Bachelor-Thesis

in

Medieninformatik

Umsetzung eines Deep Learning Systems zur Generierung von Videospielmusik

Referent:	Prof. Dr. Ruxandra Lasowski
Korreferent:	Prof. Dr. Norbert Schnell
Vorgelegt am:	28.02.2021
Vorgelegt von:	Philipp Oeschger
	257184
	Bregstraße, 90
	78120, Furtwangen
	philipp.oeschger@hs-furtwangen.de

Abstract

Die Videospielebranche stellt einen wachsenden Markt dar. Musik ist seit jeher ein essenzieller als auch kostspieliger Aspekt von Videospielen. Bei der Betrachtung aktueller Entwicklungen im Bereich Deep Learning erweisen sich künstliche Intelligenzen als nützlich zur Generierung neuer Bild-, Audio- und Musikdaten.

Im Rahmen dieser Bachelorarbeit wurde ein Deep Learning System entwickelt, welches dazu in der Lage ist, musikalische Inhalte für den Einsatz in Videospielen zu generieren. Für diesen Zweck wurden ausgewählte Modelle der automatisierten Musikproduktion betrachtet und analysiert. Anschließend wurde ein System konzipiert, umgesetzt und mit verschiedenen Datensätzen trainiert. Resultat waren Deep Learning Modelle, welche zur Generierung von Musikstücken eingesetzt werden können. Abschließend wurde die Qualität der Testergebnisse anhand ausgewählter objektiver Metriken bewertet.

The video game industry is a growing market. Music is since the beginning an integral and expensive aspect of videogames. When you examine current developments in the area of deep learning the usefulness of artificial intelligence in generating new picture, audio, and music data is examinable.

As part of this Thesis, a deep learning system has been developed which is capable of generating music for usage in video games. For this purpose, selected models for automated music production were considered and analyzed. Subsequently, a system was designed, implemented, and trained with different datasets. This resulted with deep learning models which can be used for generating pieces of music. In conclusion, the quality of samples was evaluated through designated objective metrics.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ADP	Average Drum Pattern
AEB	Average Empty Bars
ANC	Average Note Count
API	Application Programming Interface
APP	Average Polyphonicity
ATD	Average Tonal Distance
AUPC	Average Used Pitch Classes
CNN	Convolutional Neural Networks
D	Diskriminator
DAW	Digital Audio Workstation
DGD1	Demographic Game Design Model
DL	Deep Learning
DNN	Deep Neuronal Networks
DP	Drum Pattern
EB	Empty Bars
ELBO	Evidence Lower Bound
FFN	Feedforward Netzwerk
G	Generator
GAN	Generative Adversarial Network
KI	Künstliche Intelligenz
KL	Kullback-Leibler
KNN	Künstliches Neuronales Netz
LAPGAN	Laplacian Generative Adversarial Netzwork
LMD	Lakh MIDI Dataset
LPD	Lakh Pianoroll Dataset
LSTM	Long Short Term Memory
MIDI	Musical Instrument Digital Interface
MSD	Million Song Dataset
NN	Neuronales Netz
PP	Polyphonicity

QN	Qualified Notes
RNN	Recurrent Neural Network
TD	Tonal Distance
UPC	Used Pitch Classes
VAE	Variational Autoencoder

1 Einleitung

1.1 Ausgangssituation

Schon seit den frühen 80’er Jahren ist Musik ein wichtiger Bestandteil von Videospielen und seit jeher nicht mehr aus dem Unterhaltungsmedium wegzudenken. Eine der wichtigsten Eigenschaften von Videospielen ist die Immersion, die Fähigkeit so auf eine Videospielwelt fokussiert zu sein, das andere Vorgänge in der realen Welt mental ausgeblendet werden. (vgl. Sanders und Cairns 2010, S. 160–161)

Für ein neues Videospiel werden verschiedene Musikstücke, meist eine Gruppe an Fachleuten, Fachwissen und eine Menge Zeit benötigt. Ein Aufwand, den sich kleine unabhängige Entwicklerteams mit beschränkten Ressourcen nur schwer leisten können.

