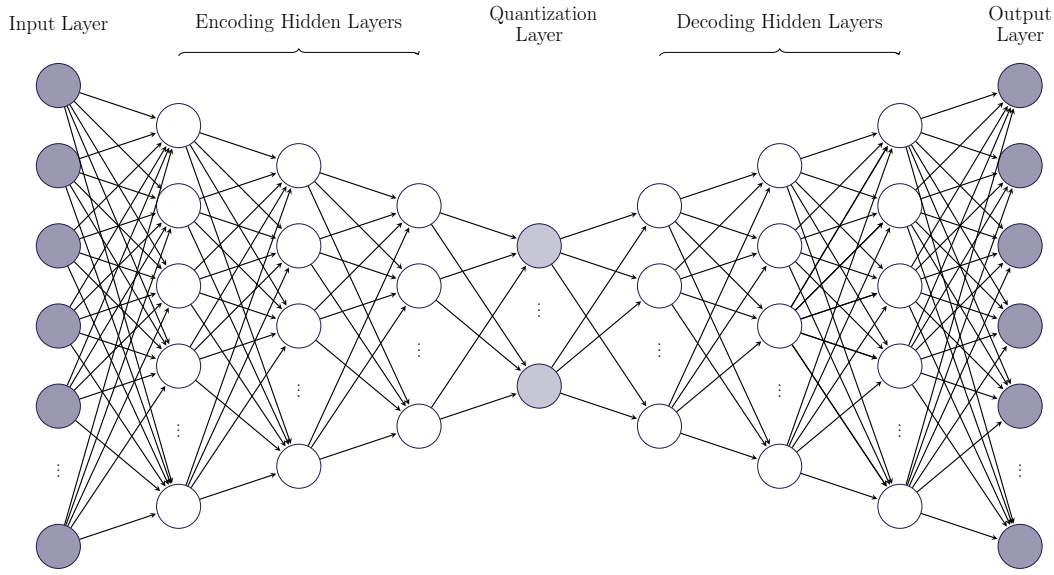


Figure 8.2: Network with uncercomplete structure, i.e. $N_x > N_q$. For simplicity the biases are here omittied.

In both figures, the amount of nodes in the input and output layers are illustrated to be the same. However this depends on the input and output types and will be described in the section below.

8.1.1 Input and Output Types

For the three datasets $\boldsymbol{X}_{AR}, \boldsymbol{X}_{Res}$ and $\boldsymbol{X}_{Mixed}$ described in Section 7.1 and the two network structures from Figures 8.1-8.2, five different input/output strategies are examined. In this section the different input/output strategies are described and illustrated with the balanced network structure. In the figures below, the number of nodes in each layer does not reflect the actual dimension of the layers. The figures serve to illustrate the concept of the different input/output strategies and the dimension of the layers should be seen relative to each other.

AR Process as both Input and Output

The simplest input/output strategy is shown in Figure 8.3. The input to this network is a frame of size N_x of the AR(1) process from Section 7.1, i.e. a row of $\boldsymbol{X}_{AR}$. The output of the network is a reconstructed version of the input. The task of the network is thus to encode and decode the AR(1) process.

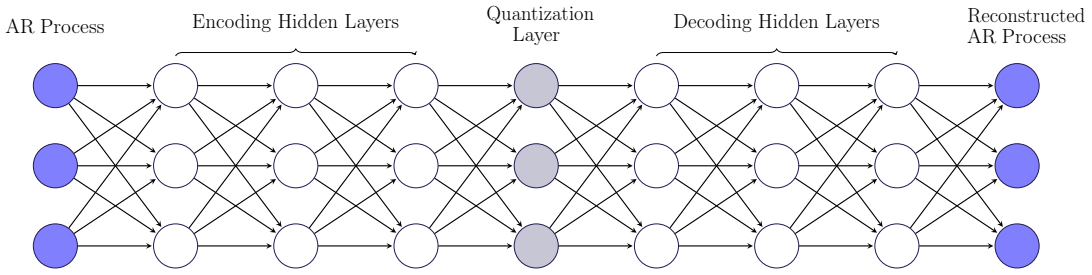


Figure 8.3: Neural network with the AR(1) process as input and output. For simplicity the biases are here omittied.

Residual as both Input and Output

Instead of giving a frame of size N_x of the AR(1) process as input to the network, the idea is to give the corresponding residual as input in the strategy of Figure 8.4. I.e. the input to the network is a row of $\boldsymbol{X}_{Res}$. The task of the network is thus to encode and decode the residual. The AR(1) process is then reconstructed from the residual separately. This idea is inspired by LPC coding briefly described in Section 2.1.2. However compared to the network in Figure 8.3, one have to keep in mind that in order to reconstruct the AR(1) process, the AR parameter, ρ, is needed. Using this input/output strategy, we assume that the AR parameter is encoded, transmitted and perfectly reconstructed using an external processes. Thus the bit rate for this strategy is higher than the bit rate for the strategy in Figure 8.3.

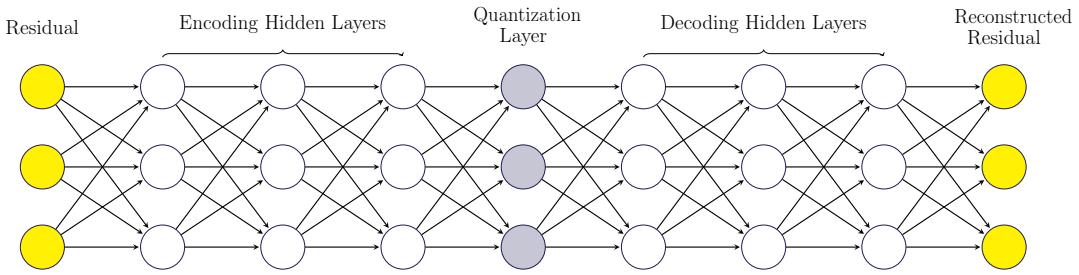


Figure 8.4: Neural network with the residual as input and output. For simplicity the biases are here omittied.

Residual as Input and Reconstructed AR Process as Output

Instead of considering only the AR(1) process or its residual as both input and output, the strategy shown in Figure 8.5 is a combination of the strategies in Figure 8.3 and 8.4. The idea is to give the residual process as input and let the network reconstruct the AR(1) process. The task is therefore different from the previous strategies, however the overall goal is still to reconstruct the AR(1) process. Compared to the strategy in Figure 8.4 the AR parameter is not needed to be transmitted separately.

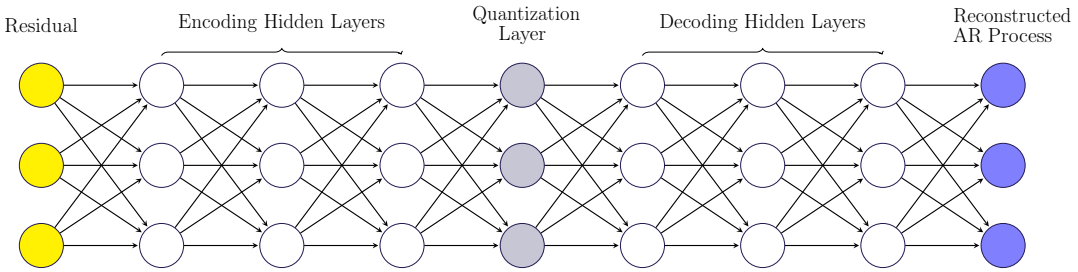


Figure 8.5: Neural network with the residul as input and the reconstructed AR(1) process as output. For simplicity the biases are here omittied.

Residual as Input, Reconstructed AR Process as Output, AR Parameter Given Additionally

Instead of letting the network reconstruct the AR(1) process from only a quantized version of the residual, the strategy in Figure 8.6 involves giving the AR parameter as additional input to the decoding part of the network. This still means that the AR parameter is assumed to be encoded and transmitted separately and thus the bit rate is higher as with the strategy in Figure 8.4. The hope is that by giving the AR parameter to the network, the network can easier discover the relation between the input and the output.

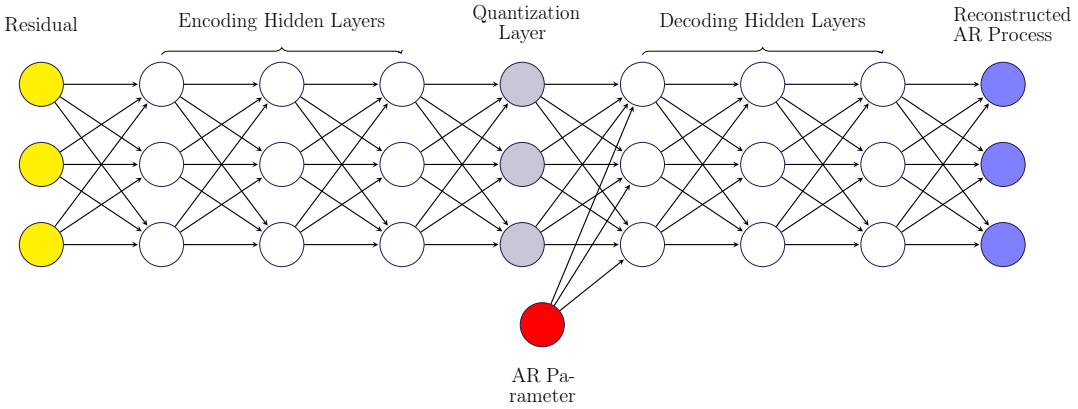


Figure 8.6: Neural network with the residul as input and the reconstructed AR(1) process as output. The AR parameter is encoded seperately using external processes and is given to the network in the decoding part for reconstruction of the AR(1) process. For simplicity the biases are here omittied.

Residual and AR Process as Input with the AR Process as Output

The previous strategies involve giving the network either the AR(1) process or residual as input and possibly additionally the AR parameter. Figure 8.7 illustrates a strategy where the network gets all the data and information known about the signal. By giving both the AR(1) process and residual as input to the network, the hope is that the network is more guided in terms of discovering relations between the input and output relevant for reconstructing the AR(1) process. Since the autoencoders are data-driven, it indeed seems reasonable to provide all possible data to the network. If we disregard the AR parameter an advantage of this strategy is that the bit rate is not effected even though more data is provided as input.
Since the input dimension is twice as large as the number of nodes in the output layer, a decreasing amount of nodes in the encoding part is employed. Thus using this strategy, the network can not be regarded as a balanced network entirely. The decreasing amount of nodes is employed in order to guide the network, so that the additional information given at the input is not lost in the first hidden layer due to an abrupt dimensionality reduction.

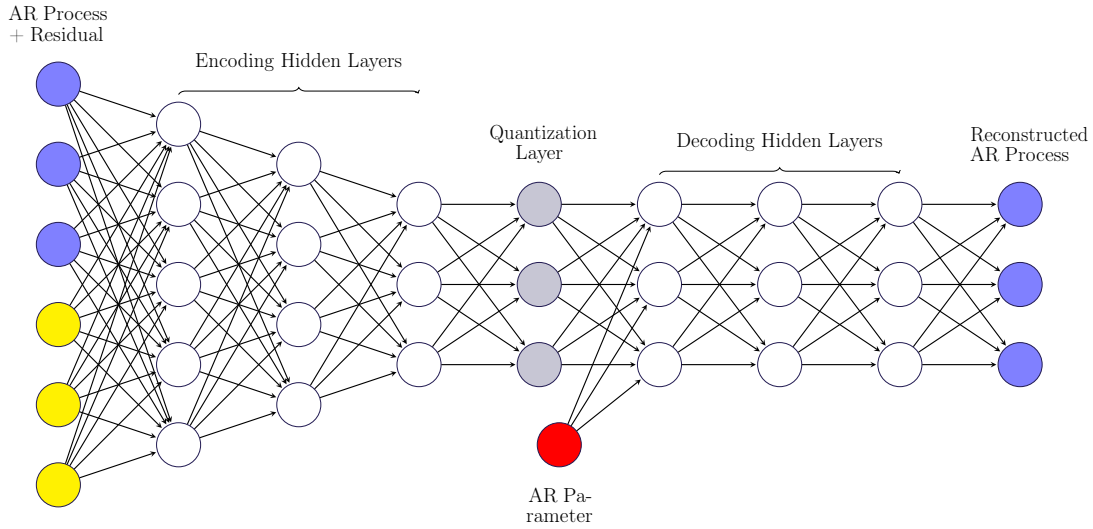


Figure 8.7: Neural network with both the residual and AR process as input and with the AR process as output. The AR parameter is encoded seperately using standard techniques but is given to the network in the decoding part for reconstruction of the AR process. For simplicity the biases are here omittied.

The above input/output strategies are also examined with the undercomplete structure. The exact amount of nodes in each layer is described in Section 8.2.

8.2 Results

The results of the experiments with the two network structures and five input/output strategies are presented in this section.

The number of nodes in the balanced network structure are kept the same for each layer and thus the dimension of each layer is solely dependent on the frame size N_x of the signal. In this chapter we consider only frame sizes of $8, 16$ and 32 samples. If we consider the AR(1) process in Section 7.1 to be a signal "sampled" with a sampling frequency of 16 kHz as the TIMIT database (see Section 7.2), this corresponds to frame sizes of 0.5 ms, 1 ms and 2 ms.

The choice of the number of nodes in the undercomplete structure is not immediately obvious. In the undercomplete strategy the number of nodes must be decreased and increased in such a manner, that the networks are comparable for different frame sizes. The dimensions of the hidden layers including the quantization layer are shown in Table 8.1 for the frame sizes $N_x = 8, 16, 32$.

Frame Size, N_x	8	16	32
Dimension of Encoding Hidden Layers, $[N_{e1}, N_{e2}, N_{e3}]$	[6,4,2]	[12,8,4]	[24,16,8]
Dimension of Quantization Layer, N_q	1	2	4
Dimension of Decoding Hidden Layers, $[N_{d1}, N_{d2}, N_{d3}]$	[2,4,6]	[4,8,12]	[8,16,24]

Table 8.1: The dimension of the layers in the undercomplete structure for different frame sizes.

In Table 8.1 $[N_{e1}, N_{e2}, N_{e3}]$ and $[N_{d1}, N_{d2}, N_{d3}]$ refer to the dimensions of the encoding and decoding hidden layers, respectively. Thus N_{e1} and N_{d1} refer to the dimension of the first hidden layer in the encoding and decoding part, respectively. From Table 8.1 it can be seen that the dimension of the quantization layer is $1, 2$ and 4 for frame sizes of $8, 16$ and 32, respectively. The dimension of the quantization layer is chosen such that the ratio between the frame size and the dimension of the quantization layer is the same for all frame sizes. This ensures that for a fixed b in the b-bit quantizer, the same bit rates are achieved for the different frame sizes. The dimensions of the encoding and decoding hidden layers are chosen such that the relative decrease and increase from one layer to another is the same. By choosing the layer dimensions in this way, the networks are comparable even though the frame sizes are different since they are "scaled" in the same manner.
Regarding the input strategy where both the AR(1) process and residual are given as input, the case is a bit more complicated. For a frame size of 8 one can not directly adapt the layer dimensions used for $N_x = 16$ in Table 8.1, since this will affect the bit rate. For this reason we chose to use the same layer dimensions as for the other input types when considering the undercomplete structure. For the balanced network and the input strategy where both the AR process and residual are given as input, the dimensions of the encoding layers N_{e1} and N_{e2} are the same as given in Table 8.1 to avoid an abrupt dimensionality reduction.

To keep the amount of tunable parameters to a minimum and to focus the examination on how well the network performs for different bit rates, all the networks

were trained with the same training setup, i.e. the networks were trained using the same training parameters, training algorithm, amount of epochs, batch size and initial weights. The value of the parameters are based on initial experiments. The networks were all trained using the Adam algorithm (see Section 3.5.2) with default parameters and with the MSE (see Section 3.5.1) as a loss function. initial experiments showed that the networks obtained significantly better results in terms of lower training and test losses with the Adam algorithm compared to using SGD (see Section 3.5.2). The networks were all trained for 300 epochs and with a batch size of 100.

Initial experiments showed that the networks obtained better results with the weights matrices initialized as identity matrices. Thus a weight matrix $\boldsymbol{W}$ associated with two layers of same dimension N is initialized as:

$$\boldsymbol{W} = \boldsymbol{I}_N \tag{8.1}$$

where $\boldsymbol{I}_N$ is the $N \times N$ identity matrix. Weight matrices associated with layers of different dimensions are initialized in a similar manner. A weight matrix $\boldsymbol{W} \in \mathbb{R}^{N_{l+1} \times N_l}$ associated with layer l of dimension N_l and the succeeding layer $l+1$ with dimension N_{l+1} is initialized as:

$$\boldsymbol{W} = \begin{cases} \begin{bmatrix} \boldsymbol{I}_{N_{l+1}} & \mathbf{0}_{N_{l+1},N_1-N_{l+1}} \end{bmatrix} & N_{l+1} < N_l \\ \begin{bmatrix} \boldsymbol{I}_{N_l} \\ \mathbf{0}_{N_{l+1}-N_l,N_l} \end{bmatrix} & N_{l+1} > N_l \end{cases} \tag{8.2}$$

where $\mathbf{0}_{N_{l+1},N_1-N_{l+1}}$ is a zero matrix of dimension $N_{l+1} \times N_1 - N_{l+1}$. The initialization in (8.2) means that some nodes are not connected at all to a layer, but initial experiments showed that this did not have a negative impact on the final performance. The biases are initialized with zeros as is standard in Tensorflow.

The ELU activation function was used for all encoding and decoding hidden layers. The identity function was used as an activation function for the input and output layer.

8.2.1 General Performance

Recall from Section 4.2 that the bit rate in bits per sample for the autoencoder networks is given as:

$$R = \frac{b \cdot N_q}{N_x} \tag{8.3}$$

for a b-bit quantizer, a quantization dimension of N_q and a frame size of N_x. Using this bit rate and MSE loss function from Section 3.5.1 we can obtain an operational rate-distortion curve for a network and compare it to the rate-distortion function from Theorem 5.1:

$$R(D) = \frac{1}{2} \log \left(\alpha^2 + \frac{\sigma_\epsilon^2}{D} \right) \tag{8.4}$$

where $\alpha = \rho = 0.95$ and $\sigma_\epsilon^2 = \sigma_w^2 = 1 - \rho^2$in our case.

In the following we examine the obtained operational rate-distortion curves for the two different network structures and five different input/output strategies presented in Section 8.1. We emphasize that the bit rates shown in the rate-distortion curves below are the bit rates from (8.3), i.e. the bit rate does not include additional encoding such as the encoding of the AR parameter. From a information theoretic perspective the obtained rate-distortion curves are comparable across all input/output strategies, since the AR parameter is the same for all inputs. The results obtained from both the training and test datasets are similar and we only show the results from the test dataset since this better explains how the networks would work in practise on unseen data.

In the following we examine networks with b-bit quantizers where $b = \{1, \ldots, 8\}$.

Balanced Network

From (8.3) it follows that $R = b$ for the balanced networks. The theoretical rate-distortion in (8.4) and the obtained operational rate-distortion curves for the different input/output strategies with a frame size of 8 can be seen in Figure 8.8.

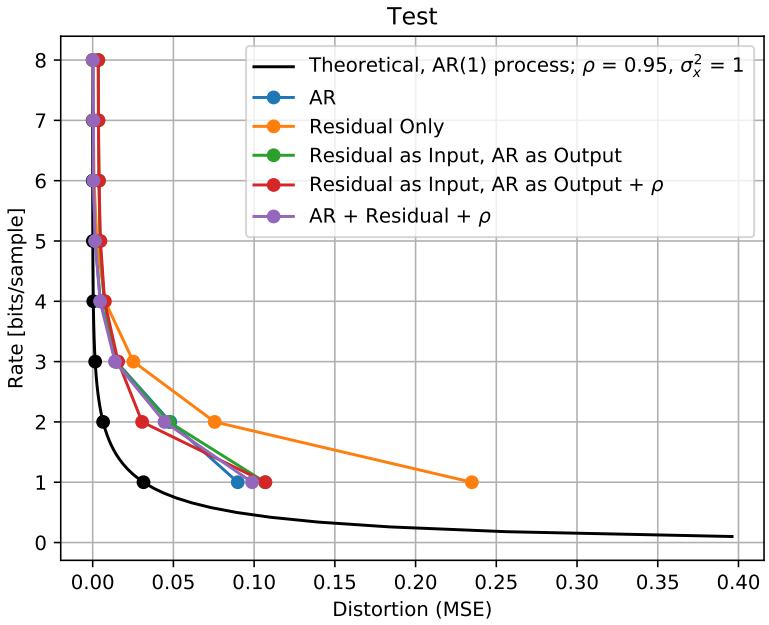


Figure 8.8: Operational rate-distortion curves for the balanced network with different input/output strategies and a frame size of 8.

As seen in Figure 8.8 and as expected the distortion decreases as the bit rate increase. Using the residual both as input and output seems to perform worse for the lower bit rates than other input/output strategies. In general there seem to be no difference in the performance among the remaining input/output strategies. Similar results are obtained using frame sizes of 16 and 32 and the corresponding rate-distortion curves can be seen in Figure A.1 and A.2 in Appendix A.1. However the larger frame

sizes experience some variations in the distortions for some input/output strategies. Furthermore the rate-distortion curves for the networks are in general closer to the theoretical bound for larger frame sizes. One can not necessarily deduce that it is better to use larger frame sizes. The networks are more prone to overfitting for larger frame sizes due to the quantization layer being a vector quantizer and a decreasing amount of training examples per dimension for an increasing frame size.
To get an understanding of how the networks improve as the bit rate increases, we examine the signal-to-reconstruction error ratio (SRR) . We define the SRR in dB as:

$$SRR_{dB} := 10 \log_{10} \left(\frac{\sigma_x^2}{D} \right) \tag{8.5}$$

for distortion D and variance of the AR process σ_x^2.

In Figure 8.9 the SRR as a function of bit rate can be seen for the different input/output strategies and a frame size of 8.

