Deep Neural Yodelling

A thesis to attain the degree of
Master of Science in Engineering

presented by
Daniel Pfäffli

Departement Information Technology
Lucerne University of Applied Sciences and Arts
6343 Rotkreuz, Switzerland

March 7, 2018

Advisor Prof. Dr. Marc Pouly
Lucerne University of Applied Sciences and Arts, 6343 Rotkreuz, Switzerland
marc.pouly@hslu.ch
Expert Dr. Matthias Buchs
Innovation Process Technology AG, 3011 Bern, Switzerland
matthias.buchs@ipt.ch

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche verwendeten Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Rotkreuz, 7. März 2018, Daniel Pfäffli

Abstract

Jodelmusik unterscheidet sich von anderen Genres dadurch, dass beim Übergang von der Bruststimme zum Falsett der Kehlkopfschlag gut hörbar ist. Des Weiteren besteht sie oft aus einem Vorjodler, der von einem Chor begleitet wird. In der Schweiz wird zwischen Jodelliedern und Naturjodel unterschieden, wobei Jodellieder Text enthalten und für Naturjodel nur sogenannte Jodelsilben verwendet werden, wie "u" für hohe und "o" für tiefe Noten. Die heutigen Methoden für Musikgenerierung mit maschinellen Lernverfahren basieren auf neuronalen Netzwerken, welche in Schichten von Knoten, die aufeinandergelegt werden, aufgebaut sind. Diese sind dabei mit Knoten der unteren und oberen Schicht verbunden. Mit Hilfe von echter Musik werden die Netzwerke trainiert, wobei die Daten in Form von Notenblättern oder Audiosignalen vorliegen können. Nach dem Trainieren generiert das Modell neue Stücke desselben Types mit Hilfe eines zufälligen Anfangswertes, welcher als Eingabe dient. Als Trainingsdaten verwenden viele Verfahren Audioaufnahmen eines Pianos (van den Oord et al. (2016a)) oder eines klassischen Orchesters (Mehri et al. (2016)). Gesang (Nayebi and Vitelli (2015)) hat bisweilen, gemessen an der Tonqualität der generierten Musik, schlechtere Resultate geliefert als die anderen Genres.

Diese Arbeit ist die erste, nach bestem Wissen des Autors, die versucht mit maschinellen Lernverfahren Jodelmusik zu produzieren. Das Departement Musik der Hochschule Luzern stellte für diese Thesis 17h Naturjodel und 17h Jodellieder in Audioaufnahmen zur Verfügung. Diese Masterarbeit verwendete sogenannte Convolution neuronale Netzwerke, wobei Musik in Form von Audiosignalen als Trainingsdaten dienten. In einem ersten Schritt wurde untersucht, ob generative Modelle typische Charakteristiken von Naturjodel wiedergeben können. Experimente mit dem Verfahren WaveNet (van den Oord et al. (2016a)), welches für die Generierung von Musik als Audiosignal vorgeschlagen wurde, belegten, dass generative Ansätze die Klangfarbe von Jodel erlernen können. Zwei Experten bestätigten diesen Befund in dieser Arbeit. Weitere Untersuchungen zeigten, dass das Modell eine Jodel-ähnliche Vokalisierung verwendet. Für hohe Töne ist ein "u" hörbar und für Tone des Brustregisters ein "o", wie dies sonst in diesem Genre üblich ist. Zusätzlich in allen generierten Stücken ist ein Jodelchor hörbar, der akkordisch begleitet. Beides sind typisch Merkmal von Jodel. Ob der charakteristische Kehlkopfschlag hörbar ist, darüber herrschte unter den beiden Experten keine Einigkeit. Die generierten Stücke haben jedoch keinen Melodieaufbau und keine Liedform. Das Metrum wurde für monoton und unpräzis befunden. Zusätzlich ist ein störender Schlag zu hören, welcher die Harmonie der Musik bricht. Zusammenfassend konnte ein WaveNet-basiertes Modell die Klangfarbe von Jodel wiedergeben, jedoch nicht Jodel komponieren.

Mit Hilfe des Verfahrens Variational Autoencoder wurde ein anderer Modell-Typ untersucht, welcher die erwähnten Defizite beheben könnte. Jedoch gelang es bis zum Ende dieser Arbeit nicht, etwas anderes als Rauschen zu produzieren, was mit der zu kurzen Experimentier- und Experiment-Dauer zu begründen ist. Deshalb kann keine Schlussfolgerung zu diesem Verfahren gezogen werden. Es wurden zusätzlich Experimente zu unterschiedlichen Ausgangsverteilungen gemacht, welche eine Reduktion der letzten Schicht von 256 auf 60 Knoten erlauben würden. Jedoch ergab das Experiment keine aussagekräftigen Resultate.

Zusammenfassend kann gesagt werden, dass das Model WaveNet die Klangfarbe des Jodels wiedergeben kann. Jedoch wäre mehr Forschung notwendig, um eine Jodelkomposition zu produzieren.

Abstract

Yodel music differs from most other genres by exercising the transition from chest voice to falsetto with an audible glottal stop which is recognised even by laymen. Yodel often consists of a yodeller with a choir accompaniment. In Switzerland, it is differentiated between the natural yodel and yodel songs. Yodel songs contain text, and natural yodel uses yodel syllabus such as “u” for high notes and “o” for low notes. Today’s approaches to music generation with machine learning algorithms are based on neural networks, which are best described by stacked layers of neurons which are connected with neurons of the lower layer. The networks are trained using real music samples in lead sheet or audio waveform to adapt their weights, so they produce similar ones. So far, proposed procedures have used piano music (van den Oord et al. (2016a)) or polyphonic music (Mehri et al. (2016)) that consisted of a classical orchestra. Approaches to vocal music were rarely found (Nayebi and Vitelli (2015)), and measured in sound quality less successful than for instrumental music.

To the best of the author’s knowledge, up until this thesis, it has not been tried to produce yodel music with machine learning algorithms. The Lucerne University of Applied Sciences and Arts, Department Music, provided in total 17h of natural yodel and 17h of yodel songs. For this work, convolution neural networks were used to investigate if generative approaches can produce yodel-like music in the audio waveform. The first research question examined if generative approaches can imitate characteristics of natural yodel music. Experiments with the model WaveNet (van den Oord et al. (2016a)) which was proposed for music synthesis in the audio waveform confirmed the ability of generative methods to produce the timbre of yodel music. Two experts affirmed the conclusion in this thesis. Further investigations revealed that the model reproduces yodel-like vocalisation where high notes are sung with an “u” and breast notes with an “o”. Additionally, all samples contain an audible chorus that accompanies in triads. Both are typical characteristics of natural yodel. For other attributes such as the glottal stop, there was no consensus among the experts. The model fails to provide a melody structure, and changes in metre are missed utterly. It produces a disturbing ”clapping”-effect which breaks the harmony of the samples. In summary, a WaveNet-based model successfully produced the timbre of yodel but was unable to compose yodel music.

The generative approach Variational Autoencoder was investigated to tackle the shortcomings of WaveNet. Unfortunately, it did not succeed in producing anything different than noise which is explained by its short experimental time. Therefore, it is inconclusive whether the approach is appropriate for yodel music. Additionally, another output distribution was investigated which would allow a reduction of the output layer from 256 to 60 neurons. However, the experiment reported inconclusive results as the model did not converge.

To conclude, the model WaveNet achieved to produce the timbre of yodel music. However, more research is necessary to generate entire yodel compositions.

Contents

Chapter 1

Introduction

This chapter provides an introduction to "Deep Neural Yodelling". Section 1.1 describes the motivation of this thesis. In Section 1.2 the research questions that have been proposed for this project are introduced. Section 1.4 describes the data corpus and the available infrastructure.

1.1 Motivation

In recent years, deep learning with neural networks has led to significant progress in computer intelligence. Algorithms outplay the abilities of humans in areas such as image recognition and recently in the board game Go where Google's Alpha-Go[1] won against the world champion. New services like chat-bots, simultaneous translation in real time and autonomous driving cars are no longer research projects only, but developed and used by companies all over the world. New approaches, based on so-called "generative models", claim to produce creativity and demonstrate this by computer-generated jazz and watercolour-like pictures. The project "Deep Neural Yodelling" has been proposed by the Departments Information Technology(IT) and Music of the Lucerne University of Applied Sciences and Arts to investigate if neural networks can acquire the ability to compose alphorn and yodel songs and to explore the application in industrial projects (Pouly and Kammermann (2015)).

Generative models already proved their ability to produce music in genres pop, jazz and classical piano. To the best of the author's knowledge, it has never been applied to yodel and alphorn. So far, approaches were investigated based on technical analysis such as spectrum analysis of frequencies or reconstruction error. An evaluation based on a scientific background in music-theory is missing. The involvement of experts for yodelling and alphorn allows assessing the output in such a way. Thereby shall be investigated if the peculiar characteristics of alphorn and yodel music, like the usage of harmonic series instead of the classical, occurs in generated samples.

In this thesis, the ability of generative approaches to the task of yodel music is explored. A state of the art analysis has shown that successful approaches usually contained two success factors: a large data corpus of the specific type to reproduce and music based on instruments only(no vocals). Furthermore, the approaches all rely on music that uses the tempered series. The peculiarity of the harmonic series in combination with vocals used in yodel turns the subject to an interesting research question and, to the best of the author's knowledge, has not been done before. Promoted results of generative-music approaches have been mostly studied on a technical level but missed to perform a music scientific investigation of the outcome so far. The involvement of the Department of Music allows giving a profound analysis of the result based on music theory.

[1] https://deepmind.com/research/alphago/, accessed 28.12.2017

1.2 Research Questions and Expectations

The following research questions have been compiled for this project:

1. Can a generative model produce characteristics of yodel music recognisable by human experts?
2. Does a generative model produce yodel music which in a significant number of cases is not distinguishable from *real* samples for human experts?

The following goals have been defined:

1. A researcher in music science certifies typical characteristics of yodel music in produced samples in a qualitative way.
2. 40% of the generated tunes are not detected by human experts for yodel music. For the quality assessment, three human experts will perform a majority voting. There are 10 seconds of music or a tune of the length of 40 beats to decide whether a sample was generated or not.

1.3 Introduction to experts

Andrea Kammermann completed her training as a primary school teacher at the Pädagogische Hochschule Luzern. She studied music pedagogy at the Zurich University of the Arts, specialising in music and movement (rhythmic). She worked for several years at primary schools and music schools. Since 2015, Andrea Kammermann has been a research associate at the Department of Music of the Lucerne University of Applied Sciences and Arts where she investigates the musical relationship between alphorn and yodel[2]. Additionally, she examines yodel and emotions as part of her dissertation.

Nuria Richner, a soprano who grew up in Flims, initially studied singing with Hans-Jürg Rickenbacher at the music academy of the city of Basel. In 2012 she completed her Bachelor's degree in singing at the Lucerne University of Applied Sciences and Arts with Barbara Locher as the lecturer. After completing her Master of Performance in January 2015, she studied at Lucerne University of Applied Sciences and Arts with the lecturers Barbara Locher and Petra Hoffmann in the Master of Arts in Music Pedagogy programme and received her pedagogical diploma in June 2017. Since September 2017, Nuria Richner has been attending a CAS continuing education course in music research at the Lucerne University of Applied Sciences and Arts. In addition to her studies and her work as a singer, she teaches solo singing at the music schools Schwyz and Muotathal and leads the children's choir Muotathal. Her passion for contemporary music enabled her to collaborate with various composers, including Helmut Lachenmann and Thomas Fortmann. Through her mother, who runs a yodel club and is an enthusiastic alpine horn player, folk music and yodelling are a constant companion alongside classical singing. Nuria Richner attended various yodelling courses and combines baroque music with yodelling in duet "Barodel" together with the countertenor and yodeler Stefan Wieland. Nuria Richner is a laureate of the Swiss Youth Music Competition and the Ruth and Ernst Burkhalter Foundation and was selected in 2017 for the Young Professionals Program of Evta.

1.4 Starting Position

The project "Deep Neural Yodelling" is a joint-work between the Department Information Technology and the Department of Music of the Lucerne University of Applied Sciences and Arts. The project team provided audio files in WAV-format to be used within this thesis. In a total, there are 34h of yodel music available. The focus of this thesis was set to natural yodelling, and therefore, the corpus was categorised as follows:

1. Natural yodelling without instrumental accompaniment: ≈15h
2. Natural yodelling with instrumental accompaniment: ≈2h
3. Yodelling without instrumental accompaniment: ≈11h

[2]https://www.hslu.ch/en/lucerne-university-of-applied-sciences-and-arts/research/projects/detail/?pid=2162, accessed 15.01.2018

4. Yodelling with instrumental accompaniment: ≈6h

Furthermore, all songs have been annotated with the information to what region they belong. Additionally, there are 300 pieces of alphorn scores and 13h of alphorn music available.

1.4.1 Problematic Generative Models

Generative models have to capture dependencies between predicted data points. E.g. for images adjacent pixels should have a similar colour. The same accounts for music where certain notes do not follow others as it would result in a strange sound (e.g. tone C followed by a $C\#$).

Assuming a model $D(x)$ that can distinguish between

- x sounds like yodelling and returns a high probability,
- x does not sound like yodelling, therefore results in a low probability.

Even though a perfect discriminator $D(x)$, which identifies yodel music as mentioned above, might be already challenging to formulate, it would not help much to synthesise music as well. The learnt structure within the model may be appropriate for classification but on an abstract level. Details which are necessary to produce colourful music, may not be relevant for the discriminator. Therefore, a second model G, called generator, is needed for the model's output distribution p_g to be similar to an unknown samples distribution p_{gt} (e.g. yodel music). In simple words, the model produces songs such as yodel music. Note that for all machine learning approaches the assumption is made that p_{gt} is covered through a sufficiently large data corpus $\mathcal{X}$ from which countless samples $x \in \mathcal{X}$ can be drawn. In reality, it is often unknown if $\mathcal{X}$ is representative. Many different approaches have been proposed to create such generative models, but most suffer from three serious drawbacks:

1. They require strong assumptions about the structure of the samples to produce.
2. Severe approximations are necessary which results in suboptimal modules (e.g. music always has the same metre or does only use high notes).
3. They rely on computational expensive inference procedures like Markov Chain Monte Carlo.

In recent years, new approaches based on neural networks were proposed like generative adversarial networks (GAN) or Variational Autoencoders (VAE) which enable overcoming some of the disadvantages mentioned above. The computational capacities have been increased tremendously through developments in graphics processing units(GPU) which allow using up to 24 GB RAM with up to 3560 floating point units now. Therefore, larger models with more expressiveness can be trained in less time. Both factors are essential when it comes down to learn complex structures such as an audio signal.

1.4.2 Resources

Deep neural networks architectures are computationally expensive what results in long-running experiments. Using powerful GPUs reduces the processing time up to a factor of 20. The Lucerne University of Applied Science and Arts supported this thesis with a GPU Nvidia M40, 12 GB RAM, and, from November 2017 till the end of January 2018, with a GPU Nvidia P6000, 24 GB RAM. Additionally, they allowed the use of other GPUs whenever they were unused.

Furthermore, SWITCH[3] provided two GPUs Nvidia P100, 16 GB RAM, to be used exclusively during this thesis for a discount price of 2000 CHF. SWITCH is a cloud provider in Switzerland and plans to offer GPUs as part of their cloud services. This thesis served as a test of their infrastructure. Additionally, all Switch Engine services were free to use. In a total 48 CPUs, 512 GB RAM, 2 TB hard disk drive(HDD) space and 2 GPUs allocatable to maximal two instances (one per GPU) were available. Storage which was used within the S3 Object Storage[4] did not count to the 2 TB HDD limit.

[3] https://www.switch.ch/, accessed 15.01.2018
[4] https://help.switch.ch/engines/documentation/s3-like-object-storage/, accessed 15.01.2018

1.5 Demarcation

This thesis investigates the ability of generative approaches to yodel audio files directly. Some techniques compose music based on a lead sheet notation. Because it is uncommon for yodel music to be written in lead sheet notation and since there is no synthesiser available which can produce yodel, it has been decided to reproduce the audio signal directly.

1.6 Terms and Notation

Yodel in Switzerland is differentiated between the natural yodel and yodel songs. Where yodel songs contain text, natural yodel uses yodel syllabus only. In this thesis, the term yodel refers to natural yodel. If yodel songs are meant, then it is stated as "yodel with text".

The term deep learning is often used to denote approaches that contain many layers. While it is not directly related to neural networks most "deep learning" approaches in recent years, have been based on neural networks architectures.

Table 1.1 contains the notation used in this thesis.

Term	**Definition**
x	input
$g_\Theta(x)$	non-linear activation function
$a_i^{(l)}$	activation of node i of layer l
$z_i^{(l)}$	Inner activation value of node i of layer l
$\Theta^{(l)}$	matrix of weights controlling function mapping from layer l with $\Theta^{(l)} \in \mathbb{R}^{n^l \times n^{l-1}}$
Θ	The model's parameter, weights of all layers $\Theta = (\Theta^{(1)}, .., \Theta^{(L)})$
$f_\Theta(x)$	The model function
L	Number of layers
$\mathcal{L}$	Loss function
n	number of features
m	sample count
$\cdot$	Multiplication
$\odot$	elementwise multiplication
$\delta_j^{(l)}$	Error of node j at layer l
RNN specific	
$a_i^{(l,t)}$	activation of node i of layer l at time step t
$z_i^{(l,t)}$	Inner activation value of node i of layer l at time step t
$\Theta_{(c)}^{(l)}$	matrix of weights of layer l with c denoting the component or gate
CNN specific	
$(f \star g)(c)$	convolution of function f and g at position c
$k^{(l)}$	Filter size of layer l
$M^{(l)}$	Resulting feature maps of convolutional layer l
VAE specific	
$p(x)$	probability density function
$\mathcal{X}$	entity of samples x
$z \in \mathcal{Z}$	latent variables
$p(z)$	probability of latent variables
$f(z;\theta)$	family of deterministic functions
I	Identity matrix
$p_\pi(x\|z)$	probability of decoder constructing x under z
$q_\phi(z\|x)$	probability of encoder constructing z under x

Table 1.1: Terms and notation used in this thesis.

1.7 An introduction to yodelling

In Europe, yodelling is mainly cultivated in regions with alpine traditions. It is not a sole achievement of inhabitants of the Alps. Yodel-music has been reported to exist with the pygmies, African tribes, tribes of the Bantu, cameleer in south Arabia and Indian tribes in America.

According to Leuthold (1981) yodelling is a song without words born of joy. Some more specific characteristics are given as follows:

1. The transition of the voice from chest voice to falsetto. While classical singers try to avoid that the change of register is heard by any listener, yodellers force this change which is recognisable as a glottal stop. It is a unique characteristic of yodelling.

2. Bridging large musical intervals with legato. Legato is a technique of a smooth transition between notes.

3. As a result of the usage of falsetto, the melodies have large ranges.

4. No text but yodel syllables.

5. Vocalisation of high notes often with "u" and low notes with "o"(depending on the region).

6. Question-and-answer characteristic in phrases.

7. Most yodel contains a lead singer and a polyphonic choir accompaniment switching between the tonic and the dominant (if major C series then C-G). Often, the lead singer starts the song and is joined after a while by the chorus.

The typical transition of the voice is an indication that the yodelling may originate from the imitation of wind instruments, specifically of those playing the harmonic series.

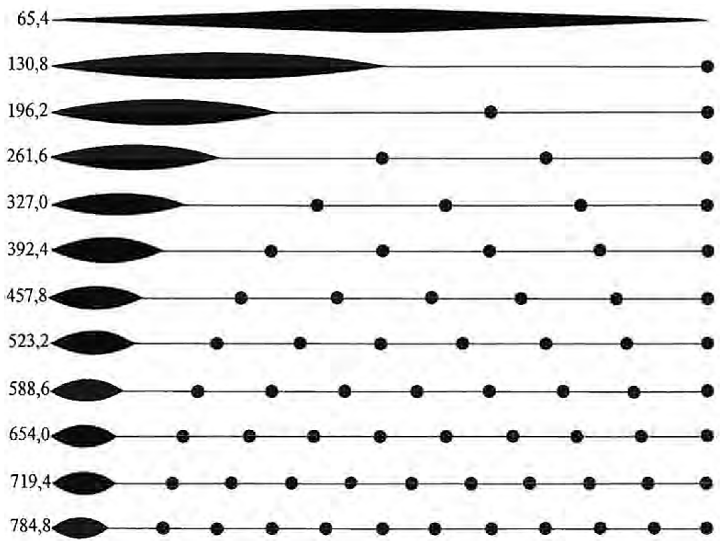


Figure 1.1: Creating the just scale explained on a string ((Leuthold, 1981, p.29)).

Notes of the harmonic series are created by the multiplication of frequency. It is best explained by imagining a single string of a guitar. Given the resonance is tone C with a frequency of 65.4 by increasingly dividing the string it follows what is shown in Table 1.1. One of the characteristics of the harmonic series lies in its just temperament. The harmonics 7, 11, 13 and 14 differ recognisably in sound from the equal tempered tuning. The tempered tuning is best explained by the cent measure which is a logarithmic unit that expresses the difference of the interval from frequency of a note a to frequency of the note b. It is given by (Ellis, 1885, p.487)

$$cent(a,b) = 1200 \cdot \log_2 \left(\frac{b}{a}\right). \tag{1.1}$$

The intuition behind this unit is that semitones of the twelve-tone equal temperament chromatic scale are 100 cents apart. Table 1.2 shows the cent measure in the twelve-tone equal temperament chromatic scale between C4 to C5 in respect to the note C4.

Notes	**Frequency**[5]	**Cents**	**Interval to C4**
C4	261.63	0	Unison
C#4	277.18	100	Minor second
D4	293.66	200	Major second
D#4	311.13	300	Minor third
E4	329.63	400	Major third
F4	349.23	500	Perfect fourth
F#4	369.99	600	Tritone
G4	392	700	Perfect fifth
G#4	415.3	800	Minor sixth
A4	440	900	Major sixth
A#4	466.16	1000	Minor seventh
B4	493.88	1100	Major seventh
C5	523.25	1200	Octave

Table 1.2: The frequencies of notes of the twelve-tone equal temperament chromatic scale with the difference in cents in respect to C4((Ellis, 1885, p.488)).

Yodel in Switzerland is sometimes based on the harmonic series and singers will use the ecmelic harmonics. Classical singers would adjust their voice to sing the pure note. The alphorn instrument is capable of playing this harmonic series only which led to the hypothesis that yodelling is related to these old instruments. So far such a relation has not been confirmed nor rejected.

Yodel music is mainly based on the major scale, and it is rarely available in written form. Only in the last few decades, some began to write down the songs as sheet music. This informal characteristic influenced the way how yodel singers interpreted the songs. There is much more space for interpretation than in classical music which makes it especially hard for the translation into formal music sheets. Therefore, yodel songwriters annotate their pieces of music usually with an informal expression like, e.g. “Alplerisch langsam”(Alplary slow) or “Etwas schneller”(a little bit faster) to refer to the pace what is not as strictly to interpret as in other genres.

According to (Leuthold, 1981, p.50), an additional characteristic of yodel marks its rhythmical structure which is based on a question-and-answer game that is repeated throughout the song. Despite the fact that in music sheet notation the metric is fixed, a yodel singer would interpret it as a proposition rather than a restriction. However, the yodeller will keep the rhythmical structure as an arc of suspense. Figure 1.2 shows the music sheet notation of the song *”Luegere Juiz (Unterwalden)”*. In part one, the opening is proposed and answered as a variation in part 2. The structure is repeated in part 3 and 4 which Leuthold calls a ”two-bar answer”. The follow-up mark parts 5-8.

1.7.1 Yodel genres

There are several definitions of yodel regions in Switzerland, and there is no consensus reached about it. For this thesis, the definition of Leuthold (1981) is used which is briefly introduced next.

Yodel songs of Appenzell often contain a closing portamento that falls at the end. Additionally, the chorus applies often the so-called ”Gradhebe”, monotonously keeping the keynote. Other marks are upward leading triads and large seventh downwards with an ”I” vocalisation. From Appenzell Ausserrhoden the so-called ”Trüllerli” emerged. It is a quaverish decoration and variation of a fast waltz-like yodel. Toggenburg knows the ”Zungenschlagjodel”. A type of yodel where the tongue is used to create the sounds. It uses the higher registers more often than in Appenzell.

Muotathaler yodel is the easiest to distinguish from another yodel, even for non-specialists. The yodel contains both: often ecmelic notes and strong vocalisation on ”ä”, ”e”, ”i” and sometimes ”o”. The rhythm is not restricted to eight bar but is a more free and independent cadence. Unterwaldner yodel is slow and sedate. It makes use of the nature fa and uses simple repeating motifs in the melody. E.g. the pickup motif is reused in a second part but started with the dominant. The postscript is then

[5]https://pages.mtu.edu/ suits/scales.html, accessed 15.01.2018

the same as in the first part. Obwaldner yodel is even slower than in Unterwalden and is sung more deliberate.

Yodel in Bern is, in general, more cosy and relaxing. However, it is also famous for the "Schnetzler" which is a large interval that is sung clarinet-like downwards.

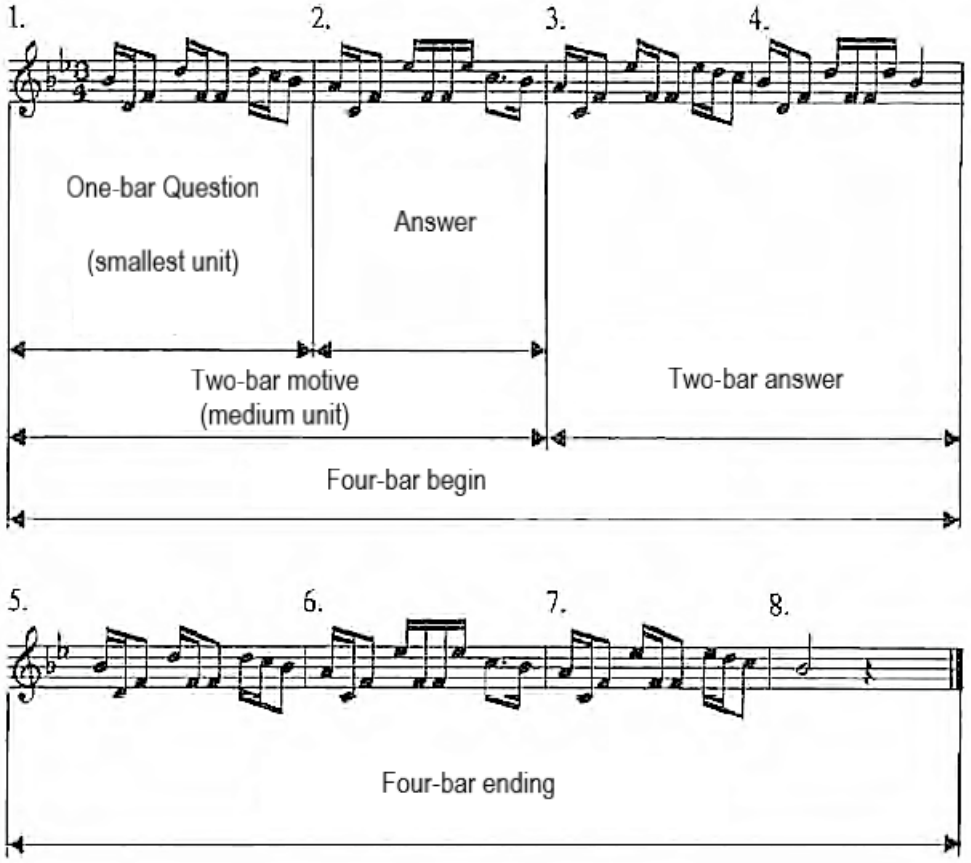


Figure 1.2: Question and answer phrases in yodelling. 1 and 2 show a simple question and answer part and 3 and 4 mark a repetition. The following parts 5 to 8 repeat the structure to end eight-bar structure((Leuthold, 1981, p.52)).

1.8 Acknowledgements

First, I would like to thank Prof. Marc Pouly who supported me throughout the master studies. He worked with me at the blackboard to understand proofs of the different approaches and did not back off if it was necessary to recapitulate them.

This thesis would have been impossible without the support of Andrea Kammermann who provided the whole data corpus and taught me about yodel theory. She always spared time to answer questions or explain music-theoretical concepts. Additionally, she was willing to assess the results. Without her, this thesis would not be of the same depth as it is. Many thanks to her team member Yannick Wey how showed me analysis techniques for music.

Third, I would like to thank Prof. Thomas Koller who supported me with his knowledge about deep learning. He took time to answer questions and discuss some of the approaches.

Many thanks to Nuria Richner who was willing to assess the samples based on her experience in music. Her input was priceless and led to a deeper understanding of the result.

I thank Annika Sonderegger who spared time for an interview about generative approaches and law. Special thanks belong to Pustulka Elzbieta. She showed me how a press review is done and provided me with many articles and information.

I would like to thank Tobias Baltensperger and Stefan Schnürle for proofreading. They worked through the formulas and pointed out weaknesses which improved the quality of the document tremendous. Special thanks to Rahel John for proofreading. Without her, the interview about law would have been impossible as she provided the contact with Annika Sonderegger.

Many thanks to the ABIZ research team of the Department Information Technology of the Lucerne University of Applied Sciences and Arts. Roland Christen supported me with docker, and Tim vor der Bück gave advise about several topics. Additionally, I would like to thank the many friends and families for proofreading and supporting me throughout the thesis.

Last but not least I want to thank my fiancée Flurina Hari who supported me in many ways and encouraged me throughout my studies.

Chapter 2

Related work

This chapter describes relevant related works. Section 2.1 contains a press review of the success of generative model for music synthesis. In Section 2.2, basic concepts are explained like neural networks and maximum likelihood estimation. Sections 2.3 and 2.4 introduce recurrent and convolution neural networks. Optimisation techniques are reviewed in Section 2.5. State of the art generative frameworks is discussed in Section 2.6 and approaches to music synthesis in 2.7.

2.1 Press Review

With the advances in deep learning with neural networks in the last 2 years, music synthesis has gained increased attention by media outlets. Table 2.1 points to some of the recently published articles.

THE VERGE, Vox Media	**Date:** 27.08.2017 Musician Taryn Southern on composing her new album entirely with AI *https://www.theverge.com/2017/8/27/16197196/taryn-southern-album-artificial-intelligence-interview*
BBC News, BBC	**Date:** 08.08.2017 Artificial music: The computers that create melodies *http://www.bbc.com/future/story/ 20140808-music-like-never-heard-before*
heise online, Heise Medien	**Date:** 14.07.2017 Künstliche Intelligenz: "Fake Music" auf dem Vormarsch *https://www.heise.de/newsticker/meldung/ Kuenstliche-Intelligenz-Fake-Music-auf-dem-Vormarsch-3772188.html*
20min, Tamedia AG	**Date:** 11.06.2017 In 11 Jahren schreibt eine Maschine einen Hit-Song *http://www.20min.ch/digital/news/story/ In-11-Jahren-schreibt-eine-Maschine-einen-Hit-Song-17641204*
Schweizer Radio und Fernsehen	**Date:** 13.04.2017 Eine Maschine meistert traditionelle Folk-Music *https://www.srf.ch/kultur/netzwelt/eine-maschine-meistert-traditionelle-folk-music*
Deutschlandfunk	**Date:** 19.10.2016 Musik aus der Maschine *http://www.deutschlandfunk.de/kuenstliche-intelligenz-musik-aus-der-maschine.807.de.html?dram:article_id=368956*
THE VERGE, Vox Media	**Date:** 26.09.2016 Researchers restore the first ever computer-generated music, made in Alan Turing's lab *https://www.theverge.com/2016/9/26/13058638/listen-to-first-ever-computer-generated-music*

WIRED.de, Condé Nast Verlag GmbH	**Date:** 27.07.2016 Aus Malerei wird Musik – per Algorithmus *https://www.wired.de/collection/tech/ aus-malerei-wird-musik-algorithmus*
Schweizer Radio und Fernsehen	**Date:** 30.03.2016 Mensch vs. Maschine: Wer empfiehlt den besseren Sound? *https://www.srf.ch/radio-srf-virus/musikplus/mensch-vs-maschine-wer-empfiehlt-den-besseren-sound*
BBC News, BBC	**Date:** 03.01.2013 Iamus: Is this the 21st century's answer to Mozart? *http://www.bbc.com/news/technology-20889644*

Table 2.1: Press review over the last two year to music synthesis.

2.2 Basic Concepts

In this section, some basic concepts used in this work are introduced. Section 2.2.1 contains an introduction to neural networks. In Section 2.2.2, the concept of the maximum likelihood estimation is presented, and Section 2.2.3 includes a description of the Kullback-Leibler divergence.

2.2.1 Introduction to Neural Networks

The beginning of neural networks marks an algorithm, proposed by McCulloch and Pitts (1943), where the authors used observations about the neurophysiology of the brain. In simplified terms, the brain can be described as a net of neurons where each neuron has a soma and an axon. The soma denotes the corpus of the neuron and the axon a nerve's extension for the connection with others. Synapses exist between the axon of one neuron and the soma of another. The neuron sends an impulse if its excitation exceeds a certain threshold. Based on these findings, McCulloch and Pitts proposed a neural network algorithm using a threshold logic. Frank Rosenblatt extended this idea to the Perceptron formulation (Rosenblatt, 1958). It is one of many neuron representations that have been used within neural networks so far. Because it explains the essential functions that are used by most types, it will be introduced next.

The structure of a Perceptron is shown in Figure 2.1. Its activation value is calculated as follows:

$$a_i = \quad g(z_i), \text{ with } z_i = \sum_{j=1}^{n} \Theta_{ij} \cdot x_j + b_i(+1), \tag{2.1}$$

where i refers to the i-th Perceptron and n to the input size. The non-linear activation function $g(z_i)$ is tractable and differentiable for any z_i. $x_1, ..., x_n$ are the inputs, $\Theta = (\Theta_{i1}, ..., \Theta_{in})$ are the weights that are multiplied with the inputs, and the node $+1$ is a bias unit with constant value 1 and b_i denotes the bias weight. Therefore, the term $(+1) \cdot b_i$ is often shortened to b_i. z_i denotes the inner activation of node i which is used as input to $g(z_i)$.

A neural network consists of an input layer 0, $L-1$ hidden layers and an output layer. A layer contains several neurons that do not have to be of the same type. For simplicity, consider the neurons to be Perceptrons. Each neuron on layer l is connected to neurons of the layer $l-1$. A schematic representation of a simple network is given in Figure 2.2. It has two inputs in the input layer 0, x_1 and x_2, two nodes $a_1^{(1)}$ and $a_2^{(1)}$ in the hidden layer 1 and one node in the output layer. $a_i^{(l)}$ denotes the activation value of node i in layer l. $\Theta_{ij}^{(l)} \in \mathbb{R}$ stands for the weight on the connection between node j of layer l and node i of layer $l+1$ (see Figure 2.3). The number of nodes in layer l is given by $n^{(l)}$. $f_\Theta(x_1, ..., x_{n^{(0)}})$ denotes the vector of output activations of the network with weights Θ and inputs $x_1, ..., x_{n^{(0)}}$ of input size $n^{(0)}$.

To assess the i-th output activation $f_\Theta(x_1, ..., x_{n^{(0)}})_i$ the forward propagation has to be applied, which means to calculate all activation values of all neurons. By reformulating Equation 2.1, the inner activation of node i in layer l can be derived as

$$z_i^{(l)} \quad = \quad \sum_{j=1}^{n^{(l-1)}} \Theta_{ij}^{(l)} \cdot a_j^{(l-1)} + b_i^{(l)}, \tag{2.2}$$

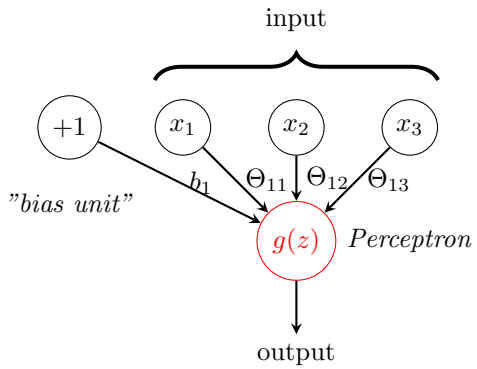


Figure 2.1: Schematic representation of a neuron of type Perceptron. x_1,x_2 and x_3 mark the components of the input, the +1-node is a bias unit with constant value 1 that is multiplied with the bias weight b_1, and $g(z)$ denotes a non-linear activation function of the neuron with the weights $\Theta = (\Theta_{11}, \Theta_{12}, \Theta_{13})$.

with $1 \leq l \leq L$ and $a_j^{(0)} = x_j$. The activation value is given by

$$a_i^{(l)} \quad = \quad g(z_i^{(l)}). \tag{2.3}$$

Finally, the i-th model's output is defined as

$$f_\Theta(x_1, ..., x_{n^{(0)}})_i \quad = \quad a_i^{(L)}. \tag{2.4}$$

Note that Equation 2.2 assumes that layer $l-1$ and layer l are fully connected. A partial connection is achieved by setting $\Theta_{ij}^{(l)} = 0$.