In der Zwischenzeit kommen künstliche Intelligenzen in Anwendungsfällen wie z. B. der Bild- und Sprachverarbeitung bereits erfolgreich zum Einsatz. Neben den bereits genannten Bereichen wird in der Musikproduktion mit der Verwendung neuronaler Netze experimentiert. Systeme wie der „BachBot“ (Liang et al. 2017), eine künstliche Intelligenz (KI) welche dazu trainiert wurde, Musik im Stil des Komponisten Johann Sebastian Bach zu komponieren zeigt, dass die Möglichkeit besteht, mithilfe neuronaler Netze originäre Musik zu generieren. Anknüpfend an diese Grundlagen stellte sich die Frage, ob aktuelle Entdeckungen in der Forschung der künstlichen Intelligenzen sich auf den Videospielbereich anwenden lassen.

1.2 Zielsetzung

Die vorliegende Arbeit beschäftigt sich mit der Generierung originären Musikstücken für die Verwendung in Videospielen. Ziel ist auf den Grundlagen aktueller Entwicklungen im Teilbereich des Generativen Deep Learnings ein neuronales Netz umzusetzen, welches ein Wahrscheinlichkeitsmodell erlernt und auf Fundament dessen neuen Musikstücke generieren kann.

Die Arbeit dient als Grundlage dienen für die Automatisierung des Musikproduktionsprozesses in der Videospielbranche und für diesen Bereich neu Erkenntnisse aufdecken.

1.3 Vorgehensweise

Im Bereich des Deep Learnings, ein wachsender Teilbereich des Machine Learning, gibt es verschiedene generative Ansätze, wie z. B. das an Popularität gewinnende Generative Adversarial Netzwork (GAN) (Goodfellow et al. 2014) oder der nicht weniger vielversprechende Variational Autoencoder (VAE) (Kingma und Welling 2013). Beide Modelle weißen Stärken und Schwächen auf, die sich je nach Anwendungsfall vermehrt oder vermindert auf die Lösung der Problemstellung auswirken.

Zunächst soll analysiert werden, welche Architektur sich für den gegebenen Anwendungsfall besser eignet. Hierfür wurden Beispiele für Modelle der Musikproduktion zur genaueren Betrachtung herangezogen. Auf Basis dieser Analyse soll eine geeignete Architektur gewählt werden.

Abhängig vom Aufbau der Architektur werden die Datensätze in einem speziellen Format vorausgesetzt. Es werden Datensätze zusammengestellt, welche den Themenbereich der Videospiele verkörpern und in ein für das neuronale Netz geeignetes Format transformieren.

Nach dem Zusammenstellen der Datensätze werden passende Hyperparameter bestimmt und mit diesen das Neuronale Netz trainiert.

Abschließend befasst sich die Arbeit mit der qualitativen Auswertung der Resultate. Hierzu wurde sich mit der Frage auseinandergesetzt, wie sich die Qualität der Ausgangsdaten metrisch ermitteln lässt und inwiefern diese Metriken aussagekräftig bezogen auf die Qualität der musikalischen Inhalte sind. Zur Beantwortung dieser Frage wurden Metriken bestimmt und damit die Stichproben des Netzwerks ausgewertet. Die berechneten Metriken wurden abschließend interpretiert und evaluiert.

1.4 Aufbau der Arbeit

Zuerst wird sich das zweite Kapitel mit den Grundlagen der Arbeit befassen. Hierbei wird auf die Grundlagen des Deep Learnings eingegangen.

Danach behandelt das dritte Kapitel im Kontext der Musikproduktion die Modelle MuseGAN (Dong et al. 2018) als Vertreter des GAN und MusicVAE (Roberts et al. 2018) als Vertreter des VAE. Die beiden Modelle werden analysiert und deren Nachteile und Vorteile gegenübergestellt.

Das vierte Kapitel beschreibt die genauen Details der Umsetzung. Zum Inhalt des Kapitels gehören die Beschaffenheit der Datensätze, der detaillierte Aufbau des neuronalen Netzes, der Prozess der Hyperparameterbestimmung, das Training des Netzwerks und die Realisierung der objektiven Metriken.

Im vorletzten Kapitel werden die Ergebnisse der Auswertung vorgestellt.

Ein Fazit und ein Ausblick auf mögliche Folgeprojekte schließen die Arbeit ab.

2 Grundlagen

In diesem Kapitel werden die Grundlagen zum Verständnis der in dieser Arbeit behandelten Themen nahegelegt. Zunächst wird der Begriff des generative Deep Learnings (DL) erklärt und dessen Modelle vorgestellt.

