Vector quantization in deep neural networks for speech and image processing

Mohammad Hassan Vali

Aalto University

Aalto University publication series
Doctoral Theses 13/2025

Vector Quantization in Deep Neural Networks for Speech and Image Processing

Mohammad Hassan Vali

A doctoral thesis completed for the degree of Doctor of Science (Technology) to be defended, with the permission of the Aalto University School of Electrical Engineering, at a public examination held on 7 February 2025.

Aalto University
School of Electrical Engineering
Department of Information and Communications Engineering
Speech Interaction Technology Group

Supervising professor
Prof. Tom Bäckström, Aalto University, Finland

Preliminary examiners
Prof. Tuomas Virtanen, Tampere University, Finland
Prof. Jennifer Williams, University of Southampton, United Kingdom

Opponent
Dr. Jean-Marc Valin, Senior Research Scientist at Google, Montreal, Canada

Aalto University publication series
Doctoral Theses 13/2025

Image on the cover: Mohammad Ali Mirbagheri / Instagram @mirbagheri.ph

ISBN 978-952-64-2361-6 (printed)
ISBN 978-952-64-2362-3 (pdf)
ISSN 1799-4934 (printed)
ISSN 1799-4942 (pdf)

http://urn.fi/URN:ISBN:978-952-64-2362-3

Unigrafia Oy
Helsinki 2025

Finland

Author
Mohammad Hassan Vali

Name of the doctoral thesis
Vector Quantization in Deep Neural Networks for Speech and Image Processing

Thesis format
Article-based

Number of pages 139

Keywords
Vector Quantization, Deep Neural Networks, Space-Filling Curves, Space-Filling Vector Quantization, Gradient Collapse, Interpretability, Speaker Anonymization

Abstract
Vector quantization (VQ) is a classic signal processing technique that models the probability density function of a distribution using a set of representative vectors called the codebook, such that each codebook vector represents a subset of the distribution's samples. Deep neural networks (DNNs) are a branch of machine learning that has gained popularity in recent decades as they can solve complicated optimization problems. Since VQ provides an abstract high-level discrete representation of a distribution, it has been widely used as a beneficial tool in many applications based on DNNs, such as image generation, speech recognition, text-to-speech synthesis, and speech and video coding. Regarding VQ's broad utilization in applications based on DNNs, a small improvement in VQ can result in a huge boost in the performance of many applications installed on devices dealing with different data types such as speech, image, video, and text.

This thesis mainly focuses on improving various VQ methods within deep learning frameworks. We propose using vector quantization instead of scalar quantization in a speech coding framework. The experiments show that the decoded speech has a higher perceptual quality because VQ considers the correlation between different dimensions of spectral envelopes. As another contribution, we propose a new solution to the *gradient collapse* problem called noise substitution in vector quantization (NSVQ), in which we model VQ as the addition of a noise vector to the input. Experiments show that NSVQ gives a faster convergence, more accurate gradients, and less hyperparameters to tune than two state-of-the-art solutions, i.e., straight-through estimator and exponential moving average. We further demonstrate that NSVQ can also optimize other variants of VQ that use multiple codebooks, e.g., product VQ, residual VQ, and additive VQ. Experimental results under different speech coding, image compression, and approximate nearest neighbor search show that VQ variants optimized by NSVQ can perform comparably to the baselines.

By incorporating space-filling curves into VQ, we introduced a novel quantization technique called space-filling vector quantization (SFVQ), which quantizes the input on a continuous piecewise linear curve. Because of inherent order in the SFVQ codebook, adjacent codebook vectors refer to similar contents. We used this property of SFVQ, which allows us to interpret the underlying phonetic structure of the latent space of a voice conversion model. Moreover, we used SFVQ to interpret the intermediate latent space of StyleGAN2 and BigGAN image generative models. SFVQ gives good control over generations, such that we found the mapping between the latent space and generative factors (e.g., gender, age, etc.), and we discovered the interpretable directions to change the image's attributes (e.g., smile, pose, etc.). In another work, we used SFVQ to cluster the speaker embeddings to enhance the speaker's privacy in speech processing tools based on DNNs.

Preface

First of all, I would like to express my sincere gratitude to my supervisor, Professor Tom Bäckström, for his strong support and motivation during this thesis. I was really lucky to be supervised by him, as he gave me the opportunity and courage to dive into completely new topics and taught me how to take my steps toward addressing fundamental research problems. I will never forget our deep mathematical discussions through our weekly meetings while getting surprised at his great expertise on the subject. However, Tom's support was not limited to our working atmosphere. He also cares about student's well-being in his research group as we have a nice habit of talking about our feelings every day in our daily group meetings. For these reasons, I see him not only as a supervisor, but also as a friend, and I hope our paths keep crossing in the future.

In addition, I would like to thank my super-nice colleagues Pablo, Abraham, Silas, Esteban, and Mariem, without any specific order. During our time in the office, we had really nice discussions about research which gave me valuable insights. We also had really fun moments during coffee times.

I appreciate Emmanuel Vincent for supervising me during my research visit at Inria research center at university of Lorraine in Nancy, France. I also thank Natalia Tomashenko and Colotte Vincent who collaborated me during this research visit. Thanks to these nice persons, this research visit enriched my research experience and broadened the scope of my work.

I am thankful to the Jane & Aatos Erkko Foundation and also Aalto Electrical Engineering School for providing me with financial support to complete this thesis. Also, I thank the Aalto Scientific Computing team for all their great technical support especially for the Triton computing cluster that made my research experiments much simpler and quicker.

I am grateful to the pre-examiners of my thesis, Professor Tuomas Virtanen and Professor Jennifer Williams, for their thoughtful comments, which contributed to improving the quality of this thesis. Additionally, I would like to thank Professor Jean-Marc Valin for kindly agreeing to serve as the opponent at my defense.

A special mention goes to Reyhane, my wife and soulmate, without whom completing this stage of my academic career was impossible. When I was doubtful whether to accept this Ph.D. position and emigrate from my home country, she made me feel decisive and now I am really proud of making that decision. During the ups and downs of my Ph.D., she was always there to support me emotionally and encouraged me not to give up on even the smallest things. I am highly appreciative of her companionship during all my conference trips and my research visit.

My deepest gratitude goes to my parents for their all-time sacrifices, emotional support, and unbounded love throughout my life. They are the biggest teachers of my life who taught me to always dream big, and shaped my personality to who I am today. I also immensely thank my brother and my sister for their continuous encouragements and unwavering belief in my potential that gave me the confidence to follow my dreams.

Last but not least, I would like to thank my dearest friends who accompanied me during my Ph.D. stage in Finland. We gathered and shared most of our time together on the weekends which made me feel refreshed and relieved from the work tension. I am really proud of having such amazing friends and hope this friendship continues forever.

This dissertation is a testament to the collective efforts of all those mentioned above. Thank you for being an integral part of this journey.

Helsinki, January 3, 2025,

Mohammad Hassan Vali

Contents

List of Publications

This thesis consists of an overview of the following publications which are referred to in the text by their Roman numerals.

I M. H. Vali, T. Bäckström. End-to-End Optimized Multi-Stage Vector Quantization of Spectral Envelopes for Speech and Audio Coding. In *Interspeech 2021*, pp. 3355-3359, Brno, Czechia, August-September 2021.

II M. H. Vali, T. Bäckström. NSVQ: Noise Substitution in Vector Quantization for Machine Learning. *IEEE Access*, Volume 10, pp. 13598-13610, January 2022.

III M. H. Vali, T. Bäckström. Stochastic Optimization of Vector Quantization Methods in Application to Speech and Image Processing. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP) 2023*, pp. 1-5, Rhodes, Greece, June 2023.

IV M. H. Vali, T. Bäckström. Interpretable Latent Space Using Space-Filling Curves for Phonetic Analysis in Voice Conversion. In *Interspeech 2023*, pp. 306-310, Dublin, Ireland, August 2023.

V M. H. Vali, T. Bäckström. Unsupervised Panoptic Interpretation of Latent Spaces in GANs Using Space-Filling Vector Quantization. Submitted to *Conference on Computer Vision and Pattern Recognition*, 10 pages, November 2024.

VI M. H. Vali, T. Bäckström. Privacy PORCUPINE: Anonymization of Speaker Attributes Using Occurrence Normalization for Space-Filling Vector Quantization. In *Interspeech 2024*, pp. 2230-2234, Kos, Greece, September 2024.

Author's Contribution

Publication I: "End-to-End Optimized Multi-Stage Vector Quantization of Spectral Envelopes for Speech and Audio Coding"

The author had the main responsibility for the development and implementation of the proposed method. He also designed the experimental scenarios, carried out the experiments and the relevant evaluations. The author was the main writer of the article.

Publication II: "NSVQ: Noise Substitution in Vector Quantization for Machine Learning"

The author had the main responsibility for the development and implementation of the proposed method. He also designed the experimental scenarios, carried out the experiments and the relevant evaluations. The author was the main writer of the article.

Publication III: "Stochastic Optimization of Vector Quantization Methods in Application to Speech and Image Processing"

The author had the main responsibility for the development and implementation of the proposed method. He also designed the experimental scenarios, carried out the experiments and the relevant evaluations. The author was the main writer of the article.

Publication IV: "Interpretable Latent Space Using Space-Filling Curves for Phonetic Analysis in Voice Conversion"

The author had the main responsibility for the development and implementation of the proposed method. He also designed the experimental scenarios, carried out the experiments and the relevant evaluations. The author was the main writer of the article.

Publication V: "Unsupervised Panoptic Interpretation of Latent Spaces in GANs Using Space-Filling Vector Quantization"

The author had the main responsibility for the development and implementation of the proposed method. He also designed the experimental scenarios, carried out the experiments and the relevant evaluations. The author was the main writer of the article.

Publication VI: "Privacy PORCUPINE: Anonymization of Speaker Attributes Using Occurrence Normalization for Space-Filling Vector Quantization"

The author had the main responsibility for the development and implementation of the proposed method. He also designed the experimental scenarios, carried out the experiments and the relevant evaluations. The author wrote sections 4, 5, and 6 of the article and prepared all of the figures.

List of Figures

List of Tables

Abbreviations

AdaGrad Adaptive Gradient

Adam Adaptive Moment

ANN Approximate Nearest Neighbor

AQE Average Quantization Error

AVQ Additive Vector Quantization

CNN Convolutional Neural Network

DBSCAN Density-Based Spatial Clustering of Applications with Noise

DNN Deep Neural Network

EA Evolutionary Algorithm

EM Expectation Maximization

EMA Exponentially Moving Average

GAN Generative Adversarial Network

GPU Graphics Processing Unit

LSTM Long Short-Term Memory

ML Machine Learning

MLP Multi-Layer Perceptron

MSE Mean Squared Error

NN Neural Network

NSVQ Noise Substitution in Vector Quantization

PESQ Perceptual Evaluation of Speech Quality

PDF Probability Density Function

PVQ Product Vector Quantization

RMSProp Root Mean Square Propagation

RNN Recurrent Neural Network

RVQ Residual Vector Quantization

SFVQ Space-Filling Vector Quantization

SGD Stochastic Gradient Descent

STE Straight-Through Estimator

TSP Traveling Salesman Problem

VQ Vector Quantization

VQ-VAE Vector Quantized Variational Autoencoder

Symbols

Greek symbols

α learning rate of the optimizer

γ smoothing factor of the exponential moving average

π_i weight of the i-th codebook vector occurrence

τ annealing (or temperature) factor of soft quantization

Latin symbols

B quantization bitrate

CB codebook matrix

CB_i i-th codebook vector

CB^j j-th codebook matrix

Cr crossover rate in differential evolution algorithm

D dimension

F mutation factor in differential evolution algorithm

G generation index in differential evolution algorithm

$H(y)$ entropy of random variable y

I tuple of codebook indices

K number of codebook vectors

M number of quantization stages

$\mathcal{N}(0,1)$ normal distribution

N_b number of samples in a batch

N_{batch} number of batches

N_{epoch} number of training epochs

N_g number of generations in differential evolution algorithm

N_p population size in differential evolution algorithm

$p(y)$ probability of random variable y

Q quantizer

$\mathbb{R}$ set of real values

t time index

T_i^G i-th trial solution in generation G of differential evolution algorithm

V_i^G i-th mutant solution in generation G of differential evolution algorithm

W beam width

x input vector

x_r reconstructed input vector at the decoder output

$\hat{x}$ quantized input vector

$\mathbb{X}$ set of possible input vectors

X_i^G i-th target solution in generation G of differential evolution algorithm

z_e continuous latent vector

z_q quantized latent vector

Operands

$\|v\|_m$ m-th norm of the vector v (ℓ_m norm)

$\frac{\partial f(a)}{\partial a}$ gradient (or derivative) of $f(a)$ with respect to the variable a

$d(x,\hat{x})$ distance measure between vectors x and $\hat{x}$

E expectation operand

$sg(.)$ stop gradient

1. Introduction

Vector quantization (VQ) is a classic data compression technique that is widely used in the signal processing domain. The main objective of VQ is to model the probability density function of a distribution by a set of codebook vectors such that each codebook vector is the representative vector for a group of distribution samples. In other words, VQ compresses a subset of data samples (existing in a Voronoi cell) by representing them with their corresponding codebook vector, which is the closest codebook vector in terms of a distance metric. The optimization goal of VQ is to find a set of codebook vectors that minimizes the distance between all data samples and codebook vectors. The Euclidean distance is commonly used as the distance metric for VQ. Fig. 1.1 shows a VQ applied on a 2D Gaussian distribution using 32 codebook vectors.

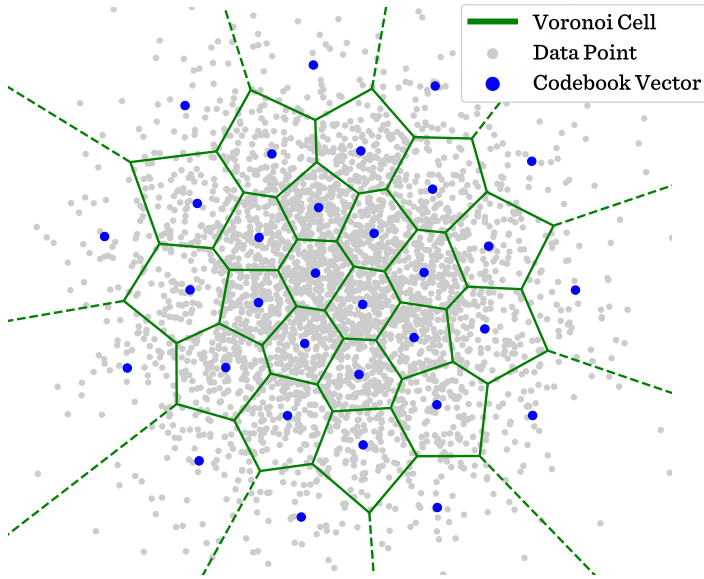

Figure 1.1. Vector quantization of a 2D Gaussian distribution using 5 bits (32 codebook vectors). Each Voronoi cell accommodates only one codebook vector that is the closest codebook vector to all data points inside that cell. All data points inside each Voronoi cell will be mapped to their corresponding codebook vector. Figure adapted from [1].

Deep neural networks (DNNs) is a successful branch of the machine learning field that has gained popularity in recent decades. The massive increase in digital data availability and significant advancements in computational resources (especially GPUs) are two main factors that play a crucial role in the success of DNNs. DNNs can extract complex non-linear patterns from data, and thus, they use these patterns to solve highly complicated problems. This ability enables them to obtain state-of-the-art performance in a broad range of domains [2].

VQ presents an ideal abstract discrete representation of a distribution. Hence, it can be a good fit for modeling any categorical distribution regardless of the data type (e.g., speech, image, video, text, etc.). This is why VQ is widely used in various DNN architectures for a broad range of applications such as image generation [3, 4], speech and audio coding [5, 6], speech recognition [7], music generation [8], text-to-speech synthesis [9, 10], and video generation [11]. According to this broad applicability of VQ in DNNs, any slight improvement in VQ efficiency will certainly result in a significant performance gain in all DNN-based applications for different data types.

1.1 Objectives

In general, the main contributions of the research in this thesis is to improve the perception quality of decoded speech in a DNN-based speech coding that employ scalar quantization, to resolve the *gradient collapse* problem when training DNNs that use VQ or other VQ variants (e.g., product VQ) in a more efficient way, to interpret the latent spaces of DNNs by our new proposed quantization method called space-filling vector quantization (SFVQ), and to enhance the speakers' privacy in DNN-based speech processing tolls which employ ordinary VQ. These contributions can be categorized into four different categories as follows. Fig. 1.2 represents an abstract visualization of the research works done under these four categories.

1. Improvement in performance: When representing a data sample by a vector, there might be correlations among different dimensions of that vector due to the existing patterns in the data distribution. However, this important fact is neglected in some of the existing methods as they quantize all dimensions independently (scalar quantization) [12] or divide the vector into several subvectors with smaller dimensionality (product VQ) [7, 13]. Therefore, in such circumstances, VQ methods that do not split a vector into subvectors can potentially result in less quantization error because they take the correlation between all data dimensions into account.

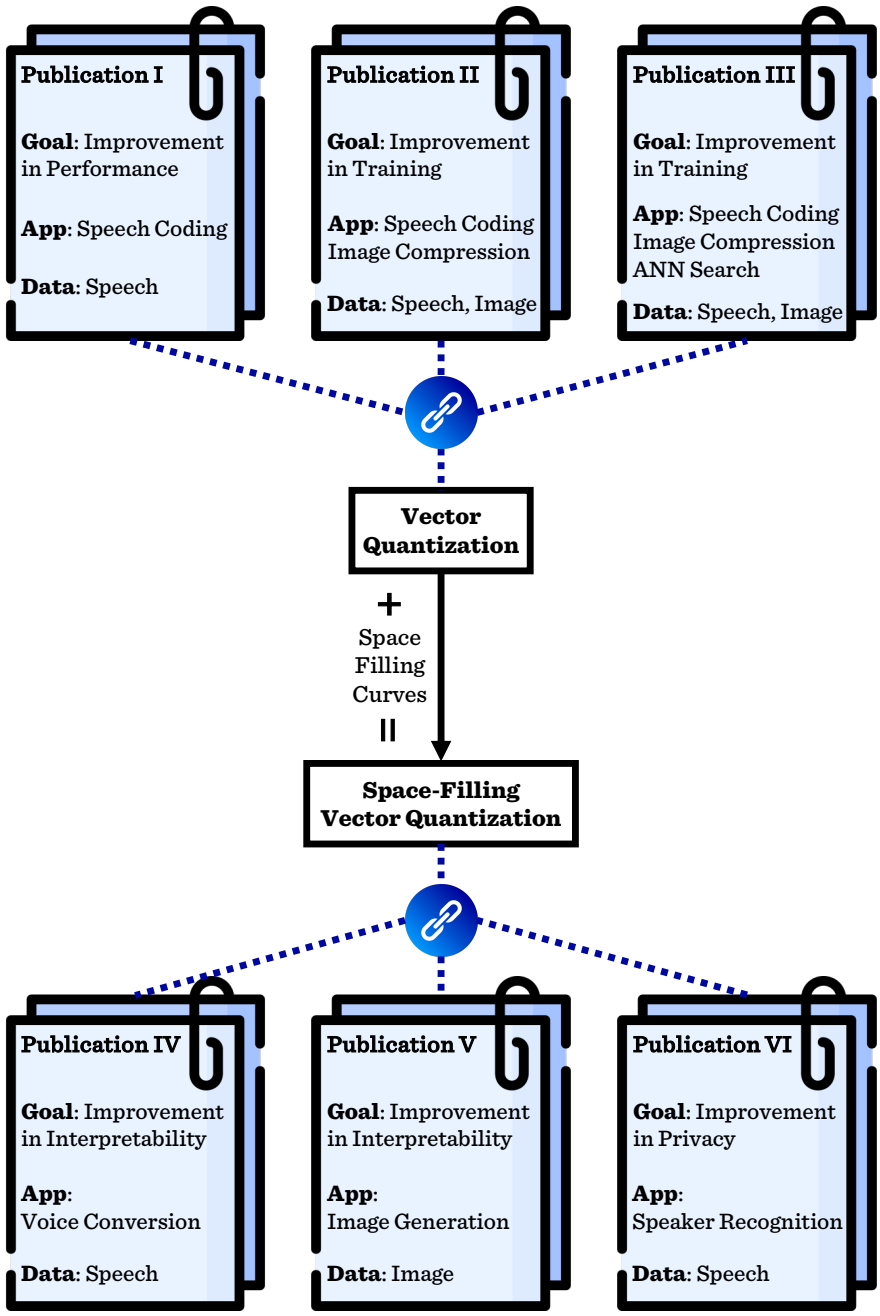

Figure 1.2. Visualization of the research works in the thesis.