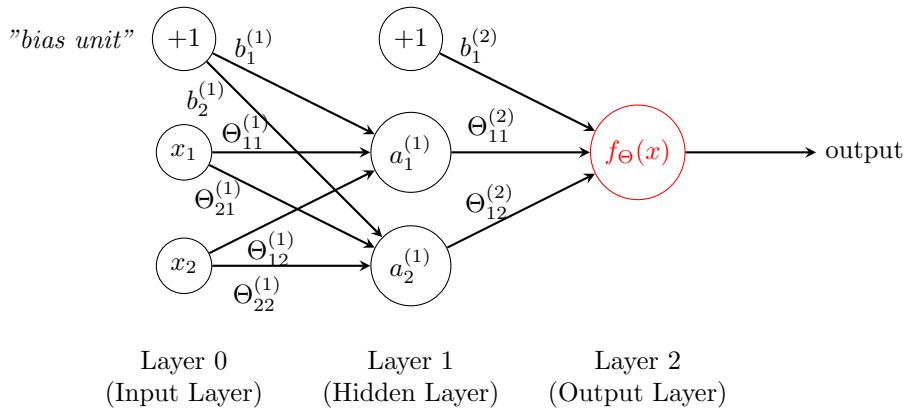


Figure 2.2: A simple network with inputs x_1 and x_2, bias units +1, and activations of node 1 and 2 of hidden layer 1, $a_1^{(1)}, a_2^{(1)}$. $\Theta_{ij}^{(l)}$ are the weights on the connection between one node and another. $b_i^{(l)}$ denotes the bias weights of layer l. $f_\Theta(x)$ marks the activation function of the output.

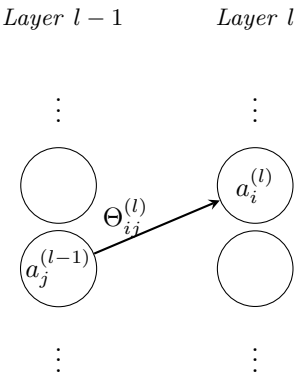


Figure 2.3: The weight $\Theta_{ij}^{(l)}$ is applied to the output of node j in the layer $l-1$ and acts as input to the node i in layer l.

For simplification, the vector notation is introduced. The forward propagation is written as follows:

$$\begin{aligned} a^{(0)} &= x, \\ z^{(l)} &= \Theta^{(l)} \cdot a^{(l-1)} + b^{(l)} \quad l = 1, ..., L. \end{aligned} \tag{2.5}$$

x is a vector of inputs $[x_1, ..., x_{n^{(0)}}]^T$ with T denoting the transpose, $\Theta^{(l)}$ is a matrix of weights with dimensions $n^{(l)} \times n^{(l-1)}$ and l denoting the layer, and $b^{(l)} \in \mathbb{R}^{n^{(l)}}$. The activation value and model output are given as

$$a^{(l)} = g(z^{(l)}), \tag{2.6}$$

$$f_\Theta(x) = a^{(L)}. \tag{2.7}$$

The number of nodes in each layer and the activation function of each node can vary. Moreover, other neuron types include additional input parameters, e.g. the activation value of the time step $t-1$ (recurrent node). This thesis refers to the whole model as $f_\Theta(x)$ where Θ represents the weights of all layers, the model's parameters.

In following, we will give a numerical example of a neural network using the sigmoid function as activation function defined as

$$g(z) = \frac{1}{1+e^{-z}}, \tag{2.8}$$

for $z \in \mathbb{R}^n$.

Example 2.1. *Consider the architecture given in Figure 2.2. We will calculate the forward propagation for this network and set the weights in such a way that it will mimic the function*

$$f(x_1, x_2) = x_1 \ XNOR \ x_2 \tag{2.9}$$

The truth table is given in Table 2.2a. First, we use Equations 2.2 and 2.3 to write the explicit Equations for layer 1*:*

$$a_1^{(1)} = g\left(\Theta_{11}^{(1)} \cdot x_1 + \Theta_{12}^{(1)} \cdot x_2 + b_1^{(1)}\right), \tag{2.10}$$

$$a_2^{(1)} = g\left(\Theta_{21}^{(1)} \cdot x_1 + \Theta_{22}^{(1)} \cdot x_2 + b_2^{(1)}\right), \tag{2.11}$$

and the model output

$$f_\Theta(x) = g\left(\Theta_{11}^{(2)} \cdot a_1^{(1)} + \Theta_{12}^{(2)} \cdot a_2^{(1)} + b_1^{(2)}\right). \tag{2.12}$$

As an example to reproduce the XNOR-function, we set the weights of the network as follows:

$$\begin{aligned} &\textit{Layer 1} && \Theta^{(1)} = \begin{bmatrix} \Theta_{11} & \Theta_{12} \\ \Theta_{21} & \Theta_{22} \end{bmatrix} = \begin{bmatrix} 20 & 20 \\ -20 & -20 \end{bmatrix} \\ & && b^{(1)} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} -30 \\ 10 \end{bmatrix} \\ &\textit{Layer 2} && \Theta^{(2)} = \begin{bmatrix} \Theta_{11} & \Theta_{12} \end{bmatrix} = \begin{bmatrix} 20 & 20 \end{bmatrix} \\ & && b^{(2)} = \begin{bmatrix} b_1 \end{bmatrix} = \begin{bmatrix} -10 \end{bmatrix} \end{aligned} \tag{2.13}$$

We turn our attention to the first case of Table 2.2a and set $x_1 = x_2 = 0$. *Applied to Equations 2.10 and 2.11, we find the activation values of layer* 1 *to be*

$$\begin{aligned} a_1^{(1)} &= g(20 \cdot 0 + 20 \cdot 0 - 30) = g(-30) \approx 0, \quad \textit{and} \\ a_2^{(1)} &= g(-20 \cdot 0 + (-20) \cdot 0 + 10) = g(10) \approx 1, \end{aligned}$$

using the sigmoid function $g(z)$ *given by Equation 2.8 (see Figure 2.4a for visual verification). We insert* $a_1^{(1)}$ *and* $a_2^{(1)}$ *into Equation 2.12 and find*

$$f_\Theta(x) = g(20 \cdot 1 + 20 \cdot 0 + (-10)) = g(10) \approx 1,$$

which is what we expected from Table 2.2a. For the other cases the activation values $a_1^{(1)}, a_2^{(1)}$ *and the corresponding output of the network* $f_\Theta(x)$ *are provided in Table 2.2b.*

We have shown how the forward path of the network Figure 2.2 is calculated and that it is able to reproduce the XNOR-function with the exemplary parametrisation. Note that the weights $\Theta^{(1)}$ *of the input values* x_1 *and* x_2 *change the steepness of the activation function (see Figure 2.4a). Because the bias unit is a constant value, the weight* $b_i^{(l)}$ *will shift the activation function to the left or right as shown in figure 2.4b and thereby adjust the range of the activation.*

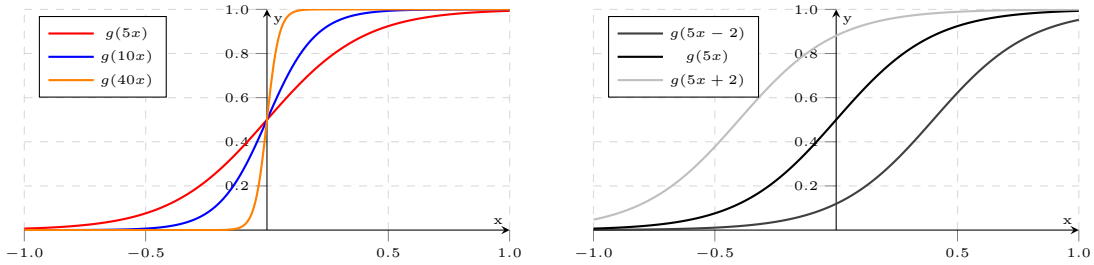
x_1	x_2	x_1 *XNOR* x_2
0	0	1
0	1	0
1	0	0
1	1	1

(a) Truth table for function x_1 *XNOR* x_2.

x_1	x_2	$a_1^{(1)}$	$a_2^{(1)}$	$f_\Theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

(b) Given the architecture Figure 2.2 and the weights as defined in Equation 2.13, the activations are calculated as given in this table.

Table 2.2: (a) shows the truth table of the function x_1 *XNOR* x_2 and (b) a possible solution for the architecture in Figure 2.2.



(a) Effects of the multiplier of x on the sigmoid function g.

(b) Effects of the bias unit on the sigmoid function defined in Equation 2.8.

Figure 2.4: The sigmoid function $g(z) = \frac{1}{1+e^{-z}}$ given for different inputs.

Loss function choice

Given a neural network model comprising Equations 2.6 and 2.7, it is necessary to calibrate it to fit a certain task. For instance, the parametrization of Example 2.1 reproduces XNOR, another parametrisation of the same model, however, would reproduce a different function. In order to find a parametrisation of the model weights, which reproduce the desired function as closely as possible, a quantification of the difference between the actual and desired output is necessary. The difference is measured by the loss function, which has to be continuous and differentiable in order to be compatible with back-propagation, a concept introduced later. Most common, loss functions are of type classification or regression.

For a classification problem, the loss function can be set to the cross-entropy function. Consider a K-multi-class classifier with $f_\Theta(x) \in \mathbb{R}^K$ where K is the number of classes to look for, the cross-entropy loss function over m samples is defined by

$$\mathcal{L} = -\frac{1}{m}\left[\sum_{j=1}^{m}\sum_{i=1}^{n^{(L)}} y_i^{(j)} \cdot \log\left(f_\Theta(x^{(j)})_i\right) + (1 - y_i^{(j)}) \log\left(1 - f_\Theta(x^{(j)})_i\right)\right], \tag{2.14}$$

where $n^{(L)}$ is the output of layer L, $y_i^{(j)}$ is the real value of the i-th class of sample j and usually given by zero or one, and $x^{(j)}$ is the input data. Note that the size of the layer L must be $n^{(L)} = K$.

To penalise large weights and avoid over-fitting, a regularisation term E is often added to $\mathcal{L}$:

$$E = \lambda\left[\frac{1}{2}\sum_{l=1}^{L}\sum_{i=1}^{n^{(l)}}\sum_{j=1}^{n^{(l-1)}} (\Theta_{ij}^{(l)})^2\right], \tag{2.15}$$

where $n^{(l)} \times n^{(l-1)}$ is the dimensions of weight matrix $\Theta^{(l)}$ of layer l, λ denotes the regularisation parameter.

If the model's task is instead to perform a regression, then the mean squared error (MSE) is an appropriate loss function. It is given by

$$\mathcal{L} = \frac{1}{2m}\sum_{k=1}^{m}(f_\Theta(x^{(k)}) - y^{(k)})^2. \tag{2.16}$$

Note that Equation 2.16 assumes a univariate regression.

Model fitting

Once an appropriate loss function has been defined, its weights are tuned such that the loss function value is minimised for given inputs $x = \bar{x}$ and desired outputs $y = \bar{y}$:

$$\min_{\Theta} \quad \mathcal{L}(x, y, \Theta) \tag{2.17}$$

(with $\bar{x}$ and $\bar{y}$ indicating that the function's variables x and the outputs y are fixed to certain values). In the following, gradient descent is introduced, which is the basic concept of most optimisation strategies used for neural networks. The method has several drawbacks, for instance, it can easily get stuck in local minima, and the gradients are difficult to calculate for generic functions. The difficulty of calculating the gradient can be overcome by choosing appropriate loss functions, as introduced in Section 2.2.1, and activation functions, like the sigmoid, which takes an easy form and thus a steepest descent algorithm such as gradient descent can be executed extremely fast.

In order to solve the optimisation problem in Equation 2.17 efficiently, it is an advantage to know about the characteristics of the surface of $\mathcal{L}$. Figure 2.5 provides an example. The surface has its global optimum at the centre; in addition, multiple local optima exist. In all local and the global minimum all partial derivatives of the loss function $\mathcal{L}$ equal zero, i.e.,

$$f'_{x_i}(x) = \frac{\partial f(x)}{\partial x_i} \quad = \quad 0 \quad \forall i \in n,$$

where $f : \mathbb{R}^n \to \mathbb{R}$.

The gradient of $f(x_1, x_2, ..., x_n)$ is defined as

$$\nabla f = \left[f'_{x_1}, f'_{x_2}, f'_{x_3}, \ldots, f'_{x_n}\right]. \tag{2.18}$$

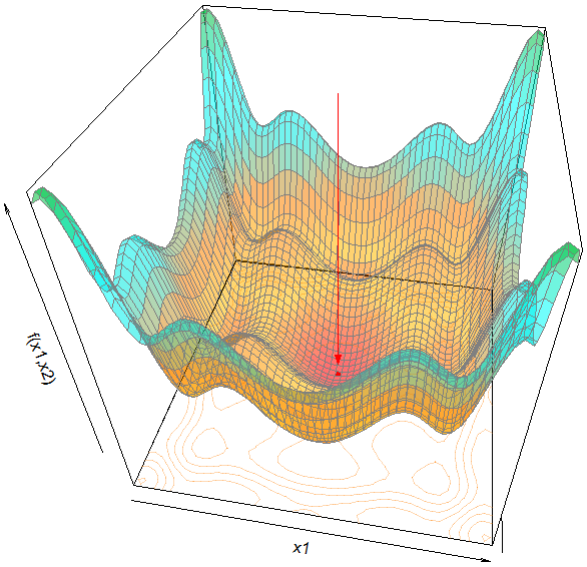


Figure 2.5: The function $f : \mathbb{R}^2 \mapsto \mathbb{R}$ $f(x_1, x_2) = \frac{(x_1^2+x_2^2)}{2} + \sin(x_1^2) + \sin(x_2^2)$. The red arrow marks the global optimum (minimum) of the function but there are multiple local optima around the centre.

The gradient points in the direction of the steepest increase of a function. For the given optimisation problem (2.17) it is not sufficient to solve $\nabla f(x) = 0$ because multiple local optima exist. Furthermore, the computational time required to solve $\nabla f = 0$ will be too large for practical applications. Instead, the steepest descent method is applied: a starting point $x^{(0)}$ is chosen, and a sequence of points $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$ for which the function value decreases monotonically at these positions. Given a point $x^{(i)}$, we find new points with smaller function values in direction of $r \in \mathbb{R}^n$ where

$$r \cdot (-\nabla f(x^{(i)}))^T > 0 \tag{2.19}$$

with T being the transpose operation. In other words, the scalar product of the anti-gradient (steepest descent) at point $x^{(i)}$ and the directional vector r is positive, the two vectors enclose an acute angle. $x^{(i+1)}$ can be found by solving the following optimisation problem:

$$\min_{\alpha} \quad h(\alpha) = f\big(x^{(i)} - \alpha \cdot \nabla f(x^{(i)})\big) \tag{2.20}$$

(see Figure 2.6). Based on the previous position $x^{(i)}$ a new approximation $x^{(i+1)}$ to the solution of the

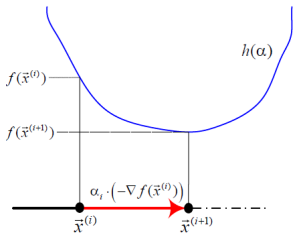


Figure 2.6: An α step in direction of the steepest descent on the function $f(x)$ (Vogel (2013)).

optimisation problem is found with

$$x^{(i+1)} = x^{(i)} - \alpha_i \cdot \nabla f(x^{(i)}) \tag{2.21}$$

(see figure 2.6).

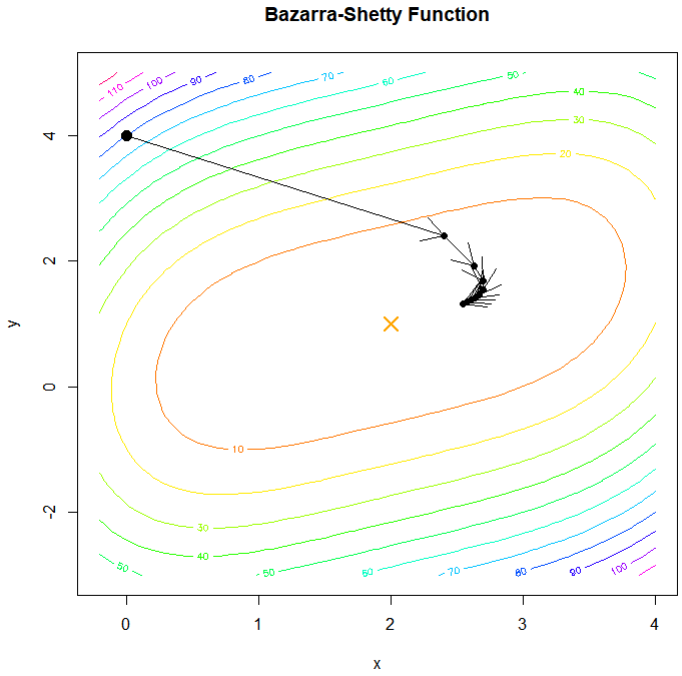


Figure 2.7: A contour plot of the function $f(x_1, x_2) = (x_1 - 2)^4 + (x_1 - 2x_2)^2$ with 10 steps of gradient descent applied with constant step size $\alpha = 0.05$.

Example 2.2. *We want to minimise the function*

$$f(x_1, x_2) = (x_1 - 2)^4 + (x_1 - 2x_2)^2,$$

which is called Bazarra-Shetty function (Vogel (2013)). The function has a global minimum at $[2, 1]$. *Let the starting point for gradient descent be* $x^{(0)} = [0, 4]$. *We find the gradient at this position to be*

$$\nabla f(x) = \begin{bmatrix} 4(x_1 - 2)^3 + 2(x_1 - 2x_2) & -4(x_1 - 2x_2) \end{bmatrix} \implies \nabla f(0, 4) = \begin{bmatrix} -48 & 32 \end{bmatrix}.$$

For simplification, the step size is fixed to the value of $\alpha = 0.05$. *We find the next point* $x^{(1)}$ *with*

$$x^{(1)} = x^{(0)} - \alpha \cdot \nabla f(x^{(0)}) = \begin{bmatrix} 0 & 4 \end{bmatrix} - 0.05 \begin{bmatrix} -48 & 32 \end{bmatrix} = \begin{bmatrix} 2.4 & 2.4 \end{bmatrix}$$

Figure 2.7 shows ten steps of gradient descent. Note that it is increasingly hard to improve as the gradient becomes smaller (the plane gets more and more shallow). More sophisticated optimiser are described in Section 2.5. ⊖

Back-propagation

Back-propagation algorithms are recursive procedures that back propagate the error from the output layer all the way back to the first layer. The concept is shown in Figure 2.8. Note that in recurrent networks the back-propagation is not only applied over the layers but also through time, which leads to the name back-propagation through time (BPTT). These algorithms use the concept of gradient descent to minimise the loss in an iterative process until convergence.

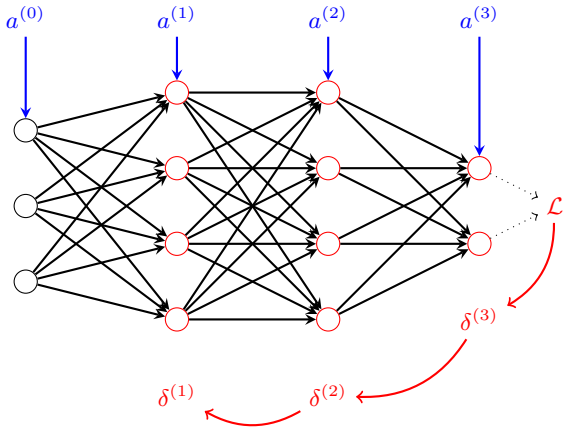


Figure 2.8: Back-propagation of the error for each layer.

To apply back-propagation any loss-function $\mathcal{L}$ has to be globally continuous and differentiable. We want to minimise the error, which is the difference between input and desired output, by changing the weights Θ,

$$\min_{\Theta} \quad \mathcal{L}.$$

According to gradient descent (Section 2.2.1) we can determine how to change the weights of layer l by calculating the gradients for each weight $\Theta_{ji}^{(l)}$ over the loss and follow the steepest descent. We want to show that the gradient for the weight $\Theta_{ji}^{(l)}$ is given by

$$\nabla \mathcal{L}_{\Theta_{ji}^{(l)}} = \frac{\partial \mathcal{L}}{\partial \Theta_{ji}^{(l)}} = \delta_j^{(l)} \cdot a_i^{(l-1)} \tag{2.22}$$

with $\delta_j^{(l)}$ being the error of node j in layer l.

The gradients $\nabla \mathcal{L}_{\Theta^{(l)}}$ depend on the architecture of the networks as well as on the activation function of each neuron. For the following section we restrict the activation function to the sigmoid. Given its Equation in 2.8, we find the derivative to be

$$g'(z) = g(z)(1 - g(z)). \tag{2.23}$$

The proof is found by chain rule given in appendix A.2.1. Furthermore, the error of the inner activation at output node j is given by

$$\delta_j^{(L)} = \frac{\partial \mathcal{L}}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} = a_j^{(L)} - y_j \tag{2.24}$$

The proof for the cost functions (2.14 and 2.16) can be found in Appendices A.2.2 and A.2.3.

From Figure 2.9 we see that the chain rule has to be applied twice in a right-to-left order to get the weight's gradient.

$$\frac{\partial \mathcal{L}}{\partial \Theta_{ji}^{(l)}} = \frac{\partial \mathcal{L}}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial \Theta_{ji}^{(l)}} \tag{2.25}$$

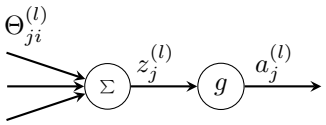


Figure 2.9: A representation of the computation graph of a neuron. $\Theta_{ji}^{(l)}$ denotes the weight factor on the connection, summed up we get the inner activation $z_j^{(l)}$ and after the application of a non-linear function it outputs the activation value $a_j^{(l)}$.

By calculating each partial derivative for the two cases $l = L$ and $l < L$ we find the gradient of the weight

$$\begin{aligned} \nabla \mathcal{L}_{\Theta_{ji}^{(l)}} &= \delta_j^{(l)} \cdot a_i^{(l-1)}, \\ \text{where } \delta_j^{(l)} &= \begin{cases} (a_j^{(L)} - y_j) & \text{if } l = L, \\ \sum_{k=1}^{n^{(l+1)}} \delta_k^{(l+1)} \cdot \Theta_{kj}^{(l+1)} \cdot a_j^{(l)} \cdot (1 - a_j^{(l)}) & \text{if } 1 \leq l < L. \end{cases} \end{aligned} \tag{2.26}$$

The derivation is given in full in Appendix A.2.4.

2.2.2 Maximum likelihood estimation

The maximum likelihood estimation is used to find a model with parameter Θ that maximises the likelihood concerning a given data space. The maximum likelihood estimation is given by

$$\max_{\Theta} \log p(\mathcal{X}) = \max_{\Theta} \prod_{i=1}^{m} p_\Theta(x^{(i)}), \tag{2.27}$$

where Θ denotes the model's parameters, and m is the number of training samples $x^{(i)}$ of a data set. Note that Θ is utilised in $p_\Theta(x^{(i)})$ to refer to the model's parameters which have to be adapted.

Generative frameworks aim at maximising the likelihood $p(x^{(i)})$. This objective is called the maximum likelihood estimation and reflects the intuition described by Doersch (2016): "(. . .) that if the model is likely to produce training set samples, then it is also likely to produce similar samples, and unlikely to produce dissimilar ones" (p.3).

In practical applications, the multiplication of small values like probabilities are prone to floating point errors that often impede a successful learning. For this reason the log-likelihood is used which can be written as a sum of log probabilities. The optimisation criteria can be written as

$$\begin{aligned} \max_{\Theta} p(\mathcal{X}) &= \log \prod_{i=1}^{m} p_\Theta(x^{(i)}) \\ &= -\min_{\Theta} \sum_{i=1}^{m} \log p_\Theta(x^{(i)}) \end{aligned} \tag{2.28}$$

As an example, the cross-entropy loss-function (2.14) optimises the log-likelihood estimate for a K-classifier. Another example would be a model that learns to output samples $x^{(i)}$ which sounds like yodel music. The maximum likelihood estimation is therefore applied to optimise the model's parameters Θ, so it is likely to output the probability $p(\mathcal{X}) = 1$, which means that $\mathcal{X}$ is following the distribution of yodel songs.

2.2.3 Kullback-Leibler divergence

The Kullback-Leibler (KL) divergence was proposed by Kullback and Leibler (1951) and is a quantification of how much one probability distribution diverges from another probability distribution. A KL divergence of 0 is an indication that two distributions behave similar whereas a KL divergence of 1 means that given the first distribution no inference can be done to the second one.

Given a probability density function $p(z)$ and $q(z)$ the KL divergence is defined by

$$KL\big(q(z)\|p(z)\big) = \sum_{z} q(z) \log\left(\frac{q(z)}{p(z)}\right) \tag{2.29}$$

if distribution P and Q are both discrete and defined over the same space.

As an example, imagine a Gaussian distribution $\mathcal{N}(0, 0.5)$ that is approximated by an uniform distribution $uniform(-1, 1)$. Figure 2.10 shows both probability density function. It is obvious that the uniform distribution yields, for instance, higher probabilities at the edges $x = -1$ and $x = 1$. Therefore, the KL divergence of the uniform to the normal distribution will not output 0 as it does not behave similar. On the other hand, it outputs neither 1 because some probabilities around -0.5 and 0.5 are similar and therefore, some inference can be done.

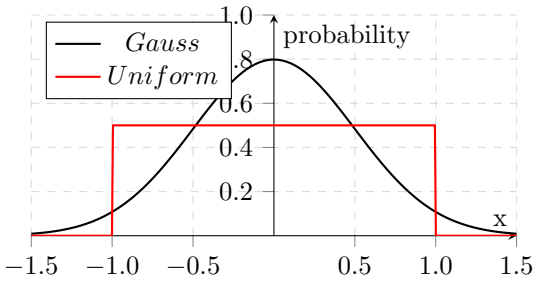


Figure 2.10: An uniform distribution (-1,1) and a Gaussian distribution $\mathcal{N}(0, 0.5)$.

2.3 Recurrent Neural Networks

Recurrent neural networks (RNN) consist of special neural layer types called the recurrent layers. The most important aspect of every recurrent network lies in a cyclic connection from the output of a node to its input, i.e. the previously calculated value. These previously calculated values are lined up on a time dimension. Time step $t+1$ takes t previously calculated activations into account (see Figure 2.11). This setup makes RNN particularly suitable to be trained on samples in form of sequences, such as texts, continuous data, or time series, since it is assumed that a data point depends on its predecessor (e.g. if a word is plural or singular). Because it is impractical to work with infinite time lines, networks are used that can process a fixed amount of time steps t. Additionally, these recurrent networks can be expressed as a feed forward-network visualised in Figure 2.12. A single recurrent layer is replicated t times so each input represents one time step. The same way the input is transformed, thereby defining the order of output values. This transformation is referred to as "unrolling" the RNN through time.

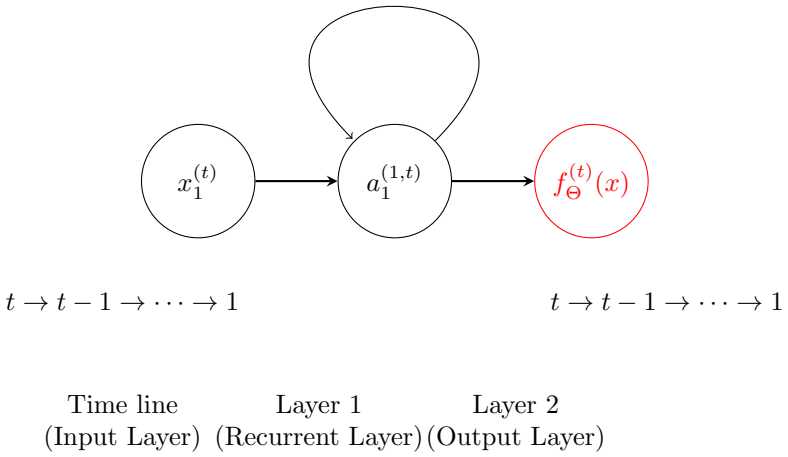


Figure 2.11: A recurrent network with one hidden layer and one recurrent node. Note that the result is influenced by the previous $t-1$ time steps.

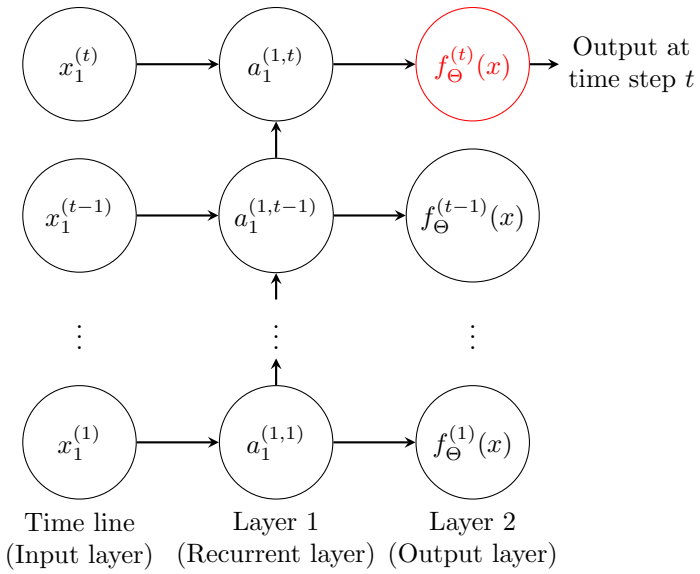


Figure 2.12: An unrolled recurrent network for t time steps with one hidden layer containing one recurrent node.

A simple recurrent node (SRN) computes its inner activation as

$$z^{(l,t)} = \Theta^{(l)}_{(R)} \odot z^{(l,t-1)} + \Theta^{(l)} \cdot a^{(l-1,t)} + b^{(l)} \quad l \in (1, \dots L), \tag{2.30}$$

where $\Theta^{(l)}_{(R)}$ denotes the weights on the recurrent inputs and $\odot$ is an element-wise multiplication. Note that $z^{(l,t-1)}$ denotes the inner activation at time step $t-1$. The activation value is given by

$$a^{(l,t)} = g(z^{(l,t)}). \tag{2.31}$$

2.3.1 Gated recurrent unit

The gated recurrent unit (GRU) was proposed by Cho et al. (2014) as part of a recurrent neural network encoder-decoder for the task of machine translation. The GRU includes a reset gate and an update gate to control what each hidden unit remembers. They reached a bilingual evaluation understudy (BLEU) score of 33.87 compared to their baseline, consisting of a statistical machine translation system, which reached 33.30. The BLEU score is a metric for evaluating a generated sentence to a reference sentence. However, Chung et al. (2014) evaluated the performance of a GRU against a long short-term memory (LSTM, Hochreiter and Schmidhuber (1997)) and reported mixed results. Next, the GRU is explained in greater detail.

Often, the simplicity of GRUs over LSTMs is mentioned as advantage. While LSTM were extended over time by numerous kind of connections and gates, the GRU consists of only four components: the activation value, a hidden state and two gates. Figure 2.13 illustrates the GRU schematically. First, the hidden state $h^{(t)}$, with t referring to time step, is calculated. If the reset gate $r^{(t)}$ is open then $h^{(t)}$ is calculated from the input $x^{(t)}$ and the old activation value $a^{(t-1)}$. Otherwise, only $x^{(t)}$ is used. Second, the the update gate $q^{(t)}$ decides whether the activation value $a^{(t)}$ is to be updated with $h^{(t)}$ (the gate is open) or if the old activation value $a^{(t-1)}$ is used. Third, the $a^{(t)}$ is returned as output.

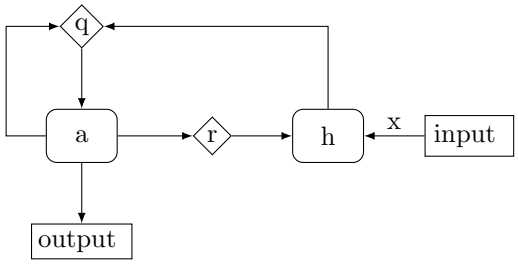


Figure 2.13: An illustration of GRU proposed by Cho et al. (2014). The diamond shaped nodes mark gates and the others states. If the reset gate r is open the hidden state h is updated with x and the previous activation value a. The result is delivered to the update gate q. If it is open then the activation value is calculated from the hidden state h. If it is closed then the activation value is updated with its previous value. The hidden state h is ignored.

Formally, the activation of the j-th hidden unit at time step t is denoted by $a^{(t)}_j$. First, the reset gate $r^{(t)}_j$ is computed by

$$r^{(t)}_j = g\Big(\big(\Theta^{(t)}_{(r)} \cdot x^{(t)}\big)_j + \big(U^{(t)}_{(r)} \cdot h^{(t-1)}\big)_j \Big), \tag{2.32}$$

where g is the sigmoid function, $x^{(t)}$ is the input value and $U^{(r)}_{(t)}$ is an additional weight matrix. The update gate $q^{(t)}_j$ is given by

$$q^{(t)}_j = g\Big(\big(\Theta^{(t)}_{(q)} \cdot x^{(t)}\big)_j + \big(U^{(t)}_{(q)} \cdot h^{(t-1)}\big)_j \Big). \tag{2.33}$$

The hidden state $h^{(t)}_j$ is computed by

$$h^{(t)}_j = \phi\Big(\big(\Theta^{(t)}_{(h)} \cdot x^{(t)}\big)_j + \big(U^{(t)}_{(h)}(r^{(t)} \odot h^{(t-1)})\big)_j \Big), \tag{2.34}$$

where ϕ denotes the hyperbolic tangent (tanh) activation function given by

$$\phi(x) = 2 \cdot g(2 \cdot x) - 1. \tag{2.35}$$

Figure 2.14 visualises the tanh and the sigmoid function for comparison. Note that the tanh function shifts the input towards range $[-1, 1]$ whereas the sigmoid function results in range $[0, 1]$. Finally, the activation value $a_j^{(t)}$ of the unit j is given by

$$a_j^{(t)} = q_j^{(t)} \cdot a_j^{(t-1)} + (1 - q_j^{(t)}) \cdot h_j^{(t)} \tag{2.36}$$

Therefore, the activation value does not take into account the input value $x^{(t)}$ directly. Instead, the input is used to steer the gates and to update the hidden state.

For back-propagation, RNN are more complex to handle as the time step t has to be considered. The weights $\Theta^{(0)}_{(r,q,h)j}$ and $U^{(0)}_{(r,q,h)j}$ of node j at time step $t = 0$ depend on the error of the upper layer $l+1$ as well as the errors of the node "through time" (time steps $1, 2, ..., t$). The gradients for the GRU can be derived similarly as the gradients for Θ in Section 2.2.1, but are not shown here as they have little relevance for this thesis.

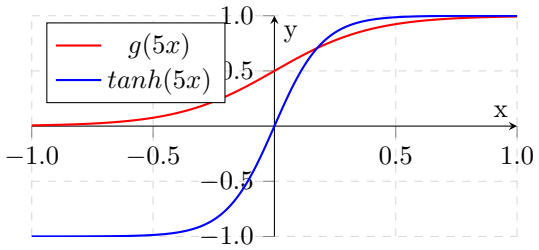


Figure 2.14: A comparison of the tanh (blue) and the sigmoid (red) function. Note that the tanh ranges from $[-1, 1]$ where as the sigmoid is in $[0, 1]$.

2.3.2 Related techniques

Yao et al. (2015) propose a depth-gated recurrent neural network which is based on the LSTM and additionally connects the cell-state of the different layers. Their model performed better than a RNN consisting of LSTM-cells or GRU-cells on the task of machine translation, and reached a BLEU score of 32.19 compared to LSTM (31.99) and GRU (30.72).

Greff et al. (2015) tested the LSTM cells in different variants. The influence of the different components on the result was evaluated with an analysis of variance. They found the forget gate and the output activation to be of significance, whereas all other added components to the basic LSTM (introduced by Hochreiter and Schmidhuber (1997)) were not statistically relevant.

Wang et al. (2016) propose an architecture consisting of a RNN and convolutional neural network (CNN) component. Both networks are combined by a projection layer which serves as input for the prediction layer. On the task of the NUS-Wide dataset (images from Flickr on which the k top tags are predicted) the performance was improved by about 4 tags mean average precision compared to the stand-alone CNN.

Recurrent Highway Networks (RHN) were proposed by Zilly et al. (2016) and are an advancement on the LSTM unit since they tackle the vanishing and exploding gradient behaviour. Like LSTM in its basic formulation they use forget, carry and transform gates, which are then connected with their upper layers at the specific time step. This provides a more versatile setup for dynamically remembering, forgetting and transforming information compared to standard RNNs. If the network contains only one layer it deduces to a LSTM.

2.4 Convolution Neural Networks

Convolution neural networks (CNN) were proposed by Le Cun et al. (1989) for the task of hand written digit recognition. They became popular as Krizhevsky et al. (2012) proposed a deep CNN which beat state-of-the-art performance in the ImageNet image classification challenge. They are widely used in computer graphics and had recently become popular for audio and speech synthesis (van den Oord et al. (2016a)). By definition a convolution of two functions f and g over an infinite range is given by

$$(f \star g)(c) = \int_{-\infty}^{\infty} f(a) \cdot g(c-a) \mathrm{d}a, \tag{2.37}$$

where $(f \star g)(c)$ denotes the convolution of f and g at position c. For convolutional neural networks c is a discrete number and therefore the convolution can be reformulated as a sum that is defined as

$$(f \star g)(c) = \sum_{a=-\infty}^{\infty} f(a) \cdot g(c-a). \tag{2.38}$$

For an intuition of convolution, consider playing dice. Let $f(a)$ be the probability of the first die showing a 3. Let $g(b)$ be the probability of the second die being a 1. Note that f and g are both probability density function of arbitrary shapes. The probability of the first dice is 1 and the second is 3 is given by $f(a) \cdot g(b)$. If we want to know the total probability of both dice together showing a value of $c = a + b = 4$ we have to sum up over the probabilities of each event with

$$\sum_{a+b=c} f(a) \cdot g(b).$$

Substituting $b = c - a$ and comparing with Equation 2.38 we find that we perform a convolution.

For CNN, convolutions are performed over the inputs with some filter weights. Figure 2.15 represents a convolution with filter size $k = 2$. In terms of Equation 2.38, $f(a)$ denotes the inputs $a_i^{(l-1)}$ and $g(c-a)$ the filters weight $\Theta_k^{(l)}$.