Anschließend werden die Datentypen präsentiert, auf die innerhalb der Arbeit eingegangen wird.

Als Abschluss des Kapitels wird die Musikkomposition in der Videospielbranche genauer betrachtet.

2.1 Generatives Deep Learning

2.1.1 Generative Modellierung

Grundlegendes Ziel der generativen Modellierung ist es, einen Datensatz im Rahmen eines Wahrscheinlichkeitsmodells zu erzeugen. Beispielweise ist ein Datensatz mit Bildern gegeben. Es soll ein Modell erstellt werden, welches die Fähigkeit besitzt, selbst Bilder zu generieren. Das Wissen zur Generierung neuer Bilder soll durch das Erlernen des Datensatzes erlangt werden. Diese Aufgabe lässt sich mit generativen Modellen lösen (vgl. *Abbildung 1*). Der Datensatz wird in diesem Zusammenhang Trainingsdatensatz genannt.

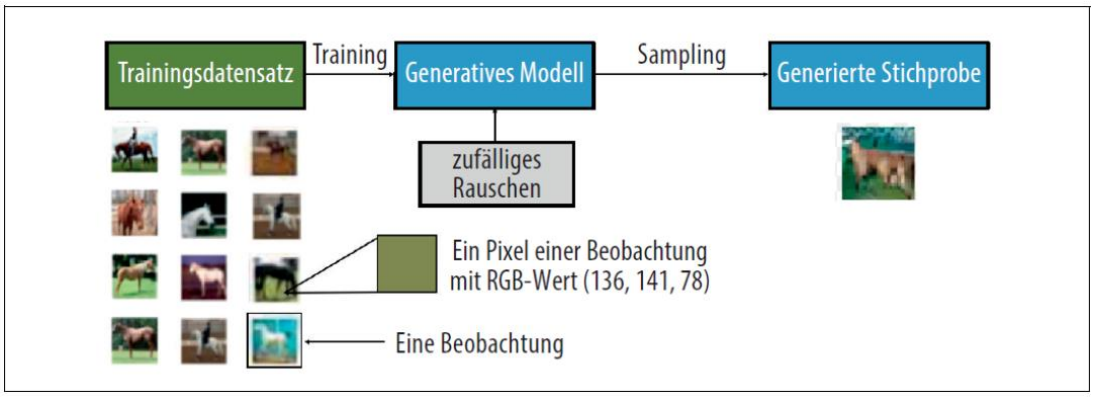


Abbildung 1: Veranschaulichung Generative Modellierung (Quelle: Foster 2020)

Die Schwierigkeit liegt bei der Erkennung eines probabilistischen und keines deterministischen Modells. Das Modell sollte stochastische Aspekte mit einfließen lassen, durch die das Erstellen der Daten beeinflusst wird.

Die Idee ist, dass die Trainingsdaten Teil einer Wahrscheinlichkeitsverteilung sind. Ziel ist es, diese Verteilung so genau wie möglich einzugrenzen, um daraus neue Bilder zu generieren, die möglicherweise auch Teil des Trainingsdatensatzes sein könnten.

Ein generatives Modell lässt sich folgendermaßen beschreiben:

- Es existiert ein Beobachtungssatz X.
- Es wird davon ausgegangen, dass die Beobachtung auf einer unbekannten Verteilung p_{data} basiert.
- Ziel ist es, ein generatives Modell p_{model} zu erstellen welches versucht p_{data} nachzubilden. Wenn dies geglückt ist, sollte p_{model} dazu in der Lage sein, Beobachtungen zu erstellen die den Beobachtungen von p_{data} ähneln.
- Das Modell ist erfolgreich, wenn einerseits Beispiele den Beobachtungen aus p_{data} ähnlich sehen und andererseits Beispiele sich ausreichend von den Beobachtungen von p_{data} unterscheiden und nicht als Plagiat dieser Beobachtungen gesehen werden können.