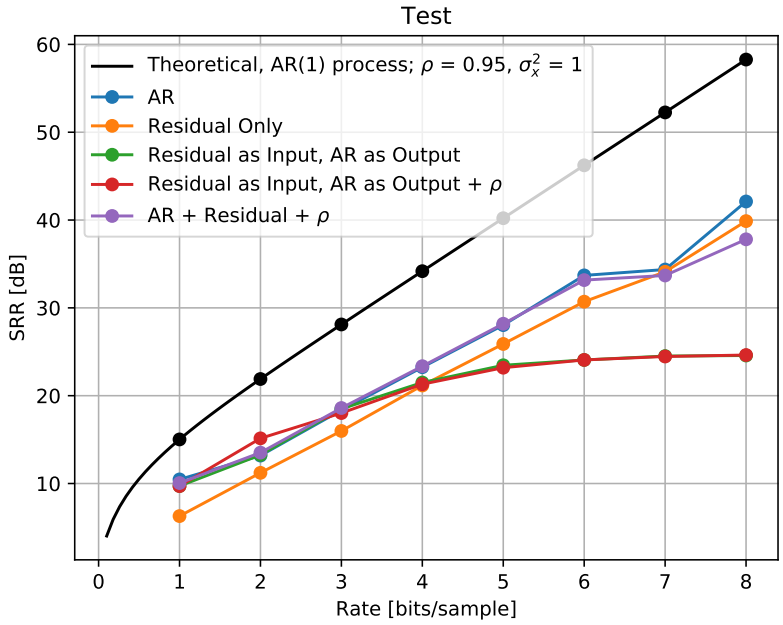


Figure 8.9: SNR and bit rates for the balanced network with different input/output strategies and a frame size of 8.

Even though the different input/output strategies seem to perform similarly for higher bit rates in Figure 8.8, Figure 8.9 shows that this is indeed not the case. For the input/output strategies involving only the residual as input and AR process as output, i.e. the red and green curves in the figure, the SRR does not improve for higher bit rates. For instance the SRR is approximately the same for bit rates of 5 and 8. The SRR for the remaining input/output strategies increases as the bit rate increases. However the distance to the theoretical SRR curve also increases slightly for higher bit rates. Thus in general it seems that the improvement on the distortion by using higher bit rates stagnates across all input/output strategies.

Similar results as above can be seen for frame sizes of 16 and 32 in Figure A.3 and A.4 in Appendix A.1, respectively. The variation in the distortion for some input/outputs strategies seen from Figures A.1 and A.2 is also present in Figure A.3 and A.4.

The fact that the improvement on the distortion stagnates for higher bit rates is also visible when examining the difference in bits. Let R be the bit rate used for a neural network, e.g. the bit rates obtained in Figure 8.8, and let D be the corresponding MSE-distortion obtained by using this bit rate. Let further $R(D)$ be the theoretical rate for the AR(1) process and distortion D. The difference in bits is obtained by:

$$R - R(D) \tag{8.6}$$

Thus the difference in bits is an indicator of how many more bits per sample the network uses compared to the theoretical bound for the same distortion. The difference in bits and the bit rates obtained by the balanced network for different input/output strategies and a frame size of 8 is shown in Figure 8.10.

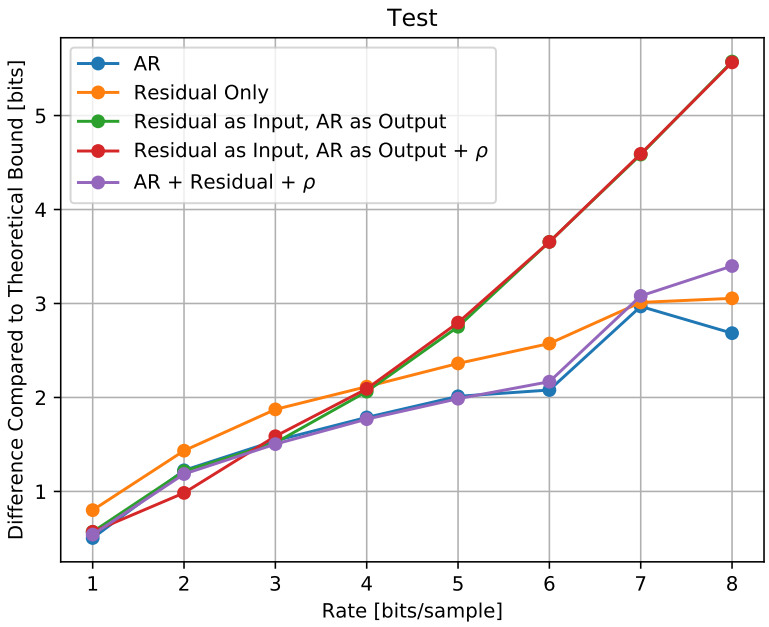


Figure 8.10: Difference in bits and the bit rate for the balanced network with different input/output strategies and a frame size of 8.

From Figure 8.10 it can be seen that the difference in bits increases as the bit rate increases. For a bit rate of 1, the difference in bits is below 1 bit per sample for all input/output strategies. For the higher bit rates the difference in bits is between 2.5 and 5.5 bits per sample. Figure 8.10 and Figure 8.9 indicates that there is a difference among the input/output strategies and the networks ability to improve the distortion for higher bit rates and furthermore that the effect of this improvement decreases for higher bit rates. Similar results can be seen for frame sizes of 16 and 32 in Figure A.5 and A.6 in Appendix A.1.

Undercomplete Network

From Table 8.1 and (8.3) bit rates $R < 1$ can be obtained using the undercomplete network. The theoretical rate-distortion curve for the AR(1) process and the operational rate-distortion curve for different input/output strategies with a frame size of 8 and bit rates $R = \frac{b}{8}, b = 1, 2, \ldots, 8$ can be seen in Figure 8.11.

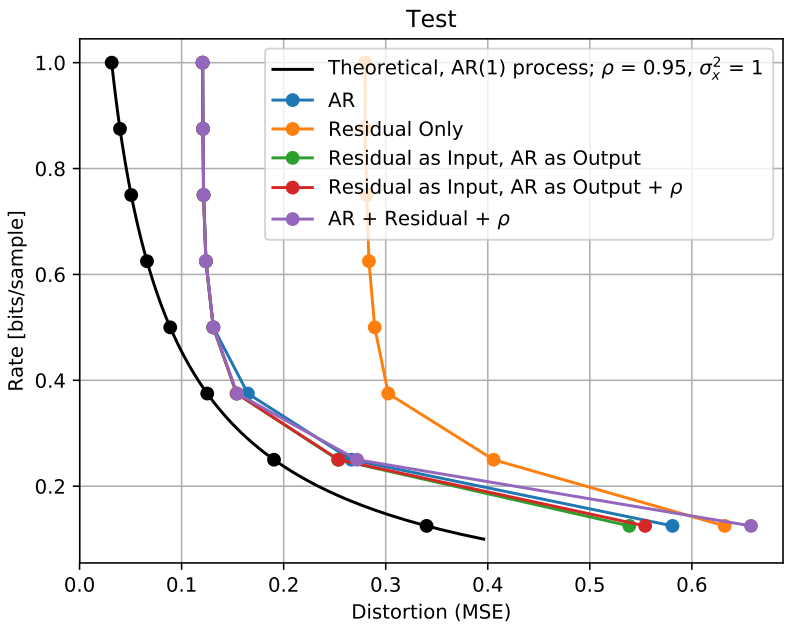


Figure 8.11: Operational rate-distortion curves for the undercomplete network with different input/output strategies and a frame size of 8.

From Figure 8.11 it can be seen that the different input/output strategies perform similarly for a bit rate of 0.125. However as the bit rate increases, there is a clear difference between the performance of the input/output strategy involving only the residual and the remaining strategies. The strategy involving only the residual as input and output performs significantly worse than the remaining strategies. However this does not seem surprising. Remember that the undercomplete network structure implies $N_q < N_x$ for dimension of the quantization layer N_q and frame size N_x, i.e. the networks perform a dimensionality reduction. Since the residual is white noise, i.e. the samples are independent and identically distributed, it is not possible to reduce the dimension without losing the information of some samples completely.
Similar results can be seen for frame sizes of 16 and 32 in Figure A.9 and A.10 in Appendix A.2. However for the frame sizes of 16 and 32 the networks perform better than the theoretical limit for some bit rates, which could imply overfitting. On the other hand the input strategy involving only the residual as input and output performs even worse for higher frame sizes. This is probably due to the dimensionality reduction being more abrupt for a frame size of 32 and a quantization dimension of 4 than for a frame size of 16 and a quantization dimension of 2.

Figure 8.11 suggest that there is no improvement on the distortion for higher bit rates. This can also be seen on the SRR in Figure 8.12. From the figure it can be seen that the SRR is approximately the same for bit rates higher than 0.4. The corresponding SRR plots for a frame sizes of 16 and 32 can be seen in Figure A.11 and A.12 in Appendix A.2, respectively. The stagnation of the SRR is also present in these figures, however there are some variations among some input/output strategies.

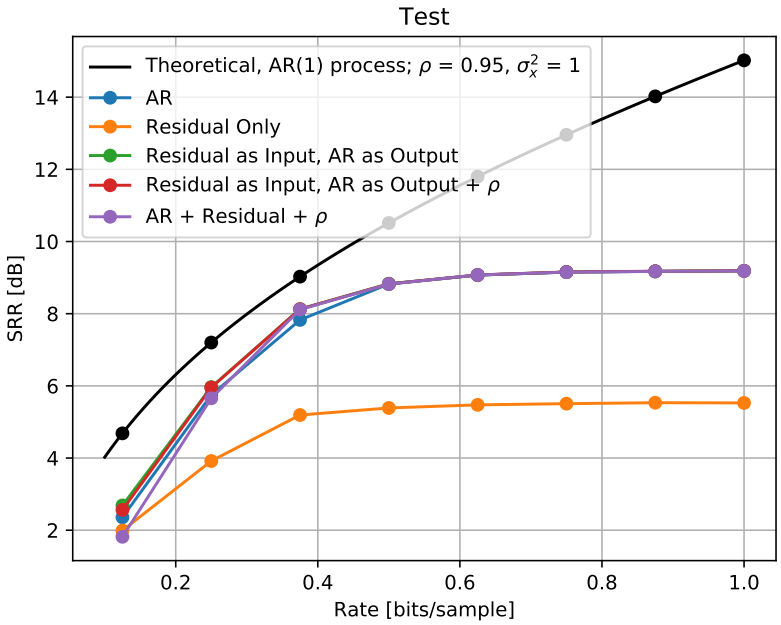


Figure 8.12: SRR and bit rates for the undercomplete network with different input/output strategies and a frame size of 8.

Examining the difference in bits in Figure 8.13 it is also visible that the networks are not able to improve the distortion for higher bit rates. It is seen that the difference grows linearly. The corresponding figures for frame sizes of 16 and 32 can be seen in Figure A.13 and A.14 in Appendix A.2, respectively.

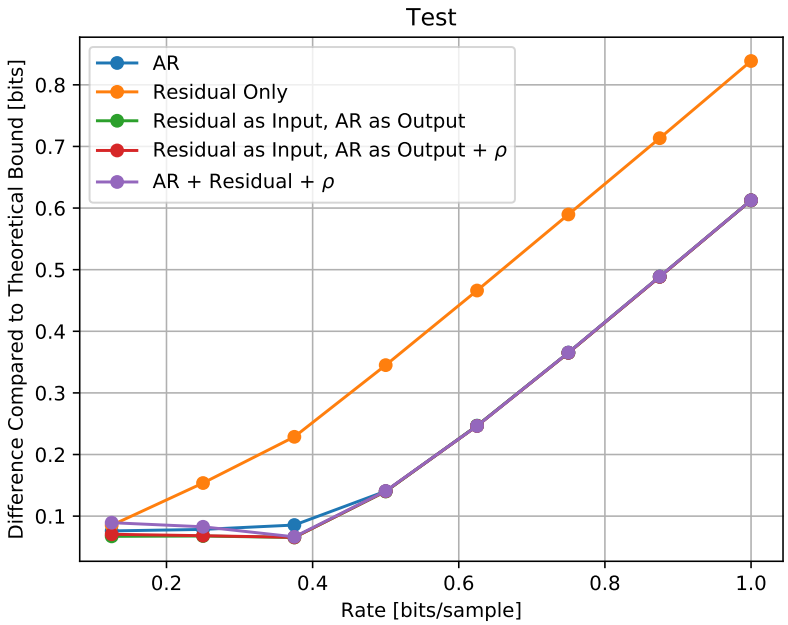


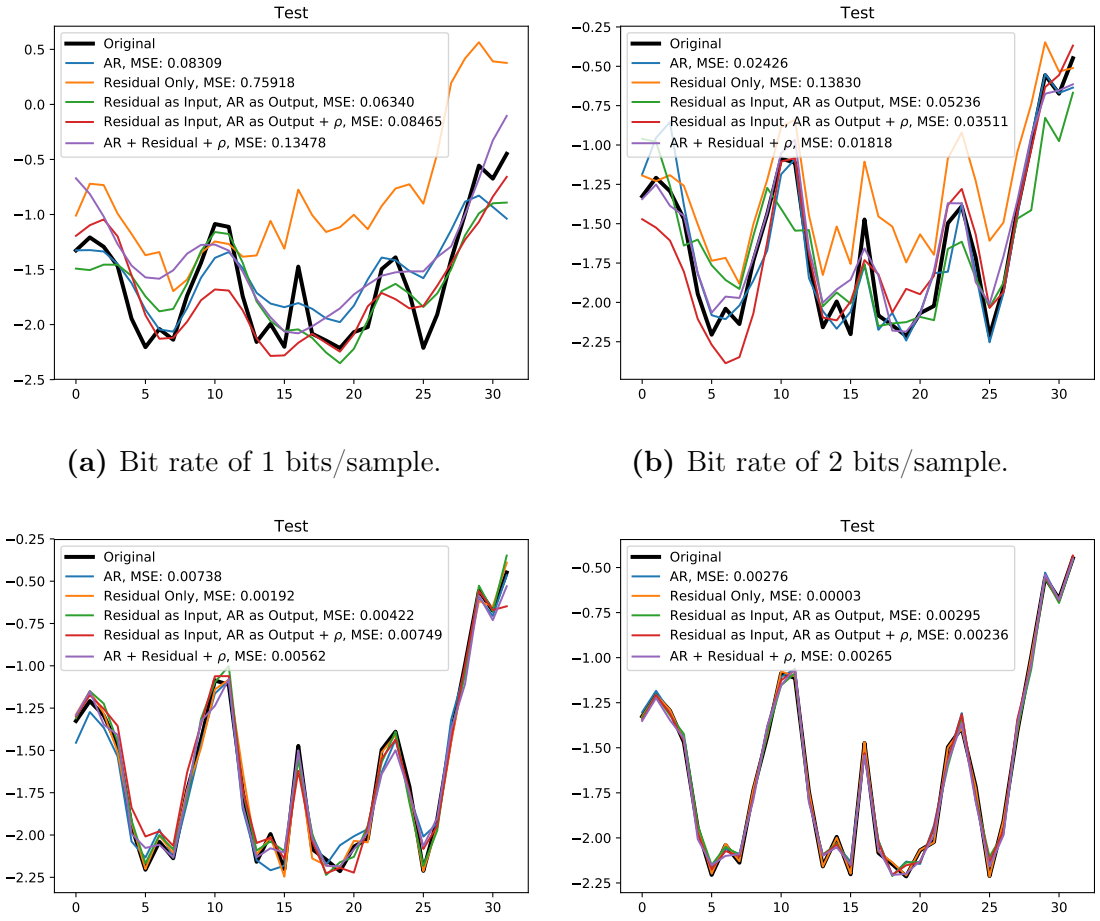
Figure 8.13: Difference in bits and the bit rate for the undercomplete network structure and a frame size of 8.

All the above obtained results for both the balanced and undercomplete networks suggest that the networks are not able to utilize the extra bits per sample given by a higher bit rate to obtain a significantly smaller distortion.

8.2.2 Analysis of the Reconstruction

In this section we examine the obtained reconstructed versions of the AR(1) process. The results are shown here for a single reconstructed signal. The corresponding figures for other signals can be seen in Appendix A.1 and Appendix A.2 for the balanced and undercomplete networks, respectively.

Figure 8.14 shows a reconstructed signal for the balanced network and a frame size of 32 for different bit rates and different input/output strategies alongside the original signal. The MSE values indicate the mean squared error between the shown original and reconstructed signal.



(a) Bit rate of 1 bits/sample.

(b) Bit rate of 2 bits/sample.

(c) Bit rate of 4 bits/sample.

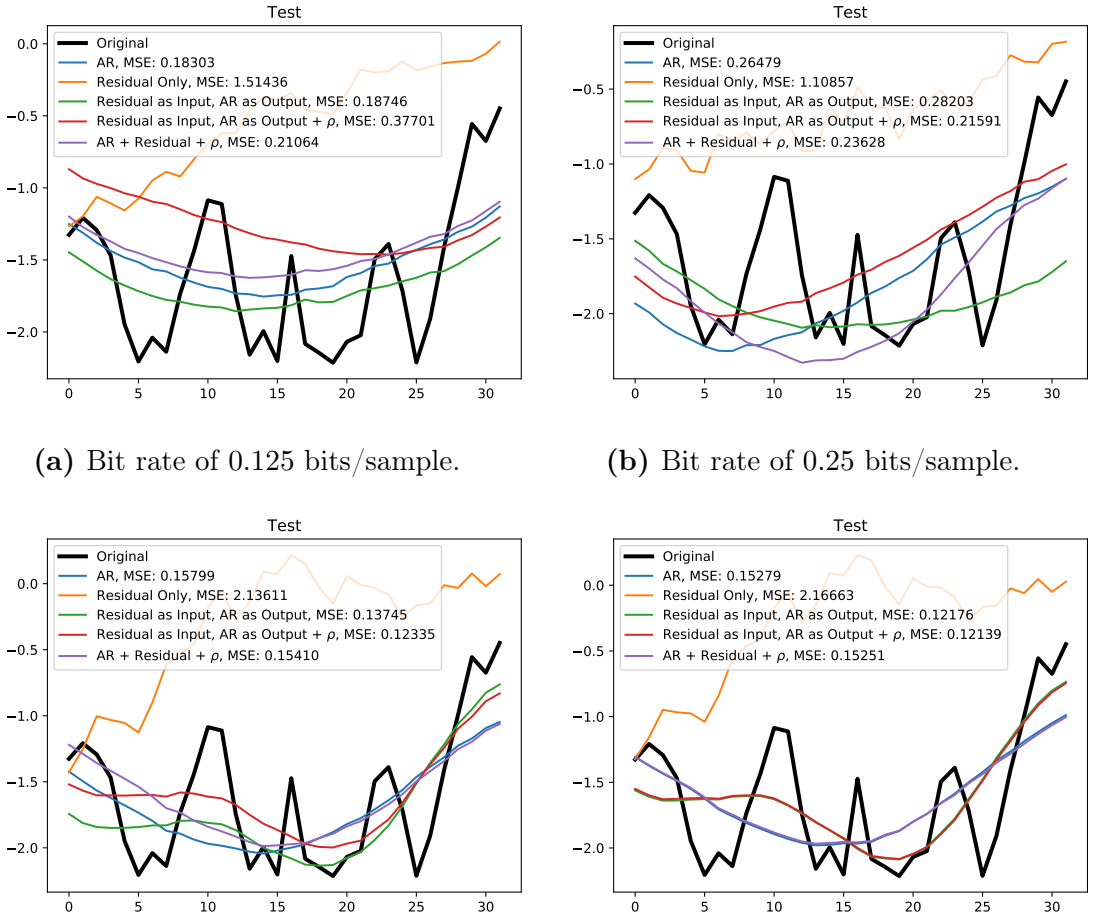
(d) Bit rate of 8 bits/sample.

Figure 8.14: Reconstructed signals and the original signal for different input/output strategies and bit rates in the balanced network. The MSE values are calculated for the shown original and reconstructed signals.

As seen from the figure, the reconstruction becomes better for higher bit rates, as expected. However considering that the reconstructions in Figure 8.14d is obtained by using twice the number of bits per sample as in Figure 8.14c, the improvement is, although present, not significant. This can also be seen on the MSE values. As

expected from the rate-distortion curve in Figure A.2, the input/output involving the residual input and output obtains poor reconstructions for the low bit rates. However as discussed before this strategy is able to compete with the remaining strategies for the higher bit rates. Reconstruction-wise there seem to be no significant difference between the remaining strategies, e.g. there seem to be no significant difference between using only the AR(1) process as input and giving the network all possible information available. This is especially the case when using a bit rate of 8.

Figure 8.15 shows the reconstruction of the same signal as above, but for the undercomplete network and its different bit rates.



(a) Bit rate of 0.125 bits/sample.

(b) Bit rate of 0.25 bits/sample.

(c) Bit rate of 0.5 bits/sample.

(d) Bit rate of 1 bits/sample.

Figure 8.15: Reconstructed signals and the original signal for different input/output strategies and bit rates in the undercomplete network.

As seen from the figure, the reconstructed signals do not resemble the original signal. Furthermore there seem to be only a small improvement for the higher bit rates. The input/output strategy involving using only the residual as input and output performs worst. It is seen that the dimensionality reduction of the independent and identically distributed white noise samples affects the reconstruction in such a manner that for e.g. a bit rate of 0.25 none of the reconstructed signal samples

are near the original signal samples. In general using the MSE as a loss function together with the undercomplete network obtains "averaged" reconstructed signals in the sense that the reconstructed signals does not even capture the movement of the original signal. Additional experiments showed that undercomplete structure itself is inadequate since poor reconstructions were also obtained by using the undercomplete structure but without the quantizer.