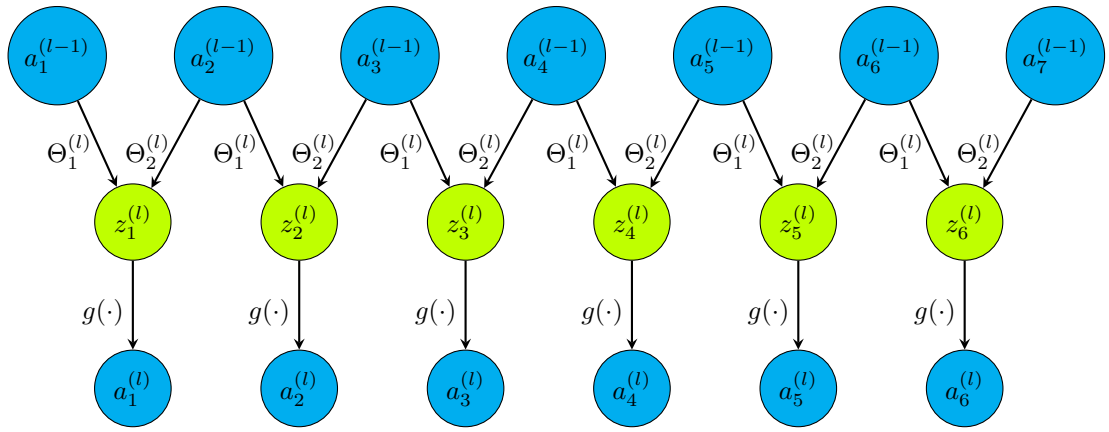


Figure 2.15: A one-dimensional convolution layer with inputs $a_i^{(l-1)}$, inner activations $z_i^{(l-1)}$ and outputs $a_i^{(l)}$. $g(\cdot)$ denotes a non-linear function. Note that the layer consists of only two weights and that each neuron has only two inputs. Therefore, this is a convolution with filter size $k = 2$.

While a neuron of a normal neural network layer (see Section 2.2.1) depends on every neuron of the previous layer, convolutional neurons only use k inputs. The inner activation is given by

$$z_i^{(l)} = \Theta_1^{(l)} \cdot a_i^{(l-1)} + \Theta_2^{(l)} \cdot a_{i+1}^{(l-1)} + ... + \Theta_k^{(l)} \cdot a_{i+k-1}^{(l-1)}.$$

Note that the calculations of $z_i^{(l)}$ involves only k weights. E.g. often $k = 9$ is used for a convolutional layer even though, the previous layer has $n > 1000$ nodes. Because k weights may not be sufficient for learning channels are added. Instead of having a normal neural network layer with n weights convolutions uses k weights for each of D channels. This means that every channel performs on the

same layer a convolution on its own. The next layer uses these output channels as inputs for its convolution. This way D distinct features are learnt and the expressiveness of each layer is scalable in D. More generally, let the input be a matrix of size $n^l \times D^{(l)}$. The inner activation of a 1-dimensional convolution over the channels $D^{(l)}$ is defined as

$$z_{io}^{(l)} = \sum_{c=1}^{D^{(l-1)}} \sum_{j=1}^{k^{(l)}} \Theta_{jco}^{(l)} \cdot a_{(i+j-1)c}^{(l-1)} + b_{jo}^{(l)}, \tag{2.39}$$

where $i \in [1, \ldots, M^{(l)}]$ and the feature maps $M^{(l)} = n^{(l-1)} - k^{(l)} + 1$, $o \in [1, \ldots, D^{(l)}]$ is the output channel, $\Theta^{(l)} \in \mathbb{R}^{k^{(l)} \times D^{(l-1)} \times D^{(l)}}$ denotes a kernel of $D^{(l)}$ filters of layer l, $b^{(l)} \in \mathbb{R}^{D^{(l)}}$ are the biases, one for each filter. From here on, a non-linear activation function as shown in Equation 2.6 is applicable. For comparison use the definition of the inner activation for a normal neuron given in equation 2.5. The advantage of convolutions becomes obvious if Equation 2.39 is reformulated. For this reason, a function $c_i^{(l)}$ is introduced which returns the inputs of a feature map i as a sequence $[a_i^{(l-1)}\ a_{i+1}^{(l-1)}\ \ldots\ a_{i+k^{(l)}}^{(l-1)}]$. The Equation 2.39 is rewritten as

$$z_{io}^{(l)} = \sum_{c=1}^{D^{(l-1)}} \sum_{j=1}^{k^{(l)}} \Theta_{jco}^{(l)} \cdot c_i^{(l)}(a^{(l-1)})_{jc} + b_{jo}^{(l)}. \tag{2.40}$$

The function $c_i^{(l)}(\cdot)$ controls which neuron of the lower layer $l-1$ are connected with the feature map i. For the example in Figure 2.15 the layer has $k^{(l)} = 2$ filter width. Therefore, for the first feature map $c_1^{(l)}(a^{(l-1)})$ would return $[a_1^{(l-1)}\ a_2^{(l-1)}]$. The sequence order is thereby controlled by $c_i^{(l)}(\cdot)$ which is used for dilation layers that are introduced later. Additionally, each feature map can be calculated independently which allows to distribute them over several floating point units. This is favourable as GPUs contain up to 3560 such units (Nvidia P100).

The gradient of the inner activation over the kernel weights is defined as

$$\frac{\partial z_i^{(l)}}{\partial \Theta_j^{(l)}} = a_{(i+j-1)}^{(l-1)}. \tag{2.41}$$

The proof is found analogously to the proof given in Appendix A.2. In contrast to fully connected layers (shown in Section 2.2.1) the gradients of all outputs of the convolution layer have to be summed up per kernel weight. The kernel weight's gradient is defined as

$$\frac{\partial \mathcal{L}}{\partial \Theta_j^{(l)}} = \sum_{i=0}^{M^{(l)}} \delta_i^{(l)} \cdot a_{(i+j-1)}^{(l-1)} \tag{2.42}$$

with $M^{(l+1)} = n^{(l)} - k^{(l)} + 1$.

Example 2.3. *Let us perform the forward propagation of a convolutional $k = 3$ kernel for an inner activation. We start defining the input map with*

$$a^{(l-1)} = [\ldots \quad 2 \quad \underset{\downarrow}{\overset{i}{1}} \quad 5 \quad 3 \quad 9 \quad \ldots].$$

We will consider the red numbers only. Next, we define the kernel as

$$\Theta^l = \begin{bmatrix} 1 & 2 & -1 \end{bmatrix}.$$

We have defined $k = 3$. For convenience, we omit the channels and the bias term. Now we use Equation 2.39 to calculate the output of the red range. We get

$$\begin{aligned} z_i^{(l)} &= \sum_{j=1}^{k} \Theta_j^{(l)} a_{(i+j-1)}^{(l-1)} \\ &= (1 \cdot 1 + 2 \cdot 5 + (-1) \cdot 3) \\ &= 8 \end{aligned}$$

The output $a^{(l)}$ is given by

$$\begin{array}{ccccccc} & & & i & & & \\ & & & \downarrow & & & \\ a^{(l)} & = & [\ldots & g(-1) & g(8) & g(2) & \ldots]. \end{array}$$

Note that the output map will shrink along the axis by $n^{(l+1)} = n^{(l)} - k_1^{(l)} + 1$. The next section introduces the concept of strides and zero padding which both influences the output size $n^{(l+1)}$. ⊖

2.4.1 Strides and zero padding

Convolutions in neural networks can be thought of as windows that are sliding over the inputs. The stride s denotes the amount of tiles by which the window is shifted. If M feature maps are given then $|M_s| = \lceil \frac{M}{s} \rceil$ are processed where $M_s = \{1, 1+s, 1+2s, ..., 1+\lceil \frac{M}{s} \rceil s\}$. The output size o for strided convolution layers is given by

$$o = \lfloor \frac{i + 2p - k}{s} \rfloor + 1 \tag{2.43}$$

where i the input size, p the amount of zeros added to each side the input (zero padding) and k the filter size.

There are two padding stragies that are introduced next:

- If "same padding" is selected then p is set so that $o = (i + 2p - k) \cdot s$.
- If "valid padding" is selected then $p = 0$ and the input size might not be reconstructed by a multiplication with s.

2.4.2 Dilation

The sliding window of normal convolutions contains the k-adjacent inputs (e.g. $a_i^{(l-1)}$, $a_{i+1}^{(l-1)}$, ..., $a_{i+k-1}^{(l-1)}$). For dilated convolutions a dilation factor d is defined such that the window contains the following inputs:

$$c_i^{(l)}(a^{(l-1)}) = \left[a_j^{(l-1)} \quad a_{j+d}^{(l-1)} \quad a_{j+2d}^{(l-1)} \quad \cdots \quad a_{j+(k-1)d}^{(l-1)} \right], \tag{2.44}$$

where $j = 1, 2, \ldots, M^{(l)}$ with $M^{(l)}$ denotes the amount of feature maps of layer l. The output size o decreases by

$$o = i + 2p - (k-1) \cdot d, \tag{2.45}$$

where i the input size, p the amount of zeros added to the input (zero padding) and k the filter size.

There are two padding stragies that are introduced next:

- If "same padding" is selected then p is set so that $o = i$.
- If "valid padding" is selected then $p = 0$ and the output size $o \leq i$.

2.4.3 Deconvolution neural networks

Zeiler et al. (2010) proposed deconvolution neural networks to reconstruct images from some latent information. The latent information can be thought of as the result of a convolution and the deconvolution shall factorise them so it restores the inputs. In terms of Equation 2.39 the deconvolution solves the equation to find $a_{jc}^{(l-1)}$. If no strides and dilations are used then $a_{jc}^{(l-1)}$ depends on the inputs $i = j - k^{(l)} + 1, j - k^{(l)} + 2, .., j$. It it is best imagined if Figure 2.15 is performed backwards. For convenience, the notation is changed to fit the notation of neural networks introduced in Section 2.2.1. Therefore, $a_{jc}^{(l-1)}$ is written as $z_{io}^{(l)}$. The equation for the deconvolution layer is written as

$$z_{io}^{(l)} = \sum_{c=1}^{D^{(l-1)}} \sum_{j=1}^{A} \Theta_{jco}^{(l)} \cdot a_{(i-j+1)c}^{(l-1)}, \tag{2.46}$$

where $i = 1, 2, ..., n^{(l)}$, $o = 1, 2, ..., D^{(l)}$ and $A = \min(i, n^{(l)} - i, k^{(l)})$ is introduced to fit the edges. For simplicity, the bias term has been omitted. The selection function $c_i^{(l)}(\cdot)$ is introduced which returns the neurons in correct order so the Equation 2.46 can be written as

$$z_{io}^{(l)} = \sum_{c=1}^{D^{(l-1)}} \sum_{j=1}^{A} \Theta_{jco}^{(l)} \cdot c_i^{(l)}(a^{(l-1)})_{jc}. \tag{2.47}$$

For dilations the function $c_i^{(l)}(\cdot)$ needs to be defined so it output $[a_{j-(k-1)*d}^{(l-1)}, a_{j-(k-2)*d}^{(l-1)}, \ldots, a_j^{(l-1)}]$. Analogous a definition for strides can be found.

2.5 Optimiser

This section introduces the optimisation algorithms used in this thesis: RMSProp and Adam.

2.5.1 RMSProp - Exponentially weighted moving average of gradients

RMSProp is a stochastic gradient descent algorithm that was proposed by Hinton et al. (2014) for usage with back-propagation. The main idea is to combine an analysis of the gradients' signs with an adaptation of step-sizes for each weight, followed by a division by the running average of its recent magnitude. This combination has been proven successful in many applications.

Imagine a neural network with different weights. Given any error function $\mathcal{L}$ the weights' gradients vary for each neuron and at each step. Let m_B be the magnitude of a gradient vector B defined by

$$m_B = \|B\|_2, \tag{2.48}$$

where $\|\cdot\|_2$ denotes the l_2-norm. Given m_B the differences between gradients are quantifiable. It is difficult to set a single global learning rate that suits all gradients. In full batch learning, all training samples are used at once, thus it can be defined to use only the sign of the gradients and decrease the learning rate if more signs get positive. In RMSProp the idea to use the gradient's sign is combined with the concept of adaptive learning rates.

Given a gradient $\nabla B^{(t)}$, with t denoting the t-th computation of the gradient (or training step), and a gradient accumulator $r^{(t)}$ with $r^{(0)} = 0$. At every step, the square of all the gradient parameters is accumulated.

$$r^{(t)} = \alpha \cdot r^{(t-1)} + (1 - \alpha)\big(\nabla B^{(t)}\big)^2 \tag{2.49}$$

where α functions as decay rate to the previous steps. The update is then computed as

$$\Delta B^{(t)} = \epsilon \cdot \frac{\nabla B^{(t)}}{\sqrt{r^{(t)}} + \varsigma}, \tag{2.50}$$

where ϵ is the global learning rate. In practice, a tiny number ζ is added to the denominator to avoid a division by zero.

2.5.2 The Momentum Method

In some cases, RMSProp is combined with the momentum method. For an intuition of the momentum method imagine a ball on a (error) surface. First, the ball gains velocity in the direction of the steepest descent. But given any acceleration the ball keeps going in its direction even if there is an ascent. However, the ball's orientation is changed gently in direction of the new steepest descent. Finally, it will reach a minimum on the surface. For gradient descent this effect is simulated with the momentum method.

Let $v_B^{(t)}$ be the velocity of the gradient of B at training step t. It is calculated as

$$v_B^{(t)} = \beta \cdot v_B^{(t-1)} - \epsilon \cdot \nabla B^{(t)}, \tag{2.51}$$

where β is called the momentum that works as a decay on the velocity and ϵ is the global learning rate. The weight change is now given by

$$\begin{aligned} \Delta B^{(t)} &= v_B^{(t)} \\ &= \beta \cdot v_B^{(t-1)} - \epsilon \cdot \nabla B^{(t)} \end{aligned}$$

and transformed

$$\Delta B^{(t)} = \beta \cdot \Delta B^{(t-1)} - \epsilon \cdot \nabla B^{(t)}. \tag{2.52}$$

If β is close to 1 and the gradient's orientation is not changing much then the velocity will accumulate and step faster ahead than simple gradient descent. Since at the beginning of learning the gradients may usually be large, Hinton et al. (2014) propose to start with a value of $\beta = 0.5$. When the large gradients disappear β can be increased to values of 0.9 or even 0.99.

If used with RMSProp, the momentum is introduced in Equation 2.50 with

$$\Delta B^{(t)} = \beta \cdot \Delta B^{(t-1)} - \epsilon \cdot \frac{\nabla B^{(t)}}{\sqrt{r^{(t)}}}. \tag{2.53}$$

However, the authors state that the momentum in RMSProp has not increased the learning speed significantly.

Sutskever et al. (2013) show that the Nesterov's Accelerated Gradient (NAG) technique acts similar to the momentum method but behaves more stable for large values of β. The authors rewrite the original formulation as

$$v_B^{(t+1)} = \beta \cdot v_B^{(t)} - \epsilon \cdot \frac{\partial \mathcal{L}(B^{(t)} + \beta \cdot v_B^{(t)})}{\partial B^{(t)}} \tag{2.54}$$

$$\Delta B^{(t+1)} = v_B^{(t+1)}, \tag{2.55}$$

which shows the relationship between both methods. With $\mathcal{L}(B^{(t)} + \beta \cdot v_B^{(t)})$, the gradient at position $B^{(t)} + \beta \cdot v_B^{(t)}$. In other words, the gradient one step ahead is calculated to perform a correction on the new velocity $v^{(t+1)}$ is calculated. Sutskever et al. (2013) show that NAG is less prone to oscillations and thus allows to use larger values of β than it is possible with the standard momentum method.

2.5.3 Adam Optimiser

Kingma and Ba (2014) proposed the Adam optimiser that combines Adaptive Gradient Algorithm (AdaGrad) and RMSProp. It is based on adaptive estimates of the first and second moments of the gradient to find a suitable step size that is approximately bounded by global learning rate ϵ. This allows to restrict the range in which the optimiser's hyper-parameters are searched, if any assumption about weight changes can be made.

Let $m^{(t)}$ be the moving average of the gradient of the first moment (the mean) and $v^{(t)}$ be the moving average of the gradient of the second moment (uncentred variance). β_1 and β_2 denote the decay rates of the moving averages. The gradient of weights at training step t is given by $\nabla B^{(t)}$. The updates of the first and second moments are found with

$$m^{(t)} = \beta_1 \cdot m^{(t-1)} + (1 - \beta_1) \cdot \nabla B^{(t)}, \tag{2.56}$$

$$v^{(t)} = \beta_2 \cdot v^{(t-1)} + (1 - \beta_2) \cdot \left(\nabla B^{(t)}\right)^2. \tag{2.57}$$

The moments are initialised as $m(0) = 0$ and $v(0) = 0$. Therefore, both moments are biased towards 0 which is sensible especially in the beginning of training (and if β_1 and β_2 are close to 1). To counteract this, bias-corrected estimates $\hat{m}^{(t)}$ and $\hat{v}^{(t)}$ are introduced as

$$\hat{m}^{(t)} = \frac{m^{(t)}}{1 - \beta_1^t}, \tag{2.58}$$

$$\hat{v}^{(t)} = \frac{v^{(t)}}{1 - \beta_2^t}. \tag{2.59}$$

Note that the t in β_1^t denotes the exponent. The weights' delta is given by

$$\Delta B^{(t)} = \epsilon \cdot \frac{\hat{m}^{(t)}}{\sqrt{\hat{v}^{(t)}} + \varsigma}, \tag{2.60}$$

where ϵ denotes the learning rate and ς is a small number to avoid a division by zero. Note that the effective step taken at training step t has an upper bound of $|\Delta B^{(t)}| \leq \epsilon \frac{1-\beta_1}{\sqrt{1-\beta_2}}$ if $(1 - \beta_1) > \sqrt{1 - \beta_2}$,

and $|\Delta B^{(t)}| \leq \epsilon$ otherwise. Given the second case ϵ can be interpreted as a trust region in which a weight update makes sense (e.g. if a prior distribution over the parameter is known).

The bias correction terms are derived by estimating the error of the zero initialisation. Kingma and Ba (2014) describe the correction term for the second moment $\hat{v}^{(t)}$ as follows ($\hat{m}^{(t)}$ is found analogously). The second moment's exponential moving average can be written as function of previous gradients:

$$v^{(t)} = (1-\beta_2)\sum_{i=1}^{t}\beta_2^{t-i}\cdot\left(\nabla B^{(i)}\right)^2. \tag{2.61}$$

The expectation of $v^{(t)}$ at training step t relates to the true second moment of $\mathbb{E}[\left(\nabla B^{(t)}\right)^2]$ which allows to find the discrepancy of both. It is given by

$$\begin{aligned}
\mathbb{E}[v^{(t)}] &= \mathbb{E}\left[(1-\beta_2)\sum_{i=1}^{t}\beta_2^{(t-i)}\cdot\nabla B^{(i)}\right] \\
&= \mathbb{E}\big[(\nabla B^{(t)})^2\big](1-\beta_2)\sum_{i=1}^{t}\beta_2^{t-i}+\xi \\
&= \mathbb{E}\big[(\nabla B^{(t)})^2\big](1-\beta_2^t)+\xi
\end{aligned}$$

where $\xi = 0$ if the true second moment is stationary. Because β_1 and β_2 should be chosen such that gradients in the past have small values assigned the error ξ will be kept small otherwise. Therefore, the term $(1-\beta_2^t)$ is left to account for the error of initialising the running averages with zeros. This has been introduced in Equation 2.58 to correct the error.

2.6 Generative frameworks

In this thesis, concepts leading to a generation of samples are referred to as generative frameworks. First, the concept of latent variables is explained. Second, frameworks used in this thesis are described. Implementation details will be described in Chapter 3.

2.6.1 The concept of latent variables

Latent variables are values defined before the generative process starts. Imagine a model that is about to produce music. It makes sense to decide first what genre to play and then to produce it with all particular characteristics. Obviously, the result will not be harmonic if, for example, half of the song is yodel, and half is *Death Metal*. A latent variable z may contain the information about the genre. It is chosen from a certain domain, e.g. $z \in \{\text{Unterwaldner, Muotathaler, Berner, ...}\}$, and is called latent because it is not necessarily known which information it includes. For each data point x of the dataset there has to be one or many settings of latent variables that causes the model to generate something similar to x. Otherwise the model is not representative.

Formally, let z be a vector of latent variables that can be sampled easily according to a probability density function $p(z)$ defined over $\mathcal{Z}$. Let $f(z;\Theta)$ be the family of deterministic functions with parameters $\Theta \in \vartheta$ where $f : \mathcal{Z}\times\vartheta \to \Psi$ with Ψ a high dimensional space. If z is random, f deterministic and parameters Θ fixed, then $f(z;\Theta)$ is a random variable in the space Ψ. Given data points $x \in \mathcal{X}$ where $\mathcal{X} \subseteq \Psi$ the aim is to optimise Θ in such a way that $f(z;\Theta)$ will likely produce samples of $\mathcal{X}$. Therefore, the maximum likelihood estimation, introduced in Section 2.2.2, is used as optimisation criterion. Given $\mathcal{X}$ and $\mathcal{Z}$ are from continuous distributions, the law of total probability is used to define $p(x)$ depending on z as follows:

$$p(x) = \int_z p(x|z;\Theta)p(z)\mathrm{d}z \tag{2.62}$$

$f(z;\Theta)$ has been replaced by the distributive density function $p(x|z;\Theta)$ (posterior distribution) to introduce the dependency of x on z. The prior distribution $p(z)$ (latent variables) is in general unknown and has to be learned. For Variational Autoencoders (VAE) the posterior distribution is often assumed to be Gaussian.

$$p(x|z;\Theta) = \mathcal{N}(x|f(z;\Theta), \sigma^2 * I) \tag{2.63}$$

The Gaussian has mean $f(z;\Theta)$ and covariance equal to the identity matrix I times a scalar σ that will be treated as a hyper-parameter. This implies that the dimensions of $\mathcal{X}$ are not correlated with each other. Early in training the generator model will not produce outputs that are identical to any particular x. The Gaussian distribution allows the application of back-propagation in order to increase the expectation $p(x)$ by adapting Θ of $f(z;\Theta)$. Therefore, the produced outcome x for a certain z becomes more likely to be in $\mathcal{X}$ under the generative model.

Note that the output distribution does not necessarily need to be Gaussian, but has to fulfil the properties that $p(x|z)$ can be computed efficiently and that it is continuous in Θ.

2.6.2 Variational Autoencoder (VAE)

Kingma and Welling (2013) propose Variational Autoencoders (VAE) which learns the latent variable representation following a predefined distribution type. The VAE approximates the maximum likelihood optimisation function with a relatively small error. The encoder learns the parameters of the latent variables' distribution which are used in a transformation function to map samples of a noise distribution into the latent distribution. The function has to be differentiable in order to work with back-propagation. The decoder maps the latent variables to the output space, and the distribution of the outputs resembles the distribution of the training samples after successful learning. The authors demonstrate the ability of the algorithm on the MNIST (hand-writing), and the Frey Face datasets where it outperformed the Wake-Sleep and Monte Carlo Expectation Maximisation algorithms in terms of the marginal log-likelihood.

Figure 2.16 visualises a possible architecture of a VAE. It uses the concept of Autoencoders which are of the shape of two cones mirrored by a vector, provoking a bottleneck in the middle. The intuition of the network is to compress the input x into a vector z (encode) such that it can be reconstructed without error (decode). The compressed feature vector z is called latent variables. Given any learned sample x and its compressed representation z, an optimal decoder is able to reconstruct x from z without additional information. The autoencoder lacks the ability to produce new samples similar to the training data $\mathcal{X}$. VAE, however, introduce this ability by additionally learning the distribution of the latent variables. If the distribution of $\mathcal{Z}$ is known z can be sampled at random from it and an optimal decoder will generate new samples x based on z that are likely to be in $\mathcal{X}$. VAEs use the maximum likelihood estimation, introduced in Section 2.2.2, as optimisation criterion. In the following, the functionality of encoders, latent variables and decoders is presented.

The decoder takes some latent variables z and "decodes" them to samples x. The likelihood of x being reconstructed under z is given by $p(x|z)$. In order to generate new samples, it is necessary that latent variables can be sampled. Therefore, an assumption is made that $z \sim \mathcal{N}(\mu, \Sigma^2)$, follows a Gaussian distribution. $p(z)$ denotes the likelihood of z being in the latent variables. Other distributions are suitable if they are differentiable, since this is a necessary condition when applying back-propagation. Another assumption is necessary to model the output distribution. In the following, it is assumed that $\mathcal{X}$ is Gaussian as well. As a result the decoder is supposed to learn $\mu_\pi(z)$ and $\sigma^2_\pi(z)$ to reconstruct x and estimate $p_\pi(x|z)$. See Figure 2.16 for a schematic representation of the decoder.

The encoder's task is to learn the prior distribution, the distribution of $\mathcal{Z}$, with which the probability $p(z|x)$ is calculated. By applying the theorem of Bayes it yields

$$\begin{aligned} p(z|x) &= \frac{p(x,z)}{p(x)} \\ &= \frac{p(x,z)}{\int\limits_z p(x,z)\mathrm{d}z} \\ &= \frac{p(x|z)p(z)}{\int\limits_z p(x|z)p(z)\mathrm{d}z}. \end{aligned} \tag{2.64}$$

The denominator of Equation 2.64 is intractable, and thus solutions can only be found by sampling over the distribution of z. Instead of sampling, a neural network $q_\phi(z|x)$ is trained that approximates the true prior $p(z|x)$ which is assumed to be Gaussian. Hence, the encoder learns

$$q_\phi(z|x) = \mathcal{N}\big(\mu_\phi(x), \sigma^2_\phi(x)\big). \tag{2.65}$$

Figure 2.18 shows a schematic representation of the encoder and the interaction with the decoder. Note that the VAE consists of two neural network: the encoder learns parameters for the latent variables,

and the decoder learns parameters for the output distribution. However, the network of Figure 2.18 suffers from a serious problem. The encoder outputs the distribution parameters $\mu_\phi(x)$ and $\sigma^2_\phi(x)$ but a latent variable z has to be sampled before the decoding process can start. Unfortunately, this is intractable, and no gradient is defined for this operation. As a result, back-propagation is not applicable. Furthermore, the definition of a loss function is needed to work with back-propagation.

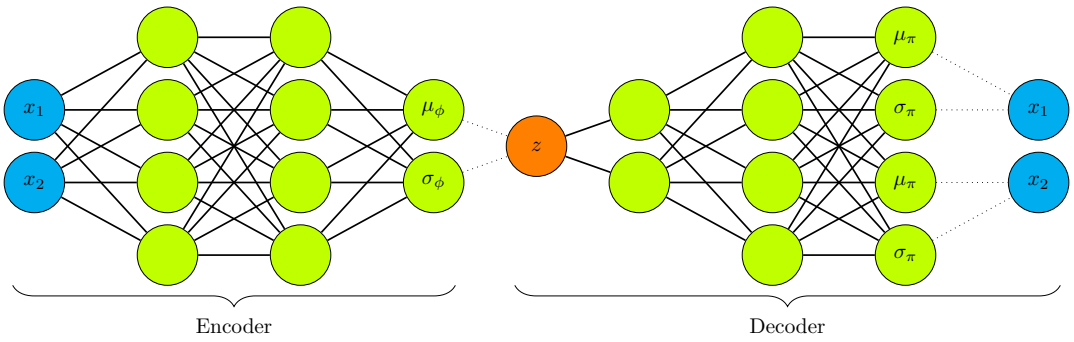


Figure 2.16: A schematic architecture of a Variational Autoencoder with to inputs x_1 and x_2 and one latent variable z.

First, the intractability of the sampling operation is solved by the so called "reparametrisation trick". The trick makes use of the fact that any distribution in d dimensions can be reconstructed by a set of d samples, normally distributed, if they are mapped through a sufficiently complicated function. Figure 2.17 shows an example of a function mapping values from Gaussian to a circle-like distribution.

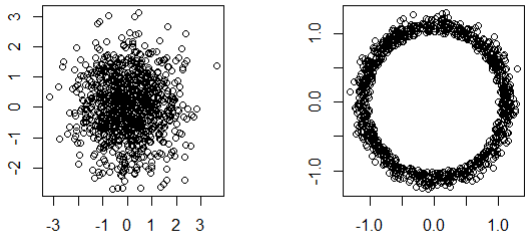


Figure 2.17: On the left side samples are drawn from a Gaussian distribution $\mathcal{N}(0, 1)$. The samples are mapped by the function $f(x) = \frac{x}{10} + \frac{x}{\|x\|_2}$ which results in the distribution shown on the right side.

Instead of sampling from $\mathcal{N}(\mu_\phi(x), \sigma^2_\phi(x))$ directly, $\epsilon \sim \mathcal{N}(0, 1)$ is sampled followed by a multiplication with $\mu_\phi(x)$ and addition with $\sigma^2_\phi(x)$. Figure 2.18 shows the VAE with the applied reparameterisation trick. ϵ is first stretched then shifted through the learnt parameters of the encoder. Multiplication and addition are both operations that are tractable and a gradient is easily found for both. Therefore, back-propagation is applicable and the first problem is solved.

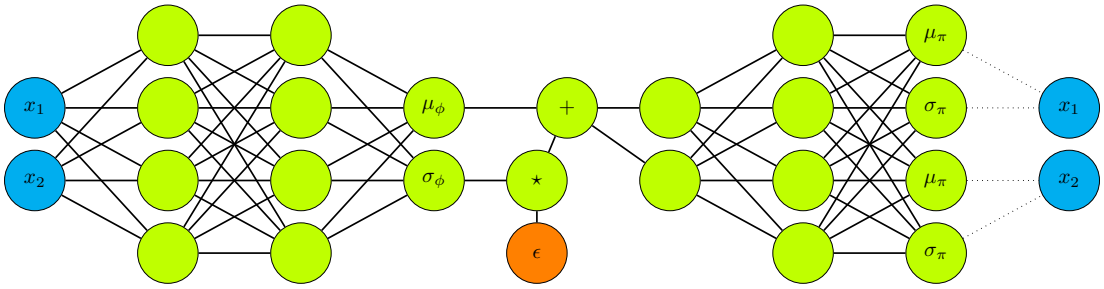


Figure 2.18: A schematic architecture of a Variational Autoencoder with to inputs x_1 and x_2 and one latent variable z.

Second, a loss function needs to be found which optimises the encoder $q_\phi(z|x)$ as well as the decoder $p_\pi(x|z)$. In general, a maximum log-likelihood estimation is carried out (Section 2.2.2). Given a training sample $x \in \mathcal{X}$, the log-likelihood is defined as $\log\big(p(x)\big)$. We want to extend this definition now to make it explicitly dependent on the encoder and decoder. Because $\sum_z p(z|x) = 1$ the log-likelihood $\log\big(p(x)\big)$ can be extended, and by approximating $p(z|x)$ the encoder is introduced with

$$\log\big(p(x)\big) = \sum_z q_\phi(z|x) \log\big(p(x)\big). \tag{2.66}$$

Next, $p(x)$ is substituted with $\frac{p(x,z)}{p(z|x)}$, and expanded it with $\frac{q_\phi(z|x)}{p(z|x)}$. We find

$$\begin{aligned} \log\big(p(x)\big) &= \sum_z q_\phi(z|x) \log\left(\frac{p(x,z)}{p(z|x)}\right) \\ &= \sum_z q_\phi(z|x) \log\left(\frac{p(x,z)}{p(z|x)} \cdot \frac{q_\phi(z|x)}{p(z|x)}\right). \end{aligned} \tag{2.67}$$

Note that if $q_\phi(z|x)$ is a perfect approximation of $p(z|x)$ then $\frac{q_\phi(z|x)}{p(z|x)} = 1$. Rearranged, Equation 2.67 reads as

$$\log\big(p(x)\big) = \sum_z q_\phi(z|x) \log\left(\frac{p(x,z)}{p(z|x)}\right) + \sum_z q_\phi(z|x) \log\left(\frac{q(z|x)}{p(z|x)}\right). \tag{2.68}$$

The first term is substituted by L^V and the last term is a KL divergence of $q_\phi(z|x)$ and $p(z|x)$ (see definition in Equation 2.29). Therefore, Equation 2.68 simplified to

$$\log\big(p(x)\big) = L^V + KL\big(q_\phi(z|x) \| p(z|x)\big). \tag{2.69}$$

Note that $L^V + KL\big(q_\phi(z|x) \| p_\pi(z|x)\big) \geq L^V$, which is why it is called variational lower bound. The KL divergence quantifies how well $q_\phi(z|x)$ approximates $p(z|x)$, and equals zero if it is perfect, which implies that $\log\big(p(x)\big) = L^V$. Because it is difficult to define the true posterior $p(z|x)$, VAE optimise L^V only. So far, L^V only depends on the encoder. Therefore, it is necessary to additionally introduce the decoder. The variational lower bound is given by

$$\begin{aligned} L^V &= \sum_z q_\phi(z|x) \log\left(\frac{p(x,z)}{p(z|x)}\right) \\ &= \sum_z q_\phi(z|x) \log\left(\frac{p(x|z) \cdot p(z)}{p(z|x)}\right) \\ &= \sum_z q_\phi(z|x) \log\left(\frac{p(z)}{p(z|x)}\right) + \sum_z q_\phi(z|x) \log\big(p(x|z)\big) \\ &= -\sum_z q_\phi(z|x) \log\left(\frac{q_\phi(z|x)}{p(z)}\right) + \sum_z q_\phi(z|x) \log\big(p_\pi(x|z)\big). \end{aligned} \tag{2.70}$$

The first term in Equation 2.70 is a negative KL divergence of $q_\phi(z|x)$ and $p(z)$. The second summand contains the decoder and is an expectation of $p_\pi(x|z)$ over $q_\phi(z|x)$. Simplified, the variational lower bound is given by

$$L^V = -KL\big(q_\phi(z|x) \| p(z)\big) + \mathbb{E}_{q_\phi(z|x)}\big(p_\pi(x|z)\big). \tag{2.71}$$

Injected into Equation 2.69, the log-likelihood is given by

$$\log\big(p(x)\big) = -KL\big(q_\phi(z|x)\|p(z)\big) + \mathbb{E}_{q_\phi(z|x)}\big(p_\pi(x|z)\big) + KL\big(q_\phi(z|x)\|p(z|x)\big). \tag{2.72}$$

The term $-KL\big(q_\phi(z|x)\|p(z)\big)$ functions as a regularisation term that forces the prior towards $\mathcal{N}(0,1)$. The reconstruction quality is measured by $\mathbb{E}_{q_\phi(z|x)}\big(p_\pi(x|z)\big)$. In practice, it is approximated by performing the calculation over n samples (minibatch used in stochastic gradient descent). Therefore, the error is given by $\sum_{i=1}^{n} \log\big(p(x^{(i)})\big)$.

2.6.3 Related techniques

The term fully visible belief networks (FVBN) was used by Frey et al. (1995) and defined as follows: "...a special case of the Helmholtz machine [Dayan et al. (1995)] where the top-down network is fully connected and there are no hidden units"(p.3). FVBNs are generative models with a tractable density, since the chain rule of probability can be applied to decompose a distribution. For a sample vector x of n-dimensions the product of n one-dimensional probability distributions is calculated to return the probability of the sample itself. A drawback of this approach is that samples must be generated one entry at a time, so the cost of generating a sample is $\mathrm{O}(n)$.

Goodfellow et al. (2014) proposed Generative Adversarial Networks (GAN), which combine a generator model with a discriminator model in a two player minimax game. The authors apply an optimisation function to combine the generator with the discriminator. They prove that given enough capacity for both models, i.e., the models have enough depth and neurons to learn the optimal distribution function, the generator's output distribution converges to the training samples' distribution if a gradient descent method with convenient step size is applied. They demonstrated their framework by combining two Multi-layer Perceptrons (MLP) and testing it on the MNIST, Toronto Face Database (TFD) and CIFAR-10 (labelled images of different classes) datasets. GANs reached state-of-the-art performance and, in contrast to other frameworks, do not depend on latent variables. The only necessary condition is that the model functions have to be differentiable.

Mescheder et al. (2017) proposed Adversarial Variational Bayes (AVB), which are based on VAE and extends the encoder to learn any distribution of latent variables. Therefore, no definition of noise distribution has to be specified, instead, noise is added as an additional input to the inference model. Because the distribution of the latent variables is unknown the expectation cannot be calculated. AVB solves this problem by introducing a discriminator which determines whether pairs of samples and latent variables are valid or not. It is connected to the algorithm as a two-player game with the encoder and decoder. The authors show that if the discriminator can represent any function the optimisation function has a Nash-equilibrium with the optimal parameters of the decoder being a maximum-likelihood assignment. They tested their approach on a synthetic dataset showing that their approach is able to learn any distribution whereas the VAE failed in this task. Using a 5-layer deep convolutional neural network as decoder and a 4-layer fully connected neural network with 1024 units for the discriminator, they achieved on the MNIST dataset a negative log-likelihood of 79.5. As a comparison, a VAE was trained reaching a negative log-likelihood of 87.2. For both tested approaches the latent variable space was set to 32 dimensions.

In van den Oord et al. (2017b), the VAE framework is extended to use discrete latent variables, using vector quantization (VQ), which leads to its name VQ-VAE. The posterior and prior distributions for the latent variables are categorical, and the samples drawn from these distributions index an embedding table which is used as input to the decoder. The encoder output is mapped to the nearest neighbour of the discrete latent variables. Because this function is intractable the loss of the decoder input is directly copied to the output of the encoder during back-propagation, ignoring the discretisation step. To learn the latent variable space the l_2 error is used to move the embedding vectors towards the encoder outputs. Because the embedding space can grow arbitrarily if it is not trained as fast as the encoder, a commitment loss is added to the variational lower bound to force the encoder on certain embeddings. For audio generation the WaveNet is used as decoder. The encoder consists of 6 strided convolutions with stride 2 and window-size 4, which results in a latent space 64 times smaller than the original waveform. The latent variable space is 512-dimensional and has one feature map. The decoder is conditioned using the latent variables and additionally the speaker id using a one-hot encoding. The authors found that the model is able to produce clear words and part-sentences[1] whereas the original

[1] https://avdnoord.github.io/homepage/vqvae/, accessed 10.10.2017

WaveNet sounds like babbling. Furthermore, the model is able to reproduce the speech of one speaker with the characteristics of another speaker showing that the encoded representation has factored out speaker-specific information.

