Variational Autoencoders with Mixture Density Networks for Sequence Prediction in Algorithmic Composition

A Musical World Model

Viktoria Røsjø

Thesis submitted for the degree of
Master in Robotics and Intelligent Systems
60 credits

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Autumn 2018

Variational Autoencoders with Mixture Density Networks for Sequence Prediction in Algorithmic Composition

A Musical World Model

Viktoria Røsjø

Variational Autoencoders with Mixture Density Networks for Sequence Prediction in Algorithmic Composition

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Acknowledgements

Thank you to my parents, for emotional and financial support, allowing me to reach as far as my ambition would take me. Without you, this would not be possible.

Thank you to my excellent supervisor, Charles, for always being available to answer questions, brainstorm, and steer me in the right direction. He is the brain behind the idea of The Musical World Model, and I'm thankful that I got to work with it.

Thank you to my roomates, Hanne and Silje, for patiently listening to me complain about bugs and errors, and cheering me all the way to the finish line.

Thank you to all my ROBIN friends, for silly conversations and rubber ducking. I have learned a lot from you.

Thank you to all my other friends, that have shown me that they will be there waiting, when I come out of this master bubble.

You are all important weights in my network, and have contributed to this output!

Abstract

Does music contain a hierarchical component which is relevant when teaching a machine learning model to create music? And, can a machine learning model learn long term structure in music, based on its own perception of data?

In 2014, Diedrik P. Kingma and Max Welling presented a novel technique in generative modelling, called the Variational Autoencoder (VAE). The method presented a technique for learning intractable data distributions, and at the same time representing the data in a compressed latent space. From this latent space, it was possible to sample new datapoints, with similar features as those from the true data set. This method was quickly adopted for modelling real valued data, with both a fixed dimensionality, and in sequences. Through the course of 2017 and 2018, Google Brain released two variational autoencoders for sequential data: SketchRNN for sketch drawings, and MusicVAE for symbolic generation of music. These models inspire the variational autoencoder framework used in this thesis. The MusicVAE has a hierarchical element to assist in creation of music: a recurrent neural network function as a composer to manage the structural development of melodies. Their studies showed that the hierarchical component helped create more probable musical compositions than the formal VAE. MusicVAE is taken as a starting point for this thesis; however, rather than the recurrent neural network, a new architecture for generating high-level structure in music is introduced, using a mixture density network.

The Mixture Density Network, a network that can predict multi-valued output, was developed in 1994 by Christopher M. Bishop. The model can utilize any kind of network to condition the probability distributions. In 2018, David Ha and Jürgen Schmidhuber used a recurrent mixture density network (MDRNN) for predicting latent vectors in a reinforcement learning model. This has inspired the idea of replacing the recurrent composer from the MusicVAE with a MDRNN. This thesis introduces this novel architecture, in which musical compositions are guided by generating sequences of vectors from a VAE's compressed latent space. This is a novel architecture, in which compositions of music is guided by learned sequences of latent vectors. The model is named *Mixture Composer Variational Autoencoder*, or *MCVAE*.

Evaluation of the models showed that a difference in the models was noticeable. An evaluation with human annotators shows that music that has been composed by the MCVAE has noticeably better musical qualities than music generated from the formal VAE. Another evaluation, using a 5-gram model show that music made with guidance from the MDN creates melodies which are a lot more probable than music made without guidance.

Contents

List of Figures

List of Tables

Abbreviations

AE	Autoencoder
AI	Artificial Intelligence
ANN	Artificial Neural Network
BDRNN	Bidirectional Recurrent Neural Network
CAP	Credit Assignment Paths
CNN	Convolutional Neural Network
FNN	Forward Neural Network
GAN	Generative Adversarial Network
GUI	Graphical User Interface
HCI	Human Computer Interface
JS-divergence	Jensen-Shannon divergence
KL-divergence	Kullback-Liebler divergence
LSTM	Long Short-Term Memory
MAP	Max a posteriori
MDN	Mixture Density Network
MDRNN	Mixture Density Recurrent Neural Network
MM	Mixture Model
RNN	Recurrent Neural Network
RHS	Right Hand Side
VAE	Variational Autoencoder

Chapter 1

Introduction

> I frequently hear music in the very heart of noise.
>
> *George Gershwin [99, p. 177]*

1.1 Motivation

1.1.1 Algorithmic composition

Algorithmic composition is the process of creating music from a set of rules. This is an old concept, with examples from the 18th century, such as *Musikalisches Würfelspiel* - the musical dice game (see Figure 1.1). Created by Johann Philipp Kirnberger in 1757, its name was *"The ever-ready minuet and polonaise composer"*. The idea of the game was to encode the conventions of musical composition into a set of easily followed rules. Players could use the game to create new pieces of music to perform. Playing both the game and the music was supposed to be an enjoyable experience.

The game was constructed like this; for each bar, there exists 11 possible sequences. The user rolls two dice for each temporal unit, and choose a bar based on the dice's number [26, p. 36]. Musikalisches Würfelspiel became popular in Western Europe. Up to 1812, 20 different games were made. Among those, two games were released under the names of Haydn and Mozart. Although it has not been proven that they created the games themselves[51].

Current news about AI-generated art and music suggest that using algorithms to create music is as relevant today as it was in the 18th century [2]. In this thesis, I will explore one of the latest methods for algorithmic composition: the *Variational Autoencoder* (VAE). Additionally, by viewing music as a sequence of probability distributions, a *Mixture Density Network* (MDN) in tandem with a *Recurrent Neural Network* (RNN) is used as a composer network, taking advantage of the RNN's ability to predict a sequences and the MDN's ability to predict probability distributions. The novel model is given the name *Mixture Composer Variational Autoencoder*

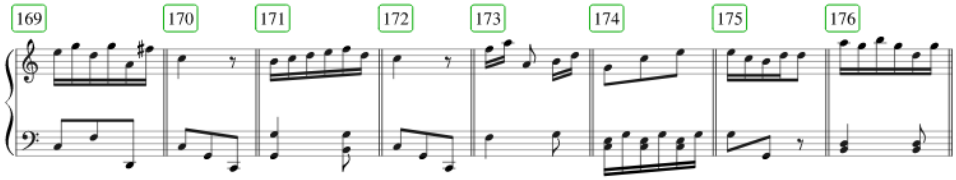
Musikalisches Würfelspiel

18th century

Rolls the dice 16 times (green row); Counts the points (orange column); Find the measure number (row/column intersection); Copy and paste in a score.

	1°	2°	3°	4°	5°	6°	7°	8°	9°	10°	11°	12°	13°	14°	15°	16°
2	96	22	141	41	105	122	11	30	70	121	26	9	112	49	109	14
3	32	6	128	63	146	46	134	81	117	39	126	56	174	18	116	83
4	69	95	158	13	153	55	110	24	66	139	15	132	73	58	145	79
5	40	17	113	85	161	2	159	100	90	176	7	34	67	160	52	170
6	148	74	163	45	80	97	36	107	25	143	64	125	76	136	1	93
7	104	157	27	167	154	68	118	91	138	71	150	29	101	162	23	151
8	152	60	171	53	99	133	21	127	16	155	57	175	43	168	89	172
9	119	84	114	50	140	86	169	94	120	88	48	166	51	115	72	111
10	98	142	42	156	75	129	62	123	65	77	19	82	137	38	149	8
11	3	87	165	61	135	47	147	33	102	4	31	164	144	59	173	78
12	54	130	10	103	28	37	106	5	35	20	108	92	12	124	44	131

(a) A dice game table



(b) A slice of a dice game music sheet, with numbered bars

Figure 1.1: An example of musical dice game from the music notation software Musescore [10]

or *MCVAE* (the model can be found at `https://github.com/vikrosj/music-variational-autoencoders`).

1.1.2 Markov models

In the 20th century, composers and computer scientists started to explore applying statistical models to algorithmic composition. In 1950, Harry F. Olson and Henry Belar developed the *"Electronic Music Synthesizer"*. The synthesizer used a *Markov model* to generate musical structure [26, p. 71]. A Markov model is a stochastic method for modelling randomly changing systems [107]. It does so by calculating the transition probabilities between adjacent events (see Figure 1.2).

Markov models have one defining limitation. When trained on a set of compositions, they can only model sub sequences that already exist in the data. These models cannot deduce any unknown musical sequences, or extrapolate new sequences from the given training data [85].

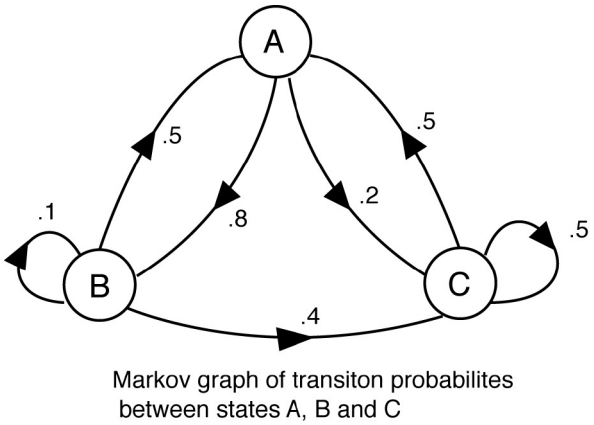


Figure 1.2: Visualization of a Markov model with three states: A, B and C, and the transition probabilities between them [45].

1.1.3 Creativity

Does rearranging existing sub sequences to make new music count as creativity? Psychologist and creativity researcher, Dean Keath Simonton stated to the American Psychological Association (APA) that "You can't be creative unless you come up with something that hasn't been done before. The idea also has to work, or be adaptive or be functional in some way; it has to meet some criteria of usefulness" [65]. This definition of creativity gives a guideline to what a creative model has to manage. It has to create something new, and at the same time be functional. A Markov model can be said to be somewhat creative, as it creates new arrangements of existing sequences. But, a fully creative model must be able to create *new sequences* of music, that it has not seen before. Secondly, the music must "meet some criteria of usefulness".

Arguably, creation of music is in itself useful, as it activates the reward system in the brain. Researchers has shown that listening to your favourite song flood the brain with dopamine [111], but the reason for this is not yet found. It could relate to the way in which music set up a pattern of expectation, and then fulfills them - in which the brain recieves a reward for predicting correctly [34]. Every human culture has their kind of music [75], which speaks to the fact that creating music is important. In this thesis, the *usefullness*-criteria is discussed, but not extensively investigated. What is investigated is the other side of creativity: *creating something new*. This is done by exploring the machine learning field of *generative modelling*, in which the ML-model seek to learn the true distribution of a data set, to produce new data points with some variations [42, p. 716].

1.1.4 Connectionism

Connectionism is a concept from cognitive science that dates back to the 1940's, but had it's breakthrough in 1986, with David E. Rumelhart and James L. McLelland two-volume compendium *Parallel Distributed Process-*

ing (PDP)[108]. Connectionism provided a new way of understanding the mind and how information is processed in the brain. The earlier idea was that a single neuron (or small bundle of) was concerned with each thing the brain had to process. This is explained via the concept "the grandmother cell", the idea of one specific neuron (or small bundle of), devoted to recognizing a persons grandmother [84]. But, evidence suggests that a thought involve a distributed pattern of activity across the cortex [121, Chapter 6]. This concept lead to connectionism in programming.

1.1.5 Recurrent neural networks (RNNs) for music modelling

Peter M. Todd, in *A Connectionist Approach To Algorithmic Composition* [125] used connectionism for music modelling. He described PDP and connectionism as a "paradigm for computing". It replaced the "strict rule-following behaviour with regularity-learning and generalization". The architecture of his network was inspired by a PDP approach by Michael I. Jordan, where each output of the network was reentered into the network via feedback connections [62]. Todd's work was the first to apply a recurrent network to generalize the structure of musical examples, and compose new pieces based on this generalization.

Music is temporal

Todd's paper also addressed a key factor for music modelling; music is a temporal process. Meaning that it is dependent of time, and often long term structure. Therefore, time must be represented somehow. The solution was either "a sliding window of successive time-periods of fixed size", where time is a spatial component. Or, time could be represented by "the relative position of a note in a sequence". Both these representations are used today, but in recurrent neural networks, the latter is used. Todd's network learned the sequence note by note. It had a two part input; one for the note and one for the *plan*. The plan input is a vector that is concatenated with the input note. It labels a full sequence with a number, every new sequence has its own number. In this way, Todd tested how interpolation between these numbers, or class labels, gave new melodies after training the network.

The work was revolutionary at this time, and as the author writes himself: "The possibilities for further work expanding the capabilities of this approach are virtually limitless". But, it also proved that the current standards of RNNs faced a problem with learning long term sequences. Todd writes: "the place for this network [...] is restricted to generating relatively short musical lines, high in local structure, but lacking in overall global organization".

1.1.6 Long Short-Term Memory Networks (LSTMs) for music modelling

LSTMs was developed by Sepp Hochreiter and Jürgen Schmidhuber in 1997, and was a revolutionary new technique for RNNs [54]. The method

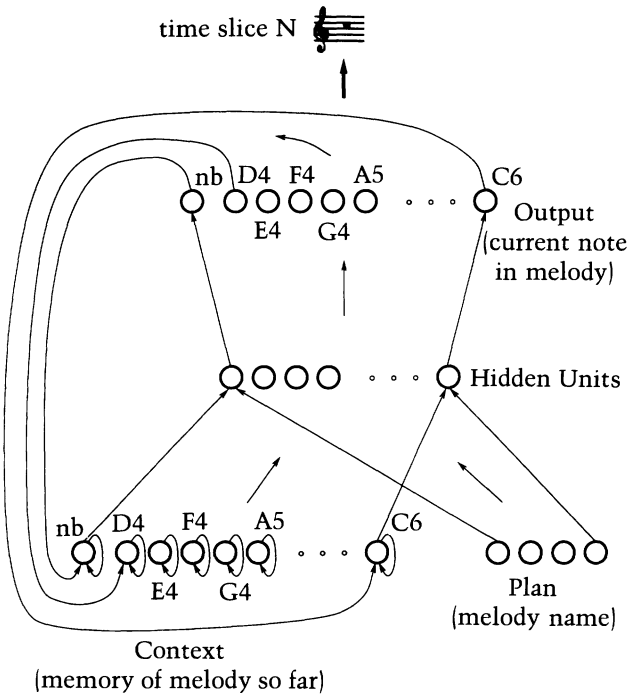


Figure 1.3: Unrolled graph of connectionist network by P.M. Todd [125].

allowed RNNs to learn long term structure in data. It is still the most popular RNN method today, although some variations also do exist [44]. Both RNNs and LSTMs are explained thoroughly in Chapter 3. Douglas Eck and Jürgen Schmidhuber used this method in their 2002 paper *Finding Temporal Structure in Music: Blues Improvisation with LSTM Recurrent Networks* [29]. The authors sought to prove that while a RNN can learn local, but not global structure in music, an LSTM can learn both. They train the network to produce both melody and chords of blues music. And, the result was that the LSTM did learn global musical structure, and thus could compose new blues pieces.

1.1.7 Variational Autoencoders

An autoencoder is an idea that dates back to 1987, originally used for feature learning and dimensionality reduction [42, p. 499]. An autoencoder is a neural network architecture where the goal is to reproduce the input. Autoencoders can be divided into three parts: the encoder, decoder and latent space. For dimensionality reduction, the latent space has fewer nodes than encoder and decoder, and thus works as a bottleneck (some autoencoders have a larger latent space than encoder and decoder, these are briefly discussed in Chapter 4). In the latent space, features in the data are represented as discrete values, and their value indicate how much of each feature is present in the current sample. The discrete representations causes the latent representation to be sparse (see Figure 1.4). But, since the network generalizes the input in the latent space, it is possible to interpolate

somewhat between the data points. Unfortunately, due to the sparsity of values in the latent space, the interpolation between known values is unlikely to be smooth [7].

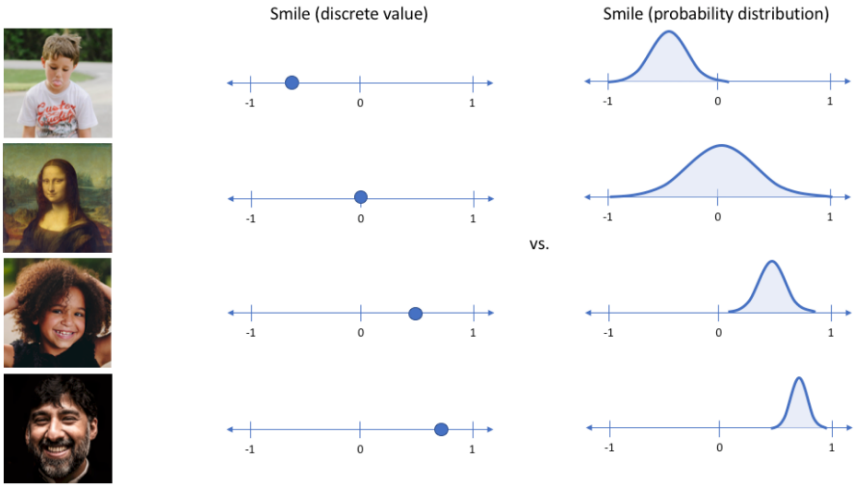


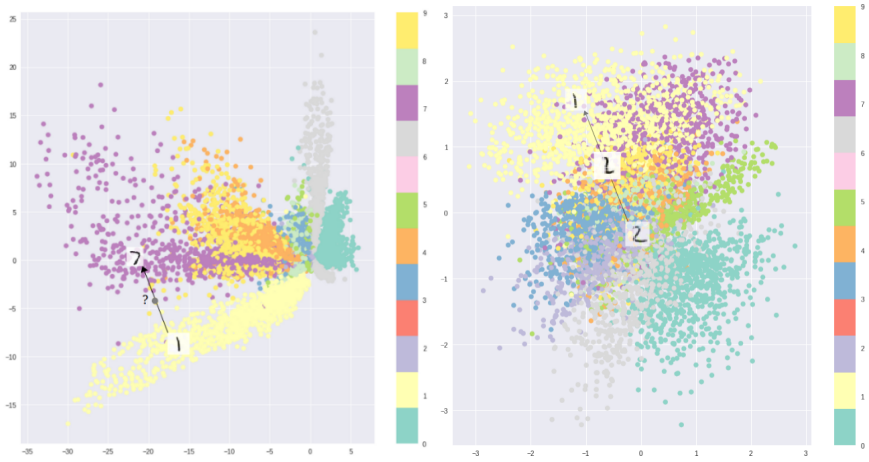
Figure 1.4: Comparison of AE and VAE latent space. The feature *smile* is presented with levels between $[-1, 1]$ [61].

In 2014, Diedrick P. Kingma and Max Welling published the paper *Auto-Encoding Variational Bayes* which explained a technique to address this issue applied in the so-called variational autoencoder (VAE) [132]. In a VAE, all data points are assumed to be distributions, and the latent space also represents data points as distributions. The latent space is initialized as a multidimensional Gaussian distribution. By enforcing a method called the *Kullback-Liebler divergence*, the latent space represent the points as Gaussian distributions. This method forces the latent space to be dense, and thus interpolating between real data points gives higher rate of semantic morphing than the autoencoder [115] (see Figure 1.5).

1.1.8 Variational Autoencoders

In 2016, Douglas Eck introduced the project *Magenta* at the Google Brain Team. The project wish to explore - in his own words - "Can we use machine learning to create compelling art and music? If so, how? If not, why not?" [30]. Motivated by these questions the team created, among others, two variational autoencoders with compelling results. The first of these was SketchRNN, which generates sketch drawings both conditionally and unconditionally [46]. It is trained on a large dataset of labeled sketch drawings, and learns to both reproduce input and create its own variants of labeled objects. The level of similarity between input and output is conditioned on a *temperature* level that indicate level of randomness during sampling.

In 2018, Magenta introduced MusicVAE in the paper *A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music* [104]. As the



(a) Autoencoder (b) Variational autoencoder

Figure 1.5: Example of the difference in latent space density between a autoencoder and a variational autoencoder, trained on the MNIST dataset [115].

title suggests, the decoder is a hierarchical model, designed to avoid a phenomenon in RNN decoders known as the *posterior collapse*. This is when the decoder model has so many neurons that it learns to memorize the entire data set. Thus, it can reproduce sequences, independent of the latent vector, and thus effectively ignore the latent vector in creation of sequences. The latent vectors are produced by a *conductor* RNN that outputs a sequence of vectors per song. And the decoder model MusicVAE only produce one sequence of a song, per latent vector. This level of hierarchy gave more pleasing musical results, than by letting the VAE predict long sequences without any conditioning.

1.1.9 Mixture Density Recurrent Neural Networks

An MDN is a neural net that predicts probability distributions [8]. In combination with an RNN, it can predict sequences of probability distributions. MDRNNs have been applied to various kinds of sequence modelling, from text and handwriting prediction [43] to musical control data in an interactive music application[81].

March of 2018, David Ha and Jürgen Schmidhuber introduced a "World Model" that made use of an MDRNN and a VAE. The World Model is a method for improving learning in reinforcement learning agents. The authors visualize the architecture by asking the question "Can agents learn inside of their own drams?" [48]. The dreams they are referring to are the latent representations of the real world. The MDRNN is trained to predict latent vectors, and then convert them to actions in the true data space. With this question in mind, the idea of a MDRNN for latent vector prediction of musical sequences came to be. The high-level question asked in this thesis is then: can a machine learning model create music from their dreams?

By combining the idea of a hierarchical model for creating music, along with the use of a MDRNN for sequence predictions, I have created a novel architecture, the *Mixture Composer VAE*, *MCVAE*. The model is a variational autoencoder, with a MDRNN for hierarchical control of latent vector predictions during music creation.

1.2 Research Questions

The thesis asks six questions:

1. What are the main technologies for using ANN to model and compose music?

In Chapter 4, the main technologies GANs and VAEs are explained in detail, with theoretical background from Chapter 2 on Artificial Neural Networks, and 3 on Recurrent Neural Networks and Mixture Density Networks.

2. In what way can music be represented to an ANN?

The main approaches for representing music to an ANN is discussed in the literature. For this thesis, a melody-only in the symbolic domain representation is chosen, explained in Section 6.1. Section 7.3 in the Discussion-chapter discuss the problem with time series data for a VAE.

3. How can a VAE be used to compose long compositions?

In Chapter 3, LSTM's advantage over vanilla RNNs for long term-depending structures is explained. Variational autoencoders and the implementation of LSTMs in them is clarified in 4.4.1.

4. How can we evaluate the success of a creative ANN model?

Creative ANN models are commonly evaluated by human annotators and by interpolating in the latent space. Details about these methods and how the evaluation for this thesis is done is explained in section 6.4.

5. Can a MDRNN be used to steer a VAE model of music?

The functionality and building blocks of a MDRNN is illustrated in Section 3.2, along with the way it is implemented in steering a musical VAE model. The question of whether it succeeds or not is evaluated in Section 6.6.

6. Does an MDRNN/VAE system produce better compositions than a VAE alone?

The findings of this thesis indicate that the MDRNN/VAE does produce better compositions, especially in terms of musical structure. The results are summarized in 6.7 and in the conclusion in Chapter 8.

1.3 Thesis Outline

Chapter 1. Introduction

Introduction to the thesis, and the motivation behind it.

Chapter 2. Artificial Neural Networks

Theoretical background for explaining the topics for the two following chapters.

Chapter 3. Sequence learning

Dividing the chapter in two, the first part elaborates on the most widely used machine learning method for sequence learning, namely LSTMs. The second part of the chapter delve into the limitations of FNNs for multi-valued output, explaining how MDNs are used to solve this problem. Lastly, clarifying why MDNs could be useful in sequence learning for VAEs, resulting in the reason this thesis investigate the topic.

Chapter 4. Generative modelling

Going into detail about the main technologies for using an ANN for creative purposes; GANs and VAEs. Looking into these methods used in image and art creation, and music modelling.

Chapter 5. Methods

Outlining the architecture of two state of the art ML models for generative modelling. Additionally, elaborating on a RL-method with MDRNN for sequence learning. These three inspirations combined make up a novel method presented lastly in this chapter.

Chapter 6. Experiments

Details of experiments conducted to evaluate the success of the models, with corresponding results.

Chapter 7. Discussion

Discussing a couple of discoveries and their effect on the results of the models. Also discussing a weakness in the VAE model when modelling sequential data, and possible workarounds for this problem. The creativity criterion mentioned in the introduction is discussed.

Chapter 8. Conclusion & Further work

In the conclusion all results are summarized, the research questions are addressed. Ending with some future directions to improve the results for

both models, and some remarks on the real world application for the novel model introduced in this thesis.

Chapter 2

Artificial Neural Networks

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.
>
> *Tom M. Mitchell, 1997 [53]*

This chapter introduces important concepts and terminology in machine learning, and particularly feedforward neural networks, that is further developed in chapters 3 and 4.

As the quote states, a computer program is said to learn from experience, performing a certain set of tasks, whilst measuring the performance.

As an example, consider a human being learning how to play a video game. The person playing performs a certain set of tasks, to complete each level. It is very likely that the person fails more than once, and then changes his or her strategy to complete the level. In this example, the performance can be measured in time, collected points per level and whether or not the person finished the level. By playing the game more than once, the person is *learning from experience*.

When it comes to machine learning, to learn from experience is referred to as *learning from data*. An algorithm is presented with data, and with the correct learning strategy, the algorithm can learn something from this data. Stanford University presents a straightforward definition:

> Machine learning is the science of getting computers to act without being explicitly programmed [93].

2.0.1 Types of machine learning

Typically, machine learning is divided into four groups [76].

Supervised learning. In supervised learning, the algorithm is presented with pairs of input and corresponding correct output. By training

on this data set, the algorithm learns to *generalize* so it outputs correctly to each input.

Unsupervised learning. Here, no correct outputs are provided. Instead the goal for the algorithm is to spot similarities in the data, and categorize them accordingly.

Reinforcement learning. In reinforcement learning, the algorithm is given a goal, but no information on how to reach it. It learns by itself how to reach the goal. It receives a reward for reaching the goal, and maybe for reaching a few sub goals. It may also receive penalties when the goal is not reached.

Evolutionary learning. The method sees biological evolution as a learning process. Inspired by how biological organisms adapt to survive, how their genes are distributed in reproduction, and how to measure *fitness* of an individual or a population are elements in this topic.

In this thesis, the model is trained with *self-supervised learning*, a sub field of supervised learning where the output data is equal to the input data. This method is used when the model seek to learn the underlying generator that creates the data.

2.0.2 Deep Learning

To distinguish if a neural net is either *shallow* or *deep*, the concept of *Credit Assignment Paths* (CAPs) was introduced by Jurgen Schmidhuber in the article *Deep Learning in Neural Networks: An Overview* (2014) [112]. A CAP is the chain of transformation that occur from input to output. The definiton of a deep model is not explicitly defined, but a CAP > 10 is considered "very deep". In the article, Schmidhuber also explains: "In a sense, RNNs are the deepest of all NNs". RNNs are generally applied to sequence learning, and is therefore applied in this thesis.

2.1 Feedforward Neural Networks

Feedforward neural networks, FNNs, form the foundation for recurrent networks, and is a subfield of supervised learning. The term feedforward describes the way information flows through the system. Information travels from input to output without feedback connections [42, p. 164]. A feedback connection is where the outputs of the system are fed back into the model. The outputs from the FNN are used to train the network in the backpropagation process, which will be explained further down.

Feedforward neural networks are called networks because they typically chain together different functions [42, p. 164]. For example, Equation 2.1 is a network consisting of three functions, connected in a chain.

$$f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x))) \tag{2.1}$$

This chain demonstrates the structure of a neural network. Each function is a layer. The inner function is the input layer, the outer function is the output layer, and any number of functions in between are the hidden layers. In supervised learning, the only known desired output is $f(x) = y$, meaning that the desired output of the inner layers are unknown. They are therefore called hidden layers.

The objective of this network is to approximate some function, f. For instance, a classifier $f(x) = y$, maps an input x to a class, y. The network defines a mapping $f(x;\Theta) = y$. Θ denote the distribution of the weight parameters. The network learns the parameters, Θ, that best approximate the function through comparing computed values to ground truth values [42, p. 164].

One defining aspect of the feedforward network is that no information is stored between computations. The network is looking for similarities and patterns in the data, but it does not interpret the data sequentially. To sequentially process data, feedback connections must be introduced to the system.