8.2.3 Dimension Versus Bits

Comparing Figure 8.14a with Figure 8.15d it is seen that different results are obtained for the balanced and undercomplete networks even though the bit rate is the same. This suggests that there is a relation between the distortion and values b and N_q defining the quantization layer. To investigate this, the balanced network was tested with different configurations of the quantization layer. Recall once again that the bit rate R in bits/sample is given as:

$$R = \frac{N_q \cdot b}{N_x} \tag{8.7}$$

for a b-bit quantizer, a quantization dimension of N_q and a frame size of N_x.
For a given bit rate and a frame size of $N_x = 8$ the network was trained with different conflagrations of N_q and b such that the rate was achieved. E.g for a bit rate and frame size of 8, then $N_q = 1$ and $b = 8$ and further $N_q = 8$ and $b = 1$ are two possible configurations. The ratio between the bits per node in the quantization layer and the number of nodes in the quantization layer in relation to the frame size N_x is given as:

$$\frac{b}{N_q \cdot N_x} \tag{8.8}$$

For $N_q \cdot N_x > b$ the term above is in the range $[0, 1]$. A value close to zero in (8.8) indicates a large quantization layer with few bits used per node, while a value of 1 indicates a quantization layer with few nodes but a large amount of bits per node. Thus the ratio can be seen as a way of describing the relationship between encoding a lot of information, i.e. using more nodes in the quantization layer, and the quality of the encoding, i.e. the amount of bits used per node.

For bit rates $R = \{1, 2, 4, 6, 8\}$ and a frame size of 8, the balanced network was trained with the AR(1) process as input and with the different configurations of b and N_q achieving these rates, where $b \in \{1, \ldots, 8\}$. The relationship between the obtained distortion and the ratio in (8.8) can be seen in Figure 8.16 for the different bit rates. The points indicated by a black-edged square are the points where the ratio corresponds to the configurations of b and N_q used to obtain the rates of the balanced network in Section 8.2.1.

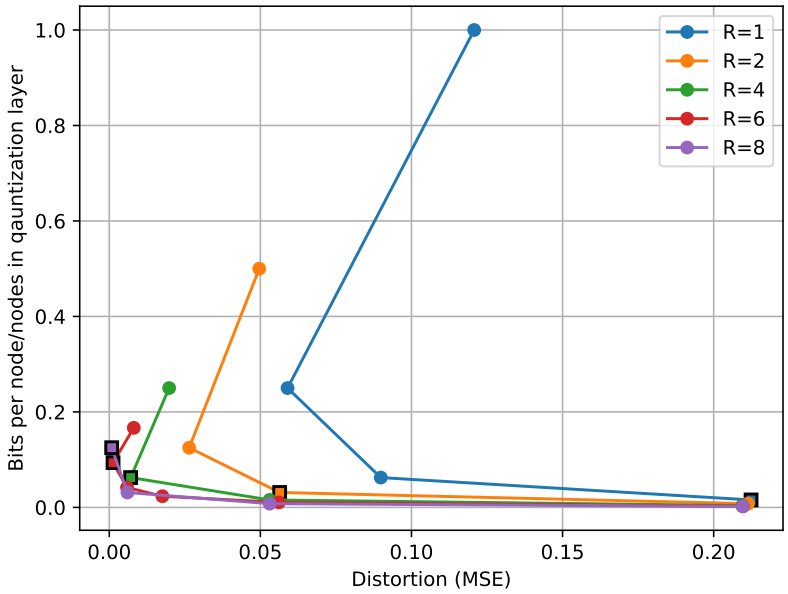


Figure 8.16: The relationship between the ratio in (8.8) and the obtained distortion for different rates. Black-edged square point are the points where the ratio corresponds to the configurations of b and N_q used to obtain the rates of the balanced network in Section 8.2.1

As seen in the figure, the distortion decreases as the ratio increases, i.e. a lower distortion is achieved when more bits are used in the quantization layer relative to the dimension of the layer. However at some point the distortion increases for all bit rates except $R = 8$. Thus there is a trade-off between the way we chose to encode a signal in the quantization layer and the achieved distortion. The decrease of distortion suggests that it is better to use fewer nodes but improve the quality of the encoding in these nodes by using more bits. On the other hand it is also naive to assume that we can encode an entire signal of frame size N_x successfully by only using e.g. a single node and b bits. From the figure it is furthermore seen that highest distortion is achieved by using the highest amount of nodes possible in the quantization layer and lowest amount of bits without violating the rate. Thus even though it is naive to assume that we can encode an entire signal of size N_x by only using one node, the achieved distortion is still smaller than the distortion achieved by using lots of nodes. It is also worth noticing that for some rates, lower distortions are achieved by using a ratio between b and N_q different from the ratio used to achieve the rate-distortion curves in Figure 8.8. Thus it is possible to achieve better rate-distortion curves than the ones shown in Section 8.2.1.

8.3 Preliminary Discussion

The above obtained results for the balanced network and the undercomplete network all suggest that the networks are not able to utilize the extra bits per sample given by a higher bit rate to obtain a significantly smaller distortion. In the context of digital coding this does not necessarily have to be a problem since we aim to achieve

the lowest possible bit rate. However since this problem is prevalent among all input/output strategies and network structures it suggest an inadequacy of e.g. the network structures. This is especially true for the undercomplete network structure. Thus if a bit rate $R < 1$ is desired, a different network architecture is needed.
Comparing the reconstructed signals with the original also revealed that it is in general difficult to obtain reconstructions similar to the original signal when using low bit rates and the MSE as a loss function.
From all the above experiments it was seen that there seemed to be no significant difference between using only the AR(1) process as both input and the remaining input/output strategies. In fact the small variations among the input/output structures from the figures shown in this chapter and in Appendix A reveal that using only the AR process as input obtains the most stable or persistent results across all bit rates and network structures. The fact that providing additional information such as the AR parameter does not seem to influence the performance, can possibly be explained by the fact that the AR parameter is always the same. We previously argued that since the AR parameter is the same for all inputs, then from a information theoretic view the obtained rate-distortion curves were comparable. From the networks perspective this means that the node containing the AR parameter is the same no matter the input, and the node is thus not useful since it outputs the same value no matter the input. Nevertheless from the above results and in the context of speech coding, using the input/output strategy involving only the AR(1) process is most bit effective.

9. Speech Coding using DNNs

In this chapter we employ the autoencoder described in Section 4 to perform speech coding on the TIMIT speech data described in Section 7.2. However different from the fully connected layers used in the DNNs employed for encoding and decoding the synthetic data in Chapter 8, a mixture of fully connected and convolutional layers is employed in this chapter. Initial experiments showed that better performance in terms of lower test and training losses was obtained when using onvolutional layers compared to using solely fully connected layers. The training loss also stagnated much faster for a network consisting of only fully connected layers, thus preventing the networks from fully exploiting all the epochs to decrease the training loss. For these reasons the network structure illustrated in Figure 9.1 is utilized.

Different structures were tested, and the structure in Figure 9.1 was chosen due to its overall performance and simplicity. When using 1D convolutional layers care must be taken regarding the kernel sizes and amount of kernels used, since these affect the dimensions of each layer. For these reasons a chosen value regarding the kernel size and amount of kernels is not necessarily scalable to networks with different frame sizes. However dimensions of the layers in Figure 9.1 are all defined by the frame size N_x, thus circumventing the need for a specific network design for every new frame size used. A filter size of 5 and stride of 1 was chosen based on initial experiments.

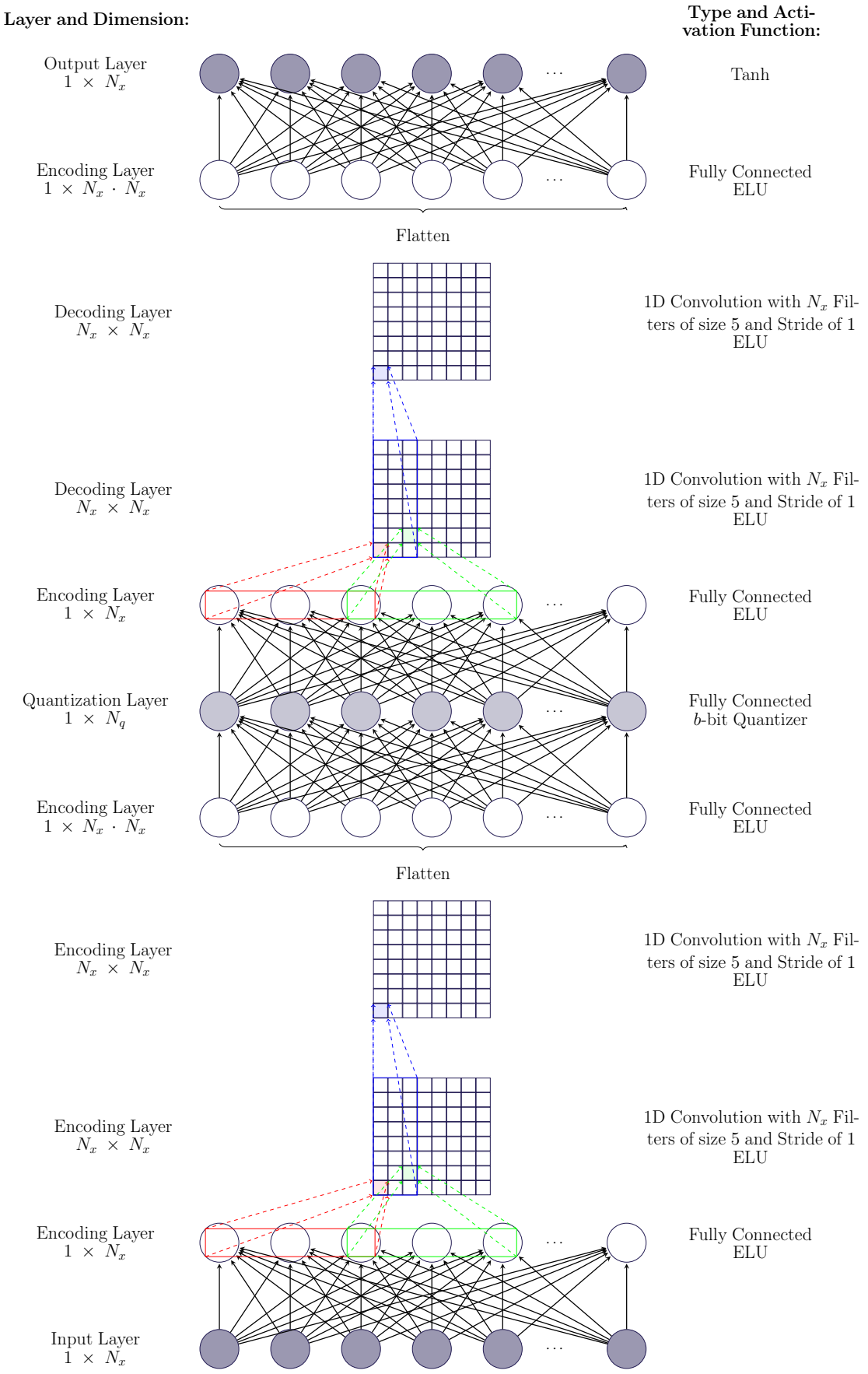


Figure 9.1: Network structure for speech autoencoder.

As previously explained we consider frames of size N_x as input to the autoencoder. Thus the TIMIT speech data examples are partitioned into non-overlapping frames of size N_x. However in order to aid the network training, each frame is normalized such that $x \in [-1, 1]$ for each sample x in a frame. Thus let $\boldsymbol{x} = [x_1, x_2, \ldots, x_{N_x}]$ be a frame of size N_x. Each sample x_i is normalized to obtain $\tilde{x}_i$ by:

$$\tilde{x}_i = \frac{x_i}{x_{max}} \tag{9.1}$$

where $x_{max} = \max_{1 \leq i \leq N_x} |x_i|$ is referred to as the *gain*. When referring to the input of an autoencoder in this chapter, we thus refer to the frame $\tilde{\boldsymbol{x}}$ of normalized samples $\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_{N_x}$.
Let $\hat{\boldsymbol{x}}$ denote the output of the autoencoder, i.e. the reconstructed version of the input $\tilde{\boldsymbol{x}}$. The reconstructed signal is transformed to its original range by $x_{max} \cdot \hat{\boldsymbol{x}}$. This means that the gain should also be encoded (and transmitted) alongside the output of the quantization layer. In this project we assume that this is done by an external quantizer and that 5 bits per gain are allocated for such process. The total bit rate in bits per sample is thus given as:

$$R = \frac{b \cdot N_q}{N_x} + \frac{5}{N_x} \tag{9.2}$$

for a frame size of N_x samples, a quantization layer of dimension N_q and a b-bit quantizer.

The presented bit rates in this chapter will refer to the total bit rate. However when reconstructing the signal and comparing it to the original signal, we will assume that the gain is perfectly reconstructed.

Since normalized inputs are used for the autoencoder, the tanh activation function (see Section 3.4.2) is used as an activation function for the output layer. For the hidden layers except the quantization layer, the ELU activation function (see Section 3.4.3) is used. This is also illustrated in Figure 9.1.

Due to the kernels in the convolutional layers, it was not possible to initialize all weight matrices using the identity matrix as in Chapter 8. The network was initialized with weights drawn from a uniform distribution with the range $[-a, a]$, where a is a small number dependent on the number of rows and columns of the weight matrix. I.e. the number of nodes in the preceding and succeeding layers associated with the weight matrix. This is known as the *Xavier* uniform initializer [21], and is the default weight initializer in Tensorflow. The choice of all training parameters was based on initial experiments.

In Section 9.1 and Section 9.2 results obtained by using the MSE as a loss function (see Section 3.5.1) are presented for frame sizes of 16 (1 ms) and 128 (8 ms) samples, respectively. In Section 9.3 results obtained by using our loss function proposed in Section 6.2 are shown, while Section 9.4 is concerned with results obtained by using a variable rate.

9.1 Frame size of 16 samples

For a frame size of 16 samples the network in Figure 9.1 was trained for 20 epochs and with a batch size of 256, the MSE as a loss function and using the Adam algorithm (see Section 3.5.2).

The network in Figure 9.1 was trained with the above specifications for different b-bit quantizers with $b = \{1, \dots, 8\}$ and with a quantization dimension of $N_q = N_x = 16$. With these network parameters, the network consists of approximately 12000 parameters, i.e. weights, biases and kernel weights. From (9.2) it can be seen that the total rate is given as $R = b + \frac{5}{16} = b + 0.3125$. The obtained operational rate-distortion curve is shown in Figure 9.2 for both the test and training data and with the MSE-distortion in dB.

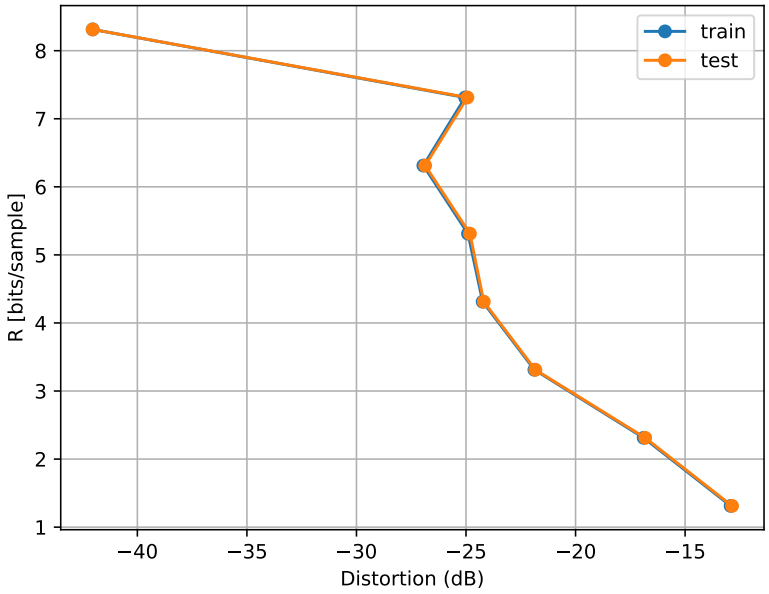


Figure 9.2: Operational rate-distortion curve in dB for networks with a frame size of 16 samples.

From Figure 9.2 it can be seen that there is no significant difference between the distortion for the test and training data. Furthermore it can be seen that as expected, the distortion decreases as the rate increases. However for b-bit quantizers with $4 \leq b \leq 7$ it can be seen that the decrease stagnates. Furthermore the distortion is worse for a 7-bit quantizer than for a 6-bit quantizer. The distortion for the quantizers with $4 \leq b \leq 7$ is approximately twice as low as the distortion obtained using 1-bit quantizer. For the 8-bit quantizer there is a sudden decrease in the distortion of more than 15 dB. Since all training losses converged it suggests that the sudden decrease in distortion is due to network initialization. For the remaining b-bit quantizers, the decrease in distortion is approximately 5 dB or less for each subsequent increase of b.

When listening to the signals reconstructed by the network using different b-bit

quantizers, it is worth mentioning that the reconstructed signals are intelligible at all bit rates. However there is some noise present, which significantly decreases as the amount of bits and thereby the bit rate increase. This is also illustrated in Figure 9.3 where spectrograms of a signal and its corresponding reconstructed versions using different quantizers are shown. Corresponding spectrograms of other signals can be seen in Appendix B.

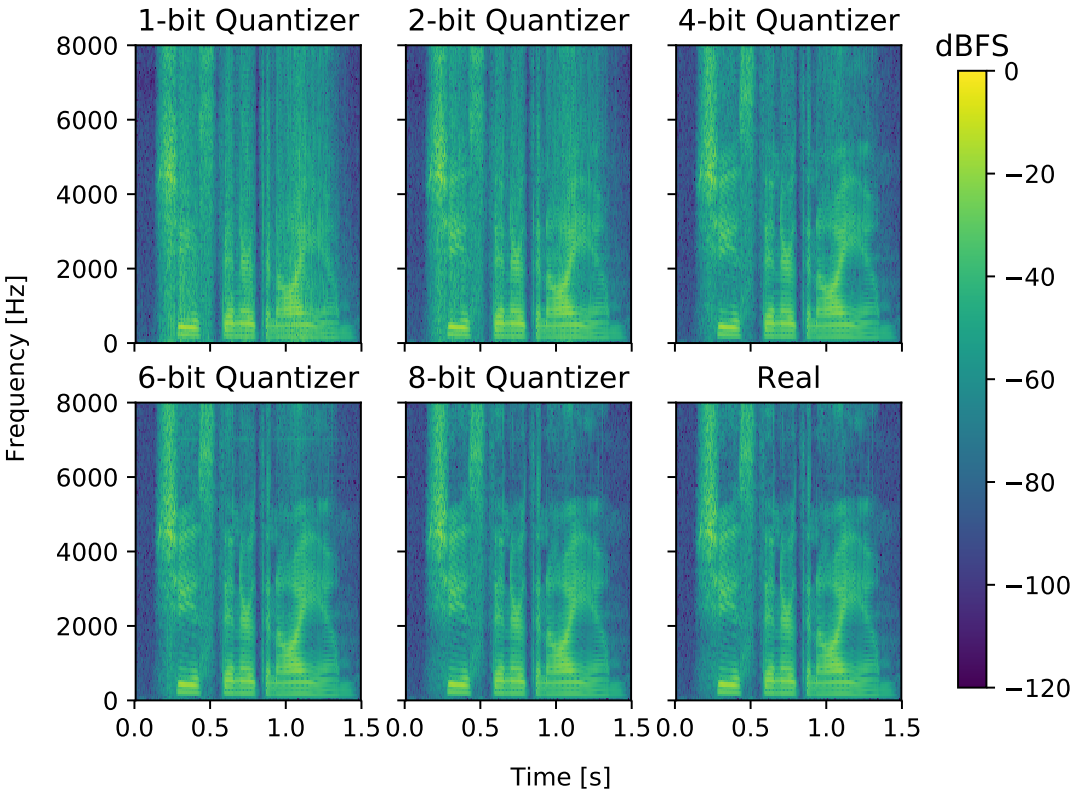


Figure 9.3: Spectrograms of a original signal (bottom right) and the corresponding reconstructed signal using networks with different b-bit quantizers.

From the figure it can be seen that the resemblance between the spectrogram of a reconstructed signal and the original increases as the amount of bits used increases. Even though the 1-bit quantizer obtains a distortion higher than -15 dB, characteristics of the original signal is still present. For the 1-bit quantizer most of the low-frequency characteristics are present, while the high frequency characteristics are hidden by noise. It can be seen that the noise decreases as the bit rate increases and thereby more characteristics are present for higher bit rates. For the 8-bit quantizer the reconstructed signal resembles the original signal almost identically. Even though the network with the 8-bit quantizer obtains a significant lower distortion compared to the other networks, there is not a significant difference between the spectrogram of a signal reconstructed with a 6 and 8-bit quantizer. This is also valid when listening to the corresponding reconstructed signals. It is worth noting that even though the noise seems to only be present at high frequencies, especially for the 1-bit quantizer, examinations revealed that the noise resembles white noise. I.e. the energy

of the frequencies are of equal intensity. It is also evident that the non-overlapping nature of the frames used as inputs to the networks does not affect the reconstructed signals. No boundary artefacts are present. Furthermore the networks are able to "detect" silence in the sense that silence is reconstructed almost identically for all networks. The lack of boundary effects is most likely caused by inputs to the network being in the time domain and the use of MSE as a loss function. As explained in Section 3.5.1 the MSE-distortion is the average distortion per sample taken across several examples. Thus it is unlikely that the network would reconstruct better or worse at the boundaries of a frame. Since we consider inputs in the time domain then poorly reconstructed samples at the boundaries do not affect the frequency content in the same manner as if we had considered inputs in the frequency domain.

We argued that the reconstructed signals are intelligible even when a 1-bit quantizer is used. This is also evident when evaluating the PESQ scores for the reconstructed signals. Figure 9.4 shows the corresponding average PESQ scores and standard deviations for the test data and the networks from Figure 9.2. The values obtained by encoding the signals using the BV32 codec (see Section 2.4) is also shown for comparison.