2.7 Music synthesis

In this section, the most common approaches to music synthesis are introduced. Generally, two techniques are distinguished:

- Lead-sheet generation: The training data consist of music in lead-sheet notation. The goal therefore is to produce new music in lead-sheet format. It is often played by a synthesiser using a piano.
- Audio generation: The training data are in waveform audio file format (WAV). The aim is to produce waveform directly. These approaches have to reconstruct not only the melody but also the timbre of their genres.

2.7.1 Lead-sheet generation

Colombo et al. (2016) proposed a deep multi-layer model for algorithmic composition of monophonic melodies based on Recurrent Neural Networks (RNN) with the Gated Recurrent Unit (GRU). The music is described as a sequence of notes where each note is given by its duration and pitch represented by a vector space model using one-hot encoding (the rhythm and melody). The training data consisted of a large dataset of Irish folk songs. The model was trained to continue an existing melody and to compose entirely new songs. During training the generative model was optimised to predict each upcoming note given the previously presented notes of songs. The model consists of two separate multi-layer RNN's, a rhythm and a melody network, which both have three hidden layers of 128 GRUs, and output a probability vector for pitch and duration, respectively. The authors observed that the model was able to preserve the metric structure and the rhythmical patterns in the continuation process. In the task of composing new songs the model produced melodies which were different from the training set but carried identical features on many time scales. The authors concluded: "The model is consequently able to generate new pieces of music in a completely autonomous manner" (Colombo et al., 2016, p.9). Their data corpus is based on the Irish music corpus of Henrik Norbeck[2].

Using a comparable but extended representation of music, Johnson and Weintraut (2017) proposed an approach consisting of an interval and a chord expert model which are connected as Product-of-Experts system. Both models are based on RNN of two LSTM layers with 300 neurons each. The output consists of the combined probability of which note to play next. Using a Product-of-Experts function the authors managed to introduce risk levels and expert weighting which modifies the behaviour of the trained model after learning. As training data Jazz-music was used which is freely available[3]. The authors concluded that their model was able to create solos of arbitrary length. The model suffered from shortcomings of occasionally "wrong" notes and rote learning of melody from the original corpus which they identified as *failure to learn music rules of thumb* or *over-fitting*.

Tikhonov and Yamshchikov (2017) used 4-layered RHN models in a Variation Autoencoder approach as encoder and decoder to the challenge monotonic music generation based on lead sheet notation. Unlike the previous approaches, they omitted the strengths of the notes and normalised the songs according to pauses and pitch. Their data corpus contained 15000 normalised tracks. For data representation they used embeddings for pitch, octave and duration. They reached a cross-entropy error of 2.11 compared with a word-based LSTM model which reached 2.34. On a subjective view they assessed their proposed model to produce more interesting macro structures than other approaches.

2.7.2 Audio generation

Nayebi and Vitelli (2015) proposed an RNN-based approach which produces an audio signal directly. They tested a single hidden layer network based on GRUs or LSTM cells. As input raw audio samples are converted into a mono-channel waveform sampled at 44.1 kHz. It is then split up into chunks of same size and fractionated to the phase and magnitude of the time domain waveform using the discrete

[2]http://www.norbeck.nu/abc/, accessed 25.09.2017
[3]https://www.cs.hmc.edu/~keller/jazz/improvisor/, accessed 25.09.2017

Fourier transform algorithm. A sequence consists of a quarter of a second. The best results were found using a layer size of 2048 neurons for which the authors concluded that only a subset of the frequencies are actually relevant for human perception. They observed that generated sequences, more than three times larger than the initial seed, tend to get stuck in generation loops. The LSTM cell produced musically plausible samples and outperformed the GRU whose results were unsatisfactory. As data corpus two artists were used, separately, each with 20 songs of 3 - 7 minutes length.

In van den Oord et al. (2016a) the authors proposed WaveNet designed to the task of speaker speech generation, text to speech and music audio modelling. It is based on dilated causal convolutions to allow their receptive field to grow exponentially with each layer. The dilation is doubled for every layer up to a limit and then repeated. The output is calculated by application of softmax to 256 classes. A μ-law transformation is applied to reconstruct the 16-bit raw audio signal. Additionally, they introduced local and global conditioning to the tanh-activation function in order to learn e.g. characteristics of a single speaker (global conditioning) and/or linguistic features (local conditioning). The authors do not give evidence about the amount of hidden layers or their size, nor about the hyper-parameters used in the experiments. In the task of speech generation the model produced language-like words with realistic sounding intonations. They identified the receptive field (of 300 milliseconds) as bottleneck for the creation of real words. The dataset included 44 hours of data from 109 speakers. For music generation the enlargement of the receptive field was found crucial. Unfortunately, it is not described by what extend. The authors concluded that the samples were often harmonic and aesthetically pleasing although the genre, instrumentation, volume and sound quality varied from second to second. The MagnaTagATune[4] dataset used for training consisted of 200 hours of music audio with different genre, instrumentation, tempo, volume and mood.

Engel et al. (2017) proposed the WaveNet-autoencoder model that removes the need for external conditioning of the WaveNet approach. They challenged the model in the task of producing sound of specific instruments. Their temporal encoder model is a 30-layer non-linear residual network of dilated convolutions followed by 1x1 convolutions. Each convolution has 128 channels and uses ReLU activation. The output is fed into another 1x1 convolution before it is downsampled with average pooling to obtain the encoding. The decoder is a WaveNet model which is conditioned by biasing every layer with a different linear projection of the temporal embeddings. They propose a new dataset NSynth consisting of 306'043 musical notes of 1006 instruments with annotations. For visual evaluation a constant-q transformation is applied to the data. Additionally, they proposed the usage of an Inception Score for audio. The model succeeded in some tasks to produce similar patterns compared to the original audio snippet, although it has a tendency to exaggerate amplitude modulation behaviour. Additionally, they found that the model gets confused with pitches one octave away which accounted for 20% of the absolute classification errors.

Another approach, called SampleRNN proposed by Mehri et al. (2016), utilises RNN's at different scales to model long ans short term dependencies in audio waveforms while training on short sequences which is more memory efficient. They challenged the model on the tasks of speech synthesis, human vocal sounds and Beethoven's piano sonatas calculating the negative log-likelihood of the produced data distribution. Their 3-tier model consists of two RNN with one hidden layer of 1024 GRU's followed by a multi-layer perceptron (2 fully connected layers with ReLU activation of 1024 neurons and 1 layer of 256 before softmax). The output is a vector of 256 length (classes) which is subsequently transformed using a non-linear quantisation as in WaveNet. Each tier applies an up-sampling by a factor. Their model outperformed the WaveNet approach with 1.076 log-likelihood (WaveNet: 1.464) in the music generation task. A human evaluation was conducted based on generated samples which were hand selected as some batches produced rather noise. They confirmed the preference of listeners for SampleRNN over WaveNet. Their data corpus involved 20.5 hours of speech, 3.5 hours of human vocal sounds and 10 hours of non-vocal audio (Beethoven).

2.7.3 Other related approaches

Salimans et al. (2017) described an extension to the PixelCNN approach (van den Oord et al. (2016b)) to generate images. It is an autoregressive model which means that it generates one pixel at a time. Using the RGB color range it starts with an all-zeroed filter for the initial pixel in order to suppress not yet generated values in the following layers. The filter is incrementally lifted the more pixels are generated. This way the following convolutional layers can include the neighbouring pixels as information similar to a recurrent node. The model consists of 6 blocks of ResNet (3 convolution, 3

[4]http://mirg.city.ac.uk/codeapps/the-magnatagatune-dataset, accessed 26.09.2017

deconvolution). Instead of diluted convolutions the authors introduced convolutions with stride of 2 which reduces the computational costs but also increases the loss of information at every layer. To compensate for this, short-cut connections are added (e.g. first to sixth, second to fifth,...). Instead of a full 256-way softmax for each color channel a mixture of logistic distributions is used. The blue channel is allowed to depend linearly on red and green and the green channel on red only. To avoid overfitting dropout is applied to each ResNet block on the residual path after the first convolution. For the CIFAR-10 dataset the model achieves a negative log-likelihood (bits per sub-pixel) of 2.92 and outperforms the PixelRNN variant which reached a score of 3.00.

2.7.4 Conclusion state-of-the-art for music synthesis

For this thesis it has been decided to use models that generate audio directly (see demarcations in Section 1.5). Therefore, approaches of Section 2.7.2 are of interest. Two approaches have been identified as relevant for this work:

- **WaveNet:** The approach succeeded in producing samples that are clearly identified by listeners to be piano music. The sound is colourful and clear. The WaveNet source code is not freely available from the authors but has been reimplemented by community on github[5].
- **SampleRNN:** It was evaluated on the task of classical music, Beethoven, and produced samples on which the different instruments were identifiable. Compared to WaveNet it succeeded in producing polyphonic music. The code is freely available by the authors[6] but unfortunately it is not executable. However, the approach does not contain complex structures and can be reimplemented with medium effort.

It has been decided that both approaches are to be tested in this thesis to have a baseline for further experiments.

[5]https://github.com/ibab/tensorflow-wavenet, accessed 15.01.2018
[6]https://github.com/soroushmehr/sampleRNN_ICLR2017, accessed 15.01.2018

Chapter 3

Setup and Models

This chapter describes the used model architectures (see Sections 3.5 and 3.6) and the training workflow (see Section 3.2.1). The analysis techniques used for this thesis are given in 3.7.

All models are written in Python 3.5 with the usage of TensorFlow™ 1.4 (Martìn Abadi et al. (2015)). TensorFlow™ is an open source software library for numerical computation. It uses a data flow graph where nodes represent mathematical operations, and edges are multidimensional data arrays, called tensors. TensorFlow™ provides a web service TensorBoard which shows histogram of specific values while training.

3.1 Data preparation

The content of the data corpus is described in Section 1.4. This section contains the data conversion and preparation before audio files were stored.

3.1.1 Audio file preparation

Audio files were delivered by the department Music that ripped the data from CDs. Additionally, the experts provided an excel list (expert's annotation list) that contained the following information:

1. CD Title
2. CD Subtitle
3. Track number to
    - Yodel with an instrument of a specific region
    - Yodel without an instrument of a specific region
    - Yodel with text with an instrument
    - Yodel with text without an instrument

The following regions have been separated:

- Unterwalden (UW)
- Schwyz (SZ)
- Lucerne (LU)
- Bern (BE)
- Appenzell Innerrhoden (AI)
- Appenzell Ausserrhoden (AR)
- Toggenburg

The audio file names were named and stored in the following folder structure:

- <CD Title name>/<CD Subtitle name>/<Track-Nr> <Name>.wav

or

- <CD Title name>/<Track-Nr> <Name>.wav

The data corpus is accessible as shown in Appendix A.4.1. All audio files were ripped in stereo and 44.1 kHz.

The file names contained special characters (e.g. ä, ö, ü) which made it necessary to transform them into an operating system with character set Latin-1. The preparation process consists of two KNIME workflow (Berthold et al. (2007)): Extract naming ("Extract_Naming_Data_Corpus") and annotation ("Create_Annotation_Data_Corpus"). Both KNIME workflows and the conversion script is accessible as described in Appendix A.4.1. After execution of all scripts and workflows, the data corpus consists of audio files without any special characters and an additional annotation file in comma-separated value format. It contains an annotation vector for each audio file which includes the information which was described above. Both workflows are briefly introduced next.

Workflow extract naming

The workflow "extract naming" prepares an excel list consisting of CD title and CD subtitle. The title was derived from the songs' top folder and the subtitle from the subfolder if existing. Figure 3.1 shows the data workflow. As input the root directory of the data corpus is necessary. File names without any special characters are a prerequisite to run the workflow. The result of the workflow has to be manually compared with the expert's annotation list, and all differences need to be eliminated. Finally, the expert's annotation list must match the directory names of the data corpus (CD Title = Directory, CD Subtitle = Subdirectory). This step is the prerequisite for the next workflow annotation.

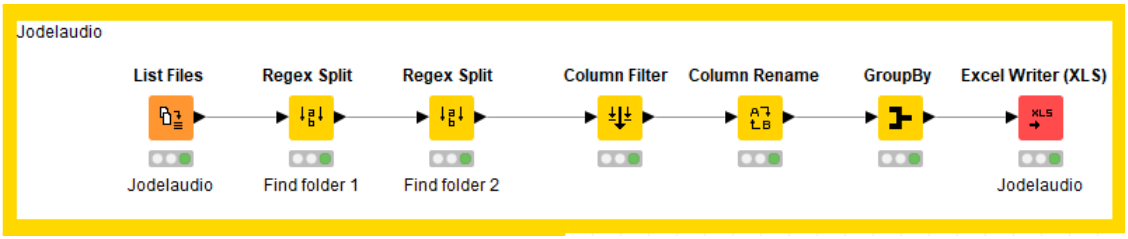


Figure 3.1: The workflow "extract naming" creates an excel list with all CD titles and subtitles that match the directory structure of the data corpus.

Workflow annotation

Figure 3.2 shows the workflow annotation. Three files in format CSV are its output:

1. List of each song with Title, Subtitle and file name according to folder structure to annotations (vector of ones and zeros)

2. List of each audio file with faulty annotations

3. List of each audio file without annotations

The prerequisite for this workflow annotation is described in the chapter workflow extract naming.

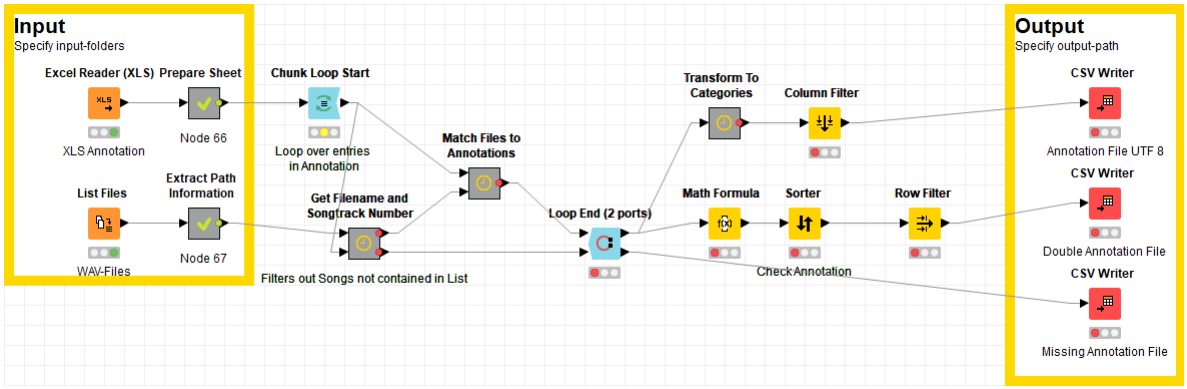


Figure 3.2: The workflow annotation creates three files in CSV format with songs to annotation mapping: Tables of all CD titles and subtitles in encoding UTF-8.

The Input XLS is the expert's annotation list which contains the information as described above. The second input is the path to the root directory of the data corpus.

3.2 Training and samples generation

This section introduces the training and samples procedures that were written in python for this thesis. Section 3.2.1 describes the training procedures and Section 3.3 how audio data are loaded and pre-processed. The source code is accessible as shown in Appendix A.4.1. Some functionality originated from WaveNet-GitHub repository[1] but was adapted to fit the cause.

All settings can be configured as a JSON-file that is delivered as a parameter ("–settings") to the start a training procedure (e.g. "train.py"). The settings file include the flowing information:

- Data repository parameters (e.g. which training files to include)
- Optimiser parameters (e.g. learning rate, optimiser name, decay rates, momentum, L2 regularisation)
- input processing parameter (e.g. batch size, sample size, sample rate, silence threshold)
- JSON filename of model

The JSON model file contains specific configuration about a model architecture. It has been chosen to use a factory pattern to load the models.

3.2.1 Training procedure

The training procedure is shown as a sequence diagram in the Appendix, Figure A.2. The input processing was outsourced to a class AudioReader that is explained in Section 3.2.2. The following pre-processing steps are done before the training procedures start:

1. The model is initiated using a factory that creates the model from the delivered JSON model file.
2. The sample size is set on the AudioReader.
3. The AudioReader returns a tensor representing an operation of the computation graph for dequeuing a mini batch from the samples queue.
4. The Model extends the computation graph and adds all calculation and the loss function to it.
5. The optimiser is initialised and returns the gradient operations.
6. The gradient operations are given to the model so gradient clipping can be applied. It allows to keep the gradients in a specific value range (e.g. gradient is in $[-100, 100]$).

[1] https://github.com/ibab/tensorflow-wavenet, accessed 15.01.2018

7. The TensorFlow™ session is initiated.
8. The AudioReader is started to enqueue new samples to the samples queue.
9. The model executes an initialisation calculation if necessary (e.g. initialisation for weight normalisation).
10. The model adds further tasks to the operation list (e.g. more histograms).
11. +12. The training starts until a maximal number of steps is reached.

3.2.2 Load audio sequence and annotations

The AudioReader loads and prepares the data. It was built using the librosa (McFee et al. (2017)), numpy in version 1.13 and pandas in version 0.20 (McKinney (2010)) to handle audio, files and annotations.

Figure 3.3 shows a sequence diagram of the method to enqueue new samples to the samples queue. The DataRepository is used to select the subset of training files defined in settings (e.g. if only yodel from Unterwalden is used for training). The Annotions class provides functionality to create the global conditioning vector for a specific filename. In the following, the steps are described.

1. The AudioReader is executed asynchronous from the training procedure and controlled by the TensorFlow™s-coordinator component. The following steps are performed until a flag switches to zero.
2. The DataRepository is called to retrieve valid filenames.
3. From the annotations, the global conditioning is extracted.
4. A file is opened with librosa and stored in a numpy array.
5. The silence is trimmed by comparing the root mean squared energy (RMSE) per frame (default 2048) with a silence threshold. Librosa provides functionality to calculate the RMSE efficiently. It is derived from the fast Fourier transform (see Section 3.7). The amplitudes are squared and summated, and root square of this is the RMSE.
6. The audio is sliced and enqueued.
7. If used the global conditioning is enqueued.

Note that the implementation uses generators and therefore, differs slightly from the described steps. The enqueueing is controlled by the TensorFlow™-coordinator which ensures that the queue as a configurable amount of samples.

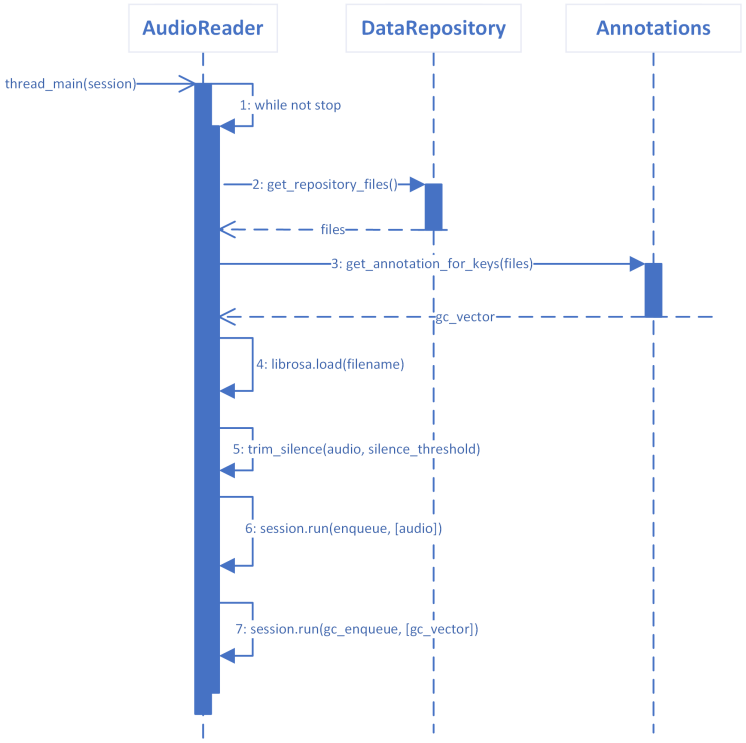


Figure 3.3: Sequence diagram of the workflow for training.

3.2.3 Generate samples procedure

The generate samples procedure is used to create new samples. It is described next:

1. The model is initiated using the factory.
2. The receptive field size denotes how many sample points are necessary to make one prediction. It is used to create the seed of this size as a tensor.
3. The model extends the computation graph and adds all calculations up and with the output layer.
4. The TensorFlow™ session is initiated.
5. The model is loaded from disk and all weights are set with their stored values.
6. The model is executed and creates a prediction.
7. The prediction is added to the seed and the result vector. The seed is cut from the beginning to have a length equal the receptive field size.
8. Step 7-9 are repeated until the result vector length equals a configurable size.
9. The mu-law decoding is applied to the outcome (see Section 3.2.4).
10. The decoded result is stored as WAV using librosa.

The seed is created as follows:

1. Random: A vector of size $n - 1$ is zeroed, and a random variable in range $[-1, 1]$ is added. The mu-law encoding is applied (see Section 3.2.4).
2. Audio File: Given a file name it is loaded as vector and trimmed using the silence threshold. The first n points are encoded with the mu-law encoding and returned as seed.

3.2.4 Raw audio signal transformation

WaveNet (van den Oord et al. (2016b)) and SampleRNN (Mehri et al. (2016)) both proposed to use the mu-law encoding as input processing step. It is described next.

A raw audio signal is usually stored as a sequence of 16-bit integer values per time-step. For a classifier to predict the correct value, 65'536 probabilities would be necessary to model all possible values. In van den Oord et al. (2016a) the μ-law transformation (ITU-T (1988)) was used followed by a non-linear quantisation to reduce it to 256 possible values (3.1). It is calculated as

$$f(x^{(t)}) = sign(x^{(t)}) \frac{\ln(1+\mu|x^{(t)}|)}{\ln(1+\mu)}, \tag{3.1}$$

where $-1 < x^{(t)} < 1$, $sign(\cdot)$ the sign of the number and μ denotes the quantisation value.

The decoding of the quantised signal $y^{(t)}$ is given by

$$x^{(t)} = \frac{1}{\mu} \cdot ((1+\mu)^{|s^t|} - 1), \tag{3.2}$$

where $s^{(t)} = 2 \cdot (\frac{y^{(t)}}{\mu}) - 1$.

3.3 Docker in Cloud

This thesis used GPUs provided by the Lucerne University of Applied Sciences and Arts and Switch[2]. Thereby, the training procedures were set up to run in docker images. Additionally, S3 like storage of Switch Engine[3] was used to store temporary results and logs. The whole setup is shown in Figure 3.4.

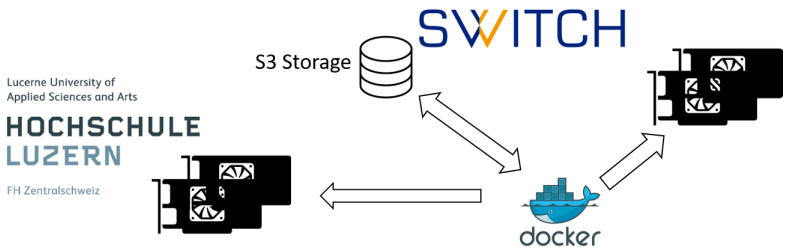


Figure 3.4: The training setup used in this thesis. The results were stored in the S3 like storage of Switch.

The docker images were developed in collaboration with Roland Christen[4] utilising docker-compose. Environment variables were used to configure the bucket name which determines where the model was stored. Three docker services were combined where the sync-image executes a setup script that creates a bucket using the s3cmd[5] or downloads a training state. After that, the sync-image starts a synchronisation service that synchronises the training state to Switch. The second app-service is waiting until the setup is done. After the setup has finished, it starts the experiment or loads a state and trains from there until it stops. The last tensorboard-service executes TensorBoard which is then accessible on a configurable port.

The described setup allowed to switch experiments at any time on a different GPU if needed at the cost of the download time for a bucket. Switch proved to have fast access timings, and all services were reliable through the whole thesis. For future projects, this setup allows to shift an experiment in the cloud at the price of a fee if more computational power is needed than available.

The docker-compose and image definitions are accessible as described in the Appendix A.4.2.

[2]https://www.switch.ch/engines/, accessed 15.01.2018
[3]https://help.switch.ch/engines/documentation/s3-like-object-storage/, accessed 15.01.2018
[4]https://www.hslu.ch/en/lucerne-university-of-applied-sciences-and-arts/about-us/people-finder/profile/?pid=1558, accessed 15.01.2018
[5]http://s3tools.org/s3cmd, accessed 15.01.2018

3.4 Output modelling

The model learn the distribution of data. Therefore, they have to be designed to output the parameters for the distribution used. In this thesis the Bernoulli triads (Section 3.4.1) and a mixture model of Logistic distributions (Section 3.4.2) were used. They are introduced next.

3.4.1 Bernoulli distribution

WaveNet uses the generalised Bernoulli distribution for its output with $K = 256$ classes to model a number between 0 and 255. E.g. a 4 will be represented by a class vector of [0 0 0 0 1 ... 0]. In machine learning, this is referred to one-hot encoding. Based on this representation 256 Bernoulli trials can be defined which represent the numbers between 0 and 255. First, the Bernoulli distribution is introduced.

The Bernoulli random variable y has a binary outcome from $\{0, 1\}$. Its probability density function is given by

$$f_y = p^y(1-p)^{1-y}. \tag{3.3}$$

Let a $y = [y_1\ y_2\ ...\ y_K]$ be the outcome of K independent Bernoulli trials, each with success probability p. The joint probability distribution of $y_1, y_1, ..., y_K$ is given by

$$f(y; p) = \prod_{i=1}^{K} p_i^{y_i}(1-p_i)^{1-y_i}. \tag{3.4}$$

A log transformation yields the cross-entropy function as shown in Equation 2.14 for one sample. Note that it is assumed that $y = [y_1\ y_2\ ...\ y_K]$ are independent and identically distributed (Watkins (2009)).

The distribution is often used with a softmax function which is applied in front of the loss function (used in van den Oord et al. (2016b)) to shift the values in a range $[0, 1]$. Additionally, it boosts the probability for the class with the highest value while the other expectations are lowered. The softmax function is given by

$$x = \frac{e^x}{\sum_{i=1}^{n} e^{x_i}}, \tag{3.5}$$

where $x = [x_1 x_2 \cdots x_n]$ (Martìn Abadi et al. (2015)).

3.4.2 Mixture Model of Discrete Logistic Distributions

For experiments with SampleRNN mixture models of logistic distribution were tested (see Section 4.2.2). The implementation was published by Salimans et al. (2017) and contains a numerically stable procedures to calculate the log-probability. It was slightly adapted for this thesis to work with one channel only. In PixelCNN three colour channels are used per pixel (RGB). The likelihood for the mixture model of logistic distributions is explained next.

It is assumed that there is a sample point y with a continuous distribution, which is rounded to its nearest 8-bit representation. Hence, the mu-law transformation described in Section 3.2.4 is applied. A mixture of K logistic distributions is chosen to model the predicted value x. For all values x, excepting the edge cases 0 and 255, it is defined as

$$y \sim \sum_{i=1}^{K} \pi_i \cdot logistic(\mu_i, s_i) \tag{3.6}$$

$$p(x|\pi, \mu, s) = \sum_{i=1}^{K} \pi_i \cdot \left(g\big(\frac{x + 0.5 - \mu_i}{s_i}\big) - g\big(\frac{x - 0.5 - \mu_i}{s_i}\big) \right) \tag{3.7}$$

where $g(\cdot)$ denotes the sigmoid function. For the edge case 0 replace $(x - 0.5)$ by $-\infty$, and for 255 set $(x + 0.5)$ by $+\infty$.

3.5 Autoregressive Models

Models are denoted as autoregressive if a prediction from a specific time series is regressed based on the previous values from the same time series. The WaveNet architecture is described in Section 3.5.1 and SampleRNN in Section 3.5.2.

3.5.1 WaveNet

WaveNet is a CNN that consists of dilated convolutions only (see Section 2.4.2). The architecture was proposed by van den Oord et al. (2016b), and the implementation used in this thesis originates from GitHub[6]. It was adapted to fit the workflow as suggested in 3.2.1. WaveNet is best explained by Figure 3.5 which shows how the dilated convolution layers are functioning. The default implementation of WaveNet uses dilations of $2^0, 2^1, 2^2, ..., 2^9$ and repeats them five-time (50 WaveNet Blocks). This way a receptive field is created that stretches for 5146 sample points (321 ms with a sample rate 16'000). The receptive field refers to the number of values of an input sample which are directly involved in the calculation of the prediction.

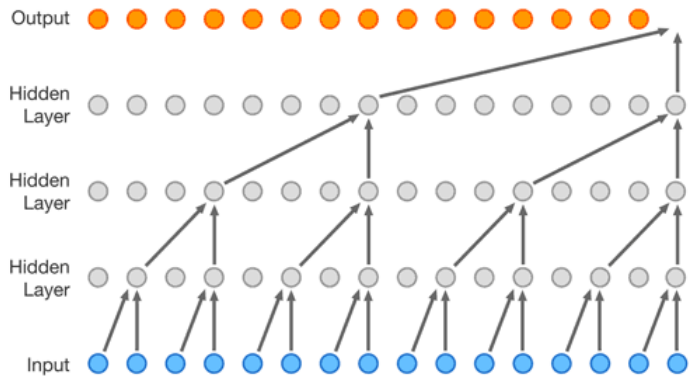


Figure 3.5: Three hidden layers and the output layer are visualised which are dilated convolutions with a dilation factor of $2^0, 2^1, 2^2, 2^3$. The filter size is set to 2. The black arrows show which nodes of lower layer contribute to a node. Thereby, from top to bottom a receptive field (blue) is created which directly influences the predicted sample point (van den Oord et al. (2016b)).

The WaveNet architecture is shown in Figure 3.6. It consists of the following component:

1. **Input batch, gc vectors:** The inputs of the network include of the "input batch" which are m audio samples of a certain sample size n and "gc vectors" that contains a global conditioning vector for each sample. The "gc vectors" are only defined when global conditioning is used. In this thesis, the "gc vectors" are given by the annotations of the audio samples, introduced in Section 3.1.1.

2. **Causal Convolution:** The "Causal Convolution" layer is a convolution with stride and dilation set to one but a certain filter width. For the default implementation, the value was set to 32. The padding strategy was set to "valid" (reference implementation). Note that after the convolution there is none non-linearity applied.

3. **WaveNet-Blocks:** The "WaveNet-Blocks" consist of one to many WaveNet Block which are dilated convolutions. Each has a specific dilation sizes but the same block filter width. The WaveNet-Block is described later in this section. van den Oord et al. (2016b) proposed to use $5 \times [2^0, 2^1, 2^2, ..., 2^9]$ dilations. The receptive field and the output size of these layers are calculated by

$$\text{receptive field} = \sum_{d \in dilations} d \cdot (\text{block filter width} - 1) \tag{3.8}$$

$$\text{and} \quad \text{output size} = \text{sample size} - \text{receptive field}, \tag{3.9}$$

[6] https://github.com/ibab/tensorflow-wavenet, accessed 15.01.2018

where the dilations are configurable. Each WaveNet Block contributes to the output of these blocks over a skip connection of size [batch size, output size, skip channels]. All these skip connection's results are summated before applying a sigmoid-activation. Its output is then handed over to the next layer.

4. **1x1 Convolution:** This convolution layer has stride, dilation and filter width set to one. The strategy "same padding" is used which means that input size = output size. After this convolution, a ReLu activation is applied. The ReLu activation is defined as

$$a_i^{(l)} = max(0, z_i^{(l)}). \tag{3.10}$$

5. **1x1 Convolution:** This is again a convolution layer with stride, dilation and filter width set to one. But there is no non-linearity applied afterwards. It is the last layer and transforms the output dimensions to a configurable size (the reference implementation sets the dimensions to 256).

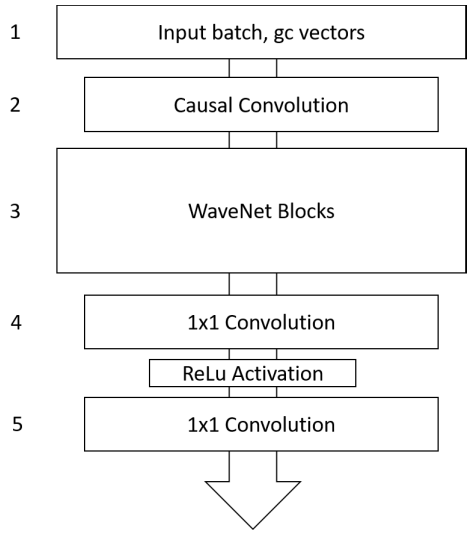


Figure 3.6: The WaveNet architecture is in abstract form. Note that the WaveNet Blocks consist of one to many WaveNet Block.

All configurable settings are defined in Table 3.1.

Configuration	**Description**
Initial filter width	Used in "Causal Convolution" layer
Block filter width	Convolution filter width for WaveNet-Blocks
Dilation layers	Dilations of each WaveNet-Block
Residual Channels	Dimensions of input batch and dense output
Dilation Channels	Dimensions of dilated convolutions
Skip Channels	Dimensions of skip connection
Quantisation Values	Number of mu-law encoding to be used for output size with Bernoulli trials distribution
Receptive Field	Calculated as shown in Equation 3.8

Table 3.1: Description of the parameters for the model WaveNet.

WaveNet Block

The WaveNet block applies dilated convolution to the input. The input is of size [batch size, input size, residual channels] where input size ≥ dilated size ≥ output size. Figure 3.7 shows the WaveNet Block schematically. The filter and gate convolution are both convolution layer with a defined dilation and a stride set to one. The block filter width defines the convolution filter width. The gate convolution uses a sigmoid activation function while the filter convolution applies a tanh activation. If global conditioning is used, then the gc vectors are first transformed by 1x1 convolution to be of the same dimension as the filter and gate outcome then added in front of the activations. The result of the tanh and the sigmoid activation is multiplied using an element-wise multiplication.

To have the skip connection tensor of the same size as the final output of all blocks the tensor is cut from the end. Thereby, the tensor [batch size, dilated size, residual channels] is shortened to [batch size, output size, residual channels]. 1x1 convolution is followed to transform the residual channels to skip channels.

The dense output consists of the output from the dilated convolutions and from the input which is summed up. Because the input tensor is larger than the dense output, it has to be shortened and thereby, is cut as described for the skip connection. The input tensor [batch size, input size, residual channels] is shortened to [batch size, dilated size, residual channels] then summed with the results of the dilation operations.

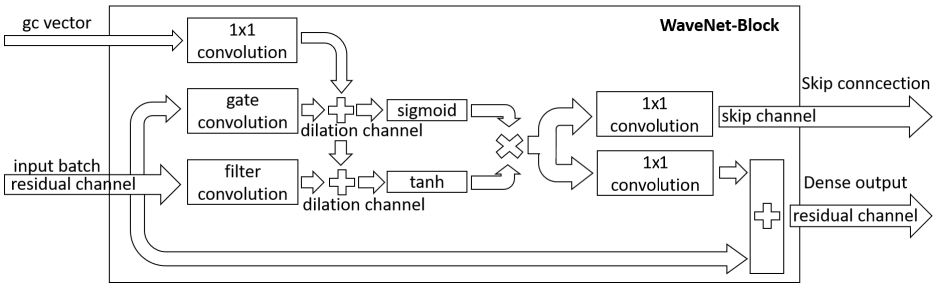


Figure 3.7: WaveNet Block explained. The input are the input batch and conditioning. The output is given by the dense output and the skip connection. In between the calculations are shown.

Sampling

Two techniques are provided by the developers to generate samples. They were adapted slightly to work with the generate samples procedure described in Section 3.2.3. Both methods are compatible with conditioning and are described in the following:

- **Random sampling:** The seed is set with a vector of the size of the receptive field which is zeroed. The last entry of the vector is set with a random variable in range $[-1, 1]$.
- **From audio:** A piece of the size of the receptive field of an audio file is provided as seed.

3.5.2 SampleRNN

SampleRNN is an n-tier architecture network based on RNN where the output of each tier is used as conditioning for the next. Figure 3.8 shows the network architecture schematically. Mehri et al. (2016) proposed the architecture in variants one, two and three tier, but stated that the three-tier worked best. It was reimplemented in this thesis and is explained next.

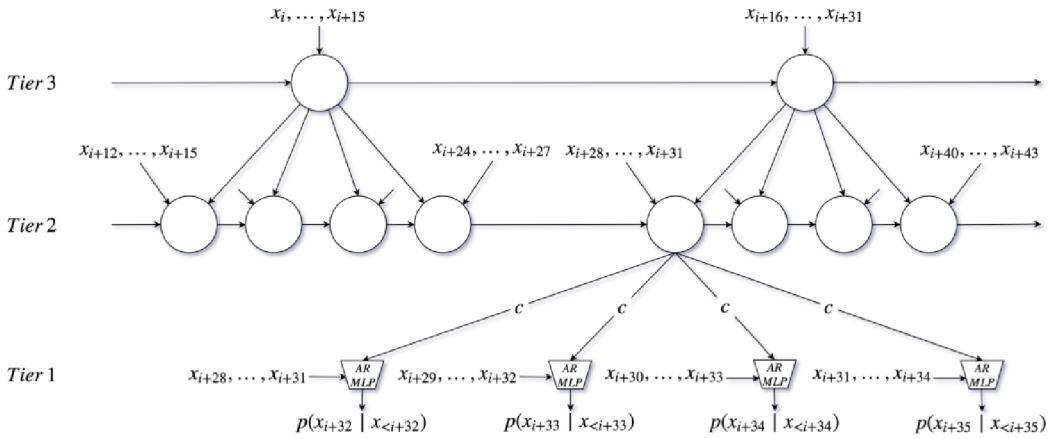


Figure 3.8: The top third tier uses a first frame part of a sequence and outputs conditioning for the second tier. The second tier takes the second frame and the conditioning to calculate the first tier conditioning. The first tier consists of 3 fully connected layers and uses the conditioning and the third frame to predict the next sample point (Mehri et al. (2016)).