In Publication I, we studied a speech coding scenario [12] in which scalar quantization is used to model the spectral envelope of the speech signal (smoothed shape of the speech signal in the frequency domain). By replacing scalar quantization method with VQ, the perception quality of the final compressed speech signal is improved. We also conducted some experiments to compress several distributions with various VQ methods over different dimensions in Publication III. Regarding the experimental results, the compressed distributions by residual VQ [14] and additive VQ [15] give less quantization distortion than the ones compressed by product VQ [13].

2. Improvement in training: The computational graph in DNNs defines the mathematical relationship between input and output. When training a DNN, all functions in the computational graph must be differentiable and have a continuous derivative. However, VQ is not a smooth function and has zero derivatives almost for all inputs. Therefore, the computed gradients for trainable parameters equal zero and thus, the DNN model cannot be trained. This issue is known as *gradient collapse* problem.

In Publication II, we proposed a new solution to address the *gradient collapse* problem called noise substitution in vector quantization (NSVQ), which simulates the behavior of the VQ function in a differentiable way. NSVQ improves the training efficiency of a DNN over two state-of-the-art methods (i.e., straight-through estimator [16] and exponential moving average [11]) such that it provides faster convergence, more accurate simulation of VQ, and it requires less parameter tunings. In Publication III, NSVQ is used to optimize other variants of VQ, which perform quantization with several codebooks (i.e., Product VQ [13], Residual VQ [14], and additive VQ [15]). Experiments show that NSVQ can optimize VQ variants efficiently in different DNN-based speech and image applications such that they perform comparable to the baseline methods.

3. Improvement in interpretability: Each DNN is made up of a series of layers: the input layer where the input data is fed to the network, the output layer that receives the final output of the network, and hidden layers that are located between input and output layers such that they compute the intermediate computations (see Fig. 3.3). One of the typical problems in DNNs is that they act as a black box such that for a given data sample as input to the network, it is not clear what information each hidden layer represents. Therefore, DNNs are not interpretable models since it is difficult or impossible to interpret the information represented in their hidden layers.

By combining space-filling curves and VQ concepts, in Publication IV, we introduced a novel technique called space-filling vector quantization (SFVQ), which helps to interpret the representations of hidden layers. Ac-

cording to the inherent structure in space-filling curves, SFVQ has a desirable arrangement in its codebook vectors which is used for interpretation purposes. In Publication IV, SFVQ grants a useful interpretation of the underlying phonetic structure in the latent space of a voice conversion task based on vector quantized variational autoencoder. Furthermore, in Publication V, SFVQ is used to interpret the latent spaces of two well-known deep image generative models, i.e., StyleGAN2 [17] and BigGAN [18]. The experiments show that SFVQ allows for a desirable interpretation of the latent spaces such that it reveals the mapping between latent space and generative factors (e.g., age, gender, zoom, pose, etc.). In addition, SFVQ discovers the interpretable directions in the latent space to change image attributes such as age, smile, hair color, beard for FFHQ [19] face dataset.

4. Improvement in privacy: Speech is the main medium for human interactions which, apart from its linguistic content, also contains private side information such as the speaker's gender, identity, state of health, and emotion. Today, a wide range of DNN-based speech processing applications exist that take the full speech data as input and process it for their downstream task, whereas they can cause privacy issues for the speakers by disclosing their private information. Hence, it is necessary to consider privacy matters when using DNNs for speech processing applications.

In DNN-based speech processing models, to remove the private information that is not needed for the downstream task, one common approach is to discretize the hidden layer's representation with VQ as in vector quantized variational autoencoder architecture. In such models, one potential issue is that speech signals from a small subset of speakers can be mapped to a specific codebook index. This is a privacy issue because the disclosure of private information will be higher for those speakers mapped to this specific codebook index. In other words, the desirable case to preserve the speakers' privacy is that the occurrence of codebook indices is uniform.

To solve this privacy problem in DNNs, in Publication VI we used our proposed SFVQ and resampled its codebook to equalize the codebook index occurrence and make the codebook vectors to be selected uniformly over all different speakers. In this way, the speaker's privacy is enhanced in DNN-based speech processing applications that employ VQ.

1.2 Thesis Structure

This thesis is comprised of six chapters that explain the essential background concepts, existing challenges, and state-of-the-art solutions, which helps to gain a comprehensive understanding of the proposed ideas in the publications and what the motivation is behind them. The thesis starts with this introduction chapter, which describes why this area of research is important, what some of the current challenges are, and the main objectives and contributions of this thesis.

Chapter 2 discusses vector quantization, its evaluation metrics, and different ways to optimize the VQ codebook vectors. It also describes other variants of VQ (i.e., product VQ, residual VQ, and additive VQ), which use multiple codebook vectors for a higher bitrate quantization as well as variable bitrate VQ. Chapter 3 describes the basics of training procedure in machine learning, neural networks as a widely-used tool in machine learning, and neural network architectures that adopt VQ. Furthermore, this section explains the *gradient collapse* problem, which is the main challenge preventing the optimization of VQ-based neural networks by machine learning optimizers, and also the state-of-the-art solutions to this problem.

Chapter 4 discusses space-filling curves, some of their well-known examples, and their applications. Chapter 5 gives a summary of all publications forming this dissertation. Conclusions drawn from the thesis are presented in Chapter 6. Finally, all of our six publications are attached at the end of the dissertation.

2. Vector Quantization

Vector quantization (VQ) is a tool that is widely used in the signal processing domain for data compression or modeling a data distribution. This chapter explains how VQ works, which applications it is used for, and the most common way to evaluate it. In addition, the chapter includes different approaches to optimize VQ, including k-means, evolutionary algorithms, and gradient-based optimizers. Later in this chapter, we point out some recognized variants of VQ and how to exploit variable-rate VQ in various applications. To start the chapter, we briefly explain scalar quantization to get an intuition for VQ and highlight how it differs from VQ.

2.1 Scalar Quantization

Scalar quantization refers to mapping values from a continuous space to a limited discrete set that includes a finite number of values. It is an inevitable factor in the digital signal processing field to transform an analog signal to its digital form by quantizing its continuous-valued amplitudes in analog-to-digital converters. The name *scalar* refers to the fact that the quantization is performed on a single number (scalar) that involves only one dimension ($D = 1$) [20]. Generally, scalar quantizer Q can be defined as a mapping function $Q : \mathbb{R} \longrightarrow CB$, where $\mathbb{R}$ is the continuous real values and $CB = \{CB_1, CB_2, \ldots, CB_K\} \subset \mathbb{R}$ is the quantization codebook which includes K quantization levels. Fig. 2.1 illustrates an example of scalar quantization of real values in the range $[-1, 1]$ to four quantization levels. In other words, all real values that lie inside two quantization borders (*red ticks*) will be mapped to their corresponding quantization level $CB_i \; ; \; i \in \{1, 2, 3, 4\}$.

Quantization bitrate (resolution) is defined as $B = \log_2 K$ which is the number of required bits to uniquely identify all K quantization levels. Fig. 2.2 demonstrates a cosine wave whose amplitude is scalar quantized into nine quantization levels. The bitrate for this quantization is $B = \log_2 9 \approx 3.17$. It means that to transmit any sample of the quantized

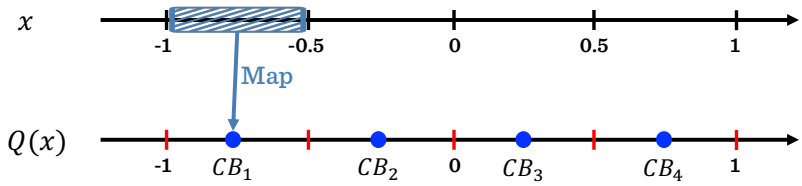

Figure 2.1. Scalar quantization of real values between -1 and 1 to four discrete quantization levels (*blue circles*). Quantization borders are shown in *red ticks*. For instance, all real values between -1 and -0.5 will be represented by the value of CB_1 after quantization. Figure adapted from [20].

amplitude, 3.17 bits are needed. A higher quantization bitrate (resolution) leads the quantized waveform to model the original signal with higher accuracy [20].

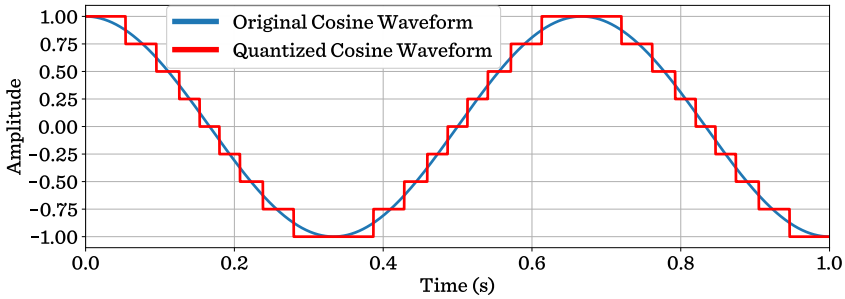

Figure 2.2. Scalar quantization of the amplitude of a cosine waveform using nine quantization levels. In the quantized waveform, all real values between -1 and 1 (on y-axis) are represented with the discrete set of quantization levels that is $[-1, -0.75, \ldots, 0.75, 1]$. Figure adapted from [1].

2.2 Definition

Vector quantization (VQ) is a data compression technique that models the probability density function of a distribution using a set of codebook vectors. VQ was originally used in signal processing applications for data compression purposes. VQ is a generalization of scalar quantization that involves quantizing a vector with a dimensionality of more than one ($D > 1$). A vector is an arranged set of real values that can represent a kind of pattern, e.g., spectral envelop of a speech signal or a patch of an image. VQ is similar to the k-means algorithm in that it clusters a large set of vectors to a finite set of groups (Voronoi cells), each represented by a codebook vector [20].

As shown in Fig. 1.1, in VQ, each Voronoi cell contains a codebook vector, which is the closest codebook vector to all data points located in that Voronoi cell. In other words, each codebook vector represents all data

points existing in its corresponding Voronoi cell. Therefore, applying VQ on this Gaussian distribution means to map all data points inside a Voronoi cell to its closest codebook vector. For an input vector $x \in \mathbb{R}^{1\times D}$ and codebook matrix $CB \in \mathbb{R}^{K\times D}$, VQ function can be formulated as

$$\hat{x} = \arg\min_{i} \|x - CB_i\|_2, \quad 0 \le i < K, \tag{2.1}$$

where $\hat{x} = CB_{i^*}$ is the vector quantized form of the input vector x, and i^* refers to the index of closest codebook vector from codebook matrix CB to the input vector x in terms of Euclidean distance shown by $\|\cdot\|_2$.

2.3 Applications

VQ can be used in various applications, such as lossy data compression, video and audio codecs, pattern recognition, clustering, and discretizing representations in neural networks. As illustrated in Fig. 1.1, VQ can compress a large number of data samples from a distribution to a limited set of codebook vectors, which entails a loss in accuracy of representing the original data distribution. This property of VQ is called lossy data compression, which is used for data storage and transmission applications [20–22]. Furthermore, VQ is used in some image codecs (GIF and JPEG-LS), video codecs (MPEG-4 visual, and H.263), and audio coding (AMR-WB+, CELP, G.729) standards.

One of the most well-known applications of VQ in the pattern recognition field is the approximate nearest neighbor (ANN) search. In ANN search, VQ is used to compress a large number of high-dimensional vectors to a limited set of codebook vectors (with the same or lower dimensionality) with the aim of reducing the computational complexity of searching [13–15]. In addition, VQ can be used in clustering tasks [23, 24]. Furthermore, apart from classical signal processing applications, VQ has also caught the attention in the machine learning world. Since discrete representation is a more suitable fit for some data modalities, many recent deep neural networks use VQ to discretize the continuous representation of their latent space to enhance their performance [3–6, 8–11, 25–28].

2.4 Evaluation Metrics

The main target of vector quantization (VQ) is to represent the input vector x with a limited bitrate. The input to the VQ function is not deterministic nor known in advance. It is a random variable defined by its probability density function (PDF), and thus the quantization error will also be a random variable. To evaluate the VQ performance, a distance measure $d(x,\hat{x})$ is required to determine the magnitude of quantization error (or

distortion) between inputs x and their quantized forms $\hat{x}$. As the inputs are random, the overall quantization error should be quantified with a statistical averaging measure that takes the PDF of inputs into account [1, 20]. Therefore, the average quantization error (AQE) can be computed as

$$\mathrm{AQE} = \underset{x \in \mathbb{X}}{E}\big[d(x,\hat{x})\big] \approx \frac{1}{N}\sum_{i=1}^{N} d(x_i,\hat{x}_i), \tag{2.2}$$

where x_i and $\hat{x}_i$ refer to the i-th input and its quantized form, respectively. $\mathbb{X}$ refers to the set of possible input vectors, E is the statistical expectation operand, and N is the total number of input vectors. Generally, a lower average quantization distortion indicates a better VQ performance.

The most prevalent distance measure used for VQ is the squared error or squared Euclidean distance (ℓ_2 norm) [20]. The Euclidean distance between the input vector x and its quantized version $\hat{x}$ is computed as

$$d(x,\hat{x}) = \|x-\hat{x}\|_2^2 = \sum_{j=1}^{D} (x_j-\hat{x}_j)^2, \tag{2.3}$$

where the index j refers to the j-th element of x and $\hat{x}$ vectors, and $\sum$ is over all samples of D-dimensional vectors. Now, by incorporating Eq. (2.3) in Eq. (2.2), we can define the mean squared error (MSE) as the performance measure for VQ such that

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N} \|x_i-\hat{x}_i\|_2^2. \tag{2.4}$$

x_i and $\hat{x}_i$ refer to the i-th input and its quantized form, respectively, and N is the total number of input vectors. In fact, MSE is equal to the average power of quantization error.

Another widely used distance metric is the absolute error (or ℓ_1 norm) which is computed as $d(x,\hat{x}) = \|x-\hat{x}\|_1 = \sum_{j=1}^{D} |x_j-\hat{x}_j|$. Mainly, we can define the general distance measure of $d_m(x,\hat{x}) = \|x-\hat{x}\|_m$ for VQ that computes the m-th power of the quantization error, such that m is a positive integer value. When $m=1$, the measure is known as the ℓ_1 norm of the quantization error, and when $m=2$, it is known as the ℓ_2 norm of the quantization error.

Now that the MSE is defined as the metric to assess the performance of VQ, the main objective of designing a vector quantizer is to find the best set of codebook vectors that minimize the MSE. Finding the best codebook matrix is an optimization problem with no closed-form analytical solution, which entails using numerical iterative solutions [1]. One of the most popular optimization algorithms for VQ is the k-means approach [29], which is discussed in the following section.

2.5 Optimization Methods

2.5.1 K-means Algorithm

Clustering is a technique that makes use of the existing structure in the data distribution to divide it into multiple groups of data samples with similar properties [30]. K-means [29] is one the most popular unsupervised clustering algorithms, which is widely used for codebook optimization of vector quantizers in the signal processing domain, which aims to cluster a set of vectors to K codebook vectors (clusters) by minimizing the average squared Euclidean distances in Eq. (2.4). Because of its close similarity to Lloyd's algorithm, k-means is also referred to as Lloyd's technique [31]. Expectation maximization (EM) is another clustering algorithm that is similar to k-means. However, EM has some differences. It is more flexible than k-means in that it allows for soft clustering, i.e., each data point has a probability of belonging to each cluster. As a result, EM can model more complex cluster shapes than k-means [32].

Fig. 2.3 illustrates the flowchart of how the k-means algorithm works. The first step is the *initialization*, in which the user selects K codebook vectors as the initial codebook to start the k-means algorithm. The selection can be made by randomly choosing from the data samples or using the efficient k-means++ method [33], which is proposed for a better initialization. In the *assign* step, the Euclidean distance (Eq. (2.3)) of all data samples to all codebook vectors is calculated. Then, each data sample is assigned to its closest codebook vector. In the *codebook update* stage, the mean of data samples assigned to each codebook vector is computed, and then each codebook vector is updated with this newly computed mean. Next, the convergence condition is checked, and if it is passed, the k-means algorithm is finished. If k-means is not converged yet, the algorithm goes to the second step and will continue until the convergence condition is satisfied.

At each iteration, the k-means ensures the rendering of an improved codebook over the previous iteration, and this improvement continues until the k-means reaches a local minimum, where there will be no room for more codebook enhancements. In other words, at each iteration, the newly updated codebook (at *codebook update* step) would result in a more accurate codebook that reduces the mean squared error of clustering. K-means algorithm converges when the newly updated codebook at the current iteration is equal to the previous stage codebook. At this time, the convergence condition is passed, and we obtain the final optimized codebook for the given data distribution [1].

Although the k-means algorithm is simple to implement on most computing devices, it has its own downsides. Firstly, the number of clusters

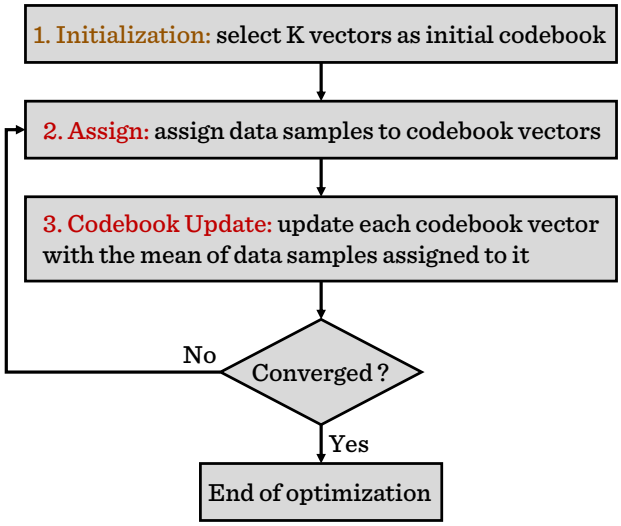

Figure 2.3. Flowchart to optimize the codebook of vector quantization with k-means algorithm. Figure adapted from [34].

(K) must be specified before running the k-means algorithm. This is a big challenge for unknown and high-dimensional data, as it is not known how many well-defined clusters the data contains. The elbow method [35] is one the most common techniques to find the actual number of clusters in the data. It offers running the k-means algorithm for several rounds with different numbers of clusters such that $K \in \{1,2,3,4,5,\ldots\}$. Then, for each value of K, the sum of the mean squared errors (MSE) between all data samples and their closest codebook vector is computed. Finally, by plotting the value of MSE for each K, a plot similar to Fig. 2.4 will be obtained.