2.1.1 Single Layer Perceptron

The concepts of feedforward networks can be described with the example of the single layer perceptron, SLP, first introduced by American psychologist Frank Rosenblatt [106] in 1958. The SLP is a neural network with no hidden units, meaning it comprises only an input and an output layer. The chain function is described in Equation 2.2.

$$f(x) = f^{(1)}(x) \tag{2.2}$$

The input layer consists of n weights. The n number of input weights match the number of data points that are processed per iteration. The input layer also contains a bias and bias weights, so the output is shifted by the bias term. This is demonstrated at the end of this section. The output layer is one or more nodes with an activation function.

The process of learning an SLP start with the multiplication of the inputs and their corresponding weights. The sum of these products, or the *weighted sum*, is passed to the activation function [126].

2.1.2 Activation function

The activation function mimics a neuron in the brain. The goal of the function is therefore to *"decide"* whether the neuron should *"fire"* a signal or not. The signal and the threshold for firing it varies with each function. As mentioned, the activation function receives a *weighted sum* of its inputs and adds bias.

$$Y = \sum(weight \cdot input) + bias \tag{2.3}$$

The value of Y can be any number in the range $(-\infty, \infty)$. Thus, the activation function is necessary to implement a transformation on Y and create a firing threshold for the signal [126].

The activation function in the SLP output layer is classically a step function.

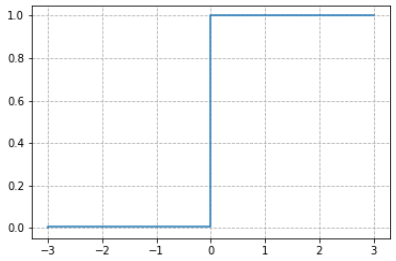


Figure 2.1: The step function.

The step function outputs either 0 or 1. This means that the the function is activated if Y is above some threshold, and not activated if Y is below the threshold. In Figure 2.1 the threshold is 0.

Defining the general activation function g at output o as:

$$o = g(x^t \cdot w + b) = g(Y) \tag{2.4}$$

The output of the activation function is:

$$o = \begin{cases} 1 & \text{if } Y \geq threshold \\ 0 & \text{if } Y < threshold \end{cases} \tag{2.5}$$

This function has limitations. Consider connecting multiple neurons to an output neuron in the case of distinguishing between more than two classes. Since the neuron only fires at 100% or 0%, deciding which class it predicts is impossible. For this task, a function with intermittent activation values is necessary [126].

In recurrent neural nets, the hyperbolic tangent *(tanh)* and logistic function (σ) are most typically used [73, p. 89].

The logistic function

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.6}$$

The logistic function has intermittent activation values, in the range $(0, 1)$. This is beneficial, because it enforces a boundary on the activation, so it doesn't blow up to $-\infty$ or ∞. Because of the benefits of this function, it is one of the most widely used activation functions today [126]. Albeit,

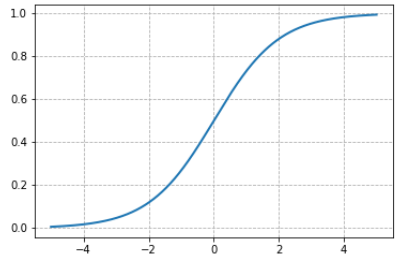


Figure 2.2: The logistic function.

there are complications with this functions also. Noticeably in Figure 2.2, towards either end of the output, the function responds less to changes in x. Meaning that for large inputs x, the function saturates and provides small changes to its output. This causes an issue called *vanishing gradients*, which will be discussed more extensively later.

The hyperbolic tangent function

$$tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} = 2\sigma(2x) - 1 \tag{2.7}$$

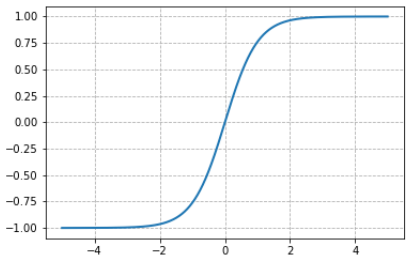
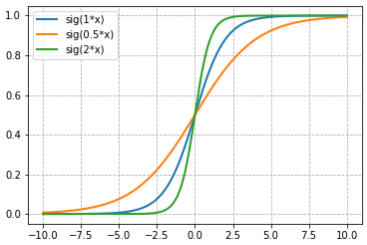


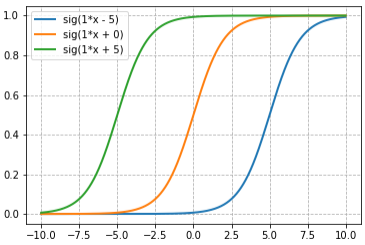
Figure 2.3: The tanh function.

The *tanh*-function is a scaled σ-function, so it has similar properties. The range of *tanh* is larger, binding the output to $(-1, 1)$. In the process of training the neural net, the gradient of the activation function is computed. The gradient of *tanh* is steeper than that of σ, so the choice of activation function depends on the requirement of gradient strength [126].

Bias added to the system is demonstrated below. In this example, the activation function g is the logistic function. Altering the weight w changes the *steepness* of the curve. But, the function clamps the input to a range of numbers between 0 and 1, with rotational symmetry at $(0, \frac{1}{2})$ [131]. To alter the inflection point, a bias must be inserted.



(a) Output without bias.



(b) Output with bias.

Figure 2.4: Logistic output function with and without bias.

2.1.3 Loss Function

The terms loss function and error function are used interchangeably in machine learning literature. In this thesis, *loss function* is used. The loss function is defined on a data point, prediction or label and quantifies the misclassification [55]. It does so with computing $E(Y, f(X))$ where $f(X)$ is a function predicting Y given values from input X. The loss function is used for penalizing errors in prediction. One example of a loss function is the square error computation [49]:

$$e(\theta) = ((f(x_i) \mid \theta) - y_i)^2 \tag{2.8}$$

Cost Function

The cost function is a sum of loss functions over the training set [55]. Both the cost and loss function are used for penalizing errors, but they are used for different learning methods [76, p. 82]. The most common cost function is the Mean Squared Error function (MSE) [71]:

$$MSE(\theta) = \frac{1}{N} \sum_{i=1}^{N} ((f(x_i) \mid \theta) - y_i)^2 \tag{2.9}$$

2.1.4 Backpropagation

The backpropagation algorithm is a method for automatic differentiation of complex, nested functions. The counterpart to automatic differentiation is hand-engineering, where experts hard code the features they know enhance learning in the neural nets. [60].

For the FNN, the backpropagation algorithm requires two things [60]:

1. A dataset
2. A loss or cost function

And the dataset most commonly consist of three sets [76, p. 20]:

1. A training set
2. A validation set
3. A test set

The *training set* is the part of the dataset that is used to train the model. The *validation set* is used to keep track of the models performance, hence where the loss function is applied. Lastly, the *test set* is used once the model is completely trained to provide an unbiased evaluation of the final model [116].

For each instance in the training set, the network computes the output of every neuron in each layer. Finally, it measures the output on the last layer, and compare this to the *true* answer. By doing this comparison, the networks error is computed for each neuron. With this information, the network computes how much each neuron contributed to the error in the successive neuron it is connected to. It does this by computing the error gradient with respect to the size of the neuron. The size of the neuron is typically called its *weight*. This computational process continues backwards in all the previous layers of the network, until the input layer is reached [37, p. 261].

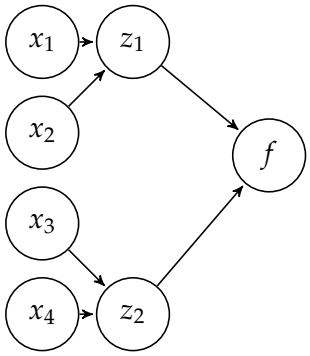


Figure 2.5: A network with sparse connections

Computing the gradient Figure 2.5 is an example of a sparse net. The nodes are multiplicative. As previously mentioned, the gradient is computed with respect to each input neuron. A few, selected gradients from the figure above look as follows:

$$\frac{\delta f}{\delta z_1} = \frac{\delta(z_1 \cdot z_2)}{\delta z_1} = z_2 \tag{2.10}$$

$$\frac{\delta f}{\delta x_2} = \frac{\delta f}{\delta z_1}\frac{\delta z_1}{\delta x_2} = x_1 \tag{2.11}$$

$$\frac{\delta f}{\delta x_4} = \frac{\delta f}{\delta z_2}\frac{\delta z_2}{\delta x_4} = x_3 \tag{2.12}$$

Overfitting and underfitting

The images in Figure 2.6 exemplify the problems that occur when training neural network to approximate a function, using linear regression. It shows that MSE is not always indicative of a good solution. Model number two, a fourth degree polynomial, adapts best to the original function. The third model is apparently not a good representation of the original function - yet, it has a smaller MSE. This is known as *overfitting*. The function has adapted perfectly to the data, but it does not *generalize* well. Many of the points in a dataset may be noise, and when the final model is overfitted, it has also learned the noise. Likewise, the first model does not generalize well. It is almost not adapted to the datapoints at all. This is known as *underfitting*. Both overfitting and underfitting prevents the neural net from learning, therefore there exists many methods to avoid these events from happening [14].

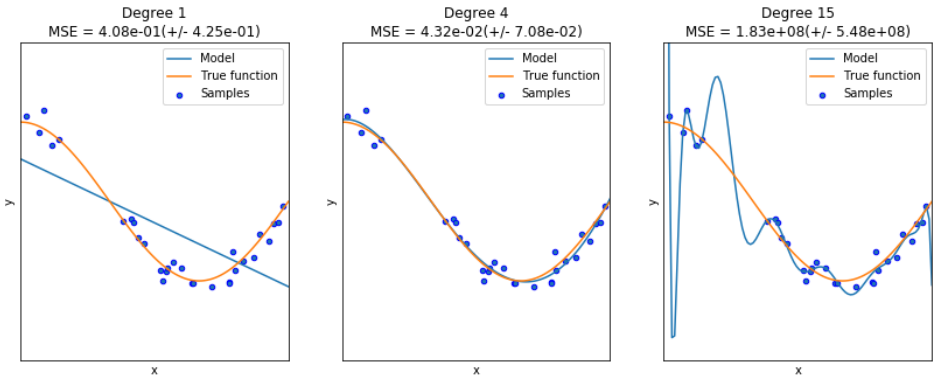


Figure 2.6: Training a neural net, example from scikit-learn[113].

Batch, sequential or stochastic training

The learning method explained at the beginning of this section is what's known as *sequential* training. This is where the error is computed and the weights are updated after each input. This is not necessarily the most efficient method, but it may have better results than *batch learning*, because it converges slowly [76, p. 82]. Batch learning is where all of the training data is presented to the network, in what is called an *epoch*. The cost is computed, and then the weights are updated once every epoch. The training data can be divided into smaller pieces in what's called *minibatches*, and the gradients are updated for every pass through a minibatch. One benefit of batch training is that the analysis of the convergence rate and the weight dynamics are simpler [71]. Lastly, *stochastic* or *online* training is where a single example is chosen randomly from the training set at every iteration. An estimate of the gradient is then computed, and the network is

updated. This is a preferred method for basic backpropagation, because it is faster than batch learning. It often results in better solutions, particularly for redundant datasets [71].

Weight initialisation

Initialisation of the weights is important for the progress of the network. Most importantly, they cannot all be the same number. This is due to the gradient computation. If the weights are equal, the gradients and the parameter update will also be equal. The weights will then learn the same thing, and they're not separable. It is therefore common to initialise the weights to small, random numbers. The concept is that they will compute distinct updates and integrate themselves as divergent parts of the full network, when they are random and unique in the beginning [63].

Vanishing gradients Although, in some cases, small weights leads to computing small gradients, because the gradient is proportional to the weights. The gradient is repeatedly multiplied through the network, thus very small gradients can lead to a slow or no update of the network. This is what's referred to as *vanishing gradients*. As previously mentioned, for sigmoidal functions, large inputs can cause the function to saturate and also produce a small change in output. Both of these instances cause vanishing gradients.

Exploding gradients On the other hand, repeatedly multiplying gradients that are larger than 1 might cause the gradients to grow too large, causing *exploding gradients*. This may lead to an unstable model that is unable to increase the loss during training [12].

Xavier initialisation A method for weight initialisation introduced in 2010 is the *Xavier initialisation*, where the weights are drawn from the Gaussian distribution with zero mean and variance at

$$var(w_i) = \frac{1}{N_{avg}} \tag{2.13}$$

$$N_{avg} = \frac{N_{in} + N_{out}}{2} \tag{2.14}$$

N is the number of input and output neurons. This method keeps the variance of the weights across the layers, and reduce the problem of vanishing gradients. It is recommended to use this method for *tanh* networks. There is also a version for *ReLU* networks. [38].

2.1.5 Softmax

The softmax function is often placed at the output layer of a neural network. It is a classification function, commonly used in multiclass learning problems where each input is categorized into more than two categories, and the categories don't overlap. [120]. The softmax function computes the normalized probabilities for each class.

$$softmax(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{2.15}$$

The exponentiation is due to interpreting the scores inside the vector as the unnormalized logarithm of the probabilities. As the logarithm is the inverse operation of exponentiation, log loss is commonly used as a loss function together with the softmax activation function [90].

2.1.6 Cross Entropy

The multiclass cross-entropy is also known as negative log likelihood, as they are two different interpretations of the same function [49, p. 32].

$$C = -\frac{1}{n}\sum_{x}(y \ln a + (1-y)\ln(1-a)) \tag{2.16}$$

Equation 2.16 is the negative log likelihood of the Bernoulli distribution, which is a discrete distribution with two possible outcomes [3]. Here n is the total number of training data points. The sum is over all training inputs, x, and y is the corresponding desired output. The probability mass function for the Bernoulli distribution is shown in definition 2.17.

$$o(k;p) = \begin{cases} 1-p & \text{if } k = 0 \\ p & \text{if } k = 1 \end{cases} \tag{2.17}$$

The variable, k, takes value 1 with probability p, and value 0 with probability *1-p*. Equation 2.18 is the negative log likelihood of the binomial distribution, which is a discrete probability distribution. In other words, the cross-entropy of a multiclass version of Bernoulli [4]. Viewing the softmax function, it is visible that the log likelihood is logarithm applied to a softmax output.

$$L_i = -log(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}) \tag{2.18}$$

The cross-entropy is reasonable to use as a cost-function for classification [94]. This is because the individual parts of the sum are all negative because they are the logarithms of the softmax output (in a range between zero and one). Hence, the sum is negative, but then this sum is then again

negated. Thus, the output of the cross-entropy function will always be positive. The higher the probability for the correct class gets, the smaller the likelihood gets. Since the function is used to calculate the distance between the correct outputs and the computed output, this means that the closer the predictions get to the truth, the lesser the distance get. Consequently, the loss is minimized.

Chapter 3

Sequence learning

> It's something you learn after your second theme party, it's all been done before.
>
> *Prior Walter, Angels in America [70]*

In this chapter, sequence learning is divided in two sections: Recurrent Neural Networks (RNNs) and Mixture Density Networks (MDNs). RNNs and their ability to perform sequence prediction is explained. Starting with vanilla RNNs, their structure and limitations are discussed, including the problems they face with learning long term structure . This leads to introducing Long Short-Term Memory Networks (LSTMs), a specialised RNN unit design that handles long term structure. Mixture Density Networks (MDNs) can represent conditional probability distributions just as a commonplace neural net can represent arbitrary functions. Finally, explaining the need for the combined use of these (in algorithmic composition) concludes the chapter.

3.1 Recurrent Neural Networks (RNNs)

As the quote suggests, current events may have similar features to previous ones. So, to see the broader picture of an event, i.e. it's context or development, perhaps a good idea is to remember important parts of the past. This is the concept of recurrent neural nets, RNNs. In all RNNs, the past is an input to the future. And in some RNNs, the algorithm has measures for remembering, forgetting and selecting parts of data that ultimately should be remembered in the future. This way, an RNN can process sequences, more powerfully than a FNN can do [42, p. 368].

3.1.1 Parameter Sharing

The definition of recurrence is that something is returning or happening time after time [88]. Examine the classical function for a dynamical system:

$$s^{(t)} = f(s^{(t-1)}; \theta) \tag{3.1}$$

The function 3.1 is a recurrent function, since the function at time t refers back to the same function at time t - *1*. Here, the function f is used at every time step in a sequence from $t = 1, ..., \tau$. The same method takes place in a RNN; the same formula f is used at every time step, to process a vector x. The function f use the same set of parameters for every time step [73, p. 20]. This feature is called *parameter* or *weight sharing*.

Consider the case of training an FNN to process sentences of fixed length. The network would have separate weights for each input feature. Therefore, in each position of the sentence, the net would have to learn the rules of the language separately. Because the FNN does not master sequential processing, the net is compelled to learn the rules independently of the previous computations. In contrast, because the RNN shares the same weights across several time steps, it can therefore detect rules and sequences [42, p. 368].

3.1.2 Vanilla RNN

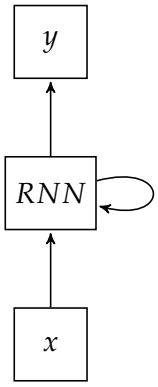


Figure 3.1: RNN illustration

In general, a recurrent neural net has a recurrent core, which takes an input, x, and updates its hidden state, h. This hidden state is then fed back into the core at the next time step, t, when the next input is fed into the model. Usually, at a frequent rate, the goal is to get an output, y, from the RNN [73, p. 19]. The general hidden state function is similar to function 3.1.

$$h^{(t)} = f_W(h^{(t-1)}; x^{(t)}) \tag{3.2}$$

Where:

$h^{(t)}$ current state

$h^{(t-1)}$ previous state

f_W function with parameters, W

x_t input vector at time step, t

Note that the same function, f_W, and the same parameters, W, are used in every step of the computation. In the standard RNN, the hidden state, h, consists of a single layer, usually a *tanh*-function [28].

$$h^{(t)} = tanh(W_{hh}h^{(t-1)} + W_{hx}x^{(t)}) \tag{3.3}$$

$$y^{(t)} = W_{hy}h^{(t)} \tag{3.4}$$

Where:

tanh hyperbolic tangent function, to create non-linearity

W_{hh} weight matrix for previous hidden state

W_{hx} weight matrix for input, $x^{(t)}$

W_{hy} weight matrix to compute output from state $h^{(t)}$

$y^{(t)}$ output, y, at time, t

Computational graph

In Figure 3.1 the intermediate computations are hidden. To get a clearer view of the inner workings of the RNN, an *unrolled computational graph*, as shown in Figure 3.2 and 3.3, is useful.

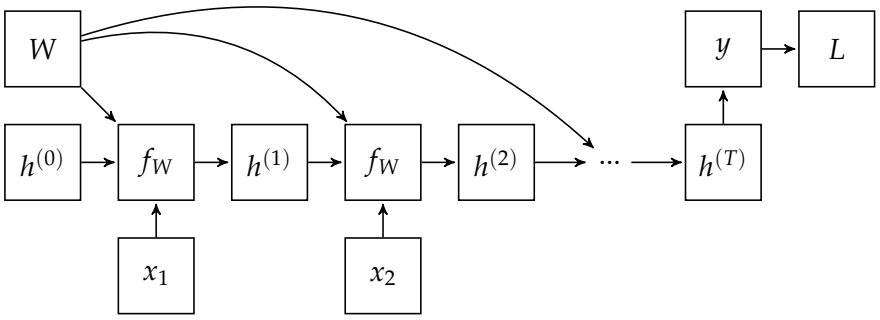


Figure 3.2: Many to one-architecture

Figure 3.2 is an example of a *many to one*-architecture, in which a new input, x, is provided at each timestep, t, and the output, y, is computed at the end of a timeseries. In this case, the *loss* is computed at time T. In this architecture, a variably sized input is transformed into a single vector output. As mentioned, the same weight matrix, W, is used at each time

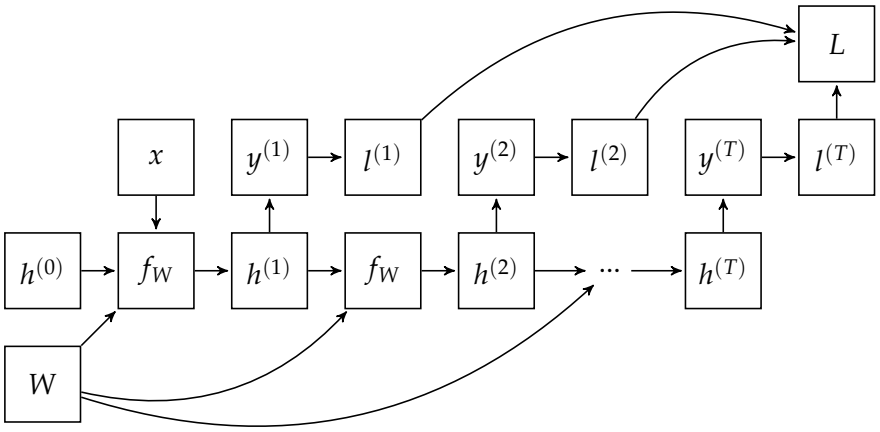


Figure 3.3: One to many-architecthure

step. When computing the gradient for W in backpropagation, $\delta W^{(t)}$ is then computed at each time step. The final δW is the sum of the gradients from $t = 0$ to $t = T$. This is used to update W before next forward pass [33].

Figure 3.3 is a *one to many-architecture* RNN, receiving a fixed size input, and producing a variably sized output. In this case, the model is making predictions at each time step and computing intermediate loss, and lastly summarizing this into a total loss at time T [33].

Sequence modelling

For sequence modelling, used for instance in language translation, both the input and the output are of variable sizes. This is typically solved by a *many to one*-network, followed by a *one to many*-network. This is also known as a *Encoder-Decoder Network* [13] (see Figure 3.4). The *many to one*-network receives a variably sized sentence vector and *encodes* this to a fixed size output. Then, the *one to many*-network receives the fixed size input and *decodes* this to a new, variably sized sentence vector.

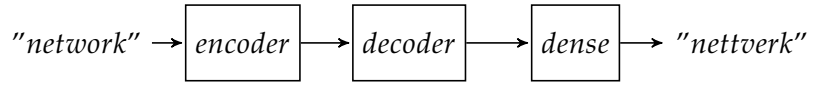


Figure 3.4: Encoder-decoder network translating from english to norwegian.

3.1.3 Long and short term dependencies

Consider reading a sentence, and then trying to predict the next word. For instance *"I'm writing a master thesis about [blank]"*. To fill in the blank here, only relying on recent information, can prove to be difficult. The information to finish the sentence may have been given a long time ago. This exemplifies the difference in short and long term dependencies.

RNNs handle short term dependencies, but struggles with long term dependencies [95]. In practice, the problem a vanilla RNN faces is that the error tend to vanish or explode in backpropagation. This is due to the recurrent update of the hidden state, where the weight matrix W_{hh} is multiplied with the differentiated function for each update of the function in the backwards pass. This causes the gradient to increase or decrease exponentially with time when the absolute value of the gradients are less than or larger than 1, respectively [54, p. 4]. This inspired a new version of RNNs, called *Long Short Term Memory Networks*.

3.1.4 Long Short Term Memory Networks (LSTMs)

LSTMs (see Figure 3.5) are a special kind of RNNs, and they are specifically designed to tackle the long-term dependency problem. In the original paper, by Sepp Hochreiter and Jurgen Schmidthuber, the authors claim that "In comparisons with RTRL, BPTT [...], LSTM leads to many more successful runs, and learns much faster. LSTM also solves complex, artificial long time lag tasks that have never been solved by previous recurrent algorithms" [54, p. 1].
RNNs store representations of recent events in form of activations. Storing recent events is analogous with a short term memory. But, it does not select what is important to remember, but rather remembers everything. What an LSTM does is introduce *gating functions* that can open and close access to the gradient flow. Thus, allowing the gradient to flow unchanged upstream, and also selecting what parts of past events that are beneficial to remember. A long term memory is characterized by slowly changing weights. An LSTM store recent events, but it does allow the weights to flow unchanged through time, thus the name *Long Short-Term Memory*.

Whilst the repeating module in the standard RNN contains a single layer, the LSTM contains two. These are the *cell state* and the *hidden state*. They are denoted by the following functions, respectively:

$$c^{(t)} = f \odot c^{t-1} + i \odot s \tag{3.5}$$

$$h^{(t)} = o \odot tanh(c^{(t)}) \tag{3.6}$$

Cell state

The cell state is where the gradient is allowed to flow unchanged. It has similar functionality as a conveyor belt. Information can be kept in this state throughout the recurrent chain, without any losses. Following the conveyor belt analogy, there are four workers that manipulate the items on the belt in an LSTM architecture. These are called *gating functions*.

The gating functions

The four gates of an LSTM, and a short summary of their operations are:

Input gate, i	Determine quantity of new input to store in cell state.
Output gate, o	Determine quantity of current cell state to send to current hidden state.
Forget gate, f	Determine quantity of previous input to remove from cell state.
Squash gate, s	Scaling of new input.

The equations are, respectively:

$$i^{(t)} = \sigma(W_i[h^{(t-1)}, x^{(t)}] + b_i) \tag{3.7}$$

$$o^{(t)} = \sigma(W_o[h^{(t-1)}, x^{(t)}] + b_o) \tag{3.8}$$

$$f^{(t)} = \sigma(W_f[h^{(t-1)}, x^{(t)}] + b_f) \tag{3.9}$$

$$s^{(t)} = tanh(W_s[h^{(t-1)}, x^{(t)}] + b_s) \tag{3.10}$$

In the original LSTM, the authors mention a *input gate*, *output gate* and a *memory cell*. Thus, the update of the cell state was originally as in Equation 3.11 [54, p. 7].

$$c^{(t)} = c^{t-1} + i \odot s \tag{3.11}$$

The original LSTM's hidden state is similar to Equation 3.6. In later years, the name *forget gate* has also been brought into the vocabulary [42, p. 405]. The output from the forget gate is multiplied element-wise with the previous cell state, resulting in the cell state Equation 3.5. This distinction is also used by the deep learning library Keras [22], which is the library used in this paper. The last gate, s, is used for squashing the new input before multiplying it with the input gate. This will be referred to as the *squashing gate, s* (Equation 3.10) in this paper.

The forget, input and output gate use sigmoid nonlinearity, and the squashing gate may have any squashing nonlinearity [42, p. 405]. Although, the most common function to use for all but the squashing gate is the logistic sigmoid function. It is what the Keras library declare as the default for these functions. This has the best results in the original experiments, as it is reported that they need no fine-tuning of the initial bias with this choice

[54, p. 9]. The choice of a logistic sigmoid for gates that control quantity of input is also intuitively a logical choice. In the original paper, the hidden state is updated with a *tanh*, as in Equation 3.6, and the squash gate is updated with a *scaled tanh* with range from $[-2, 2]$ [54, p. 24]. But, in the Keras library, the default functions for both the hidden state and the squash gate is a regular *tanh*, and these settings will be used in the experiments of this paper.

Hidden state

In vanilla RNN, the hidden state is a concatenation of the previous hidden state and the new input, which is then squashed in a *tanh*-layer to produce a new hidden state. But, as seen in Equation 3.6, in an LSTM, the hidden state is an element-wise multiplication between the output gate and a tanh-squashed current cell state.

Bidirectional RNN

Both the vanilla RNN and LSTM learn representations from previous time steps. But, to learn context in a time series, it may sometimes be better to learn representations from future time steps. As, for instance, in "The bass must be salted" or "The bass player was ready", the future representations give context about the past. This is the motivation behind a bidirectional RNN (BRNN) [36]. The BRNN process input in both directions, both forward and backwards. This eventually results in two cell states and two hidden states for each time series; forward h and c, and backwards h and c. This kind of architecture is used for the encoder in this thesis. After processing one sequence, all of the states are added together, before being applied to the next step of the network.

3.2 Mixture Density Networks (MDNs)

The regular sum-of-squares or cross-entropy loss function mentioned in Section 2.1.6 outputs conditional averages of the targets (conditioned on the input data). The outputs often represent probabilities of class membership, with one class being the true class. This output is a single probability distribution. The method works well for single-valued output, but has limited mapping to multi-valued outputs. In the 1994 paper *Mixture Density Networks* by Christopher Bishop, the author provides a solution for multi-valued output situations: "The [...] Mixture Density Network [...] can in principle represent arbitrary conditional probability distributions in the same way that a conventional neural network can represent arbitrary functions" [8].