Consider a sample as a sequence of three frames given as sample=(FS1, FS2, FS3). The top third tier uses a first frame FS1 and outputs conditioning for the second tier. The second tier takes the second frame FS2 and the conditioning to calculate the first tiers' conditioning. The first tier calculates now the output from the last frame FS3 and the conditioning of the second tier.

The second and third tier are schematically shown in Figure 3.9. Note that for the third-tier the conditioning is unnecessary. The blocks "weight norm" refer to a fully connected layer which applies weight normalisation. The concept is explained later in the section. The RNN layer consists of GRUs described in Section 2.3.1. The first tier is schematically shown in Figure 3.10. An one-hot encoding is applied on the quantised frame. Then it is transformed with convolution and filter size equal the FS2. It follows a weight normalised layer and summation with the conditioning. Mehri et al. (2016) proposes to use two weight normalised layers with ReLu activation and the last without non-linear activation. The outcome is the prediction of the next sample point.

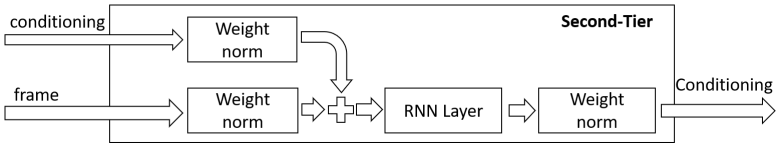


Figure 3.9: The figure schematically shows the second tier of SampleRNN.

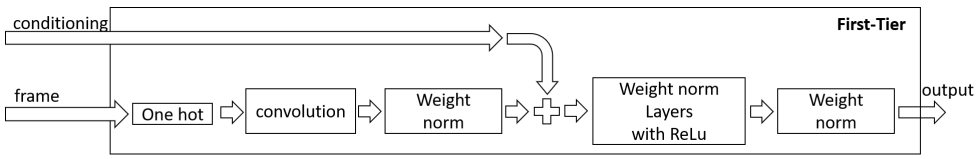


Figure 3.10: The figure schematically shows the first tier of SampleRNN.

All configurable settings are defined in Table 3.2.

Configuration	**Description**
Tier-Number	Controls how man tier are used. It has been set to 3 for all experiments
Number of Layers	Controls the number of RNN layers in the second and third tier. It has been set to 1.
Last tier Layers	Controls the amount of Weight normalisation layers in the first tier. It was set to 3.
3-Tier frame size	Frame size that is used for the third tier.
2-Tier frame size	Frame size that is used for the second tier.
1-Tier frame size	Frame size that is used for the first tier.
Quantisation Values	Number of mu-law encoding
Layer Dimensions	The number of neurons in each layer. It was set to 1024.

Table 3.2: Description of the parameters for the model SampleRNN. All mentioned default values were proposed by Mehri et al. (2016).

Weight Normalisation

Weight normalisation was proposed by Salimans and Kingma (2016) and was used for SampleRNN (Mehri et al. (2016)). It is described next. A vanilla neuron is given by $a = g(\Theta \cdot x + b)$ where x is the input, b the bias term, Θ the weights, g a non-linear activation function such as the sigmoid and a the activation of the neuron. Salimans and Kingma (2016) calculates the weight now as

$$\Theta = \frac{s}{\|v\|} \cdot v, \tag{3.11}$$

where v is of the same dimensions as Θ, s is a scalar and $\|\cdot\|$ denotes the Euclidean norm. Thereby, the Euclidean norm of the weight is fixed as $\|\theta\| = s$. The gradients are found with

$$\nabla\mathcal{L}_s = \frac{\nabla\mathcal{L}_\Theta \cdot v}{\|v\|}, \tag{3.12}$$

$$\nabla\mathcal{L}_v = \frac{s}{\|v\|} \cdot \nabla\mathcal{L}_\Theta - \frac{g \cdot \nabla\mathcal{L}_s}{\|v\|^2} \cdot v. \tag{3.13}$$

Salimans and Kingma (2016) proposes to initialise the variables with a single mini batch x. Let the pre-activation of a node be $t = \frac{v}{\|v\|} \cdot x$ and μ_t and σ_t be the mean and standard deviation of the pre-activation t over the mini batch. The bias b and scale g are initialised as follows

$$g = \frac{1}{\mu_t}, \quad b = \frac{-\mu_t}{\sigma_t}. \tag{3.14}$$

This ensures that all features initially have zero mean and unit variance (Salimans and Kingma (2016)).

3.6 VAE Acoustic

The VAE Acoustic is a model architecture that uses VAE (see Section 2.6.2) where an adapted WaveNet (see Section 3.6.3) is used as an encoder and a deconvolution WaveNet as a decoder (see Section 3.6.1). The idea originates from van den Oord et al. (2017b) where WaveNet was applied similarly.

The VAE architecture is schematically shown in Figure 3.11. The encoder WaveNet (see Section 3.6.1) calculates from the input the distribution of the latent variables. In the next step, the latent variable vector is calculated (see Section 3.6.2) using the reparametrisation trick introduced in Section 2.6.2. With the latent variable vector, the decoder Deconvolution WaveNet (see Section 3.6.3) reconstructs the input.

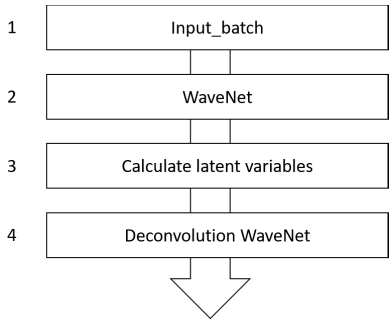


Figure 3.11: Architecture of the VAE Acoustic.

VAE Acoustic has been implemented so it is configurable what architecture should be used as encoder or decoder. While construction, the VAE Acoustic calls the model factory (see workflow description in Section 3.2.1) and transmits the corresponding model parameters. The only requirement for the decoder is that its output is of dimension [batch size, latent variable size, 2] where 2 refers to the parameters μ and σ of the Gaussian distribution of the latent variables. All configurable settings are defined in Table 3.3.

Configuration	**Description**
encoder	Corresponding parameters to the encoder model.
decoder	Corresponding parameters to the decoder model.
Quantisation Values	Number of mu-law encoding to be used for output size with Bernoulli distribution
Sample rate	Sample rate of the audio file. This is used to output samples while training in TensorBoard.

Table 3.3: Description of the parameters for the model VAE.

3.6.1 WaveNet with strides and dilations

WaveNet in its default implementation uses dilated convolution layers only. The maximal compression rate per layer is given by the reduction of dimension caused by the convolution (see Section 2.4, "same padding"). Given the dilation factor d and the block filter width k, the dimensionality reduction is given by $r = d \cdot (k - 1)$ (see Section 2.4.2). In WaveNet, d was chosen in a way that it is dependent on a connected receptive field. Changing d might lead to a segmented area which conflicts with the idea of WaveNet. Furthermore, d has to be increased dramatically to reduce a sample size of, e.g. 16'000 to an output size of, e.g. 512. Therefore, strided convolution was introduced (see Section 2.4.1) that compresses the dimension by a factor s. Note that van den Oord et al. (2017b) used strided convolutions without dilations in his approach. Even though it is technically possible to use strides and dilation at the same time, it was not implemented. To the best of the author's knowledge, there was no application known that applied this combination.

Additionally to strided convolution, batch normalisation was introduced which is described later in this section. Figure 3.12 shows the changes (orange) to the WaveNet architecture. It has been decided to add batch normalisation only at some points, so the number of additional weights is limited. The causal convolution applies batch normalisation in front of the non-linear activation similar to step 4.

After the WaveNet Blocks, an extra convolution layer was added with a configurable filter width to control the encoder output size more granularly. The WaveNet Block was changed (orange) as shown in Figure 3.13. The gate and filter convolutions were extended so strides or dilations can be used. Batch normalisation is applied only once before the summation with the input which yields the dense output.

The output of the network was modelled to have dimension [batch size, latent variable size, 2] where 2 is related to the Gaussian distribution of the latent variables $z \sim \mathcal{N}(\mu, \sigma)$. All configurations are described in Table 3.4.

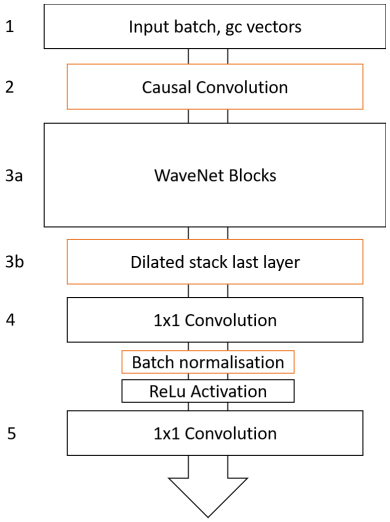


Figure 3.12: WaveNet architecture extension with batch normalisation

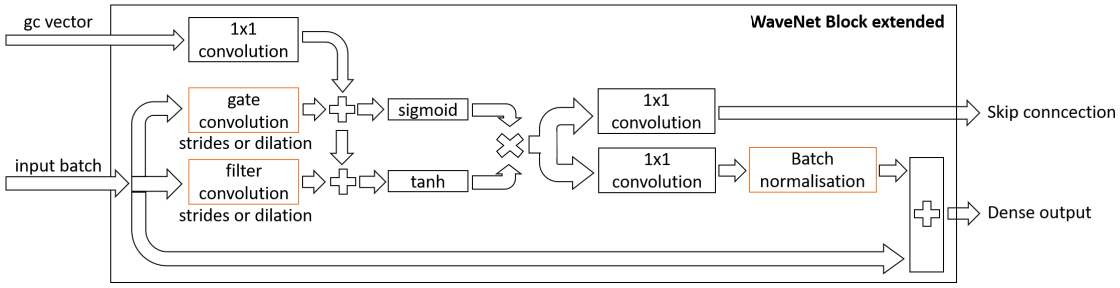


Figure 3.13: WaveNet Block extension with batch normalisation. The orange marked boxes show the changes to the vanilla implementation.

Configuration	**Describition**
Initial filter width	Used in "Causal Convolution" layer
Block filter width	Convolution filter width for WaveNet-Blocks
Last layer filter width	Convolution filter width for the dilated stack last layer
Dilation layers	Dilations of each WaveNet-Block
Strides layers	Strides of each WaveNet-Block. Note that only dilation or strides can be used.
Residual Channels	Dimensions of input batch and dense output
Dilation Channels	Dimensions of dilated convolutions
Skip Channels	Dimensions of skip connection
Quantisation Values	Number of mu-law encoding to be used for output size with Bernoulli distribution
Receptive Field	Calculated as shown in Equation 3.8

Table 3.4: Description of the parameters for the model WaveNet.

3.6.2 Latent variables

It has been chosen to use the Gaussian distribution to model the latent space. The encoder outputs a tensor of size [batch size, latent variable size, 2]. To sample the latent variable vector the reparametrisation trick is applied as described in Section 2.6.2. Figure 3.14 schematically shows how the computation graph is built. The output is of size [batch size, latent variable size].

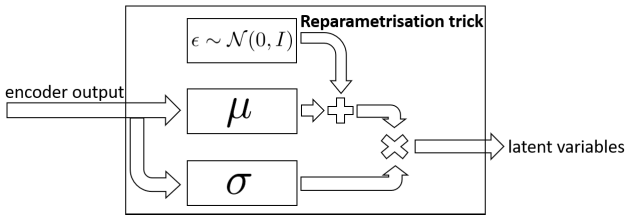


Figure 3.14: The figure schematically shows how the latent variables are retrieved.

3.6.3 Deconvolution WaveNet

Deconvolution WaveNet was proposed in this thesis as a counterpart to the WaveNet architecture. The underlying design was constructed as for WaveNet with batch normalisation (see Section 3.6.1). It is composed of the following steps (Figure 3.15):

1. **Latent variables:** The inputs of the network are of form [batch size, latent variable size].

2. **Causal deconvolution:** The layer is a deconvolution with stride and dilation set to one and a filter of size "initial filter width". The padding strategy "valid" is used which causes the output size to grow. Afterwards, batch normalisation is applied.

3. **Deconv Blocks:** The "Deconv Blocks" consists of one or many "Deconv Block". They are strided or dilated deconvolutions of specific stride or dilation sizes and a constant block filter width. The Deconv Block is described in the next section.

4. **Dilated stack last layer:** This layer is added to adjust the output size. It is a deconvolution with stride and dilation set to one. The padding strategy is "valid". Batch normalisation is applied here but not a non-linear activation function.

5. **1x1 Deconvolution:** The deconvolution layer has stride, dilation and filter width set to one. The strategy "same padding" is used which means that input size = output size. After this convolution, batch normalisation and a ReLu activation are applied.

6. **1x1 Deconvolution:** This is again a deconvolution layer with stride, dilation and filter width set to one. No non-linearity or batch normalisation is involved here. It is the last layer and transforms the output dimensions to a configurable size (e.g. the number of Bernoulli triads).

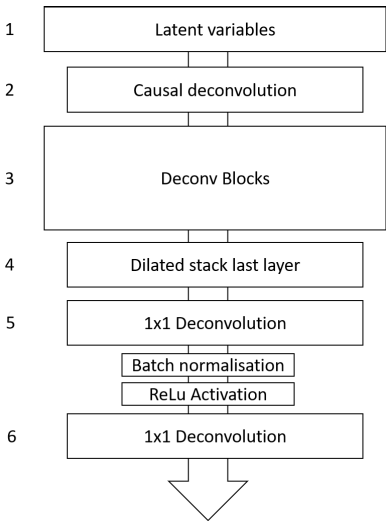


Figure 3.15: Deconvolution WaveNet architecture

The configurations for this architecture are given in Table 3.5.

Configuration	**Describtion**
Initial filter width	Used in "Causal Convolution" layer
Block filter width	Convolution filter width for WaveNet-Blocks
Post filter width	Convolution filter width for the dilated stack last layer
Dilation layers	Dilations of each WaveNet-Block
Strides layers	Strides of each WaveNet-Block. Note that only dilation or strides can be used.
Residual Channels	Dimensions of input batch and dense output
Dilation Channels	Dimensions of dilated convolutions
Quantisation Values	Number of mu-law encoding to be used for output size with Bernoulli distribution

Table 3.5: Description of the parameters for the model WaveNet.

Deconv-Block

The deconvolution block was designed in this thesis as a complement of the reconstruction phase to the WaveNet Block. It has two inputs: latent input and a dense input. It is schematically shown in Figure 3.16.

The latent input is the original latent variables that are used as conditioning for each Deconv Block. First, a 1x1 deconvolution is applied that is followed by a sigmoid. It is then applied to the dilation channels of the dense input as an elementwise multiplication. The idea is that it stretches or shrinks the dense input.

The dense input is the information that is reconstructed. After a 1x1 deconvolution, it is normalised in a range [-1, 1] and multiplied by the latent channel.

The gate and filter deconvolutions are applied with a specific stride or dilation size and the block filter width. After this, batch normalisation is calculated for both, and the gates are summed up. The result is added to the dense input and returned as dense output.

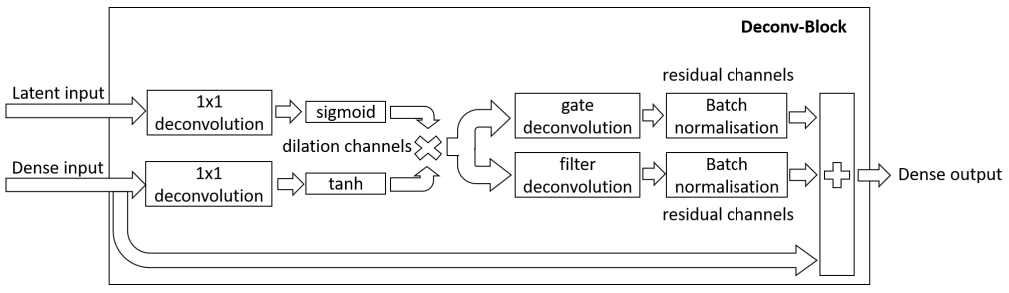


Figure 3.16: Deconvolution Block

Batch normalisation

Sønderby et al. (2016) suggested applying batch normalisation for deep VAEs which is the reason that it was applied in this thesis. TensorFlow™ provides functionality for it. Ioffe and Szegedy (2015) proposed batch normalisation for work with deep neural networks. They observed that *the distribution of each layer's inputs changes during training as the parameters of the previous layers change*(Ioffe and Szegedy (2015)). Therefore, lower learning rates must be used which increases the training time. This change of the distribution is denoted as internal covariate shift and is tackled by fixing the distribution of the layer inputs as the training progresses. This is achieved by normalising the inputs over the mini-batch for each dimension. Batch normalisation is integrated into the model architecture as a feed forward operation. They observed that 14 times fewer training steps were necessary to achieve the same performance. Furthermore, it eliminated the need for drop-out in some cases. Batch normalisation is explained next.

Batch normalisation operates on a mini-batch of m samples on each of the d dimensions. $x_j^{(i)}$ denotes the j-th dimension of the i-th input sample. The normalised values are denoted with $\hat{x}_j^{(i)}$ and given by

$$\hat{x}_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{\sqrt{\sigma_j^2 + \zeta}}, \tag{3.15}$$

where $j = 1, 2..., d$ and ζ denotes a tiny number to avoid a division by zero. The mini-batch mean μ_j is calculated as

$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}, \tag{3.16}$$

and the mini-batch variance σ_j^2 is given by

$$\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)} - \mu_j)^2. \tag{3.17}$$

Finally, the normalised value is scaled by γ_j and shifted with β_j which yields

$$a_j^{(i)} = \gamma_j \cdot \hat{x}_j^{(i)} + \beta_j. \tag{3.18}$$

Note that the input values are normalised under a Gaussian distribution $\mathcal{N}(0, 1)$, and then stretched and shifted by γ_j and β_j. Therefore, the assumption is made that the input values are following a normal distribution.

For back-propagation, the chain rule is used to find the gradients with

$$\frac{\partial \mathcal{L}}{\partial \hat{x}_j^{(i)}} = \frac{\partial \mathcal{L}}{\partial a_j^{(i)}} \cdot \gamma_j, \tag{3.19}$$

$$\frac{\partial \mathcal{L}}{\partial \sigma_j^2} = \sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{x}_j^{(i)}} \cdot (x_j^{(i)} - \mu_j) \cdot (-\frac{1}{2}) \cdot (\sigma_j^2 + \zeta)^{-\frac{3}{2}}, \tag{3.20}$$

$$\frac{\partial \mathcal{L}}{\partial \mu_j} = \left(\sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{x}_j^{(i)}} \cdot \big(-\frac{1}{\sqrt{\sigma_j^2 + \zeta}}\big)\right) + \frac{\partial \mathcal{L}}{\partial \sigma_j^2} \cdot \frac{1}{m}\sum_{i=1}^{m} -2 \cdot (x_j^{(i)} - \mu_j), \tag{3.21}$$

$$\frac{\partial \mathcal{L}}{\partial x_j^{(i)}} = \frac{\partial \mathcal{L}}{\partial \hat{x}_j^{(i)}} \cdot \frac{1}{\sqrt{\sigma_j^2 + \zeta}} + \frac{\partial \mathcal{L}}{\partial \sigma_j^2} \cdot \frac{2(x_j^{(i)} - \mu_j)}{m} + \frac{\partial \mathcal{L}}{\partial \mu_j} \cdot \frac{1}{m}. \tag{3.22}$$

Finally, the gradients for γ_j and β_j are calculated as

$$\frac{\partial \mathcal{L}}{\partial \gamma_j} = \sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{a}_j^{(i)}} \cdot \hat{x}_j^{(i)} \tag{3.23}$$

$$\text{and} \quad \frac{\partial \mathcal{L}}{\partial \beta_j} = \sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{a}_j^{(i)}} \tag{3.24}$$

(Ioffe and Szegedy (2015)).

3.7 Analysis with Fast-Fourier Transform

The fast Fourier transform (FFT) decomposes a given signal into its frequencies. This is helpful when working with audio signals as the theoretical frequencies of notes are known (see Table 1.2 in Section 1.7). Additionally, the FFT exposes the amplitude of each frequency where frequent notes are expected to align with frequencies having higher amplitudes. Together, it is possible to analyse the interval between notes for a given audio sample. In the following a short introduction to the FFT is made.

Working with WAV-files the input and output of the models are discrete-time signals. It means it is represented as sequence $x = \{x[n]\}$ with $-\infty < n < \infty$ and the delay between each number is given by 1. Figure 3.17 shows how an analogue signal is transformed to a discrete-time signal. Nevertheless, the sequence order n of a data point $x[n]$ is referred to as time. The basic concept of the Fourier transform is that a signal in the time domain can be described by a sum of sinusoidal of different amplitude and phase in the frequency domain.

The following explanation are taken from Cattin (2012). The discrete Fourier transform (DFT) transfoms a discrete time signal of finite duration from time domain to frequency domain, and backwards. A continuous sinusoidal signal with a period T_c is given by

$$x(t) \quad = \quad A \cdot \cos \omega_c \cdot t = A \cdot \cos\left(\frac{2\pi}{T_c} \cdot t\right). \tag{3.25}$$

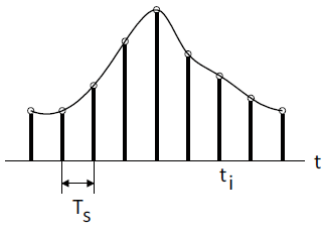
where $t \in \mathbb{R}$, A denotes the amplitude and ω_c denotes the frequency of the continuous signal. If the same signal is sampled with a sample rate T_s then it given by

$$x[t] \quad = \quad A \cdot \cos \omega \cdot n = A \cdot \cos\left(\frac{2\pi \cdot T_s}{T_c} \cdot n\right). \tag{3.26}$$

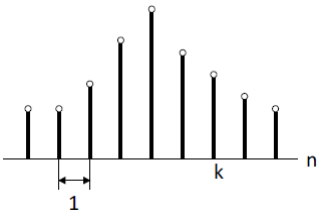
where $n \in \mathbb{N}$. Thereby, the discrete frequency is computed as

$$\omega = \frac{2\pi \cdot T_s}{T_c} \tag{3.27}$$

and is a dimensionless relation between the signal frequency and the sample rate. Figure 3.17 shows the relationship between a continuous-time and a sampled discrete-time signal. Given a sequence of



(a) A signal that is sampled to a discrete-time sequence on a time interval of T_s. Note that the axis is given as time t.



(b) The discrete signal runs on a axis N for number of points. The time between one point and another is no more determinable.

Figure 3.17: The left picture shows a analogue signal with given sampling rate T_s. The left side shows the discrete signal that does no more contain the time information(Cattin (2012)).

N samples the DFT analysis and synthesis pair is computed as

$$X[k] \quad = \quad \sum_{i=0}^{N-1} x[i] W_N^{k \cdot i}. \quad 0 \leq k \leq N-1, \tag{3.28}$$

$$x[i] \quad = \quad \frac{1}{N} \sum_{k=0}^{N-1} X[k] \cdot W_N^{-k \cdot i}, \quad 0 \leq i \leq N-1, \tag{3.29}$$

where the twiddle coefficients are given by

$$W_N = e^{-j\frac{2\pi}{N}} \tag{3.30}$$

(Cattin (2012)).

In the following the basic theorem is described and some more insight to FFT is given.

Shannon sampling theorem A continuous-time signal $x(t)$ with frequencies no higher than f_{max} can be reconstructed exactly from its samples $x[t] = x(n \cdot T_s)$, if the samples are taken at a rate $f_s = 1/T_s$ that is greater than $2 \cdot f_{max}$.

f_s is denoted as Nyquist rate and it means that at least two samples are required per period for a signal so it can be reconstructed. Therefore, the discrete frequency $\omega = 0, \frac{2\pi}{N}, \frac{2\pi}{N} \cdot 2, \ldots, \frac{2\pi}{N} \cdot n, \ldots, \frac{2\pi}{N} \cdot (N-1)$ is zero or a multiple of its fundamental frequency.

To understand k a complex exponential signal $x[n]$ is defined with a frequency of $\omega_r = r \cdot \frac{2}{N}$. It is given by

$$x_r[n] = A \cdot e^{j \cdot \frac{2\pi \cdot r}{N} \cdot n}, \tag{3.31}$$

where $0 \leq k \leq N-1$. Applying the signal to the Equation 3.28 if follows

$$X[k] = \sum_{i=0}^{N-1} A \cdot e^{j \cdot \frac{2\pi \cdot r}{N} \cdot i} \cdot A \cdot e^{-j \frac{2\pi}{N} \cdot k \cdot i}, \quad 0 \leq k \leq N-1,$$

and reformulated

$$X[k] = \sum_{i=0}^{N-1} A \cdot e^{j \cdot \frac{2\pi \cdot i}{N}(r-k)}, \quad 0 \leq k \leq N-1. \tag{3.32}$$

Given $r \in \mathbb{N}$ then means that if there are an integer number of cycles in the sequence of length then it states

$$X[k] = \begin{cases} N \cdot A & \text{if } (r-k) \text{ is zero or a multiple of } N \\ 0, & \text{otherwise.} \end{cases} \tag{3.33}$$

Therefore, if $k = r$ then $X[k]$ contains the amplitude multiplied with the sample points N. By a division through N the amplitude of the frequency is returned. Now the transformation of the discrete frequency to the frequency of the real signal is given by

$$\omega_k = \frac{2\pi \cdot k}{N} \rightarrow \Omega_k = \frac{k}{N} \cdot f_s, \quad 0 \leq k \leq N-1. \tag{3.34}$$

For summary, the DFT reveals to used frequencies and their amplitude.

Chapter 4

Result

This chapter presents the results and experiments conducted in this thesis. Section 4.1 describes the results of tests with WaveNet. In Section 4.2, results of the SampleRNN model are described, and Section 4.3 contains the results of experiments with VAEs. A summary is presented at the end of each section.

In conclusion, the State-of-the-Art analysis (see Section 2.7.4) identified two approaches that seemed promising to the task of yodel music synthesis: WaveNet and SampleRNN. Both models are autoregressive and make a prediction one step ahead. Such models can produce samples of variable size. Contrary to this, VAE predicts total samples at once. The size is fixed and cannot be changed. Both model types optimise the negative log-likelihood(NLL), as introduced in Section 2.2.2. However, the NLL of a sample with size one is not directly comparable with a sample of larger size. In addition to this, most published approaches for music synthesis do not report a score such as the NLL that is optimised (Nayebi and Vitelli (2015), van den Oord et al. (2016a), van den Oord et al. (2017b)). Instead, preference or opinion scores, where an audience expresses a preference for a specific model over another, are used to determine a models' performance. In this thesis, the NLL is used to visualise the training progress of a particular model and to determine whether more training could lead to an improvement. It is batch size and data dependent and thus, it has been decided to calculate the mean over randomly chosen results out of a specific range of training steps. So progress while training can be monitored from region to region. The mean NLL is not used to compare the performance of models. It has been decided to apply a qualitative analysis with experts to evaluate the final performance as suggested in the literature.

4.1 WaveNet

This section contains the tests and results of the model architecture WaveNet (see Section 3.5.1). Experiments of Section 4.1.1 and 4.1.2 used a limited data corpus where for examination in Section 4.1.3 the whole data corpus, as described in 3.1, was available. An analysis of the best WaveNet models is given in Section 4.1.6.

4.1.1 WaveNet Default

This experiment was proposed to test the reference implementation of WaveNet on GitHub[1]. The code was used as it is and was extended minimally to process the audio files of the data corpus. At the start of the experiment the data corpus contained the following music:

- yodel: 3.5 h
- Yodel with text: 5.5 h
- Yodel with text and instruments: 5 h

All available music was used for training. The model is described in Section 3.5.1.

[1] https://github.com/ibab/tensorflow-wavenet, accessed 15.01.2018

Setup

The proposed default parameters of the reference implementation are given in Table 4.1 for the optimiser and Table 4.2 for the model. The batch size is by default set to 1 with a sample size of 5 seconds (see table 4.3). It is optimised to run on a GPU with 12 GB RAM. The experiment was executed on a Nvidia M40 GPU with 12 GB RAM. Tests with different learning rates revealed that the default settings worked best with the given data.

Optimiser Parameters	
Optimiser	Adam
Decay rate β_1	0.9
Decay rate β_2	0.9
Learning rate	10.0^{-3}
Loss function	Softmax with cross-entropy
L_2 Regularisation strength	0

Table 4.1: Hyper-parameters related to the optimiser (see Section 2.5.3) used in experiment WaveNet Default.

WaveNet	
Initial filter width	32
Block filter widht	2
Dilation layers	5x [1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
Residual Channels	32
Dilation Channels	32
Quantisation Values	256
Skip Channels	512
Receptive Field	5146($\sim$ 321 ms)
Output Modelling	Bernoulli distribution with $K = 256$

Table 4.2: The table describes the properties of the model of experiment WaveNet Default. In total, the model contains a causal convolutional layer, 50x dilated convolutional layers and two post-processing convolutional layers. In total, the model consists of 53 layers. The receptive field of the model covers 32 ms.

Input Processing	
Sample size	80000(5 s)
Sample rate	16000
Batch size	1
Silence threshold	0.3

Table 4.3: Parameters of input processing to experiment Wavenet Default. The default settings are optimised to a GPU with 12 GB RAM.

Results and Samples

The model was trained 33 days in total. The progress was regularly observed by generating new samples every week. The models and samples were stored in a data store at HSLU (see Appendix A.4.3). Figure 4.1 shows the learning curve for the experiment from step 0 to 1.5 million training steps. The mean NLL was calculated and is reported in Table 4.4. The model reached a mean NLL of 2.29. From the trajectory of the learning curve, it might well be that more extended training could still improve the model's performance. However, according to the mean NLL, progress is quite slow.

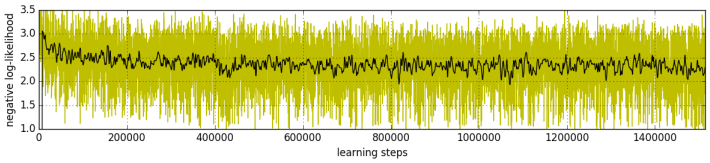


Figure 4.1: The learning curve for experiment WaveNet Default. Yellow shows the reported negative log-likelihood at each step (downsampled to 100'000 events). Black is a smoothed value to make progress apparent.

Training steps(in k)	**NLL Mean**	**NLL Std.**	**NLL Min**	**NLL Max**	**Sample**
450-550	2.38	0.42	1.12	3.22	generated_yodelling_494350.wav
900-1000	2.31	0.43	0.96	3.24	generated_yodelling_925750.wav
Last 50	2.29	0.52	0.97	3.15	generated_yodelling_1514700.wav

Table 4.4: Mean, Variance, Minimum and Maximum are derived from 200 random samples drawn from the region of the training steps given in the first column. For this experiment with WaveNet Default, the batch size was set to 1. See Appendix A.2 to get access to the samples.

The first sample was generated using a random seed after 80k steps. Noise dominates the audio signal, but there is already a pattern in it that is not further identifiable. The next sample was created with the model at step 224k where something voice-like is recognisable. After eight days of training (sample 372k) it is apparent that the model produces singing and not only voices. The evolution of the described stages is well explained by the progress of the NLL in Figure 4.1. Between 450k and 550k training steps, the NLL was at 2.38 and until the end improved by only -0.09. However, when listening to the samples generated during these training steps, the differences are tremendous. The sample of 450-550k sounds blurred as if some music is playing somewhere, but noise predominates it. In comparison, the sample of 900-1000k contains audibly better recognisable voices. The sound is much brighter and less noisy. The timber of samples of the model from the end of training is even more defined and, according to the authors' opinion, audibly better than the output of previous models. Despite the fact that the improvement according to NLL is almost negligible (only -0.02) it represents considerable progress according to an acoustic-subjective assessment. It must be concluded that a human listener does not evaluate music on the same scale as the NLL operates and questions its usage as optimisation criterion.

It was observed that the model occasionally produced sounds including an accordion (step 494k and 1138k). The accordion is explicable as the data corpus of this experiment contained one third yodel music with instruments. However, samples generated at other training steps never sounded like an accordion. A reason for this behaviour might be that because one piece of yodel music at the time was optimised, the model specialised in the characteristics of the last processed music. Therefore, the hypothesis is drawn that if accordions are present in the few latest training samples, then the model most likely produces yodel music with accordion. If this is true, then it might also work for a particular yodel region. This hypothesis was tested in the next experiment WaveNet UW.

4.1.2 WaveNet UW

The experiment WaveNet UW is a follow-up experiment to WaveNet Default. The model was taken at training step 1'514'700 and trained with natural yodel music from Unterwalden (UW) only. Thereby, the data corpus was limited to 170 songs with approximate $7h$ music in total. Note that during the same time in the previous experiment more yodel music of different regions was gathered and stored in the data corpus. The yodel music available in total is described in 1.4. This research serves to reveal whether further improvement is possible and whether the model can imitate characteristics of a particular yodel region. The attributes of Unterwaldner yodel is described in Section 1.7.1.

The developers on GitHub of WaveNet use a silence threshold to eliminate silence in training samples (see Section 3.2.4). This limit was set by default to 0.3 on the advice of developers that a value of zero can cause the model learning noise only. For yodel music, it is a common characteristic that yodellers fade out at some point and start a new part, occasionally at a faster pace (agogic). To include these more quiet elements the threshold was set to 0.01, and with these settings, the model was trained in a pre-experiment. First, the model improved as before, but after a while, the loss drifted away as shown in Figure 4.2. It is unclear whether the silence threshold accounts for this or if it is merely a coincidence with an inappropriate learning rate that caused the model to overshoot and leave the local optimum. Note that the learning rate was not changed during training, but the optimiser was set to RMSProp instead of Adam. As a result, the silence threshold was set back to 0.3, and the experiment was restarted. The behaviour has not been observed since.

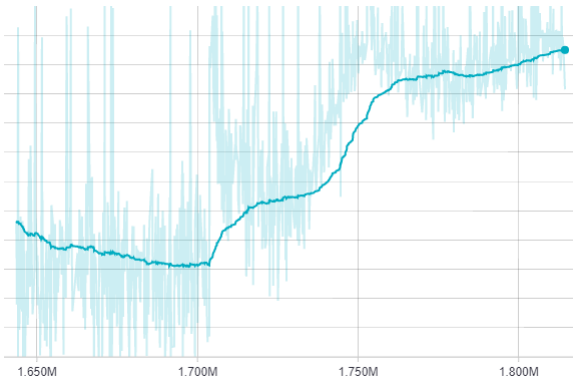


Figure 4.2: Learning behaviour with silence threshold set to 0.01

Setup

This experiment used the same model as described in Table 4.2 and the input processing settings of Table 4.3. The optimiser was changed to RMSProp to test if in combination with a high momentum of 0.9 more progress can be achieved (see Table 4.5). Unfortunately, the event logs were lost, and it could not be determined if the optimiser had an influence.

Optimiser Parameters	
Optimiser	RMSProp
Decay rate	0.9
Momentum	0.9
Learning rate	10.0^{-3}
Loss function	Softmax with cross-entropy
L_2 Regularisation strength	0

Table 4.5: Hyper-parameters related to the optimiser (see Section 2.5.1) used in experiment WaveNet UW.

Results and Samples

The model has trained for additional 2'143k training steps (one more month of training) and reached a final mean NLL of 2.04. Because the log files were lost the model was trained for additional 25k steps (in total 2'168k) to evaluate the mean NLL. The result is given in Table 4.6. Compared with the original model the NLL was optimised to 2.04. Note that the data corpus was changed and the NLL is calculated concerning the data corpus UW.

Training steps(in k)	**NLL Mean**	**NLL Std.**	**NLL Min**	**NLL Max**	**Sample**
Next 25k	2.04	0.37	1.04	2.89	wavenet_uw_3658500.wav

Table 4.6: Mean, Variance, Minimum and Maximum are derived by 200 random samples out of the training steps given in the first column from experiment WaveNet UW.

Of more interest is the progress in acoustic colour and characteristics that appeared. In the best sample of WaveNet Default (sample at training step 1'514k) several voices are apparent that sing something but more turbulent than in harmony. The model WaveNet UW produces samples in which a chorus is present, singing the keynote, while in the foreground a voice makes some melody. A lead singer and a choir are both typical characteristics of yodel (see Section 1.7) and marks a significant step forward in the direction of yodel music. Although the data corpus was limited to $7h$ and 2 million training steps were performed, no indication for the model to overfit could be found. Evidence would be that a voice is recognisable as a yodeller of Unterwalden or a part of a real song is present in the samples. As both are not the case, it is concluded that a data corpus of $7h$ is sufficient for this model.

Despite the progress, it must be stated that the samples do not contain a structured melody. The sample size was set to 5 seconds which is less than 4% of a song ($\sim$2 min. 30 sec.). Even for humans, it is difficult to catch a song structure in 5 seconds duration. The sample size might be a limitation for further progress. Besides, the receptive field, which is given by $\sim$ 32 ms for the used architecture, might be too small. The field describes the sample duration the next prediction is based on. It is unclear if the model develops a structured melody given more training steps or if the mentioned factors limit its ability.

4.1.3 Conditioning

Conditioning is a technique where a supervised feature is provided during the learning process for the behaviour of the model to be steered according to the users' preference. Here, the model was conditioned to produce yodel music of the different regions (see Sections 3.1 and 3.5.1). In contrast to WaveNet UW (see the previous section 4.1.2) that was optimised directly on Unterwaldner yodel, this model additionally had a global condition vector to learn how the different regions sound. Thereby, the training samples included all yodel samples. While generating new samples a global condition vector for the specific region was used to force the model to produce different styles. With this experiment, the technique of conditioning is explored.