We observe that the increase in K continuously decreases the MSE value because with more clusters (or codebook vectors), the data samples will be closer to their codebook vectors. The curve in Fig. 2.4 looks like an *elbow*, and there is a point in the figure where adding more clusters will not yield a considerable amount of decrease in MSE. In the elbow technique, this point is selected as the exact number of clusters existing in the data, which is $K = 3$ in the case of Fig. 2.4.

Secondly, k-means is really sensitive to initialization. By random initialization, k-means might end up with different results for different executions, and it is prone to reaching suboptimal results. The k-means++ [33] technique is proposed for an improved initialization. Suppose x shows a data sample from the set of all possible data samples $\mathbb{X}$, and $L(x)$ represents the smallest possible distance from data samples to their closest codebook vectors. Then, Tab. 2.1 explains how k-means++ works.

Lastly, k-means may not properly cluster the sparse and irregular data distributions. It usually works fine for distributions with equally-sized

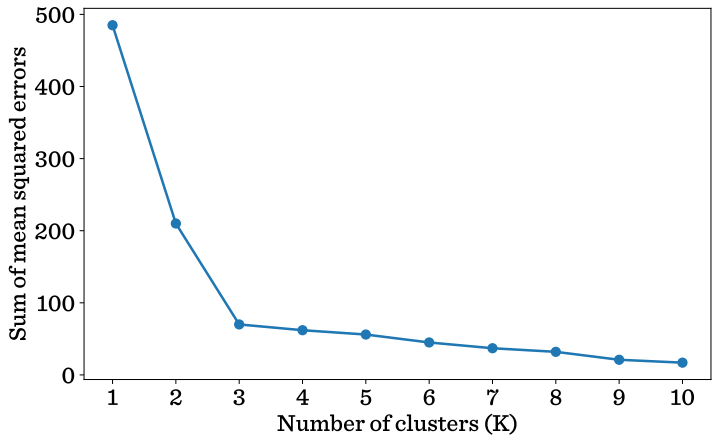

Figure 2.4. Choosing the right number of clusters (K) for k-means using the elbow method. According to the plot that looks like an elbow, the right number of clusters for the data is chosen $K = 3$ since the error is not considerably reduced when clustering the data with more than $K = 3$ clusters. Figure adapted from [34].

Table 2.1. K-means++ algorithm [33] for better initialization of codebook vectors.

1. Define the first codebook vector by uniform random sampling from $\mathbb{X}$.
2. Define a new codebook vector by sampling x from $\mathbb{X}$ with probability of $\frac{L(x)^2}{\sum_{x \in \mathbb{X}} L(x)^2}$.
3. Repeat step 2 until obtaining K required codebook vectors.
4. Use these K selected codebook vectors for k-means initialization.

clusters with hypersphere shapes (a circle in N-dim space). There is no solution for this k-means limitation, and we can use other clustering algorithms suitable for more complex distributions, such as DBSCAN [36]. Fig. 2.5 illustrates clustering the *moons* distribution with k-means and DBSCAN methods, in which k-means cannot cluster the data properly into two clear individual clusters shown in red and blue.

2.5.2 Evolutionary Algorithms

Evolutionary algorithms (EAs) are heuristic methods to solve computationally complex optimization problems, such that it takes a long time to exhaustively search for the optimal solution. Also, they are used in problems where the output has a highly non-linear and non-differentiable relation with optimization parameters. By getting intuition from biological evolutionary mechanisms, EAs try to improve a candidate solution over different generations (or iterations) of an evolutionary process. In general, EAs define a population consisting of a set of potential solutions and then mutate these solutions based on specific strategies to obtain a better solution. Afterward, they compare these mutated solutions with the already

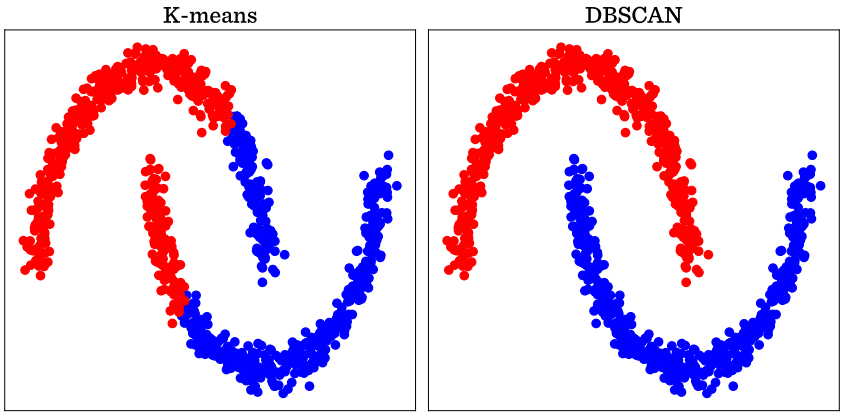

Figure 2.5. Clustering the *moons* dataset with k-means and DBSCAN algorithms. Two dataset clusters are shown in red and blue. Figure adapted from [37].

existing ones under a fitness function (or evaluation metric) and keep the top best solution for the next generation. This process will be continued for a predetermined number of generations when the best solution in the last generation is selected as the final optimized solution.

In the literature, several EA algorithms have been proposed for the optimization of the vector quantization (VQ) codebook, such as genetic algorithm [38], particle swarm optimization [39], firefly algorithm [40], and differential evolution [41]. By getting intuition from the method proposed in [41], as a typical example, here we explain how to optimize the VQ codebook using the differential evolution algorithm [42]. Suppose we intend to quantize N input vectors with a codebook comprised of K codebook vectors. The optimization steps are:

1. **Configuration**: Select the mean squared error (MSE) in Eq. (2.4) as the fitness function, the number of generations N_g to iterate, and the population size N_p (number of solutions (codebooks) in each population). Put $G = 1$, which shows the first generation.

2. **Initialization**: Initialize N_p codebooks in the population by randomly selecting from N input vectors.

3. **Mutation**: For each target solution (codebook vector) X_i^G in the population, create the mutant solution as:

$$V_i^G = X_{r_1}^G + F \times \left(X_{r_2}^G - X_{r_3}^G\right); V_i^G \in \mathbb{R}^{K \times D}, \tag{2.5}$$

where i indexes the solution in the population, G indexes the generation, $F \in (0,1)$ is the mutation factor, and $r_1, r_2, r_3 \in \{1, 2, \ldots, N_p\}$ are random target solutions indices such that $r_1 \neq r_2 \neq r_3 \neq i$. This operation applies vector differences between various solutions of the population with the aim of perturbing them in different directions.

4. **Crossover**: For each target solution X_i^G and its mutant version V_i^G, generate a trial solution by crossover operation as:

$$T_{ijk}^G = \begin{cases} V_{ijk}^G, & \text{if uniform random number} \leq Cr. \\ X_{ijk}^G, & \text{otherwise.} \end{cases} \tag{2.6}$$

   where $i \in \{1,2,\ldots,N_p\}$, $j \in \{1,2,\ldots,K\}$, $k \in \{1,2,\ldots,D\}$ such that D refers to the data dimension and $Cr \in [0,1]$ is the crossover rate. This operation generates a new solution by swapping elements in the target and mutant solutions. The swapping operation adds more diversity to mutant solutions (V_i^G), and it is governed by the crossover rate.

5. **Selection**: A tournament is held between target solution X_i^G and its trial counterpart T_i^G, and the solution which has a better fitness function (lower MSE value) will enter the next generation as:

$$X_i^{G+1} = \begin{cases} T_i^G, & \text{if} \quad \mathrm{MSE}\left(\mathrm{T_i^G}\right) < \mathrm{MSE}\left(\mathrm{X_i^G}\right). \\ X_i^G, & \text{otherwise.} \end{cases} \tag{2.7}$$

6. **Condition Checking**: If $G = N_g$, stop the optimization process and select the best solution from the current population, which gives the lowest MSE value. Otherwise, set $G := G+1$ and go to the *mutation* step.

Since searching for the optimum codebook is completely random in evolutionary algorithms, they are not commonly used for VQ codebook optimization. They are mainly used for providing a more suitable codebook only for initialization purposes [41].

2.5.3 Gradient-based Optimizers

Gradient descent is an iterative optimization algorithm that is commonly used in classification, regression, and backpropagation in neural networks. It can be used in cases where the loss function can be defined as a differentiable function of optimization parameters. It works by computing the first-order derivatives of the loss function w.r.t the optimization parameters, and these derivatives are used to find the direction to reach the minimum of the loss function.

Gradient descent calculates the derivatives over all data samples, making it challenging to be used for huge datasets or those with high dimensionality because it might take a very long time to compute gradients over all data samples, leading to memory run-out. Hence, the solution is to use other variants of gradient descent such as stochastic gradient descent (SGD) [43], RMSProp [44], AdaGrad [45], and Adam [46], in which the

gradient is calculated over a batch of data samples and parameters are updated more frequently based on the batch gradients.

We can use these gradient-based optimizers to optimize the codebook of a VQ function. Suppose a dataset that is divided into batches with N_b samples. We want to find the optimal codebook CB that minimizes the MSE loss over these N_b samples using the SGD optimizer. Thus, we can update the codebook for each batch of data as

$$CB_{i^*} := CB_{i^*} - \alpha \frac{1}{N_b} \sum_{N_b} \frac{\partial \text{MSE}}{\partial CB_{i^*}} = CB_{i^*} - \alpha \frac{1}{N_b} \sum_{j=1}^{N_b} \frac{\partial \left\| x_j - CB_{i^*} \right\|_2^2}{\partial CB_{i^*}}, \tag{2.8}$$

where x_j is the j-th data vector, CB_{i^*} is the closest codebook vector to the x_j, and α is the learning rate of the optimizer which determines the magnitude of the step that the optimizer takes towards minimum. If a small value for α is selected, the optimizer will take small steps towards the minimum, and it might cause a slow convergence. On the other hand, a big value for α (or a big step size) might result in an overshoot from the minimum point. Note that the SGD optimizer might get stuck in a local minimum according to the shape of the loss function and the learning rate selection. However, if the loss function is convex, SGD would find the global minimum, since there is only one minimum in the loss curve to land. Fig. 2.6 displays how SGD finds the global minimum in a convex loss function that is dependent only on the parameter θ. The figure shows the trend of SGD steps when the learning rate value is selected properly. The magnitudes of step sizes are different because the gradient $\left(\frac{\partial \text{Loss}}{\partial \theta}\right)$ value decreases along the loss curve.

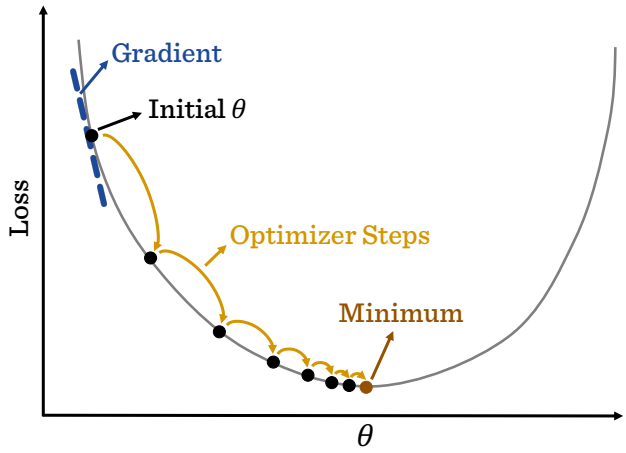

Figure 2.6. A visualization of optimization steps for stochastic gradient descent. Figure adapted from [47].

2.6 Lattice Vector Quantization

A lattice can be defined as an infinite number of points that are located next to each other in a well-ordered manner in a D dimensional space. These ordered points can be considered the codebook vectors of a vector quantizer. In fact, lattice VQ is a particular class of vector quantizers that can be expressed as the intersection of a lattice and a distribution [48], as shown in Fig. 2.7. Since the codebook vectors in lattice VQ have a regular arrangement, all Voronoi cells would have the same shape and size such that each Voronoi cell can be described as a translation of the Voronoi located in the origin of coordinates in space. According to the figure, here we see a lattice with hexagonal Voronoi cells that is projected on a 2D trapezoid distribution. To apply lattice VQ, the next required step is to discard the lattice points outside the trapezoid distribution, which is called *truncation*. Truncated codebook vectors that land outside the trapezoid distribution are shown in gray.

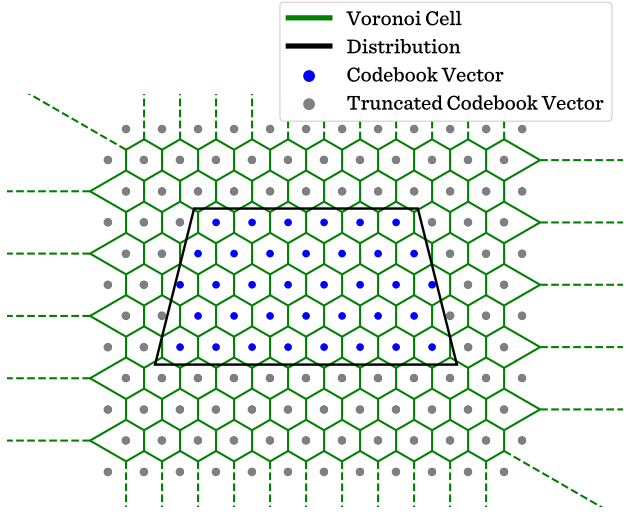

Figure 2.7. Hexagonal lattice vector quantization of a 2D trapezoid distribution. The lattice is projected on the trapezoid distribution and it is quantized with the codebook vectors that land inside of the distribution (shown in blue). Figure adapted from [20].

Choice of the lattice and truncation are two important factors that significantly impact the performance of the lattice VQ. Choice of the lattice refers to selecting the shape of the lattice Voronoi cells and the density of the lattice points, i.e., the number of lattice points in the unit of space volume. A higher density (larger codebook size) would result in a higher quantization bitrate and smaller quantization error $\|x-\hat{x}\|_2$ [20]. However, when quantizing reasonably smooth and uniform distributions with suitably high bitrates, the lattice can be chosen independent of the distri-

bution's probability density function and the truncation style [20]. Various approaches on how to select proper lattice characteristics and truncation styles can be studied in [48].

Lattice VQ has two main weaknesses. First, lattice VQ is not optimal in minimizing the average quantization error due to its regular structure of locating codebook vectors. The reason is that lattice VQ quantizes the dense and sparse regions of nonuniform distributions with equal quantization accuracy using equidistant and equal-sized Voronoi cells. However, for uniform distributions, lattice VQ can be near optimal. The second problem is the implicit assumption of having a dense lattice (or a large codebook size), which makes it suitable only for quantizations with reasonably high to moderate bitrates and applications that could tolerate small quantization distortions [20].

2.7 Variants

The best approach to quantize random vectors of a probability density function is to apply ordinary VQ without any constraints with only one codebook matrix [20]. This approach is known as *unconstrained* VQ. However, due to some restrictions in signal processing hardware, there is always a complexity barrier to use unconstrained VQ with high quantization bitrates (B) and high data dimensions (D). In more detail, a high bitrate (or a large codebook size) can exceed the memory of a processor. Moreover, the complexity of searching for the closest codebook vector among a large codebook size (K) is really high, which in turn scales exponentially with the bitrate $K = 2^B$. Current state-of-the-art methods use VQ with maximum bitrates ranging from 10 to 12 bits (1024 to 4096 codebook vectors) and maximum data dimensions ranging from 64 to 512.

There are multiple variants of VQ that change the design of the quantization process by applying various constraints. These variants are known as *constrained* VQ methods. To pass the complexity barrier, these variants use multiple codebooks for quantization, which makes it feasible to apply VQ with high bitrates and dimensions. The change of the design in constrained VQ variants will sacrifice the best performance attainable by unconstrained VQ to some extent, but it leads to efficient trade-offs between complexity and performance. In the following, some constrained variants of VQ are explained.

2.7.1 Residual Vector Quantization

Also known as multi-stage VQ, residual VQ (RVQ) [14] quantizes a vector in multiple successive stages (M) using multiple codebook matrices. Fig. 2.8 illustrates a 3-stage ($M = 3$) RVQ applied on the input vector x. In the first

stage, the input x is quantized with the first VQ block using CB^1, which is the corresponding codebook for the first stage. The quantized input equals $CB^1_{i^*}$, which refers to the closest codebook vector (i^*-th entry) from codebook CB^1 to input x. Using the second codebook CB^2, the second stage will quantize the error between input x and its quantized form $CB^1_{i^*}$, named R_1. Accordingly, the third stage will quantize the error between R_1 and its quantized form $CB^2_{i^*}$ by utilizing the third codebook CB^3. This procedure will be continued for M stages until we find M closest codebook vectors from each individual codebook matrix. Finally, the input x is quantized by addition of all M closest codebook vectors as

(2.9)

Not VQ
fun

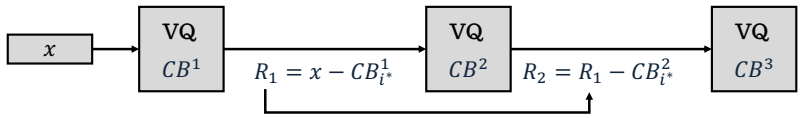

Figure 2.8. 3-stage residual VQ applied on the input vector x. CB^j is the j-th codebook matrix, and $CB^j_{i^*}$ refers to the i^*-th entry of the j-th codebook matrix, which is the closest codebook vector to the input of the VQ block.

2.7.2 Product Vector Quantization

Product VQ (PVQ) [13] quantizes a vector by splitting it into M subspaces and then applies M independent VQ operations on these subspaces. Fig. 2.9 displays a 3-stage ($M = 3$) PVQ applied on the input vector x with dimension of D. According to the figure, PVQ splits the input vector x to 3 subspaces with the dimension of $D/3$ and then applies ordinary unconstrained VQ (Eq. (2.1)) independently on each of these 3 subspaces. After finding the closest codebook vector for all three stages, the input vector is quantized by concatenating all these vectors as

$$\hat{x} = \text{Concatenate}\,[CB^1_{i^*}, CB^2_{i^*}, CB^3_{i^*}], \tag{2.10}$$

where CB^j is the j-th codebook matrix and $CB^j_{i^*}$ refers to the i^*-th entry of the j-th codebook matrix, which is the closest codebook vector to the input of VQ block.

PVQ is desirable in scenarios where the computational complexity due to the high input dimensionality is too big for other variants of VQ. The main drawback of PVQ is that it will lose some quantization efficiency since it ignores the statistical dependency between different dimensions of the input vectors.

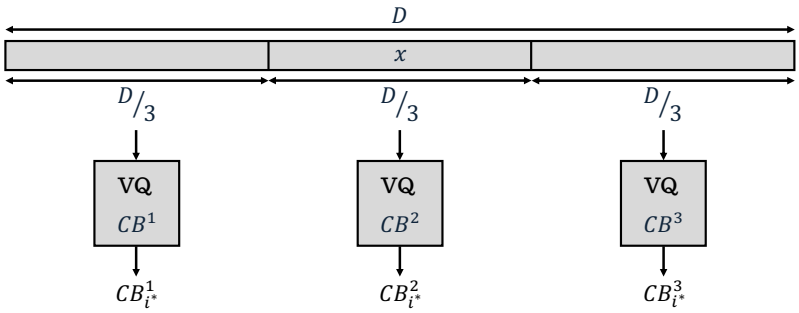

Figure 2.9. 3-stage product VQ applied on the input vector x. CB^j is the j-th codebook matrix and $CB^j_{i^*}$ refers to the i^*-th entry of the j-th codebook matrix, which is the closest codebook vector to the input of the VQ block.