3.2.1 Mixture Models (MMs)

The MM is a method for optimizing the fit between a statistical model and the observed data. The statistical model can, for instance be a normal

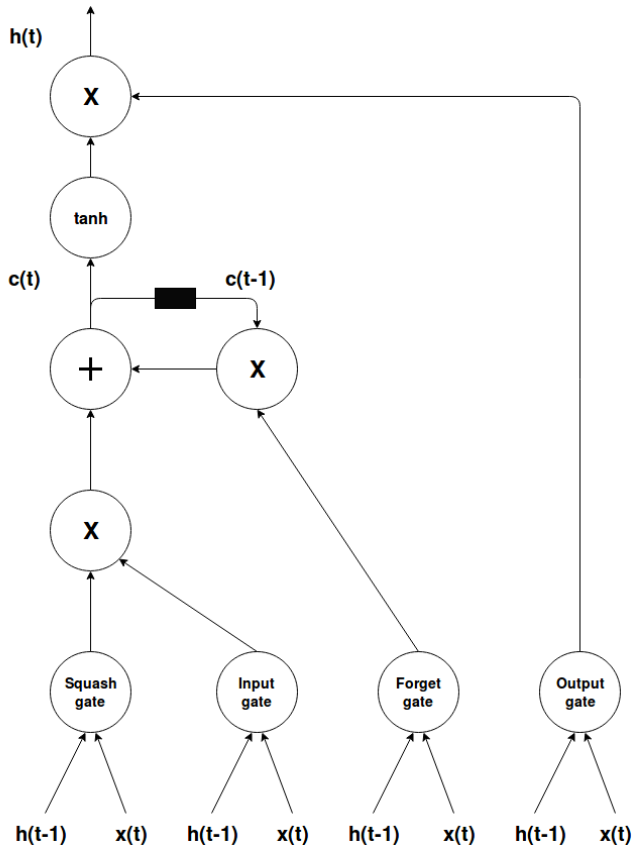


Figure 3.5: Block diagram of an LSTM. The black square indicate a delay of one time step. X-nodes indicate element-wise multiplication, and the +-gate indicate addition. The gates use nonlinear functions, defined by the user.

distribution, in that case the MM will assure normal distributions for all subgroups in the data set. The mixture density *network* combines a mixture density model with a neural network. The outputs from the neural network determine the parameters for the mixture density model. The mixture density model then create the conditional pdf for the target values t, conditioned on the input to the neural net, x [8]. A hyperparameter for the mixture model is its *mixture components*. The mixture components are the number of distributions the MDN *assumes* that the true distribution has. Meaning, that the number of mixture components are the number of distributions the MDN outputs, regardless of the dimensionality of the input.

As the goal for this thesis is to predict sequences, the MDN is a mixture density model combined with a recurrent neural network, a MDRNN. Specifically, a MDRNN layer for Keras [77], created by Martin as part of work in modelling creative sequences [78, 81], was applied to the existing VAE model created for this thesis.

A premise for the VAE is to assume that the data is identical and independently distributed, and that the data points are normally

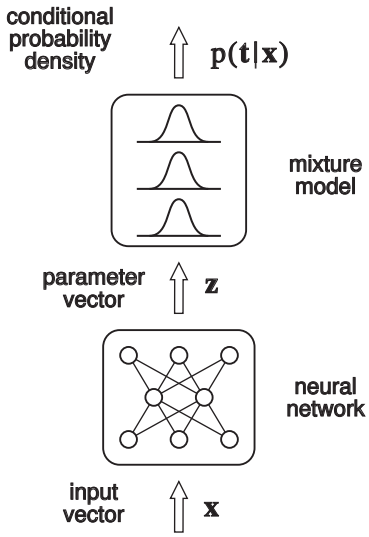


Figure 3.6: The mixture density model [8].

distributed [132]. This means that all input sequences (one sequence is one data point) are assumed to be Gaussian distributions. The encoder will compress the input into Gaussian representations, and the decoder will sample from these distributions to recreate the sequences. The Gaussian representations are called *latent vectors*, denoted with the letter z, see Figure 3.7.

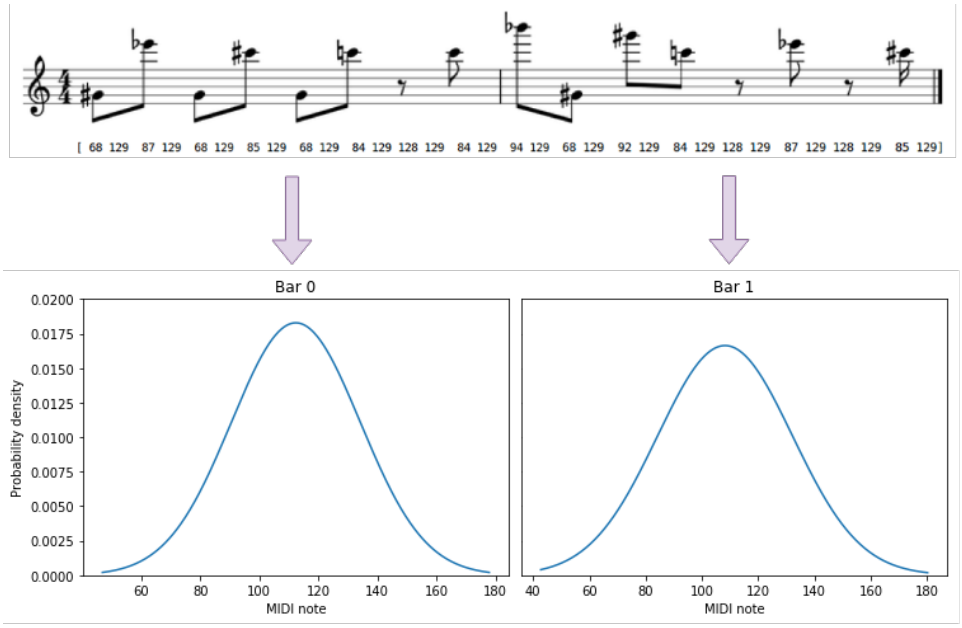


Figure 3.7: Representation of the conversion from bars to distributions, with the underlying note vectors. The image is a simplification of the creation of distributions, as the distributions produced by the VAE-encoder are multivariate with 64 dimensions.

3.2.2 Combining the models

In the context of music generation, an MDN can be used to predict a sequence of real valued-representations of musical events. Martin et al. [81] used an MDRNN to model sequences of musical control instructions, but we seek to generate sequences of musical notes. In the next chapter, we discuss variational autoencoders, which provide a method to encode a bar of musical notes into compressed representations as a real-valued vector, called *z-vectors* . Sequences of these vectors can be predicted by an MDRNN to compose music as follows: a dataset for the MDRNN is created by inferring z-vectors for all songs in the dataset (in this case, the minimum song length was set to $\geq$ 16 bars). The z-vectors are then saved for each song (see Figure 3.8a). After this procedure, the MDRNN is trained to predict the next z_{n+1} given z_n (Figure 3.8b). After finishing this process, the mixture density network can predict songs, bar by bar, only being conditioned on one z_0 in the start of a sequence (see Figure 5.11).

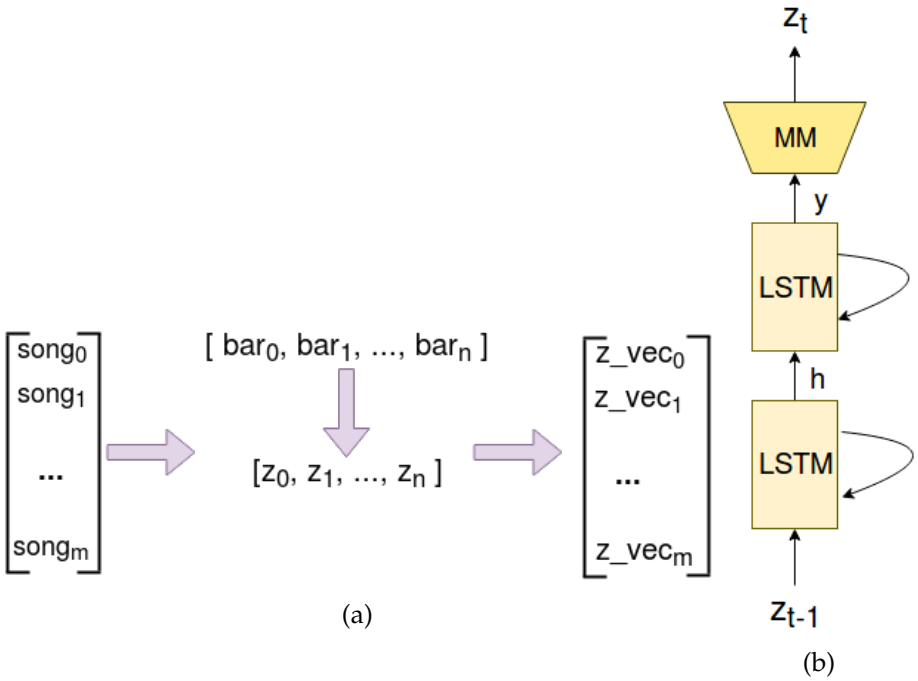


Figure 3.8: Illustration of the creation of a dataset for the MDRNN (figure a), and the mixture density network composer in MCVAE (Figure b).

Chapter 4

Generative modelling

> "Stories", he'd said, his voice low and almost husky, "we are made up of stories. And even the one's that seem the most like lies can be our deepest hidden truths".
>
> *Stan, Briar Rose [138]*

This chapter elaborates on the two state-of-the-art methods for generative modelling that exist today; GANs and VAEs. Explaining the difference between explicit (VAEs) and implicit (GANs) probabilistic models, and how the models can update their knowledge about the true data distributions. GANs main building blocks and its strengths and weaknesses are addressed. Then follows an in-depth section on the VAE concepts and optimization method. Lastly, a few weaknesses in the VAE along with proposed solutions of disentanglement and dropout is introduced.

The goal for a generative model is to learn the true distribution of a data set, to be able to produce new data points with some variations. The generative models parameters should be fewer than the training data points, because this will force the model to discover the essence of the data, in order to generate it. The true distribution of the data may be intractable. Therefore, the approach for a generative model is to model a distribution of data that is as close as possible to the true distribution. In a broader context, generative models may provide a framework for artificial intelligence, "for all the many different intuitive concepts they need to understand, giving them the ability to reason about these concepts in the face of uncertainty" [42, p. 716].

4.1 Inference in probabilistic models

A probabilistic model estimates, on basis of historical data, the probability of an event occurring again [16]. Inference is "a conclusion or opinion that is formed because of known facts or evidence" [87]. In machine

learning, ANNs can perform inference after they have been trained. During inference, the network is fed new input, and draws conclusions about this input based on knowledge attained during training. Inference in a probabilistic model is concerned with two questions [31]:

Marginal inference Infer the probability of a given variable, after summing everything else out.

Max a posteriori inference (MAP). Assign the most likely label to a variable in the model.

Inference can very often be a NP-hard problem, meaning that the time it takes to find a solution grow exponentially with the problem size [130]. The tractability of the inference depends on the structure of the probability graph. But, even if a problem is intractable, approximate inference can get useful answers [31]. There exist many approximate inference methods, examples are Laplace approximation and variational inference [109]. The variational autoencoder (VAE) use variational inference.

In the next sections, the variations of probabilistic models and which methods they use for updating their representations of true data is elaborated, before going into detail on the building blocks of GANs and VAEs.

4.1.1 Directed and undirected probabilistic models

Directed probabilistic models, or directed *graphical* models is a structured probabilistic model, also known as a *belief networks* or *Bayesian networks* [42, p. 560]. The variational autoencoder seeks to solve the inference problem in directed graphical models [132]. The name *directed* indicate that the models edges are directed, and the direction is represented with an arrow. These models are relevant when there is a clear reason to point to a direction. These are often situations where the causality is understood, and it is pointing in only one direction. The arrow indicate which variable's distribution is defined by other variables' distributions. Drawing an arrow from a to b means that b's distribution is conditioned on a. Going from left to right, a variable pointing to another is the *parent* of this variable. And a variable being pointed to is the *child* [56].

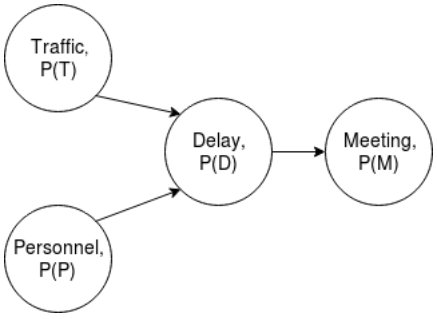


Figure 4.1: Directed graphical model.

The directed graphical model in Figure 4.1, model the likelihood for taking part in a meeting, depending on the delay in public transport. The delay in public transport is again depending on two independent variables; traffic and personnel. The full joint probability is:

$$P(T, P, D, M) = P(T) \cdot P(P) \cdot P(D|T, P) \cdot P(M|D) \tag{4.1}$$

In undirected probabilistic models the direction between events is not clear, or may be going in both directions [42, p. 563]. The difference between a directed and undirected graphical model is important when defining the joint probability that the graph can represent. But, for inference, it's generally common to have a specified fixed probability distribution, which renders the difference between these models less important [128].

4.1.2 Implicit and explicit models

There is a convenient distinction between two types of probabilistic models; prescribed or *explicit models* and *implicit models* [92]. Explicit models contribute an explicit parameterization of the distribution of a random variable x, with parameters θ and log-likelihood function $log\, q_\theta(x)$. The majority of machine learning models are explicit. The alternate implicit models define stochastic procedure that generates data directly. There exist many different methods for generative modelling, but among todays most popular methods are VAEs and GANs [64]. Using the distinction between explicit and implicit models, a VAE falls under the category explicit, and GAN under the category implicit (see Figure 4.2).

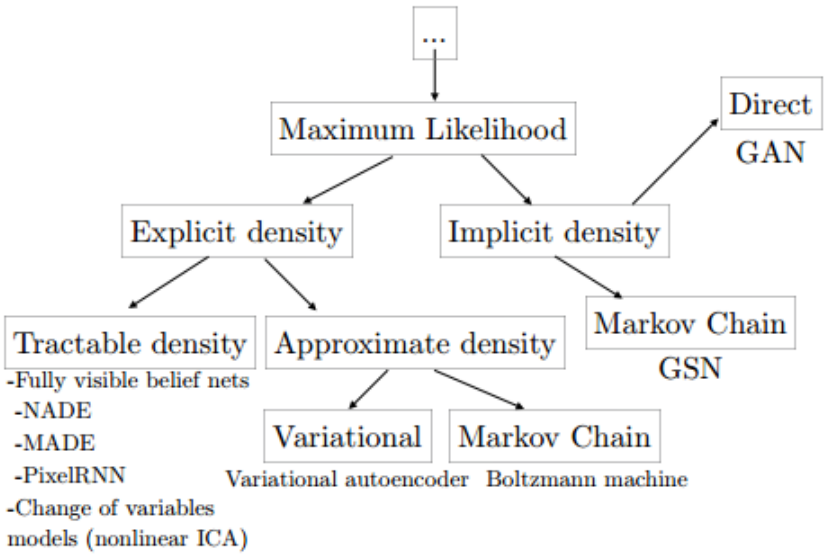


Figure 4.2: Overview of generative models, from Ian Goodfellows NIPS tutorial in 2016 [39].

4.2 Measuring difference between distributions

The VAE makes a prior assumption of the distribution of the true data, draws samples from real data, and then adjust the parameters of the prior distribution [132]. While the GAN creates data from noise, compare this with real data, and then adjust its method for generating new data [41]. Thus, both VAEs and GANs needs methods for quantifying the difference between the generative data and the true data distribution. GANs typically use either *Jensen Shannon-divergence* or *Wasserstein distance*, while VAEs moreoften use *Kullback-Liebler divergence*.

4.2.1 Information entropy

Entropy, in thermodynamics, is defined broadly as "the degree or disorder of uncertainty in a system" [86]. One can consider the level of motion in water molecules, in the three states of water; solid, liquid and gas. When water is in solid form, the water molecules stand still in a rigid structure - therefore this structure has low entropy. When water turns to liquid, the molecules have more positions to move between, giving this structure higher entropy. When water is a gas, the molecules can move almost everywhere - thus this state has high entropy.

The mathematician Claude Shannon introduced the term "entropy of an information source" in 1948 [117]. It is a definition of the entropy of a random variable. This is the measure of the uncertainty associated with the random variable. Beneath is the entropy function for multiple classes:

$$H(X) = -\sum_{i=1}^{n} p_i \cdot log_b(p_i) \tag{4.2}$$

Shannon also defined the entropy for a joint distribution *H(X,Y)*.

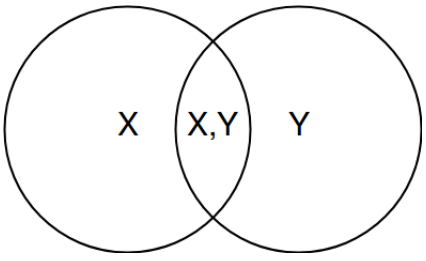


Figure 4.3: Joint distribution of X and Y.

$$H(X,Y) = -\sum_{n}^{i=1}\sum_{m}^{j=1} P(x_i, y_j) \cdot log_b(P(x_i, y_j)) \tag{4.3}$$

Equations 4.2 and 4.3 yield common methods for measuring the distance between two distributions, namely the Kullback-Liebler divergence

and the Jensen-Shannon divergence. These distance measurements are used as loss functions in generative models.

4.2.2 Kullback-Liebler divergence (KL)

Kullback-Liebler divergence is also known as *cross entropy* [119].

$$KL(P||Q) = \sum_{i=1}^{n} P(x) \cdot log\frac{P(x)}{Q(x)} = H(P,Q) - H(P) \tag{4.4}$$

$H(P,Q)$ is the joint entropy of the probability distributions and $H(P)$ is the entropy of probability distribution P. The Kullback-Liebler divergence is a non-symmetric measure of the difference between two distributions.

Forward and reverse KL divergence

The description *non-symmetric* in the KL definition mean that the order of the operators matter. Meaning that $KL(P||Q) \neq KL(Q||P)$. To optimize and reduce the difference between two distributions, one would assume a true distribution *P(X)*, and optimize an approximate distribution *Q(X)*. In this case, there are two ways to optimize. One is from the true distribution to the approximate - *forward*. The second is from the approximate distribution to the true - *reverse* [9].

Forward KL divergence.

In the exemplified case, Equation 4.5 define the forward situation.

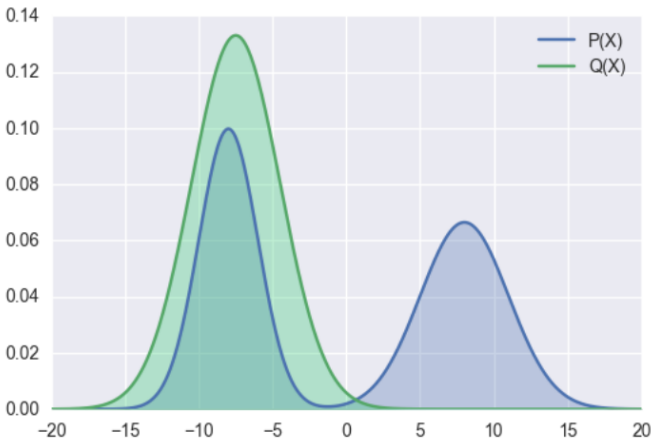


Figure 4.4: Approximating a distribution [67].

$$KL(P||Q) = \sum_{i=1}^{n} P(x) \cdot log\frac{P(x)}{Q(x)} = H(P,Q) - H(P) \tag{4.5}$$

In this case, $P(x)$ is the weight of the divergence. Meaning that when $P(x) = 0$, the divergence is zero. This way, when optimizing the difference between two distributions as in Equation 4.4, the error for the divergence will be the smallest around $P(x) = 0$. Conversely, when $P(x) > 0$, the $log\frac{P(x)}{Q(x)}$ plays an important role in the divergence. This method leads to $Q(x)$ averaging over the entire space of $P(x)$. Forward KL divergence is also known as *zero avoiding*, because it avoids $Q(x) = 0$ when $P(x) > 0$ [67]. This leads to the approximation seen in Figure 4.5.

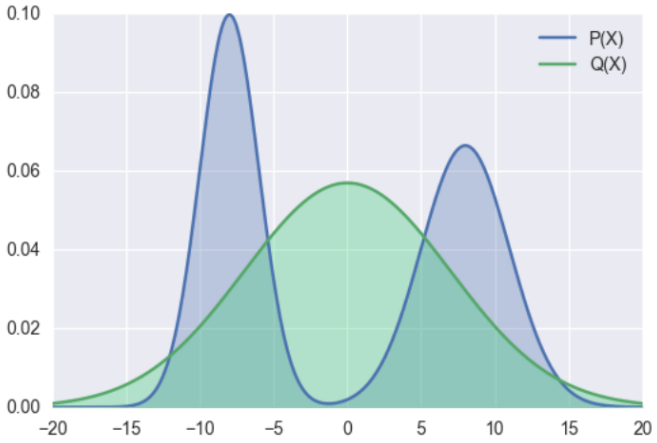


Figure 4.5: Forward approximation [67].

Reverse KL divergence.

In reverse KL divergence, the order in the KL function is reversed.

$$KL(Q||P) = \sum_{i=1}^{n} Q(x) \cdot log\frac{Q(x)}{P(x)} = H(Q,P) - H(Q) \tag{4.6}$$

This yields the opposite result. Here, $Q(x)$ is the weight of the divergence. Parts of $P(x)$ can be ignored, when $Q(x) = 0$. When $Q(x) > 0$, the difference between the two distributions must be as low as possible. In this case, the end result of the optimization would be as in Figure 4.4. Although parts of $P(x)$ is overlooked, $Q(x)$ fit parts of the true distribution with higher precision, than by averaging the entire true distribution. This method is called *zero forcing*, as it forces $Q(x)$ to be zero on some parts of $P(x)$ [67].

4.2.3 Jensen-Shannon divergence (JS)

The Jensen-Shannon divergence is a symmetrized and smoothed version of the KL-divergence.

Where $R = 0.5 \cdot P + Q$ denotes the midpoint between P and Q. This method was originally used for optimizing Generative Adversarial

$$JS(P||Q) = 0.5 \cdot (KL(P||R) + KL(Q||R)) \tag{4.7}$$

Networks (see Section 4.3), but a recent research in GAN-optimization suggest that the *Wasserstein distance* is a better method [6] for this network.

4.2.4 Wasserstein distance (W)

The Wasserstein distance is a measure of the distance between two probability distributions. It is also known as Earth-Mover (EM) distance, because the distributions can be interpreted as piles of dirt, and EM measures how many shovelfuls of dirt that needs to be moved [133].

Consider the example below of two, discrete distributions, P and Q.

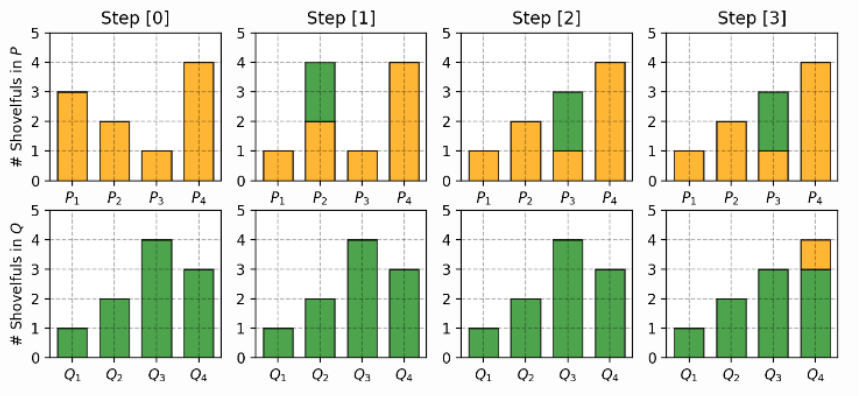


Figure 4.6: Moving dirt between piles to make them match [133].

Moving from left to right, each pile is compared and aligned. For (P_1, Q_1), two blocks are moved from $P_1 \rightarrow P_2$. For (P_2, Q_2), two blocks are moved from $P_2 \rightarrow P_3$. For (P_3, Q_3), one block is moved from $Q_3 \rightarrow Q_4$. In 2017, Martin Arjovsky et. al suggested that the Wasserstein distance is a better method for optimizing Generative Adversarial Networks than the original Jensen-Shannon divergence [6]. See Section 4.3.6.

4.3 Generative Adversarial Networks (GANs)

A GAN is an implicit probabilistic model, developed by Ian J. Goodfellow et.al, based on game theory. The method consist of two players, and the goal is for the players to reach a Nash equilibrium. Nash equilibrium is where each player in a game has a set of strategies, and no incentive to change their strategy, given what the opposite player is doing. This means that the players have no benefit in changing their strategy, and thus don't [101].

4.3.1 The generative and discriminative models

In a GAN, two models are trained simultaneously; a generative model, *G*, and a discriminative model, *D*. For example, the goal is to create music with this network. The generative model is seeded with random noise, and outputs examples of songs. The discriminative model is fed both the samples from model *G*, and samples from a dataset of real songs, and its purpose is to classify whether the sample it is evaluating is real or fake.

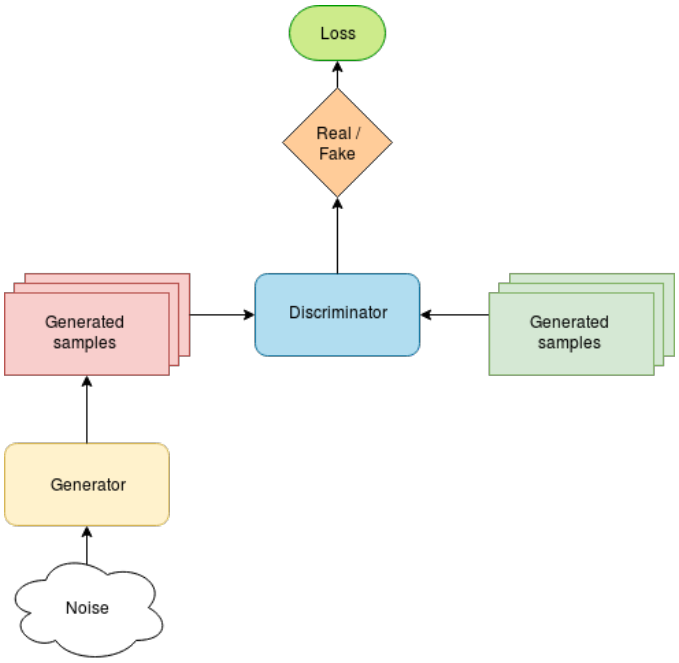


Figure 4.7: Flowchart of a GAN.

Both models are trained with backpropagation. During backpropagation, *G's* incentive is to maximize the probability that *D* classifies incorrectly. And *D's* motivation is the opposite. The authors of the original GAN-paper explain the procedure like this: "The generative model can be thought of as analogous to a team of counterfeiters, trying to produce fake currency and use it without detection, while the discriminative model is analogous to the police, trying to detect the counterfeit currency." [41].

4.3.2 Latent variable interpolation

In general, probabilistic models are modelled as a collection of observable (x), and hidden, or latent (z) variables [59]. Both GANs and VAEs use latent variables to downsample the true data distribution. For a VAE, input data is transformed into latent variables, and then mapped from these latent variables back to the true data distribution. The GAN use latent variables to map from noise into true data. With the GAN, mapping is done with the generator. While in the VAE, mapping is done with a decoder (see Section 4.4.4).

The model DCGAN [103] from 2016, by Alan Radford and Luke Metz, show one example on latent variable interpolation on bedroom images (see Figure 4.8). The example show that with interpolating between different representations of bedrooms, all transitions appear plausible.