A Nvidia P6000 with 24 GB RAM was available for this experiment which means a twofold resource compared with the investigations of Sections 4.1.1 and 4.1.2. The newly available resource can be used to:

- increase the batch size
- increase the sample size
- add additional layers to WaveNet that increases the receptive field

With an increased batch size most probably convergence is reached faster than with batch size 1. However, experiment WaveNet Default and WaveNet UW showed good results could be achieved if training time is not an issue. Increasing the sample size allows the model to be trained on more seconds and more extended parts of a song structure are learnt at once. Therefore, by increasing the sample size, it is more likely that the model memorises the question-and-answer game of yodel music

(see Section 1.7) than with higher batch size. The last possibility is to add additional dilation layers to WaveNet. Additional layers would increase the receptive field and would make a predicted sample point dependent on a more extended sequence. It is unclear how the samples' size and receptive field interact, but it is likely that the receptive field accounts for the acoustic colour more than for a melody.

Based on the mentioned evaluation it has been decided to increase the sample size to investigate if the model can produce more structured songs.

First attempt

First, the sample size was increased to the maximal possible value of 20 seconds (320'000 sample points at a time) and trained for about 500k training steps with a learning rate of 10^{-3}. Unfortunately, the model did not improve at all. Figure 4.3 shows the log-likelihood over time. Furthermore, between training step 50-150k, the mean NLL was 5.29 and worsened to the end to 5.30. It seems that the sample size was too large for the model to be trained on or more training time would be necessary to see progress. However, the experiment was aborted and restarted with a sample size of 240'000.

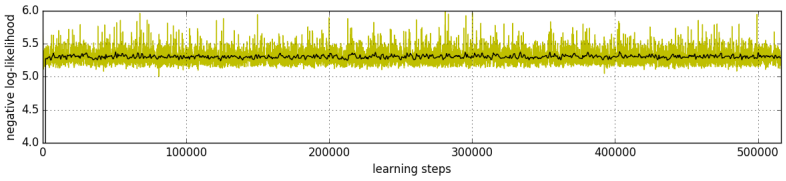


Figure 4.3: NLL over training steps for a sample size of 20 seconds for the first attempt.

Setup

WaveNet was trained as proposed but with a sample size of 5 seconds (240'000 sample size, see Table 4.3). For this experiment, the silence threshold was decreased to 0.1 which meant more quiet sequences of yodel are present in the training samples. The model's parameters were left unchanged (see Table 4.8). The optimiser was changed to RMSProp to evaluate if faster progress can be achieved with the momentum technique (see optimiser parameters in Table 4.1). Additionally, weak L_2 regularisation of 10.0^{-4} was used to penalise large weight values, and the learning rate was set to 10^{-1} to reach convergence faster. Both parameters have been evaluated through a manual hyper-parameter tuning. While training, the learning rate was decreased manually after 195k to 10^{-3} and after 1'093k to 10^{-5} to regain a smooth learning curve.

Optimiser Parameters	
Optimiser	RMSProp
Decay rate	0.9
Momentum	0.9
Learning rate	10.0^{-1}
Loss function	Softmax with cross-entropy
L_2 Regularisation strength	10.0^{-4}

Table 4.7: Hyper-parameters related to the optimiser (see Section 2.5.1) used in experiment WaveNet Conditioning.

WaveNet	
Initial filter width	32
Block filter widht	2
Dilation layers	5x [1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
Residual Channels	32
Dilation Channels	32
Quantisation Values	256
Skip Channels	512
Receptive Field	5146($\sim$ 321 ms)
Output Modelling	Bernoulli distribution with $K = 256$

Table 4.8: Properties of the model of experiment WaveNet Conditioning. In total, the model contains a causal convolutional layer, 50x dilated convolutional layers and two post-processing convolutional layers. Therefore, the model consists of 53 layers. The receptive field of the model covers 32 ms.

Input Processing	
Sample size	240000(5 s)
Sample rate	16000
Batch size	1
Silence threshold	0.1
Data corpus	yodel

Table 4.9: Parameters of input processing to experiment Wavenet Conditioning optimised for a Nvidia P6000 with 24 GB RAM.

Results and Samples

In total, 2'194k training steps were performed to reach a final mean NLL of 2.25. According to the NLL, the last model or the model of 1515k returns the best samples. Unfortunately, when listening to all generated samples, the only improvement was found between samples of the model with 494k training steps and the model of 1515k training steps. Model 1903k sounds almost the same as its predecessor. From the first sample at 89k to the last a distinct cracking noise, like clapping is audible. There are some excellent sequences in the sample of 1515k and 1903k, but it is far from the performance of the experiment WaveNet Default that was trained to only 1514k training steps. According to the learning curve in Figure 4.4 it is questionable if further training can improve the result. It seems that the model did not profit from more extended sample sizes as anticipated.

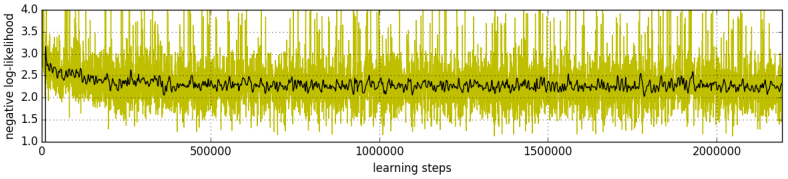


Figure 4.4: NLL over training steps for a sample size of 15 seconds of experiment Conditioning.

Training steps(in k)	**NLL Mean**	**NLL Std.**	**NLL Min**	**NLL Max**	**Sample**
450-550	2.30	0.38	1.48	4.09	wavenet_conditioned_494000.wav
900-1000	2.27	0.46	1.36	4.21	wavenet_conditioned_926000.wav
1500-1600	2.23	0.40	1.17	4.46	wavenet_conditioned_1515000.wav
1850-1950	2.32	0.39	1.37	4.26	wavenet_conditioned_1903500.wav
Last 50	2.25	0.33	1.43	3.10	

Table 4.10: Mean, Variance, Minimum and Maximum are derived by 200 random samples out of the training steps given in the first column from experiment WaveNet Conditioning.

The assessment of conditioning was made with the model of 1903k because its samples sounded a little better than that of the model of 1514k. Meanwhile, the experiment was kept running in the hope for an improvement. It was expected that the model would produce a different acoustic colour according to the conditioning used. Figures 4.5 and 4.6 show the FFT over the audio signal for a song from Unterwalden and Toggenburg. They naturally look different in this analysis and therefore, it is expected that generated samples conditioned to specific regions also differ in the FFT analysis. No visible difference would mean that the conditioning does not work (the model produces the same no matter the conditioning). When listening to samples conditioned on regions, they sound almost the same (with one exception). The best candidates are sample "wavenet_conditioned_1903500_to.wav" (conditioned to Toggenburg) and sample "wavenet_conditioned_1903500_ai.wav" as they contain less disturbing noise. See Appendix A.2, repository "WaveNet Conditioning" to get access to the samples. The sample conditioned to Toggenburg (see Figure 4.8) produces some notes with high frequency while the sample from Appenzell Innerrhoden shows in the FFT analysis almost no structure (see Figure 4.7). Based on these results it is difficult to conclude. For Toggenburg, the results are repeatable. Samples conditioned to Toggenburg contain less disturbing noise than all other regions. Therefore, conditioning makes a difference for Toggenburg (1 of 8 regions).

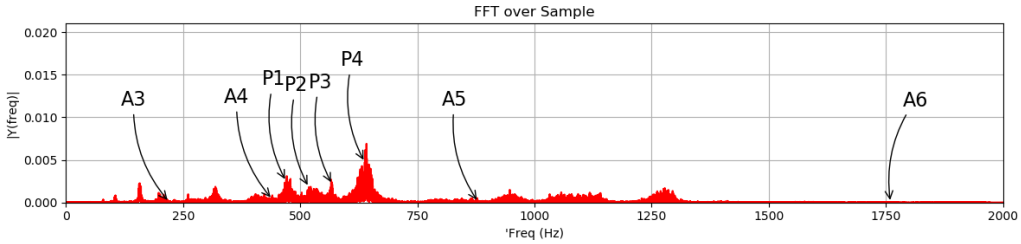


Figure 4.5: An Unterwaldner yodel song for CD 13. UJV Stalden (1994) - CD 1 - track 03. A3, A4, A5 and A6 are orientation markers. The Peaks P1-4 have a cent distance of $cent(P1, P2) \approx 169$, $cent(P2, P3) \approx 160$ and $cent(P3, P4) \approx 200$.

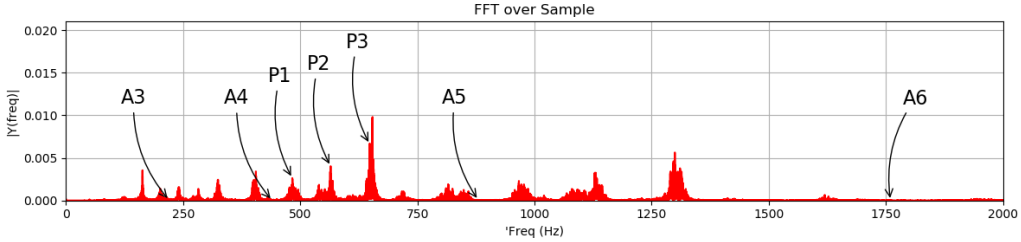


Figure 4.6: A Toggenburger yodel song for CD Bazenheid (2006) Naturjodel-Konzert - CD 1 - track 03. A3, A4, A5 and A6 are orientation markers. The Peaks P1-3 have a cent distance of $cent(P1, P2) \approx 271$ and $cent(P2, P3) \approx 238$.

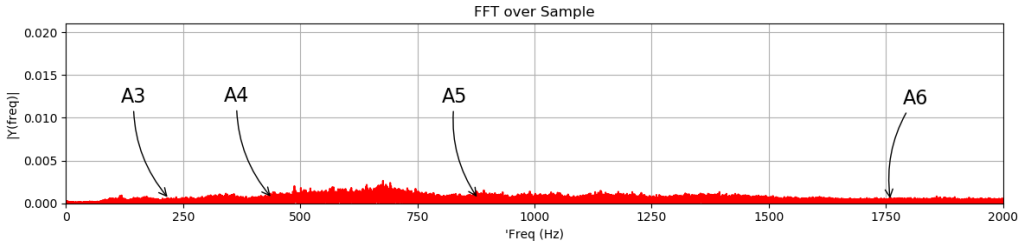


Figure 4.7: FFT analysis of a generated song conditioned to Appenzell Innerrhoden. A3, A4, A5 and A6 are orientation markers. No peaks are recognisable.

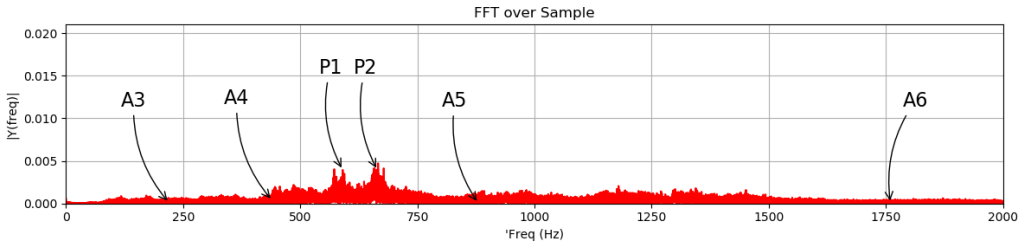


Figure 4.8: FFT analysis of a generated song conditioned to Toggenburg. A3, A4, A5 and A6 are orientation markers. The Peaks P1-2 have a cent distance of $cent(P1, P2) \approx 205$.

Because convergence is reached very slowly and it is unclear whether more training could improve the result, it is difficult to analyse if the enlarged sample sizes lead to better-structured melodies. Further experiments should concentrate on expanding the receptive field to assess its influence.

4.1.4 The "slapping" effect

The "slapping" effect is a disturbing sound which sounds like a slap has been observed in the samples of the WaveNet Default and UW model. Its evolution started with the model WaveNet after 17 days of training (> 800000 training steps) where it was not as pronounced as in the latest model version. The effect disappeared after 24 days of training as an accordion was prominent in all samples. By 33 days of training, the effect was observed several times per sample. After that, the training was changed to yodel from Unterwalden only (experiment WaveNet UW, see Section 4.1.2).

With natural yodelling from Unterwalden as training sources, the effect disappeared utterly at first, but high notes such as the ones from microphones with inadequate settings were present (1.6 million training steps) instead. From 3 million steps on, the effect became more and more prominent within the samples.

Figure 4.9 shows the "slapping" effect in the signal of a sample. It might be caused by signals that exist in the original audio files like clapping, which is present in some audio files, or some side effects of the mu-law encoding that is performed prior to the learning process. Any feedback loops or some combination of effects that might not be analysable may lead to these "slappings".

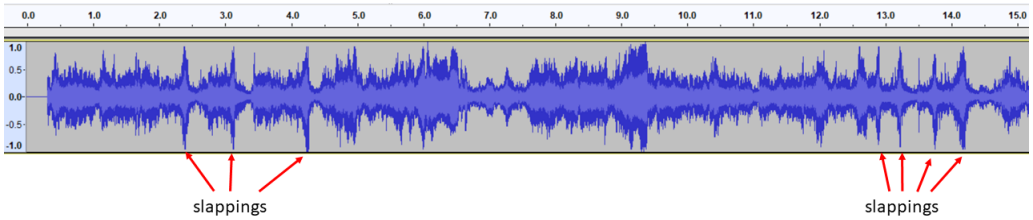


Figure 4.9: The "slapping" effect in sample "wavenet_conditioned_3658500_2.wav" detected seven times in 15 seconds.

Clapping appears as random noise in the signal (see Figure 4.10) and tests with mu-law encoding and decoding process on several yodel files revealed no evidence that it could cause something similar to the "slapping" effect. Therefore, it is unlikely that the behaviour is caused by the training data itself or the input process.

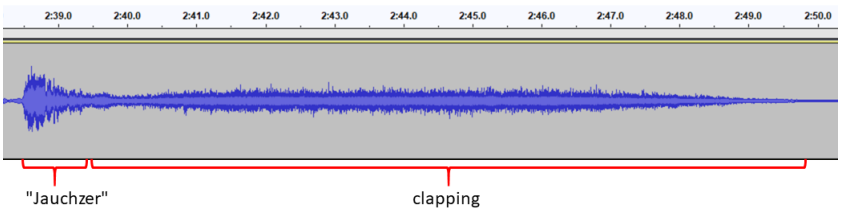


Figure 4.10: A jauchzer followed by clapping in the song 13. UJV Stalden (1994) - CD 2 - track 15.

4.1.5 Analysis of frequencies

Next, the signals are analysed using a fast Fourier-transform (FFT) to decompose it into its frequencies. The result is used to construct a histogram that gives clues about which notes were used at what rate. Often used notes should result in a spike and according to the cent measure, introduced in Section 1.7, have a defined distance to each other of around 100 cents. The histogram of the opera singer Ingrid Kaiserfeld is shown in Figure 4.11. The notes A3(220Hz), A4(440Hz), A5(880Hz) and A6(1760Hz) are indicated for orientation. If the peaks P1-3 are analysed with the cent measure (see Equation 1.1), it is apparent how precise the intervals are sung. From P1(220.6 Hz) to P2(306.3 Hz) it is a distance of 306 cents (almost the minor third equal temperament with 300 cents) and for P2(263.3 Hz) to P3(296.2 Hz) about 203 cents (almost the major second equal temperament with 200 cents). A yodel with a choir accompaniment will not produce as accentuated spikes as shown for the opera singer. This, because more singers introduce more variation in frequency and thereby, it is expected to see this variance, which cannot be mistaken for being worse singers. An Unterwaldner yodel song (13. UJV Stalden (1994) - CD 2 - track 15) produces a histogram as given in Figure 4.12. The cent distance between the peaks is given as follows

$$\begin{aligned} cent(P1, P2) &\approx 172.1, \\ cent(P2, P3) &\approx 201.2, \\ \text{and} \quad cent(P3, P4) &\approx 298.4. \end{aligned}$$

For the twelve-tone equal temperament chromatic scale, the major third is set to 400 cents and the minor third to 300 cents. On the harmonic scale, the major third is set to 386 cents and the minor third to 314 cents. The distance from P1 to P3 is about 373.4 and therefore, more or less a major third on the harmonic scale. The Wavenet UW model achieved to produce two prominent peaks (P2 and P3) and one height P1 that is less defined (see Figure 4.13). By calculation of the distances, given by

$$\begin{aligned} cent(P1, P2) &\approx 373.3 \\ \text{and} \quad cent(P2, P3) &\approx 269.07, \end{aligned}$$

it is apparent that the difference of P1 to P2 in cents is almost identical as the difference of P1 to P3 in the Unterwaldner yodel sample. Because the peak P1 is not well defined, it is uncertain if this peak is a note. The second is not well approximated by the twelve-tone equal temperament chromatic scale but fits the septimal minor third on the harmonic scale (267 cents). It may be derived from the harmonic series from the seventh harmonic, the natural fa. See Figure 1.1, the proportion from the sixth string to the seventh is 6:7. Here, P2 is given by 617.1 Hz and P3 denoted with 720.9 Hz and hence, having the approximate proportion of 6:7 from P2 to P3. It is not possible to state that it is the seventh harmonic as the keynote is unknown. It is an indication, at least, that the model learnt something about yodel.

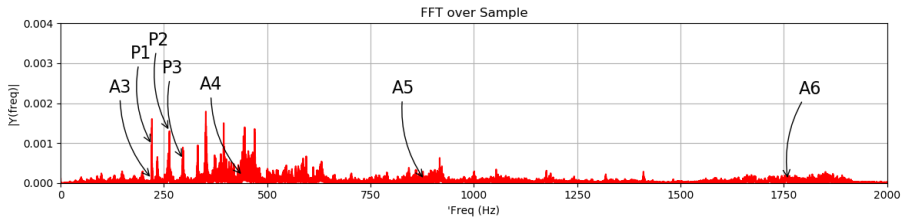


Figure 4.11: Histogram over the frequencies using the fast Fourier-transform over an opera song of Ingrid Kaiserfeld. A3, A4, A5 and A6 are markers of the equal temperament chromatic scale. The difference of P1-3 in cents are given by $cent(P1, P2) \approx 306$, $cent(P2, P3) \approx 204$.

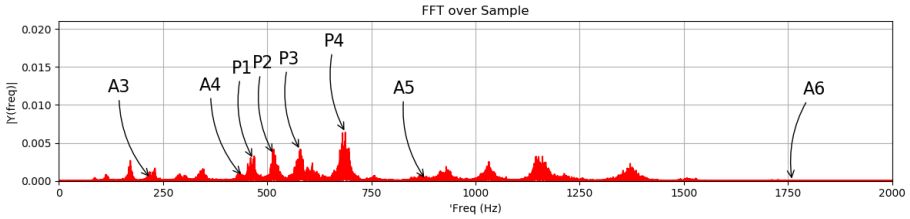


Figure 4.12: Histogram over the frequencies using the fast Fourier-transform over a yodel song 13. UJV Stalden (1994) - CD 2 - track 15. A3, A4, A5 and A6 are markers of the equal temperament chromatic scale. The difference of P1-4 in cents are given by $cent(P1, P2) \approx 172$, $cent(P2, P3) \approx 201$ and $cent(P1, P4) \approx 298$.

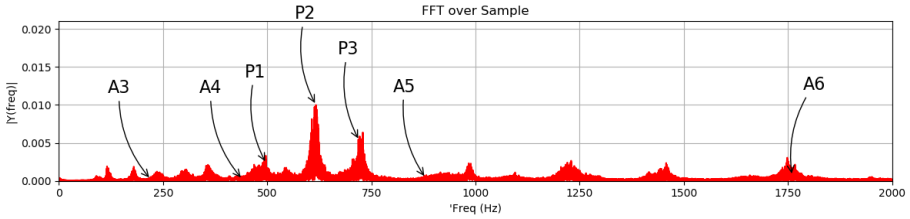


Figure 4.13: Histogram over the frequencies using the fast Fourier-transform over a generated sample of model Wavenet UW. A3, A4, A5 and A6 are markers of the equal temperament chromatic scale. The difference of P1-3 in cents are given by $cent(P1, P2) \approx 373$ and $cent(P2, P3) \approx 269$.

4.1.6 A qualitative analysis of Wavenet

Based on the available WaveNet models the best was selected (WaveNet UW) to generate samples for qualitative analysis. For access to the samples, see Appendix A.2, repository "WaveNet Qualitative Analysis". Four samples of sizes 9 to 30 seconds have been generated, using a random input, and were evaluated by Andrea Kammermann (see Section 1.3). This qualitative analysis answers research question 1 (see Section 1.2) whether generative models can reproduce characteristics of yodel music. The following aspects have been assessed in details

1. **Characteristics:** What typical yodelling characteristics are recognisable?
2. **Melody:** Are there any particular melodies used in yodelling in the available samples?
3. **Metre:** Does the sample contain specific yodelling metre?
4. **Song:** Are yodelling song-structures observable and if so, of how many seconds?
5. **Creativity:** Is creativity recognisable within the samples in comparison to yodel-music?

Characteristics In all samples, the music consists of voices which can be identified by the listener as such. All tunes have an audible chorus that accompanies in triads (if major C then this would be C-G). Additionally, this choir is polyphonic, which is a common property for yodel music. The melody consists of low and at times high notes that in yodelling would be the change of registers from breast voice to the falsetto. However, the glottal slop characteristic is not detectable. Only the low and high notes are present. The model succeeded in reproducing the correct vocalisation of high notes with an "u" and breast notes with an "o". Overall, the music is much too fast, restless, and it is interrupted here and there by some slapping (sounds like an abrupt thunk, clapping). A clear separation of the chorus and the leading melody, sung by the yodeller, is missing as well. The model failed in producing other characteristics such as enduring notes and the cosiness of yodel music.

Restlessness in generated music was observed within the results of Mehri et al. (2016) and van den Oord et al. (2016a) as well, even if the authors did not explicitly mention this. It may be the result of a receptive field of $\approx 250ms$(5146 steps, Wavenet UW model) which is too small to reproduce longer more relaxed structures (see Section 4.1.3 where the sample size was increased without success).

Melody As already stated, the model succeeded in reproducing music ranging from low to high notes. Additionally, the chorus returns in all samples back to its keynotes which is typical for yodel music.

What is missing is a certain structure, sequence of notes typical for the yodel music's question-and-answer game (see Section 1.7). The melody of yodel music always contains repetitions. The model failed in reproducing these properties.

Metre All samples seem to have the same metre that appears monotonous for the whole duration. A monotonous metre is typical for yodel music, but here it does not appear to be intended. It means that the rhythm is not precise (e.g. the first note of a bar is usually accentuated). A particular structure in the metre is not found. Additionally, some dynamic is missing such as the agogic of yodel music (from slow to fast and back).

Song All samples are far from being a song. Repetitions of individual parts do not occur, and an intro is missed entirely. A typical trait of yodel music is that the yodeller starts solo and after a while, the choir joins in. Such behaviour has not been observed in any of the samples.

Creativity The produced samples are mostly distinguishable by the slapping. The slapping gives the samples a specific structure that can be memorised by the listener. Therefore, it must be concluded that there is not much creativity produced by the model in comparison to yodel-composers.

The overall performance of the model broken down to a measure from "random noise" to "A yodeller's performance" is given by Figure 4.14. The model can reproduce typical characteristics of yodel music, but samples lack song-structure, metre and creativity.

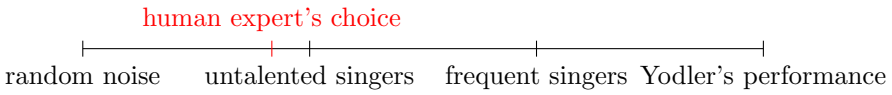


Figure 4.14: Opinion score ranging from "random noise" to "A yodeller's performance" assessed by a human expert.

4.1.7 A second opinion

Nuria Richner is a soprano of Schwyz (see Section 1.3) and teaches music beside professional singing. She was asked about her opinion of the performance of the model WaveNet UW. Before the interview, she had received a sample "wavenet_uw_3658500_2.wav" for a first impression. After listening to the sample "wavenet_uw_3658500_1.wav", she immediately recognised that it was not the same sample as she had heard before. Therefore, she was asked the following questions:

- What is the recognition effect of sample "wavenet_uw_3658500_2.wav"?
- Why does it sound yodel-like?
- What shortcomings have the samples?

The following is a summary of the discussion with her.

Recognition effect

The sample "wavenetuw36585002.wav" is recognisable by clappings that are followed by the leading voice singing an F#-G-A where the G is unusual for a B major scale. Typically, it is denoted by C#-D#-E-F#-G#-A#-B. From the keynote B the sequence F#-G-A-B would be the second tetrachord in B minor (not B major). Additionally, it is unusual the melody remains on the interval of a minor seventh (1000 cents equal temperament) and does not lead upwards to the keynote. However, according to the expert, it is difficult to identify a note.

Yodel timbre

The sample has a sonority that sounds like a yodel choir. The bass sings a fifth (700 cents equal temperament). The B major, or maybe F#, is sung as keynote, maybe it is a chord H-F#-D#. Note that the frequency analysis in Section 4.1.5 revealed the frequencies

- 497.4 Hz $\sim$ B4 (theoretical 493.9 Hz) equal temperament,
- 617.1 Hz $\sim$ D#5 (theoretical 622,3 Hz) equal temperament,
- 720.9 Hz $\sim$ F#5 (theoretical 739.9 Hz) equal temperament,

almost fits these mentioned notes. Additionally, it was stated that the vocalisation used in the sample is like yodellers would do and that the glottal stop is audible at the position where it should be for women voices. The last statement is in disagreement with experts of Section 4.1.6 which did not recognise any glottal stop.

Shortcomings

The expert mentioned as a shortcoming that the sample has no song form at all. Furthermore, the lead singer does not develop a melody, and there is no arc of suspense recognisable. Melodies usually contain a development such that an instrument could be used as an accompaniment (from one chord to the next). Here, the melody does not change in such a way. Additionally, there is no metre determinable, and the vocalisation of the choir accompaniment is not consistent and precise (at times a "jo" is audible for low tones).

Comparison of experts

In comparison with the expert of Section 4.1.6, there has been only one disagreement about the glottal stop. All in all, both experts agreed that the model reproduces the timbre of yodel. They both stated that it is, however, no music composition as melody development and the metre are missing.

4.1.8 A conclusion to Wavenet

The WaveNet UW model has trained a total of about 45 days. The final model succeeded in producing an easily identifiable timbre of yodel music that has been confirmed by several listeners and experts. Qualitative analysis revealed that several characteristics, such as the switch from low to high notes, and back, or the typical yodel syllabus, were learnt by the model. However, the model fails to produce a structured melody over more than several seconds and does not provide repetition of some parts that would be common for yodel. Experiments with increased sample sizes neither produced structured songs nor samples of similar quality as WaveNet UW. Compared with a yodeller composer the model shows almost no creativity.

To conclude, the model achieved to produce the timber of yodel music in a quality that most listeners identify the samples as a yodel, but the model failed to compose yodel music.

4.2 SampleRNN

SampleRNN was proposed by Mehri et al. (2016) which made the source code available on GitHub[2]. Unfortunately, the implementation was not executable due to unknown reason, and it has been decided to reimplement the network in TensorFlow™. According to the authors, SampleRNN achieved a better preference-based score than WaveNet by human listeners. This experiment was to evaluate its performance concerning yodel music.

4.2.1 SampleRNN Default

First, an experiment was conducted that used proposed hyper-parameters of SampleRNN. The results of SampleRNN for Bethoveen music (Mehri et al. (2016)) are impressive, and therefore, the settings might work for yodel music too. Unfortunately, not all hyper-parameters were specified precisely, and the implementation does not contain any further details. For this reason, the used hyper-parameters might differ from the setup of Mehri et al.. As training data, all available yodel music (natural yodel, yodel with and without instrumental accompaniment) was used (see Section 1.4).

Setup

Hyper-parameters have been used as proposed by Mehri et al. (2016). It was found that the settings were optimised for a Nvidia P100 with 16 GB RAM (not mentioned by the authors). The hyper-parameters for the optimiser are given in Table 4.11 and for the input processing in Table 4.13. The sample size is not explicitly mentioned by Mehri et al. but was derived from the frame size of the third and second tier and the available resources to 512.

Optimiser Parameters	
Optimiser	Adam
Decay rate β_1	0.9
Decay rate β_2	0.999
Learning rate	10.0^{-3}
Loss function	Softmax with cross-entropy
L_2 Regularisation strength	0.1

Table 4.11: Hyper-parameters related to the optimiser used in experiment SampleRNN (see Section 2.5.3).

SampleRNN	
3-Tier frame size	256
2-Tier frame size	128
1-Tier frame size	512
Quantisation Values	256
Layer dimension	1024
Output Modelling	Bernoulli distribution with $K = 256$

Table 4.12: SampleRNN model parameters for experiment SampleRNN (see description of SampleRNN in Section 3.5.2).

Input Processing	
Sample size	512(32 ms)
Sample rate	16000
Batch size	128
Silence threshold	0.1
Data corpus	yodel, yodel with text and with/without instruments

Table 4.13: Parameters of input processing to experiment SampleRNN Default optimised for a Nvidia P100 with 16 GB RAM.

[2]https://github.com/soroushmehr/sampleRNN_ICLR2017, accessed 15.01.2018

Results

As shown in Figure 4.15 no progress is apparent for 419k training steps. Thereby, the experiment was aborted. Note that if the mean NLL was calculated on different training steps progress was measurable but on a level that it is unlikely ever to reach an appealing quality. It is explicable that with a sample size of 32 ms a network is not able to reproduce a harmonic yodel sound and it is doubtful that Mehri et al. achieved their results with these settings. Either the interpretation of the published settings were wrong, or the implementation differs fundamentally.

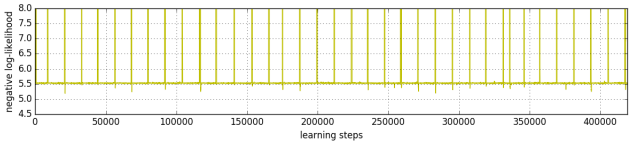


Figure 4.15: NLL of SampleRNN with default settings over time.

4.2.2 SampleRNN Mixture Model

This experiment was conducted to test SampleRNN with different settings. The sample size was increased to 0.5 seconds what resulted in a smaller batch size to fit on the GPU. In Salimans et al. (2017), the output was model as a mixture model of logistic distribution to reduce the output size from 256 to 30 nodes. This technique was adopted by van den Oord et al. (2017b) to a VAE-WaveNet version and achieved high-grade results for speech with a mixture model of 20 logistic distributions. Salimans et al. (2017) published his source code on GitHub[3]. It was adopted in this thesis and tested with SampleRNN.

Setup

In a hyper-parameter search, the learning rate was tuned testing values from 10^{-1} to 10^{-5}. The best was found at 10.0^{-4}. Other hyper-parameters were set as proposed by Mehri et al. (2016). The optimisation hyper-parameters are given in Table 4.14 and for input processing in Table 4.16. It has been decided to use frame sizes of 1024 for the third tier, 256 for the second tier and 8192 for the first tier (see Table 4.15). The maximal possible batch size was found at 32 (running on a Nvidia P100, 16 GB RAM).

Optimiser Parameters	
Optimiser	Adam
Decay rate β_1	0.9
Decay rate β_2	0.999
Learning rate	10.0^{-4}
Loss function	log-likelihood for mixture of discretized logistics
L_2 Regularisation strength	0.1

Table 4.14: Hyper-parameters related to the optimiser used in experiment SampleRNN Mixture Model (see Section 2.5.3).

SampleRNN	
3-Tier frame size	1024
2-Tier frame size	256
1-Tier frame size	8192
Quantisation Values	256
Layer dimension	1024
Output Modelling	logistic mixture model with 20 distributions

Table 4.15: SampleRNN model parameters for experiment SampleRNN Mixture Model (see description of SampleRNN in Section 3.5.2).

[3]https://github.com/openai/pixel-cnn, accessed 28.12.2017

Input Processing	
Sample size	8192($\sim$ 0.5 seconds)
Sample rate	16000
Batch size	32
Silence threshold	0.1
Data corpus	yodel, yodel with text and with/without instruments

Table 4.16: Parameters of input processing to experiment SampleRNN Mixture Model optimised for a Nvidia P100 with 16 GB RAM.

Results and Samples

The experiment was run for 19 days (1247k training steps) and reached a final mean NLL of 2.74. Regrettably, the produced samples contained almost nothing that was distinguishable from noise although, for WaveNet experiments with such a score, there were already certain melodies audible. Progress for this model was considerably slow. From steps 450-550k to the end the NLL reports a difference of only -0.03 which is very little for ten days of training on a P100. The experiment was aborted afterwards to give space for other surveys.

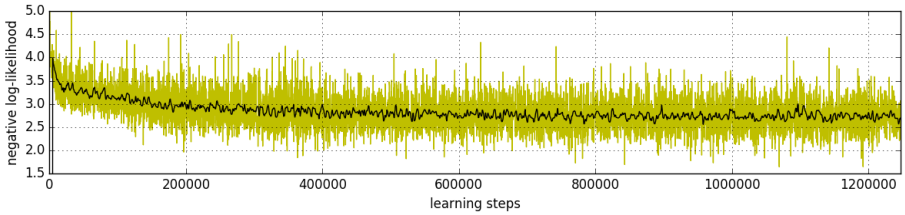


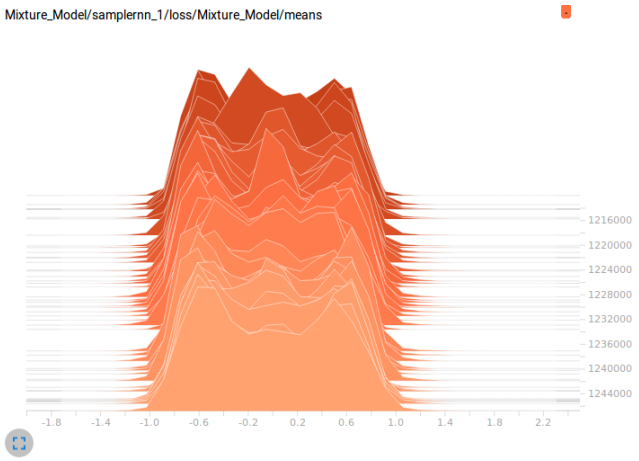
Figure 4.16: NLL of SampleRNN with default settings over time.

Training steps(in k)	**NLL Mean**	**NLL Std.**	**NLL Min**	**NLL Max**
450-550	2.77	0.28	2.17	3.66
900-1000	2.75	0.29	1.91	3.70
Last 50	2.74	0.30	2.00	3.60

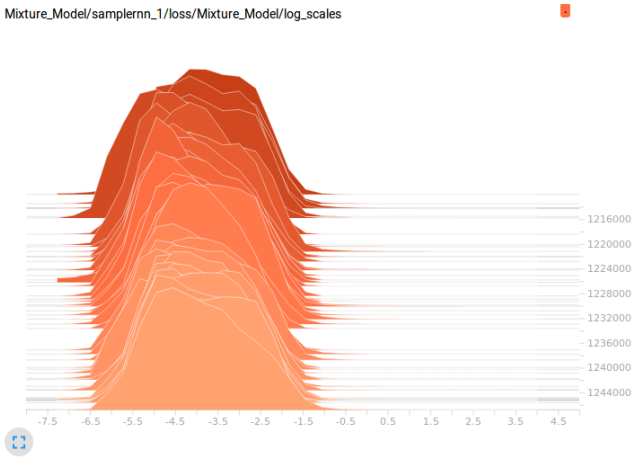
Table 4.17: Mean, Variance, Minimum and Maximum are derived by 200 random samples out of the training steps given in the first column. The batch size for the experiment SampleRNN Mixture Model was set to 32.

In comparison with other models, the output distribution was set to a mixture model of 20 logistic distributions. Tensorboard (Martìn Abadi et al. (2015)) is a tool for TensorFlow^{TM} to observe the performance of a model during training. It provides functionality to plot histograms over a batch of a tensor per training step. The behaviour of the mixture model was investigated with histograms over logged values such as mean and scale of logistic distribution and log-probabilities. Figure 4.17 shows the histogram of each tensor over the training steps. The mean values (see Figure 4.17a) successfully converged into a range of $[-1, 1]$ which covers the scope of the input data (input data are normalised to $[-1, 1]$, see Section 3.5.2). Note that the networks output layer does not contain a non-linear activation which shifts the values in this range. The log-scale values are found in range -7.5 and -1.5 (see Figure 4.17b). A log-scale of -7.5 makes perfect sense if it is considered that the input is quantised to 256 then normalised in between $[-1, 1]$. The distance from one quantised value to another is therefore $\frac{1}{256} \approx 0.004$. If a logistic distribution is plotted at $\mu = 0$ and log-scale$= -7.5$ (see Figure 4.18), it can be observed that it more or less covers this distance. The other end of the range log-scale$= -1.5$ is less favourable as it stretches almost over the whole scope of the values. However, because it is mixture model and the final value is calculated out of all mixtures, it cannot be concluded that the model failed to fit the distributions. The predicted log-probabilities raised concerns as log-probabilities larger than zero do not make sense because the log-probability of $p(x) = 1.0$ is given by $\log(p(x)) = \log(1.0) = 0.0$.