Generative Modelle bauen auf den Prinzipien des Repräsentationslernens (engl. Representation Learning) auf, welches aussagt, dass jede Eigenschaft eines Datensatzes in einem kleinen latenten Dimensions-Raum dargestellt werden kann. In der Theorie verweist jeder Punkt des latenten Raums auf eine eigene Darstellung eines Elements auf Basis des Datensatzes. (vgl. zu diesem Kapitel Foster 2020, Teil 1 - Kapitel 1)

2.1.2 Deep Learning

Deep Learning beschreibt den Vorgang, wobei ein System Erfahrungen lernt, in dem Konzepte hierarchisch aufgebaut werden. Durch das hierarchische Aufspalten der Konzepte in kleinere Unterkonzepte ist es für ein System möglich komplexe Aufgaben zu erlernen. Wenn dieses Modell grafisch dar-

gestellt werden würde, sähe man viele Schichten, welche sich in die Tiefe ausbreiten. Diese Methodik bezeichnet man Deep Learning. (vgl. zu diesem Abschnitt Goodfellow 2018, S. 2)

In den meisten Fällen handelt es sich bei Deep Learning Modellen um künstliche neuronale Netze (KNNs) mit mehreren hintereinander platzierten verborgenen Schichten. Andere mehrschichtige Systeme mit Ziel der Erlernung komplexer Probleme fallen ebenfalls in die Kategorie des Deep Learnings. Beispiele hierfür währen ein Deep-Believe-Netzwerk oder eine Deep-Boltzmann-Maschine. (vgl. zu diesem Abschnitt Foster 2020, Teil 1 - Kapitel 2)

2.1.3 Tiefe Feedforward Netze

Die meisten heutig verwendeten künstlichen neuronalen Netzwerke sind tiefe Feedforward Netze (FFNs). Das Ziel eines solchen Netzes ist eine Funktion f^* nächstmöglich anzunähern. Die Bezeichnung Feedforward-Netzwerk kommt daher, weil Informationen in einer klaren Vorwärtsbewegung bearbeitet werden. X ist Eingang einer Funktion f, die den Ausgang Y zurückgibt. Innerhalb des Systems ist keine Rückkopplung vorgesehen. (vgl. zu diesem Abschnitt Goodfellow 2018, S. 185)

Ein klassisches neuronales Netz setzt sich aus mehreren Funktionen bzw. Schichten zusammen (vgl. *Abbildung* 2). In einem neuronalen Netz bestehen die Schichten aus mehreren Recheneinheiten, welche Neuronen genannt werden. Die erste Schicht besteht aus Eingangsneuronen und wird die Eingangsschicht genannt. (vgl. zu diesem Abschnitt Goodfellow 2018, S. 185–186; Gupta 2017)

Betrachtet man das Beispiel der Klassifizierung, hat das FFN ein klares Ziel der richtigen Einteilung der Eingangswerte. Wie dieses Ziel erreicht werden soll, ist nicht vorgegeben. Da die Vorgänge der Zwischenschichten in den Ausgaben des Netzwerks nicht ersichtlich sind, werden die Zwischenschichten verdeckte Schichten (engl. hidden layers) genannt. (vgl. zu diesem Abschnitt Goodfellow 2018, S. 185–186)

In einer Kette von mehreren aneinander gereihten Funktionen eines Netzes, ist das letzte Kettenglied die Ausgangsschicht (vgl. Goodfellow 2018, S. 185–186).

Eine verdeckte Schicht benötigt eine Aktivierungsfunktion, die Funktion, mit der die Einheiten der Schicht ihre Werte berechnen. (vgl. zu diesem Abschnitt Goodfellow 2018, S. 185–188)

Die Neuronen der Schichten sind durch Gewichte miteinander verbunden, durch die sich die Beziehung, zu denen die Neuronen zueinander stehen, beeinflussen lassen (vgl. Goodfellow 2018, S. 255). Zusätzlich besitzen diese Neuronen eine Verzerrung (Bias) welche das Ergebnis der Berechnung in eine bestimmte Richtung lenkt (vgl. Goodfellow 2018, S. 191–192).