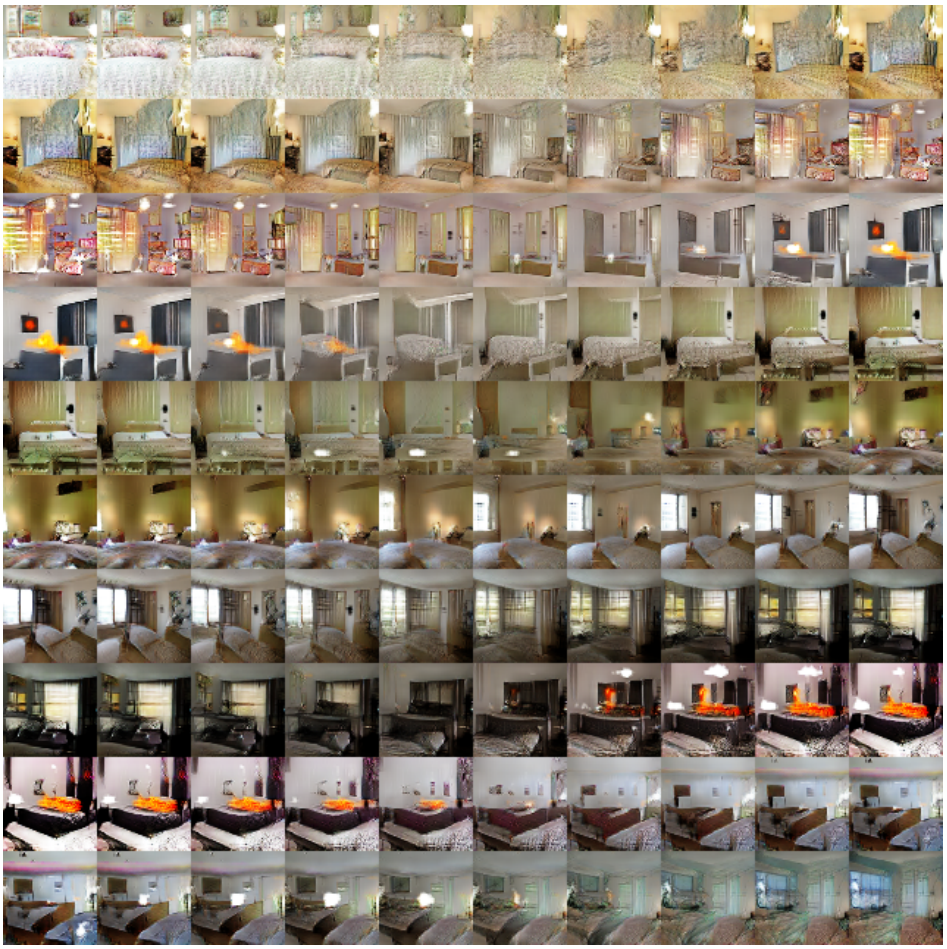


Figure 4.8: Bedroom interpolation in DCGAN [103].

The same paper also show that feature extraction is possible, by performing vector arithmetic on features (see Figure 4.8). The example show that the model harnesses features like "glasses", "woman" and "man", by being able to subtract man from an image of a man with glasses, and then adding a woman instead.

4.3.3 MidiNet

In 2016, a team from DeepMind at Google presented an autoregressive method for music generation in raw audio domain. The model was *Net*, a speech generator conditioned on text [97]. The network used convolutional neural networks, showing that CNNs can be used on temporal data. The team also trained the WaveNet to create music, stating "Since WaveNets can be used to model any audio signal, we thought it would also be fun

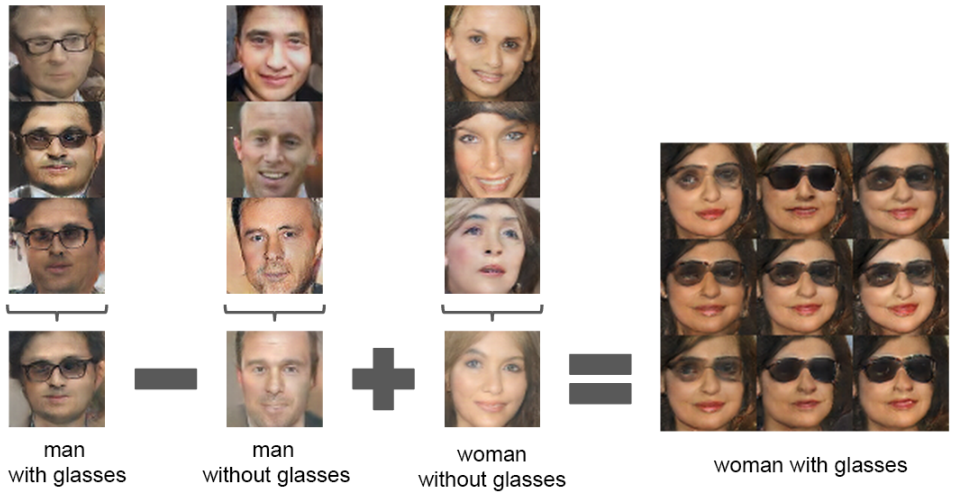


Figure 4.9: Feature extraction in DCGAN [103].

to try to generate music" [96]. The network was trained on raw audio, and created melodies in raw audio domain. This method inspired another team, at Research Center for IT innovation in Taiwan, to try GANs for music creation in the symbolic domain. They released the model MidiNet in 2017, approximately a year after WaveNet [136]. The model is trained to produce music, bar by bar, in the symbolic domain. MidiNet was compared with three generative RNN models from Googles Magenta project *MelodyRNN* [127]: 1. Baseline model, 2. Lookback model, 3. Attention model. 21 subjects with musical background was asked to listen to three sets of music. They were asked to evaluate how pleasing, interesting and realistic the music was, on a Likert-type scale. The results showed that MidiNet produced equally pleasing and realistic music as the comparative RNN models, but the music did appear to be more interesting.

4.3.4 Mode collapse

Despite GANs impressive results in both image and music creation, the model have two dominant disadvantages, that still is a subject for research. These are *mode collapse* and *unstable training*. Mode collapse is a situation that may arise during training. That is when the generator produce similar outputs with very small diversity. It still manages to fool the discriminator, but it fails to learn to represent the whole data distribution.

4.3.5 Unstable training

Goodfellow has commented on the unstable traning of GANs. Gradient descent does not guarantee finding the Nash equilibrium. "Sometimes gradient descent does this, sometimes it doesn't", he states [40]. Meaning that training a GAN is relatively hard, as there is no guarantees that the Nash equilibrium is found. Salimans et.al. have discussed this further in *Improving Techniques for training GANs*.

They exemplify the problem of finding Nash equilibrium in a non-cooperative game.

> For example, when one player minimizes xy with respect to x and another player minimizes -xy with respect to y, gradient descent enters a stable orbit, rather than converging to $x = y = 0$ [110].

Illustrated, this gives:

$$f_0(x) = xy$$
$$f_1(x) = -xy$$

Where $f_0(x)$ change x to minimize xy, and $f_1(x)$ change y to minimize -xy. Their derivatives are:

$$\frac{\delta f_0(x)}{\delta x} = y$$
$$\frac{\delta f_1(x)}{\delta y} = -x$$

$f_0(x)$ is updated with $x - \eta \cdot y$ and $f_1(x)$ is updated with $y - \eta \cdot x$ in each iteration, where η is the learning rate. When x and y have different signs, the gradient updates cause oscillations that increase over time [133].

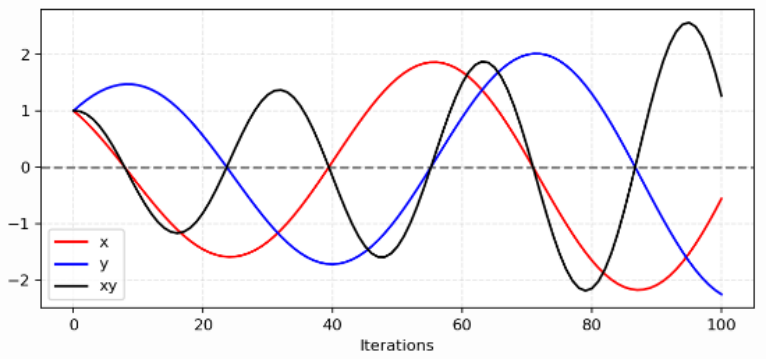


Figure 4.10: Oscillating gradients in GAN [133].

4.3.6 Wasserstein distance to avoid mode collapse

Martin Arjovsky et.al. investigate the benefits of using Wasserstein distance instead of Jensen-Shannon divergence in the paper *Wasserstein GAN* [6]. They explore a setting where both probability distributions P and Q are disjoint.

$$\forall(x, y) \in P, x = 0 \text{ and } y \sim \mathcal{U}(0, 1) \tag{4.8}$$

In this case, there is no overlap between the distributions when $\theta \neq 0$ (see Figure 4.11). In that case, Kullback-Liebler goes towards infinity and

$$\forall(x, y) \in Q, x = \theta, 0 \leq \theta \leq 1 \text{ and } y \sim \mathcal{U}(0, 1) \tag{4.9}$$

Jensen-Shannon is not differentiable in $\theta = 0$. But, the Wasserstein function provides a smooth computation, which is necessary for a stable update of the gradients [133].

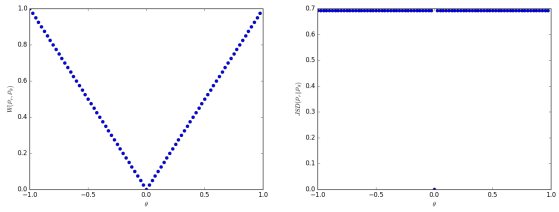


Figure 4.11: W-distance (left) and JS-divergence (right) for $\rho(\mathbb{P}_\theta, \mathbb{P}_0)$ as a function of θ. W-distance is continuous, while JS has a sudden jump in $\theta = 0$.

In their results, Wasserstein seems to give a more stable optimization. They also state that in "no experiment did we see evidence of mode collapse for the WGAN algorithm."

4.4 Autoencoders

An autoencoder is a neural network whose goal is to recreate its input, while learning useful information about the input dataset [42, p. 499]. It is sometimes referred to as a unsupervised learning algorithm, as the input does not come with an expected target value. But, since the target value is the same as the input, the more precise label for this network is a *self-supervised* algorithm [21].

An autoencoder consists of three parts: an encoder, a latent code and a decoder. Depending on the motivation behind the autoencoder, the latent code may have smaller or larger dimensionality than the input and output layers [42, p.500-501].

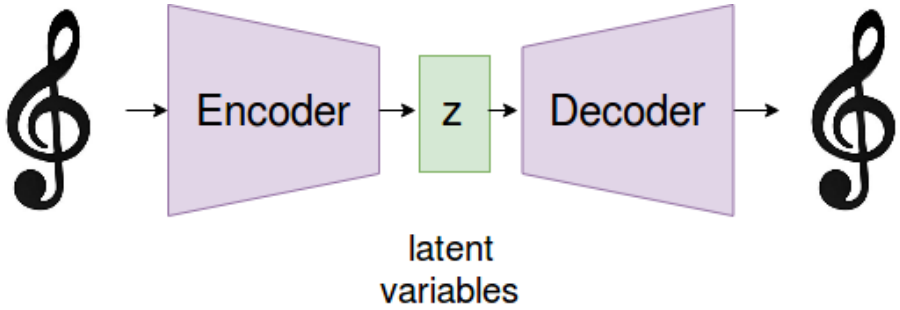


Figure 4.12: General representation of autoencoder.

In general, the encoder alters the dimensionality of the input, d_i, to a fixed size vector, or latent space representation with dimension d_l. Then,

the decoder reconstructs the input based on this latent space. The only requirement for an autoencoder is that the output dimension is the same as the input dimension [27].

Latent space dimensionality

In an *undercomplete autoencoder*, $d_l < d_i$, the latent space captures the most noticeable features of the training data. While in a *regularizing autoencoder*, $d_l > d_i$, the network is encouraged to learn other properties than the ones required to copy input to output. These properties are for instance the sparsity of the representation, smallness of the derivatives, robustness to noise and missing inputs [42, p. 501].

4.4.1 Variational autoencoders

While an autoencoder maps the input to a fixed vector, a *variational autencoder* maps the input to a distribution. The idea of variational autoencoders was first published by Diedrik P. Kingma and Max Welling in 2014, with the title *Auto-Encoding Variational Bayes* [132]. The authors ask the question:

> How can we perform efficient inference and learning in directed probabilistic models, in the presence of continuous latent variables with intractable posterior distributions, and large datasets?

Prior and posterior probability

A prior probability is a subjective estimate, before any experiments has been made. The posterior probability is the probability of an event happening, after all the evidence is taken into account [74, p. 101]. When modelling real-world data, the posterior distribution may be *too complex* to model. Therefore the posterior distribution is *intractable*. The question asked by Kingma and Welling is then; how can we learn and draw conclusions about a system that is too complex to model precisely?

4.4.2 Inference as an optimization method

The method used for variational autoencoders is known as *Mean-Field Approximation* or *Variational lower bound* [59]. This method rewrites a statistical inference problem to an optimization problem. The idea is to perform inference on a set of known distributions, Q, all with a fixed set of parameters, (μ, σ^2), to find the $q \in Q$ that is most similar to the true distribution p [69]. Then, it is possible to get approximate solutions by querying q instead of p. Adjusting the parameters of q is done by sampling from the true distribution p, and updating the parameters (μ, σ^2) of q, in a method called *Bayesian inference*. Bayesian inference is the method of updating a hypothesis as more evidence come to light [11]. The parameters are updated applying reverse Kullback Liebler divergence (see

Section 4.2.2) [67]. The many unknown distributions are all assumed to be Gaussian for real-valued data, both $p_\theta(z)$, $p_\theta(x|z)$, $p_\theta(z|x)$ and $q_\phi(z|x)$ [59]. Note that since $p_\theta(x)$ is intractable, the posterior distribution $p_\theta(z|x)$ is also intractable, and therefore $q_\phi(z|x)$ is used as an approximation instead [72, p. 68].

4.4.3 Latent variables

In an encoder/decoder-network, the hidden variables are the latent ones, and the observable are the input and output. The latent variables hold information about the different features for the output. Table 4.1 display different features for describing music. In addition to the notes themselves, music holds information that may be difficult to describe in a discrete manner.

Feature	**Related term**
Rhythm	beat, meter, tempo, syncopation, polyrythm
Dynamics	crescendo, descendo, forte, piano, etc
Melody	pitch, range, theme
Harmony	chord, progression, key, tonality, consonance, dissonance
Tone color	register, range
Texture	monophonic, polyphonic, homophonic
Form	binary, ternary, strophic, etc

Table 4.1: Musical features [58] that could be learned by a VAE. We might expect that the latent space would represent some of these features.

4.4.4 VAE optimization

The method for optimizing in a VAE is visualized as a two part system (see Figure 4.13), like the autoencoder.

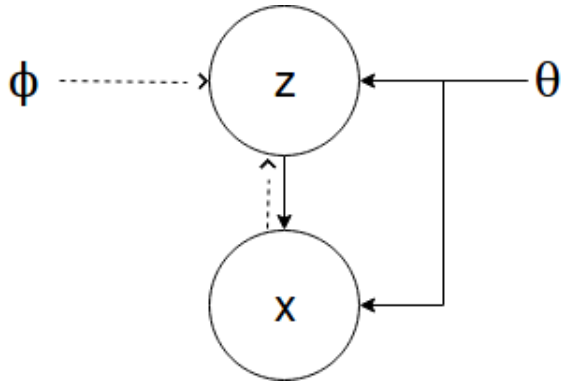


Figure 4.13: Graphical model of variational autoencoder. The dashed lines indicate the variational approximation, and the solid lines indicate the generative process.

Explained in autoencoder terms, the two processes are:

Encoder - variational approximation $q_\phi(z|x)$ to true posterior, $p_\theta(z|x)$ [72, p. 71]. Parameters are called variational parameters, with notation ϕ.

Decoder - generative model, $p_\theta(x|z)$ [72, p. 71]. Generates outputs, x, from latent vectors z. Parameters are called generative parameters, with notation θ.

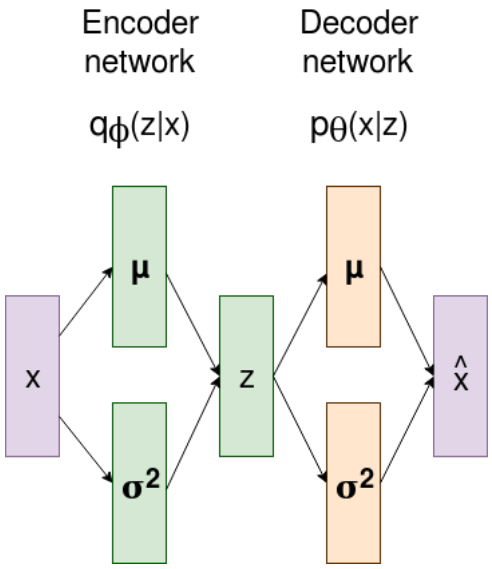


Figure 4.14: Visualization of the encoder and decoder network in a variational autoencoder.

4.4.5 The variational lower bound

The variational autoencoder seek to maximize the log likelihood of data point x, $log\ p_\theta(x)$. In Equation 4.10, Bayes' rule is used to rewrite $p_\theta(x)$. This is used to represent the data likelihood in Equation 4.11 [72, p. 82].

$$\begin{aligned} p_\theta(x|z) &= \frac{p_\theta(x) \cdot p_\theta(z|x)}{p_\theta(z)} \\ p_\theta(x) &= \frac{p_\theta(x|z) \cdot p_\theta(z)}{p_\theta(z|x)} \end{aligned} \tag{4.10}$$

The two first terms on the right hand side is what's known as the variational lower bound, or evidence lower bound, *ELBO* [59]. Rewriting the function, for clarity:

The log likelihood of data point x is thus described by two parts:

$$
\begin{aligned}
log\ p_\theta(x) = &\ \mathbf{E}_{z\sim q_\phi(z|x)}[log\ p_\theta(x^{(i)})] \\
= &\ \mathbf{E}_z[\frac{log\ p_\theta(x|z)\cdot p_\theta(z)}{p_\theta(z|x)}] \\
= &\ \mathbf{E}_z[\frac{log\ p_\theta(x|z)\cdot p_\theta(z)}{p_\theta(z|x)}\cdot\frac{q_\phi(z|x)}{q_\phi(z|x)}] \\
= &\ \mathbf{E}_z[log\ p_\theta(x|z)] - \mathbf{E}_z[log\frac{q_\phi(z|x)}{p_\theta(z)}] + \mathbf{E}_z[log\ \frac{q_\phi(z|x^{(i)})}{p_\theta(z|x)}] \\
= &\ \mathbf{E}_z[log\ p_\theta(x|z)] - D_{KL}(q_\phi(z|x)||p_\theta(z)) \\
&\ + D_{KL}(q_\phi(z|x)||p_\theta(z|x))
\end{aligned}
\tag{4.11}
$$

$$
log\ p_\theta(x) = \mathcal{L} + D_{KL}(q_\phi(z|x)||p_\theta(z|x)) \tag{4.12}
$$

Part 1: $\mathbf{E}_z[log\ p_\theta(x|z)] - D_{KL}(q_\phi(z|x)||p_\theta(z))$:
the variational lower bound, $\mathcal{L}$. The expected value of the output from the generative model is the reconstruction accuracy. This, minus the distance between the approximated conditional z and the prior z, is the lower bound. Both these distributions are Gaussians, so this KL divergence has a closed form-solution [72, p. 79].

Part 2: $+D_{KL}(q_\phi(z|x)||p_\theta(z|x))$:
The distance between the approximated conditional z-distribution and the true conditional z-distribution. The approximated $(z|x)$ is a Gaussian, while the true distribution is intractable. Therefore, this term is not computable.

As KL divergence always is ≥ 0, this leads to $\mathcal{L} \leq log\ p_\theta(x)$. Because of this, $\mathcal{L}$ is named the *lower bound* on the solution. The variational lower bound is the limit to how close the prior distribution can get to the posterior distribution. As the difference between the lower bound and the data log likelihood is the KL divergence between the true and the approximated conditional z-distribution, this means that $\mathcal{L} = log\ p_\theta(x)$, if (and only if) the distributions are equal [137].

4.4.6 The reparameterization trick

As the last KL term in 4.12 is intractable and not computable, the goal is to *maximize* $\mathcal{L}$ [72, p. 79]. By maximizing the lower bound, the likelihood of the data points increase. The KL divergence in the lower bound can be integrated analytically, so that only the reconstruction error has to be estimated by sampling [132]. But, the prior z is stochastic (sampled from a Gaussian), meaning that it is changing with every iteration, regardless of the input. Using a fixed input, presented to the network through many iterations would still give various outputs. It is as sending a clear

signal through a noisy channel. A fully stochastic node like this is not differentiable.

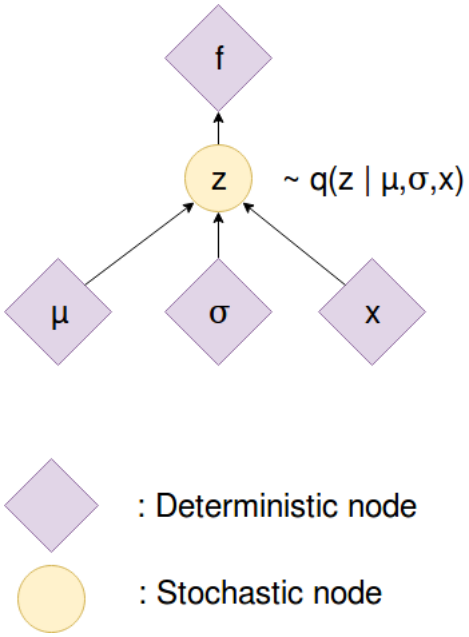


Figure 4.15: Graphical model of stochastic node.

Reparameterization is the method of rewriting statistical problems. It means to define a *new* function that describes the *original* function with new parameters [134]. A subset of reparameterizing tricks is to substitute a variate by a deterministic transformation of a simpler variate [91]. These transformations involve combining the parameters of the desired distribution with a *base distribution*, $p(\epsilon)$, where ϵ is easy to sample from the distribution. Combining the parameters, and this ϵ, gives the desired distribution, $p(z)$. Specifically, for a Gaussian distribution [91]:

p(z; θ)	**Base p(ϵ)**	**Rewritten as**
$\mathcal{N}(\mu; \sigma^2)$	$\epsilon \sim \mathcal{N}(0; 1)$	$\mu + \sigma \odot \epsilon$

Table 4.2: Reparameterizing a Gaussian distribution

The lower bound function for datapoint $x^{(i)}$ is as seen in Equation 4.13 [132]:

$$\mathcal{L}(\theta, \phi; x^{(i)}) \simeq \frac{1}{2} \sum_{j=1}^{J} (1 + log((\mu_j^{(i)})^2) - (\sigma_j^{(i)})^2 - (\mu_j^{(i)})^2) + \frac{1}{L} \sum_{l=1}^{L} log\ p_\theta(x^{(i)}|z^{(i,l)}) \tag{4.13}$$

where $z^{(i,l)} = \mu^{(i)} + \sigma^{(i)} \odot \epsilon^{(l)}$ and $\epsilon^{(l)} \sim \mathcal{N}(0, \mathbf{I})$.

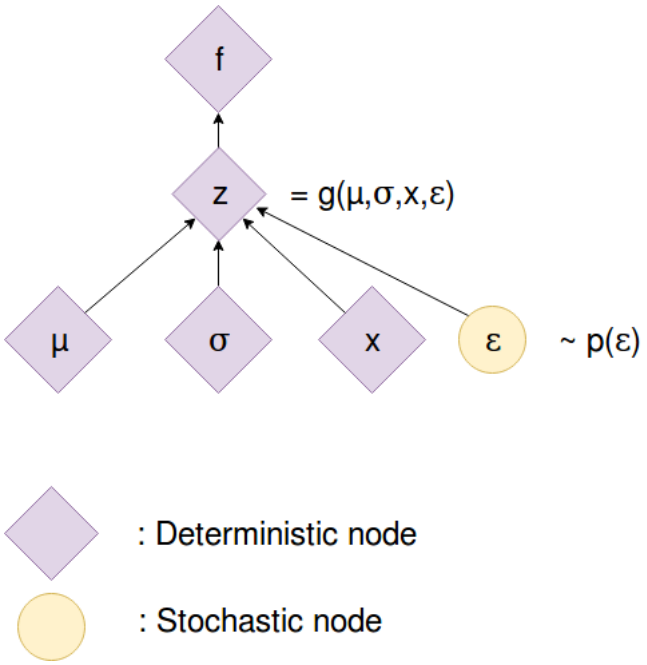


Figure 4.16: Graphical model of deterministic node.

The first RHS term is the analytic integration of the KL divergence, and the second term is the reparameterized sample of expected output. The symbol "$\simeq$" describe that the LHS and RHS are *homotopically equivalent*. This means that one mathematical object can be continuously transformed into the other, and is an implication of the reparameterization [129].

4.4.7 Disentangled variational autoencoder

Recent research [15] has been looking into the information bottleneck of the variational autoencoder, and ways to optimize it. When optimizing the lower bound, KL-divergence is zero when $q_\phi(z|x) = p_\theta(z)$, meaning that $\mu_i = 0$ and $\sigma_i = 1$, $\forall i$. This means that the latent channels are forced to encode all data points with $\mu = 0$ and $\sigma = 1$. The way to increase the capacity of the latent channels is by scattering the means for the datapoints, or decreasing the standard deviations, but this leads to a higher KL-divergence. This bottleneck enforce an embedding in which adjacency in latent space implies adjacency in data space. On the other hand, this may lead to overlapping encoded distributions that will give a higher cost on the log-likelihood, due to reduced discriminability.

The mentioned study propose that a the KL-term alone puts too little pressure on enforcing this adjacency constraint. With a high KL-term, they postulate that only information about the data points that give significant improvement in log-likelihood is encoded.

$$\gamma \cdot |D_{KL}(q_\phi(z|x)||p_\theta(z) - C| \tag{4.14}$$

The authors suggest a flexible KL-term (see equation 4.14), annealing its value from a large value to a smaller one during optimization. The C is a selected KL-divergence value, while γ is the weight of the KL-penalty. This method force the latent space to first encode the most important features, and then slowly adding features that improve reconstruction accuracy. The results from their experiments show that this method encode more information, and produce a smoother latent dimension.

4.4.8 Posterior collapse

In RNN autoencoders, the decoder model may have so many neurons that it learns to memorize the entire training set. This means that it can reproduce sequences, independent of the latent vector, and thus ignore the latent vector. This is what's known as *the posterior collapse* [104]. A proposed solution is to introduce a lossy representation of the data [19]. For LSTMs, dropout on the internal gates, and not the cell state, has seemingly the best results [20].

4.4.9 Summary

This chapter gives an overview of today's state-of-the-art models for generative modelling, namely GANs and VAEs. The core strengths and weaknesses for both models are mentioned. In the remainder of the thesis, the focus will be on the VAE method applied to the problem of well-structured musical composition.

Chapter 5

Methods

> We cannot predict the future,
> but we can invent it.
>
> *Dennis Gabor [24, p. 210]*

In this chapter, the three main inspirations for the thesis' novel architecture are explained. Disclosed chronologically, two VAE models from Google Brain are first discussed, then the World Model. Lastly, the novel architecture *Mixture Composer VAE* or *MCVAE* is explained, visualized, and its choice of hyperparameters justified.

The solution used in this thesis builds on three existing machine learning frameworks; *SketchRNN* and *MusicVAE* from the Google Brain Team, and David Ha and Jürgen Schmidhuber's *World Model*. In 2017 and 2018, the Google Brain Team produced SketchRNN and MusicVAE, two VAE-architectures for generative sequence modeling that combined outlines the VAE architecture applied in this thesis. In MusicVAE, the authors seek to improve the formal VAE's ability to predict long term sequences, by adding a hierarchical "conductor" RNN between the latent layer and the decoder (see Figure 5.3).

In this thesis, a novel solution to the hierarchical composer network is proposed (see Figure 5.11), by exchanging the RNN composer with a MDRNN. This idea stems from Ha and Schmidhuber's World Model (Figure 5.6), in which a reinforcement learning system use a MDRNN to predict latent vector representations of the "real world" environment the agent operates in. The novel solution in this thesis is named *MCVAE*.