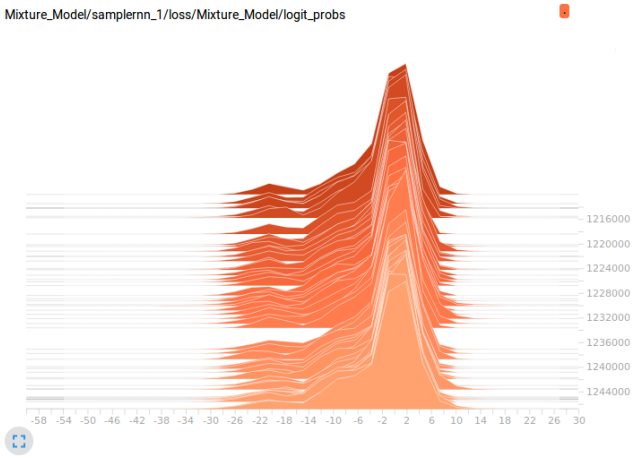
This behaviour is still explicable because the implementation (see Section 3.4.2) which shifts such values in a valid range. However, the model failed to adjust the scope of the log-probabilities which might be an explanation for its poor performance.



(a) Histogram of the mean parameters of the 20 logistic distributions.



(b) Histogram of the log-scale parameters of the 20 logistic distributions.



(c) Histogram of the log-probabilities predicted by SampleRNN Mixture Model.

Figure 4.17: The histogram of the mixture model parameters mean and scale of the logistic distribution and log-probabilities for the experiment SampleRNN Mixture Model.

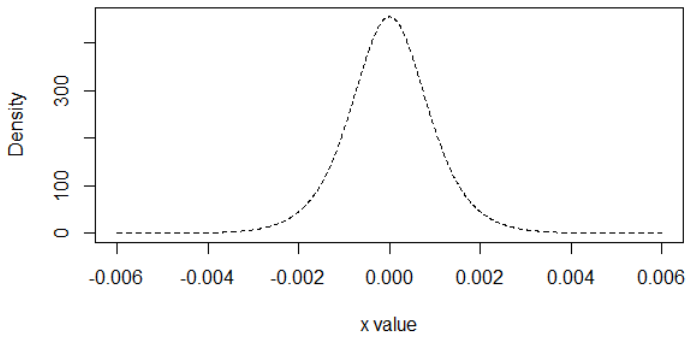


Figure 4.18: A logistic distribution with $\mu = 0$ and log-scale$= -7.5$.

4.2.3 Conclusion to SampleRNN

The results of Mehri et al. (2016) for Bethoveen could not be reproduced for yodel music. In fact, the performance gap is too large to originate from different training data only. For further experiments with SampleRNN, the implementation must be reviewed thoroughly. However, the experiments showed a drawback not mentioned so far. To generate a new sample of the size of 30 seconds with SampleRNN takes up to 3 days. Even if the model would outperform WaveNet, it is far from being practical. It has been decided to suspend further experiments with SampleRNN.

Note that another implementation written in Torch is available on GitHub[4]. It has been decided not to repeat the experiment with their implementation.

4.3 Variational Autoencoder (VAE)

VAE proved their ability to produce audio waveform signals in van den Oord et al. (2017b). The training data were speech snippets of 1.4 seconds (32'768 sample points) which are successfully compressed to a latent space of 512 dimensions. The VAE (described in Section 2.6.2) consists of an encoder that transforms the samples from sample space to a latent space of fewer dimensions. The decoder reconstructs the sample with the information given by the latent variables. It transforms latent variables from the latent space to the sample space. This section describes all experiments using VAE.

4.3.1 Evaluation to use VAE

WaveNet succeeded to reproduce the acoustic colour as described in Section 4.1.8 but failed in the creation of a melody structure. Unlike autoregressive models, generative frameworks such as GAN, VAE and AVB, produce a whole sample at once. WaveNet needs 50 layers and 5146 sample points to make one prediction (see Section 4.2.1). Given a sample rate of 16000 and a song of length 2 minutes 30 seconds, e.g. a GAN has to produce 2.4 million sample points at once. It is evident that more resources than available are needed to train such a model successfully. For now, it is impossible for GAN, VAE and AVB to produce yodel samples of such size. van den Oord et al. (2017b) used a kind of a VAE for short speeches and investigated the timbre of their model imitating some speakers. They achieved to process samples of the size of 1.4 seconds. To generate samples of larger size an actual speech was sliced and processed in an autoregressive way. This way samples of arbitrary size were produced. This idea can be adapted to yodel music. Once a VAE is trained, an encoder will translate any kind of sample from sample space to latent space, whether it is yodel or not (see the description of VAE in Section 2.6.2). It means, the sample to encode does not have to be yodel which can result in a yodel version of AC/DC or another band. Additionally, if the compression from sample to latent space is large enough a whole song is represented in such a short form in the latent space that it is possible to learn the melody structure from it. This compressed representation is best explained as a lead sheet format. It has been shown by, e.g. Colombo et al. (2016) that models based on lead-sheet inputs produce coherent melody structures. However, this might be an experiment for later

[4]https://github.com/richardassar/SampleRNN_torch, accessed 28.12.2017

studies. Nevertheless, the mentioned considerations led to the decision to use Variational Autoencoder.

AVB would also have been a possible choice as it learns a latent variable distribution too. VAE was used because the technique had been studied in several publications and there is more advice available than for AVB. Additionally, a VAE model can be extended to AVB and tested in later experiments.

4.3.2 Build VAE Acoustic

From the considerations in Section 4.3.1, the requirements evolved that a VAE has to reproduce the timbre of yodel. Therefore, "VAE Acoustic" needs an encoder and decoder network with such an ability. WaveNet succeeded in reproducing this, as described in 4.1.8. Additionally, van den Oord et al. (2017b) successfully adapted WaveNet to be used within a VAE (they did not publish the source code therefore many implementation details are uncertain). It has been decided to build a deconvolution WaveNet as described in Section 3.6.3 that is used as a decoder and to use strided-version of WaveNet (described in Section 3.6.1) as the encoder. Strides have to be introduced because the compression rate of the latent variables is too low otherwise.

The VAE Acoustic consists of the following components:

1. Encoder: A WaveNet implementation of convolution layers using strides and dilations (see Section 3.6.1).
2. Decoder: Deconvolution WaveNet with strides and dilated convolutions (see Section 3.6.3).

4.3.3 VAE Acoustic

Four experiments were performed in parallel, all with different properties to evaluate a successful network architecture and determine the size of the latent space. The general requirements for the network were to the following:

- It produces the timbre of yodel music.
- Samples size ≥ 1 second: A listener can assess a sample size of one second. It allows observing the progress of a model in tensorboard directly which has a function to play music.
- Compression rate ≥ 7: The latent variable space should have as few dimensions as possible. A compression rate of 7 is set as a minimum for later experiments with a "VAE Structure".
- Batch size > 1: Batch normalisation is used in experiments. If the batch size is set to 1, it will not have any effect.

An overview of all experiments is given in Table 4.18.

Name	**Latent Variable Size**	**Encoder Layers**	**Decoder Layers**
VAE Acoustic 512	512	9	10
VAE Acoustic 1024	1024	8	9
VAE Acoustic 2048	2048	9	9
VAE Acoustic 512, 2 seconds	512	12	13

Table 4.18: Overview over experiments for VAE Acoustic

Experiments with deep VAEs have shown that batch normalisation as proposed by Ioffe and Szegedy (2015) is essential for successful training. In fact, it was not possible to optimise the networks without batch normalisation. The gradients either vanished and half of the decoder was not trained, or the gradients exploded which caused NaN values. These observations are in alignment with the findings of Sønderby et al. (2016) who found that batch-normalisation and a deterministic warm-up, where at the beginning only the reconstruction error is optimised, are crucial. It was decided to use batch-normalisation for VAE as described in Section 3.6. A warm-up procedure may be added later. Additionally to batch normalisation, gradient clipping had to be introduced (clipped at $]-100.0, 100.0[$) to deal with large gradients. Sadly, an error in the application of batch normalisation caused the experiment to be repeated what limited the maximal experiment time window to one week.

It was found that the optimiser Adam works better for training VAE than RMSProp. The decay rates allow controlling oscillations for the latent variable parameters that were frequently observed while experimenting with VAE. Hyper-parameter tuning was done manually for all experiments. For this reason, the model was trained up to 30k training steps and its progress evaluated. It was focused on having a smooth decline for the reconstruction error and prevention of oscillation of the KL-term.

VAE Acoustic 512

The experiment explored if a small latent variable space of 512 can be used for training a VAE. This model consisted of an encoder with nine layers and a decoder with ten layers. The layers were modelled for the latent variable's dimensions to be 512. The compression rate is thereby ≈ 31. The models' parameters are given in Table 4.20. The optimiser settings are described in Table 4.19. Note that the learning rate of 10.0^{-3} was found best for most experiments. The silence threshold was set to zero for all experiments. In comparison to WaveNet, the VAE Acoustic shall learn quiet parts too (like the fade out of individual notes that is common in yodel). The data corpus contained yodel and yodel with text but without instruments. The batch size was optimised to fit a Nvidia P6000 with 24 GB RAM and set to 56.

Optimiser Parameters	
Optimiser	Adam
Decay rate β_1	0.9
Decay rate β_2	0.9
Learning rate	10.0^{-3}
Loss function	ELBO
L_2 Regularisation strength	0

Table 4.19: Hyper-parameters related to the optimiser (see Section 2.5.3) used in experiment VAE Acoustic 512.

VAE Acoustic	
Encoder	
Initial filter width	1
Block filter width	9
Dilation/Stride layers	Strides: [2, 3, 4], Dilations: [16, 2]
post filter width	9
Residual Channels	32
Dilation Channels	32
Quantisation Values	256
Skip Channels	512
Output Modelling	Gaussian
Decoder	
Initial filter width	9
Block filter width	9
Dilation/Stride layers	Dilations: [2, 16], Strides:[4, 3, 2, 1]
post filter width	8
Residual Channels	32
Dilation Channels	32
Quantisation Values	256
Output Modelling	Bernoulli distribution with K=256

Table 4.20: Model parameters of experiment VAE Acoustic 512 with a latent variable size of 512.

Input Processing	
Sample size	16000(1 second)
Sample rate	16000
Batch size	56
Silence threshold	0.0
Data corpus	yodel, yodel with text but without instruments

Table 4.21: Parameters of input processing to experiment VAE Acoustic 512 optimised for a Nvidia P6000 with 24 GB RAM.

The KL-term and the reconstruction error behaved as expected (see Figure 4.19). First, the decoder was optimised without effect on the KL-term (until training step ≈10k). Then the latent variables distribution diverged from the standard Gaussian distribution and the KL-term raised. Meanwhile, the reconstruction error decreased. Until the end of this thesis, convergence was not reached. So far, generated samples did not contain anything else than noise. More training time will show whether this model can achieve to produce yodel-like sound.

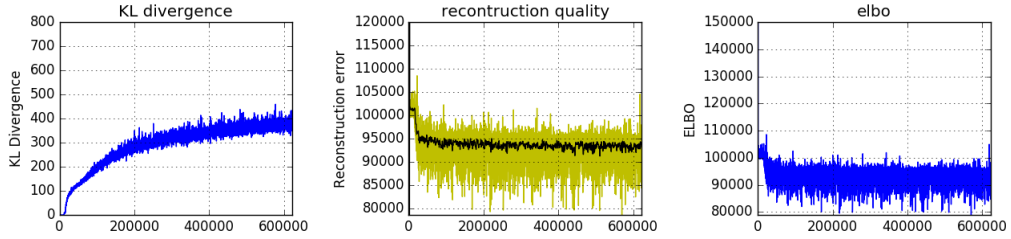


Figure 4.19: This picture shows the learning curve for the experiment VAE Acoustic 512. The left picture shows the KL-divergence term and the middle picture the reconstruction quality. The total loss is given in the right view.

VAE Acoustic 1024

The model for this experiment contained a latent variable space of 1024 dimensions. The encoder includes eight layers while the decoder has nine. The models' parameters are given in Table 4.23. This experiment proved the most difficult for hyper-parameter tuning. Oscillation in the KL-term frequently occurred but with some delay. 30'000 training steps were performed per settings to test if it happens again. It was found that a lowered learning rate (10.0^{-5}) in combination with a decreased decay rate β_2 (0.8) was adequate. Section 4.3.4 gives some more information about this observation.

Optimiser Parameters	
Optimiser	Adam
Decay rate β_1	0.9
Decay rate β_2	0.8
Learning rate	10.0^{-5}
Loss function	ELBO
L_2 Regularisation strength	0

Table 4.22: Hyper-parameters related to the optimiser (see Section 2.5.3) used in experiment VAE Acoustic 1024.

VAE Acoustic	
Encoder	
Initial filter width	1
Block filter width	9
Dilation/Stride layers	Strides: [4, 3], Dilations: [32, 4]
post filter width	19
Residual Channels	32
Dilation Channels	32
Quantisation Values	256
Skip Channels	512
Output Modelling	Gaussian
Decoder	
Initial filter width	19
Block filter width	9
Dilation/Stride layers	Dilations: [4, 32], Strides:[3, 4, 1]
post filter width	4
Residual Channels	32
Dilation Channels	32
Quantisation Values	256
Output Modelling	Bernoulli distribution with $K = 256$

Table 4.23: Model parameters of experiment VAE Acoustic 1024 with a latent variable size of 2048.

Input Processing	
Sample size	16000(1 second)
Sample rate	16000
Batch size	32
Silence threshold	0.0
Data corpus	yodel, yodel with text but without instruments

Table 4.24: Parameters of input processing to experiment VAE Acoustic 1024 optimised for a Nvidia P100 with 16 GB RAM.

At first, the model showed similar progress as "VAE Acoustic 512" (see Figure 4.20). The KL-divergence increased while the reconstruction error declined (till 100k training steps). Despite the effort in hyper-parameter tuning, the KL divergence started oscillating afterwards, but the reconstruction error could still improve (for comparison, see Figure 4.23). After training step 250k, the reconstruction error began to drift away. It is suggested that the learning rate is lowered even more to 10.0^{-6} for further testing. The total loss stayed at a high level and, so far, nothing else than noise is produced, the model might improve with better hyper-parameter settings.

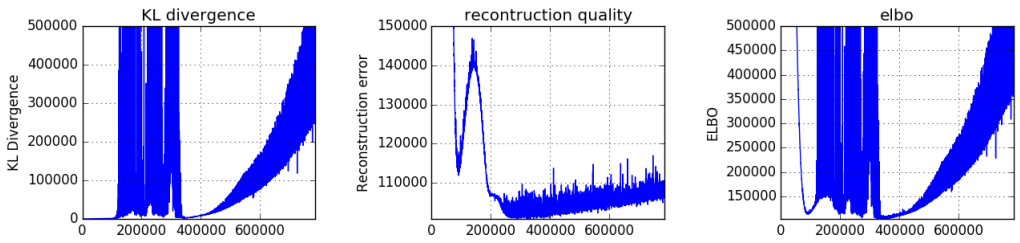


Figure 4.20: This figure shows the learning curve for the experiment VAE Acoustic 1024. The left picture shows the KL-divergence term and the middle image the reconstruction quality. The total loss is given in the right picture.

VAE Acoustic 2048

For this experiment, the model was built to have a latent variable space of dimension 2048. It is the model with the most extensive latent variable space and tested what reconstruction error is reached if the compression rate is relatively small. The encoder and decoder have both nine layers in total. The model's parameters are given in Table 4.26. The hyper-parameters of the optimiser are shown in Table 4.25, and the hyper-parameters for input processing are given in Table 4.27. The experiment was run on a Nvidia P100 with 16 GB RAM.

Optimiser Parameters	
Optimiser	Adam
Decay rate β_1	0.9
Decay rate β_2	0.99
Learning rate	10.0^{-4}
Loss function	ELBO
L_2 Regularisation strength	0

Table 4.25: Hyper-parameters related to the optimiser (see Section 2.5.3) used in experiment VAE Acoustic 2048.

VAE Acoustic	
Encoder	
Initial filter width	1
Block filter width	9
Dilation/Stride layers	Strides: [3, 2], Dilations: [64, 8, 4]
post filter width	7
Residual Channels	32
Dilation Channels	32
Quantisation Values	256
Skip Channels	512
Output Modelling	Gaussian
Decoder	
Initial filter width	7
Block filter width	9
Dilation/Stride layers	Dilations: [4, 8, 64], Strides:[2, 3]
post filter width	8
Residual Channels	32
Dilation Channels	32
Quantisation Values	256
Output Modelling	Bernoulli distribution with $K = 256$

Table 4.26: Model parameters of experiment VAE Acoustic 2048 with a latent variable size of 2048.

Input Processing	
Sample size	16000(1 second)
Sample rate	16000
Batch size	32
Silence threshold	0.0
Data corpus	yodel, yodel with text but without instruments

Table 4.27: Parameters of input processing to experiment VAE Acoustic 2048 optimised for a Nvidia P100 with 16 GB RAM.

Figure 4.21 shows the learning curve split up into KL-term and reconstruction error. Additionally, the total loss is shown. The KL divergence had optimised to almost zero. Meanwhile, the reconstruction error improved to around 103'000. Both values seem trapped in a local minimum, unable to escape. This experiment showed a behaviour that is described by Sønderby et al. as:

"We observed that such units (of latent variables) remain uninformative for the rest of the training, presumably trapped in local minima or saddle point at $KL\big(q_\phi(z|x)\|p(z)\big) = 0$, with the optimisation algorithm unable to re-activate them"(Sønderby et al. (2016)).
The authors suggest a warm-up phase in training where first the reconstruction error is optimised only. The KL-term is added granularly during the first t training steps. Further experiments with this model should use the proposed warm-up phase for training.

Figure 4.21: This picture shows the learning curve for the experiment VAE Acoustic 2048. The left view shows the KL-divergence term and the middle image the reconstruction quality. The total loss is given in the right figure.

VAE Acoustic 512, 2 seconds

With this experiment, it was explored whether a deep VAE with large sample size can be trained. The latent space dimensions were set to 512 to have a high compression rate. It is the deepest trained VAE in this thesis with an encoder of 12 layers and a decoder of 13 layers. The model parameters are described in Table 4.29 and the input processing details in Table 4.30. Because of its size, it was decided to run the experiment on a Nvidia P6000 with 24 GB RAM. The maximal possible batch size was found to be 3 which is quite small to be used with batch normalisation. The hyper-parameters for the optimiser are given in Table 4.28.

Optimiser Parameters	
Optimiser	Adam
Decay rate β_1	0.9
Decay rate β_2	0.99
Learning rate	10.0^{-4}
Loss function	ELBO
L_2 Regularisation strength	0

Table 4.28: Hyper-parameters related to the optimiser (see Section 2.5.3) used in experiment VAE Acoustic 512, 2 seconds.

VAE Acoustic	
Encoder	
Initial filter width	9
Block filter width	9
Dilation/Stride layers	Strides: [2], Dilations: [8, 64, 512], Strides: [4, 4], Dilations: [8, 16]
post filter width	7
Residual Channels	32
Dilation Channels	32
Quantisation Values	256
Skip Channels	512
Output Modelling	Gaussian
Decoder	
Initial filter width	7
Block filter width	9
Dilation/Stride layers	Dilations: [16, 8], Strides: [4, 4], Dilations: [512, 64, 8, 2], Strides: [2]
post filter width	8
Residual Channels	32
Dilation Channels	32
Quantisation Values	256
Output Modelling	Bernoulli distribution with $K = 256$

Table 4.29: Model parameters of experiment VAE Acoustic 512, 2 seconds, with a latent variable size of 512, 2 seconds.

Input Processing	
Sample size	32000(2 seconds)
Sample rate	16000
Batch size	3
Silence threshold	0.0
Data corpus	yodel, yodel with text but without instruments

Table 4.30: Parameters of input processing to experiment VAE Acoustic 512, 2 seconds optimised for a Nvidia P100 with 16 GB RAM.

The KL divergence term and the reconstruction error, shown in Figure 4.22, indicated that the optimiser was most probably trapped in a local minimum. Both much improved at first but showed no more progress until the end of this thesis. It is suggested to use a warm-up phase as described for experiment VAE Acoustic 2048.

Figure 4.22: This picture shows the learning curve for the experiment VAE Acoustic 512, 2 seconds. The left figure shows the KL-divergence term and the middle view the reconstruction quality. The total loss is given in the right image.

4.3.4 Oscillations of the KL-term

It was observed on many occasions that the KL divergence term started oscillating during training (see Figure 4.23). Note that the KL divergence yielded such large values that the reconstruction quality was almost negligible. Large gradients most probably cause the oscillation at the latent variables operation that produces z coming from the decoder (observed with tensorboard Martìn Abadi et al. (2015)). Even though gradient clipping was used, it could not avoid this behaviour. Indeed, it prevented the occurrence of NaN values. Effective countermeasures were found in smaller learning rates and the reduction of the β_2 term (Adam optimiser, see Section 2.5.3). Note that in comparison to the experiment VAE Acoustic 1024 (see Figure 4.20) here a learning rate of 10^{-4} was used with $\beta_2 = 0.99$.

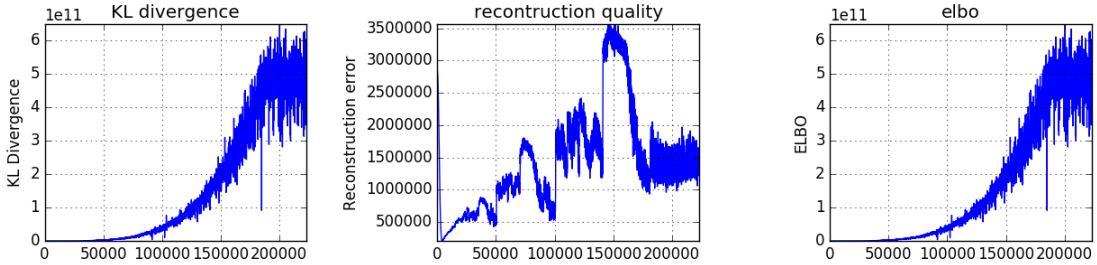


Figure 4.23: Oscillation in the KL-term prevents the reconstruction error from decreasing.

4.3.5 A Conclusion to VAE Acoustic

VAE proved to be hard to train and difficult in finding expressive model architecture. Batch normalisation was found essential for training, and appropriate hyper-parameters for the Adam optimiser were identified. By this, the models could be prevented from oscillating heavily. Anyhow, given the results at the end of the thesis, it must be concluded that none of the proposed models satisfies the first requirement of a VAE Acoustic (see Section 4.3.3) to produce the timbre of yodel music. However, in comparison to WaveNet UW with a total training time of 2 months, the VAE experiments had only a fraction of GPU time for improvement. The investigation "VAE Acoustic 512" showed promising as it has been improving until the end of this thesis, however slowly. It should be trained to convergence to see if it can produce yodel-like sounds. Experiments "VAE Acoustic 2048" and "VAE Acoustic 512, 2 sec" were stuck in a local optimum and should be restarted with the warm-up technique of Sønderby et al. (2016). Experiment "VAE Acoustic 1024" might still improve given a lowered learning rate. If done, it might be possible to draw a conclusion to VAE Acoustic for the task of yodel-music.

4.4 Conclusion to experiments

The experiments with autoregressive models revealed the preference of WaveNet over SampleRNN if measured on their generation time for new samples. WaveNet also demonstrated that it is capable of producing the timbre yodel music with high recognition factor. Experts attested that specific typical characteristics were provided by the model such as the switch from low to high notes, and back, or the yodel syllabus.

Experiments with conditioning showed the technique could be used to steer the models' behaviour but caused the model to converge very slowly. Different output distributions were investigated such as mixture models of logistic distributions, which showed the ability to model the signal and allowed to reduce the dimension from 256 to 60. However, the experiment reported inconclusive results as the model did not converge.

VAE proved difficult in finding appropriate architectures as well as in training the models. Due to the circumstances, the experiments with VAE had only seven days of experimental time which is a fraction of the time used for the other models. Therefore, more experiments have to be conducted in order to assess its use for yodel music.

To conclude, the autoregressive model WaveNet achieved to produce in a recognisable manner the timbre of yodel music while experiments with VAE failed in creating music, so far.

Chapter 5

Discussion

This chapter discusses the results of this thesis. Section 5.1 describes an analysis of the shortcomings of approaches based on autoregressive models to the audio waveform. In Section 5.2 non-autoregressive methods and their challenges are discussed, and Section 5.3 points out difficulties of comparing methods so far. An outlook is described in Section 5.4. It follows a classification and evaluation of the results of this thesis.

Yodel music differs from most other genres by using ecmelic notes, such as the nature fa. Nevertheless, it has been influenced which led to some yodel region to use them no more. Other characteristics like the transition from chest voice to falsetto with an audible glottal stop are denoted as unique characteristics of yodel. This thesis investigated if generative machine learning approaches can produce yodel-like music. So far, proposed procedures have used music in lead-sheet notation, which was played by a synthesiser, used piano music (van den Oord et al. (2016a)) or polyphonic music (Mehri et al. (2016)) that consisted of a classical orchestra. Approaches to music containing vocals were rarely found (Nayebi and Vitelli (2015)) and in quality less successful than for instruments. Up until this thesis, and to our best knowledge, it has not been tried to produce yodel music with machine learning algorithms.

The first research question (see Section 1.2) was focused on the characteristics of yodel music. Experiments conducted in this thesis confirmed the capabilities of WaveNet for music synthesis in the audio waveform. WaveNet-based models were able to produce the timbre of yodel music, and an evaluation by experts ascribed the generated samples to have specific yodel characteristics. The model reproduced a yodel-like vocalisation (yodel syllabus), high and low notes, a chorus, which sings around a keynote, and a voice like a yodeller. Therefore, it is stated as an achievement that a model outputs an audio signal which carries some yodel characteristics. For other typical attributes such as the glottal stop, there was no consensus if it is audible or not. When the samples were examined as a yodel composition, the shortcomings were evident. A melody structure and changes in metre were missed. In summary, a WaveNet-based model successfully produced the timbre of yodel but failed to compose yodel music.

The quality of recordings of the used audio files differs from data corpus used by other researchers (Mehri et al. (2016), van den Oord et al. (2016b)) as they include the noise of an audience. Many audio samples contain applause and other disturbance which might have prevented the models from further improvements. But a large data corpus offers the advantage that such effects should be minimised as these parts only make up for a fraction of the whole data. Contrary, Szegedy et al. (2013) discussed if artificially created noise may increase the performance for classification of images but reported mixed results. Therefore, the additional disturbance might have its effects, but it is unlikely for the outcome to have been entirely different.

The comparison of the sound quality from generated samples of this thesis with the results of Mehri et al. (2016) and van den Oord et al. (2016b) is almost impossible as an evaluation of a human singer against an instrument is likewise absurd. But Mehri et al. (2016) and van den Oord et al. (2016b) managed to produce samples without any unintended interruption such as the "slappings", described in Section 4.1.4. Because of the slappings, the presented results might be assessed by listeners as less

harmonic. If the model is trained further and this slapping effect vanishes, the outcome might be as harmonious as both published results. GRUV (Nayebi and Vitelli (2015)) used pop music as training samples and as a result, produced something pop-like. Compared with their results this WaveNet-based model achieved to create samples with voice-like singing, which was not found in the published samples of GRUV.

The second research question (see Section 1.2) investigated if the sound quality of produced samples is comparable to real yodel music. It was found that the proposed models of this thesis were unlikely to deceive an audience to be real yodel music. The qualitative analysis in Section 4.1.4 showed many weaknesses such as a missing melody structure or agogic. Thereby, this question is not answered by this thesis, and more research is necessary.

5.1 Where autoregressive models fail

The qualitative analysis in Section 4.1.6 uncovered that the samples do not contain melody structure, nor they have the agogic of yodel music. According to experts, the metre appeared monotonous and more random than intended. The question-and-answer game of yodel music was not observed in any generated samples. It seems that WaveNet is limited to the timbre of music and not able to catch up long-term structures such as melodies or metre. Experiments with enlarged sample sizes did not help, and it is doubtful that larger receptive fields will change the outcome if not increased drastically. Analysing the results of WaveNet proposed by van den Oord et al. (2016a) or SampleRNN by Mehri et al. (2016), it seems to show that the published samples include some parts which could be attested as agogic or melody structure. However, a closer examination reveals that it is difficult to identify some repetitive elements which would be an indication for a melody structure for all kind of music. Experts agreed to this finding and pointed out that the published samples are almost too short to recognise a melody (SampleRNN with Beethoven 30 seconds and WaveNet with piano 10 seconds). Hence, it must be concluded that autoregressive approaches for audio waveform fail so far to compose music.

In theory, both methods could increase their receptive field in some ways but at the cost of RAM resources. This limitation will be boosted by future GPU generations that contain more RAM.

The first experiment included yodelling with accordion. At some point, the model started to produce samples with a hand organ in the background. Despite the fact that the sound quality of the most trained model is much better, generated samples with accordion seem to have more of a melody structure than without, according to subjective perception. It leads to the hypothesis that instrumental music appears more structured than vocal music. A follow-up experiment trained with yodel and accordion might answer this question. Nevertheless, it is doubtful this leads to song compositions with 8-bar structures.

More research and resources are necessary for autoregressive models to compose music in the audio waveform.

5.2 Beyond autoregressive Models

Generative frameworks such as GAN, VAE and AVB were successfully applied to image generation (e.g. Goodfellow et al. (2014)). Approaches using these techniques for music synthesis to audio waveform are rare. A successful application is VQ-VAE proposed by van den Oord et al. (2017b) which can process an audio sample size of 32'768 ($\approx$ 1.4 s with sample rate 24'000) and achieved a remarkable sound quality for speech. While autoregressive models can output samples of any size, these generative frameworks are limited by resources to specific sample size as they output a training sample at once. It is seen, however, as an advantage that samples are created in a single process step thereby it is less prone to error propagation. No matter the gain, 1.4 seconds of music is definitively not long enough to be called a composition.

Approaches which report producing new compositions are mainly found based on lead-sheet notation. Colombo et al. (2016) stated that their model was able to maintain a rhythmical structure and created new tunes which carried features of the training set. However, the produced tunes were

of short length. The difference between music in the audio waveform and lead-sheet notation lies in its size. If music is available in lead-sheet notation and waveform, it is favourable to use an approach such as WaveNet-autoencoder (Engel et al. (2017)). It is a trainable synthesiser that might learn to play many timbres. The conditioning of the encoder could originate from a decoder of VAE which has learnt to compose music in lead-sheet notation. This way, the new composition would be generated with an own trained timbre. Because lead-sheet notation is not frequent for yodel music, this approach is not applicable. But any autoencoder is able for data compression and can create a compressed representation of yodel music. Thereby, generated compositions with melody structures are still in reach. The experiment VAE Acoustic was set up with this intention.

So far, VAE Acoustic showed difficulties in training and were not successful in producing anything else than noise. However, the results do not allow to conclude as more experimenting is needed. Several findings, which are described in Section 4.3.5, will contribute future experiments to improve. In van den Oord et al. (2017b), VQ-VAE reached good results for speech synthesis. It consists of strided convolutions only and has no dilated layers. The model showed that it has enough capacity to produce the characteristics of speech with high quality. Tests with an own adapted implementation of this network failed because of an incorrect application of batch normalisation, which prohibited any progress. Therefore, it should be re-evaluated for future experiments to measure its performance for yodel music.

5.3 Comparison of different approaches

The evaluation of techniques for music synthesis proved difficult as the approaches are almost not comparable. Some authors reported the log-likelihood over epochs (Nayebi and Vitelli (2015)), others used a mean opinion score (five-point Likert scale score in van den Oord et al. (2016a)) or an AB preference test (Mehri et al. (2016)) to evaluate their approach over another. None of them reported additional statistical quantities such as participant count and sample characteristics. It is comprehensible to use a different measure than the log-likelihood to evaluate a models' performance. Unfortunately, none of the scores mentioned above allows assessing approaches differently than qualitative.

For image generation, Salimans et al. (2016) introduced the inception score where a trained classifier is applied to the generated image to evaluate how much characteristics of the original pictures are recognised. Engel et al. (2017) adopted this idea to an inception score for instrumental musical notes in audio waveform. Thereby, future approaches might be comparable with this score. But so far, the quality of generated samples of a method are not assessable other than qualitatively.

More research is needed to get accurate measures which allow evaluation of an approach concerning quality.

5.4 Outlook

This section describes next experiments that should be conducted for improvement. Section 5.4.1 contains recommendations related to WaveNet and Section 5.4.2 to VAE. Section 5.4.3 describes the suggestion of how a music teacher would teach a model. A survey on the Swiss law and generative approaches is provided in Section 5.4.4. Last but not least a brief evaluation of generative approaches to industrial projects is given in Section 5.4.5.

5.4.1 WaveNet next steps

WaveNet-based models were found to work best with yodel music, so far. The model WaveNet UW was trained for three million training steps and produced samples with the best sound quality. As discussed, it suffers from shortcoming such as the slapping-effect which is disturbing. It was observed that these effects occasionally disappeared while training. Therefore, it is suggested to select a small, high-quality subset of natural yodel music without any disturbing noises such as clapping, screams and announcements and train the model for another 100k training steps to determine any progress. The result of this experiment will add explanation from where this slapping originates and answers the question if high-quality data can eliminate distracting behaviours of models. The findings might be useful for all future experiments with this data corpus.

If the above-described experiment does not lead to progress, the model should be trained with random noise only. Thereby, the model is forced to forget structures, and it is observable how fundamental this slapping-effect structure is. If it disappears after few training steps, then it might be solvable with data that do not contribute to this formation. Note that this experiments used Unterwalden yodel only. Yodel from other regions could solve the issue. If the slapping-effect disappears with or later as yodel, then it might be a very fundamental structure, and thereby, more effort is necessary.

5.4.2 Define VAE Acoustic

The experiments with VAE Acoustic were not finished at the time of this thesis (see Section 4.3.3). The model "VAE Acoustic 512" should be trained till convergence to have a baseline for VAE Acoustic. Meanwhile, the warm-up technique of Sønderby et al. (2016) is suggested to be used for the other proposed architecture. A change of the loss is necessary where an additional warm-up parameter $\gamma^{(t)}$ with t denoting the training step is introduced into Equation 2.71 which yields

$$L^V = -(1 - \gamma^{(t)}) \cdot KL\big(q_\phi(z|x) \| p(z)\big) + \mathbb{E}_{q_\phi(z|x)}\big(p_\pi(x|z)\big). \tag{5.1}$$

This warm-up parameter is granularly reduced from $\gamma^{(0)} = 1.0$ to $\gamma^{(t)} = 0.0$ over t training steps. The results with VAE showed that the KL-term reached zero around 50k training steps. It is suggested to set $\gamma^{(t)}$ to

$$\gamma^{(t)} = \max(0, \gamma^{(t-1)} - \frac{1}{50000}) \tag{5.2}$$

where $\gamma^{(0)} = 1.0$. "VAE Acoustic 2048" and "VAE Acoustic 512, 2 seconds" should both be restarted using the warm-up technique. The experiment "VAE Acoustic 1024" should lower the learning rate and decay rate β_2 as proposed (see Section 4.3.3) and use the warm-up technique described above. All four experiments will give evidence of how a VAE Acoustic is to be built. It is suggested to test the network architecture of van den Oord et al. (2017b) without vector quantisation at first. The network uses strided convolutions only and achieves a compression rate of 64. This architecture might increase the performance and compression rate of a VAE Acoustic. Additionally, its performance is comparable with the proposed models that use dilation and strides. The results will show if strided-only convolutions are more successful than combined convolutions.

Given a VAE Acoustic that fits the requirements vector quantisation should be implemented as proposed by van den Oord et al. (2017b) to test finite latent variable vectors. Future analysis about questions such as if yodel regions are explainable from their latent representation or if alphorn does share common properties with specific yodel region are more comfortable to answer with a simple representation of the latent variables. The performance loss of a VQ-VAE Acoustic is determinable as it can be compared with a VAE Acoustic. Vector quantisation needs effort to implement because

no gradient can be defined over the nearest neighbour function to determine the latent variables. van den Oord et al. (2017b) added the loss of the decoder's first directly to the encoder's last layer. It will be tricky, however, to implement this because the gradients would have to be manipulated which is done so far by tensorflow framework functions. Additionally, it has to be verified that introducing discrimination of latent variables into VAE does not limit its ability to generate new samples. van den Oord et al. (2017b) proposes an autoregressive approach to sample new latent variables which do not fit the intention of VAE Acoustic. In summary, this experiment allows exploring if a defined latent variable space can be used to model yodel music.

Given that VAE Acoustic reaches a sufficient large compression rate a "VAE Structure" should be built. Its purpose is to predict the structure of a song. The training data are produced by the encoder of VAE Acoustic that transforms chunks of a yodel song into the latent variable space and concatenates them keeping the order. If these long vectors are of the size around 32'000, it is possible (as shown by van den Oord et al. (2017b)) to build a VAE that predicts the whole vector at once. The decoder of the VAE Structure can now sample new song structures that are transformed by the decoder of "VAE Acoustic" to the audio waveform. This approach avoids resource limitation at the cost of training time as two networks have to be trained in sequential order.

5.4.3 Teacher learning

In van den Oord et al. (2017a), an approach is proposed to use trained WaveNet model as a teacher to teach a highly distributable model to output the same distribution. After training, the second model achieved an identical performance but with profoundly decreased process time. So far, there is no trained model available for yodel music. Thereby, the soprano Nuria Richner was asked after the analysis in Section 4.1.7 to imagine the model WaveNet UW as a yodel choir and suggest next lessons for further improvement. The shortcomings are outlined in the results section Section 4.1.7. The teachers' suggestions are translated into machine learning approaches, not all too serious, however. If conducted, it may be an indication whether expert knowledge of artists can help improving an algorithm's performance.

1. **Beginner: Learn the song form**

   Yodel choir In the first lesson, the yodel choir should learn a song. The teacher selects a song of simple song form and sings it together with the yodel choir. If necessary, the singing is accompanied by an instrument to accustom the yodel choir to harmonic structures. For starter, the teacher sings the song with instrumental accompaniment, and the yodel choir repeats it. The teacher suggests using a yodel song from UW as the yodel choir already exercised with such songs.

   In machine learning A simple yodel song from UW is selected. The model is to be trained such that it reproduces the yodel song almost correctly. In other words, the model should overfit heavily.