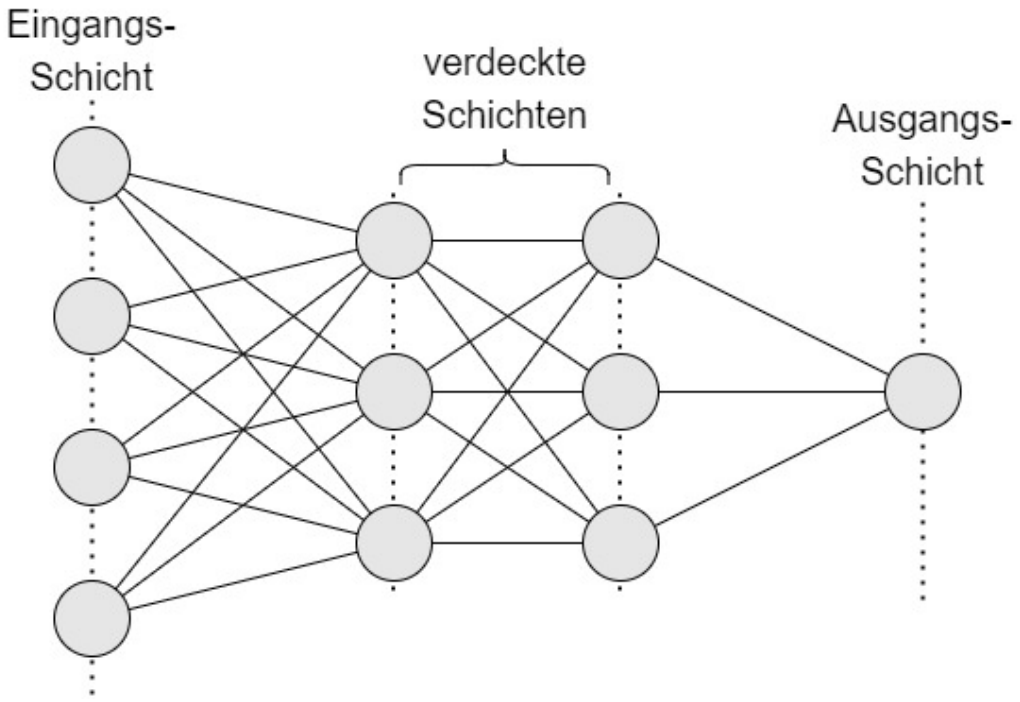


Abbildung 2: Beispiel eines neuronalen Feedforward Netzes (vgl. Gupta 2017)

Ziel eines solchen neuronalen Netzwerks ist es, für jede Schicht die optimalen Gewichte zu finden. Diesen Vorgang der optimalen Gewichtsermittlung bezeichnet man als Training. (vgl. Foster 2020, Teil 1 - Kapitel 2).

Beim Training werden Stapel (engl. Batches) von Daten durch das Netzwerk geschickt. Um eine Verbesserung des Modells zu ermöglichen, muss zunächst die Ausgabe mit dem Wahrheitswert der Daten verglichen werden. Der Fehler des Netzwerks wird zurück durch das Netzwerk propagiert und die Gradienten berechnet. Dieser Prozess nennt sich Backpropagation. Mithilfe dieser Gradienten kann ein Gradientenabstiegsverfahren angewendet werden, wodurch das Netzwerk lernt. Mit der Zeit spezialisieren sich die Neuronen auf gewisse Merkmale der Eingangswerte, wodurch das Netzwerk bessere Vorhersagen treffen kann. (vgl. zu diesem Abschnitt Foster 2020, Teil 1 - Kapitel 2; Goodfellow 2018, S. 225)

Für das Berechnen des Fehlers wird eine Kostenfunktion eingesetzt. Als Folge hat das Netzwerk ein klares Ziel, was die Anpassung von Gewichten und Neigungen von Neuronen erheblich erleichtert, nämlich die Reduktion der Kostenfunktion (vgl. Gupta 2017).

Das Gradientenabstiegsverfahren passt Gewichte und Verzerrungen an. Aufgrund der Non-Linearität vieler neuronalen Netze sind deren Kostenfunktionen nicht in der Lage zu konvergieren. Für diesen Zweck werden Gradienten-basierte Optimierer verwendet, um die Kostenfunktion zu einem möglichst niedrigen Wert zu führen. (vgl. zu diesem Abschnitt Gupta 2017)

2.1.4 Convolutional Neural Networks (CNN)

CNN sind Neuronale Netzwerke, welche vor allem nützlich im Bereich der Bildverarbeitung sind (vgl. Saha 2018). Sie sind Netze, die für mindestens eine Schicht die mathematische Faltungs- (engl. convolution) Operation verwenden anstatt für gewöhnlichen Matrizenmultiplikation wie bei einer Dichte-Schicht (engl. dense layer) (vgl. Goodfellow 2018, S. 369–371).