5.1 SketchRNN

SketchRNN originates from the paper *A Neural Representation of Sketch Drawings* by David Ha and Douglas Eck (2017) [46]. SketchRNN is constructed as a one layer bidirectional RNN-encoder (see Figure 5.1), and a one layer unidirectional RNN-decoder (Figure 5.2). The layers are of size 512 and 2048, respectively, and the latent layer is size 128. They

use a Gaussian Mixture Model on the decoder output, that takes the softmax output, y, and a temperature parameter, $\mathcal{T}$ that affects the softmax sampling, when creating the next data point. The sampled z from the encoder, and a *"start of sequence"*-token is concatenated as input to the decoder. The sampled z is also squashed through a *tanh*-gate to initialize the hidden states and cell states in the decoder.

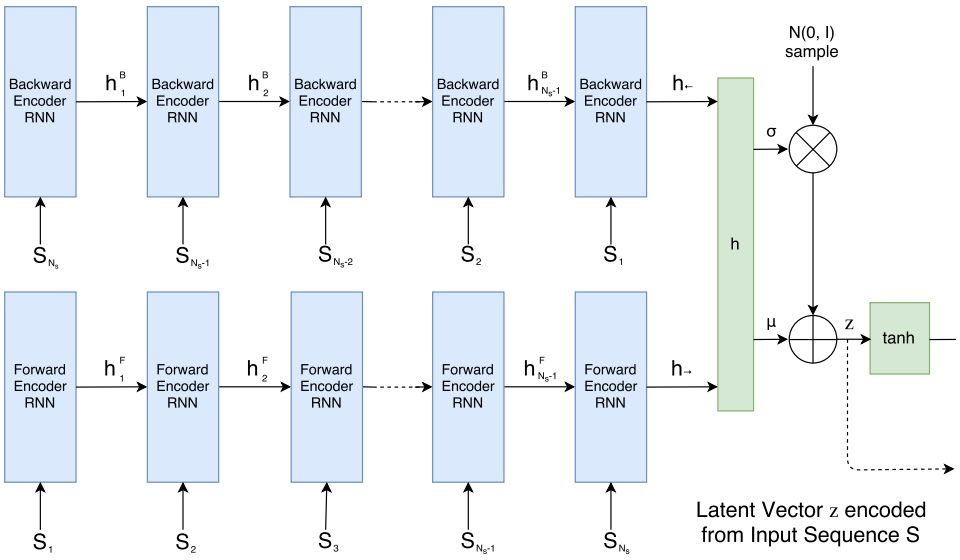


Figure 5.1: SketchRNN encoder architecture, reproduced from [46]

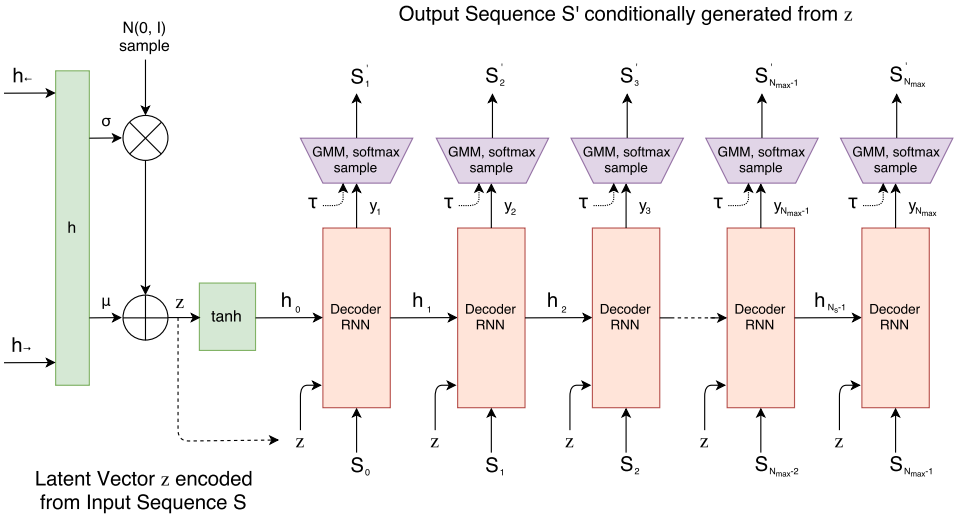


Figure 5.2: SketchRNN decoder architecture, reproduced from [46]

5.2 MusicVAE

MusicVAE stems from the paper *A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music* by Adam Roberts, Jesse Engel, et.al (2018) [104].

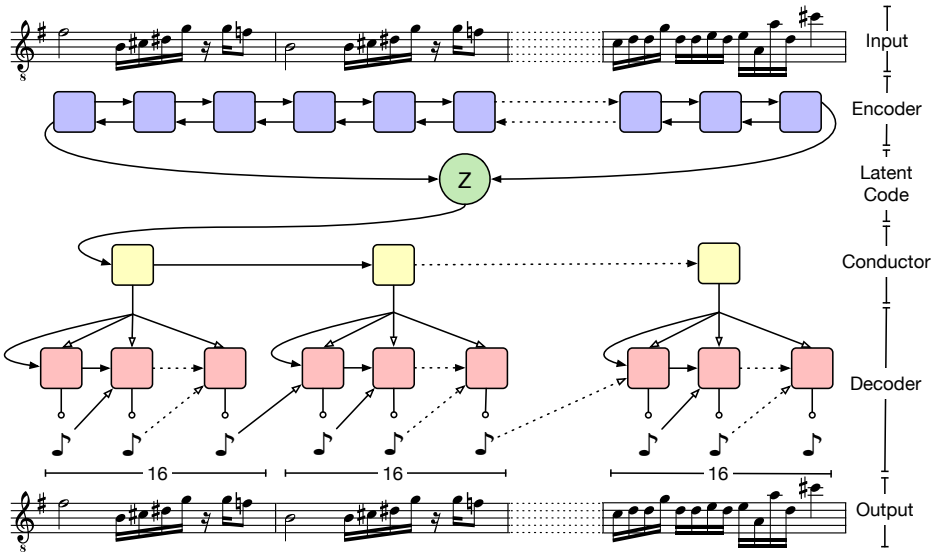


Figure 5.3: MusicVAE architecture, reproduced from [104]

MusicVAE use a two-layer bidirectional encoder, as well as a two-layer unidirectional decoder. Additionally, they have implemented a two-layer unidirectional *Conductor RNN*, that produce embedding vectors for each 16 bar sub sequence in the output. The encoder and decoder all have 2048 nodes per LSTM, the conductor has 1024, and the latent space dimensionality is 512.

The conductor is created, seeking to avoid the *posterior collapse* problem that may occur in RNN autoencoders. They use the conductor to "limit the scope of the decoder to force it to use the latent code to model long-term structure". The decoder receives a new embedding vector for each 16 note sub sequence, or bar. The last token from each sequence is used as input for the next sequence, concatenated with the corresponding embedding vector.

5.2.1 Disentanglement

The MusicVAE is trained using annealing of the KL-cost, as mentioned in Section 4.4.7, but with in a simplified fashion (see equation 5.1), suggested by Bowman et al. (2016) [102].

$$\beta \cdot D_{KL}(q_\phi(z|x)||p_\theta(z)) \tag{5.1}$$

By multiplying the KL-term with a β-variable, and annealing this from zero, the model is forced to encode as much information as possible into

the latent layer. The MusicVAE only anneal β from 0-0.2, meaning that this model mostly emphasize reconstruction accuracy.

5.2.2 Flat model vs. hierarchical model

In the MusicVAE-paper, the hierarchical model is compared to a "flat" model to endorse the need for the hierarchy. All melodies in the data set have $\frac{4}{4}$ time signature, resulting in 16 sixteenth-notes per bar. The models were compared on 16-bar melodies, meaning 256 notes. The flat model output the whole melody without any hierarchical dependencies. While the MusicVAE output the melody with a hierarchical dependency for every 2 bars (32 notes). The models were evaluated quantitatively (see Figure 5.4) on latent space interpolation between real data points, using a language model probability evaluation, and secondly by computing hamming distance. In these evaluations, the hierarchical model produced the most probable melodies when interpolating. The Hamming distance varies monotonically for both models, but the hierarchical decoder has a lower intercept, this is explained as a result of better reconstruction accuracy.

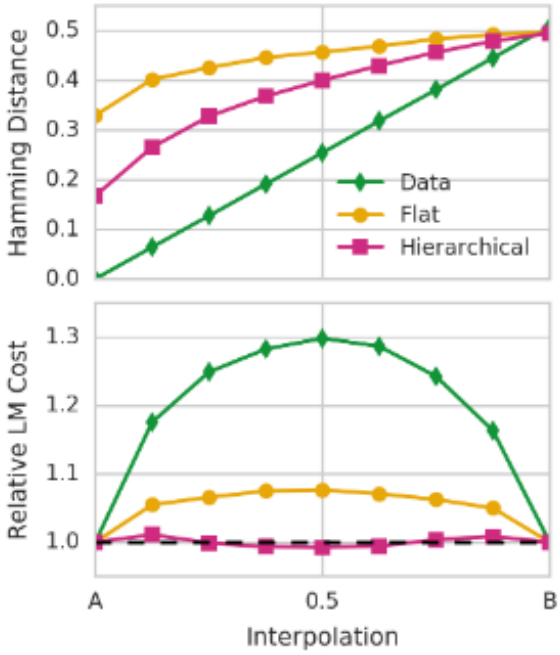


Figure 5.4: Results for quantitative evaluation of true data interpolation, and latent space interpolation with the flat model versus the MusicVAE, reproduced from [104].

Secondly, the model was evaluated with human annotators performing a listening test. The flat and hierarchical model composed 30 second compositions of three types: monotonic melody, trio (polyphonic melody) and a monotonic drum composition. The listeners were asked to evaluate pairs of compositions. The pairs were of melodies from the true data set,

compared to either an example from the flat or the hierarchical model. The listeners were asked to evaluate which model was more musical, on a Likert-type scale. The comparisons show that the hierarchical model produced more musical results than the flat one, and the composed drum sequences were even higher rated than real data. These results suggest that a hierarchical intermediate layer for algorithmic composition helps to preserve information about melodic structure, in a better way than just the VAE alone.

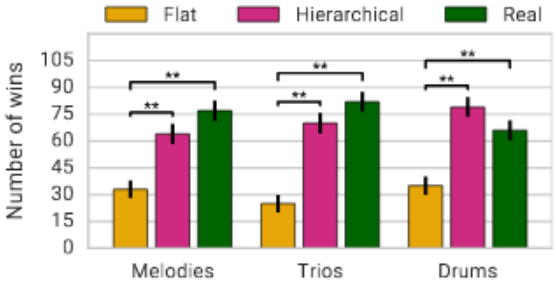


Figure 5.5: Results from listening tests evaluation comparison of true melodies, and melodies from the flat model and MusicVAE, reproduced from [104]. This shows the number of times that each of the two models or a true melody, was considered most musical by the participants.

5.3 World Models

The World Model is an architecture from the paper *World Models* by David Ha and Jürgen Schmidhuber (2018) (Figure 5.6). The model solved a reinforcement learning task of playing two video games *VizDoom* [135] and *CarRacing-v0* [18]. The World Model learns a compressed representation of the RL-environment with self-supervised learning, and then train a subsystem on features extracted from this learning. The subsystem is trained to predict the next feature in a sequence, and then transfer this feature along with the previous feature into an action for the RL-agent. The *features* in this context is latent vectors from a convolutional VAE. The subsystem part that learns to predict latent vectors is an MDRNN. It is this part of the subsystem (Figure 5.7) that is adopted into the The Musical World Model in this thesis, the MCVAE.

5.4 Novel solution: MCVAE

The *Mixture Composer VAE*, MCVAE (Figure 5.11) is a variational autoencoder for symbolic music modelling, with a mixture density network that governs the latent vector sequences, similar to Ha's World Model (the model can be found at `https://github.com/vikrosj/music-variational-autoencoders`). In contrast to the World Model, the MCVAE use a RNN network in the VAE, rather than a CNN. The MCVAE

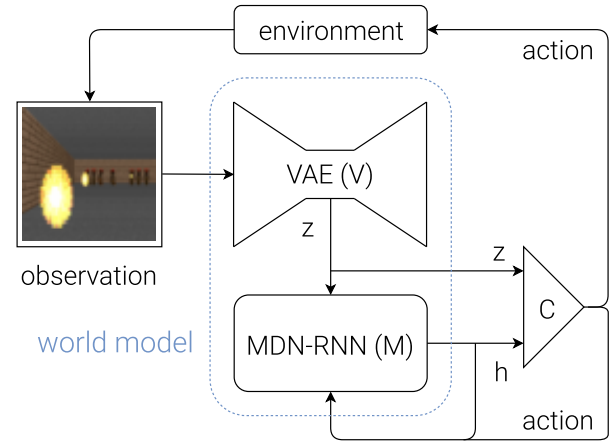


Figure 5.6: World Model architecture, reproduced from [47]

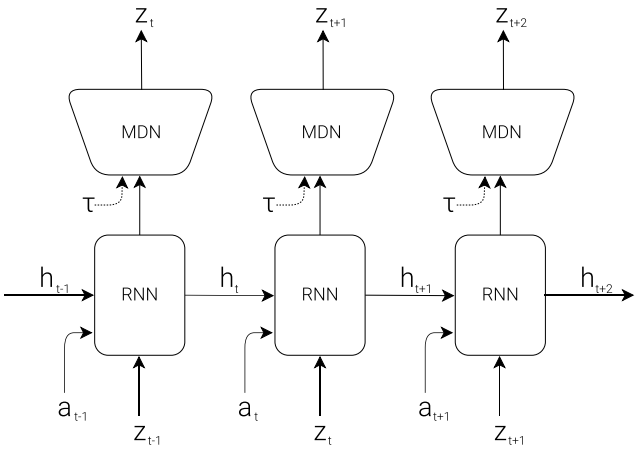


Figure 5.7: Mixture density network predictor in World Models, reproduced from [47]

has a two layer, bidirectional LSTM encoder (see Figure 5.8), and a two layer, unidirectional LTSM decoder (see Figure 5.9). Both with 512 units. Dropout is used on the input of the first encoder layer, with probability $p = 0.3$. Disentanglement for the KL-divergence (see Section 4.4.7) is not implemented, meaning that $\beta = 1$. The latent layer is of size 64. These sizes were chosen after conducting a hyperparameter-test, testing all combination of sizes [64, 128, 256, 512, 1024] for encoder/decoder units and latent layer (see Section 5.4.1 and 6.2). The MDRNN is a two layered unidirectional LSTM, with a MM component on its output (see Figure 5.10). The full model can predict an unlimited number of bars, conditioned on one input bar (see Figure 5.11). The LSTMs have 512 units, and the MM has 10 mixture parameters, resulting in 1290 units (10 mixtures $\cdot$ (64[1] μs $\cdot$ 64 σs + π)). The hyperparameters for the MDRNN were a result of trial and error, but a extensive search was not performed, as it was for the VAE. This will

[1]The input vectors from the latent layer is size 64.

rather be performed in further works.

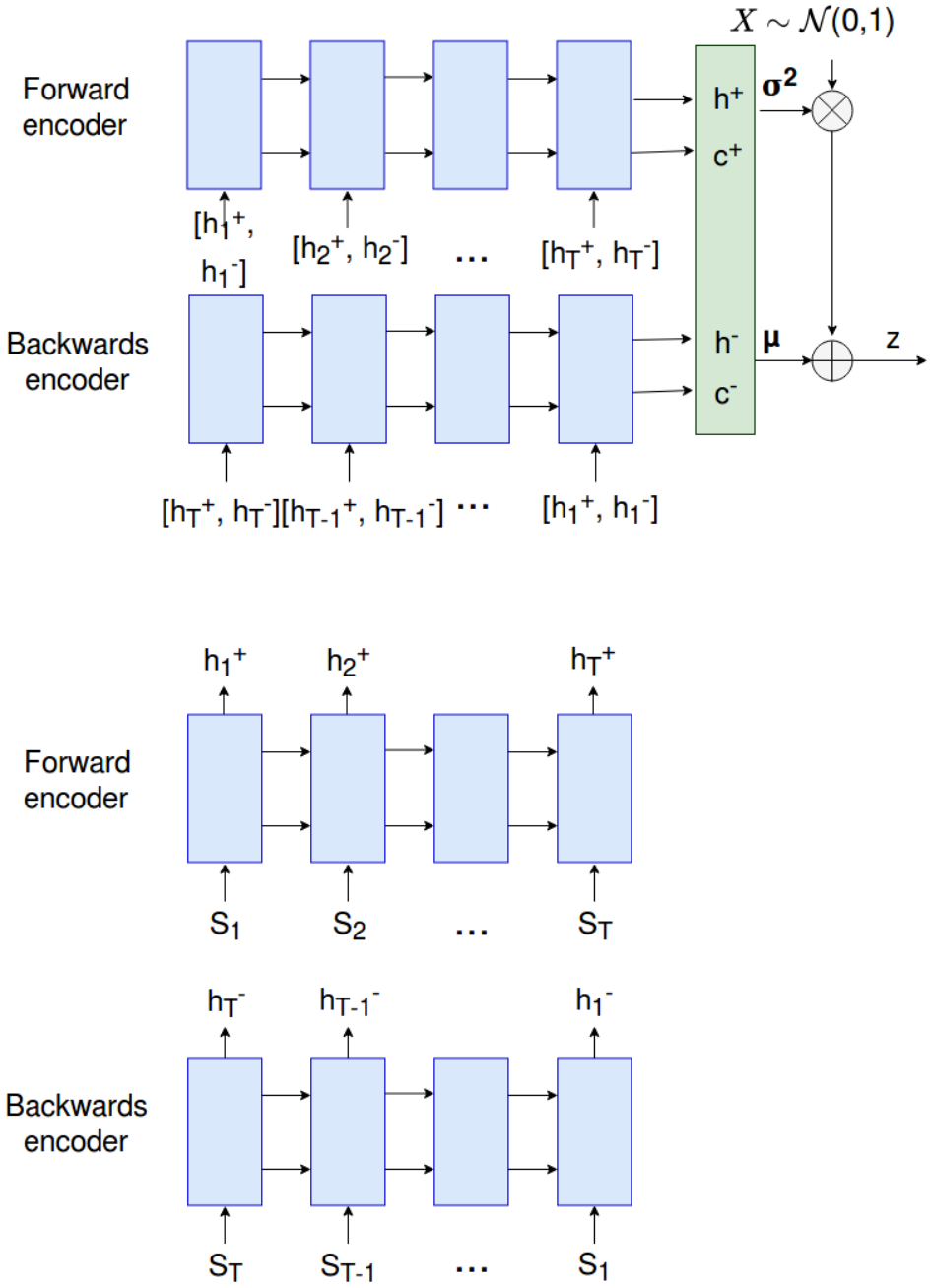


Figure 5.8: Stacked bidirectional LSTMs [139], in the MCVAE encoder.

5.4.1 VAE hyperparameter choices

In the hyperparameter-tests, sizes of 512 (Figure 6.2d) and 1024 (Figure 6.2e) were the ones with a clear indication of learning for the first 200 epochs. But, for size 1024, the network converged at 70% after 50 epochs,

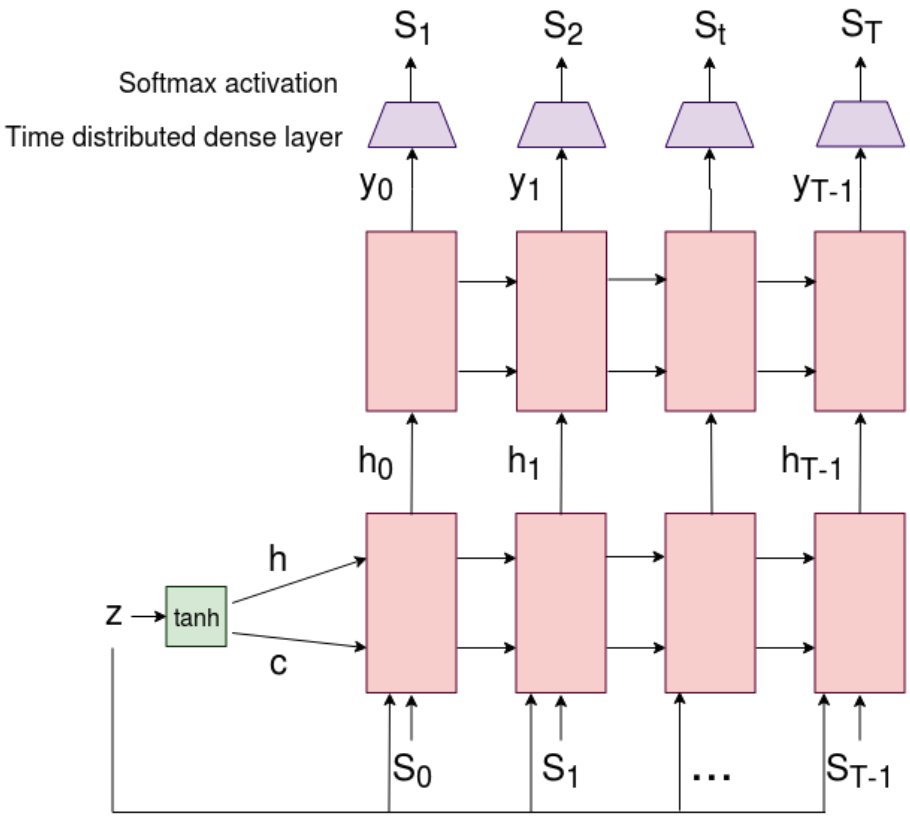


Figure 5.9: Stacked unidirectional LSTMs in the MCVAE decoder.

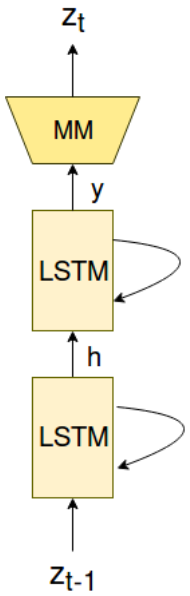


Figure 5.10: Mixture density network composer in MCVAE.

and didn't improve much for consecutive epochs. Encoder/decoder size of 512 showed a steady rate of learning for all 200 epochs, ending at

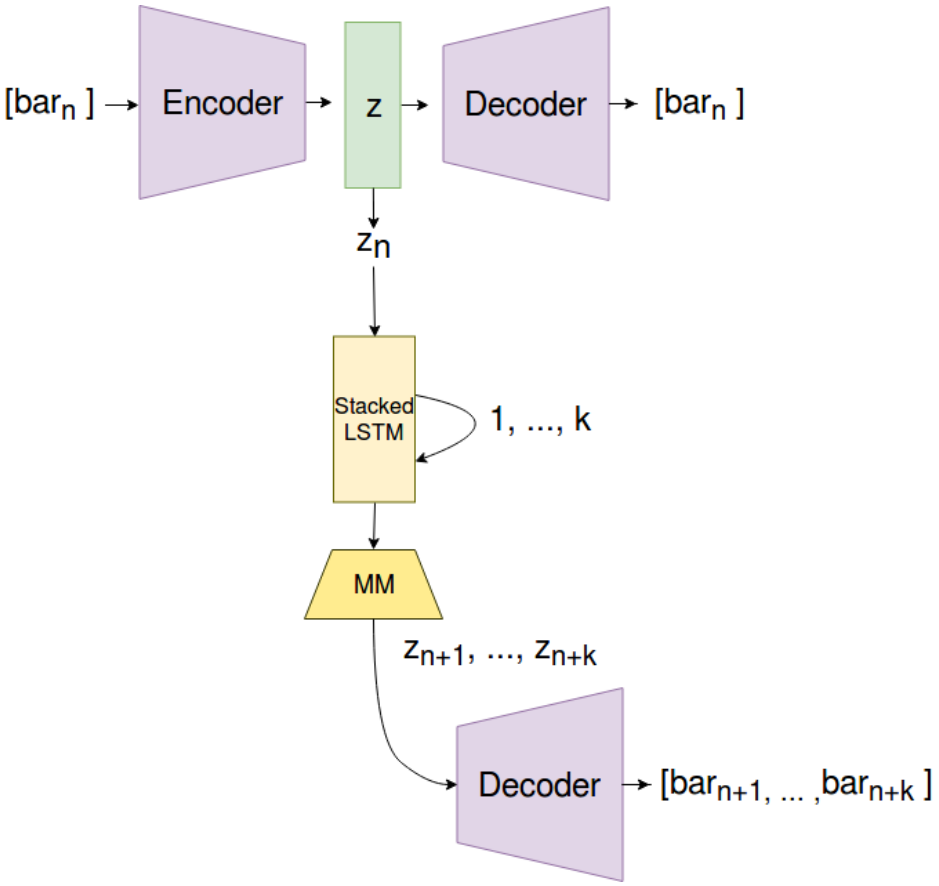


Figure 5.11: Full VAE with MDRNN sequence prediction.

an accuracy of 20% (Figure 6.2d), while sizes 64, 128 and 256 showed serrated learning curves and did not achieve more than 3.5% accuracy at their best (Figures 6.2a, 6.2b, 6.2c, respectively). The latent dimension size didn't appear to have a great impact on the networks accuracy (Figure 6.3), therefore the smallest size (in this trial) was selected. The results from the hyperparameter-tuning is discussed further in Section 6.2.

Chapter 6

Experiments

> You know my method. It is founded upon the observation of trifles.
>
> *Sherlock Holmes, The adventures of Sherlock Holmes [25]*

In this chapter, explanation of and results from all experiments is collected. First, an explanation of how the MIDI files were converted into a dataset for the VAE. Then, how the hyperparameter-tuning was conducted, with a following result evaluation. Explanation on the two choices of model evaluation: human critics and n-gram evaluation. Followed by results from the model evaluations, and finally a summary of the results.

6.1 Dataset

The dataset consist of 10k songs from a subreddit on reddit.com, named *WeAreTheMusicMakers*, posted by a user named *midi_man* [123]. The dataset has been constructed by crawling a large number of websites and eliminating any duplicate files with a checksum function. All songs are from publicly available midi repositories. Originally, the dataset consists of 130k songs, but due to time constraints in parsing and translating the files to note vectors, 10k songs was set as a maximum.

6.1.1 Slicing songs into bars

The songs are sliced into arrays of 16 digits, each digit representing one 16th note, or *semiquaver*. Meaning that one vector is a bar, for a melody with a $\frac{4}{4}$ time signature. While the creators of the MusicVAE chose only $\frac{4}{4}$ songs, that selection has not been done for this thesis. Therefore, a 16 note vector in this case may represent an incomplete bar, or an "overcomplete" bar, as well as a complete bar. The LSTM in the VAE is only concerned with learning sequences of numbers, so the underlying time signature for the notes could be unimportant. The question is whether the time steps for the

LSTM has to be of same length or if it's enough that they are consecutive actions. This could be a topic for further investigation in later works.

The MIDI range

The maximum MIDI note range is from 0 to 127, with the frequency range 8,18-13289,75 Hz [89]. The most common range is for the piano, with MIDI range from 21-108 and frequency range 27,5-4186,01 Hz (from A2 to C8). For the note arrays, the full MIDI range was chosen. In addition to these 128 numbers, two more events were needed to encode tempo in a song, NOTE_OFF and NO_EVENT. The events are encoded with 128 and 129, respectively. The idea for this kind of encoding were first publicly shared by the creators of MelodyRNN and inspired the preprocessing method for MIDIs used here [1]. Lastly, to enable the decoder to create music without conditioning it on a note, a "start of sequence" event was added to the vocabulary, encoded with the number 130. Figure 6.1 illustrates how two bars are translated with this framework.

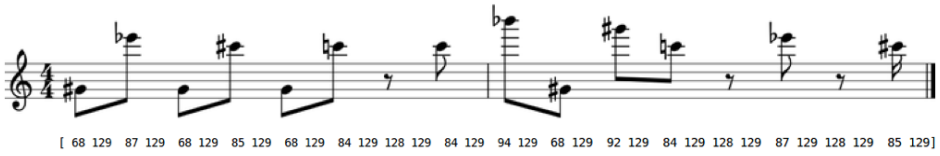


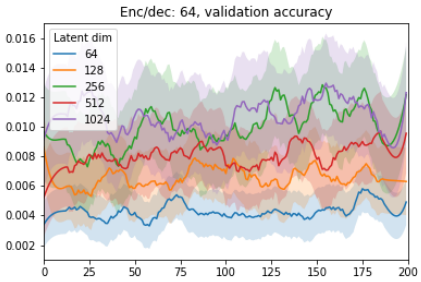
Figure 6.1: Sheet music translated to a note vector. For the input of the LSTM, the full song is sliced into 1×16 vectors.