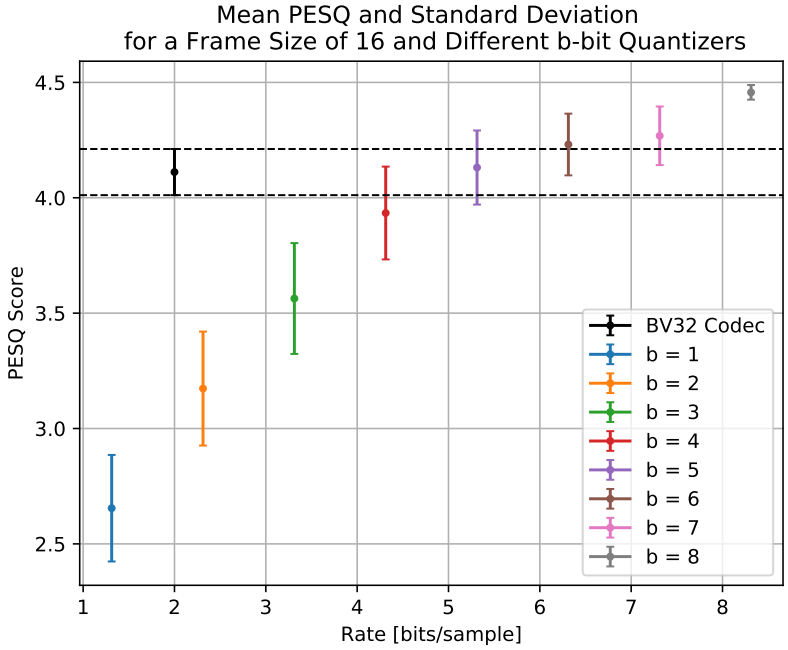


Figure 9.4: Average PESQ score and standard deviation the test data and for networks with different quantizers and a frame size of 16. The black plot correspond to the values obtained for encoding and reconstructing the same signals using BV32 Codec.

From the figure it can be seen that the average PESQ score increases as the bit rate increases. The average PESQ score is increased by approximately 0.5 by using a 2-bit quantizer instead of a 1-bit quantizer. The increase is less than 0.5 by successively increasing the bit rate, and the PESQ scores seem to stagnate for b-bit quantizers with $5 \leq b \leq 7$. The significant decrease in distortion obtained by using a 8-bit quantizer is also evident in Figure 9.4, where the network with a 8-bit quantizer is

able to obtain an average PESQ score of 4.46 and a standard deviation of 0.03. The standard deviation slowly decreases as the bit rate increases, meaning that not only does the distortion decrease but the network generalizes better for higher bit rates. From the standard deviation for the 8-bit quantizer it can further be seen that some signals obtain the maximum value of 4.5, meaning there is no objective difference between the reconstructed and original signal. Thus by using a bit rate of 8.31 bits per sample it is possible to reconstruct signals using a DNN autoencoder in such a manner that there is no perceptual difference. For a bit rate of 2 bits per sample, the BV32 codec obtains an average PESQ score of 4.11 and a standard deviation of 0.11. The networks obtained better PESQ scores than the BV32 Codec for bit rates higher than 5 bits per sample. Thus by using more than twice the bit-rate used in the BV32 Codec, one can obtain better PESQ scores than the BV32 codec.

9.2 Frame size of 128 samples

As discussed previously in this report, changing the dimension of the quantization layer N_q enables one to both perform dimensionality reduction and obtain bit rates $R < b$ for a b-bit quantizer. As seen from Chapter 8 using a frame size 16 does not provide much room for performing dimensionality reduction. In this section we investigate the performance of the network in Figure 9.1 with a frame size of 128 and for different dimensions of the quantization layer.

Using the Adam algorithm for training the network with a frame size of 128 proved to be inadequate. When using the Adam algorithm, the training and test loss quickly decreased as was the case in the previous section. However after a few epochs the losses suddenly increased and the training got stuck in the sense that the loss became constant. The problem can be circumvented by stopping earlier and restoring the model with the lowest test loss using early stopping (see Section 3.5.4). However these situations also occurred within the first epoch, making the Adam algorithm unusable. Experiments showed that the unstable training obtained by using Adam decreased to fewer cases when the frame size decreased. However even with a frame size of 32 the problem was present. Thus the networks in this section are trained using SGD (see Section 3.5.2) with a learning rate of 0.01 and a momentum of 0.9. The networks were trained for 200 epochs and the training was terminated earlier if the test loss did not decrease for 10 epochs. As can be seen on the amount of epochs, the use of SGD comes with a price of slower training time.

The networks were trained with the above specifications for different b-bit quantizers with $b = \{1, \dots, 8\}$ and with different quantization dimensions $N_x = \{128, 96, 64, 32\}$ corresponding to $100\%, 75\%, 50\%$ and 25% of the frame size. With these network parameters, the network consists of approximately 4 mio parameters. An immediate consequence of using a higher frame size is that the bits used for encoding the gain constitute a much smaller fraction of the total bit rate. The contribution is $5/128 = 0.0391$ compared to 0.3125 for a frame size of 16. The obtained operational rate-distortion curves for the test data and for the different b-bit quantizers and quantization dimensions are shown in Figure 9.5 with the distortion in dB. The

operational rate-distortion curve for a frame size of 16 samples is also shown for comparison.

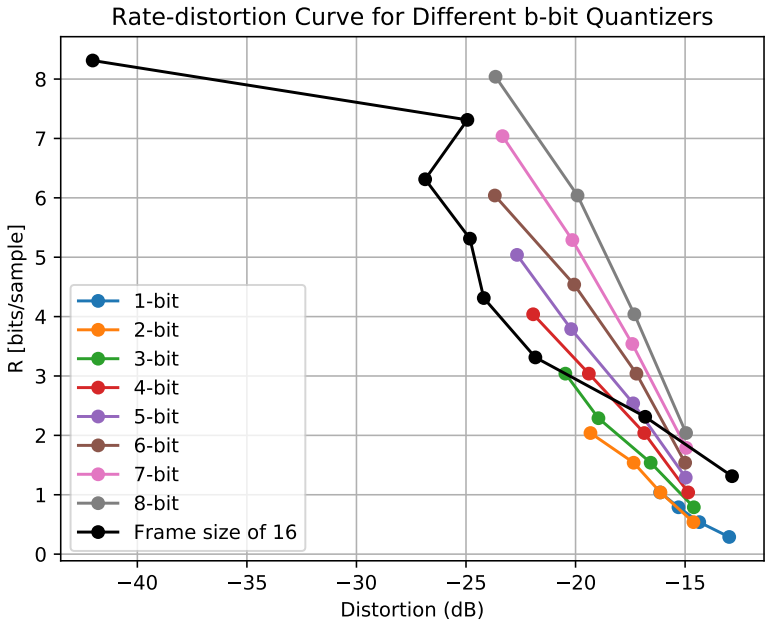


Figure 9.5: Operational rate-distortion curve in dB for the test data and for networks with different b-bit quantizers, quantization dimensions and with a frame size of 128 samples. For a given b the leftmost point correspond to a quantization dimension of 128 while the rightmost point corresponds to a quantization dimension of 32. The rate-distortion curve from Figure 9.2 is shown for comparison.

In Figure 9.5 the leftmost point for a given b-bit quantizer corresponds to a quantization dimension of 128, while the rightmost point corresponds to a quantization dimension of 32. From the figure it can be seen that for a fixed b the distortion decreases as the quantization dimension and thus rate increases. For a fixed quantizer, the distortion decreases by 2.5 dB in most cases for each subsequent increase in the quantization dimension. Such decreases are also present in the rate-distortion curve obtained when using a frame size of 16. However for a given quantization dimension it can also be seen that the distortion is approximately the same across all b-bit quantizers, with a slightly higher variance for larger quantization dimensions. Thus for small quantization dimensions their is no difference in the distortion by using higher bit rates.
Looking at the lower bound of the operational rate-distortion curve for networks with a frame size of 128, it can be seen that the lowest distortion for bit rates $R \geq 2$ is obtained when the dimension of the quantization layer is equal to the frame size. For $R < 2$ it is the 2-bit quantizer that performs best, where only the 1-bit quantizer is adequate for a quantization dimension of 32. However when comparing the results to the operational rate-distortion obtained for a frame size of 16, it can be seen that the lowest overall distortion for bit rates $R \geq 3$ is obtained when using a frame size of 16. For the lower bit rates it is more beneficial to use a frame size of 128 in terms of bit rate and distortion.

In Figure 9.6 the mean PESQ scores and standard deviations of the test data are shown alongside the PESQ score and standard deviation obtained by using the BV32 Codec. The PESQ scores from Figure 9.4 are also shown for comparison.

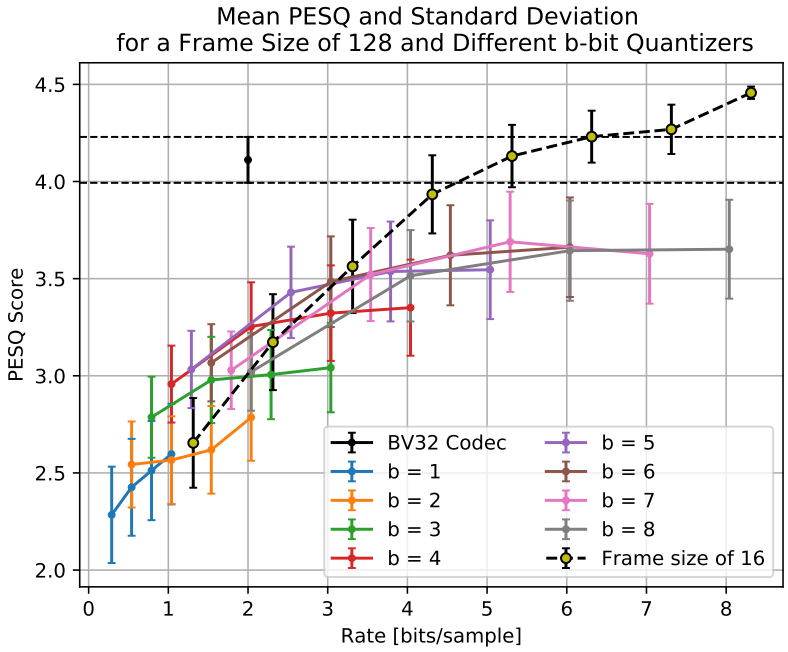


Figure 9.6: Average PESQ score and standard deviation for the test data and for networks with different quantizers, quantization dimensions and a frame size of 128. For a given b the leftmost point correspond to a quantization dimension of 32 while the rightmost point corresponds to a quantization dimension of 128. The PESQ score and standard deviation obtained by using BV32 is shown for comparison alongside the PESQ scores from Figure 9.4.

From the figure it can be seen the average PESQ score increases as the bit rate increases. However the increase is not as significant compared to the PESQ scores obtained when using a frame size of 16. For instance PESQ scores for b-bit quantizers with $b \geq 5$ are approximately the same and in general the increase in PESQ score is around 0.25. The standard deviation does not decrease when the bit rate increases as was the case for a frame size of 16. Furthermore the PESQ scores for a frame size of 128 do not at any time exceed the PESQ score for the BV32 codec. Nevertheless it is worth mentioning that the reconstructed signals are still intelligible across all quantizers and quantization dimensions.
Even though the average PESQ score increases as the bit rate increases, there are some differences between the distortions obtained in Figure 9.5 and the corresponding PESQ scores in Figure 9.6. For instance a network with a 2-bit quantizer and with a quantization dimension of 128 (leftmost orange point in Figure 9.5) achieves approximately the same distortion as a network with a 4-bit quantizer and a quantization dimension of 96 (2nd leftmost red point in Figure 9.5). However in Figure 9.6 the network with a 4-bit quantizer and a quantization dimension of 96 obtains a higher PESQ score (2nd rightmost red point in Figure 9.6) compared to the 2-bit quantizer with a quantization dimension of 128 (rightmost orange point in Figure 9.6). The

difference is approximately 0.5. Nevertheless using a frame size of 128 obtains better PESQ scores for rates $R < 3$ than using a frame size of 16. This coincides with the results observed in Figure 9.5.

9.3 Speech Coding using the Information Bottleneck Principle

In this section we employ our loss function inspired by the IB problem (see Chapter 6). Before we present results of networks trained with our loss function proposed, we first present an analysis of the trained networks from the previous section using the IB principle and our loss function.

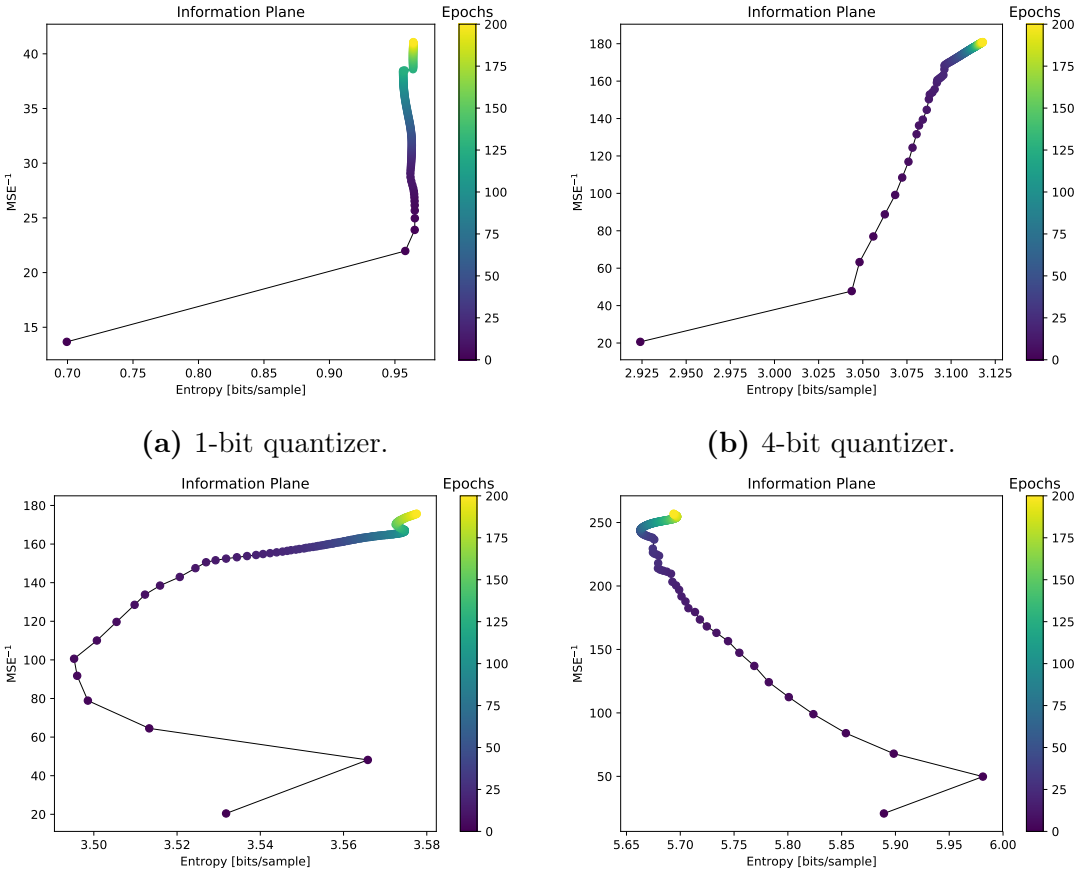
Recall from Section 6.2 that our IB inspired loss function is given as:

$$\min \mathcal{L}_{MSE}(\boldsymbol{w}) + \beta \sum_{i=1}^{N_q} \hat{H}(X_{q_i}) \tag{9.3}$$

where $\mathcal{L}_{MSE}(\boldsymbol{w})$ is MSE loss function (see Section 3.5.1) used for training the networks in the previous sections and $\hat{H}(X_{q_i})$ is the estimated entropy of node i in the quantization layer.

Recall furthermore from Section 6.1 that, as argued by the authors in [47], a neural network encounters two phases during training. At first both the MI between a layer and the input and between a layer and the desired output increases. At some point however a compression phase is observed, where the MI between the input and a layer decreases while the MI between a layer and the desired output increases. As argued in Section 6.2 the use of MSE in (9.3) is related to the mutual information between the input and output of a neural network, i.e. $I(X;\hat{X})$, in such a manner that a decrease in MSE implies an increase in the MI. Thus by measuring the MSE and marginal entropies of the networks trained with only MSE in the previous section, one can obtain an information plane plot.

Figure 9.7 shows information plane plots for the training data and the networks from Section 9.2 with a frame size and quantization dimension of 128 and for different b-bit quantizers. Notice that the entropies are shown in bits per sample. The corresponding figures for all quantizers can be seen in Appendix C. Since a decrease in MSE implies an increase in the mutual information, the MSE is inversed in the figures in order to obtain a similar figure as the information plot from Figure 6.1 in Section 6.1.



(a) 1-bit quantizer.

(b) 4-bit quantizer.

(c) 5-bit quantizer.

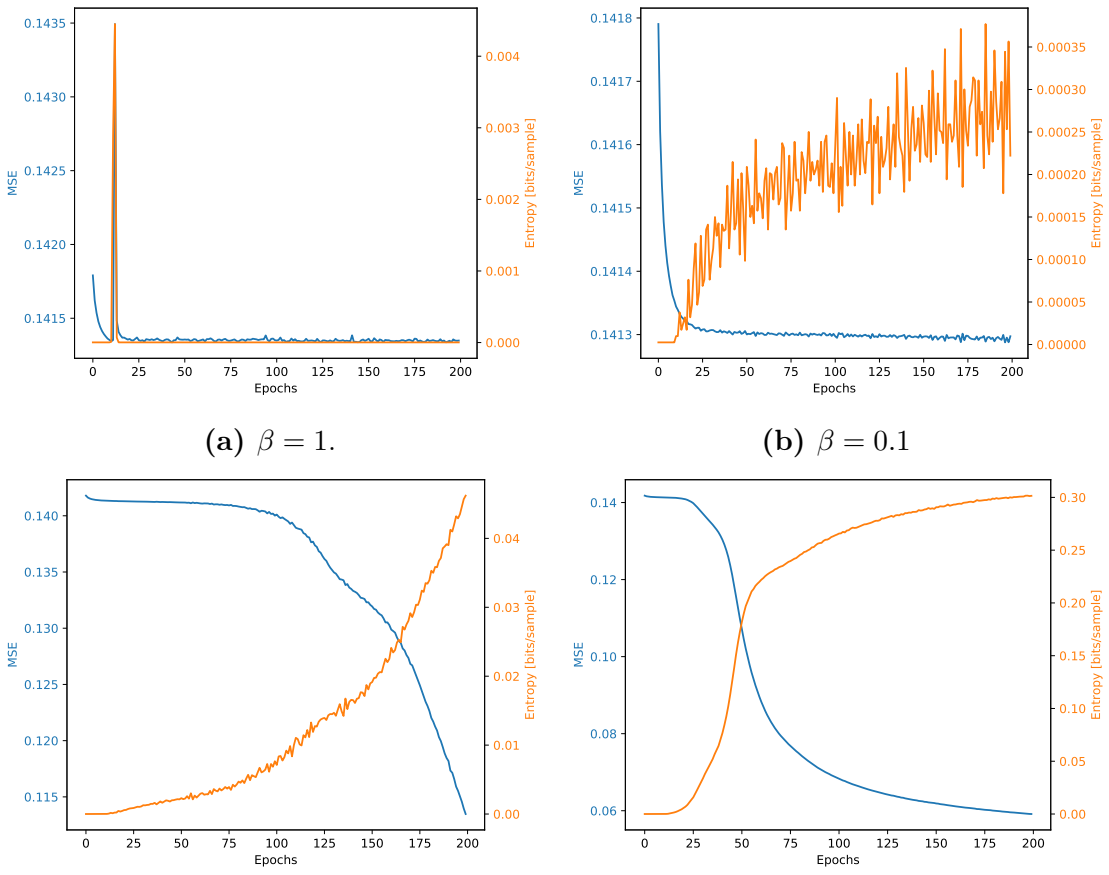
(d) 8-bit quantizer.

Figure 9.7: Information plane plots for the training data and for networks with a frame size and quantization dimension of 128 and for different b-bit quantizers. The x-axis is the entropy of the quantization layer in bits per sample, while the y-axis is the MSE^{-1}.

From the figure it can be seen that both the entropy and MSE^{-1} increase as a function of epochs for the 1- and 4-bit quantizer. Thus the entropy increases as the MSE decreases. However for the 5- and 8-bit quantizers, it is seen that the entropy encounters two phases. At first the entropy achieves an initial value, increases for 1 epoch until it decreases for an amount of epochs. After this, the entropy increases again. The MSE decreases for all epochs and quantizers. In general the characteristics of Figure 9.7a are present for all b-bit quantizers with $b \leq 4$, while the characteristics of Figure 9.7c are present for all remaining quantizers (see Appendix C).
Intuitively it makes sense that the entropy increases as the training proceeds since a decrease in MSE implies that the networks learn more features about or in the input. The decrease in the entropy at the beginning of the training experienced by quantizers with $b \geq 5$ can possibly be explained by the random initialization of weights. For the first couple of epochs the entropy of the quantization layer is most likely highly dependent on the initialized weights and gradient updates. However as the training proceeds, the networks learn more about the input, thus resulting in an increasing entropy. This can also explain why the seen characteristics for the higher bit quantizers are different from the characteristics seen for the lower bit quantizers. For a 1-bit quantizer there are only two possible values; 0 and 1. There

is less randomness involved for a 1-bit quantizer compared to a 8-bit quantizer. The authors in [44] argued that the compression phase is only visible when using a saturating activation function. The use of the non-saturating ELU activation function in this section and our findings coincide with the findings found in [44].

Regardless of the information plane characteristics it was attempted to train the networks using the loss function in (9.3). Even though no compression phase is observed the intuition behind using our loss function is still valid. As described earlier in Section 6.2 the goal of using our proposed loss function is to minimize the entropy such that the network is forced to only extract relevant parts of the input signal. Where the relevance is measured implicitly in terms of the MSE. Thus one might simulate or initialize a compression phase by using our loss function.
As seen from (9.3) a value of β is required. Figure 9.8 shows the MSE and entropy in bits per sample for the training data as a function of epochs for different values of β and for networks with a frame size and quantization dimension of 128 and a 1-bit quantizer.



(a) $\beta = 1.$ **(b)** $\beta = 0.1$

(c) $\beta = 0.01$ **(d)** $\beta = 0.001$

Figure 9.8: The impact on the entropy and MSE of the traning data for different values of β. The entropy is normalized to bits per sample.