2. **Advanced: Precise singing**

   Yodel choir The singing of the choir so far is imprecise and unclear. The notes should be sung more defined and pure. Therefore, each is taught individually to lower to physical tension if it sings too high or to increase it if it sings too low. The teacher suggests covering one ear to hear themselves differently.

   In machine learning The model cannot be split up into single voices. Therefore, all have to be trained the same time. Regularisation can be introduced to control the "physical tension". So far this has not been used for the WaveNet UW model. Additionally, drop-out should be added to every 10th WaveNet block, so the model has different feedback.

Finally, *the yodel choir should sing playback to yodel songs and practice hard to accustom themselves to yodel and its melody structures. They should believe in themselves as they show great potential for yodelling*, (Soprano Nuria Richner of Schwyz, 10. January 2018, Lucerne).

5.4.4 Generative approaches and law

Annika Sonderegger is a lawyer admitted to the bar in Switzerland and a notary public for St. Gallen, Switzerland. She is a specialist in tenancy law, family law, labour law and intellectual property law and frequently negotiates cases in court. She studied law at the University of St. Gallen. The following is summary of an interview with her (see Section Appendix A.1) concerning generative approaches and law. The examination was done based on the Swiss law.

The Swiss copyright law (URG; SR 231.1) in application to music and generative approaches. Creation is an intellectual property (e.g. music, acoustic work, software). Intellectual property presumes a human author. An algorithm has, therefore, no intellectual property on its creations.

The law states that generated samples must differ from its origin, so they are not recognisable. If it is partially identifiable, then a licence is necessary. However, the author must prove infringement of a copyright. An algorithm must not change a music composition without proper permission. But it can learn a technique from specific music and use it for new creations (e.g. it determines that the note A followed by a G sounds harmonic). It is prohibited to produce music with the timbre of private persons. E.g. it is not allowed to publish photographs of people without their agreement. Caricatures of a public person (e.g. the voice is imitated) may be legal, but it must not be a deceitful imitation. A model that reproduces the sound of a singer recognisably would be a misapplication of personality traits which is illegal and strictly handled by the court. The yodeller can file a suit for abuse of the voice and reputational damage which is both handled by the criminal law. Anything that is technically feasible to prevent an infringement of the copyright must be done.

As long as generative approaches to music synthesis are used in a research environment, there is more freedom of action. Nevertheless, if results are published the mentioned statements should be considered.

5.4.5 Generative approaches for industrial projects

Generative approaches show great potential for industrial applications. Beside an audio cloud service which produces new music according to the user's preferences (see Appendix A.1) there are similar services conceivable like story writing. Other utilisation might be simulations for medical training such as for disease outbreaks. Despite the vast field of application, there are only few that do not need a large sample size. As discussed in Section 5.2 the methods are limited because of resource constraints and therefore not yet applicable to many challenges. However, autoregressive approaches might be successful for some of these tasks.

The results of Colombo et al. (2016) show that models succeed partially in creating creativity if the information is in compressed form. For example, successful industrial applications could be designed based on text such as the creation of new recipes. The training data is in dense form and the outcome not too large to suffer from limitations. However, the resulting text should be reviewed not to risk any poisoning. Similar, a newspaper could use the approach for the creation of short stories as entertainment. Or in combinations with conditioning, there might be a chance that new commercials could be created. Maybe at first, only the slogan is generated as an inspiration for a marketing specialist. Later, generative approaches might shape the entire advertisement.

However, the key to a successful industrial project lies in its final sample size. If it has to be large, then the project has a high risk to fail. If a small sample size is sufficient, it is worth a try.

Appendix A

Appendix

A.1 Interview: Generative approaches and law

Annika Sonderegger is a lawyer admitted to the bar in Switzerland and a notary public for St. Gallen, Switzerland. She is a specialist in tenancy law, family law, labour law and intellectual property law and frequently negotiates cases in court. She studied law at the University of St. Gallen.

Generative approaches use training data to learn specific probabilities. They are trained using a maximum likelihood estimation which maximises the likelihood that a sample is created like the training data under the generation process. Figure A.1 gives an overview of techniques that are relevant for evaluation in respect to the law.

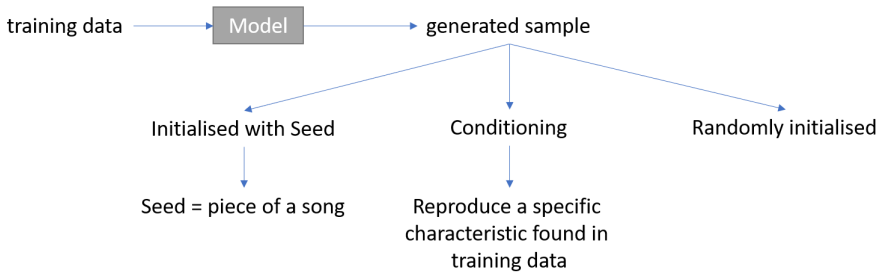


Figure A.1: Overview of the different aspects of the law to generative approaches.

In the following, cases are examined to Swiss law where a fictional company commercialises a generative approach to music. Section A.1.2 describes cases concerning the training data and Section A.1.3 generated samples.

A.1.1 Intellectual property rights

In the following, a summary is given from the Swiss copyright law (URG; SR 231.1) in application to music and generative approaches. Works are literary and artistic intellectual creations with an individual character, irrespective of their value or purpose (e.g. music, acoustic work, software). Thereby, intellectual property presumes an intellectual creation which is not accredited to software and animals likewise. Therefore, an algorithms' invention has no intellectual property. The following are examples which are copyright protected concerning music:

- Sequence of notes such as a tune
- A composition that is written on a paper
  - If a singer purchases a right to use and he/she sings that song, then he/she has the copyright to this sung version. If a CD is produced, then it is again copyright protected.
- The copyright expires for musical composition 70 years after the author's death

A.1.2 Training data

Example A.1. *A company wants to become a yodel music producer. For this reason, it plans to use a generative approach as described in this Thesis. The firm buys yodel CDs, purchases collections from SRF (Schweizer Radio und Fernsehen) and additionally, employees of the company gather yodel music through personal contacts with the organiser of yodel celebrations which recorded their events.* ⊖

Questions

1. The licence of CDs is restricted to personal use. Is it allowed to train a model with it and use the result commercially afterwards?

Lawyer: The result must differ from its origin, which must not be recognisable. If it is partially identifiable, then the right to change must be purchased. However, the author must prove infringement of a copyright. An algorithm must not change a musical composition. But it is allowed to learn a technique from an artist and to use this method. Therefore, if the algorithm determines that a note C followed by a G sounds harmonic, then it is safe.

2. SRF is a company and certainly clarifies any copyright issues. Is the company allowed to use these collections or is there anything else to consider?

Lawyer: It depends on what licences SRF possesses. If it is a private label, then SRF owns the copyright and can sell permits. The sales contract will usually regulate the further use.

3. The organisers of yodel celebrations are mostly private individuals. It is unclear whether the singers were informed that all is recorded and used for any purpose. Can these sources be used for training?

Lawyer: It depends on the case. For private individuals the personal rights are applicable. It is prohibited to produce music with the timbre of private persons (BV, SR 101 and ZGB, SR 210). E.g. it is not allowed to publish photographs of people without their agreement. For a public person, it is permitted if there is a general interest. If a caricature is made of a public person (e.g. the voice is imitated), it may be in alignment with the law. But if an algorithm echoes the timbre of a singer such that it is recognisable, then it is attempted deception and illegal.

A.1.3 Generated samples

Example A.2. *The company succeeded in training a model to an acceptable level and starts to generate samples to be sold. It initialises the model with a piece of real yodel song from one of the CDs and the model samples a new one but maintaining the structure such that the original, however different, still is recognisable. The new sample is sold and played by the radio. The composer of the original song recognises his song.* ⊖

- What actions can be taken by the composer?

Lawyer: The composer can request an omission in court. If accepted it is prohibited to play the song until the infringement of copyright is solved. Additionally, compensation and a share of the profit can be claimed. It applies to the song if the original is recognisable.

Example A.3. *The company trained their model with a technique called conditioning. It allows to steer some behaviour of the model and has been used in this case to generate samples of a different region (e.g. Berner yodel, Muotathaler yodel). Because the area Muotathal has one particular famous yodeller and the training data from Muotathal is mainly from his work the model creates a voice easily mistakable for him.* ⊖

- What actions can be taken by the yodeller?

Lawyer: The misapplication of personality traits is illegal and strictly handled by the court. The yodeller can file a suit for abuse of the voice and reputational damage which is both handled by the criminal law. Additionally, the competition law is applicable as it is a deception of the customer.

Example A.4. *The company produces new songs with the model and uses a random initialisation (random numbers are given as input). The produced new yodel songs are played on the radio. One is especially successful. However, a yodeller accuses the company to have stolen his work, and indeed it sounds very similar to one of his songs.* ⊖

- What has the company to do?

Lawyer: The copyright applies to intellectual individuals only. The company has a copyright on the software, but not on the song. If the yodeller sells the song, the company might file a suit based on the unfair competition act (UWG, SR 241). If the company can prove that the song was created first, then the yodeller would lose the intellectual properties. If it remains unclear, both can use the song.

Example A.5. *The company creates a web platform on which the model is hosted. For an appropriate fee, users can rent the model and listen to yodel music that is generated on demand. It means a yodel song is at most heard once. After that, the composition is lost.* ⊖

- Must the company prevent the model from reproducing an existing yodel song? What restrictions/ limitations exist?

Lawyer: Anything that is technically feasible to prevent an infringement of the copyright must be done. For the generation of new samples with a random seed, imitations must be avoided if it is foreseeable. If the company provides real songs as a seed, a licence is needed so the song can be altered. If the users provide the seed on their own, there is no infringement with copyright issues, as the private use of a protected material is allowed (Art. 19 URG). Conditioning must not be used to imitate personality traits as this is an infringement of their personality. But conditioning to a specific region is legal as is it does not infringe the personality of a specific person. Conditioning to a specific region does also not infringe on any intellectual property law as the programme learns the specific technique of the region and not a tune itself.

A.2 Neural Networks

This section deduces some functions concerning neural networks.

A.2.1 Derivative of Sigmoid Function

The sigmoid function is given by

$$g(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+a(z)} \text{ with } a(z) = e^{-z}$$

with $z \in \mathbb{R}$. The derivative of $a(z)$ is defined with

$$a'(z) = \frac{\partial}{\partial z} a(z) = -e^{-z}$$

We find the derivative of the sigmoid with

$$\begin{aligned}
\frac{\partial}{\partial z} g(z) &= -1(1+a(z))^{-2} a'(z) \\
&= (1+e^{-z})^{-2} e^{-z} \\
&= \frac{e^{-z}}{(1+e^{-z})^2} \\
&= \frac{1}{(1+e^{-z})} \frac{e^{-z}}{(1+e^{-z})} \\
&= g(z) \frac{e^{-z}}{(1+e^{-z})} \\
&= g(z) \frac{1+e^{-z}-1}{(1+e^{-z})} \\
&= g(z)(1 - \frac{1}{(1+e^{-z})} \\
&= g(z)(1-g(z))
\end{aligned}$$

A.2.2 Derivative of Cross-Entropy function

Let $\mathcal{L}$ be the cross-entropy function given in Equation 2.14. For simplicity, we set $m = 1$ which allows us to write the equation without summation over the sample count. Figure 2.9 represents the computation graph of a neuron which is considered here to be in the output layer. To change the weights we have to know the error of each inner activation $z_j^{(L)}$ of node j that can be found by their derivatives. The delta $\delta_j^{(L)}$ is given by

$$\delta_j^{(L)} = \underbrace{\frac{\partial \mathcal{L}}{\partial a_j^{(L)}}}_{C} \underbrace{\frac{\partial a_j^{(L)}}{\partial z_j^{(L)}}}_{B} = a_j^{(L)} - y_j \tag{A.1}$$

Note that the notation **B** and **C** has been chosen to match the notations of the following subsequent proofs.

We start with **B** that is given by

$$\frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} = \frac{\partial g(z_j^{(l)})}{\partial z_j^{(l)}} = g(z_j^{(l)}) \cdot (1 - g(z_j^{(l)})) = a_j^{(l)} \cdot (1 - a_j^{(l)}) \tag{A.2}$$

where $g(\cdot)$ denotes the sigmoid which is deviated in Section A.2.1. The derivative of $\mathbf{C}$ we find with

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial a_j^{(L)}} &= \frac{\partial - \left[\sum_{i=1}^{n^{(L)}} y_i \log\left(a_i^{(L)}\right) + (1 - y_i \log\left(1 - a_i^{(L)}\right)\right]}{\partial a_j^{(L)}} \\
&= -y_j \frac{1}{a_j^{(L)}} + (1 - y_j)\frac{1}{1 - a_j^{(L)}}(-1) \\
&= -\frac{y_j}{a_j^{(L)}} - \frac{1 - y_j}{1 - a_j^{(L)}} \\
&= -\frac{(1 - a_j^{(L)})y_j}{a_j^{(L)}(1 - a_j^{(L)})} - \frac{a_j^{(L)}(1 - y_j)}{a_j^{(L)}(1 - a_j^{(L)})} \\
&= -\frac{y_j - a_j^{(L)}}{a_j^{(L)}(1 - a_j^{(L)})}
\end{aligned}$$

As a summary, we have

$$\frac{\partial \mathcal{L}}{\partial a_j^{(L)}} = -\frac{y_j - a_j^{(L)}}{a_j^{(L)}(1 - a_j^{(L)})}. \tag{A.3}$$

By inserting Equation A.2 and A.3 in A.1 we get

$$\begin{aligned}
\delta_j^{(L)} &= -\frac{y_j - a_j^{(L)}}{\left(a_j^{(L)}(1 - a_j^{(L)})\right)}\left(a_j^{(L)}(1 - a_j^{(L)})\right) \\
&= -\left(y_j - a_j^{(L)}\right)
\end{aligned}$$

and finally, find the error of node j to be

$$\delta_j^{(L)} = a_j^{(L)} - y_j$$

□

A.2.3 Derivative of the MSE function

Let $\mathcal{L}$ be the MSE function given by Equation 2.16. To simplify the equation, we consider one sample only. Figure 2.9 represents the computation graph of a neuron. We assess the error δ_j of the activation $a_j^{(L)}$ of node j with the derivative over the loss. In contrast to the cross-entropy function, a regression output-node will not apply the sigmoid function on its inner activation $z_j^{(L)}$ because we want to output values $a_j^{(L)} \in \mathcal{R}$ without restrictions. Therefore, $z_j^{(L)} = a_j^{(L)}$ and we write the delta of the inner activation to be

$$\delta_j^{(L)} = \frac{\partial \mathcal{L}}{\partial a_j^{(L)}} = a_j^{(L)} - y_j \tag{A.4}$$

Therefore, we find the derivative with

$$\frac{\partial \mathcal{L}}{\partial a_j^{(L)}} = \frac{\partial \frac{1}{2}(a_j^{(L)} - y)^2}{\partial a_j^{(L)}} = a_j^{(L)} - y_j$$

and conclude

$$\delta_j^{(L)} = a_j^{(L)} - y_j$$

□

A.2.4 Derivative of the weights

We want to show that the gradient of the weight $\Theta_{ji}^{(l)}$ is given by

$$\nabla \mathcal{L}_{\Theta_{ji}^{(l)}} = \frac{\partial \mathcal{L}}{\partial \Theta_{ji}^{(l)}} = \delta_j^{(l)} \cdot a_i^{(l-1)} \tag{A.5}$$

with $\delta_j^{(l)}$ be the error of node j in layer l. In Equation 2.25 we use the chain rule twice in a right-to-left order to get the weight's gradient. We rewrite the equation here as

$$\frac{\partial \mathcal{L}}{\partial \Theta_{ji}^{(l)}} = \underbrace{\frac{\partial \mathcal{L}}{\partial a_j^{(l)}}}_{C} \underbrace{\frac{\partial a_j^{(l)}}{\partial z_j^{(l)}}}_{B} \underbrace{\frac{\partial z_j^{(l)}}{\partial \Theta_{ji}^{(l)}}}_{A} \tag{A.6}$$

For the following section, we restrict the activation function to the sigmoid. The derivative is given by Equation 2.23.

We start with **A** and find

$$\frac{\partial z_j^{(l)}}{\partial \Theta_{ji}^{(l)}} = \frac{\partial \left[\sum_{k=1}^{n_{l-1}} \Theta_{jk}^{(l)} \cdot a_k^{(l-1)} + b_j^{(l)} \right]}{\partial \Theta_{ji}^{(l)}} = a_i^{(l-1)} \tag{A.7}$$

If $l = L$ we know from the derivative of the loss functions given in Appendix A.2.2 and A.2.3 that **C** and **B** simplifies to $\delta_j^{(L)}$. By insertion A.7 and 2.24 into Equation A.6 we prove

$$\frac{\partial \mathcal{L}}{\partial \Theta_{ji}^{(L)}} = (a_j^{(L)} - y_j) \cdot a_i^{(L-1)} = \delta_j^{(L)} \cdot a_i^{(L-1)} \tag{A.8}$$

□

In the case of the inner layer $1 \leq l < L$ the expression has to take into account the error of the upper layers. From Equation A.2 we know the term **B** already.

C depends on the error $\delta_j^{(l+1)}$ of the upper layers. We examine $\delta_j^{(l)}$ for the inner layers $1 \leq l < L$ and find using the chain rule

$$\delta_j^{(l)} = \frac{\partial \mathcal{L}}{\partial z_j^{(l)}} = \underbrace{\frac{\partial \mathcal{L}}{\partial a_j^{(l)}}}_{C} \underbrace{\frac{\partial a_j^{(l)}}{\partial z_j^{(l)}}}_{B} \tag{A.9}$$

Remind that we want to express the gradient of a weight $\nabla \mathcal{L}_{\Theta_{ji}^{(l)}}$ as an expression of the error from the layer. This way we construct a recursion which can be computed efficiently by an algorithm. We want to have a formula as follows

$$\frac{\partial \mathcal{L}}{\partial \Theta_{ji}^{(l)}} = \delta_j^{(l)} \frac{\partial z_j^{(l)}}{\partial \Theta_{ji}^{(l)}}$$

We already know **B** that is given by Equation A.2. We express **C** as a function of the upper-layer errors and find

$$\frac{\partial \mathcal{L}}{\partial a_j^{(l)}} = \sum_{k=1}^{n^{(l+1)}} \underbrace{\frac{\partial \mathcal{L}}{\partial z_k^{(l+1)}}}_{E} \underbrace{\frac{\partial z_k^{(l+1)}}{\partial a_j^{(l)}}}_{D} \tag{A.10}$$

Note that we switched to left-to-right order application of the chain rule. Therefore, the sum of the input connections of the upper layer has to be introduced.

D simplifies to

$$\frac{\partial z_k^{(l+1)}}{\partial a_j^{(l)}} = \frac{\partial \left[\sum_{i=1}^{n^{(l)}} \Theta_{ki}^{(l+1)} \cdot a_i^{(l)} + b_i^{(l+1)} \right]}{\partial a_j^{(l)}} = \Theta_{kj}^{(l+1)} \tag{A.11}$$

E we get

$$\frac{\partial \mathcal{L}}{\partial z_k^{(l+1)}} = \frac{\partial \mathcal{L}}{\partial a_k^{(l+1)}} \frac{\partial a_k^{(l+1)}}{\partial z_k^{(l+1)}} = \delta_k^{(l+1)} \tag{A.12}$$

By insertion of A.11 and A.12 into Equation A.10 it follows

$$\frac{\partial \mathcal{L}}{\partial a_j^{(l)}} = \sum_{k=1}^{n^{(l+1)}} \delta_k^{(l+1)} \cdot \Theta_{kj}^{(l+1)} \text{ with } 1 \leq l < L \tag{A.13}$$

Furthermore, we use A.2 and A.13 and rewrite Equation A.9

$$\delta_j^{(l)} = \frac{\partial \mathcal{L}}{\partial z_j^{(l)}} = \underbrace{\sum_{k=1}^{n^{(l+1)}} \delta_k^{(l+1)} \cdot \Theta_{kj}^{(l+1)}}_{C} \underbrace{a_j^{(l)} \cdot (1 - a_j^{(l)})}_{B} \text{ with } 1 \leq l < L \tag{A.14}$$

We insert A.7 and A.2 into Equation A.6

$$\frac{\partial \mathcal{L}}{\partial \Theta_{ji}^{(l)}} = \underbrace{\frac{\partial \mathcal{L}}{\partial a_j^{(l)}}}_{C} \underbrace{a_j^{(l)} \cdot (1 - a_j^{(l)})}_{B} \cdot \underbrace{a_i^{(l-1)}}_{A}$$

Hence, we find

$$\frac{\partial \mathcal{L}}{\partial \Theta_{ji}^{(l)}} = \sum_{k=1}^{n^{(l+1)}} \delta_k^{(l+1)} \cdot \Theta_{kj}^{(l+1)} \cdot a_j^{(l)} \cdot (1 - a_j^{(l)}) \cdot a_i^{(l-1)} \text{ with } 1 \leq l < L \tag{A.15}$$

With Equation A.14 we finally get

$$\frac{\partial \mathcal{L}}{\partial \Theta_{ji}^{(l)}} = \delta_j^{(l)} \cdot a_i^{(l-1)} \text{ with } 1 \leq l < L \tag{A.16}$$

Accomplish Equation A.5 with A.8 and A.15 we write

$$\nabla \mathcal{L}_{\Theta_{ji}^{(l)}} = \delta_j^{(l)} \cdot a_i^{(l-1)}$$

$$\text{with } \delta_j^{(l)} = \begin{cases} (a_j^{(L)} - y_j) & \text{if } l = L \\ \sum_{k=1}^{n^{(l+1)}} \delta_k^{(l+1)} \cdot \Theta_{kj}^{(l+1)} \cdot a_j^{(l)} \cdot (1 - a_j^{(l)}) & \text{if } 1 \leq l < L \end{cases}$$

□

A.3 Sequence diagram of the training procedure

In Figure A.2 the Workflow Train used to train the models is shown as a sequence diagram.

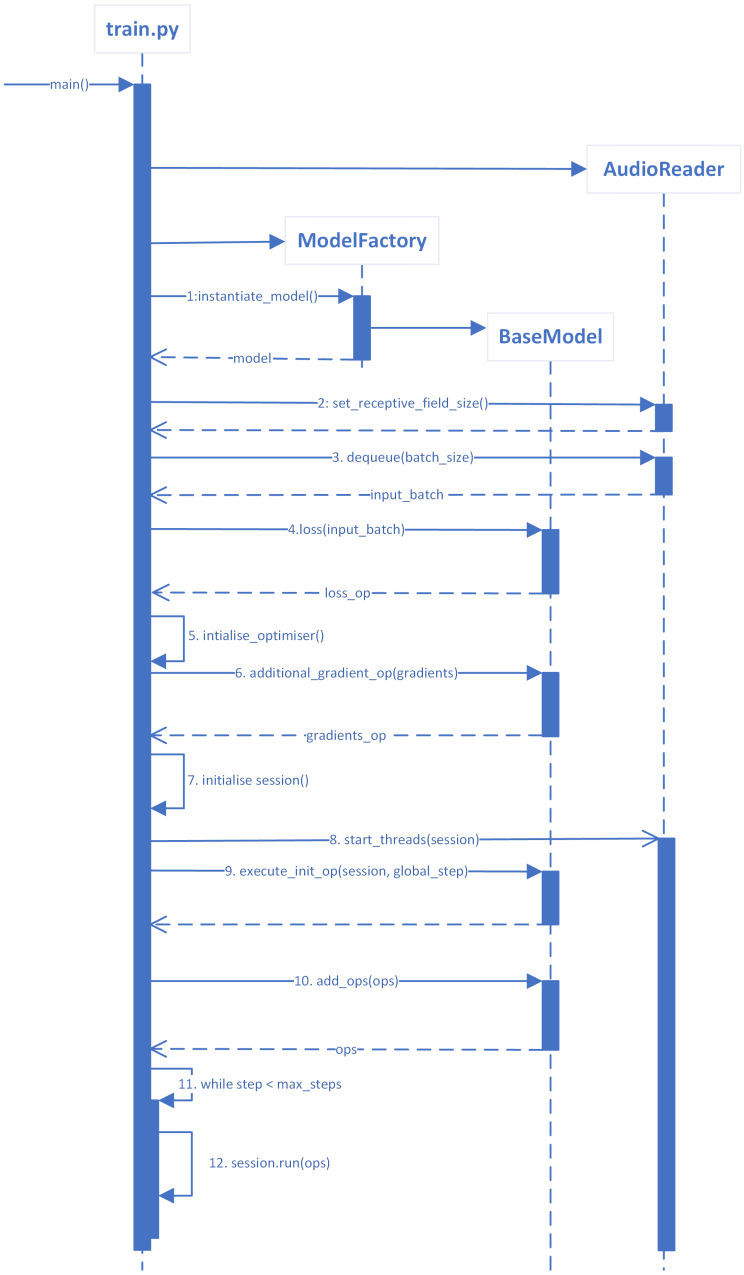


Figure A.2: Sequence diagram of the workflow for training.

A.4 Repositories and Results

All repositories and results are stored at the enterpriselab of HSLU I. It is necessary to connect by VPN to the HSLU internal network to get access to abiz-filehost.res.el.eee.intern. Make sure that your user can access the server over ssh with:

```
$ ssh <username>@abiz-filehost.res.el.eee.intern
```

All generated samples which are referenced by this thesis can be downloaded over SCP using the following command

```
$ scp <username>@abiz-filehost.res.el.eee.intern:/media/abiz/
    ↪ jodel/results/generated_samples/* <directory>
```

A.4.1 Data Corpus

The data corpus is accessible with git+ssh on:

```
$ git+ssh://<username>@abiz-filehost.res.el.eee.intern:/media/
    ↪ abiz/jodel/repo/music_raw/Jodelaudios
```

The KNIME workflows for data preparation are stored as a KNIME export and can be downloaded from

```
Host: abiz-filehost.res.el.eee.intern
Directory: /media/abiz/jodel/knime_workflows
```

Use

```
$ scp <username>@abiz-filehost.res.el.eee.intern:/media/abiz/
    ↪ jodel/knime_workflows/* ./
```

to retrieve the workflows.

A.4.2 Docker Repository

The docker repository is stored in Table A.5.

Name	**Description**	
Models and Evaluation	Git-Repository:	git@gitlab.enterpriselab.ch:abiz-lab/deep-yodeling-training-service.git
	***t*Branch**	**Description**
	master	Reviewed docker compose setup
	qa	Setup used for experiments on Switch Engine. Is to be reviewed
	local	Setup used for experiments at HSLU. The paths are mapped to be in the user folder
	testing	The docker image app has the vim installed. It used for manual experimenting.
Backup Job	Git-Repository:	git@gitlab.enterpriselab.ch:tapfaeff/docker_backup_job.git
	***t*Branch**	**Description**
	master	Docker image that synchronises models from the S3 Bucket with a specific training step to a folder.
	sync_everything	Docker image that synchronises the S3 Buckets to a folder

Table A.1: Source code repository.

A.4.3 WaveNet

In this section the connection details for access to results of WaveNet experiments are provided (see Table A.2).

Experiment Name	**Description**	**Repository**
WaveNet Default	Contains models, tensorflow event data and samples.	Protocol: scp, Hostname: abiz-filehost.res.el.eee.intern, folder: /media/abiz/jodel/↲ results/wavenet_run_v1/train
WaveNet UW	Contains models, tensorflow event data and samples.	Protocol: scp, Hostname: abiz-filehost.res.el.eee.intern, folder: /media/abiz/jodel/↲ results/wavenet_transferlearning_uw/
WaveNet Conditioning	Contains models, tensorflow event data and samples.	Protocol: scp, Hostname: abiz-filehost.res.el.eee.intern, folder: /media/abiz/jodel/↲ results/wavenet_conditioned_v2/
WaveNet Qualitative Analysis	Contains samples used for analysis.	Protocol: scp, Hostname: abiz-filehost.res.el.eee.intern, folder: /media/abiz/jodel/↲ results/wavenet_qualitative_analysis/

Table A.2: Repositories for WaveNet experiments.

A.4.4 SampleRNN

In this section the connection details for access to results of SampleRNN experiments are provided (see Table A.3).

Experiment Name	**Description**	**Repository**
SampleRNN Default	Contains models, tensorflow event data and samples.	Protocol: scp, Hostname: abiz-filehost.res.el.eee.intern, folder: /media/abiz/jodel/↲ results/samplernn
SampleRNN Mixture Model	Contains models and tensorflow event data.	Protocol: scp, Hostname: abiz-filehost.res.el.eee.intern, folder: /media/abiz/jodel/↲ results/samplernn_mixmodel

Table A.3: Repositories for SampleRNN experiments.

A.4.5 VAE Acoustic

In this section the connection details for access to results of VAE Acoustic experiments are provided (see Table A.4).

Experiment Name	**Description**	**Repository**
VAE Acoustic 512	Contains models	Protocol: scp, Hostname: abiz-filehost.res.el.eee.intern, folder: /media/abiz/jodel/↲ results/vae_wavenet_512_v2
VAE Acoustic 1024	Contains models	Protocol: scp, Hostname: abiz-filehost.res.el.eee.intern, folder: /media/abiz/jodel/↲ results/vae_wavenet_1024_v2
VAE Acoustic 512, 2 seconds	Contains models	Protocol: scp, Hostname: abiz-filehost.res.el.eee.intern, folder: /media/abiz/jodel/↲ results/vae_wavenet_2048_v2
VAE Acoustic 2048	Contains models	Protocol: scp, Hostname: abiz-filehost.res.el.eee.intern, folder: /media/abiz/jodel/↲ results/vae_wavenet_512_2sec_v2

Table A.4: Repositories for VAE Acoustic experiments.

A.4.6 Code Repositories

The code repository is accessible as given in Table A.5.

Name	**Description**	
Models and Evaluation	Git-Repository: git@gitlab.enterpriselab.ch:tapfaeff/Yodeling.git	
	***t*Branch**	**Description**
	Master	Contains source code for training of autoregressive models
	vae_wavenet	Contains the source code for working with VAE Acoustic
	samplernn	Experiment SampleRNN Default
	samplernn_mixmodel	Experiment SampleRNN Mixture Model
	transferlearning_wavenet	Experiment WaveNet UW
	wavenet_conditioned	Experiment WaveNet Conditioning
	vae_wavenet_512	Experiment VAE Acoustic 512
	vae_wavenet_512_2sec	Experiment VAE Acoustic 512, 2 seconds
	vae_wavenet_1024_v2	Experiment VAE Acoustic 1024

Table A.5: Source code repository.

The following dependencies exist:

- Python 3.5/ 3.6
- TensorFlow™ 1.4
- librosa 0.5 (McFee et al. (2017))
- numpy 1.13
- pandas 0.20

A.5 Curriculum vitae

Personalien

Name	Daniel Pfäffli
Adresse	St. Niklausengasse 9 6010 Kriens
E-Mail	daenu.pfaeffli@gmail.com

Education

2015 – 2018	Master of Science in Engineering, Lucerne University of Applied Sciences and Arts Machine Learning Promotion Master of Sciences in Engineering MSE Master thesis: Deep Neural Yodelling
2008 – 2013	Bachelor of Science, Bern Universiyt of Applied Sciences Computer Perception and Virtual Reality Promotion Bachelor of Science BUAS Undergraduate thesis: SquashI3D virtualisiert im CAVE/ Haptic
2002 - 2006	Gewerblich-Industrielle Berufsschule Bern, Bern Certified in Information Technology
2002 - 2006	Berufsmaturität Technische Richtung, Bern
1999 - 2002	Secondary education, Zollikofen
1992 - 1999	Primary education, Zollikofen

Professional Experience

2015 - 2018	**Scientific Assistent** Lucerne University of Applied Sciences and Arts, Rotkreuz Clustering, time-series prediction Machine Learning Continous Integration(Jenkins)
2012 - 2015	**Software Architect** Glaux Soft AG, Bern Software architecture Automated Testing Software development
2006 – 2012	**Project leader** Glaux Soft AG, Bern Project management Team leader Software development

List of Figures

List of Tables

Bibliography

Berthold, M. R., Cebron, N., Dill, F., Gabriel, T. R., Kötter, T., Meinl, T., Ohl, P., Sieb, C., Thiel, K., and Wiswedel, B. (2007). Knime: The konstanz information miner. In *Studies in Classification, Data Analysis, and Knowledge Organization (GfKL 2007)*. Springer.

Cattin, R. (01.02.2012). Signals in frequency domain.

Cho, K., van Merrienboer, B., Gülçehre, Ç., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078.

Chung, J., Gülçehre, Ç., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555.

Colombo, F., Muscinelli, S. P., Seeholzer, A., and Wulfram, G., editors (2016). *Algorithmic Composition of Melodies with Deep Recurrent Neural Networks*.

Dayan, P., Hinton, G. E., Neal, R. M., and Zemel, R. S. (1995). The helmholtz machine. *Neural computation*, 7(5):889–904.

Doersch, C. (2016). Tutorial on variational autoencoders.

Ellis, A. J. (1885). *On the musical scales of various nations*. London.

Engel, J., Resnick, C., Roberts, A., Dieleman, S., Eck, D., Simonyan, K., and Norouzi, M. (2017). Neural audio synthesis of musical notes with wavenet autoencoders.

Frey, B. J., Hinton, G. E., and Dayan, P. (1995). Does the wake-sleep algorithm produce good density estimators?

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial networks.

Greff, K., Srivastava, R. K., and Schmidhuber, J. K. (2015). Lstm: A search space odyssey. *CoRR*, abs/1503.04069.

Hinton, G., Srivastava, N., and Swersky, K. (01.02.2014). Overview of mini-batch gradient descent.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift.

ITU-T (1988). Pulse code modulation (pcm) of voice frequencies.

Johnson, D. D. and Weintraut, N., editors (2017). *Learning to Create Jazz Melodies Using a Product of Experts*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. pages 1097–1105.

Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86.

Le Cun, Y., Jackel, L. D., Boser, B., Denker, J. S., Graf, H. P., Guyon, I., Henderson, D., Howard, R. E., and Hubbard, W. (1989). Handwritten digit recognition: Applications of neural network chips and automatic learning. *IEEE Communications Magazine*, 27(11):41–46.

Leuthold, H. J. (1981). *Der Naturjodel: In der Schweiz: Wesen, Entstehung, Charakteristik, Verbreitung*. Robert Fellmann-Stiftung, Altdorf.

Martìn Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Corrado, G. S., Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Jia, Y., Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng (2015). Tensorflow: Large-scale machine learning on heterogeneous systems.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

McFee, B., McVicar, M., Nieto, O., Balke, S., Thome, C., Liang, D., Battenberg, E., Moore, J., Bittner, R., Yamamoto, R., Ellis, D., Stoter, F.-R., Repetto, D., Waloschek, S., Carr, C. J., Kranzler, S., Choi, K., Viktorin, P., Santos, J. F., Holovaty, A., Pimenta, W., and Lee, H. (2017). librosa 0.5.0.

McKinney, W. (2010). Data structures for statistical computing in python. In van der Walt, S. and Millman, J., editors, *Proceedings of the 9th Python in Science Conference*, pages 51–56.

Mehri, S., Kumar, K., Gulrajani, I., Kumar, R., Jain, S., Sotelo, J., Courville, A., and Bengio, Y. (2016). Samplernn: An unconditional end-to-end neural audio generation model.

Mescheder, L., Nowozin, S., and Geiger, A. (2017). Adversarial variational bayes: Unifying variational autoencoders and generative adversarial networks.

Nayebi, A. and Vitelli, M. (2015). Gruv: Algorithmic music generation using recurrent neural networks.

Pouly, M. and Kammermann, A. (2015). Projektförderung interdisziplinärer schwerpunkt datenwelten: Deep neural yodeling.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016). Improved techniques for training gans.

Salimans, T., Karpathy, A., Chen, X., and Kingma, D. P. (2017). Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications.

Salimans, T. and Kingma, D. P. (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks.

Sønderby, C. K., Raiko, T., Maaløe, L., Sønderby, S. K., and Winther, O. (2016). Ladder variational autoencoders.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In Dasgupta, S. and McAllester, D., editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA. PMLR.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2013). Intriguing properties of neural networks.

Tikhonov, A. and Yamshchikov, I. P. (2017). Music generation with variational recurrent autoencoder supported by history.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016a). Wavenet: A generative model for raw audio.

van den Oord, A., Kalchbrenner, N., Vinyals, O., Espeholt, L., Graves, A., and Kavukcuoglu, K. (2016b). Conditional image generation with pixelcnn decoders.

van den Oord, A., Li, Y., Babuschkin, I., Simonyan, K., Vinyals, O., Kavukcuoglu, K., van den Driessche, G., Lockhart, E., Cobo, L. C., Stimberg, F., Casagrande, N., Grewe, D., Noury, S., Dieleman, S., Elsen, E., Kalchbrenner, N., Zen, H., Graves, A., King, H., Walters, T., Belov, D., and Hassabis, D. (2017a). Parallel wavenet: Fast high-fidelity speech synthesis.

van den Oord, A., Vinyals, O., and Kavukcuoglu, K. (2017b). Neural discrete representation learning.

Vogel, M. (01.04.2013). Nichtlineare optimierung.

Wang, J., Yang, Y., Mao, J., Huang, Z., Huang, C., and Xu, W. (2016). Cnn-rnn: A unified framework for multi-label image classification. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2285–2294. IEEE.

Watkins, J. C. (November 2009). Topic 14: Maximum likelihood estimation.

Yao, K., Cohn, T., Vylomova, K., Duh, K., and Dyer, C. (2015). Depth-gated lstm. *CoRR*, abs/1508.03790.

Zeiler, M. D., Krishnan, D., Taylor, G. W., and Fergus, R. (2010). Deconvolutional networks. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2528–2535, Piscataway. IEEE.

Zilly, J. G., Srivastava, R. K., Koutník, J., and Schmidhuber, J. (2016). Recurrent highway networks.