2.7.3 Additive Vector Quantization

Similar to residual VQ, additive VQ (AVQ) [15] quantizes a vector as the sum of M codebook vectors coming from M different codebook matrices. AVQ aims to find a tuple of codebook indices ($I = [i_1, i_2, \ldots, i_M]$) which minimizes the quantization error $\|x - \hat{x}\|_2$ between input vector x and its quantized version $\hat{x}$, such that

$$\hat{x} = CB^{j_a}_{i_1} + CB^{j_b}_{i_2} + \ldots + CB^{j_m}_{i_M}; \quad j_a \neq j_b \neq \ldots \neq j_m; \quad j_x \in \{1, 2, \ldots, M\}.$$

To find the best tuple for the quantization, AVQ employs a beam searching approach [49], which entails defining a parameter called beam width (W). As the name suggests, beam width refers to the largeness of the search span used to find the best codebook elements. A wider beam width results in a higher computational cost for the quantization process.

Fig. 2.10 demonstrates a 3-stage ($M = 3$) AVQ applied on the input vector x. In the first step, AVQ defines W different tuples for quantization. Then, it finds W closest codebook vectors to the input x from the union of all M existing codebook matrices, i.e., $CB^1 \cup CB^2 \cup CB^3$. These W best indices will be stored in the first position (i_1) of all W tuples. These W tuples are used as the basis tuples for the remaining quantization stages. Afterward, we compute residuals for the first quantization stage (R_1) for these W basis tuples by subtracting W selected codebook vectors from the input x. Now, we have W residual vectors available. For each residual, we take the union of remaining codebooks that have not yet contributed to the quantization process as the search set. For example, in the second VQ stage of Fig. 2.10, we take the union of $CB^1 \cup CB^3$ as the search set because, for the first VQ stage, the best codebook vector is selected from the codebook CB^2. Then, for each residual, we find its W closest codebook vectors by looking up only its specific search set. Now we have W^2 tuples with length 2. Among all these tuples, we select the W best tuples that give

the minimum quantization error $\|x-\hat{x}\|_2$ and use them as the basis tuples for the remaining quantization stages. We will continue this trend for M stages, in which we have W best tuples, each filled with M indices. Among all these tuples, we finally select the best tuple which gives the minimum quantization error $\|x-\hat{x}\|_2$. For the AVQ represented in Fig. 2.10, the final quantized input is defined as

$$\hat{x} = CB^2_{i^*} + CB^3_{i^*} + CB^1_{i^*}, \tag{2.11}$$

wh y of
the put
of V

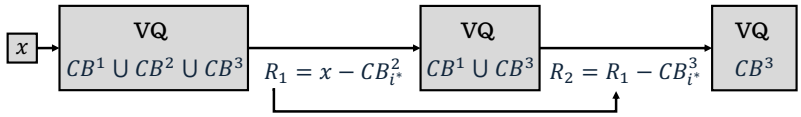

Figure 2.10. 3-stage additive VQ applied on the input vector x. CB^j is the j-th codebook matrix and $CB^j_{i^*}$ refers to the i^*-th entry of the j-th codebook matrix, which is the closest codebook vector to the input of the VQ block.

2.8 Variable Bitrate Vector Quantization

So far, all the discussions have been related to fixed bitrate vector quantizers, i.e., VQ methods which use a fixed number of bits to quantize each input vector regardless of its statistical properties. Until now, the main goal of all VQ methods has only been to minimize the average quantization error. Hence, they only take care of locating the codebook vectors such that it minimizes the quantization distortion. However, for variable bitrate VQ, we also take the probability density function of input vectors into account [20]. One principal basis of variable bitrate VQ is that by considering the probability density function of input vectors, we can assign a lower number of bits for more probable inputs and a higher number of bits to quantize the rare inputs. For example, to quantize a speech signal, we can assign fewer bits for the voiced frames as they are more likely to occur, and more bits for silence frames.

There are some applications that are desirable for variable bitrate VQ since the overall VQ bitrate would be lower than the fixed bitrate VQ. As the first example, consider an image storage application where the data should be compressed and retrieved upon request. We can assign a different number of bits for quantizing different areas of the image based on the intricacy and amount of details it contains. As another application, video transmission is really favorable for variable bitrate VQ because of the

high variations in consecutive video frames over time. In a video sequence, we can use a small number of bits to model the frames that are mainly similar to the previous frame, which models a small amount of motion. On the other hand, we need a higher bitrate for the frames modeling large amounts of motion that do not happen very often.

For a fixed bitrate VQ, the average bitrate equals the logarithm of the number of codebook vectors ($B = \log_2 K$). However, through the use of entropy coding algorithms (e.g., Huffman coding [50] and arithmetic coding [51]), the average required bitrate for VQ can be reduced from $\log_2 K$ to the *entropy* of codebook vectors [20]. Considering the codebook matrix as a set of discrete random variables which are possible outcomes of a VQ process, the entropy refers to the average amount of *uncertainty* of possible outcomes or codebook vectors. Hence, the entropy can be formulated as

$$H(y) = -\sum_{y \in CB} p(y) \log_2 p(y), \tag{2.12}$$

where y is a random variable referring to the output of a VQ function, CB is the codebook matrix, and $\sum$ is the sum of all possible outcomes. Note that taking the logarithm function in base 2 gives the entropy in terms of bits. Suppose we want to quantize a nonuniform distribution with a codebook matrix with four vectors. As the distribution's input vectors occur with different frequencies, the codebook vectors will be selected with different probabilities of $\{p_1 = 0.5, p_2 = 0.15, p_3 = 0.3, p_4 = 0.05\}$. Then, the average bitrate to quantize this distribution can be computed as

$$\begin{aligned} &\text{Fixed bitrate VQ}: && H(y) = \log_2(4) = 2 \text{ bits} \\ &\text{Variable bitrate VQ}: && H(y) = -1 \times \sum_{i=1}^{4} p_i \log_2 p_i \approx 1.65 \text{ bits}. \end{aligned} \tag{2.13}$$

Therefore, by taking advantage of the statistical properties of the codebook vectors, we can improve the quantization efficiency by reducing its average bitrate. Note that the value of entropy is always less than or equal to $\log_2 K$.

Variants of VQ can also benefit from variable bitrate with different strategies. For example, in residual VQ (RVQ), we can use a family of codebook matrices of different sizes for variable-rate quantization. As the first strategy, depending on the best codebook vector index in the first VQ stage, we can choose which set of codebooks (of different sizes) to use for the second stage. Accordingly, depending on the best codebook vector index in the second VQ stage, we can determine which set of codebooks to utilize for the third VQ stage. As a more straightforward approach to making RVQ work in variable bitrate, we can compute the quantization error of the first stage. If the quantization error is higher than a prespecified threshold, we involve the second stage for quantization. We can use multiple VQ stages until we reach the desirable error below the threshold.

3. Vector Quantization in the Machine Learning Domain

Machine learning is a branch of artificial intelligence inspired by brain studies, and it aims to replicate the learning style of humans. It explores the data and finds its underlying patterns to develop algorithms that can be generalized to unseen data. This chapter explains an overview of machine learning and how its training procedure works. It will also discuss neural networks as widely used machine learning models and their variants which use vector quantization in their architectures. Since vector quantization renders only derivatives of zero or infinity, it leads to the *gradient collapse* problem when optimizing neural network architectures based on vector quantization by gradient-based optimizers. This problem and its most well-recognized solutions are elaborated later in this chapter.

3.1 Machine Learning

Machine learning (ML) is one of the most effective and powerful tools in a wide range of applications. Recently, we have witnessed an explosion of data generation. This massive pile of data will be useful if we analyze it to extract new knowledge. ML techniques can automatically discover intricate underlying patterns in the data that are difficult (or even impossible) to find by human investigation. This extracted knowledge and the patterns from the data can then be used for accurate future predictions and complex decision-making. In other words, we can use computing machines to learn the rules that control a phenomenon by experiencing the data behavior collected from that phenomenon. This is the reason why it is called machine learning [52, 53].

ML can be used for a variety of tasks, such as:

Classification: This refers to the task of assigning unseen data to a set of known categories. The output (prediction) of the ML classifier is the number of the predicted category. For example, scene classification can consist of two categories, *indoor* and *outdoor*, with category labels $\{0, 1\}$ respectively. Several examples of ML classification applications include

face recognition [54], speaker recognition [55], spam detection [56], fraud detection [57], and pathological speech detection [58].

Regression: This is the task of predicting a real value as the output of the ML algorithm for an unseen data sample. Its main difference with classification is the type of output, which is a real number that can take an infinite number of values. For example, the price of a house can be predicted based on input features such as number of bedrooms, area, age, etc. [59]. Stock price prediction [60] and weather forecasting [61] are two applications of regression models in ML.

Clustering: This is the task of partitioning a large number of data samples into several homogeneous subgroups according to their specific features. An example is clustering the customers of a market into several subgroups based on their interests and demands. ML-based clustering has applications in text mining [62], genomic data analysis [63], and recommender systems [64].

Dimensionality Reduction: The main goal of this task is to reduce the dimensionality of data samples (or input features) by transforming them from an initial representation to a lower-dimensional representation while preserving the main characteristics of the initial representation. Dimensionality reduction reduces the number of irrelevant features, which improves model generalization and reduces the computational complexity, which results in more efficient and faster training. One of the most important applications of dimensionality reduction is feature extraction, whose target is to extract more complex and useful features (with a lower dimension) out of input features for an ML task [65].

For a better understanding, here we define some essential terminologies that are used in the rest of the thesis.

Dataset: The set of data examples (or their relevant features) used to train the ML algorithm. The ML algorithm would explore the data examples in the dataset and, by experience, learn how to get fitted to this dataset for more accurate future predictions. In the case of *supervised* algorithms [53], each data example is a pair including the feature and its corresponding label as (x^i, y^i). Here, for the i-th data example, x and y refer to the feature and label of that example, respectively.

Model: The function used for mapping features (x^i) to their corresponding predicted output (y^i_{pred}). By exploring the dataset examples, this function is going to be learned during training in order to give predictions (y^i_{pred}) as close to the real data output (y^i).

Loss: The metric used for evaluating how much the ML model prediction (y^i_{pred}) deviates from the true answer (y^i). It is the metric that the ML optimizer aims to minimize or maximize during training.

Optimizer: The optimization algorithm that the ML model uses to optimize values of trainable parameters, which minimizes the loss function. Stochastic gradient descent (SGD) [43], RMSProp [44], AdaGrad [45], and Adam [46] are among the most common ML optimizers.

Hyperparameters: The parameters that are not trained by the learning algorithm and must be determined in advance as inputs to the learning algorithm. For example, learning rate (α in Eq. (2.8)), batch size (N_b in Eq. (2.8)), number of epochs N_{epoch} (number of times the optimizer explores the whole dataset), and number of layers and neurons in each layer of a neural network are among the most important hyperparameters that have significant impact on the ML model's performance.

3.1.1 Training Procedure

After selecting the model, loss function, optimizer, and necessary hyperparameters, we need to partition the dataset into different equally-sized data batches to start training the parameters of an optimization problem with ML. By selecting the batch size as N_b data samples, the dataset with N samples would consist of $N_{batch} = N/N_b$ data batches. Each data batch is considered one iteration of optimization, as the trainable parameters are updated using the derivatives of that batch by SGD as in Eq. (2.8). An *epoch* is defined as one iteration in which the ML optimizer goes through the whole dataset samples (or all N_{batch} training batches). For N_{epoch} number of epochs, the training comprises of $N_{epoch} \times N_{batch}$ number of training updates (iterations).

Each training update step has three key components:

1. **Forward Pass**: Forward pass gets the current data batch as input and computes the loss value as output of the ML model, which defines the mathematical relation between the trainable parameters and the loss function. The forward pass refers to the computation of the loss value as a function of the trainable parameters a and b, and it is shown with blue arrows in Fig. 3.1.

2. **Backpropagation**: Since in ML optimization tasks the usual choice is a gradient-based optimizer (section 2.5.3), in backpropagation the derivative of the loss function with respect to the trainable parameters needs to be calculated. Backpropagation means to calculate the gradient of the loss with respect to a and b, and it is shown with red arrows in Fig. 3.1. The gradients of the loss with respect to trainable

parameters of a and b can be calculated with the *chain rule* as

$$\begin{cases} \dfrac{\partial Loss}{\partial a} = \dfrac{\partial Loss}{\partial c} \times \dfrac{\partial c}{\partial a} = d \times b \\ \dfrac{\partial Loss}{\partial b} = \dfrac{1}{2}\left[\left(\dfrac{\partial Loss}{\partial c} \times \dfrac{\partial c}{\partial b}\right) + \left(\dfrac{\partial Loss}{\partial d} \times \dfrac{\partial d}{\partial b}\right)\right] \\ \qquad\quad = \dfrac{1}{2}\left[(d \times a) + (c \times 1)\right] = \dfrac{1}{2}\left[(d \times a) + c\right] \end{cases} \tag{3.1}$$

3. **Parameter Update**: In this stage, the trainable parameters values are updated based on the SGD update formula (Eq. (2.8)) as

$$\begin{cases} a := a - \alpha \dfrac{\partial Loss}{\partial a} = a - \alpha\,(d \times b) \\ b := b - \alpha \dfrac{\partial Loss}{\partial b} = b - \alpha \dfrac{1}{2}\left[(d \times a) + c\right] \end{cases} \tag{3.2}$$

where α is the learning rate of the SGD optimizer.

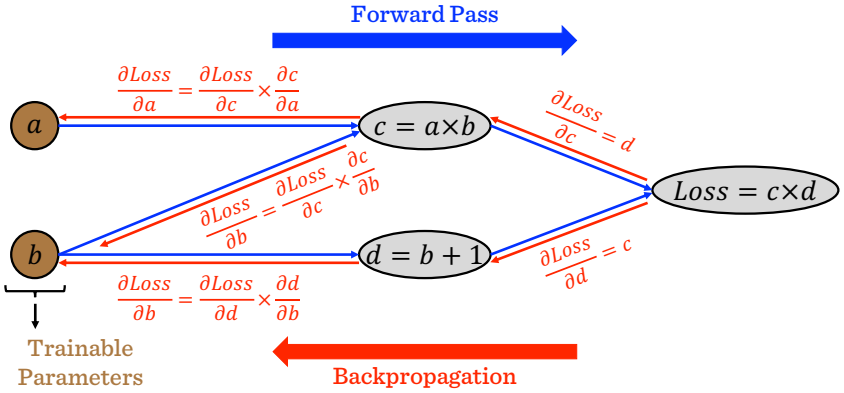

Figure 3.1. Forward pass and backpropagation. Forward pass computes the loss value based on the connections between nodes of a computational graph. Backpropagation computes the derivative of loss with respect to the trainable parameters. Figure adapted from [66].

These three key components are performed for each training batch, and the optimization is iteratively continued until it reaches the total number of training updates which equals $N_{epoch} \times N_{batch}$. After the last training batch update step, the trainable parameters are saved as the final optimized values.

3.1.2 Neural Networks

A neural network (NN) is a computational model inspired by how the human brain's neural system works. The brain's nervous system comprises billions of *neurons* that are connected, and the human learning process happens when these *neurons* pass information to each other. In a similar way, a *neuron*, also called a node, is the fundamental building block of an NN. As shown in Fig. 3.2, a node gets inputs (x_i) from other nodes and does some calculations to compute the output ($\hat{y}$). Each input has a weight (w_i) which specifies its influence on the output. The output is computed such that an activation function ($f(.)$) is applied on a weighted linear combination of inputs added by a bias value (b). The weights of the nodes and bias are trainable parameters that are trained according to the optimi

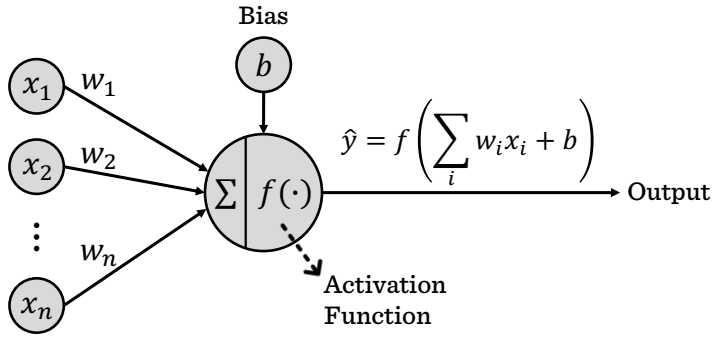

Figure 3.2. Computations in a node of a neural network. Figure adapted from [67].

Fig. 3.3 demonstrates a typical NN design. An NN usually consists of multiple computational layers, each made up of nodes interconnected with each other in a forward pass, and it computes the output as a function of inputs using these connections and their corresponding weights. The first layer of an NN is called the input layer, in which no computation is done, and it only passes information about the input features to the subsequent layers. Hidden layers take care of intermediate computations; they get information from previous layers and pass it forward to the next layers. The output layer is the last computation layer in an NN, which gets the processed information from previous layers and computes the final predicted value for the output.

Similar to Fig. 3.1, an NN (shown in Fig. 3.3) can be considered a computational graph of interconnected nodes. Therefore, an NN can be trained in a similar training procedure as in Sec. 3.1.1. In this case, forward pass refers to getting a batch of input features in the input layer and computing the output nodes' values according to the weights and connections of the NN. Then, backpropagation refers to calculating the gradients of the loss

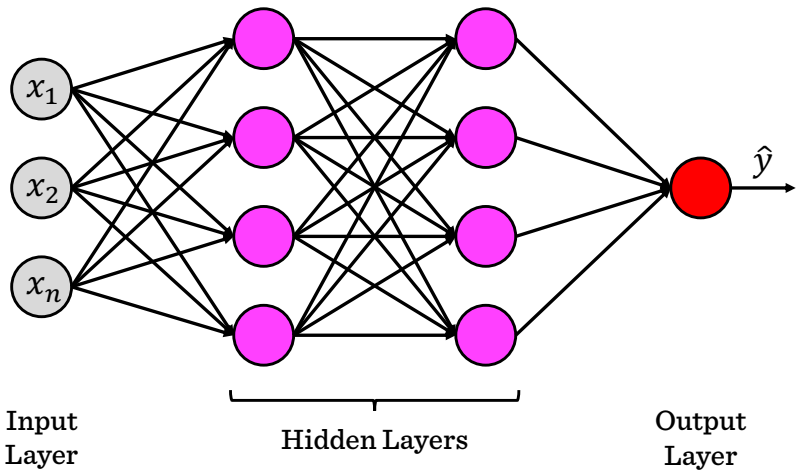

Figure 3.3. Architecture of a typical neural network. Figure adapted from [68].

value with respect to the trainable parameters (weights and biases) of the NN. These trainable parameters can then be updated based on the SGD update formula in Eq. (3.2). These three training steps will be iteratively performed and continued until the training reaches the required N_{epcoh} epochs. During training, the trainable parameters of each node are adjusted to minimize the loss, and they learn to give the best predictions for the ML task.

The architecture of an NN model can be chosen to match the difficulty of the task and available computational resources. Such choices in architecture include the number of hidden layers, the number of nodes in different layers, the way nodes are connected to each other, and the choice of activation function. Note that the activation function is usually a nonlinear function. As a result, it brings nonlinearity to the computations, enabling the output to be a complex nonlinear function of the input features. The activation function can be chosen based on the layer and the type of ML task, e.g., the *softmax* function is chosen for multi-class classification tasks. An NN with more than one hidden layer is generally considered a deep neural network (DNN). Depending on the ML task, there are various types of DNNs with different architectures such as multi-layer perceptrons (MLPs) [69], convolutional neural networks (CNNs) [70], recurrent neural networks (RNNs) [71], long short-term memory networks (LSTMs) [72], and generative adversarial networks (GANs) [73].