From the figures it can be seen that when $\beta = 1$, i.e. both terms of the loss function are weighted equally, only the entropy is minimized. Both terms stagnate quickly

and a low value of the entropy is attained only after a few epochs. This makes sense since the MSE has a range between 0 and 4 for normalized inputs in the range $[-1, 1]$, while the upper bound of the entropy term is $b \cdot N_q$ for a b-bit quantizer and a quantization dimension of N_q. Thus the entropy term has a larger range than the MSE. For $\beta = 0.1$ the entropy increases as a function of epochs, however the value of the entropy is in general less than 0.00035 which is significantly lower compared to the value of > 0.95 for the same network structure in Figure 9.7a. As seen this comes at a price of high MSE value of 0.1413 corresponding to approximately -8.5 dB. For both β values the MSE achieves approximately the same value. Thus with these values of β it is not possible to achieve suitable values of the entropy and MSE, i.e. it is not possible to reduce the entropy without obtaining a significant increase in the distortion. Further decreases of β results in an increase of the entropy and decrease of the MSE. By continuing the training for more epochs the entropy quickly stagnates. However the value of the MSE is still significantly higher than the MSE obtained by solely using the MSE as a loss function. After $10,000$ epochs the MSE is higher than 0.05 when using our loss function, which is the initial value of the MSE when training networks using only the MSE as a loss function. Thus even for smaller values of β were are not able to achieve suitable values of both the entropy and MSE. The value of β determines the training time in the sense that a smaller value of β results in more epochs before the entropy an MSE stagnates.

Similar characteristics were seen for networks with different values of b in the b-bit quantizers. It can thus be concluded that is was not possible to find adequate values of β making the our proposed loss function suitable for training the networks under the IB principle.

9.4 Variable Rates

Even though no compression phase was observed and no adequate values of β were found, the entropy still provides an insight to the characteristics and training of the DNN speech autoencoders.

From Figure 9.7d it can be seen that the entropy is between $5.5 - 6.0$ bits per sample for a 8-bit quantizer. Thus the network with a 8-bit quantizer does not fully explore the entropy of the quantization layer. Figure 9.9 shows the entropy of the quantization layer in bits per sample for the networks from Figure 9.5 with a frame size and quantization dimension of 128 and different b-bit quantizers at the final epoch. The entropy is normalized to bits per sample.

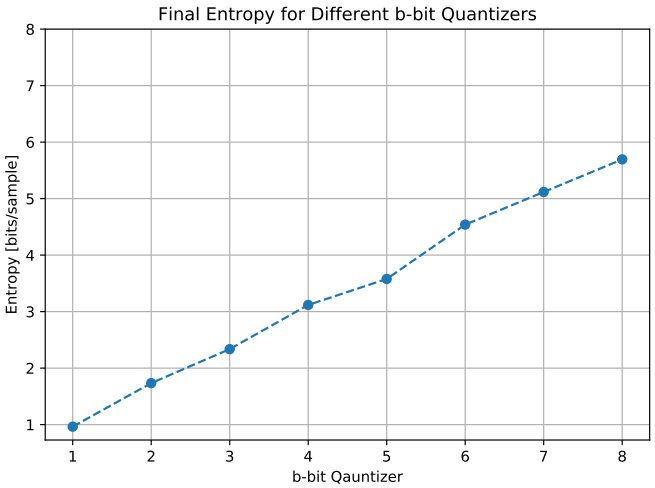


Figure 9.9: Final entropy in bits per sample for different b-bit quantizers in a network with a frame size and quantization dimension of 128.

Due to the quantization layer being equal to the frame size, the upper bound of the entropy is b bits per sample for a network with a b-bit quantizer. From Figure 9.9 it can be seen that the entropy for a network with a 1-bit quantizer is approximately 1 bit per sample. The entropy is thus close to the upper bound. It can furthermore be seen that the difference between the entropy at a final epoch and the upper bound increases as b increases. This means that lower bit rates can be obtained if entropy coding (see [59] and [12]) were to be employed. Entropy coding involves variable bit rates, where the average rate is given as the entropy of the variable. Even though we mainly consider fixed bit rates in this project we briefly explore whether significant results regarding the bit rate can be obtained.
For a b-bit quantizers with $b \geq 5$ the difference between the entropy and upper bound is more than 1.5 bits per sample. Ignoring the bits used for encoding the gain for a moment, this means that the bit-rate of a DNN autoencdoer with e.g. a 6-bit quantizer can be reduced by 25% if entropy encoding were to be employed.

By measuring the marginal entropies of the quantization layer nodes at the end of training and on the test data, new operational rate-distortion curves can be obtained for the networks in Figure 9.5. This is shown in Figure 9.10. Notice that the bit rate for encoding the gain is included too for comparability with Figure 9.10.

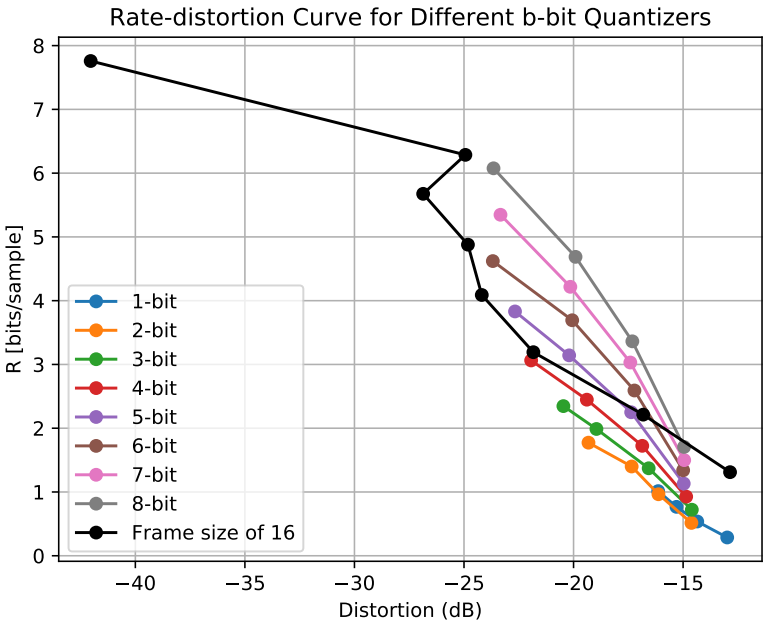


Figure 9.10: Rate-distortion curves with entropy-corrected bit rates for networks in Figure 9.5.

Comparing Figure 9.10 with Figure 9.5 there seem to be no significant difference. The same characteristics are present. However in general the difference in rate between the networks with a frame size of 16 and the networks with a frame size and quantization dimension of 128 is larger in Figure 9.10 compared to Figure 9.5. This implies that networks with a frame size of 16 utilize the bits in the quantizers more in some sense. The rate for a network with a frame size of 16 and a 8-bit quantizer is for instance close to 8 bits per sample in Figure 9.10.
Examining the lower bound of the operational rate-distortion curve reveals some differences. In Figure 9.5 the lower bound for bit rates $R > 3$ was achieved by using a frame size of 16, while the lower bound for lower rates was achieved by using a frame size of 128 and a 2-bit quantizer except the lowest rate which was achieved by a 1-bit quantizer and a quantization dimension of 32. However in Figure 9.10 it can be seen that the lower bound for rates $2 \leq R \leq 4$ is achieved by using 3- and 4-bit quantizers in a network with a frame size and quantization dimension of 128.

Figure 9.11 shows the corresponding average PESQ scores and standard deviations for the test data and the entropy-corrected networks of Figure 9.10.

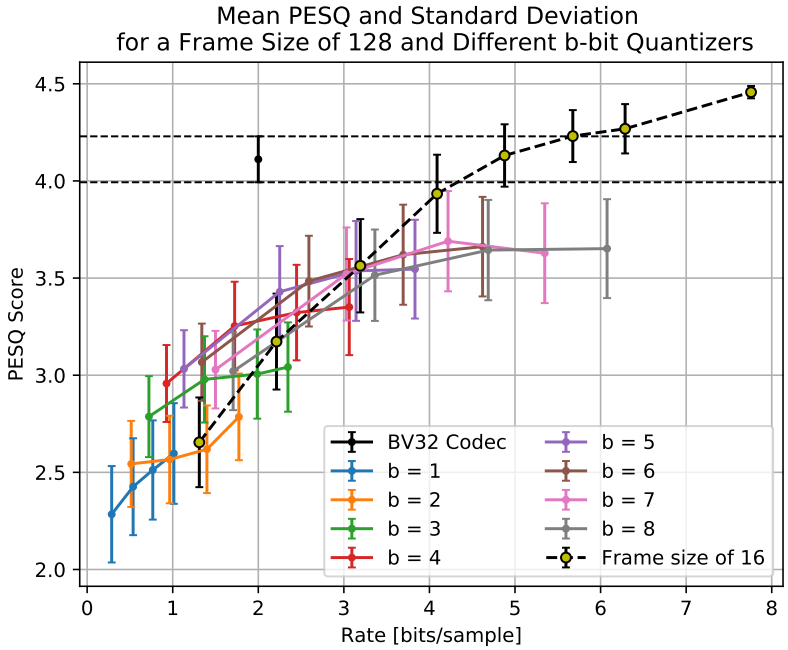


Figure 9.11: Corresponding average PESQ score and standard deviation for entropy-corrected networks of Figure 9.10.

Even though a network with a frame size and quantization dimension of 128 and a 4-bit quantizer is slightly better in terms of distortion than using a frame size of 16 and a 3-bit quantizer, the network with a frame size of 16 and a 3-bit quantizer obtains higher PESQ scores in Figure 9.11. In general the characteristics of Figure 9.6 are also present in Figure 9.11 and thus the even the entropy-corrected networks are not able to outperform the BV32 Codec rate-wise. However by exploring the entropy of the quantization layer it is possible to achieve the average PESQ score of 4.46 and standard deviation of 0.03 using half the bit rate used for standard 16-bit PCM encoding.

III Discussion and Conclusion

10 Discussion 99

11 Conclusion 103

10. Discussion

In this project DNN autoencoders were developed with the aim to perform speech coding. The autoencoders were developed using a end-to-end strategy meaning that only transmission and pre- and post-processing steps were external factors. This choice was based on the immediate achieved flexibility in terms of compatibility with Tensorflow, network training, easy control of bit rates and compatibility with external systems should the autoencoders be employed in real life. However a consequence of such strategy is that the end-to-end DNN structure complicates the ability to investigate, understand and control the internal characteristics and dynamics. For instance it is unclear how the encoding and decoding parts of the developed DNN autoencoders affects each other and the results. E.g. it is unclear whether a high distortion is due to a poor encoder, decoder or both.
A possible different strategy could be to encode signals with an existing codec such as BV32 and train a DNN to decode the signal. After ended training one could replace the existing encoder with a DNN and train the DNN to encode the signal. In this way a similar end-to-end autoencoder is obtained but the training is more controlled. Since the results especially regarding the synthetic data were highly dependent on the initialization of the networks, another strategy could be to pre-train all the layers of a end-to-end DNN autoencoder as individual autoencoders. This is known as *stacked autoencoders* [57]. By doing so one could imagine that it is easier in some sense for the network to "understand" that the goal is autoencoding, because the pre-trained layers can be seen as an initialization implying such.

The developed quantizer, which was used in the quantization layer as an activation function, had a dynamic range between 0 and 1. As argued in this project the dynamic range of the quantizer should not matter due to the adjustable weights. However it is possible that the developed quantizer experience similar problems as the ReLU activation function explained in Section 3.4.3. Negative inputs to the quantizer are quantized to zero and have a gradient of zero. Hence it is possible that our quantizer can lead to dead nodes, which is a problem similar to the problems experienced when using ReLU. For networks where the Tanh activation function was used in the output layer, it is possible that a dynamic range between -1 and 1 would have been more beneficial.

Different input/output strategies were tested using synthetic data and it was found that using the signal itself was most bit-effective. This lead to using solely speech signals as inputs for the speech autoencoder DNNs. However regarding the difference between using synthetic data with and without the AR parameter as input, it is worth noting that the AR parameter was a constant. From a DNN perspective this means that the node containing the AR parameter is the same no matter the input. The node is thus not useful since it outputs the same value no matter the input. If the same strategy were to be employed using speech data it is likely that different results would be obtained, since inputs with e.g. autoregressive parameters would vary for each frame. Nevertheless using the input/output strategy involving only the signal itself also obtained the most stable results. In this sense the choice of using only speech signals as inputs is justified. Another possible strategy which has not been tested is to use e.g. the short time Fourier transform as (additional) input. Furthermore a different strategy could be to consider hidden layers as inputs for succeeding hidden layers in a structure inspired by the U-shaped neural networks in [42]. Where in order to aid the decoding part of the autoencoders, one could also encode hidden layers of the encoding part of the autoencoders. The encoded layers could then be transmitted alongside the quantization layer and gain to the decoding part. This comes at an expense of higher bit rates, however it is possible that such structure could reduce the distortion significantly and hence justify the higher bit rates.

The DNNs involving the speech data with frame sizes of 16 samples and the synthetic data were all trained using the Adam algorithm, while the DNNs involving speech data with frame sizes of 128 were trained using SGD. As described earlier the use of SGD was caused by unstable training obtained by using Adam. It can therefore be argued that the results are not comparable in some sense and hence whether the conclusions regarding the optimal frame size for a specific bit rate are true. This is especially the case for higher bit rates since networks with a frame size of 16 obtained significantly higher PESQ scores compared to networks with a frame size of 128. Using SGD for training requires more extensive experiments regarding learning rate, the epochs used for both training and early stopping and the value of momentum compared to using Adam. It is possible that different results and conclusion would have been obtained for different parameters of the SGD algorithm. Nevertheless all the training losses converged to a local minimum for both training algorithms. Furthermore the networks not being able to fully utilize the higher amount of bits for the higher bit rates is visible for all experiments. Hence the experiments are comparable in this sense.

No compression phase was found when using the IB principle for analysing DNNs. Since the used ELU activation function does not saturate, our results regarding no compression phase coincides with the results found in [38]. However since the reduction in entropy seen at the beginning of the training is in general insignificant in our experiments, it can be discussed whether phase changes are seen at all. On the other hand the networks in this report were only trained for a maximum of 200

epochs, while the networks in [47] with a compression phase were trained for 10^4 epochs. It can therefore be discussed whether the missing of the compression phase in this project is due to the amount of epochs. However similar experiments were conducted where more epochs were used and no compression phase was observed. Regarding the IB principle for DNNs in general, it is also discussable how the theory coincides with terms such as over- and under training and early stopping. The compression phases in [47] only occurred relatively late in the training process and early stopping will in general prevent one from reaching a large amount of epochs. That is at least the case in this project. Since the networks in [47] were trained using only SGD without e.g. momentum, it is also possible that the use of Adam and SGD with momentum has affected the results in this project.

Regarding our proposed loss function inspired by the IB principle, it is worth discussing whether our proposed loss function violates the nature of the IB principle. As mentioned the compression phase observed in [47] occurred relatively late in the training and prior to this phase a phase of information increase was observed. Thus one could argue that the phase of information increase is skipped when using our loss function. This can also explain the inability of finding adequate values of the weight β. However experiments were also conducted where networks obtained by using the MSE as loss function were trained further using our loss function. In such experiments it was also not possible to reduce the entropy while maintaining a suitable level of distortion. Experiments where β was gradually increased as a function of epochs were also conducted with similar results. A more comprehensive study of the trade-off between distortion and entropy was not conducted due to time constrains.

Our proposed loss function was based on the marginal entropy of each node. It is possible that the reason for the networks being unable to train using our loss function is due to the joint entropy not being considered. It is possible the marginal entropies are not suitable guidances for extracting relevant information, since the interactions and dependencies between nodes are not considered. The distribution of quantized values within a node and whether nodes in the quantization layer are independent of each other were also not examined.

11. Conclusion

In this project a speech coding algorithm was developed using an end-to-end approach, where an encoding DNN and a decoding DNN was trained as a single DNN autoencoder. Signals were encoded using a b-bit scalar quantizer acting as an activation function for each node and forming an quantization layer. The bit rate is easily controllable by changing the parameters of the quantizer, frame size of the input and the dimension of quantization layer. Experiments using synthetic data showed that most stable and bit effective results were obtained using only frames of the signal itself. This lead to using solely frames of speech signals as input to the DNN autoencoder.

It can be concluded that it is possible to encode and decode both synthetic data and speech signals. It can further be concluded that training DNN speech autoencoders using the MSE as a loss function and with frames of speech as inputs, results in no visible nor perceivable boundary effects in the reconstructed speech signals. For bit rates higher than 3 bits per sample using a frame size of 16 obtains better results in terms of PESQ scores, bit-rate and distortion compared to a frame size of 128. For lower bit rates it can be concluded that a frame size of 128 obtains the best results. In general it can be concluded that using a quantization dimension equal to the frame size obtains better results than using a quantization dimension smaller than the frame size.

From the results obtained by training DNN autoencoders with the MSE as a loss function, it can also be concluded that the autoencoders are not able to outperform the existing BV32 codec in terms of both bit rate and PESQ scores. Encoding TIMIT speech signals using the BV32 codec achieves an average PESQ score of 4.11 and a standard deviation of 0.11 at a bit rate of 2 bits per sample. Encoding the same signals using the same bit rate and the developed DNN autoencoder achieves an average PESQ score of 3.25 and a standard deviation of 0.23. However it can also be concluded that signals encoded using DNNs are intelligible across all bit-rates, quantization dimensions and frame size, achieving a average PESQ score of 2.25 or more. For bit rates higher than 5 bits per sample it is possible to outperform the BV32 codec in terms of PESQ scores only. By using a bit rate of 8.31 bits per sample it is possible to achieve an imperceptible distortion, where an average PESQ

score of 4.46 with a standard deviation of 0.03 is obtained.

It can be concluded that a compression phase is not observed when extending the IB principle to DNN speech coders. The findings in this project coincides with already existing papers. A loss function inspired by the IB principle was proposed consisting of a superposition of the MSE and sum of marginal entropies of nodes in the quantization layer. The importance of the terms in the loss function was controlled by a weight β. Based on experiments it was found that training DNN autoencoders using such loss function is not suitable, in the sense that it was not possible to both decrease the entropy and maintain a suitable MSE-distortion. The lowest MSE value obtained using this loss function was 0.05 after $10,000$ epochs which corresponds to the initial MSE value obtained after 1 epoch using solely the MSE as a loss function. However by exploring the marginal entropies of the nodes in the quantization layer it is possible to achieve the average PESQ score of 4.46 and standard deviation of 0.03 for the DNN speech autoencoders trained solely with the MSE as a loss function and by using a bit rate less than half the bit rate used for standard 16-bit PCM encoding.

In general it can be concluded that the DNN autoencoders are not able to fully explore the extra bits imposed by using a higher bit rate. The distortion and PESQ scores stagnates for higher bit rates. All the results are possibly affected by the training algorithms used, training parameters, chosen network structures and possible dependencies between nodes in the quantization layer.

IV

Bibliography

Bibliography 107

Bibliography

[1] Perceptual evaluation of speech quality (pesq): An objective method for end-to-end speech quality assessment of narrow-band telephone networks and speech codecs.

[2] Alexander A. Alemi, Ian Fischer, Joshua V. Dillon, and Kevin Murphy. Deep variational information bottleneck. 12 2016.

[3] O. Abdel-Hamid, A-R Mohamed, H. Jiang, and G. Penn. Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition. *Acoustics, Speech and Signal Processing (ICASSP)*, 2012.

[4] Guillaume Alain and Y Bengio. Understanding intermediate layers using linear classifier probes. 04 2017.

[5] R. A. Amjad and B. C. Geiger. How (not) to train your neural network using the information bottleneck principle. *CoRR*, abs/1802.09766, 2018.

[6] C.M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, 2006. ISBN 9780387310732.

[7] M. Bosi and R.E. Goldberg. *Introduction to Digital Audio Coding and Standards*. The Springer International Series in Engineering and Computer Science. Springer US, 2002. ISBN 9781402073571.

[8] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. ISBN 9780521833783.

[9] J. Chen and J. Thyssen. The broadvoice speech coding algorithm. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*, volume 4, pages IV–537–IV–540, April 2007. doi: 10.1109/ICASSP.2007.366968.

[10] Jitong Chen, Yuxuan Wang, Sarah Yoho, DeLiang Wang, and Eric Healy. An algorithm to increase speech intelligibility for hearing-impaired listeners in entirely novel noises. *Journal of the Acoustical Society of America*, 139: 1995–1995, 04 2016. doi: 10.1121/1.4949848.

[11] D. Ciregan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3642–3649, June 2012. doi: 10.1109/CVPR.2012.6248110.

[12] T.M. Cover and J.A. Thomas. *Elements of Information Theory*. Wiley, 2012. ISBN 9781118585771.

[13] G. Cybenko. Approximation by superpositions of a sigmoidal function. *MCSS*, 2:303–314, 1989.

[14] D-A., Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *CoRR*, abs/1511.07289, 2015.

[15] M. S. Derpich and J. Østergaard. Improved upper bounds to the causal quadratic rate-distortion function for gaussian stationary sources. In *2010 IEEE International Symposium on Information Theory*, pages 76–80, June 2010. doi: 10.1109/ISIT.2010.5513282.

[16] Andrzej Drygajlo and Martin Rajman. Speech coding techniques and standards. 06 2019.

[17] J. Du, Y. Tu, Y. Xu, L. Dai, and C. Lee. Speech separation of a target speaker based on deep neural networks. In *2014 12th International Conference on Signal Processing (ICSP)*, pages 473–477, Oct 2014. doi: 10.1109/ICOSP.2014.7015050.

[18] H.C. Edwards and D.E. Penney. *Calculus, Early Transcendentals: Pearson New International Edition*. Pearson custom library. Pearson Education, Limited, 2013. ISBN 9781292022178.

[19] Ian Fischer. The conditional entropy bottleneck, 2019.

[20] J.S. Garofolo, L.F. Lamel, W. M. Fisher, J.G. Fiscus, D.S. Pallet, and N. L. Dahlgren. "the darpa timit acoustic-phonetic continuous speech corpus", 1993.