6.2 Hyperparameter tuning for VAE

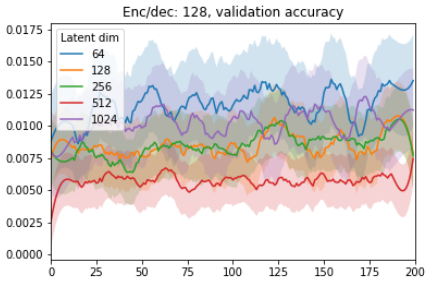
To decide hyperparameters for the VAE, a hyperparameter test of varying encoder/decoder sizes and latent dimension sizes was performed. The motivation was to find the best combination of encoder/decoder size and latent dimension. Different size for encoder and decoder in the same model were not considered, due to time constraints. The evaluation was divided in two: first, a fixed encoder/decoder size against a varying latent dimension (Figure 6.2). Secondly, a fixed latent dimension against varying encoder/decoder sizes (Figure 6.3). The network was trained for 200 epochs for each combination. The sizes for both encoder/decoder layers and latent layer were $[64, 128, 256, 512, 1024]$.

6.2.1 Evaluating the results

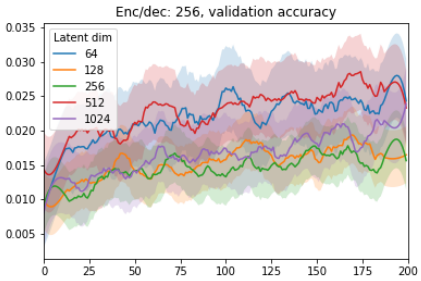
Evaluating the results, it appears that the number of encoder/decoder units are important for the accuracy of the net. Size 1024 showed the steepest learning curve for the first 50 epochs (see Figure 6.2e), regardless of the latent dimension (see Figure 6.3. But, the method also appear to converge at approximately 70% accuracy after 50 epochs, and then not improve much for the next 150 epochs. Encoder/decoder sizes 64, 128 and 256 all



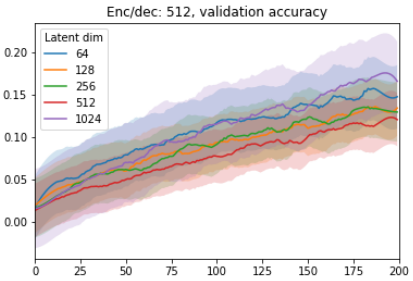
(a) Encoder: 64



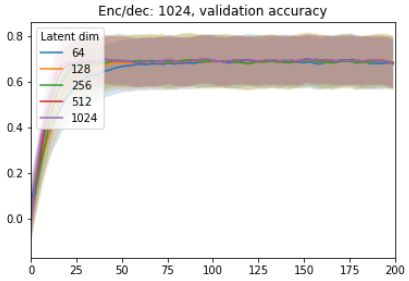
(b) Encoder: 128



(c) Encoder: 256



(d) Encoder: 512



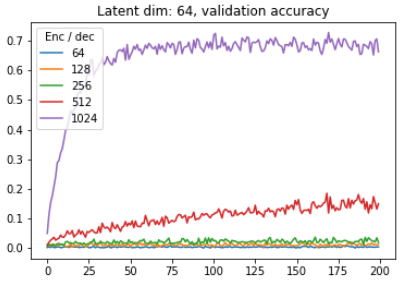
(e) Encoder: 1024

Figure 6.2: Holding encoder/decoder dimension fixed, and varying the latent dimension. The smoothed line is a fitted 3-dimensional polynomial, created by averaging each point x_i by all points $\pm$ 25 from x_i. This method is also known as Savitzky-Golay filtering [100, p. 650]. The errorbar is given by the standard deviation for the current distribution.

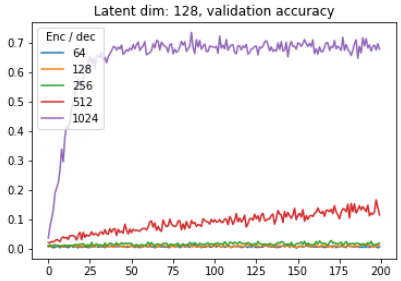
show signs of unstable learning, as the smoothed 3-dimensional polynomials all are serrated for all 200 epochs (Figures 6.2a, 6.2b, 6.2c, respectively). The maximum accuracy for the three methods are respectively 1.4%, 1.5% and 3.5%, approximately. Encoder/decoder size 512 (Figure 6.2d) show a steady rate of learning, as it did not converge during the 200 epochs of hyperparameter-tuning, and the smoothed signal is less serrated than for the smaller encoder/decoder sizes. The accuracy appear to be climbing steadily, and ending up at 20% accuracy after 200 epochs. Therefore, 512 was chosen as the encoder/decoder size. The latent dimension size didn't appear to have a great impact on the networks accuracy (Figure 6.3), there-

fore the smallest size (in this trial) was selected. Possibly, the size could have been reduced even more for the latent dimension, too increase its ability to be a bottleneck for the input data. This can be investigated in further works. The selected sizes after the hyperparameter-tests were:

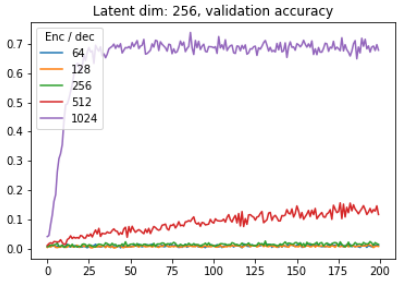
Encoder/decoder: 512
Latent dimension: 64



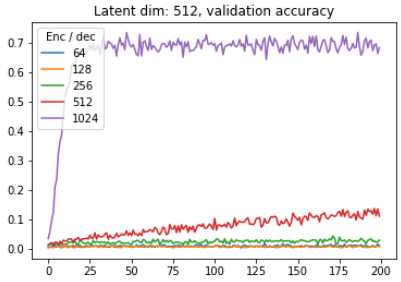
(a) Latent: 64



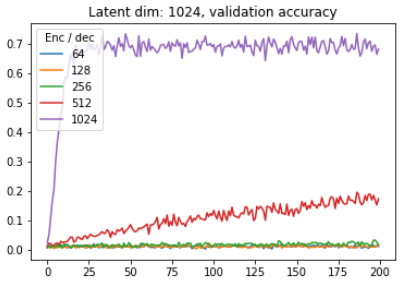
(b) Latent: 128



(c) Latent: 256



(d) Latent: 512



(e) Latent: 1024

Figure 6.3: Holding latent dimension fixed, and varying the encoder / decoder size.

6.3 Training a MDN

After training the VAE, a dataset for the MDRNN was created by inferring z-vectors for all songs in the dataset ($\geq$ 16 bars) and saving these for each

song. The MDRNN was then trained to predict the next z_{n+1} given a z_n (see Figure 3.8) .

During this process, it was discovered that the dataset had some dominating outliers. While most of the songs had number of bars in the range [70-120], some songs were around 1000 bars, and one song had 88000 bars. Elaboration on this discovery, its impact and suggested solutions are discussed further in Section 7.1. Before training the MDN to create music for the human annotators, the dataset was processed, removing all songs with a bar length above 339 bars, or $3 \cdot 113$, where 113 is the average song length of the top 100 song on iTunes in 2012, this is explained further in Section 7.1.

6.4 Evaluating the models

Generative models exist with a variety of applications; compression, denoising and unsupervised feature learning, to name a few. Good performance at one field does not directly imply good performance in another field. It is therefore recommended that the model is evaluated directly with regards to the application it is intended for [124]. To generate music or images with a neural network involves unsupervised feature learning. The model learns the underlying features that make up the data set, and can create new samples from these features. In this field, human annotators are often used as evaluation metric, to evaluate the quality of the samples. Both WaveNet [97], MusicVAE [104], MidiNet [136] and RoboJam [81] use human critics for their evaluations. All of these, except MusicVAE, ask the human critics questions where the response is given in a Likert-type scale with five scales.

After training the model, the true data is represented in a latent space which one can draw samples from. The metric for checking the generative quality of the model is by evaluating the samples from this space. It should contain both a representation of the real data, and new data points with similar features as the real data. If this is the case, then we can say that the model has generalized the data well. Human annotators usually evaluate whether the generated output is distinguishable from real data. While interpolation is a method to evaluate the space between real data points. David Berthelot et al. at Google Brain define high-quality interpolation with two characteristics:

> First, that intermediate points along the interpolation are indistinguishable from real data; and second, that the intermediate points provide a semantically smooth morphing between the endpoints [7].

The metric "semantically smooth morphing" is problem-dependent. Here too, human annotators are used to evaluate how meaningful the interpolation is. Latent space interpolation is typically used in generative modelling of images, like DCGAN [103], IcGAN (an invertible GAN for image editing) [98], and DiscoGAN (a GAN for discovering cross-domain

features in images) [66]. But, in MusicVAE [104], latent space interpolation was also used for evaluating the interpolation between true data points.

After training the VAE, the model was evaluated qualitatively and quantitatively, based on the mentioned methods. The qualitative assessment consisted of a human evaluation, while the quantitative assessment was a probability evaluation using a 5-gram model, performed on latent space interpolation.

6.4.1 Human annotators

8 people with a musical background (either band or orchestra members) participated in evaluating 10 songs: 5 from the VAE and 5 from the MDN. The 5 examples for the VAE and MCVAE evaluation were created by seeding the models with the first bar from the same 5 sample songs from the true dataset. Then, both models inferred melodies of 25 bars, for each sample song. During listening to the music from the outputs, the songs had a noticeably large range that sounded unnatural. Because of this, the darkest notes were transposed up two octaves (+24) and the brightest notes were transposed down two octaves (-24) during preparation. The human critics were asked to evaluate 6 statements in a Likert type-scale. The question topics are inspired by [81] and [83].

The statements were:

1. The compositions had a high musical quality *(quality)* .
2. The compositions showed musical creativity *(creativity)*.
3. The compositions had a melody *(melody)*.
4. The compositions had a good musical structure *(structure)*.
5. The compositions seem easy to play *(technical structure)*.
6. The melodies from the model seem similar *(similarity)*.

6.4.2 Language model evaluation

The quantitative evaluation was a probability evaluation using a n-gram-model with $n = 5$. A n-gram model is a language model, that computes the probabilities of a *n-length* sequence, using the Markov assumption. The Markov assumption states that the probability of an element only depends on the *n-1* previous elements [32].

$$P(w_1, ..., w_n) \approx \prod_i P(w_i | w_{i-n+1}, ..., w_{i-1}) \tag{6.1}$$

The 5-gram probabilities were computed by linearly interpolating in the latent space representations for the five first bars of 200 real songs. 100 songs in *Set A*, and 100 songs in *Set B*. Interpolation of the latent vectors was performed with Equation 6.2. Interpolation was done with intercept degree $\alpha \in [0.1, 0.3, 0.5, 0.7, 0.9]$. Equation 6.2 describes the interpolation

between the latent vectors for song $a \in Set\ A$, and song $b \in Set\ B$. Equation 6.3 describes the true data interpolation (tdi_i). Results from interpolating between the melodies in true data space is presented in Figure 6.6, for comparison with the generative models.

$$z_i = \alpha \cdot z_{a_i} + (1 - \alpha) \cdot z_{b_i} \tag{6.2}$$

$$tdi_i = \alpha \cdot a_i + (1 - \alpha) \cdot b_i \tag{6.3}$$

Figure 6.6 show that the true songs (on the ends A and B), have a low probability on average, around 35%. Indicating that many of the true melodies are considered unlikely by the 5-gram-model. This could mean that some sequences of notes have a much higher probability than others. There is no visible change in probability before $\alpha = 0.5$ in which the melodies are combined 50/50. This results in especially unlikely songs. It appears that on average are songs in Set B slightly less likely than the songs in Set A, and as long as the interpolated song has above 70% of a song from one section present, this song is dominating the outcome. This could mean the songs have somewhat similar structure, or contain many of the same notes, and therefore mixing them slightly does not give very different results.

6.5 Evaluating VAE outputs

6.5.1 Human annotators

The results from the evaluation (Figure 6.4) show that the music from the VAE was lacking in both melody and quality, and the melodies seemed hard to play. The majority agreed that the songs seemed to be similar, created with the same kind of structure. The participants were asked to add their observations, if they had any. The feedback pointed out that the melodies contained a valid bass line. But, the range of notes in one song was considered to be too spread. The melodies seemed to be composed for multiple musicians at once. One also mentioned that if the patterns had been repeated more, then the melodies could sound ”human”. But, the overall feedback was that the melodies sounded random.

6.5.2 Language model evaluation

Conditional interpolation

Using the five first bars' latent vectors from set A and set B, new latent vectors were constructed by linearly interpolating between z_a and z_b (see Algorithm 1). The decoder inferred the melody based on these vectors, and then the probability of the melody was computed using the 5-gram model.

Algorithm 1 Algorithm for the conditional interpolation for the VAE.

```
procedure N-GRAM(A, B)
    for all a,b pairs in A,B do
        α = 0.1
        for i in range 5 do
            pred_song = []
            for j in range 5 do
                z = α· z_a[j] + (1- α) · z_b[j]
                predict sequence, length 16
                pred_song.append(sequence)
            end for
            compute n-gram for pred_song
            α+ = 0.2
        end for
    end for
end procedure
```

The results from the language model (Figure 6.9) show that the interpolation in the latent space produce on average melodies with a low probability, the mean lays approximately around 20%, and the median is even lower than that. As the VAE predict songs with lower probability than the original ones (Figure 6.6), this indicates that interpolation in z-space is not directly translated to interpolation in data space. It could also mean that the latent vectors does not represent the data as they should, or that the models weights aren't adjusted properly. Although the model was trained until accuracy was high and loss was low, this does not necessarily mean that the VAE generalizes well. If the data is biased, the VAE will be biased. The KL-penalty may also be too low, as is postulated in Section 4.4.7, and therefore will the latent space encode information that is not relevant for reconstruction.

Unconditional interpolation

For the unconditional interpolation, the VAE was fed the first vector from song a and b, and then inferred a sequence length of 400 timesteps, meaning 25 bars (see Algorithm 2).

The results from unconditional latent space interpolation (Figure 6.9) show that the probability of the outputs are averaging just below 60%, which is much higher than the conditional interpolation. But the variance in the data range from very low to very high. By only conditioning the VAE with a starting vector, z, it appears to struggle with making melodies with a consistent structure. The VAE should not be expected to perform better than true data, but by using the logic that a VAE can infer new melodies, one could expect that it would be able to create songs with somewhat the same structure as it's input vector. These results indicate that the VAE doesn't follow the structure of the input vector for a full sequence, creating output that is considered to be either very unlikely or very likely. For the

Algorithm 2 Algorithm for the unconditional interpolation for the VAE.

```
procedure N-GRAM(A, B)
    for all a,b pairs in A,B do
        α = 0.1
        for i in range 5 do
            z = α· z_a[0] + (1- α) · z_b[0]
            predict sequence, length 80
            compute n-gram for sequence
            α+ = 0.2
        end for
    end for
end procedure
```

two outermost interpolations ($\alpha = 0.9$ and $\alpha = 0.1$), 50% of the values has a probability close to 90%, while the probabilities below the median are very spread. This result is almost the same for all of the interpolation sequences, the bottom half of the results vary greatly, indicating that there is no clear pattern in the musical creation process.

6.6 Evaluating MCVAE outputs

6.6.1 Human annotators

The 5 songs for the MCVAE evaluation were created with the same 5 sample songs as the ones used for conditioning the VAE. But, here, the MDN predicted a sequence of 25 bars, only conditioned on the first latent vector, and the decoder inferred the melodies based on this sequence of latent vectors. The quality, melody and structure of this model was also considered to be poor. But, the technical structure got a higher rating than for the VAE. There were a majority agreement here as well that the melodies sounded similar. The participants' feedback showed that the melodies appeared to have a little more musical quality than the VAE. The melodies had fewer, and longer notes, and some rhythmic structure. One comment noted that the model seemed to find some patterns, but didn't evolve these. The general assessment was that these songs also appeared random to the human annotators.

6.6.2 Language model evaluation

The language model evaluation for this section was done both with the MDN trained on inferred latent vectors (from the VAE) for the unprocessed dataset, and the processed dataset. This is because the melodies for the MDN evaluated by humans were created with the MDN trained on latent vectors inferred from the processed dataset.

Conditional inference

Conditional interpolation was done with Algorithm 3. The MCVAE-results for the unprocessed dataset (Figure 6.8) show on average the same probabilities as songs in Set A and B. While the VAE performed with even lower probability than the true data, the MDRNN appear to be able to predict melodies with approximately the same probability as the ones in the true data set. For interpolation degree $\alpha = 0.5$, the MDRNN predicts sequences that are more probable than the true data interpolation. This may be due to the fact that the MDRNN has fewer number of mixtures than the original z-vector, and by that creates more definite and probable latent vectors. The MDRNN use 10 mixtures, while the original latent vectors has dimensionality of 64. The MDRNN may be filtering and improving the latent vector distributions to more clear and definite probability distributions. Figures 6.7 and 6.11 show that the VAE with an MDRNN for latent vector prediction produce more probable interpolations between true songs than the VAE does alone.

Algorithm 3 Algorithm for the conditional interpolation for MDN

```
1:  procedure N-GRAM(A, B)
2:      for all a,b pairs in A,B do
3:          α = 0.1
4:          for i in range 5 do
5:              pred_song = []
6:              pred_song.append(α· z_a[0] + (1- α) · z_b[0])
7:              for j in range (1,5) do
8:                  prev_z = α· z_a[j-1] + (1- α) · z_b[j-1]
9:                  predict z from prev_z
10:                 predict sequence from z
11:                 pred_song.append(sequence)
12:             end for
13:             compute n-gram for pred_song
14:             α+ = 0.2
15:         end for
16:     end for
17: end procedure
```

Unconditional inference

Unconditional interpolation was done with Algorithm 4. The unconditional inference for the MCVAE (Figure 6.10) created sequences with a mean of approximately 55% probability, which is more likely than the true data points in set A and B, the true data-interpolation (Figure 6.6) and the unconditional inference for the formal VAE (Figure 6.9). Additionally, 50% of the data has a probability ranging from 50-70%. This is a much more stable result than that from the unconditional interpolation of the VAE, where 50% of the data was spread from approximately 20% probability to 90%

probability. These results show that, even though the melodies from the MCVAE were not regarded as much more pleasing than the VAE-melodies by the human critics, they were on average more probable than true data-interpolation. The melodies in these samples were only conditioned on one latent vector, and the rest of the melody was inferred by the MDRNN-layer. The results from the interpolation in this case shows that the MDRNN produce latent vectors with a smooth transition and overall similar score in probability. It is reasonable to assume that with a better representation of data and a dataset with less noise, the MCVAE would compose melodies which were even more probable, and that better captured a musical structure. Unconditional interpolation was also performed after the dataset was cleaned for dominating melodies (mentioned in Section 6.3). The results from this test show that the melodies here averaged approximately as the melodies from the uncleaned set (see Figure 6.10), but the samples had fewer outliers. This may be relating to the fact that the dataset this MCVAE was trained on also had fewer outliers than the original dataset.

Algorithm 4 Algorithm for the unconditional interpolation for MDN

```
procedure N-GRAM(A, B)
    for all a,b pairs in A,B do
        α = 0.1
        for i in range 5 do
            pred_song = []
            prev_z = α· z_a[0] + (1- α) · z_b[0]
            pred_song.append(prev_z)
            for j in range (1,5) do
                predict z from prev_z
                predict sequence from z
                pred_song.append(sequence)
                prev_z = z
            end for
            compute n-gram for pred_song
            α+ = 0.2
        end for
    end for
end procedure
```

6.7 Summary of results

The MCVAE was rated higher than the formal VAE on structure, melody, creativity and musical quality, by the human annotators, see Figures 6.4 and 6.5. In the conditional interpolation, the MCVAE (Figure 6.8) performed better than the VAE (Figure 6.7) slightly below the true data interpolation (Figure 6.6), but better than true data interpolation for interpolation degree $\alpha = 0.5$. The latent vector space from the MCVAE appear to have smoother transitions of sequence interpolations than both the true data and the VAE. For the unconditional inference, the VAE performed extremely irregular (Figure 6.9), while the MCVAE created sequences that were all more likely than the true data points in set A and B (Figure 6.10). Unconditional inference by the MCVAE trained only on samples from the cleaned dataset performed better than than the MCVAE trained on the original, noisy dataset, as there were less outliers in the samples (Figure 6.11). This appear to correlate with the fact that the dataset this MCVAE was trained on had fewer outliers than the original dataset.

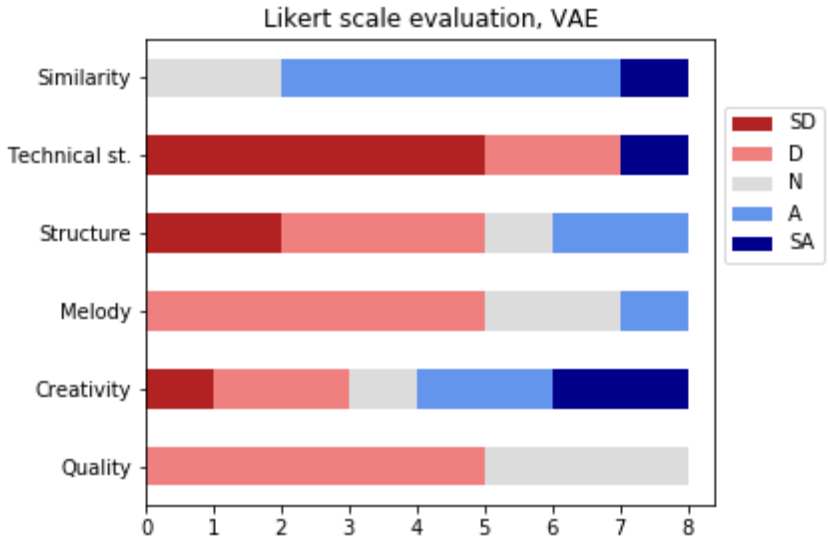


Figure 6.4: Results from the Likert type-scale evaluation of the 6 statements in Section 6.4.1 for the formal VAE. On the x-axis are number of people. Acronyms for the scale factors: SD: Strongly disagree, D: Disagree, N: Neutral, A: Agree, SA: Strongly Agree.

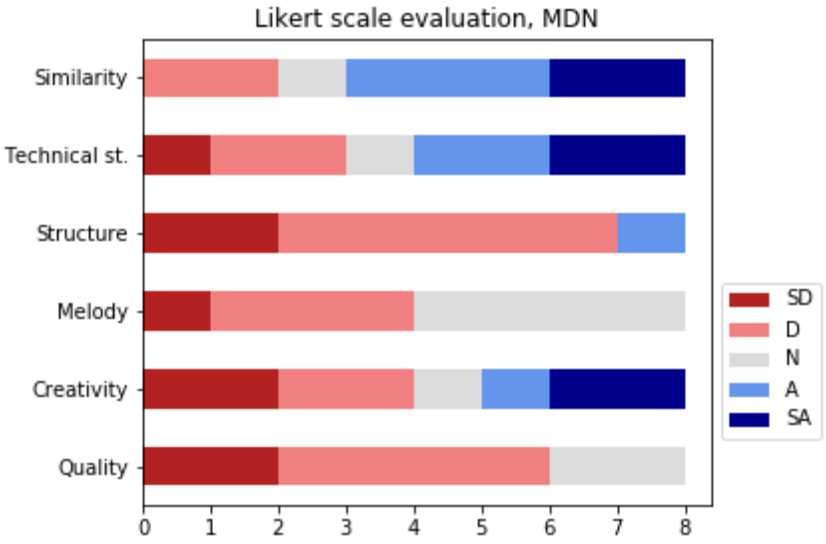


Figure 6.5: Results from the Likert type-scale evaluation of the 6 statements in Section 6.4.1 for the MCVAE. On the x-axis are number of people. Acronyms for the scale factors: SD: Strongly disagree, D: Disagree, N: Neutral, A: Agree, SA: Strongly Agree.

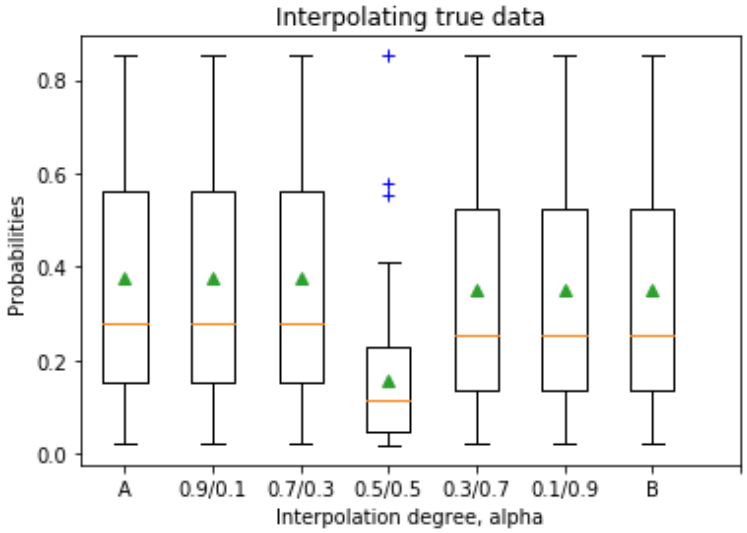


Figure 6.6: Interpolating in true data space between songs in Set A and songs in Set B.

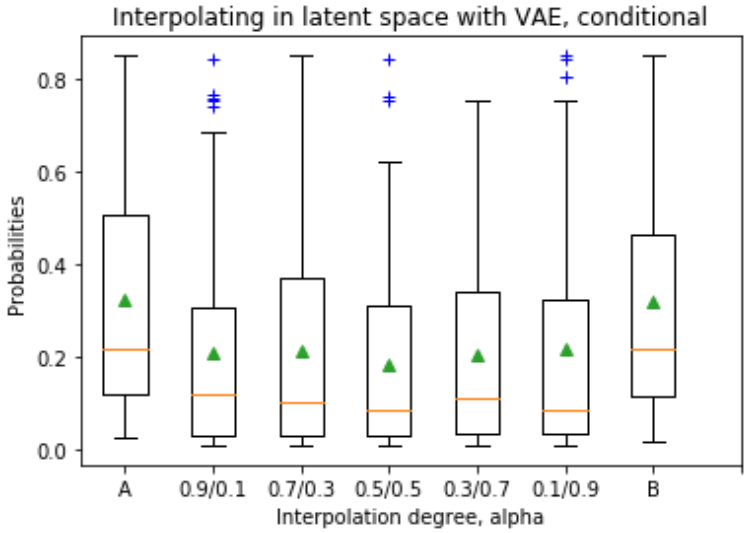


Figure 6.7: Interpolating in latent space with VAE, conditionally.

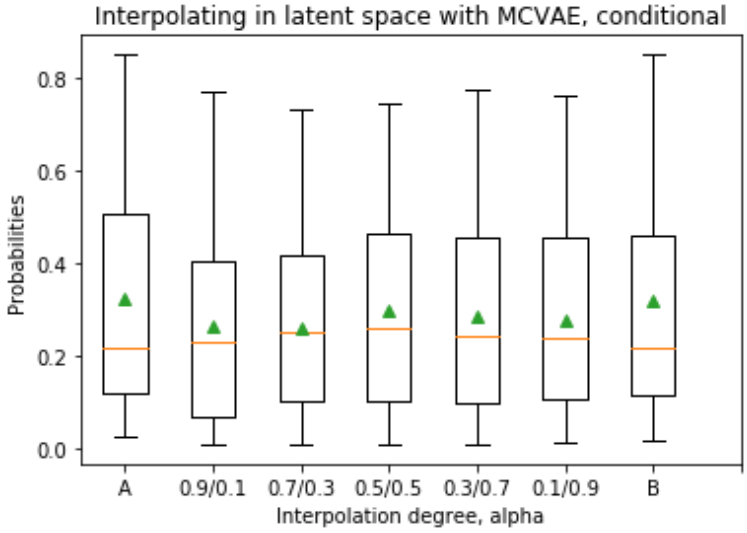


Figure 6.8: Interpolating in latent space with MCVAE, conditionally.