3.1.3 Neural Network Architectures Based on VQ

There are various DNNs that use vector quantization (VQ) in their architectures as a tool for discretizing a continuous representation [11, 74–76]. Vector quantized variational autoencoder (VQ-VAE) [11] is one of the most well-recognized architectures that employs VQ in its latent space. Fig. 3.4 illustrates the VQ-VAE architecture consisting of an encoder, a vector quantizer, and a decoder. The encoder takes the input data (x) and maps it to a lower-dimensional latent space that can represent abstract and complex features of the input. VQ is used for discretizing the latent space representation ($z_e \rightarrow z_q$). The decoder acts as a generative network that takes the discrete abstract representation of input data (z_q) and aims to reconstruct the input data (x_r) with the minimum possible error.

The reason for discretizing the latent space with VQ is that discrete representation (z_q) is a more natural fit for different data modalities such as speech, language, and image [11]. Language is naturally a discrete representation of characters, and speech can be depicted with a series of symbols. Images can also be characterized by language [77].

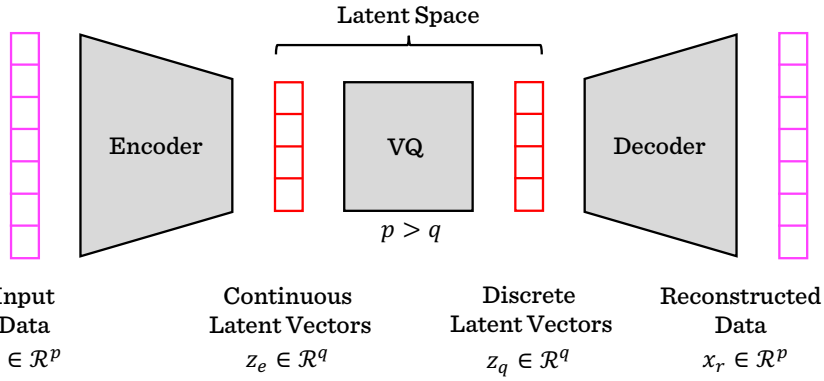

Figure 3.4. Vector quantized variational autoencoder (VQ-VAE) architecture. The encoder transforms the input x to the latent space z_e that has a lower dimensionality than input space. Latent space vectors z_e are quantized to z_q using VQ. The decoder gets the quantized latent vectors z_q and reconstructs the input x_r. Figure adapted from [78].

VQ-VAE is an architecture that has been successfully applied in a wide range of applications such as voice conversion [26, 79], image generation [3, 4], speech and audio coding [5, 6], music generation [8], and text-to-speech synthesis [9, 10].

3.1.4 Gradient Collapse Problem

Despite the usefulness of VQ in different DNN-based architectures and applications mentioned in Sec. 3.1.3, there is always a challenge to optimize the VQ codebook by gradient-based ML optimizers such as SGD. The reason is that according to the VQ formula (Eq. (2.1)), the *argmin* function is not differentiable. In other words, the stair shape of the VQ function (in one dimension) in Fig. 3.5 clearly shows that the VQ function has only two possible gradients of zero and infinity. In this case, if we use the VQ-VAE shown in Fig. 3.4 as a computational graph and compute the gradients of the loss with respect to input x, we should use chain rule as

$$\frac{\partial \text{Loss}}{\partial x} = \frac{\partial \text{Loss}}{\partial x_r} \times \frac{\partial x_r}{\partial \theta_{Dec}} \times \frac{\partial \theta_{Dec}}{\partial z_q} \times \frac{\partial z_q}{\partial z_e} \times \frac{\partial z_e}{\partial \theta_{Enc}} \times \frac{\partial \theta_{Enc}}{\partial x} \tag{3.3}$$

where θ_{Enc} and θ_{Dec} refer to the trainable parameters of the encoder and decoder, respectively. As the gradient for the VQ function equals zero or infinity $\left(\frac{\partial z_q}{\partial z_e} \in \{0, \infty\}\right)$, the final computed gradient in the above formula would equal zero or infinity $\left(\frac{\partial \text{Loss}}{\partial x} \in \{0, \infty\}\right)$. Hence, the computed gradients by Eq. (3.3) cannot be used for optimization at all. This challenge is called the *gradient collapse* problem.

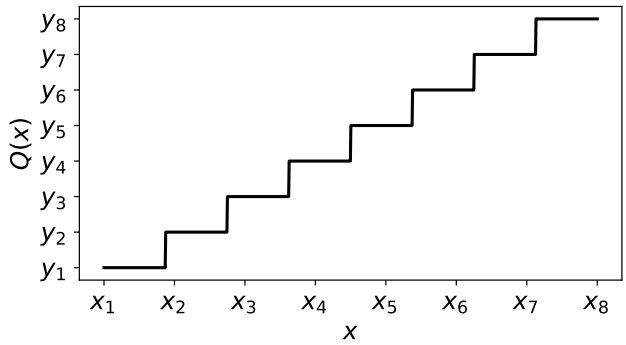

Figure 3.5. Vector quantization function in one dimension (scalar quantization). Figure adapted from [20].

3.2 Solutions to Avoid Gradient Collapse

It is impossible to optimize DNN architectures that contain VQ modules by gradient-based optimizers because VQ will collapse the gradients. As a solution, VQ can be approximated, or its behavior can be simulated with differentiable functions to obtain valid gradients for backpropagation. Several of the most practical solutions from the literature are presented below.

3.2.1 Soft Quantization

The techniques of this category refer to approximating the hard VQ function with a smoother function, which results in meaningful gradients for VQ in backpropagation. To this end, VQ can be approximated by soft clustering in which the *softmin* operand is used instead of *argmin* [80–83]. In other words, instead of mapping an input vector to only one cluster (codebook vector), soft quantizers map the input to a weighted sum of all available clusters such that the weights are dependent on the input vector's distance to each cluster. Fig. 3.6 illustrates the difference between soft VQ and hard VQ in which *argmin* and *softmin* functions are used for clustering, respectively. In soft VQ, each data sample has a weighting

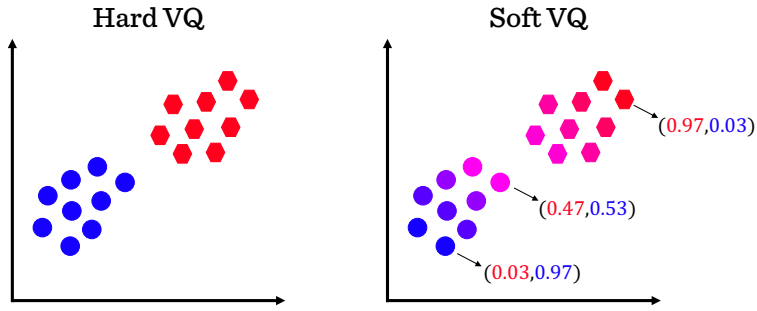

Figure 3.6. Hard VQ vs. soft VQ with two codebook vectors (or clusters). Hard VQ assigns each data point to only one cluster with a probability of 1. Whereas, in soft VQ each data point belongs to all existing clusters with a probability. Figure adapted from [84].

Suppose we calculated the Euclidean distance of the input vector x to all K available codebook vectors of $CB = [CB_1, CB_2, \ldots, CB_K]$ as $D = [d_1, d_2, \ldots, d_K]$, where d_i is the distance of x to the i-th codebook vector. In soft VQ, the input vector is quantized as a linear combination of all codebook vectors such that

$$\begin{aligned} \hat{x} &= \sum_{i=1}^{K} \pi_i \, CB_i \\ \pi_i &= \text{Softmin}(d_i) = \frac{\exp(-\tau d_i)}{\sum_{j=1}^{K} \exp(-\tau d_j)}, \end{aligned} \tag{3.4}$$

where π_i refers to the probability that x is assigned to the i-th codebook vector and $\tau > 0$ is the annealing (or temperature) factor that controls the "hardness" of assignments (or quantization). A larger value for τ would result in a harder VQ such that if τ goes to infinity, the *softmin* will be equal to *argmin*, i.e., VQ is equal to assigning x to its closest codebook vector.

In soft VQ, the final quantized input is a linear combination of all K codebook vectors, which incurs more computational cost, making it difficult to be used for real-time transmission applications. Furthermore, the efficiency of most soft quantizers is dependent on tuning the annealing factor (τ) during training. The annealing factor is increased gradually during the training to smoothly move from soft to hard quantization. A high annealing speed stops the learning process because of *gradient collapse*, and a low speed leads to large weights for trainable parameters [80].

In Publication I, we used a soft VQ technique with a fixed annealing factor ($\tau = 0.1$) to train codebooks of a residual VQ which models the spectral envelopes of speech signals in training an end-to-end speech coding framework.

3.2.2 Straight-Through Estimator

As depicted in Fig. 3.7, straight-through estimator (STE) [16] is another solution for the *gradient collapse* problem, which passes the gradients over non-differentiable parts of the computational graph in backpropagation by simply copying them. In other words, STE assumes the partial derivative for trainable parameters of VQ (i.e., codebook vectors) equals one, i.e., in Fig. 3.7, $\frac{\partial z_q}{\partial z_e} = 1$. Because of its simplicity, STE is commonly used in many NN-based optimizations which employ VQ [3, 5, 6, 11, 25, 85–87].

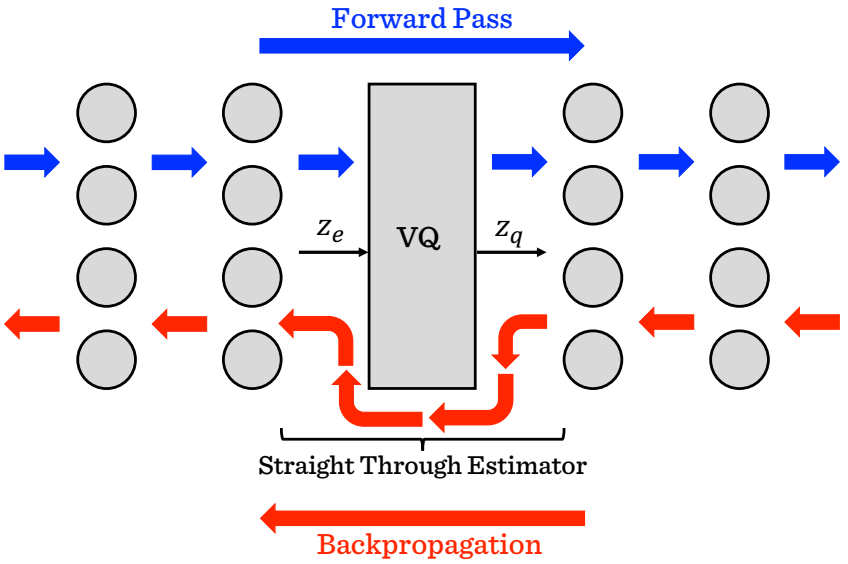

Figure 3.7. Straight-through estimator (STE) as a solution for *gradient collapse* problem. During backpropagation, STE copies the gradient intactly over the non-differentiable VQ function, i.e., STE assumes that $\partial z_q / \partial z_e = 1$. Figure adapted from [88].

In the original VQ-VAE paper [11], STE is used for training the network parameters. In the VQ-VAE architecture shown in Fig. 3.4, the global loss function is defined as

$$L = \log p(x|z_q) + \beta_1 \left\| sg\left[z_e\right] - CB_{i^*} \right\|_2^2 + \beta_2 \left\| z_e - sg\left[CB_{i^*}\right] \right\|_2^2, \tag{3.5}$$

where CB_{i^*} is the closest codebook vector to input vector x, and sg refers to the stop gradient operator that is defined as identity in forward pass and has zero partial derivatives in backpropagation. The first loss term is the reconstruction loss, which tries to reconstruct x_r as close as x, and this loss term optimizes the decoder and encoder parameters by passing the gradients intact over the VQ module. Due to copying the gradients, there are no gradients for selected codebook vectors CB_i to be updated. Hence, the second loss term is the ℓ_2 distance between latent vectors z_e and codebook vectors CB_i, through which codebook vectors of the VQ module are being optimized. The third loss term is called commitment loss, which ensures the encoder commits to the latent space and its output does not grow faster than VQ codebook vectors [11].

As observed in Eq. (3.5), using the STE technique entails adding two additional loss terms (second and third terms) to the global loss function for the training. Hence, the weighting coefficients of these loss terms (β_1 and β_2) are two added hyperparameters that need to be manually tuned in search of the best result. Furthermore, the STE method does not consider the effect of VQ and the error it brings to the latent vectors. As a result, STE causes a mismatch between the propagated gradients and the genuine behavior of the VQ function [89].

3.2.3 Exponential Moving Average

When training the VQ-VAE in Fig. 3.4, the distribution of latent vectors (z_e) changes over time after each training iteration. Since z_e vectors change over time, it is crucial to find the direction of the z_e trend to optimize the VQ codebook vectors according to this trend. On each training iteration, we have access to the previous values of z_e. Various recursive techniques can be applied to predict the next value of z_e sequence based on its previous values. The simplest way is to estimate the next value by taking the average of all previous values in the sequence. In this way, all values in the sequence have equal influence in the estimation. However, this method is not suitable for cases where the estimated parameter is more dependent on the most recent values of the sequence. For such cases, exponential moving average (EMA) [90] can give a better average estimation.

EMA is a straightforward recursive technique that is flexible and generalizes to most types of sequential data. In EMA, the main basis is to predict the next value of a sequence by a weighted average of previous values such that it assigns higher weights to more recent values and lower

weights to earlier ones. In other words, more recent sequence values would contribute more to the prediction than primary values by having higher weights in the averaging computation. It is called *exponential* because the weights magnitudes (or importance) are decreased in an exponential trend as it goes from the last value to the very first value of the sequence.

With regard to Eq. (3.5), the second loss term can be skipped, and VQ codebook vectors can be optimized by EMA [11]. Suppose $Z = [z_1, z_2, \ldots, z_N]$ is the sequence of encoder outputs (latent vectors) that are closest to the codebook vector CB_i. So, the second loss term in Eq. (3.5) is

$$\sum_{j=1} \left\| z_j - CB_i \right\|_2^2. \tag{3.6}$$

The optimal value of CB_i is the average of all values in the Z sequence as

$$\text{optimal}\, CB_i = \frac{1}{N} \sum_{j=1}^{N} z_j. \tag{3.7}$$

This formula cannot update the value of CB_i since each training update is performed on a data batch, and we do not have access to all values of the Z sequence at once. Therefore, we use EMA in this situation to have an ongoing update at each training iteration using the following formulas:

$$\begin{aligned} M^{(t)} &:= M^{(t-1)}\, \gamma + N^{(t)}\, (1-\gamma) \\ Y^{(t)} &:= Y^{(t-1)}\, \gamma + \sum_j z_j^{(t)}\, (1-\gamma) \\ CB_i^{(t)} &:= \frac{Y^{(t)}}{M^{(t)}}, \end{aligned} \tag{3.8}$$

where t refers to the time index (or training iteration), and $\gamma \in (0,1)$ is called the smoothing factor, which controls the magnitude of the weights in the moving averaging.

3.2.4 Gumble-Softmax

When applying VQ with a codebook including K vectors, the process of selecting the closest codebook vector to the input x is similar to sampling from a categorical distribution with K categories with probabilities of $[\pi_1, \pi_2, \ldots, \pi_K]$. Gumble-Softmax [91] is a technique that solves the *gradient collapse* problem by replacing the non-differentiable samples from the categorical distribution of codebook vectors with differentiable samples drawn from the Gumble-Softmax distribution. The principle of the Gumble-Softmax technique is similar to the soft quantization solution (Sec. 3.2.1) in the sense that it suggests using the *softmax* function to propagate gradients.

Suppose z is a categorical variable from codebook vectors categorical distribution ($p(z)$) with probabilities of $[\pi_1, \pi_2, \ldots, \pi_K]$. By using the Gumble-Max method, we can sample from this categorical distribution as [92, 93]

$$z_i = \arg\max_i (g_i + \log \pi_i), \tag{3.9}$$

where g_i ; $i \in \{1, 2, \ldots, K\}$ are independent and identically distributed (i.i.d) samples from $\mathrm{Gumble}(0,1)$ distribution such that $g = -\log(-\log(u))$, and u is a sample drawn from uniform distribution of $U(0,1)$. To obtain differentiable categorical samples y_i, Gumble-Softmax uses *softmax* function instead of *argmax* such that

$$y_i = \frac{\exp(\tau(\log(\pi_i) + g_i))}{\sum_{j=1}^{K} \exp(\tau(\log(\pi_i) + g_i))}, \tag{3.10}$$

where τ is the temperature (or annealing) factor which controls the "hardness" of quantization. The probability density function of Gumble-Softmax distribution is [91, Appendix B], [94]

$$p_{\pi,\tau}(y_1, y_2, \ldots, y_k) = \Gamma(k)\tau^{k-1} \left(\sum_{i=1}^{k} \frac{\pi_i}{{y_i}^{\tau}} \right)^{-k} \prod_{i=1}^{k} \left(\frac{\pi_i}{{y_i}^{\tau+1}} \right). \tag{3.11}$$

Eq. (3.10) and Eq. (3.11) demonstrate that Gumble-Softmax is a smooth differentiable function that renders valid gradients $\frac{\partial y}{\partial \pi}$ for its samples with respect to the parameter π. Therefore, by replacing the non-differentiable categorical variables (from the categorical distribution of codebook vectors) with differentiable Gumble-Softmax samples, the standard backpropagation can be used to compute the gradients. During the forward pass, we select the codebook vector $i^* = \arg\max_j y_j : j \in \{1, 2, \ldots, K\}$ for quantization, and in backpropagation, we use the standard gradients computed from Gumble-Softmax function. Note that if the temperature value goes to infinity ($\tau \to \infty$), the Gumble-Softmax distribution is annealed to the categorical distribution $p(z)$.

In Publication II, we proposed a novel solution[1] to the *gradient collapse* problem called the noise substitution in vector quantization (NSVQ) technique, in which the VQ error is simulated by adding noise to the input vector. Furthermore, we employ this NSVQ technique to optimize three variants of VQ used for image and speech applications in Publication III.

[1]While our proposed method and the above-mentioned approaches indeed "solve" the *gradient collapse* problem, they all introduce certain side effects, as they are merely tricks to approximate the behavior of VQ and cannot be as accurate as the original VQ.

4. Space-Filling Curves

Space-filling curve is a mathematical topic that was introduced in 1890 by G. Peano to answer the question: is there a curve that can pass through all existing points of a 2D space such as the unit square? Today, a space-filling curve is known as a curve that can cover an N-dimensional hypercube. This chapter will explain space-filling curves, their applications, and some of their most well-known examples.

4.1 Definition

A space-filling curve is a piecewise continuous curve that is generated with a recursive approach. If the recursion is repeated infinitely, the curve will get twisted and turned until it entirely occupies a multi-dimensional space [95]. Fig. 4.1 shows the first six recursion steps (n) of a Hilbert curve [95] that fills a 2D square space. In this figure, the direction of the

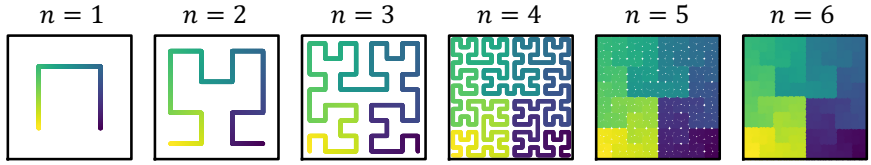

Figure 4.1. A color-coded Hilbert space-filling curve filling a 2D square space. Color-coded means that the curve starts from light color and ends in dark color. n shows the recursion step. Figure adapted from [95].

There are different types of space-filling curves, each with a specific recursion rule. The first step ($n = 1$) of Fig. 4.1 shows the basic building block of the Hilbert curve, which is determined by the location of its corner points. In the following steps ($n > 1$), this basic building block is repeated

in different locations and with appropriate rotations. This building block defines the geometric construction of various space-filling curves. In the following section, we will discuss some of the most well-known types of space-filling curves, such as Hilbert, Peano, Lebesgue, and Sierpiński.