[21] Xavier Glorot and Y Bengio. Understanding the difficulty of training deep feedforward neural networks. *Journal of Machine Learning Research - Proceedings Track*, 9:249–256, 01 2010.

[22] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2016. ISBN 9780262035613.

[23] A. Graves, A-R Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. *Acoustics, Speech and Signal Processing (ICASSP)*, 2013.

[24] Hassan Hafez-Kolahi and Shohreh Kasaei. Information bottleneck and its applications in deep learning. *CoRR*, abs/1904.03743, 2019.

[25] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, Nov 2012. ISSN 1053-5888. doi: 10.1109/MSP.2012.2205597.

[26] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251 – 257, 1991. ISSN 0893-6080.

[27] S. Ji, W. Xu, M. Yang, and K. Yu. 3d convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):221–231, Jan 2013. ISSN 0162-8828. doi: 10.1109/TPAMI.2012.59.

[28] N. Kalchbrenner, E. Grefenstette, and P. Blunsom. A convolutional neural network for modelling sentences. *CoRR*, abs/1404.2188, 2014.

[29] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[30] Morten Kolbæk, Zheng-Hua Tan, and Jesper Jensen. Monaural speech enhancement using deep neural networks by maximizing a short-time objective intelligibility measure. *CoRR*, abs/1802.00604, 2018.

[31] Artemy Kolchinsky, Brendan Tracey, and David Wolpert. Nonlinear information bottleneck. 05 2017.

[32] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017. ISSN 0001-0782. doi: 10.1145/3065386. URL http://doi.acm.org/10.1145/3065386.

[33] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998. ISSN 0018-9219. doi: 10.1109/5.726791.

[34] Y. LeCunn and Y. Bengio. Deep learning. *Nature*, 521:436–444, May 2015.

[35] Christopher Maddison, Aja Huang, Ilya Sutskever, and David Silver. Move evaluation in go using deep convolutional neural networks. 12 2014.

[36] H. Madsen. *Time Series Analysis*. Chapman & Hall/CRC Texts in Statistical Science. Taylor & Francis, 2008. ISBN 9781420059670.

[37] Gaurav Naithani, Joonas Nikunen, Lars Bramsløw, and Tuomas Virtanen. Deep neural network based speech separation optimizing an objective estimator of intelligibility for low latency applications. *CoRR*, abs/1807.06899, 2018.

[38] Morteza Noshad and Alfred O. Hero III. Scalable mutual information estimation using dependence graphs. *CoRR*, abs/1801.09125, 2018.

[39] P. Olofsson and M. Andersson. *Probability, Statistics, and Stochastic Processes*. Wiley, 2012. ISBN 9780470889749.

[40] A.V. Oppenheim and R.W. Schafer. *Discrete-Time Signal Processing: Pearson New International Edition*. Pearson Education Limited, 2013. ISBN 9781292038155.

[41] O. Rippel and L. Bourdev. Real-time adaptive image compression. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2922–2930, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[42] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[43] T. N. Sainath, A. Mohamed, B. Kingsbury, and B. Ramabhadran. Deep convolutional neural networks for lvcsr. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8614–8618, May 2013. doi: 10.1109/ICASSP.2013.6639347.

[44] A. M. Saxe, Y. Bansal, J. Dapello, M. Advani, A. Kolchinsky, B. D. Tracey, and D. D. Cox. On the information bottleneck theory of deep learning. In *International Conference on Learning Representations*, 2018.

[45] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61(Supplement C):85 – 117, 2015. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2014.09.003.

[46] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, Vol. 27:p. 379–423 (July), 623–656 (October), 1948. Harvard University reprinted version with corrections from The Bell System Technical Journal.

[47] R. Shwartz-Ziv and N. Tishby. Opening the black box of deep neural networks via information. *CoRR*, abs/1703.00810, 2017.

[48] R. J. Sluyter. The state of the art in speech coding. In *ESSCIRC '83: Ninth European Solid-State Circuits Conference*, pages 33–40, Sep. 1983.

[49] A. S. Spanias. Speech coding: a tutorial review. *Proceedings of the IEEE*, 82 (10):1541–1582, Oct 1994. ISSN 0018-9219. doi: 10.1109/5.326413.

[50] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overtting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[51] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014.

[52] N. Tishby and N. Zaslavsky. Deep learning and the information bottleneck principle. *CoRR*, abs/1503.02406, 2015.

[53] N. Tishby, F. Pereira, and W. Bialek. The information bottleneck method. In *In Proceedings of the 37-th Annual Allerton Conference on Communication, Control and Computing*, page 368–377, 1999.

[54] Michael Tschannen, Olivier Bachem, and Mario Lucic. Recent advances in autoencoder-based representation learning. *CoRR*, abs/1812.05069, 2018.

[55] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016.

[56] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016.

[57] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, December 2010. ISSN 1532-4435.

[58] Y. Xu, J. Du, L. Dai, and C. Lee. A regression approach to speech enhancement based on deep neural networks. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(1):7–19, Jan 2015. ISSN 2329-9290. doi: 10.1109/TASLP.2014.2364452.

[59] Y. You. *Audio Coding: Theory and Applications*. Springer US, 2010. ISBN 9781441917546.

[60] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *CoRR*, abs/1611.03530, 2016.

[61] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *CoRR*, 2016.

V

Appendix

A Results using Synthetic Data ... 115
A.1 Balanced Network
A.2 Undercomplete Network

B Spectrograms of Reconstructed Signals 127

C Information Plane Plots 131

A. Results using Synthetic Data

This appendix consists of results obtained for networks trained with synthetic data but not shown in Chapter 8.

A.1 Balanced Network

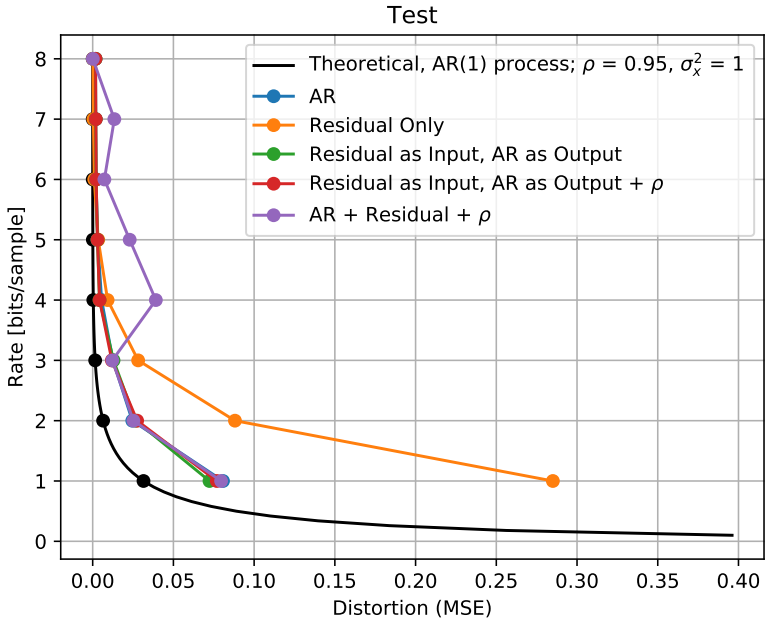


Figure A.1: Operational rate-distortion curves for the balanced network with different input/output strategies and a frame size of 16.

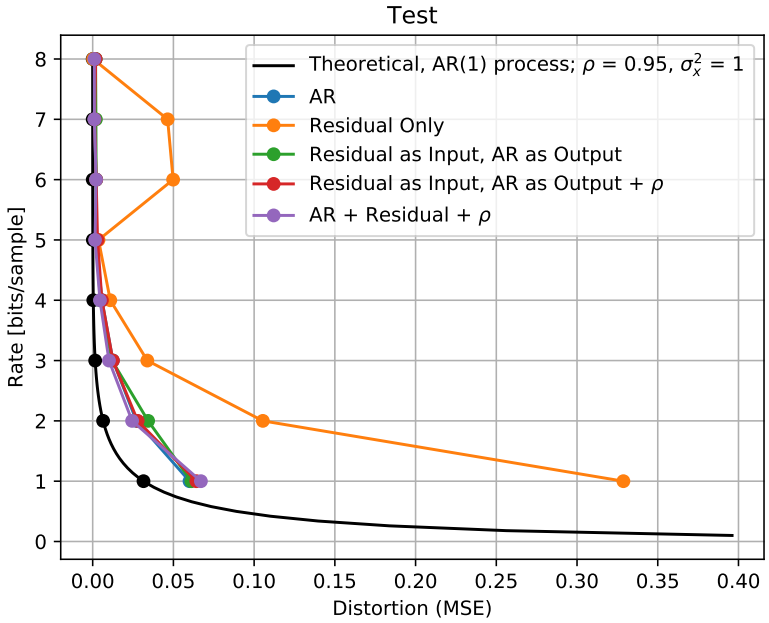


Figure A.2: Operational rate-distortion curves for the balanced network with different input/output strategies and a frame size of 32.

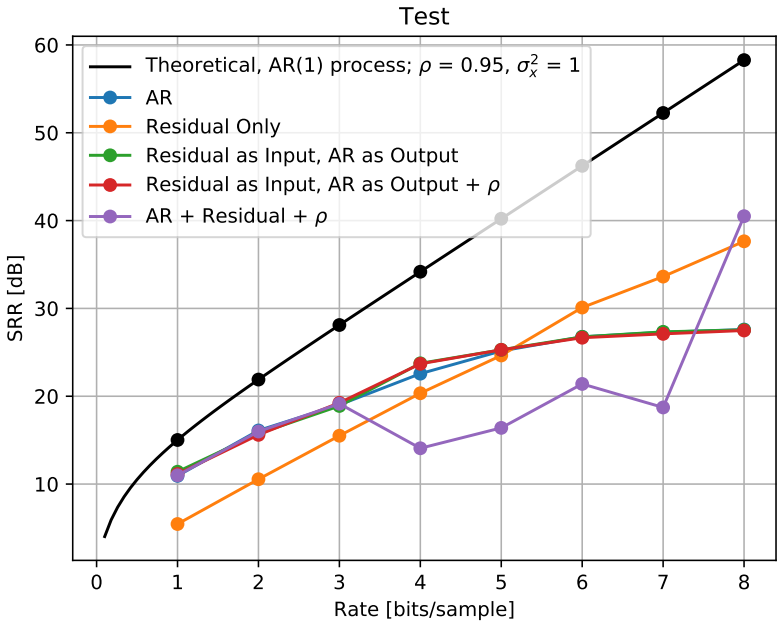


Figure A.3: SRR and bit rates for the balanced network with different input/output strategies and a frame size of 16.

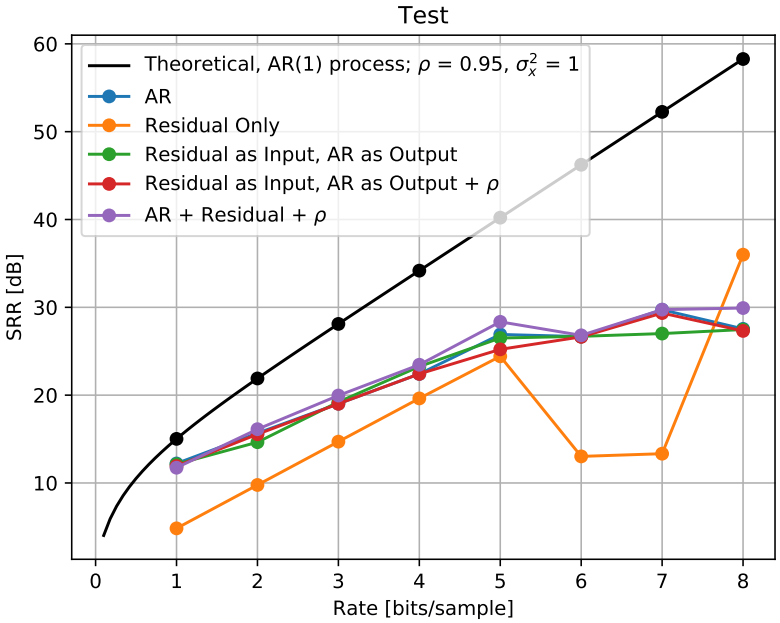


Figure A.4: SRR and bit rates for the balanced network with different input/output strategies and a frame size of 32.

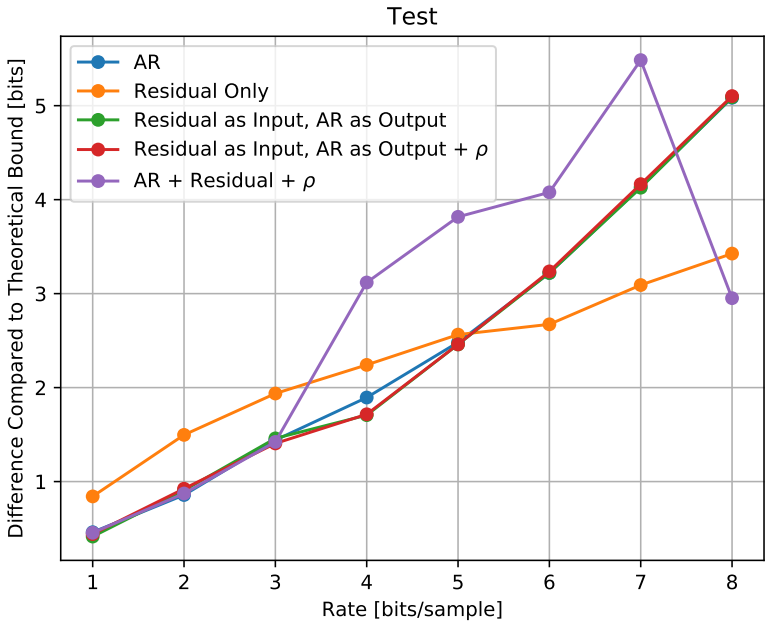


Figure A.5: Difference in bits and the bit rate for the balanced network with different input/output strategies and a frame size of 16.

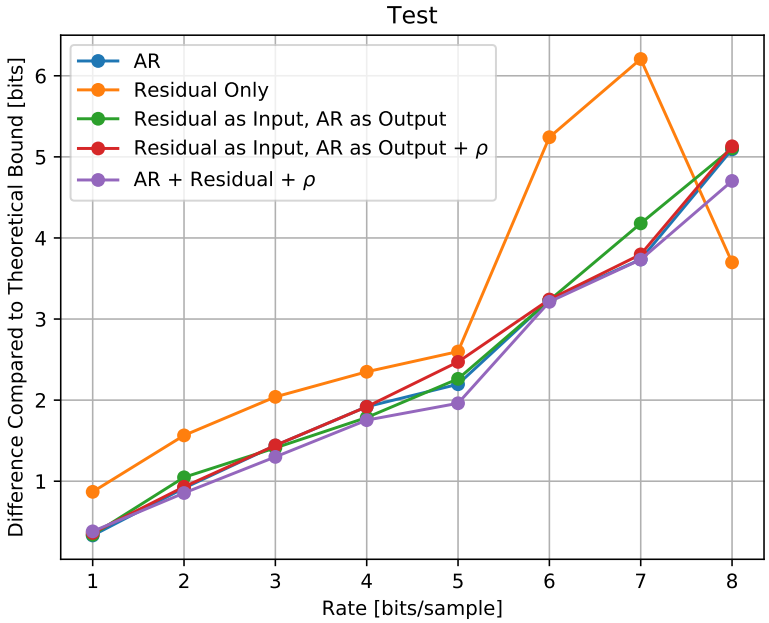
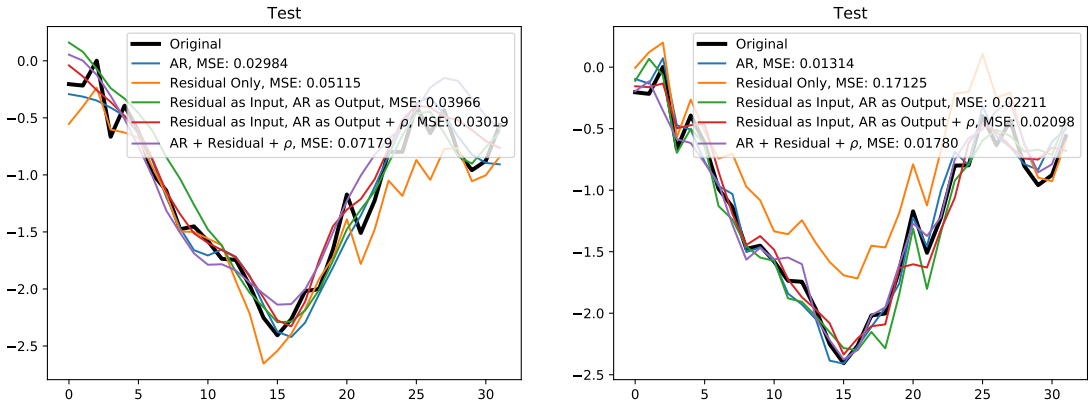
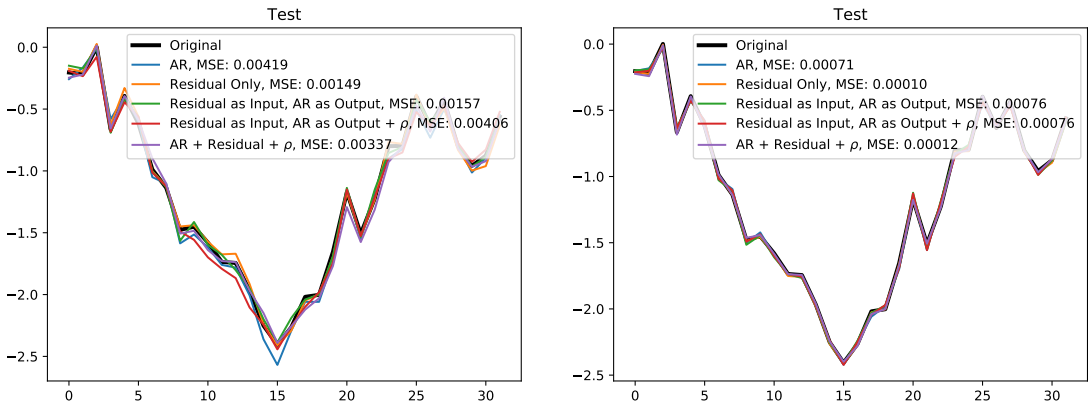


Figure A.6: Difference in bits and the bit rate for the balanced network with different input/output strategies and a frame size of 32.



(a) Bit rate of 1 bits/sample.

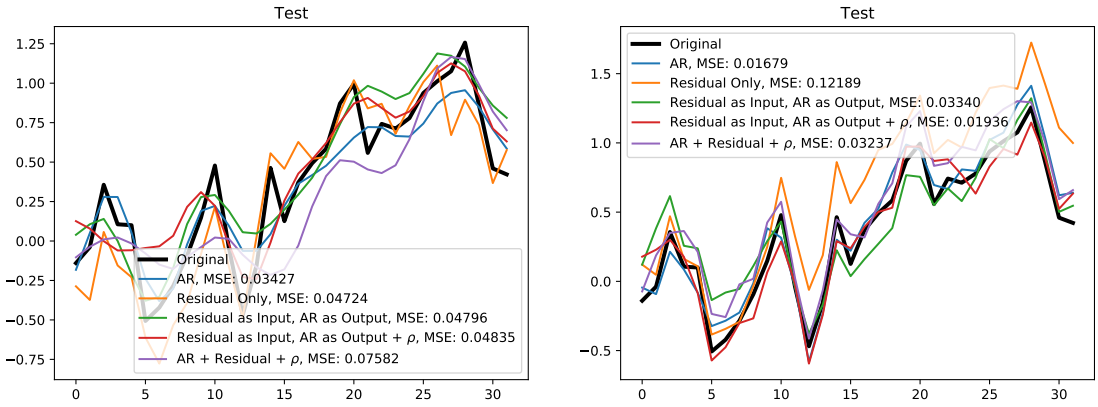
(b) Bit rate of 2 bits/sample.



(c) Bit rate of 4 bits/sample.

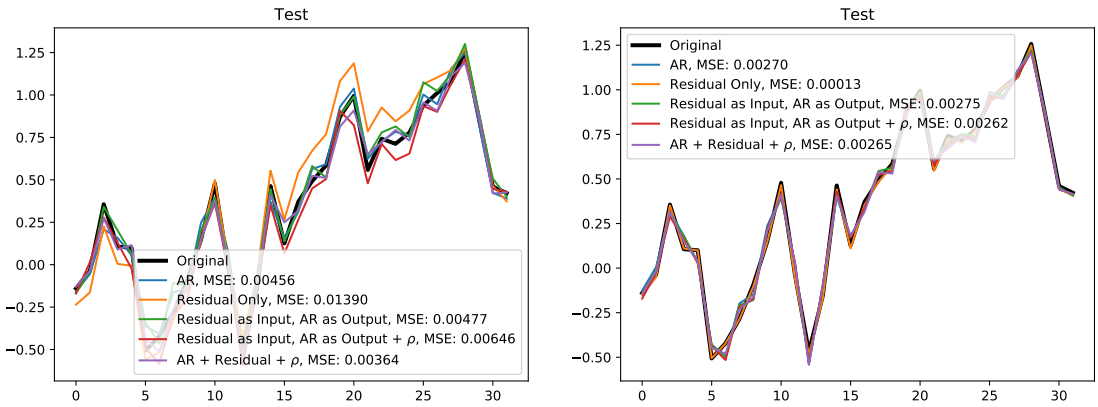
(d) Bit rate of 8 bits/sample.

Figure A.7: Reconstructed signals and the original signal for different input/output strategies and bit rates in the balanced network. The MSE values are calculated for the shown original and reconstructed signals.



(a) Bit rate of 1 bits/sample.

(b) Bit rate of 2 bits/sample.



(c) Bit rate of 4 bits/sample.

(d) Bit rate of 8 bits/sample.

Figure A.8: Reconstructed signals and the original signal for different input/output strategies and bit rates in the balanced network. The MSE values are calculated for the shown original and reconstructed signals.