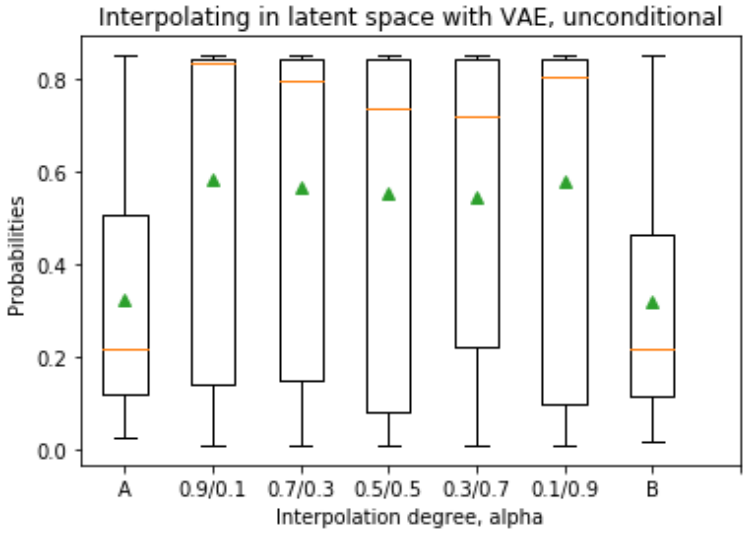


Figure 6.9: Interpolating in latent space with VAE, unconditionally.

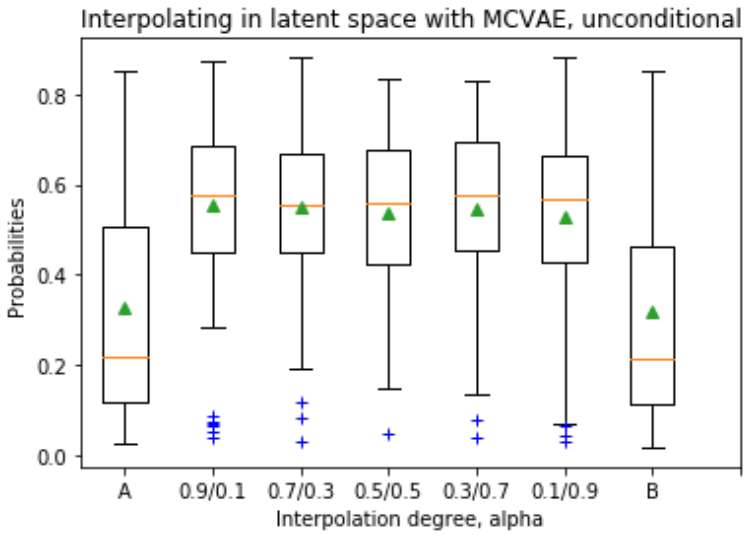


Figure 6.10: Interpolating in latent space with MCVAE, unconditionally.

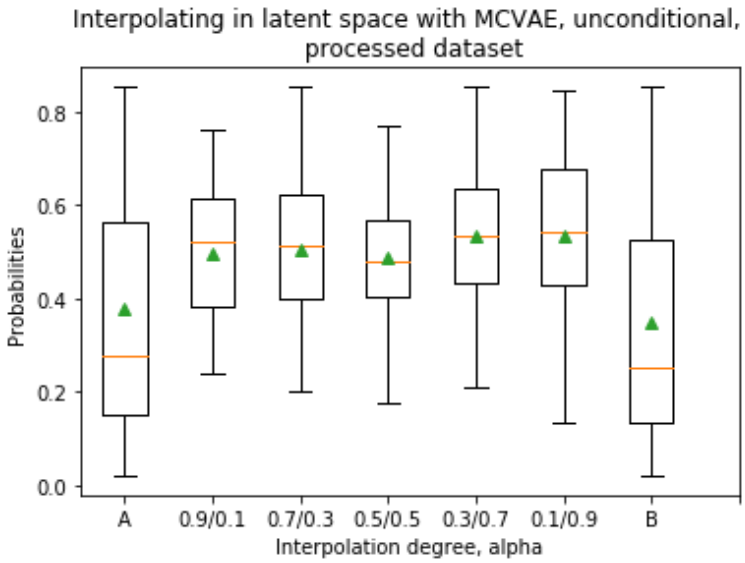


Figure 6.11: Interpolating in latent space with MCVAE, unconditionally, after cleaning the dataset.

Chapter 7

Discussion

> Problems worthy of attack
> prove their worth by fighting
> back.
>
> *Piet Hein*, Grooks 1 [52]

This chapter discuss the challenges with creating a dataset from MIDI files for a neural network. It addresses a problem that occurred for this thesis' experiments with a noisy dataset. Then, some thoughts on range selection and the premise that data for a VAE should be independent and identically distributed - which time series data is not. Lastly, the point from the introduction on the creativity criterion for a generative model is discussed.

The results from a machine learning model is only as good as the data it uses [17]. As the goal for this model was to be a versatile application for creating music, that means that the samples in the dataset should be somewhat evenly distributed - so no melody would have an increasingly higher chance of being selected for each batch during training. A flaw in the creation of the dataset used for this project was not ensuring this criterion. The MIDI files in the *Big Data Set* were expected to have somewhat similar lengths, and their sizes weren't checked before pre-processing the data for the MDN. While the evaluation supported the use of the MDN to help govern structure, the VAE may produce more pleasing bars of music if it is traines using a better structured dataset.

7.1 Dominating outliers

During the construction of the MDN dataset, the length of each song was evaluated. This lead to the discovery that the data was unevenly distributed. Creating a boxplot of the bar length distribution showed that while the mean and median of the data appeared to lay close, some outliers were dominating the data (see Figure 7.1).
The dataset's common statistical metrics were:

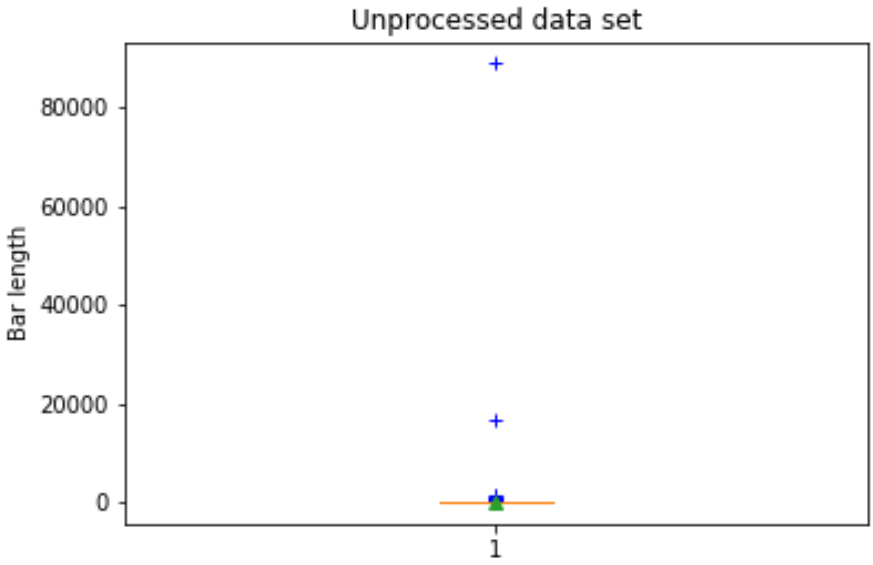


Figure 7.1: Boxplot of bar distribution in the dataset. The mean is represented by a green triangle, median by a orange line and the outliers as blue crosses.

Standard deviation	906.39
Median	82.0
Mean	95.76
Minimum value	1
Maximum value	88863

7.1.1 Computing average song length

The average song length for the top 100 songs on iTunes (in 2012) were at 226 seconds [5]. One bar equals 2 seconds [35], meaning that the average song length would be around 113 bars. To evaluate the distribution of bars in the songs of the dataset, the data was processed with two different maximum values. One keeping everything less than $3 \cdot 113$, and another keeping everything less than $2 \cdot 133$, 113 being the average song length top 100 songs on iTunes (in 2012), as mentioned.

Figure 7.2 show that while approximately the same amount of data lies between the first and third quadrant, and their means and medians are almost similar, the dataset with less than $3 \cdot avg\_len$ has more outliers. Plotting a histogram of the bar distribution for the two sets show the same results (see Figure 7.3). Viewing Figure 7.3, it is apparent that there aren't many songs above the average song length.

Both the VAE and the MCVAE got feedback from human annotators that the music sounded random, poorly structured and appeared to be

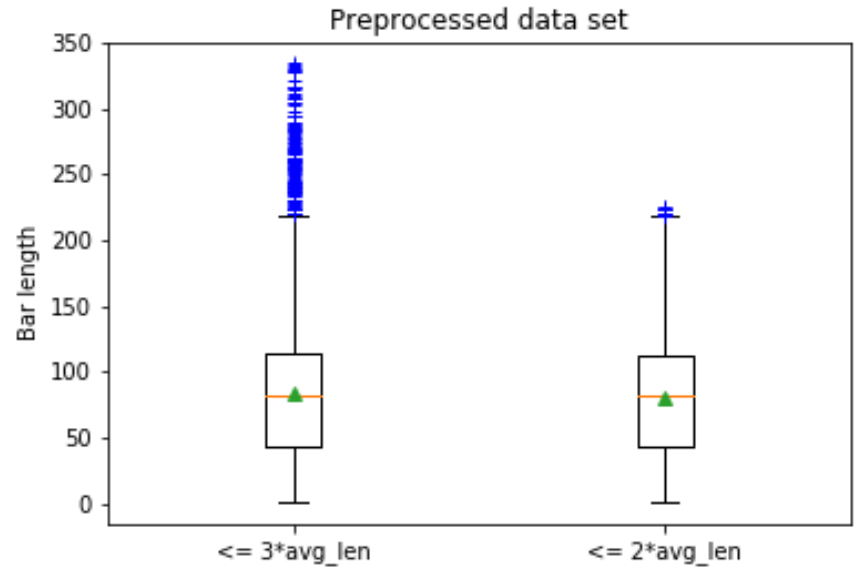


Figure 7.2: Boxplot of bar distribution in the preprocessed dataset. The mean is represented by a green triangle, median by a orange line and the outliers as blue crosses.

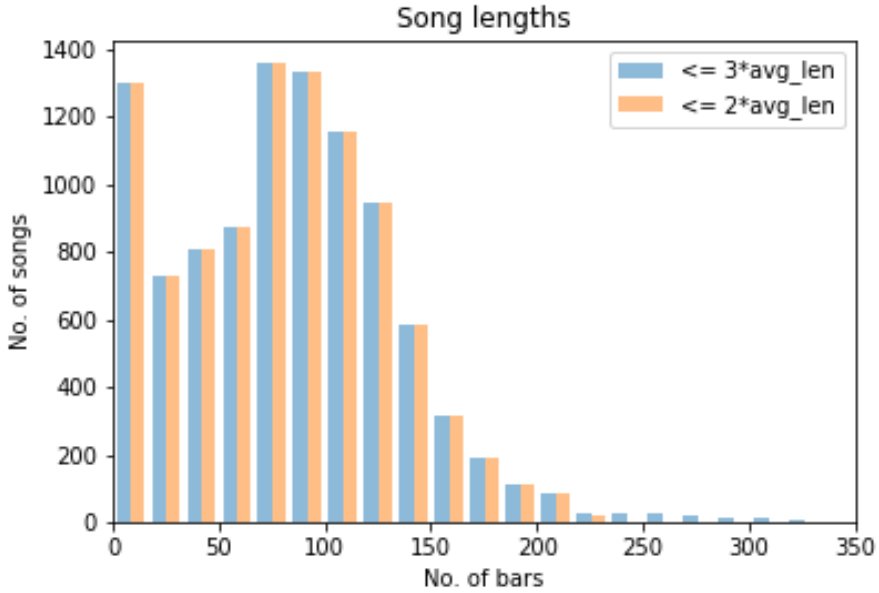


Figure 7.3: Histogram of bar distribution in the two processed datasets.

created for multiple musicians simultaneously, even though the MCVAE were rated higher. The fact that a few melodies had a much higher occurrence in the dataset than the others may have caused this. After this was discovered, the functions for parsing MIDI files and selecting songs

have been changed, keeping only songs less than $3 \cdot avg\_len$, in future use.

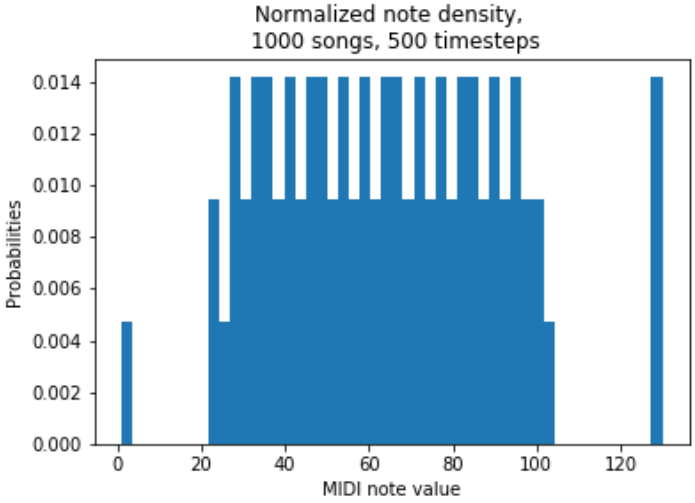


Figure 7.4: Note density for 1000 songs in the dataset.

7.2 Range selection

Although some notes have a higher probability of being created than others, there are a few measures that can be taken to create a slightly more even distribution of notes. As is apparent in Figure 7.4, the note range for the 1000 selected songs lay approximately between 20-100. As the piano range for MIDIs lay in the range 21-108 [89], this may be a more appropriate range to encode the notes into.

7.3 Independent and identically distributed data

In music modelling, some notes have a higher probability than others. The reason for this is that there exist a tonal hierarchy, where some notes are "more prominent, stable, and structurally significant than others" [68, p. 52].

A result that is not very apparent from Figure 7.4 becomes more visible when inspecting the 5 most common notes in the songs from the example above:

NO_EVENT 129.0 occurence: 259583

NOTE_OFF 128.0 occurence: 78115

A4 69.0 occurence: 8289

G4 67.0 occurence: 7287

D4 62.0 occurence: 6527

Codes 129 and 128, standing for NO_EVENT and NOTE_OFF, has a much higher occurrence than the rest of the notes. Every note in the corpus includes 0 or more NO_EVENT, and notes longer than one semiquaver require multiple NO_EVENT codes, thus it is dominant in the dataset. These two values encode rhythm in the melodies, and therefore, their prominent presence is not surprising. But, their presence also distort the tonal hierarchy. When inspecting a song with this encoding, the rhythm-values are the most prominent, stable and structurally significant. And, although rhythm is structurally significant, pauses and silence does not make music alone. So, the notes should have a more leveled presence, to counter the rhythm values.

Additionally, a variational autoencoder assumes a independent and identically distributed (i.i.d) dataset, meaning that the random variables are independent and have the same distributions [57]. An example of i.i.d. events is throwing a dice, each throw is independent of the others, and the events all have the same probability distribution. A time series is, in general *not* i.i.d. But, while keeping these two things in mind, another way of encoding the music should be investigated and pursued. In the MusicVAE-project a $3 \times N$ matrix was used [105], N being the number of notes per matrix. The three rows were assigned for *note value*, *start time* and *end time* for the note. Both start and end values are cumulative. Meaning that previous end time is the start time for the next note. This way of encoding reduces the occurrences of notes, and may be a better way of encoding. But, the optimal solution for encoding music should be investigated further before drawing any conclusions on this topic.

7.4 Creativity

7.4.1 Accuracy as a metric

Although the method for training a variational autoencoder is by teaching it to reproduce input, reproduction of true data is not the desired outcome for the model. In this sense, accuracy is a somewhat counter intuitive metric. It is not until after the model is trained, that one can evaluate the model based on its true purpose: the ability to generate new data with meaningful structure. A model that is trained to reproduce input is also just learning from the data it sees, and with this in mind, one can argue that the creativity of the model is limited by the dataset. Another point regarding accuracy is that the very failure of a model to recreate the original data may be the desired outcome, giving new and interesting samples [122]. The measure of success of a ML-model for music is more subjective than that of a ML-model used for instance for image recognition. As music is a complex form of data and experienced differently among subjects, this is an important thought to keep in mind.

7.4.2 Criteria of usefulness

Whether or not a ML model for music modelling can be useful is in itself a topic for modern day research. With one argument being that the ML models only learn limited aspects of music like melodies, harmonies and temporal structure. And, the model is first useful when human musicians contribute their expression to the machine made arrangements [80]. Keeping this in mind, a ML model can participate in musical creations as a "musical other", that can be altered and tweaked to the musicians taste. It can also challenge creativity, by creating arrangements that were unlikely to the human musician[122]. Research on ML-HCI-systems or "intelligent listener"-systems [82] that adapts to human performance, show that an intelligent agent improve aspects of improvisation of music [83]. The "intelligent listener"-system is a HCI where the GUI changes based on predictions the systems make about the human musicians input. The GUI only change occasionally, as it listens to the human performance, and only make changes when it predicts that this will fit the performance. As the adaptive system seem to improve improvisation, this is an argument for incorporating machine intelligence in digital musical instruments.

The neural network in this thesis generalizes to all forms of sequential data, in which each data point is a complex probability distribution. This neural network can be incorporated in human-machine collaboration systems, like the "intelligent listener"-system. It may also be incorporated in other musical ML-systems like systems that mimic an ensemble, that improvise along with a human musician [79]. The arguments for the usefulness of a predictive musical ML-model are compelling. A ML-model can both be tweaked to adapt the musical style of the performer. Secondly, as research indicate that a responsive ML-model may improve aspects of improvisation for human performers, the usefulness for this field of science seems obvious.

Chapter 8

Conclusion & Future work

> 'As you yourself have said, what other explanation can there be?' Poirot stared straight ahead of him. 'That is what I ask myself,' he said. 'That is what I never cease to ask myself.'
>
> *Hercule Poirot, Murder on The Orient Express [23]*

The main contribution of this thesis is the introduction of a new hierarchical sequence prediction method, applying a MDRNN for latent vector predictions for VAEs that generate symbolic music. The novel method is named *Mixture Composer VAE*, MCVAE. The method can also be seen as a generic method for hierarchical sequence prediction, and be used for other generative tasks than symbolic music modelling. Additionally, this research also confirms something that has been addressed before in research: music modelling is improved when structural hierarchy is implemented into the model.

8.1 Summary of Evaluation Findings

Before addressing the research questions, the main evaluation findings will be summarized.

8.1.1 Human annotators

8 human critics with musical background (either band or orchestra members) participated in a listening test, comparing 10 melodies of 25 bars ($\approx$ 50 seconds), 5 melodies from the VAE and 5 melodies from the MCVAE. Both the VAE and the MCVAE were seeded with the first bar from the same 5 songs, from the true data set. Then, the models inferred 25 bar melodies.

Results

The MCVAE was rated higher than the formal VAE on structure, melody, creativity and musical quality, by the human critics.

8.1.2 Language model evaluation

A 5-gram model was used for assessing the probability of sequences during linear interpolation in latent space. The evaluation was done between the latent space representations of 200 songs from the true data set, 100 songs labeled *Set A* and 100 songs labeled *Set B*. Interpolation was done for the 5 first bars of the selected songs. Interpolation was done with intercept degree $\alpha \in [0.1, 0.3, 0.5, 0.7, 0.9]$.

The models interpolated in two ways: conditionally and unconditionally. In the conditional interpolation, the VAE was given all the interpolated latent vectors between song $a \in Set\ A$ and song $b \in Set\ B$, and inferred the output from this. In the conditional interpolation of the MDN, the model predicted latent vector z_{n+1}, given z_n from the song. The first latent vector z_0 could not be predicted using this method, and was used directly.

In the unconditional interpolation, the VAE was seeded with the interpolated latent vector z_0 for each a/b-pair, and inferred a melody of sequence length 400, meaning 25 bars. The MCVAE was given the same vector z_0, and predicted 24 vectors $(z_1, ..., z_{24})$ from this.

The latent space interpolation was also compared to interpolation in true data space. In true data space, a 50/50 interpolation produced a highly unlikely output. The other intercept degrees were considered to be equally likely, this could be due to the fact that many of the songs had similar structure.

Results

The language model evaluation shows that in the conditional interpolation, the VAE produced less probable results than the true data interpolation. The MDN creates melodies with equal probability as the true songs, suggesting a smooth interpolation between the sequences. The MDN should, in theory, output the same z_n as the VAE, but the conditional interpolation results suggest that the MDN alters the output somewhat, perhaps outputting latent vectors with higher probability than that the ones from the VAE.

In the unconditional interpolation, the VAE had a mean at 60% probability, but with an extremely high variance. For most of the interpolation degrees, the VAE-probabilities ranged from 10% to 90%, suggesting a very unstable model. The MDN performed much more stable in this test, with mean at 55%, and first and second quartile at 45% to 65% for all interpolations. This output has a higher probability than the true data interpolation, suggesting that the MDN had learned a probable melodic structure.

These results suggest that using a MDRNN for hierarchical melodic composition outputs songs that are more similar to true data, than when letting the VAE infer melodies without hierarchy. According to the human listening test results, even though the MDRNN layer facilitate more probable sequences, additional measures must be taken in order to create a generative musical model that produce results that are pleasing to the human ear.

8.2 Research questions

In this section, the research questions set out in Section 1.2 will be addressed individually.

8.2.1 What are the main technologies for using ANNs to model and compose music?

The state-of-the-art methods today for generative modelling is by using either Generative Adversarial Networks (GANs) or Variational Autoencoders (VAEs). GANs model the data explicitly by training a generator that create samples from a noise vector, and a discriminator that evaluates whether a sample is from the true data distribution or created by the generator. By using GANs, the network can learn the exact distribution of the data. VAEs models the data implicitly, by assuming that the data points are represented by a Gaussian or Bernoulli distribution. The parameters for the assumed distributions are updated using Bayesian inference, in which the model samples true data and updates its parameters each time it receives new information.

8.2.2 In what ways can music be represented to an ANN?

In this thesis, music is represented to the ANN model by vectorizing MIDI files. A melody, or a monophonic track from a MIDI file, is encoded with a note range from 0-127 (the full MIDI range), and value 128 to encode "note off" and 129 meaning "no event", for rhythm encoding. Additionally, a "start of sequence"-token is encoded with the number 130. The songs are encoded as one dimensional vectors. A song is sliced into 16 semiquaver-sequences, translating to one bar with time signature $\frac{4}{4}$. However, songs are not selected based on time signatures, so a 16 length sequence can be less than or more than one bar. When presenting one bar as input to the model, the 16×1 vector is one hot encoded, resulting in a 16×131 vector. In the MusicVAE-project a $3 \times N$ matrix is used [105], N being the number of notes per matrix. The three rows were assigned for *note value*, *start time* and *end time* for the note, and start and end values are cumulative.

8.2.3 How can a VAE be used to compose long compositions?

The ways in which a VAE composes long compositions today is either by inferring infinitely long sequences, conditioned on a latent vector from

a true sample, and usually a "start of sequence"-token. The other way in which a VAE can compose long sequences is by adding a hierarchical model in between the latent layer and the decoder. In this thesis, a new hierarchical model was implemented applying a mixture density model for modeling the sequences of latent vectors in song predictions.

8.2.4 How can we evaluate the success of a creative ANN model?

Among common methods for evaluating a generative model is using human critics that evaluate the models creations. Typically, they answer questions about the samples' pleasantness, creativity, interestingness, structure and so forth. The critics use a Likert-type scale for answering. Another method is to ask the human annotators to perform a Turing test of the samples, and evaluate the samples' resemblance to real world compositions. A common quantitative method is to perform latent space interpolation between true data samples, and translate these interpolations to data space. Then, the semantic meaningfulness of the interpolation is typically evaluated, which says something about how scattered the points in the latent space are, and if they hold relevant information. In this thesis, qualitative evaluation was done by 8 human annotators who evaluated the samples' on 6 different criteria and answered using a Likert-type scale. The quantitative evaluation was done using a 5-gram model to evaluate the probability of created sequences by the VAE and the novel model, MCVAE.

8.2.5 Can a MDRNN be used to steer a VAE model of music?

This thesis concludes that a MDRNN produce more probable sequences than a VAE alone, indicating that the MDRNN learns to build probable sequences of latent vectors. Considering this, a MDRNN can steer a VAE model of music.

8.2.6 Does an MDRNN/VAE system produce better compositions than a VAE alone?

The 5-gram evaluation of the two models definitely indicates that a MDRNN/VAE system produce better compositions than the VAE alone. The human critics rated the MDRNN/VAE higher than the formal VAE on structure, melody, creativity and musical quality. Albeit, both models performed poorly in this test. The main reason for this is suspected to be concerning data quality. The dataset was found to contain noisy samples at a late stage of the project, which could have effected the distribution of probable sequences. A high rate of true melodies in the dataset are regarded as very unlikely with the 5-gram model, indicating that some features in the data have a too high presence, making the model highly specialized in these features.

As an example of low quality data in the corpus, a song of roughly 88000 bars was discovered, which turned out to be almost all 129s, during inspection of the song. This did not improve the data quality, considering

the fact that the dataset preferably should be independent and identically distributed. In future work, higher quality datasets could be used to experiment with the MDRNN/VAE-system. One possibility would be to apply the recently released MAESTRO dataset [50].

8.3 Future Directions and Final Remarks

The methods for algorithmic composition has grown in complexity since the Musikalisches Würfelspiel marked the start of the algorithmic musical exploration in the 18th century. Algorithmic composition is now a field of machine learning, where the models are able to produce new samples of music, in stead of being restricted by reiterating existing musical sequences. On a more ambitious note, generative models can contribute further than just musical creation. The most challenging tasks for machine learning models today is to understand sequences and "recreate human-like semantic understanding for complex actions" [118]. These skills are what generative models can contribute to machine learning. Therefore, generative modelling is interesting and important both for algorithmic composition, and for the general study of AI.

8.3.1 Data processing

Given the results of the evaluation, it is sensible to look again at the encoding method for training data. An evaluation of the dataset show that it has a high occurrence of 128s and 129s, which is not surprising, as these values encode duration of notes. As the VAE assume that the data points are independent and identically distributed, another way of encoding could be pursued and researched. A simple suggestion is to encode the data in two dimensional vectors instead of one dimensional. In this case, one row can encode note value, and the other row can encode duration. This will reduce the number of 128's and 129's drastically, allowing the rest of the notes to have a higher presence. This is a suggested topic for further works.

Furthermore, more supervision during data processing is recommended. Double checking the song lengths was not performed before the dataset for the MDN was due, after training the VAE. This is a simple and important cross check that could improve the quality of results.

Training the VAE with a higher pressure to the Kullback-Liebler divergence could improve the model's utilisation of the latent vector space, using a method recommended in the paper *Understanding disentangling in β-VAE* [15]. The paper explain that by redefining the KL-term to $\gamma \cdot |D_{KL}(q_\phi(z|x)||p_\theta(z) - C|$, with $\gamma >> 1$, and use simulated annealing on the term C (from zero and up), learning of the VAE is improved. This method force the latent space to first encode the most important features to the KL-divergence to be low, and then slowly adding features that improve reconstruction accuracy. Preliminary experiments with this method show that the VAE produce melodies with better musical structure than the

formal one, with more pleasing melodic patterns. The melodies from these preliminary results also have no need for the scale preprocessing as mentioned in Section 6.4.1.

8.3.2 Real world applications

The use of a sequential generative model has many real world applications. As discussed in this thesis, music is important for humans both culturally and personally. Exploring the combination of music modelling and machine learning can bring forward new musical pieces that, first of all, may be enjoyable for humans. Secondly, machine made music can exemplify how a ML-model learns, and additionally teach us to understand better the underlying structures of music. Brain scientists are investigating whether neural networks can emulate the human brain [114]. With this in mind, research on how a neural network is creative may also give insight into how the human brain is creative. To again quote Peter M. Todd, an early researcher of recurrent networks for music modelling, "The possibilities for further work expanding the capabilities of this approach are virtually limitless". Although this statement should be balanced a bit, as the capabilities may not be limitless, the limit is still far ahead and is nowhere near reached. So this field of science has many more interesting discoveries to come.