4.2 Examples

4.2.1 Hilbert Curve

Fig. 4.2 demonstrates the first three recursion steps (n) of the Hilbert space-filling curve within a unit square space. The Hilbert curve partitions the unit square into 2^{2n} subsquares ($n = 1, 2, 3, \ldots$) at each recursion step, and each subsquare encompasses one corner point of the curve. As the numbers of the corner points show, the Hilbert curve starts from the lower

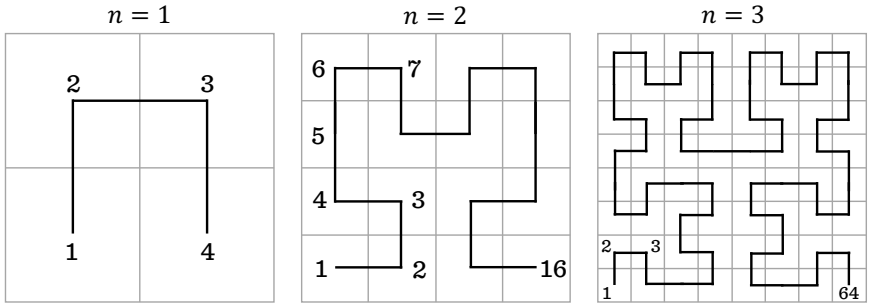

Figure 4.2. Three first recursion steps (n) of the Hilbert space-filling curve within the unit square. The numbers show the index of the corner points. Figure adapted from [95].

4.2.2 Peano Curve

Fig. 4.3 illustrates the first three recursion steps (n) of the Peano space-filling curve within a unit square. The Peano curve partitions the unit square into 3^{2n} subsquares ($n = 1, 2, 3, \ldots$) at each recursion step, and each subsquare encompasses one corner point of the curve. As the numbers of the corner points show, the Peano curve starts from the lower left side of the unit square and ends in its upper right. The geometrical shape in the first recursion step ($n = 1$) is the Peano curve's main building block, which is repeated in the following recursion steps [95].

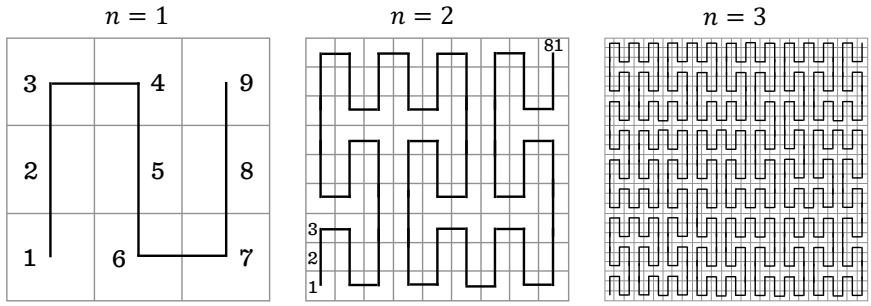

Figure 4.3. Three first recursion steps (n) of the Peano space-filling curve within the unit square. The numbers show the index of the corner points. Figure adapted from [95].

4.2.3 Lebesgue Curve

Fig. 4.4 shows the first three recursion steps (n) of the Lebesgue space-filling curve within a unit square. Similar to the Hilbert curve, the Lebesgue curve partitions the unit square into 2^{2n} subsquares ($n = 1,2,3,\ldots$) at each recursion step, and each subsquare encompasses one corner point of the curve. As the numbers of the corner points show, the Lebesgue curve starts from the lower left side of the unit square and ends in its upper right. The geometrical shape in the first recursion step ($n = 1$) is the main building block of the Lebesgue curve, which is repeated in the following

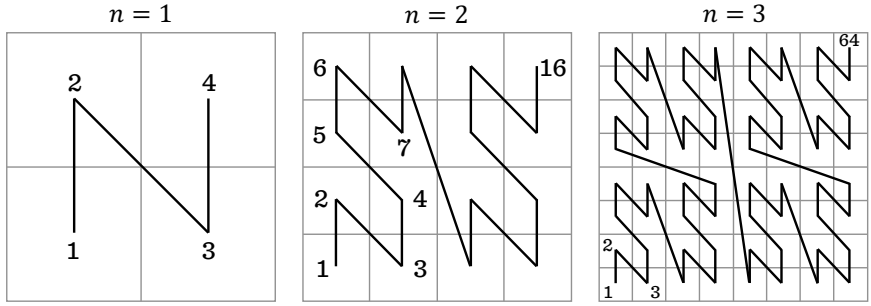

Figure 4.4. Three first recursion steps (n) of the Lebesgue space-filling curve within the unit square. The numbers show the index of the corner points. Figure adapted from [95].

4.2.4 Sierpiński Curve

Fig. 4.5 displays first three recursion steps (n) of the Sierpiński space-filling curve within a unit square. The Sierpiński curve partitions the unit square into 2^{2n} subtriangles ($n = 1, 2, 3, \ldots$) at each recursion step, and each subtriangle encompasses one corner point of the curve. The Sierpiński curve is symmetrical such that half of its curve lies in the half of the unit square sliced by its diagonal line. The geometrical shape in the first recursion step ($n = 1$) is the main building block of the Sierpiński curve, which is repeated in the following recursion steps. This building block is the closed shape of the Hilbert curve's building block, and as a result,

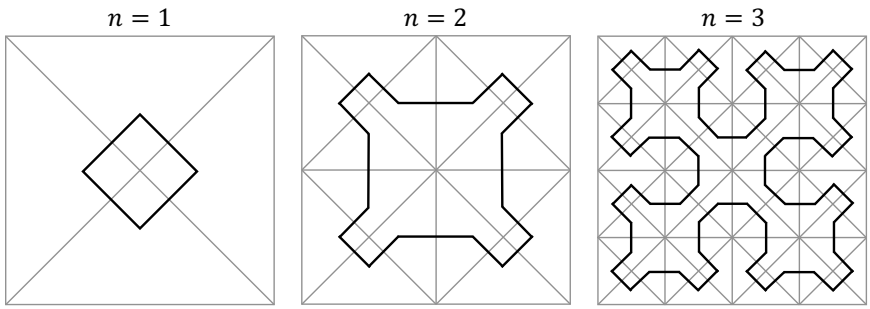

Figure 4.5. Three first recursion steps (n) of the Sierpiński space-filling curve within the unit square. Figure adapted from [95].

4.3 Applications

Space-filling curves have two desirable properties that make them suitable for various applications. They have an intrinsic sequential order of the corner points and preserve the locality features among neighboring corner points of the curve. Considering these properties, space-filling curves can be used for the parallelization of computation in multi-processor computers by partitioning the computation load into equal parts and distributing them to all available computational processors [96]. Regarding their spatial locality feature, space-filling curves can design a good data access pattern, and as a result, they can be used as a cache-efficient algorithm for data access [96].

The traveling salesman problem (TSP) [97] is the challenge of finding the shortest possible route among an existing set of cities (points) such that we only visit each city once. It is an NP-hard problem to solve, which means that for a large number of cities, it is computationally expensive and takes a prohibitively long time to exhaustively search for all possible routes to find the best out of them. Hence, the solution is to use some heuristic TSP

solvers such as nearest neighbor [98], greedy [98], or Christofides [99]. The intrinsic order in the corner points of a space-filling curve can be used to solve the TSP in an N-dimensional Euclidean space [100].

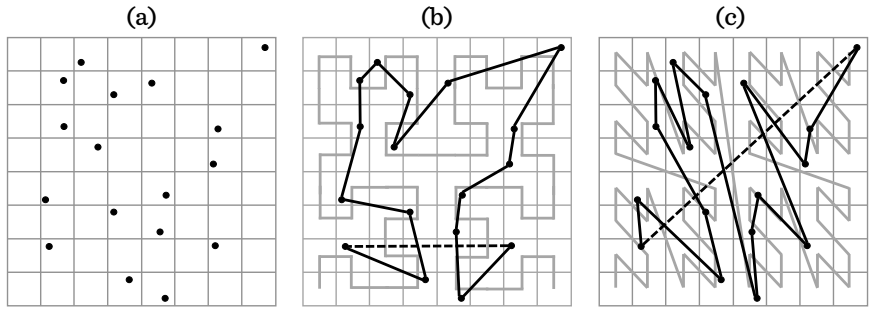

Figure 4.6. Using Hilbert and Lebesgue space-filling curves as solutions for the traveling salesman problem. (a) A set of data points which the closest route connecting them is determined by (b) Hilbert and (c) Lebesgue space-filling curves. Figure adapted from [100].

Fig. 4.6 illustrates two examples of using space-filling curves as a fast heuristic solution to TSP. Fig. 4.6(a) shows a set of points in 2D space for which we intend to find the closest route. To find the closest route using a space-filling curve, we first project the curve on the set of points, and then we connect the points in the existing order of the space-filing curve. Fig. 4.6(b-c) demonstrate the shortest routes calculated by Hilbert and Lebesgue space-filling curves, respectively.

Space-filling curves can also be used for other applications such as k-median problem [101] and nearest neighbor search [102–104]. In addition, they have applications in geographical information systems [105, 106] and multi-dimensional data storage such as geographical data bases [107]. Finally, space-filling curves are also helpful in the signal processing domain for classification [108], clustering [109], image compression [110], bandwidth reduction [111], and vector quantization [112].

With the combination of space-filling curves and vector quantization (VQ) concepts, in Publication IV we introduced the space-filling vector quantization (SFVQ) method, which resembles a space-filling curve such that its corner points are the VQ codebook vectors. In this publication, our SFVQ models the latent space of a voice conversion model with the aim of interpreting the underlying phonological structure in its latent space. Furthermore, in Publication V, we also used our SFVQ to interpret the latent spaces of two well-recognized image generative models (StyleGAN2 [17] and BigGAN [18]), and to find their interpretable directions of changes. As another application of our SFVQ technique, in Publication VI we proposed using SFVQ to anonymize the speaker attributes, which can enhance privacy in speech processing tools.

5. Summary of Publications

In this chapter, all the publications forming this dissertation are organized in chronological order, and they are attached at the end of the dissertation. All the publications are related to vector quantization (VQ) and its use in speech and image processing applications.

In our first publication (Publication I), we used VQ to model the spectral envelope of speech signals in a speech codec optimized by a machine learning framework. To resolve the *gradient collapse* problem (Sec. 3.1.4), we used the soft quantization solution (Sec. 3.2.1) to pass the gradients through VQ in the backpropagation. In the second publication (Publication II), we proposed a new solution for the *gradient collapse* problem called noise substitution in vector quantization (NSVQ), and we showed the superiority of NSVQ over two state-of-the-art solutions. In the third publication (Publication III), we studied the application of NSVQ for optimization of VQ variants (Sec. 2.7) such as additive VQ, product VQ, and residual VQ.

In the fourth publication (Publication IV), by combining the space-filling curves and VQ concepts, we introduced the space-filling vector quantization (SFVQ) technique, which is a new quantization technique that maps data points on a continuous piecewise linear curve. Here, we used our SFVQ to interpret the underlying phonetic structure in the latent space of a voice conversion model. In the fifth publication (Publication V), we used our SFVQ for interpreting the latent space of two well-known image generative models, but from a new perspective that is to discover the *interpretable directions* to change the image attributes (e.g., age for face image) in a meaningful way. In our sixth publication (Publication VI), we used the SFVQ along with a codebook resampling technique to equalize the codebook occurrences in a speaker verification task. This equalization decreases the chance of the speaker's private information disclosure.

5.1 Publication I: End-to-End Optimized Multi-Stage Vector Quantization of Spectral Envelopes for Speech and Audio Coding

Speech coding is one of the most widely used applications in the speech processing field. The main goal of speech coding is to compress the speech signal to make it suitable for processing on low-resource devices or for transmission and storage purposes while preserving its perception quality. Speech codecs consist of different modules that have complex interactions with each other. Modeling the spectral envelope (i.e., smoothed form of the magnitude spectrum) is one of the main modules of a speech codec, which has a significant impact on the compressed speech quality.

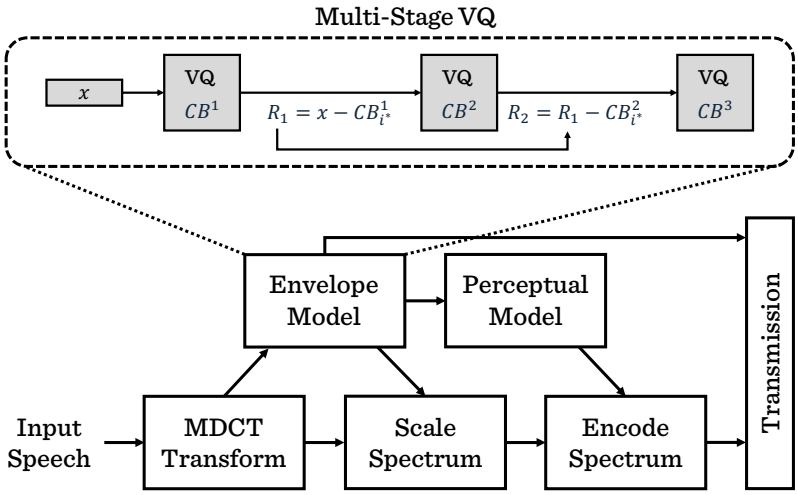

Figure 5.1. Architecture of the encoder in our speech coding model based on PyAWNeS codec.

In this publication, we improved the spectral envelope modeling of a codec called PyAWNeS. In PyAWNeS codec, the spectral envelope parameters are modeled by quantization with a uniform scalar quantizer, and then entropy coded using arithmetic coding. There are two main problems with this type of envelope modeling. First, scalar quantization does not consider the correlation among different spectral envelope parameters. Second, the envelope modeling module of PyAWNeS is designed without considering its effect on the other speech codec modules. To overcome these two problems, in this paper we proposed using multi-stage vector quantization (or residual vector quantization) to model the spectral envelope parameters.

Fig. 5.1 demonstrates the architecture of the encoder in our speech codec. Using multi-stage VQ improves the spectral envelope modeling of the PyAWNeS codec by considering the correlation between spectral envelope parameters for more efficient coding, and enabling optimization of all codec

modules in an end-to-end machine learning optimization framework. This end-to-end optimization automatically favors a more successful codec by preventing different modules from negatively impacting each other. To optimize the non-differentiable multi-stage VQ function and resolve the *gradient collapse* problem (Sec. 3.1.4), here we used the soft quantization (Sec. 3.2.1) solution. It has been shown that this way of spectral envelope modeling improves the compressed speech quality over PyAWNeS and 3GPP EVS codecs under the objective metric of perceptual evaluation of speech quality (PESQ).

5.2 Publication II: NSVQ: Noise Substitution in Vector Quantization for Machine Learning

It has recently been shown that machine learning (ML) is among the most potent solutions to optimization problems in various applications. The training procedure in ML includes the backpropagation step (Sec. 3.1.1) in which the gradients of the loss function are calculated with respect to the trainable parameters using the chain rule. If there are functions with discontinuities in the computational graph whose gradient go to zero or infinity, the final computed gradients with the chain rule would equal zero or infinity. Hence, it is impossible to optimize the trainable parameters. This problem is known as the *gradient collapse* (Sec. 3.1.4). Although vector

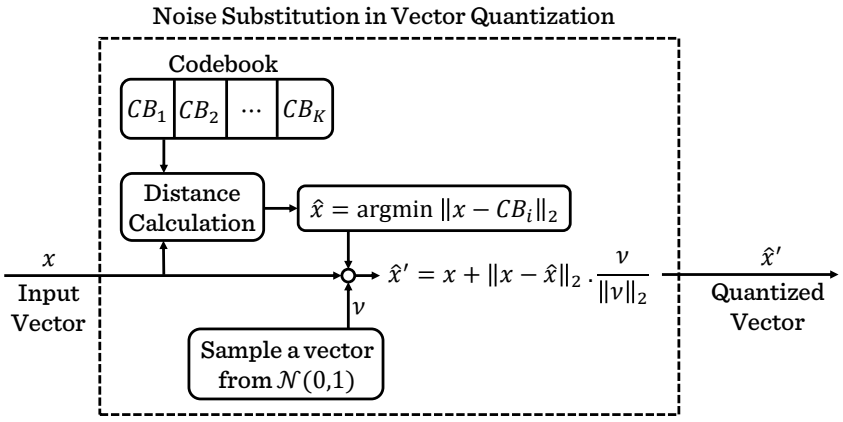

Figure 5.2. Noise substitution in vector quantization (NSVQ) module.

In this publication, we proposed noise substitution in vector quantization (NSVQ) to avoid the *gradient collapse* problem when using a VQ module in an ML computational graph. Fig. 5.2 illustrates the NSVQ technique. NSVQ simulates the quantization error by adding a noise term to the input

vector (x). This noise vector is formed by multiplying the magnitude of the original quantization error ($\|x - \hat{x}\|$) with a normalized random vector (v) sampled from the normal distribution. For each input (x), the noise term has approximately the shape of the original quantization error. Finally, by using the NSVQ method, the final quantized input ($\hat{x}'$) is a differentiable function of the input and VQ codebook vectors, and thereby the gradients can pass through the VQ function in the backpropagation step.

When training VQ codebook with ML optimization, *codebook collapse* [113] is one common challenge, in which a portion of codebook vectors are inactive during training. The reason is that the codebook vectors that are not selected for the quantization would not have any gradients, and as a result, they will not be updated and changed throughout the training. To resolve the *codebook collapse* issue, in this work we proposed a codebook replacement method in which the codebook entries that are not used for a number of training batches will be replaced. Perplexity is a measure for VQ optimization that refers to the average usage of codebook entries during training. The proposed codebook replacement improves the perplexity by making all codebook entries actively contribute to the quantization.

It has been shown that our NSVQ technique provides more accurate gradients, faster convergence, and higher perplexity than straight-through estimator (STE) (Sec. 3.2.2) and exponential moving average (EMA) (Sec. 3.2.3). Furthermore, in contrast to STE and EMA, the proposed NSVQ does not require adding additional loss terms to the global loss function. As a result, it does not need any additional hyper-parameter tuning (i.e., tuning the coefficient of additional loss terms added for optimizing the VQ codebook).

5.3 Publication III: Stochastic Optimization of Vector Quantization Methods in Application to Speech and Image Processing

Vector quantization (VQ) is a data compression technique that models the probability density function of data and is applied to any kind of data, such as speech and image. It is also effectively used in neural networks to discretize a continuous representation to a discrete categorical distribution. However, since the computational complexity of VQ increases exponentially with its bitrate, the plain form of VQ cannot be used for high bitrates. For example, the current hardware in the market cannot deal with a VQ with a bitrate of 30 as it does not have enough memory or cannot search for the closest codebook vector among 2^{30} entries. Therefore, a common modification to address this problem is to use other variants of VQ that use multiple codebooks for quantization and thus allow for quantizing with higher bitrates, such as additive VQ (AVQ), product VQ (PVQ), and residual VQ (RVQ).