A.2 Undercomplete Network

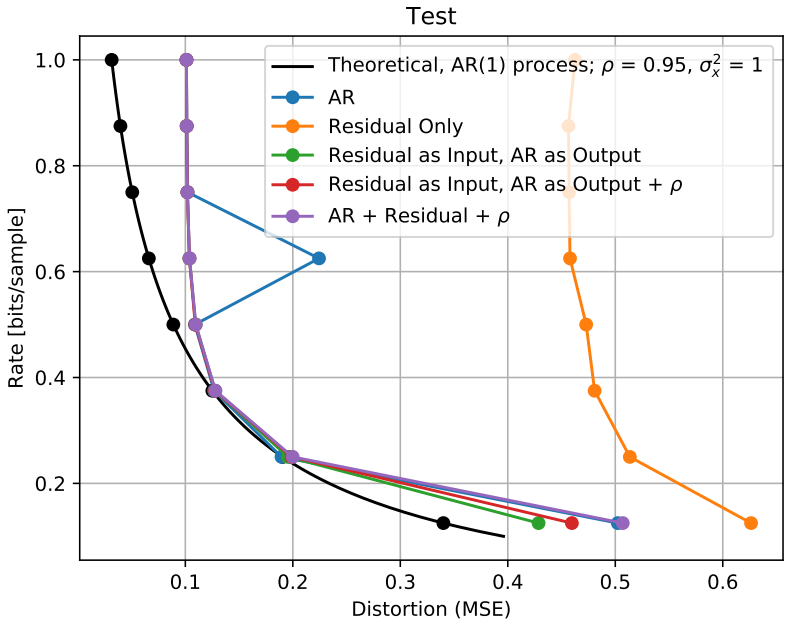


Figure A.9: Operational rate-distortion curves for the undercomplete network with different input/output strategies and a frame size of 16.

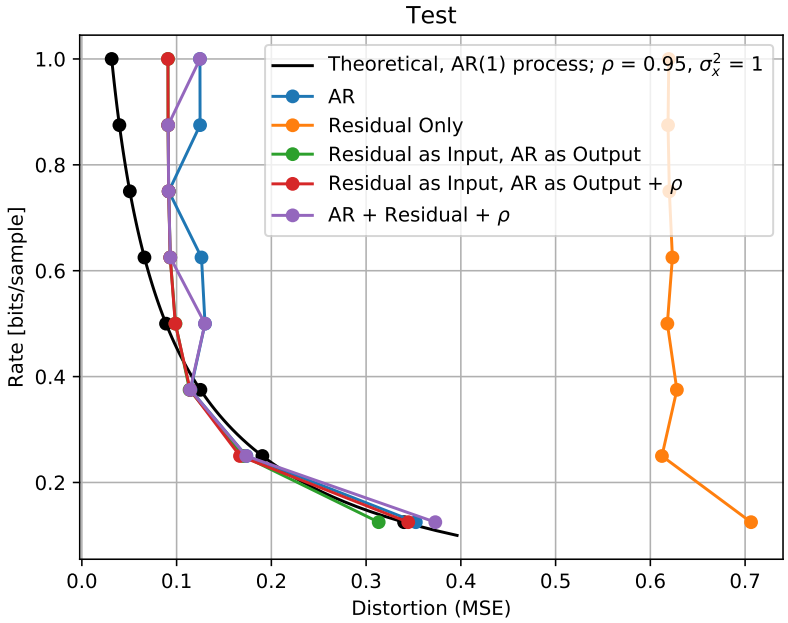


Figure A.10: Operational rate-distortion curves for the undercomplete network with different input/output strategies and a frame size of 32.

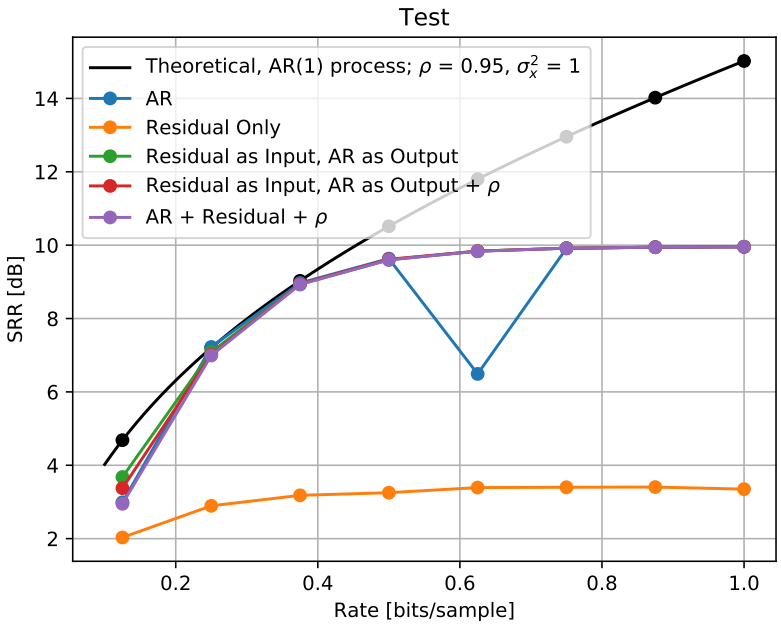


Figure A.11: SRR and bit rates for the undercomplete network with different input/output strategies and a frame size of 16.

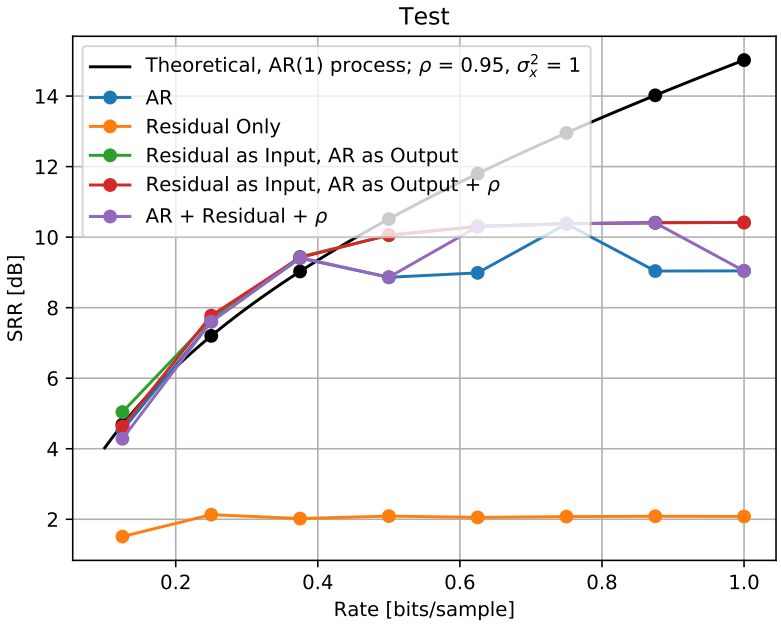


Figure A.12: SRR and bit rates for the undercomplete network with different input/output strategies and a frame size of 32.

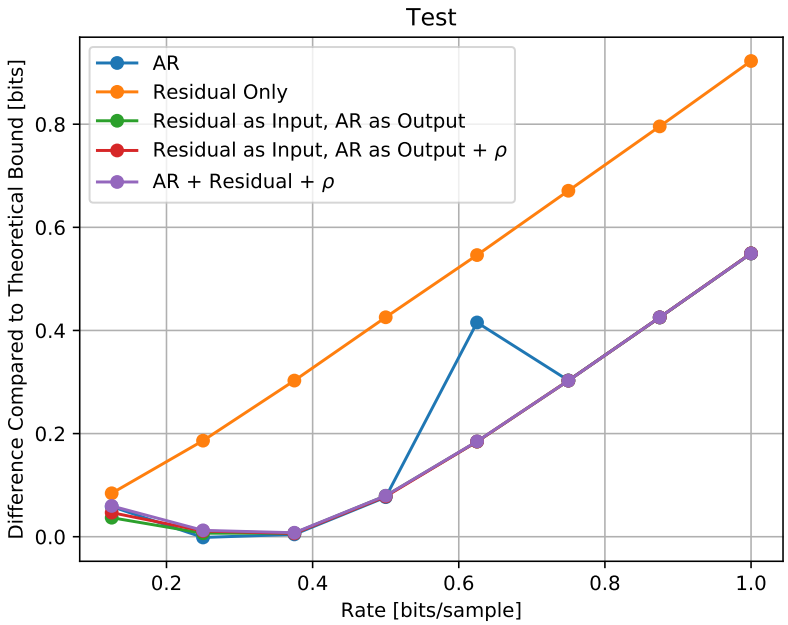


Figure A.13: Difference in bits and the bit rate for the undercomplete network structure and a frame size of 16.

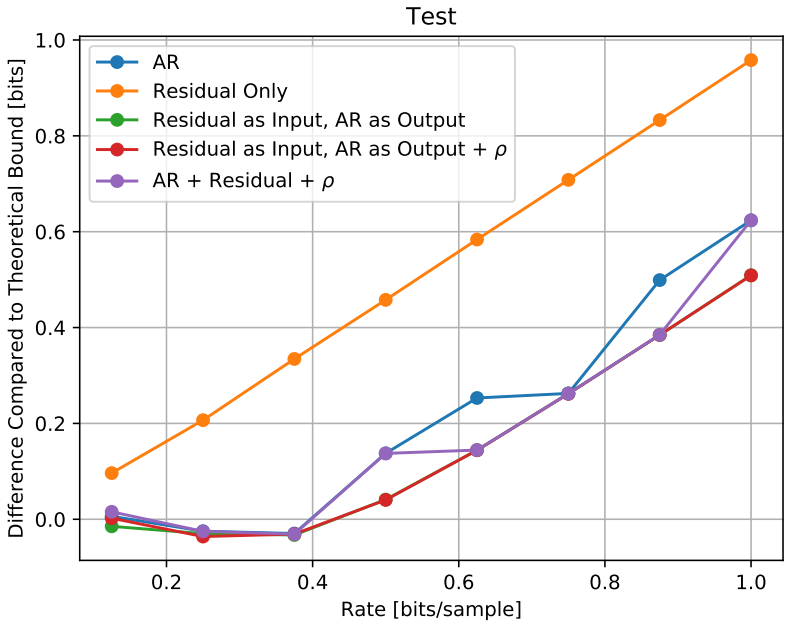
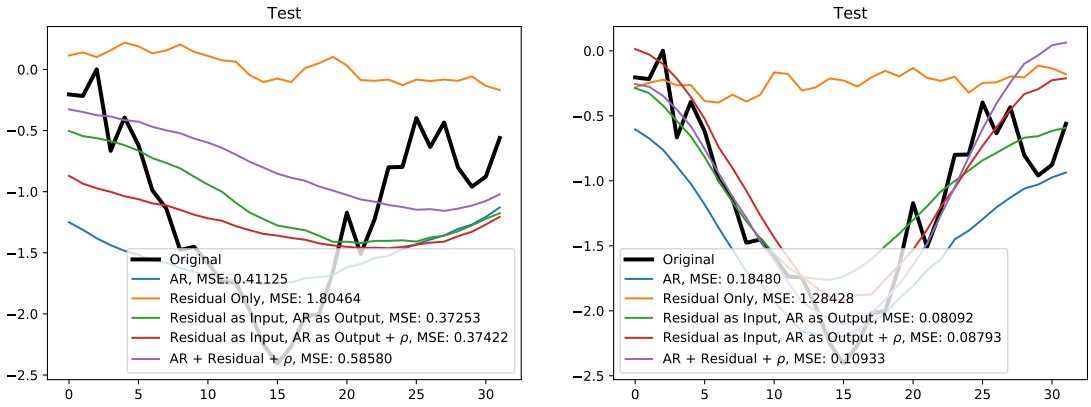
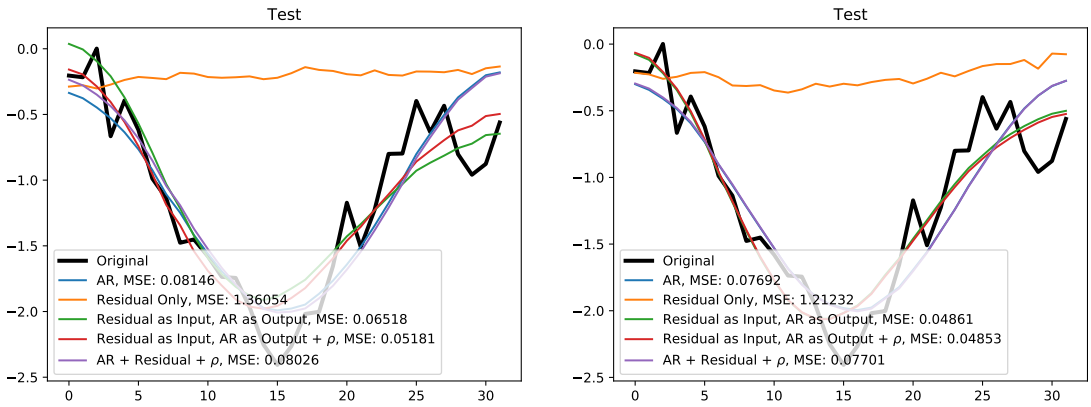


Figure A.14: Difference in bits and the bit rate for the undercomplete network structure and a frame size of 32.



(a) Bit rate of 0.125 bits/sample.

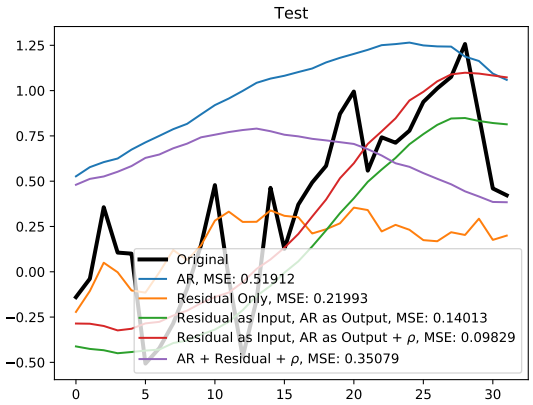
(b) Bit rate of 0.25 bits/sample.



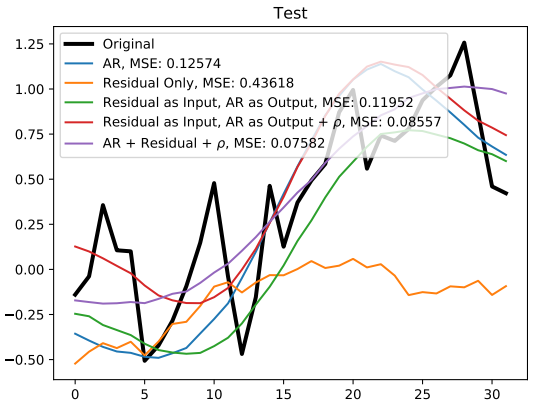
(c) Bit rate of 0.5 bits/sample.

(d) Bit rate of 1 bits/sample.

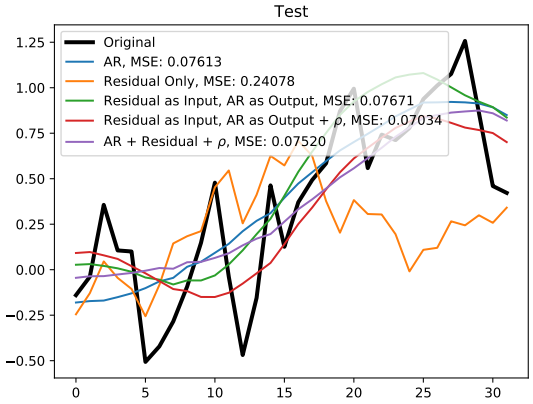
Figure A.15: Reconstructed signals and the original signal for different input/output strategies and bit rates in the undercomplete network.



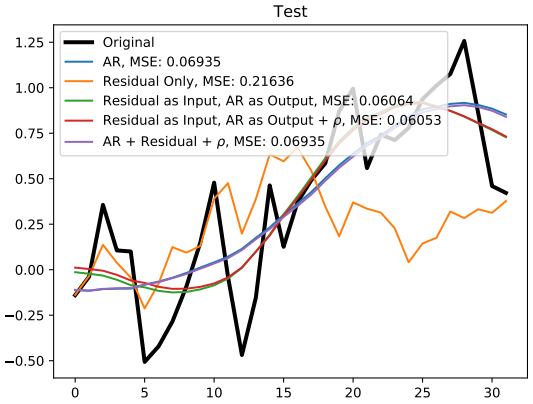
(a) Bit rate of 0.125 bits/sample.



(b) Bit rate of 0.25 bits/sample.



(c) Bit rate of 0.5 bits/sample.



(d) Bit rate of 1 bits/sample.

Figure A.16: Reconstructed signals and the original signal for different input/output strategies and bit rates in the undercomplete network.

B. Spectrograms of Reconstructed Signals

In this chapter spectrograms of reconstructed signals for the networks in Section 9.1 are shown.

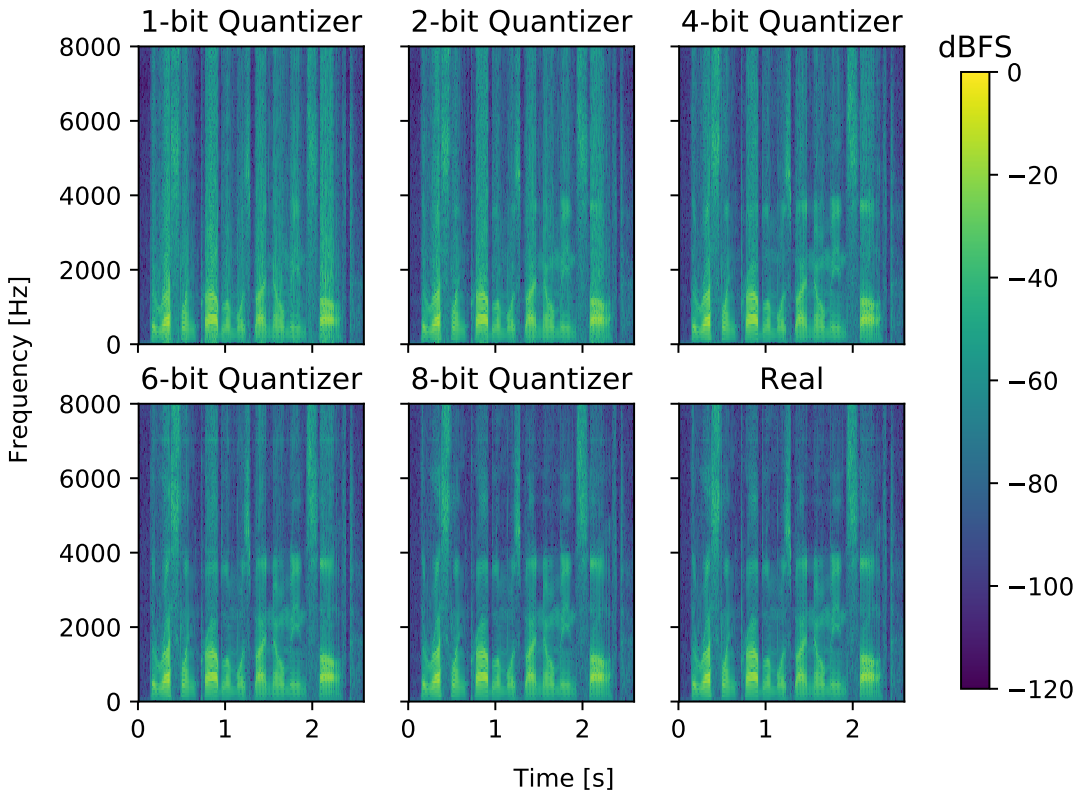


Figure B.1: Spectrograms of a original signal (bottom right) and the corresponding reconstructed signal using networks with different b-bit quantizers.

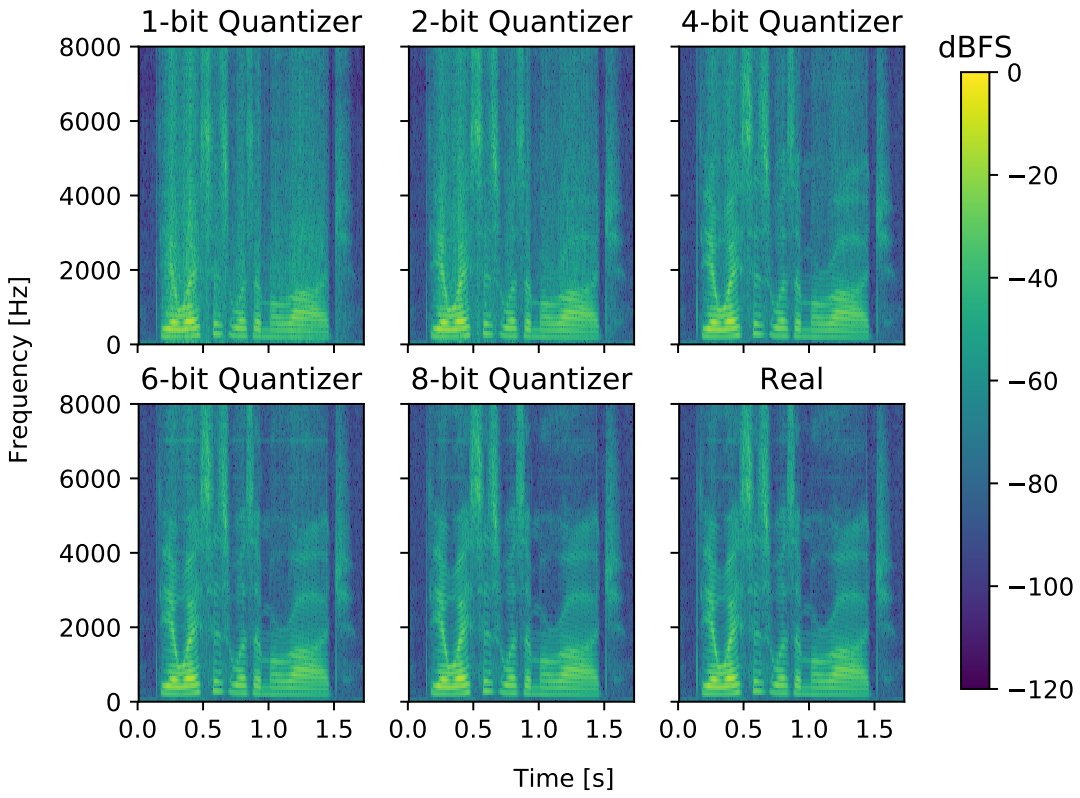


Figure B.2: Spectrograms of a original signal (bottom right) and the corresponding reconstructed signal using networks with different b-bit quantizers.