Sequence modelling has many applications in the field of language modelling: speech recognition and generation, translation, subject extraction, relation classification and so forth. In general, all sequential data needs sequential modelling, so the real world applications reach as far as the number of sequential events in the world.

In this thesis, the hierarchical nature of sequence modelling is addressed. While this work introduced a new method to music modelling, the findings mirror those of other researchers: music modelling is improved when structural hierarchy is implemented into the model. This is a contribution to research that could reinforce the view that hierarchical methods in sequence modelling is a wise direction to pursue for further research.

Bibliography

[1] Dan Abolafia. *A Recurrent Neural Network Music Generation Tutorial*. Ed. by Magenta. URL: https://magenta.tensorflow.org/2016/06/10/recurrent-neural-network-generation-tutorial.

[2] Memo Akten. *Review of machine / deep learning in an artistic context*. Jan. 2016. URL: https://medium.com/machine-intelligence-report/machine-deep-learning-in-an-artistic-context-441f28774bcc.

[3] Wolfram Alpha, ed. *Bernoulli Distribution*. Mar. 2018. URL: http://mathworld.wolfram.com/BernoulliDistribution.html.

[4] Wolfram Alpha, ed. *Binomial Distribution*. Mar. 2018. URL: http://mathworld.wolfram.com/BinomialDistribution.html.

[5] Nathan Anisko and Eric Anderson. *Average Length of Top 100 Songs on iTunes*. Ed. by StatCrunch. Dec. 2012. URL: https://www.statcrunch.com/5.0/viewreport.php?groupid=948&reportid=28647.

[6] Martin Arjovsky, Soumith Chintala, and Leon Bottou. "Wasserstein GAN". In: (2017). ISSN: 1701.07875. DOI: 10.2507/daaam.scibook.2010.27. URL: http://arxiv.org/abs/1701.07875.

[7] David Berthelot et al. "Understanding and Improving Interpolation in Autoencoders via an Adversarial Regularizer". In: (2018). URL: http://arxiv.org/abs/1807.07543.

[8] Christopher M. Bishop. "Mixture Density Networks". In: *The effects of brief mindfulness intervention on acute pain experience: An examination of individual difference* 1 (1994), pp. 1689–1699. ISSN: 1098-6596. DOI: 10.1017/CBO9781107415324.004.

[9] David M. Blei. *Variational Inference [university lecture]*. Princeton University. 2002. URL: https://www.cs.princeton.edu/courses/archive/fall11/cos597C/lectures/variational-inference-i.pdf.

[10] Thomas Bonte, Nicolas Froment, and Werner Schweer. *Musikalisches Würfelspiel sheet music for Piano download free in PDF or MIDI*. URL: https://musescore.com/user/56747/scores/1742031.

[11] Jonny Brooks-Bartlett. *Probability concepts explained: Bayesian inference for parameter estimation.* Ed. by Towards Data Science. URL: https://towardsdatascience.com/probability-concepts-explained-bayesian-inference-for-parameter-estimation-90e8930e5348.

[12] Jason Brownlee. *A Gentle Introduction to Exploding Gradients in Neural Networks*. Dec. 2017. URL: https://machinelearningmastery.com/exploding-gradients-in-neural-networks/.

[13] Jason Brownlee. *Encoder-Decoder Long Short-Term Memory Networks*. Aug. 2017. URL: https://machinelearningmastery.com/encoder-decoder-long-short-term-memory-networks/.

[14] Jason Brownlee. *Overfitting and Underfitting With Machine Learning Algorithms*. URL: https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/.

[15] Christopher P. Burgess et al. "Understanding disentangling in β-VAE". In: *Computing Research Repository (CoRR)* abs/1804.03599 (2018).

[16] BusinessDictionary. *probabilistic model*. 2018. URL: http://www.businessdictionary.com/definition/probabilistic-model.html.

[17] Li Cai and Yangyong Zhu. "The Challenges of Data Quality and Data Quality Assessment in the Big Data Era". In: *Data Science Journal* 14.0 (May 2015), p. 2. ISSN: 1683-1470. DOI: 10.5334/dsj-2015-002. URL: http://datascience.codata.org/article/10.5334/dsj-2015-002/.

[18] *CarRacing-v0*. URL: https://gym.openai.com/envs/CarRacing-v0/.

[19] Xi Chen et al. "Variational lossy autoencoder". In: (2017), pp. 1–17.

[20] Gaofeng Cheng et al. "An exploration of dropout with LSTMs". In: *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH* 2017-Augus.2 (2017), pp. 1586–1590. ISSN: 19909772. DOI: 10.21437/Interspeech.2017-129.

[21] Francois Chollet. *Building Autoencoders in Keras*. Ed. by The Keras Blog. May 2016. URL: https://blog.keras.io/building-autoencoders-in-keras.html.

[22] Francois Chollet et al. *Keras*. https://github.com/fchollet/keras. 2015. URL: https://github.com/keras-team/keras/blob/master/keras/layers/recurrent.py.

[23] Agatha Christie. *Murder on the Orient Express*. Collins Crime Club, Jan. 1934.

[24] S. Cohen. *Bayesian Analysis in Natural Language Processing*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2016. ISBN: 9781627054218. URL: https://books.google.no/books?id=oBV8DAAAQBAJ.

[25] Sir Arthur Conan Doyle. *The adventures of Sherlock Holmes*. George Newnes, Oct. 1892.

[26] María Laura T Cossio et al. "Algorithmic Composition: Paradigms of Automated Music Generation". In: *Uma ética para quantos?* XXXIII.2 (2012), pp. 81–87.

[27] Arden Dertat. *Applied Deep Learning, Part 3: Autoencoders*. Ed. by Towardssdatascience. Oct. 2017. URL: https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798.

[28] DL4J, ed. *A beginners guide to Recurrent Networks and LSTMs*. 2017. URL: https://deeplearning4j.org/lstm.html.

[29] D. Eck and J. Schmidhuber. "Finding temporal structure in music: Blues improvisation with LSTM recurrent networks". In: *Neural Networks for Signal Processing - Proceedings of the IEEE Workshop* 2002-Janua (2002), pp. 747–756. ISSN: 0780376161. DOI: 10.1109/NNSP.2002.1030094.

[30] Douglas Eck et al. *Magenta Blog*. 2016. URL: https://magenta.tensorflow.org/blog.

[31] Stefano Ermon and Volodymyr Kuleshow. *Variable elimination*. Ed. by Stanford University. 2018. URL: https://ermongroup.github.io/cs228-notes/inference/ve/.

[32] Murhaf Fares and Stephan Oepen. *INF4820-Algorithms for AI and NLP Basic Probability Theory & Language Models [university lecture]*. University of Oslo. 2017. URL: https://www.uio.no/studier/emner/matnat/ifi/nedlagte-emner/INF4820/h17/slides/8_language-models_screen.pdf.

[33] Serena Young Fei-Fei Li Justin Johnson. *Recurrent Neural Networks*. Ed. by Stanford. Aug. 2017. URL: https://www.youtube.com/watch?v=6niqTuYFZLQ&t=1025s.

[34] Marissa Fessenden. *Why Music Is Not a Universal Language [blog post]*. Smithsonian magazine. Feb. 2018. URL: https://www.smithsonianmag.com/smart-news/why-music-not-universal-language-180968245/.

[35] Peter Forret. Ed. by Toolstud.io. URL: https://toolstud.io/music/bpm.php.

[36] Rohith Gandhi. *Introduction to Sequence Models — RNN, Bidirectional RNN, LSTM, GRU*. Ed. by Towards Data Science. URL: https://towardsdatascience.com/introduction-to-sequence-models-rnn-bidirectional-rnn-lstm-gru-73927ec9df15.

[37] A. Gëron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2017. ISBN: 9781491962244. URL: https://books.google.no/books?id=bRpYDgAAQBAJ.

[38] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*. 2010, pp. 249–256. URL: http://www.jmlr.org/proceedings/papers/v9/glorot10a.html.

[39] Ian Goodfellow. *Introduction to GANs, Conference on Neural Information Processing Systems (NIPS) Processing Systems 2016 — Goodfellow, OpenAI*. URL: https://www.youtube.com/watch?v=9JpdAg6uMXs.

[40] Ian J. Goodfellow. *Can generative adversarial networks be used in sequential data in recurrent neural networks, and how effective would they be for sequential data like auto-generating music or human speech?* Quora.com. 2016. URL: https://www.quora.com/What-are-the-pros-and-cons-of-using-generative-adversarial-networks-a-type-of-neural-network-Could-they-be-applied-to-things-like-audio-waveform-via-RNN-Why-or-why-not.

[41] Ian J. Goodfellow et al. "Generative Adversarial Networks". In: (2014), pp. 1–9. ISSN: 10495258. DOI: 10.1001/jamainternmed.2016.8245. eprint: 1406.2661. URL: http://arxiv.org/abs/1406.2661.

[42] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org.

[43] Alex Graves. "Generating Sequences With Recurrent Neural Networks". In: *CoRR* abs/1308.0850 (2013). arXiv: 1308.0850. URL: http://arxiv.org/abs/1308.0850.

[44] Klaus Greff et al. "LSTM: A Search Space Odyssey". In: *IEEE Transactions on Neural Networks and Learning Systems* 28.10 (2017), pp. 2222–2232. ISSN: 21622388. DOI: 10.1109/TNNLS.2016.2582924.

[45] Leonardo Guizzetti. *Markov Models, or the Future is Now – Leonardo's blog [blog post]*. URL: http://guizzetti.ca/blogs/lenny/2012/04/markov-models-or-the-future-is-now/.

[46] David Ha and Douglas Eck. "A Neural Representation of Sketch Drawings". In: (2017). URL: http://arxiv.org/abs/1704.03477.

[47] David Ha and Jürgen Schmidhuber. "World Models". In: *Computing Research Repository (CoRR)* abs/1803.10122 (2018). arXiv: 1803.10122. URL: http://arxiv.org/abs/1803.10122.

[48] David Ha and Jürgen Schmidhuber. *World Models [blog post]*. 2018. URL: https://worldmodels.github.io/.

[49] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. "The elements of statistical learning: data mining, inference, and prediction, 2nd Edition". In: Springer series in statistics. Springer, 2009. Chap. 2.4.

[50] Curtis Hawthorne et al. "ENABLING FACTORIZED PIANO MUSIC MODELING". In: (2018), pp. 1–12.

[51] Stephen A. Hedges. "Dice Music in the Eighteenth Century". In: *Music & Letters, Vol. 59, No. 2*. Oxford University Press Stable, 1978, pp. 180–187.

[52] Piet Hein and Jens Aarup. *Grooks 1*. Ed. by Borgen. 1940.

[53] Robert M. Hierons. "Machine Learning, by Tom M. Mitchell, McGraw-Hill, 1997 (Book Review)". In: *Softw. Test., Verif. Reliab.* 9.3 (1999), pp. 191–193.

[54] Sepp Hochreiter and Jürgen Schmidhuber. "LONG SHORT-TERM MEMORY". In: *Neural Computation* (1997). ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. arXiv: 1206.2944.

[55] lejlot (https://stats.stackexchange.com/users/28903/lejlot). *Objective function, cost function, loss function: are they the same thing? [forum discussion]*. StackExchange: Cross Validated. URL:https://stats.stackexchange.com/q/179027 (version: 2017-09-04). Oct. 2015. URL: https://stats.stackexchange.com/q/179027.

[56] Bert Huang. *17 Probabilistic Graphical Models and Bayesian Networks*. Ed. by Virginia Tech. 2015. URL: https://www.youtube.com/watch?v=zCWRTKnOYYg&t=1335s.

[57] Mark Inlow. *The IID Assumption [university lecture notes]*. Rose-Hulman Institue of Technology. May 2015. URL: https://www.rose-hulman.edu/class/ma/inlow/Math223/iid_f13.pdf.

[58] Daniel Jacobson. *The Elements of Music*. Ed. by Western Michigan University. 2014. URL: https://wmich.edu/mus-gened/mus170/RockElements.pdf.

[59] Eric Jang. *A Beginner's Guide to Variational Methods: Mean-Field Approximation*. Aug. 2016. URL: https://blog.evjang.com/2016/08/variational-bayes.html.

[60] Andrew Hsu John McGonagle George Shaikouski. *Backpropagation*. URL: https://brilliant.org/wiki/backpropagation/.

[61] Jeremy https://www.jeremyjordan.me/about/ Jordan. *Variational autoencoders.* URL: https://www.jeremyjordan.me/variational-autoencoders/.

[62] Michael I. Jordan. "Serial Order: A Parallel Distributed Processing Approach". In: 1997, pp. 471–495. DOI: 10.1016/S0166-4115(97)80111-2. URL: http://linkinghub.elsevier.com/retrieve/pii/S0166411597801112.

[63] Andrej Karpathy. *Setting up the data and the model [university lecture notes]*. Stanford University. 2017. URL: http://cs231n.github.io/neural-networks-2/#init.

[64] Andrej Karpathy et al. *Generative Models*. Ed. by openAI. June 2016. URL: https://blog.openai.com/generative-models/.

[65] Karen Kersting. *What exactly is creativity? [blog post]*. American Psychological Association. Nov. 2003. URL: https://www.apa.org/monitor/nov03/creativity.aspx.

[66] Taeksoo Kim et al. "Learning to Discover Cross-Domain Relations with Generative Adversarial Networks". In: *Computing Research Repository (CoRR)* abs/1703.05192 (2017). arXiv: 1703.05192. URL: http://arxiv.org/abs/1703.05192.

[67] Augustinius Kristiadi. *KL Divergence: Forward vs Reverse?* 2018. URL: https://wiseodd.github.io/techblog/2016/12/21/forward-reverse-kl/.

[68] Carol L. Krumhansl and Lola L. Cuddy. "A Theory of Tonal Hierarchies in Music". In: *Music Perception*. Ed. by Mari Riess Jones, Richard R. Fay, and Arthur N. Popper. New York, NY: Springer New York, 2010, pp. 51–87. DOI: 10.1007/978-1-4419-6114-3_3. URL: https://doi.org/10.1007/978-1-4419-6114-3_3.

[69] Vladimir Kuleshov and Stefano Ermon. *Variational inference*. Ed. by Stanford University. URL: https://ermongroup.github.io/cs228-notes/inference/variational/.

[70] Tony Kushner. *Angels in America*. Nick Hern Books, 2017.

[71] Yann LeCun et al. "Efficient BackProp". In: *Neural Networks: Tricks of the Trade - Second Edition*. 2012, pp. 9–48. DOI: 10.1007/978-3-642-35289-8_3. URL: https://doi.org/10.1007/978-3-642-35289-8_3.

[72] Fei-Fei Li, Justin Johnson, and Serena Yeung. *Generative Models*. Ed. by Stanford University. May 2017. URL: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf.

[73] Fei-Fei Li, Justin Johnson, and Serena Yeung. *Lecture 10: Recurrent Neural Networks*. May 2017. URL: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf.

[74] G. Loevaas Gunnar. *Statistikk for universitet og hoegskoler*. Universitetsforlaget, 2013.

[75] David Ludden. *Is Music a Universal Language?* URL: https://www.psychologytoday.com/us/blog/talking-apes/201507/is-music-universal-language.

[76] Stephen Marsland. *Machine Learning - An Algorithmic Perspective*. Chapman and Hall / CRC machine learning and pattern recognition series. CRC Press, 2009. ISBN: 978-1-4200-6718-7. URL: http://www.crcpress.com/product/isbn/9781420067187.

[77] Charles Martin. *cmpercussion/keras-mdn-layer*. Ed. by GitHub. 2018. URL: https://github.com/cpmpercussion/keras-mdn-layer/blob/master/notebooks/MDN-RNN-time-distributed-MDN-training.ipynb.

[78] Charles P. Martin. "Predictive Musical Interaction with MDRNNs". In: *NeurIPS 2018 Workshop on Machine Learning for Creativity and Design*. Montréal, Canada, Dec. 2018. URL: https://nips2018creativity.github.io/doc/Predictive_Musical_Interaction_with_MDRNNs.pdf.

[79] Charles P. Martin, Kai Olav Ellefsen, and Jim Torresen. "Deep Models for Ensemble Touch-Screen Improvisation". In: *Proceedings of the 12th International Audio Mostly Conference on Augmented and Participatory Sound and Music Experiences*. AM '17. Aug. 2017. DOI: 10.1145/3123514.3123556.

[80] Charles P. Martin, Kai Olav Ellefsen, and Jim Tørresen. "Deep Predictive Models in Interactive Music". In: *Computing Research Repository (CoRR)* abs/1801.10492 (2018). arXiv: 1801.10492. URL: http://arxiv.org/abs/1801.10492.

[81] Charles P. Martin and Jim Tørresen. "RoboJam: A Musical Mixture Density Network for Collaborative Touchscreen Interaction". In: *Computing Research Repository (CoRR)* abs/1711.10746 (2017). arXiv: 1711.10746. URL: http://arxiv.org/abs/1711.10746.

[82] Charles Martin, Henry Gardner, and Ben Swift. "Tracking Ensemble Performance on Touch-Screens with Gesture Classification and Transition Matrices". In: *Proceedings of the International Conference on New Interfaces for Musical Expression*. Ed. by Edgar Berdahl and Jesse Allison. Baton Rouge, Louisiana, USA: Louisiana State University, May 2015, pp. 359–364. DOI: 10.5281/zenodo.1179130. URL: http://www.nime.org/proceedings/2015/nime2015_242.pdf.

[83] Charles Martin et al. "Intelligent Agents and Networked Buttons Improve Free-Improvised Ensemble Music-Making on Touch-Screens". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '16. New York, NY, USA: ACM, May 2016, pp. 2295–2306. DOI: 10.1145/2858036.2858269.

[84] Diane Martindale. "One Face, One Neuron". In: *Scientific American* 293.4 (Oct. 2005), pp. 22–24. ISSN: 0036-8733. DOI: 10.1038/scientificamerican1005-22. URL: http://www.nature.com/doifinder/10.1038/scientificamerican1005-22.

[85] Kyle McDonald. *Neural Nets for Generating Music*. Ed. by Medium. Aug. 2017. URL: https://medium.com/artists-and-machine-intelligence/neural-nets-for-generating-music-f46dffac21c0.

[86] Merriam-Webster, ed. *Entropy*. URL: https://www.merriam-webster.com/dictionary/entropy.

[87] Merriam-Webster, ed. *Inference*. URL: https://www.merriam-webster.com/dictionary/inference.

[88] Merriam-Webster, ed. *Recurrent*. Feb. 2018. URL: https://www.merriam-webster.com/dictionary/recurrent.

[89] *MIDI note numbers and center frequencies*. URL: http://www.inspiredacoustics.com/en/MIDI_note_numbers_and_center_frequencies.

[90] Lester James V. Miranda. *Understanding softmax and the negative log-likelihood*. URL: https://ljvmiranda921.github.io/notebook/2017/08/13/softmax-and-the-negative-log-likelihood/.

[91] Shakir Mohamed. *Machine Learning Trick of the Day(4): Reparametrizing Tricks*. Ed. by The Spectator. Oct. 2015. URL: http://blog.shakirm.com/2015/10/machine-learning-trick-of-the-day-4-reparameterisation-tricks/.

[92] Shakir Mohamed and Balaji Lakshminarayanan. "Learning in Implicit Generative Models". In: *Computing Research Repository (CoRR)* abs/1610.03483 (2016).

[93] Andrew Ng. Ed. by Stanford. Apr. 2013. URL: http://online.stanford.edu/course/machine-learning.

[94] Michael Nielsen. *Improving the way neural networks learn*. Dec. 2017. URL: http://neuralnetworksanddeeplearning.com/chap3.html.

[95] Christopher Olah. *Understanding LSTM Networks*. Aug. 2015. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[96] Aäron van den Oord, Sander Dieleman, and Heiga Zen. June 2017. URL: WaveNet:%20A%20Generative%20Model%20for%20Raw%20Audio.

[97] Aaron van den Oord et al. "WaveNet: A Generative Model for Raw Audio". In: (2016), pp. 1–15. ISSN: 0899-7667. DOI: 10.1109/ICASSP.2009.4960364. URL: http://arxiv.org/abs/1609.03499.

[98] Guim Perarnau et al. "Invertible Conditional GANs for image editing". In: *Computing Research Repository (CoRR)* abs/1611.06355 (2016). arXiv: 1611.06355. URL: http://arxiv.org/abs/1611.06355.

[99] Howard Pollack. *George Gershwin. His life and work.* University of California Press, Jan. 2007. ISBN: 9780520248649.

[100] William H. Press et al. *Numerical Recipes in C: The Art of Scientific Computing*. New York, NY, USA: Cambridge University Press, 1988.

[101] Princeton, ed. *Nash equilibrium*. URL: http://wordnetweb.princeton.edu/perl/webwn?s=nash%5C%20%5C%20equilibrium.

[102] Samuel R. Bowman et al. "Generating Sentences from a Continuous Space". In: *Computing Research Repository (CoRR)* abs/1511.06349 (2015). arXiv: 1511.06349. URL: http://arxiv.org/abs/1511.06349.

[103] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks". In: (2015), pp. 1–16. ISSN: 0004-6361. DOI: 10.1051/0004-6361/201527329. eprint: 1511.06434. URL: http://arxiv.org/abs/1511.06434.

[104] Adam Roberts et al. "A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music". In: *Computing Research Repository (CoRR)* abs/1803.05428 (2018). arXiv: 1803.05428. URL: http://arxiv.org/abs/1803.05428.

[105] Adam Roberts et al. *mm.Player*. URL: https://tensorflow.github.io/magenta-js/music/demos/player.html.

[106] F Rosenblatt. "The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain". In: *Psychological Review* (1958), pp. 65–386.

[107] Margaret Rouse. *Markov Model*. Ed. by WhatIs.com. July 2017. URL: https://whatis.techtarget.com/definition/Markov-model.

[108] David E. Rumelhart, James L. McClelland, and San Diego. PDP Research Group. University of California. *Parallel distributed processing : explorations in the microstructure of cognition*. MIT Press, 1986. ISBN: 9780262680530.

[109] Ruslan Salakhutdinov. "Approximate Inference". In: *Machine Learning* (2009), pp. 467–477. ISSN: 00223530. DOI: 10.1093/petrology/egs049.

[110] Tim Salimans et al. "Improved Techniques for Training GANs". In: *Computing Research Repository (CoRR)* abs/1606.03498 (2016). arXiv: 1606.03498. URL: http://arxiv.org/abs/1606.03498.

[111] Valorie N Salimpoor et al. "Anatomically distinct dopamine release during anticipation and experience of peak emotion to music". In: *Nature Neuroscience* 14.2 (Feb. 2011), pp. 257–262. ISSN: 1097-6256. DOI: 10.1038/nn.2726. URL: http://www.nature.com/articles/nn.2726.

[112] Jürgen Schmidhuber. "Deep Learning in Neural Networks: An Overview". In: *Computing Research Repository (CoRR)* abs/1404.7828 (2014). arXiv: 1404.7828. URL: http://arxiv.org/abs/1404.7828.

[113] Scikit-learn, ed. *Underfitting vs. Overfitting*. URL: http://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html.

[114] Kelly Servick. "Brain scientists dive into deep neural networks." In: *Science (New York, N.Y.)* 361.6408 (Sept. 2018), p. 1177. ISSN: 1095-9203. DOI: 10.1126/science.361.6408.1177. URL: http://www.ncbi.nlm.nih.gov/pubmed/30237335.

[115] Irhum Shafkat. *Intuitively Understanding Variational Autoencoders*. URL: https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf.

[116] Tarang Shan. *About Train, Validation and Test Sets in Machine Learning*. Dec. 2017. URL: https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7.

[117] Claude E Shannon. "A mathematical theory of communication". In: *The Bell System Technical Journal* 27.July 1928 (1948), pp. 379–423. ISSN: 07246811. DOI: 10.1145/584091.584093. URL: http://cm.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf.

[118] Usman Shuja. *Is Deep Learning The Big Bang Moment For AI?* Ed. by Forbes. Mar. 2018. URL: https://www.forbes.com/sites/forbestechcouncil/2018/03/20/is-deep-learning-the-big-bang-moment-for-ai/#1c49a82a4836.

[119] Ole-Johan Skrede. *Generative Adversarial Networks, INF5860 - Machine Learning for Image Analysis*. May 2018. URL: https://www.uio.no/studier/emner/matnat/ifi/INF5860/v18/undervisningsmateriale/lectures/slides_inf5860_s18_week14.pdf.

[120] Stanford, ed. *Softmax Regression*. URL: http://ufldl.stanford.edu/tutorial/supervised/SoftmaxRegression/.

[121] Stanford University. and Center for the Study of Language and Information (U.S.) *Connectionism*. Stanford University, 1997. URL: https://plato.stanford.edu/entries/connectionism/#ShaConBetConCla.

[122] Bob L. Sturm et al. "Machine learning research that matters for music creation: A case study". In: *Journal of New Music Research* 0.0 (2018), pp. 1–20. DOI: 10.1080/09298215.2018.1515233. eprint: https://doi.org/10.1080/09298215.2018.1515233. URL: https://doi.org/10.1080/09298215.2018.1515233.

[123] *The Largest MIDI Collection on the Internet, collected and sorted diligently by yours truly.* URL: https://www.reddit.com/r/WeAreTheMusicMakers/comments/3ajwe4/the_largest_midi_collection_on_the_internet/.

[124] Lucas Theis, Aäron van den Oord, and Matthias Bethge. "A note on the evaluation of generative models". In: (2015), pp. 1–10. ISSN: 1477-1535. URL: http://arxiv.org/abs/1511.01844.

[125] Peter M Todd. "A Connectionist Approach To Algorithmic Composition". In: *Computer Music Journal* 13.4 (1989), pp. 27–43. ISSN: 01489267. DOI: 10.2307/3679551.

[126] Avinash Sharma V. *Understanding Activation Functions In Neural Networks*. Ed. by Medium. Mar. 2017. URL: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0.

[127] Elliot Waite. *Generating Long-Term Structure in Songs and Stories [blog post]*. Google AI. 2016. URL: https://magenta.tensorflow.org/2016/07/15/lookback-rnn-attention-rnn.

[128] Yair Weiss and Michael I. Jordan. "Probabilistic Inference in Graphical Models". In: *Stat Sci (Special Issue on Bayesian Stat* 21.3 (2002), p. 421. DOI: 10.1016/0010-0285(84)90015-X.

[129] Eric W. Weisstein. *Homotopy*. Ed. by MathWorld - A Wolfram Web Resource. URL: http://mathworld.wolfram.com/Homotopy.html.

[130] Eric W. Weisstein. *NP-Hard Problem*. URL: http://mathworld.wolfram.com/NP-HardProblem.html.

[131] Eric W. Weisstein. *Sigmoid Function*. Ed. by MathWorld - A Wolfram Web Resource. URL: http://mathworld.wolfram.com/SigmoidFunction.html.

[132] Max Welling. "Auto-Encoding Variational Bayes". In: *International Conference on Machine Learning* (May 2014), pp. 1–14.

[133] Lillian Weng. *From GAN to WGAN*. Ed. by OpenAI. May 2017. URL: https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html#what-is-the-optimal-value-for-d.

[134] Stuart Wilson. *Reparameterization*. Ed. by created by Eric W. Weisstein MathWorld - A Wolfram Web Resource. URL: mathworld.wolfram.com/Reparameterization.html.

[135] Marek Wydmuch et al. *Institute of Computing Science, Poznań University of Technology, Poland*. URL: http://vizdoom.cs.put.edu.pl/.

[136] Li-Chia Yang, Szu-Yu Chou, and Yi-Hsuan Yang. "MidiNet: A Convolutional Generative Adversarial Network for Symbolic-domain Music Generation". In: *The International Society of Music Information Retrieval (ISMIR)* (2017), pp. 324–331. URL: http://arxiv.org/abs/1703.10847.

[137] Xitong Yang. *Understanding the Variational Lower Bound [blog post]*. https://xyang35.github.io/2017/04/14/variational-lower-bound/. Apr. 2017.

[138] Jane Yolen. *Briar Rose*. Ed. by Tor Books. 1992. ISBN: ISBN 0-8125-5862-6.

[139] Rui Zhao et al. "Learning to Monitor Machine Health with Convolutional Bi-Directional LSTM Networks". In: *Sensors* 17 (Jan. 2017), p. 273. DOI: 10.3390/s17020273.