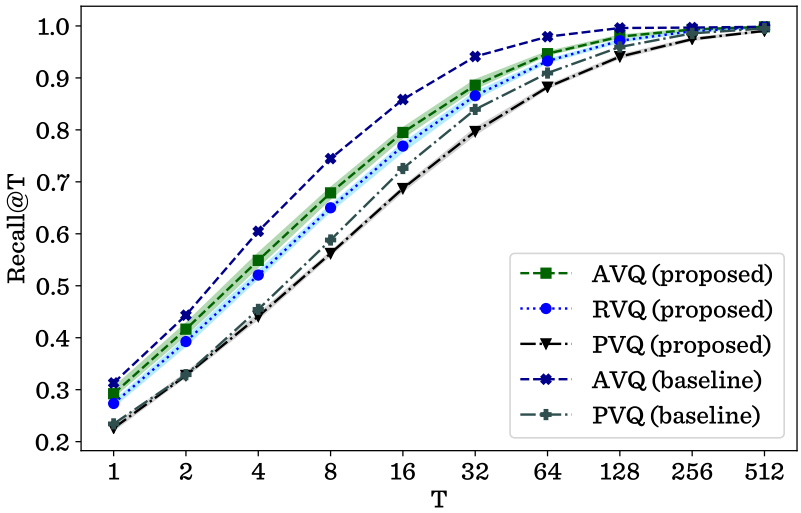

Figure 5.3. Performance comparison between traditional and NSVQ-based optimized VQ variants in the ANN search application.

VQ variants are useful tools in nearest neighbor search application. Given a query vector, nearest neighbor search is the task of searching the closest vector from a huge set of base vectors to that query vector. Approximate nearest neighbor (ANN) search is a variant of nearest neighbor, in which the huge set of base vectors is compressed to a smaller set (with a much lower number of entries) such that it eases the search operation by reducing its computational complexity and storage cost. In an ANN search, compression is usually performed using VQ variants. In the state-of-the-art methods of ANN search, PVQ and RVQ have been optimized with traditional k-means (Sec. 2.5.1), and AVQ has been optimized with the block coordinate descent optimization algorithm. Since VQ is a non-differentiable function that causes the *gradient collapse* problem, to the best of our knowledge, these VQ variants have not been optimized previously with machine learning (ML) optimization algorithms for ANN search tasks.

In this publication, we optimized these three variants of VQ with ML-based optimizers using our proposed NSVQ technique (in Publication II) over different speech and image applications. We aimed to explore the NSVQ performance to optimize the VQ variants. Fig. 5.3 shows the performance comparison between the optimization of VQ variants by traditional and ML-based methods measured with the recall metric for the ANN search task of the image database. Here, the recall@T metric indicates the probability that the actual nearest neighbor (from ground truth) exists in the first T closest vectors of the compressed base set. The metric thus measures the precision of the ANN search such that a higher recall value means a better search performance.

The experiments in Fig. 5.3 show that VQ variants optimized by our NSVQ method perform comparably to the baselines optimized by traditional techniques. Apart from ANN search, we also evaluated the performance and complexity of these three VQ variants over various bitrates for different speech and image applications. We studied the existing trade-offs between performance, complexity, and bitrate such that by using our open-source implementations, the users can readily choose the best VQ variant for their application.

5.4 Publication IV: Interpretable Latent Space Using Space-Filling Curves for Phonetic Analysis in Voice Conversion

Deep generative models are a type of neural network that map data to a representation that captures abstract and intricate features from data. This representation is called latent space. By sampling from the latent space of deep generative models, we can generate high-quality realistic data such as speech and image. One main challenge here is that the latent space acts as a black box, and thus, it is not interpretable. In other words, it is not clear what information each latent vector represents.

In this publication, we introduce space-filling vector quantization (SFVQ) as an unsupervised technique to help interpret the latent space of a voice conversion task based on VQ-VAE architecture. SFVQ is a combination of space-filling curves and vector quantization (VQ) concepts such that the codebook vectors of SFVQ play the role of corner points in a space-filling curve. Fig. 5.4(a) demonstrates a VQ and SFVQ applied on a 2D Gaussian distribution using 64 codebook vectors. In this voice conversion model, VQ works as an information bottleneck, which forces the latent space to capture only the phonetic content from the input speech signal. Hence, in this work, we applied SFVQ on the latent space to interpret the underlying phonetic structure of the latent space.

Regarding the intrinsic order in a typical space-filling curve, it is expected that index-wise adjacent codebook vectors in an SFVQ refer to similar phonetic information. Therefore, we use this SFVQ property to obtain a structured discrete representation of phonetic information out of the latent space. After training and obtaining the learned SFVQ codebook vectors, we used the phone-wise labeled TIMIT speech dataset [114] to explore the structure in the SFVQ's codebook vectors. The labeled speech phones were fed as inputs to the voice conversion model, and we extracted their corresponding SFVQ codebook index. Fig. 5.4(b) shows the histogram of SFVQ's codebook indices for different phonetic groups. We observe that different phonetic groups such as *silence*, *plosives*, *fricatives*, etc., are clearly distinct. Specifically, SFVQ's codebook indices provide a mapping between the latent space and phonetic information.

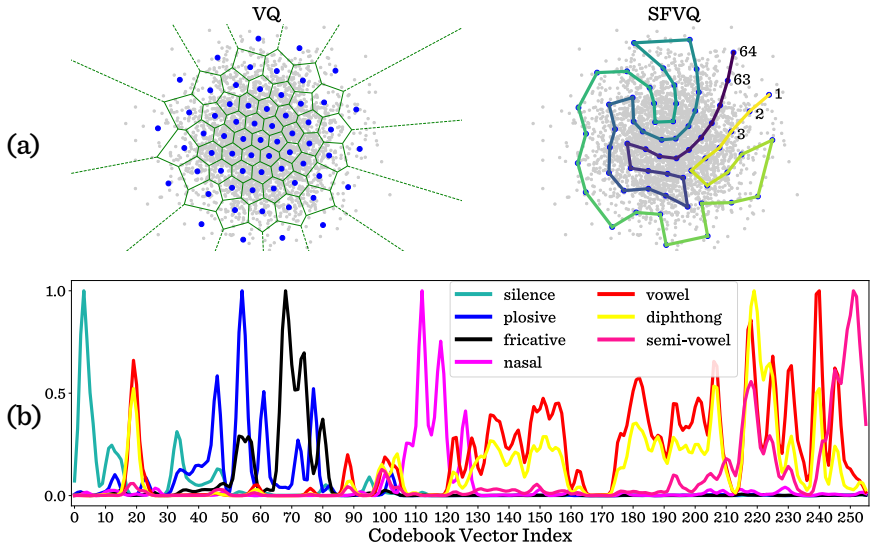

Figure 5.4. (a) Codebook vectors of a 6 bit vector quantization and space-filling vector quantization on a 2D Gaussian distribution. The order in the codebook vectors of SFVQ is color coded from light to dark. (b) Histogram of SFVQ's codebook vector indices for various phonetic groups.

5.5 Publication V: Unsupervised Panoptic Interpretation of Latent Spaces in GANs Using Space-Filling Vector Quantization

Generative adversarial networks (GANs) are among the popular generative models that map a latent space to an image space (for image generation tasks). The trained GAN model can generate high-quality realistic images by sampling from its latent space. However, the latent space is not interpretable. In other words, there is no obvious connection between vectors in the latent space and the characteristics of the output image. Furthermore, it is unclear what interpretable directions are to change these characteristics. For example, in a face synthesis model, the direction (in the latent space) to change attributes such as age or rotation is not known.

In this publication, we used our proposed space-filling vector quantization (SFVQ) technique (Publication IV) to interpret the latent spaces of two well-known pretrained, generative image models, StyleGAN2 and BigGAN, over various image datasets, such as FFHQ, AFHQ, LSUN Cars, CIFAR10, and ImageNet. Hence, we applied the SFVQ on the latent spaces, and generated and plotted the images corresponding to the learned SFVQ codebook vectors (or corner points). We aimed to interpret the latent space from two different aspects: 1) *Universal Interpretation* and 2) *Interpretable Directions*.

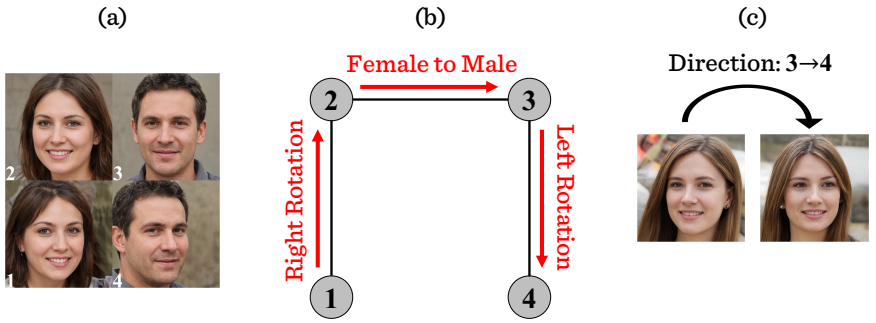

Figure 5.5. Interpretable directions of the latent space for the pretrained StyleGAN2 on FFHQ dataset discovered by SFVQ. (a) Generated images from codebook vectors of a 2 bit SFVQ and (b) their corresponding directions. (c) Applying the direction between codebook indices 3 to 4 on a random latent vector.

By *Universal Interpretation* we aimed to explore the generative features represented in the latent space. Since SFVQ has an inherent order in its learned codebook vectors, after visualizing the generated images from SFVQ's codebook vectors (Fig. 5.5(a)) we can infer, for example, which part of the latent space would generate male or female faces. By *Interpretable Directions* we aimed to discover the directions in the latent space to change attributes of an image in a meaningful manner, such as the directions to change the rotation or gender attributes. After visualizing the SFVQ's learned codebook vectors in Fig. 5.5(a), we can deduce that each SFVQ's line can refer to a meaningful direction shown in Fig. 5.5(b). Fig. 5.5(c) shows that the line connecting the codebook index 3 to 4 works as the rotation direction in the sense that it can change the rotation of a random latent vector.

5.6 Publication VI: Privacy PORCUPINE: Anonymization of Speaker Attributes Using Occurrence Normalization for Space-Filling Vector Quantization

Speech is the main medium for human interactions which, apart from its linguistic content, also encompasses private side information such as the speaker's identity, gender, age, state of health, and emotion. Hence, speech processing technologies that process, transmit, and store speech data can disclose such private side information, exposing speakers to threats such as stalking, price gouging, and identity theft. Therefore, it is necessary to use privacy-preserving speech processing tools to remove private information from speech signals that are not required for the downstream task.

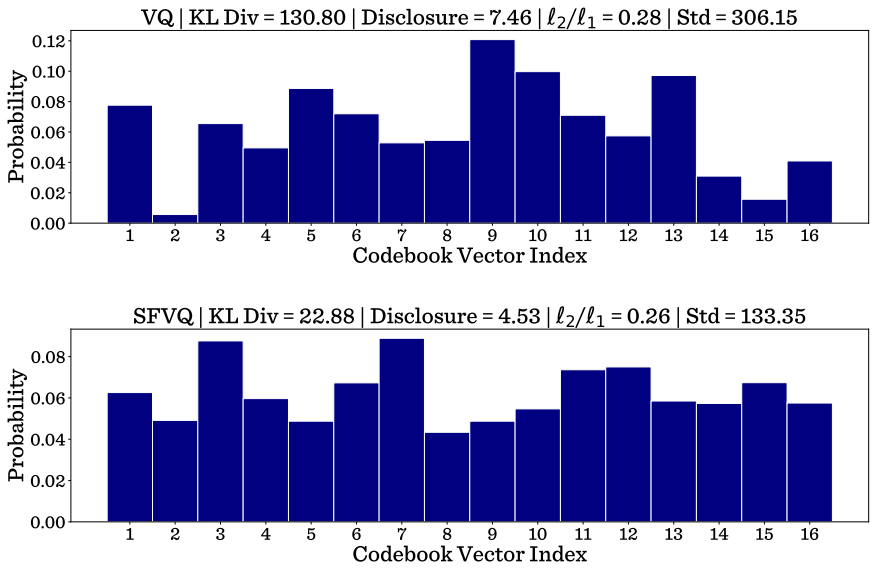

Figure 5.6. Histogram of codebook indices frequencies for 4 bit *(top)* vector quantization and *(bottom)* resampled space-filling vector quantization. KL Div refers to the Kullback-Leibler divergence of the observed histograms to the theoretical ideal distribution. Disclosure, ℓ_2/ℓ_1, and std refer to the worst-case disclosure, sparseness, and standard deviation of the histograms.

In privacy-preserving speech processing, one well-known method is to pass the speech through an information bottleneck[1] that is tight enough to remove the unnecessary private information from it. Vector quantized variational autoencoder (VQ-VAE) is a typical example in which the encoder compresses the input information to the bottleneck, and the decoder aims to reconstruct the input from bottleneck information. The information is usually quantized in the bottleneck to make the information rate quantifiable. When the bottleneck uses vector quantization, various codebook indices are generally used with different frequencies such that a small subset of speakers is potentially mapped to a specific codebook index. Therefore, in this situation, the range of speakers for this codebook index is smaller than the other codebook indices, and as a result, the chance of private information disclosure for the speakers mapped to this specific codebook index is higher.

In this publication, we used the proposed space-filling vector quantization (SFVQ) technique (Publication IV) to equalize the usage frequency for all codebook vectors and thus prevent the disclosure of the identity of speakers mapped to less frequent codebook indices. We used speech files

[1]Information funnel [115] is another concept in information theory that is closely related to information bottleneck, which aims to learn a compressed representation that retains the required information for the downstream task. In our paper, we used an information bottleneck because it is more commonly used in machine learning applications.

from the Common Voice corpus as input to the pretrained ECAPA-TDNN speaker verification model to map speakers to vectors in the embedding space. After training a B bit SFVQ on the speaker's embedding space, we map all training embedding samples on the learned SFVQ's curve and split the curve to 2^B segments such that each segment accommodates an equal number of samples. The average of samples in each segment is then defined as the resampled codebook vector, which is used for the final quantization. We call this approach the *codebook resampling* step. The occurrence probability of codebook vectors, when quantizing the speaker's embedding samples (from the unseen test set) by VQ and resampled SFVQ is illustrated in Fig. 5.6. The occurrence probability of codebook vectors is clearly more uniform in the resampled SFVQ approach, and hence it guarantees more privacy for the speakers.

6. Conclusions

In our opinion, there has not been much attention paid to improving the performance of vector quantization (VQ) in the machine learning (ML) domain, and this line of research has been neglected following the popularity of ML models (especially deep neural networks (DNNs)). Hence, in this thesis, our main goal is to enhance the performance of VQ in DNNs because any improvement could increase the performance of a large range of DNN-based tasks regardless of the application and data type. This chapter will explain how we answer this goal and the relevant outcomes.

DNNs is a category of machine learning models that learn and extract intricate patterns from data, which can be used to solve complicated problems. The availability of data and advancement in computing hardware (e.g., GPUs) in recent decades enable DNNs to achieve state-of-the-art performance in a wide range of domains.

VQ is a classic signal processing tool that can be used to model any data distribution with a set of codebook vectors, such that each codebook vector represents a limited area of the distribution. Since the nature of some data types (e.g., text) is discrete and VQ is a good fit for modeling a categorical distribution, VQ is commonly used in many DNN architectures to discretize the representation of a layer as a categorical distribution. Therefore, any small improvement in VQ can result in a significant boost in the performance of DNNs for a broad range of applications and all data types, such as speech, image, video, text, etc.

The conducted research in this thesis can be seen from four different perspectives. In the first perspective, we aim to improve the performance of some existing models by introducing VQ that reduces quantization distortion . In Publication I, we showed that by replacing VQ with scalar quantization, the quantization distortion is reduced because VQ takes the correlation among different data dimensions into account. In this paper, we modified the spectral envelope modeling in an ML-based speech codec [12] by replacing scalar quantization with VQ. By this modification, we achieved a higher perceptual quality for the compressed speech signal under the perceptual evaluation of speech quality (PESQ) metric, which is

an objective estimate of subjective quality. In Publication III, we observed that additive VQ and residual VQ lead to less quantization error than product VQ for sample distributions over various dimensionalities.

The main goal of the second perspective is to improve the efficiency of VQ training when using ML optimizers. In Publication II, we propose a new solution to address the *gradient collapse* problem (Sec. 3.1.4) called noise substitution in vector quantization (NSVQ). It simulates the behavior of VQ by adding noise to the input vector such that the noise is scaled to match the quantization error. We used NSVQ to optimize the vector quantization codebook with ML optimizers in speech coding and image compression applications. Our experiments show that NSVQ outperforms two state-of-the-art solutions of straight-through estimator and exponential moving average by providing faster convergence, more accurate gradients, and avoiding additional hyper-parameter tuning.

In Publication III, we also used NSVQ to optimize variants of VQ that use multiple codebooks for the quantization, such as additive VQ, residual VQ, and product VQ. We studied the trade-offs between the bitrate, accuracy, and complexity of these VQ variants within speech coding, image compression, and approximate nearest neighbor (ANN) search applications. In the ANN search, we compared the performance of VQ variants optimized by our ML-based NSVQ technique and the baseline results optimized by traditional optimization algorithms (e.g., k-means). The experiments show that VQ variants optimized by NSVQ achieve comparable results to the baselines under the *recall* metric. This means that variants of VQ can now be optimized in modern machine learning models without performance loss.

In the third perspective, we aim to improve the interpretability of the latent spaces in DNNs. Hence, in Publication IV, we introduced space-filling vector quantization (SFVQ) by incorporating space-filling curves into VQ. We used SFVQ to interpret the latent space of a voice conversion model based on a vector quantized variational autoencoder (VQ-VAE). Our experiments demonstrate that with SFVQ, we can explore the underlying phonetic structure of the latent space such that we can differentiate between latent vectors representing different phonetic groups such as silence, plosives, fricatives, nasals, vowels, semi-vowels, and diphthongs. We can also differentiate between different phones within a phonetic group.

In Publication V, we use our SFVQ technique to interpret the latent spaces of two well-recognized image generative models known as StyleGAN2 and BigGAN. We interpret the latent space from two different aspects: 1) by *universal interpretation* we aimed to find the mapping between the latent space and generative factors such as rotation, gender, age, race, etc, 2) by *interpretable directions* we intended to discover directions in the latent space that can change a semantic attribute of an image in a meaningful way. For example, if we shift a latent vector along the direction

corresponding to the change in gender attribute, the gender of the generated image will be altered when we move along this direction. Our results demonstrate that the latent space can be effectively interpreted by using SFVQ. Moreover, we found several more efficient interpretable directions compared to prior methods.

In the fourth perspective, we aim to improve privacy in speech processing applications based on DNNs. In Publication VI, we studied a privacy issue that had not been addressed before. The issue arises when a subset of speakers mapped to a particular codebook vector is proportionally much smaller than those mapped to other codebook vectors. In this case, the disclosure of private information is much greater for such outlier entries of the codebook. Our proposed method uses SFVQ to model the discrete embedding space of a speaker recognition network. To minimize the worst-case disclosure of private information, we equalize the number of speakers mapped to codebook indices by resampling the SFVQ's codebook. In this way, the occurrence frequencies of codebook entries are thus approximately uniform, and the worst-case disclosure of private information is minimized. Hence, an appropriate level of privacy for all speakers will be obtained.

In general, this thesis focuses on the efficient optimization of VQ models and their derivatives using modern ML and applies them to important and current challenges in the DNNs domain. Since VQ is a central element in many DNN architectures, the improvements proposed in this thesis are expected to enhance the performance of a broad range of DNN-based applications for many data types. For example, VQ is a crucial part of the most recent successful DNN-based speech and audio coding techniques [6, 116, 117].

Finally, this thesis proposes two novel and fundamental methods, NSVQ and SFVQ. NSVQ can be used to pass gradients through the non-differentiable VQ function during backpropagation in any ML optimization task that adopts gradient-based optimizers.

SFVQ is an improved version of VQ in which the data points can be mapped on a curve connecting subsequent codebook vectors. The desirable property of SFVQ is the arrangement in its codebook such that adjacent codebook indices refer to similar content. In general, SFVQ is a universal technique that can model any data distribution, and it is not confined to any specific neural network architecture or data type.