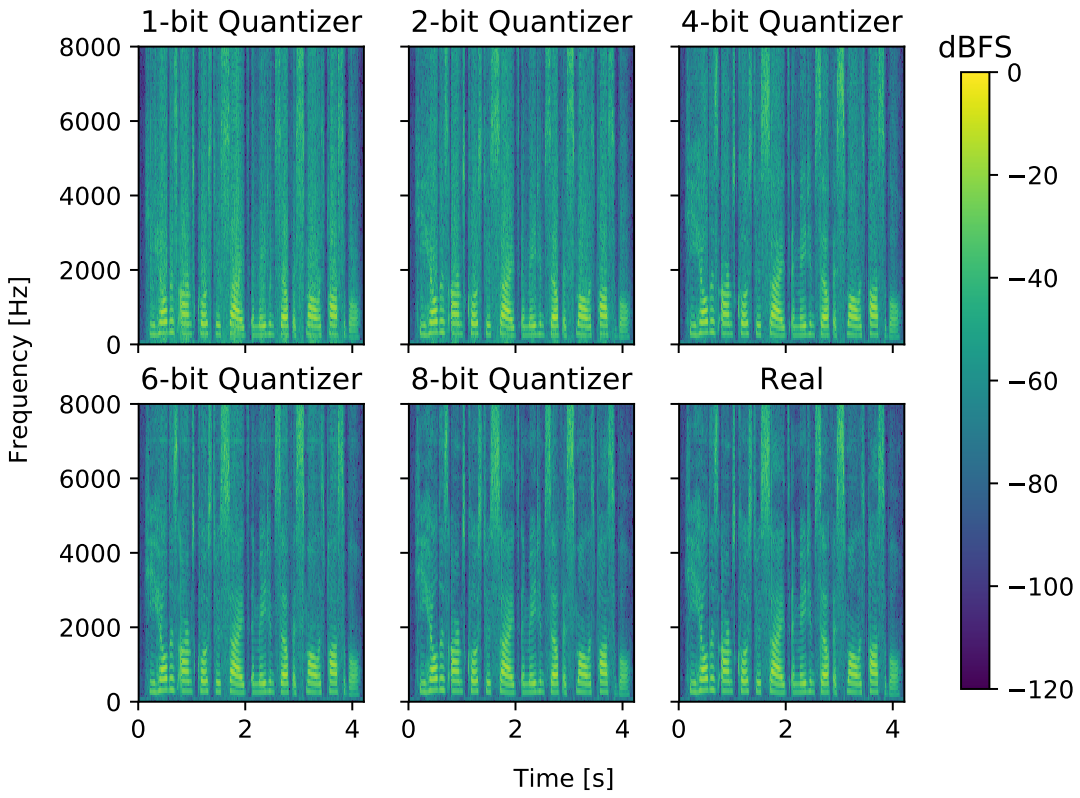


Figure B.3: Spectrograms of a original signal (bottom right) and the corresponding reconstructed signal using networks with different b-bit quantizers.

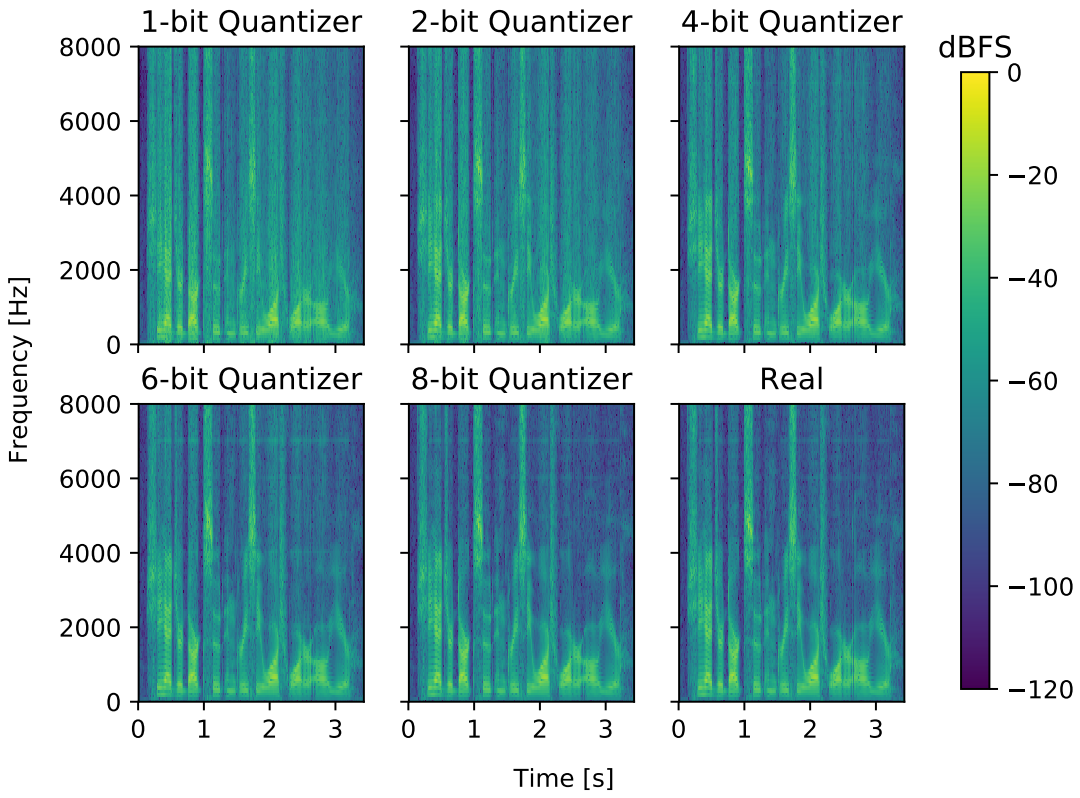


Figure B.4: Spectrograms of a original signal (bottom right) and the corresponding reconstructed signal using networks with different b-bit quantizers.

C. Information Plane Plots

In this appendix all information plots for the networks in Section 9.3 are shown.

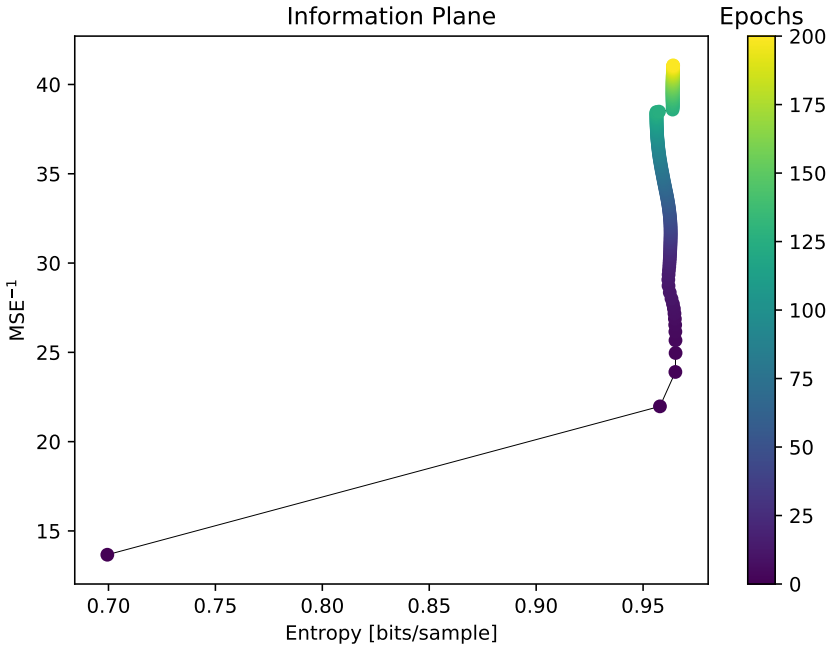


Figure C.1: Information plane plot for the training data and for networks with a frame size and quantization dimension of 128 and a 1-bit quantizer. The x-axis is the entropy of the quantization layer in bits per sample, while the y-axis is the MSE^{-1}.

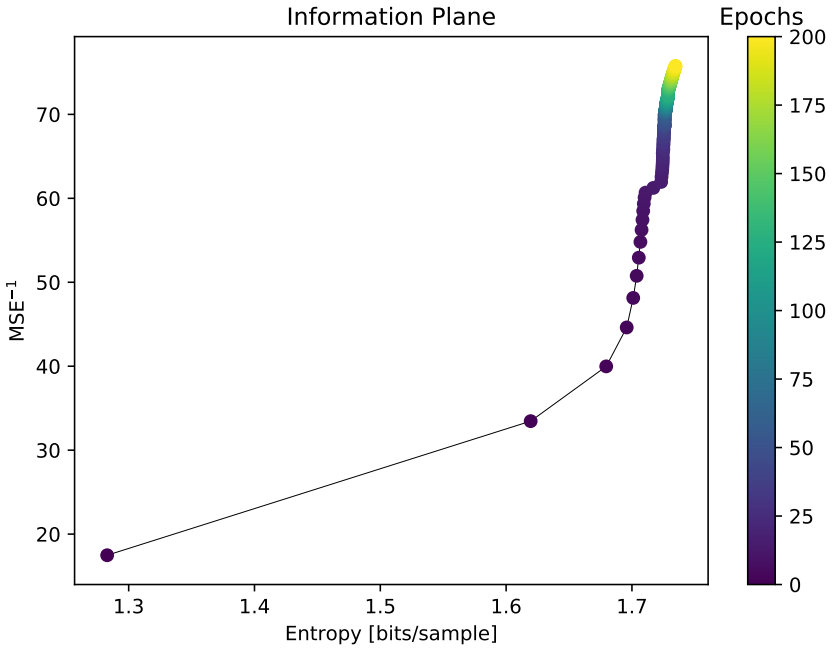


Figure C.2: Information plane plot for the training data and for networks with a frame size and quantization dimension of 128 and a 2-bit quantizer. The x-axis is the entropy of the quantization layer in bits per sample, while the y-axis is the MSE^{-1}.

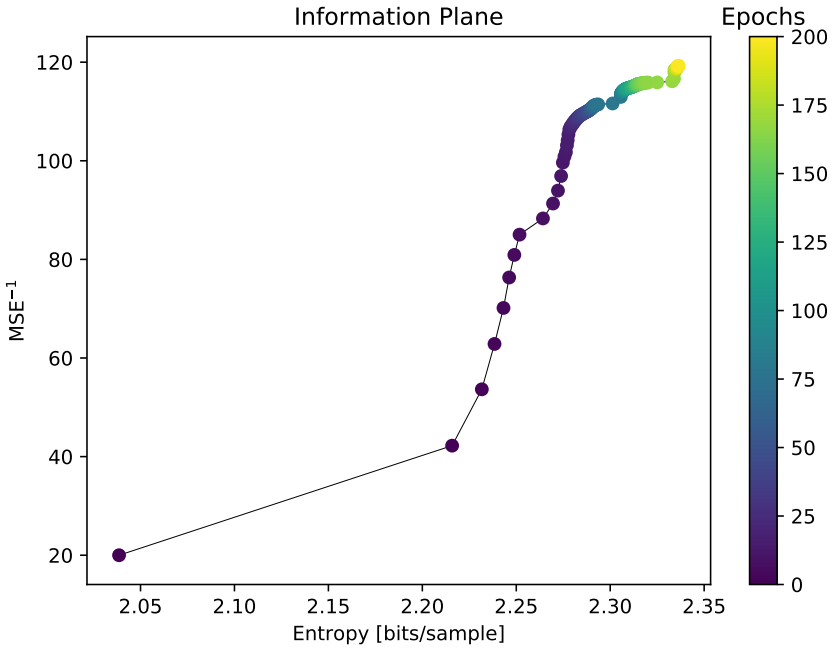


Figure C.3: Information plane plot for the training data and for networks with a frame size and quantization dimension of 128 and a 3-bit quantizer. The x-axis is the entropy of the quantization layer in bits per sample, while the y-axis is the MSE^{-1}.

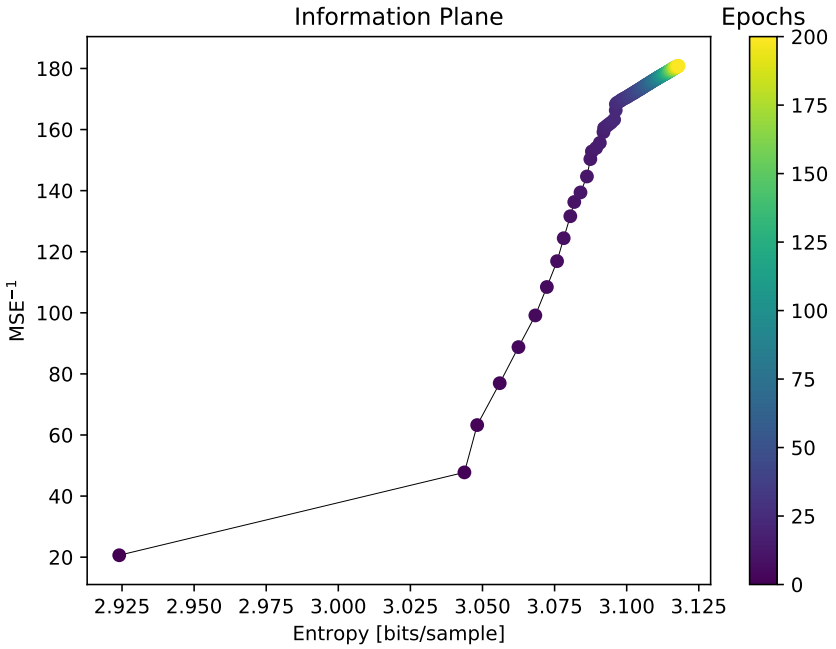


Figure C.4: Information plane plot for the training data and for networks with a frame size and quantization dimension of 128 and a 4-bit quantizer. The x-axis is the entropy of the quantization layer in bits per sample, while the y-axis is the MSE^{-1}.

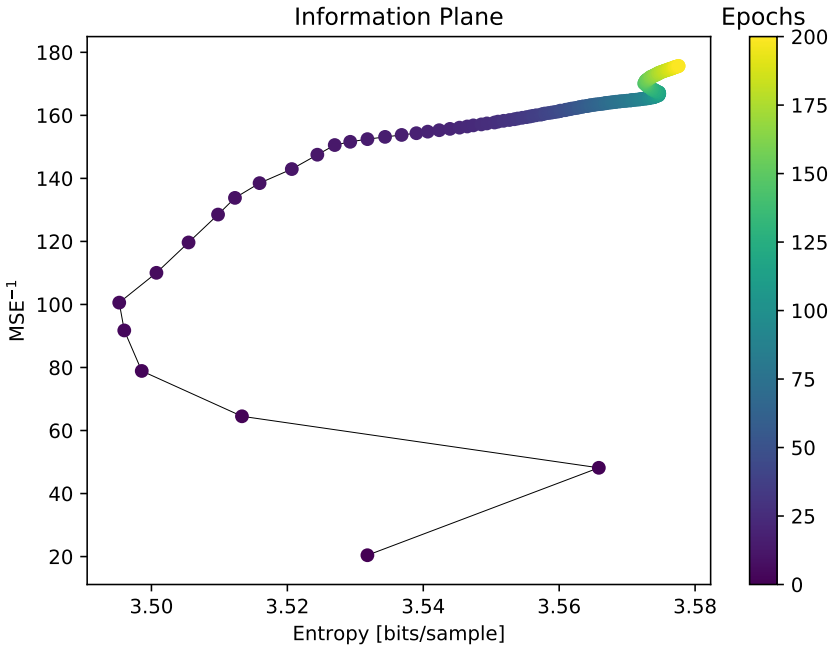


Figure C.5: Information plane plot for the training data and for networks with a frame size and quantization dimension of 128 and a 5-bit quantizer. The x-axis is the entropy of the quantization layer in bits per sample, while the y-axis is the MSE^{-1}.

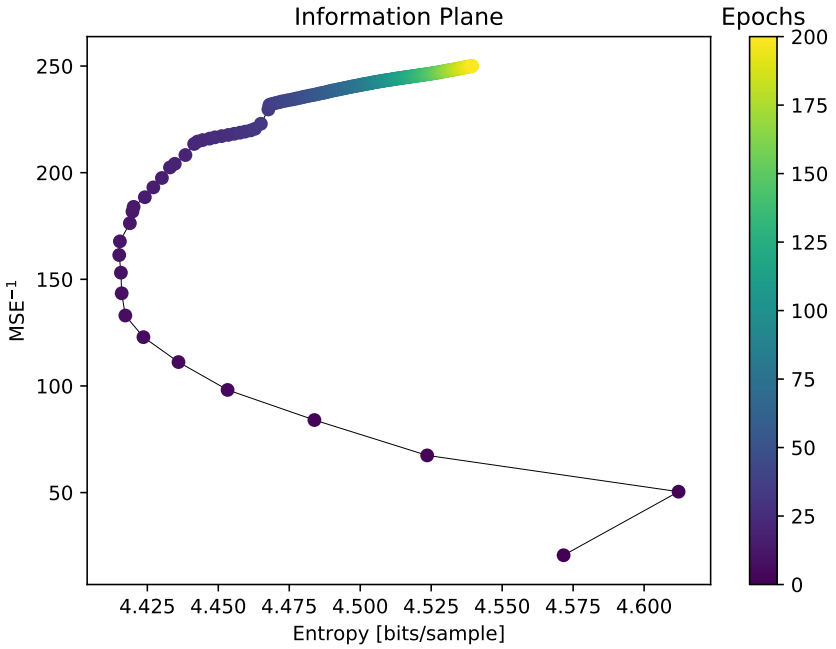


Figure C.6: Information plane plot for the training data and for networks with a frame size and quantization dimension of 128 and a 6-bit quantizer. The x-axis is the entropy of the quantization layer in bits per sample, while the y-axis is the MSE^{-1}.

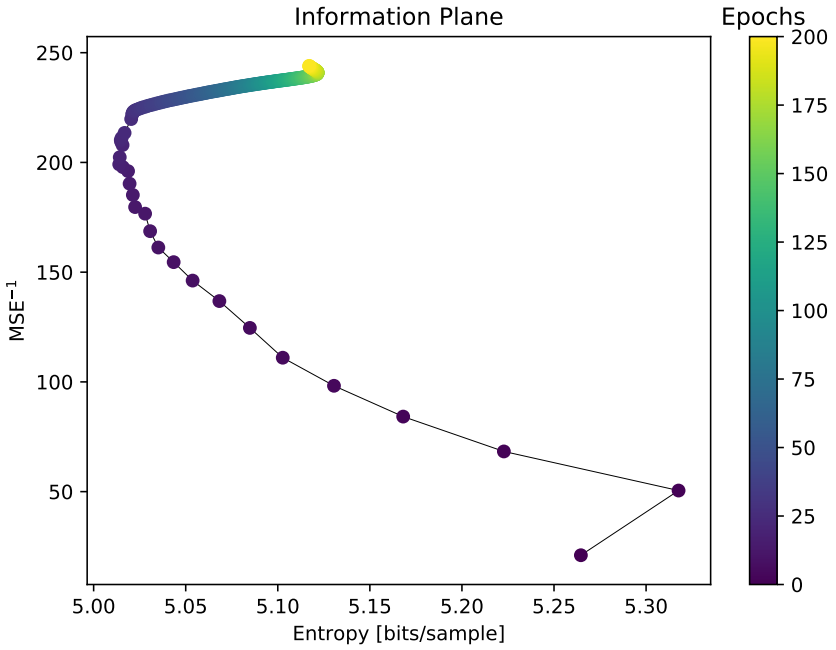


Figure C.7: Information plane plot for the training data and for networks with a frame size and quantization dimension of 128 and a 7-bit quantizer. The x-axis is the entropy of the quantization layer in bits per sample, while the y-axis is the MSE^{-1}.

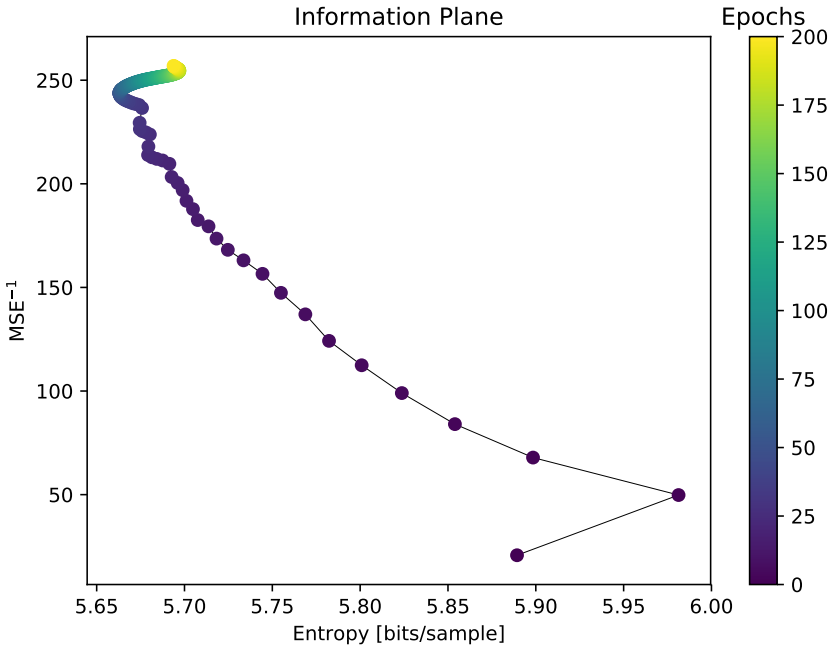


Figure C.8: Information plane plot for the training data and for networks with a frame size and quantization dimension of 128 and a 8-bit quantizer. The x-axis is the entropy of the quantization layer in bits per sample, while the y-axis is the MSE^{-1}.