In conclusion, this thesis can improve the efficiency and performance of deep learning models by enhancing vector quantization across a wide range of trending applications, such as speech recognition, natural language processing, neural speech and audio coding. It also proposes solutions to make deep learning models more interpretable and enhance their users' privacy.

References

[1] T. Bäckström, O. Räsänen, A. Zewoudie, P. P. Zarazaga, L. Koivusalo, S. Das, E. G. Mellado, M. B. Mansali, D. Ramos, S. Kadiri, P. Alku, and M. H. Vali, *Introduction to Speech Processing*. 2 ed., 2022.

[2] C. C. Aggarwal, *Neural Networks and Deep Learning*, vol. 10. Springer, 2018.

[3] A. Razavi, A. van den Oord, and O. Vinyals, "Generating diverse high-fidelity images with VQ-VAE-2," in *Proceedings of NeurIPS*, pp. 14866–14876, 2019.

[4] P. Esser, R. Rombach, and B. Ommer, "Taming transformers for high-resolution image synthesis," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12873–12883, 2021.

[5] C. Gârbacea, A. v. den Oord, Y. Li, F. S. C. Lim, A. Luebs, O. Vinyals, and T. C. Walters, "Low bit-rate speech coding with VQ-VAE and a Wavenet decoder," in *Proceedings of ICASSP*, pp. 735–739, 2019.

[6] N. Zeghidour, A. Luebs, A. Omran, J. Skoglund, and M. Tagliasacchi, "Soundstream: an end-to-end neural audio codec," *arXiv preprint arXiv:2107.03312*, 2021.

[7] A. Baevski, H. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A framework for self-supervised learning of speech representations," 2020.

[8] P. Dhariwal, H. Jun, C. Payne, J. W. Kim, A. Radford, and I. Sutskever, "Jukebox: a generative model for music," *arXiv preprint arXiv:2005.00341*, 2020.

[9] A. Tjandra, B. Sisman, M. Zhang, S. Sakti, H. Li, and S. Nakamura, "VQ-VAE unsupervised unit discovery and multi-scale code2spec inverter for Zerospeech challenge 2019," in *Proceedings of Interspeech*, pp. 1118–1122, 2019.

[10] X. Wang, S. Takaki, J. Yamagishi, S. King, and K. Tokuda, "A vector quantized variational autoencoder (VQ-VAE) autoregressive neural F_0 model for statistical parametric speech synthesis," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 28, pp. 157–170, 2020.

[11] A. van den Oord, O. Vinyals, and K. Kavukcuoglu, "Neural discrete representation learning," in *Proceedings of NeurIPS*, pp. 6309–6318, 2017.

[12] T. Bäckström, M. B. Mansali, P. P. Zarazaga, M. Ranjit, S. Das, and Z. Lachiri, "PyAWNeS-Codec: speech and audio codec for ad-hoc acoustic wireless sensor networks," in *Proceedings of EUSIPCO*, 2021.

[13] H. Jegou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 117–128, 2010.

[14] Y. Chen, T. Guan, and C. Wang, "Approximate nearest neighbor search by residual vector quantization," *Sensors*, vol. 10, no. 12, pp. 11259–11273, 2010.

[15] A. Babenko and V. Lempitsky, "Additive quantization for extreme vector compression," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 931–938, 2014.

[16] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[17] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, "Analyzing and improving the image quality of stylegan," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8110–8119, 2020.

[18] A. Brock, J. Donahue, and K. Simonyan, "Large scale gan training for high fidelity natural image synthesis," in *International Conference on Learning Representations*, 2018.

[19] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4401–4410, 2019.

[20] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*. Springer, 1992.

[21] J. M. Blackledge, *Digital Image Processing: Mathematical and Computational Methods*. Elsevier, 2005.

[22] E. A. Riskin, T. Lookabaugh, P. A. Chou, and R. M. Gray, "Variable rate vector quantization for medical image compression," *IEEE Transactions on Medical Imaging*, vol. 9, no. 3, pp. 290–298, 1990.

[23] W. H. Equitz, "A new vector quantization clustering algorithm," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, no. 10, pp. 1568–1575, 1989.

[24] E. Lughofer, "Extensions of vector quantization for incremental clustering," *Pattern Recognition*, vol. 41, no. 3, pp. 995–1011, 2008.

[25] Y. Chen, S. Yang, N. Hu, L. Xie, and D. Su, "TeNC: low bit-rate speech coding with VQ-VAE and GAN," in *International Conference on Multimodal Interaction*, pp. 126–130, 2021.

[26] S. Ding and R. Gutierrez-Osuna, "Group latent embedding for vector quantized variational autoencoder in non-parallel voice conversion," in *Proceedings of Interspeech*, pp. 724–728, 2019.

[27] D.-Y. Wu, Y.-H. Chen, and H.-Y. Lee, "VQVC+: one-shot voice conversion by vector quantization and U-Net architecture," *arXiv preprint arXiv:2006.04154*, 2020.

[28] G. E. Henter, J. Lorenzo-Trueba, X. Wang, and J. Yamagishi, "Deep encoder-decoder models for unsupervised learning of controllable speech synthesis," *arXiv preprint arXiv:1807.11470*, 2018.

[29] M. Ahmed, R. Seraj, and S. M. S. Islam, "The k-means algorithm: A comprehensive survey and performance evaluation," *Electronics*, vol. 9, no. 8, p. 1295, 2020.

[30] "Data clustering: a review," *ACM Computing Surveys (CSUR)*, vol. 31, no. 3, pp. 264–323, 1999.

[31] J. Macqueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability/University of California Press*, 1967.

[32] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 39, no. 1, pp. 1–22, 1977.

[33] D. Arthur and S. Vassilvitskii, "k-means++ the advantages of careful seeding," in *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1027–1035, 2007.

[34] C. Leo, "The math and code behind k-means clustering." https://towardsdatascience.com/the-math-and-code-behind-k-means-clustering-795582423666, 2024. Accessed on June 13, 2024.

[35] R. L. Thorndike, "Who belongs in the family?," *Psychometrika*, vol. 18, no. 4, pp. 267–276, 1953.

[36] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in sarge spatial databases with noise," in *Proceedings of Knowledge Discovery and Data Mining*, 1996.

[37] N. Tsalkitzis, "DBSCAN algorithm explained easily!." https://pub.aimind.so/dbscan-algorithm-explained-easily-170d57ca88da, 2023. Accessed on June 13, 2024.

[38] H.-C. Huang, J.-S. Pan, Z.-M. Lu, S.-H. Sun, and H.-M. Hang, "Vector quantization based on genetic simulated annealing," *Signal Processing*, vol. 81, no. 7, pp. 1513–1523, 2001.

[39] M. G. Omran, A. P. Engelbrecht, and A. Salman, "A color image quantization algorithm based on particle swarm optimization," *Informatica*, vol. 29, no. 3, 2005.

[40] M.-H. Horng, "Vector quantization using the firefly algorithm for image compression," *Expert Systems with Applications*, vol. 39, no. 1, pp. 1078–1091, 2012.

[41] S. Nag, "Vector quantization using the improved differential evolution algorithm for image compression," *Genetic Programming and Evolvable Machines*, vol. 20, pp. 187–212, 2019.

[42] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, pp. 341–359, 1997.

[43] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychological Review*, vol. 65, no. 6, p. 386, 1958.

[44] T. Tieleman and G. Hinton, "Lecture 6.5-RMSPROP: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural Networks for Machine Learning*, vol. 4, no. 2, pp. 26–31, 2012.

[45] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization.," *Journal of Machine Learning Research*, vol. 12, no. 7, 2011.

[46] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[47] D. Kapil, "Stochastic vs batch gradint descent." `https://medium.com/@divakar_239/stochastic-vs-batch-gradient-descent-8820568eada1`, 2019. Accessed on June 13, 2024.

[48] T. Eriksson and E. Agrell, "Lattice-based quantization, Part II," tech. rep., Chalmers University of Technology, 1996.

[49] S. C. Shapiro, "Encyclopedia of Artificial Intelligence," 1988.

[50] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[51] J. Rissanen and G. G. Langdon, "Arithmetic coding," *IBM Journal of Research and Development*, vol. 23, no. 2, pp. 149–162, 1979.

[52] F. Chollet, *Deep Learning with Python*. Simon and Schuster, 2021.

[53] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning*. MIT Press, 2018.

[54] S. I. Serengil and A. Ozpinar, "Lightface: A hybrid deep face recognition framework," in *2020 Innovations in Intelligent Systems and Applications Conference*, pp. 1–5, IEEE, 2020.

[55] Z. Bai and X.-L. Zhang, "Speaker recognition based on deep learning: An overview," *Neural Networks*, vol. 140, pp. 65–99, 2021.

[56] A. P. Rodrigues, R. Fernandes, A. Shetty, K. Lakshmanna, R. M. Shafi, *et al.*, "Real-time twitter spam detection and sentiment analysis using machine learning and deep learning techniques," *Computational Intelligence and Neuroscience*, vol. 2022, 2022.

[57] F. K. Alarfaj, I. Malik, H. U. Khan, N. Almusallam, M. Ramzan, and M. Ahmed, "Credit card fraud detection using state-of-the-art machine learning and deep learning algorithms," *IEEE Access*, vol. 10, pp. 39700–39715, 2022.

[58] F. Javanmardi, S. R. Kadiri, and P. Alku, "Pre-trained models for detection and severity level classification of dysarthria from speech," *Speech Communication*, p. 103047, 2024.

[59] C. R. Madhuri, G. Anuradha, and M. V. Pujitha, "House price prediction using regression techniques: A comparative study," in *2019 International Conference on Smart Structures and Systems (ICSSS)*, pp. 1–5, IEEE, 2019.

[60] Z. Hu, Y. Zhao, and M. Khushi, "A survey of forex and stock price prediction using deep learning," *Applied System Innovation*, vol. 4, no. 1, p. 9, 2021.

[61] P. Hewage, M. Trovati, E. Pereira, and A. Behera, "Deep learning-based effective fine-grained weather forecasting model," *Pattern Analysis and Applications*, vol. 24, no. 1, pp. 343–366, 2021.

[62] R. Guan, H. Zhang, Y. Liang, F. Giunchiglia, L. Huang, and X. Feng, "Deep feature-based text clustering and its explanation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 8, pp. 3669–3680, 2020.

[63] M. R. Karim, O. Beyan, A. Zappa, I. G. Costa, D. Rebholz-Schuhmann, M. Cochez, and S. Decker, "Deep learning-based clustering approaches for bioinformatics," *Briefings in Bioinformatics*, vol. 22, no. 1, pp. 393–415, 2021.

[64] Z. Abbasi-Moud, H. Vahdat-Nejad, and J. Sadri, "Tourism recommendation system based on semantic clustering and sentiment analysis," *Expert Systems with Applications*, vol. 167, p. 114324, 2021.

[65] T. Liu, Z. Li, C. Yu, and Y. Qin, "NIRS feature extraction based on deep auto-encoder neural network," *Infrared Physics & Technology*, vol. 87, pp. 124–128, 2017.

[66] R. Khan, "Nothing but NumPy: Understanding & creating neural networks with computational graphs from scratch." `https://towardsai.net/p/machine-learning/nothing-but-numpy-understanding-creating-neural-networks-with-computational-graphs-from-scratch-6299901091b0`, 2019. Accessed on June 13, 2024.

[67] J. Fumo, "A gentle introduction to neural networks series - Part 1." `https://towardsdatascience.com/a-gentle-introduction-to-neural-networks-series-part-1-2b90b87795bc`, 2017. Accessed on June 13, 2024.

[68] K. E. Koech, "The basics of neural networks (neural network series) - Part 1." `https://towardsdatascience.com/the-basics-of-neural-networks-neural-network-series-part-1-4419e343b2b`, 2022. Accessed on June 13, 2024.

[69] R. Kruse, S. Mostaghim, C. Borgelt, C. Braune, and M. Steinbrecher, "Multilayer perceptrons," in *Computational Intelligence: A Methodological Introduction*, pp. 53–124, Springer, 2022.

[70] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[71] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[72] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[73] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, 2014.

[74] A. van den Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *arXiv preprint arXiv:1807.03748*, 2018.

[75] Z. Yang, W. Dong, X. Li, M. Huang, Y. Sun, and G. Shi, "Vector quantization with self-attention for quality-independent representation learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 24438–24448, 2023.

[76] J. Park, K. Choi, H. Heo, and H.-M. Park, "Unsupervised speech representation pooling using vector quantization," *arXiv preprint arXiv:2304.03940*, 2023.

[77] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3156–3164, 2015.

[78] J. Rocca, "Understanding variational autoencoders (VAEs)." https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73, 2019. Accessed on June 13, 2024.

[79] B. van Niekerk, L. Nortje, and H. Kamper, "Vector-quantized neural networks for acoustic unit discovery in the Zerospeech 2020 challenge," in *Proceedings of Interspeech*, pp. 4836–4840, 2020.

[80] E. Agustsson, F. Mentzer, M. Tschannen, L. Cavigelli, R. Timofte, L. Benini, and L. Van Gool, "Soft-to-hard vector quantization for end-to-end learning compressible representations," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 1141–1151, 2017.

[81] X. Peng, I. W. Tsang, J. T. Zhou, and H. Zhu, "K-meansNet: when k-means meets differentiable programming," *arXiv preprint arXiv:1808.07292*, 2018.

[82] T. Yu, J. Yuan, C. Fang, and H. Jin, "Product quantization network for fast image retrieval," in *European Conference on Computer Vision*, pp. 191–206, Springer, 2018.

[83] J. Yang, X. Shen, J. Xing, X. Tian, H. Li, B. Deng, J. Huang, and X.-s. Hua, "Quantization networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7308–7316, 2019.

[84] S. Kumar, "C-means clustering explained." https://builtin.com/data-science/c-means, 2022. Accessed on June 13, 2024.

[85] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: training neural networks with low precision weights and activations," *Machine Learning Research*, vol. 18, pp. 1–30, 2018.

[86] J. Chorowski, R. J. Weiss, S. Bengio, and A. van den Oord, "Unsupervised speech representation learning using Wavenet autoencoders," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 27, no. 12, pp. 2041–2053, 2019.

[87] Y. Zhao, H. Li, C.-I. Lai, J. Williams, E. Cooper, and J. Yamagishi, "Improved prosody from learned F0 codebook representations for VQ-VAE speech waveform reconstruction," *arXiv preprint arXiv:2005.07884*, 2020.

[88] H. Askary, "Intuitive explanation of straight-through estimators with pytorch implementation." https://hassanaskary.medium.com/intuitive-explanation-of-straight-through-estimators-with-pytorch-implementation-71d99d25d9d0, 2023. Accessed on June 13, 2024.

[89] R. Gong, X. Liu, S. Jiang, T. Li, P. Hu, J. Lin, F. Yu, and J. Yan, "Differentiable soft quantization: bridging full-precision and low-bit neural networks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[90] C. C. Holt, "Forecasting trends and seasonals by exponentially weighted averages," *ONR Memorandum*, vol. 52, p. 1957, 1957.

[91] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with Gumbel-Softmax," in *International Conference on Learning Representations*, 2017.

[92] E. J. Gumbel, *Statistical theory of extreme values and some practical applications: a series of lectures*, vol. 33. US Government Printing Office, 1954.

[93] C. J. Maddison, D. Tarlow, and T. Minka, "A* sampling," *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[94] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," *arXiv preprint arXiv:1611.00712*, 2016.

[95] H. Sagan, *Space-Filling Curves*. Springer Science & Business Media, 2012.

[96] M. Bader, *Space-Filling Curves: An Introduction with Applications in Scientific Computing*, vol. 9. Springer Science & Business Media, 2012.

[97] M. M. Flood, "The Traveling-Salesman Problem," *Operations Research*, vol. 4, no. 1, pp. 61–75, 1956.

[98] D. S. Johnson and L. A. McGeoch, "The traveling salesman problem: A case study in local optimization," *Local Search in Combinatorial Optimization*, 1997.

[99] N. Christofides, "Worst-case analysis of a new heuristic for the traveling salesman problem," *Graduate School of Industrial Administration, Carnegie Mellon University*, 1976.

[100] K. Buchin, "Constructing delaunay triangulations along space-filling curves," in *European Symposium on Algorithms*, pp. 119–130, Springer, 2009.

[101] J. J. Bartholdi III and L. K. Platzman, "Heuristics based on spacefilling curves for combinatorial problems in Euclidean space," *Management Science*, vol. 34, no. 3, pp. 291–305, 1988.

[102] H.-L. Chen and Y.-I. Chang, "Neighbor-finding based on space-filling curves," *Information Systems*, vol. 30, no. 3, pp. 205–226, 2005.

[103] S. Liao, M. A. López, and S. T. Leutenegger, "High dimensional similarity search with space filling curves," in *Proceedings 17th International Conference on Data Engineering*, pp. 615–622, IEEE, 2001.

[104] G. Mainar-Ruiz and J. Perez-Cortes, "Approximate nearest neighbor search using a single space-filling curve and multiple representations of the data points," in *18th International Conference on Pattern Recognition (ICPR'06)*, vol. 2, pp. 502–505, IEEE, 2006.

[105] D. J. Abel and D. M. Mark, "A comparative analysis of some two-dimensional orderings," *International Journal of Geographical Information Systems*, vol. 4, no. 1, pp. 21–31, 1990.

[106] X. Liu and G. F. Schrack, "A new ordering strategy applied to spatial data processing," *International Journal of Geographical Information Science*, vol. 12, no. 1, pp. 3–22, 1998.

[107] J. Wang and J. Shan, "Space filling curve based point clouds index," in *Proceedings of the 8th International Conference on GeoComputation*, pp. 551–562, 2005.

[108] K. Abend, T. Harley, and L. Kanal, "Classification of binary random patterns," *IEEE Transactions on Information Theory*, vol. 11, no. 4, pp. 538–544, 1965.

[109] B. Moon, H. V. Jagadish, C. Faloutsos, and J. H. Saltz, "Analysis of the clustering properties of the Hilbert space-filling curve," *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 1, pp. 124–141, 2001.

[110] A. Lempel and J. Ziv, "Compression of two-dimensional data," *IEEE Transactions on Information Theory*, vol. 32, no. 1, pp. 2–8, 1986.

[111] T. Bially, "Space-filling curves: Their generation and their application to bandwidth reduction," *IEEE Transactions on Information Theory*, vol. 15, no. 6, pp. 658–664, 1969.

[112] M. Elshafei-Ahmed, "Fast methods for split codebooks," *Signal Processing*, vol. 80, no. 12, pp. 2553–2565, 2000.

[113] S. Dieleman, A. van den Oord, and K. Simonyan, "The challenge of realistic music generation: modelling raw audio at scale," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[114] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, and N. L. Dahlgren, *The DARPA TIMIT Acoustic-Phonetic Continuous Speech Corpus CDROM*. Linguistic Data Consortium, 1993.

[115] A. Makhdoumi, S. Salamatian, N. Fawaz, and M. Médard, "From the information bottleneck to the privacy funnel," in *2014 IEEE Information Theory Workshop (ITW 2014)*, pp. 501–505, IEEE, 2014.

[116] A. Défossez, J. Copet, G. Synnaeve, and Y. Adi, "High fidelity neural audio compression," *arXiv preprint arXiv:2210.13438*, 2022.

[117] R. Kumar, P. Seetharaman, A. Luebs, I. Kumar, and K. Kumar, "High-fidelity audio compression with improved RVQGAN," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

Business, Economy

Art, Design, Architecture

Science, Technology

Crossover

Doctoral Theses

Aalto DT 13/2025

ISBN 978-952-64-2361-6
ISBN 978-952-64-2362-3 (pdf)

Aalto University
School of Electrical Engineering
Department of Information and Communications Engineering
aalto.fi