CNN Architektur bietet gegenüber eines gewöhnlichen Feed Forward-Netzes mit vollverbundenen Schichten (engl. Fully Connected Layers) bzw. Dichte-Schichten (engl. Dense Layer) erhebliche Vorteile. Trotz weniger Gewichten ist ein CNN immer noch in der Lage, wichtige räumliche und zeitliche Informationen aus bspw. einem mehrdimensionalen Bild zu erkennen. Dabei wird das Bild zu einer Darstellung reduziert, welche einfacher verarbeitbar ist, oh-

ne dabei wichtige Eigenschaften zu ignorieren. (vgl. zu diesem Abschnitt Saha 2018)

Eine CNN beinhaltet immer Faltungs-Schichten (engl. Convolutional Layer). Diese nehmen einen mehrdimensionalen Eingang entgegen. Es wird ein „Kernel“ bzw. ein „Filter“ benötigt. Außerdem ist es möglich eine Schrittweite zu spezifizieren. Wenn im beschriebenen Beispiel die Schrittweite gleich 1 ist, so wird auf jeden 3 * 3 Block das Kreuzprodukt zwischen Filter und Eingangs-Block berechnet, danach schreitet der Block einen Schritt weiter und wiederholt den gleichen Prozess bis das Ende des Inputs erreicht wurde. (vgl. zu diesem Abschnitt Saha 2018; Li et al. 2020)

Üblicherweise werden Faltungs-Schichten mit Pooling-Schichten kombiniert. Pooling-Schichten dienen zur Reduktion der Dimensionsgröße und können dabei helfen, dominante Eigenschaften aus Eingängen herauszulesen. Es gibt zwei Arten von Pooling, das Max-Pooling (vgl. *Abbildung* 3) und das Average-Pooling. Max-Pooling wählt das Maximum aus einen durch die Kernel-Dimension definierten Daten-Ausschnitt. Average-Pooling berechnet den Mittelwert des Daten-Ausschnitts definiert durch die Kernel-Dimension. (vgl. zu diesem Abschnitt Saha 2018)

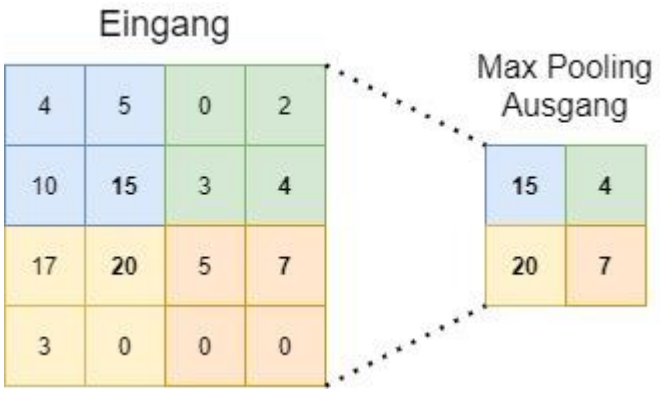


Abbildung 3*: Veranschaulichung von Max Pooling mit der Kernel Dimension (2, 2) und Schrittweite= 2 (vgl. Saha 2018)*

2.1.5 Recurrent Neural Netzworks (RNNs)

RNNs sind neuronale Netze, welche eine Rückkopplung beinhalten und dadurch längere Sequenzen von Daten verarbeiten können (vgl. Goodfellow 2018, S. 415).

Ein RNN besitzt, verglichen zu einem FNN, neben Gewichten zusätzlich einen weiteren versteckten Zustandsvektor, welcher die vorherigen Eingänge mit einbezieht. Ein normales NN hat eine feste Eingangs- und Ausgangsvektorgröße. Wenn eine Transformation mehrfach auf einen Eingang angewendet wird und eine Folge von Ausgangsvektoren erstellt wird, dann wird dieses Netzwerk rekurrent. Es gibt keine vorgeschriebene Vektorgröße. Zusätzlich ist beim Ausgang, welcher sich aus dem Eingang und einem versteckten Zustand ergibt, dieser versteckte Zustand abhängig von vorhergehenden Eingängen. (vgl. zu diesem Abschnitt Venkatachalam 2019)

Das RNN ist in der Lage durch das Einführen des versteckten Zustands, welcher vergangene Schritte betrachtet, die Eingangsdaten in einem breiteren Kontext wahrzunehmen (vgl. *Abbildung 4*). Ein Vorteil, der das RNN zu einem effektiven NN für Informationsketten macht. (vgl. zu diesem Abschnitt Venkatachalam 2019)

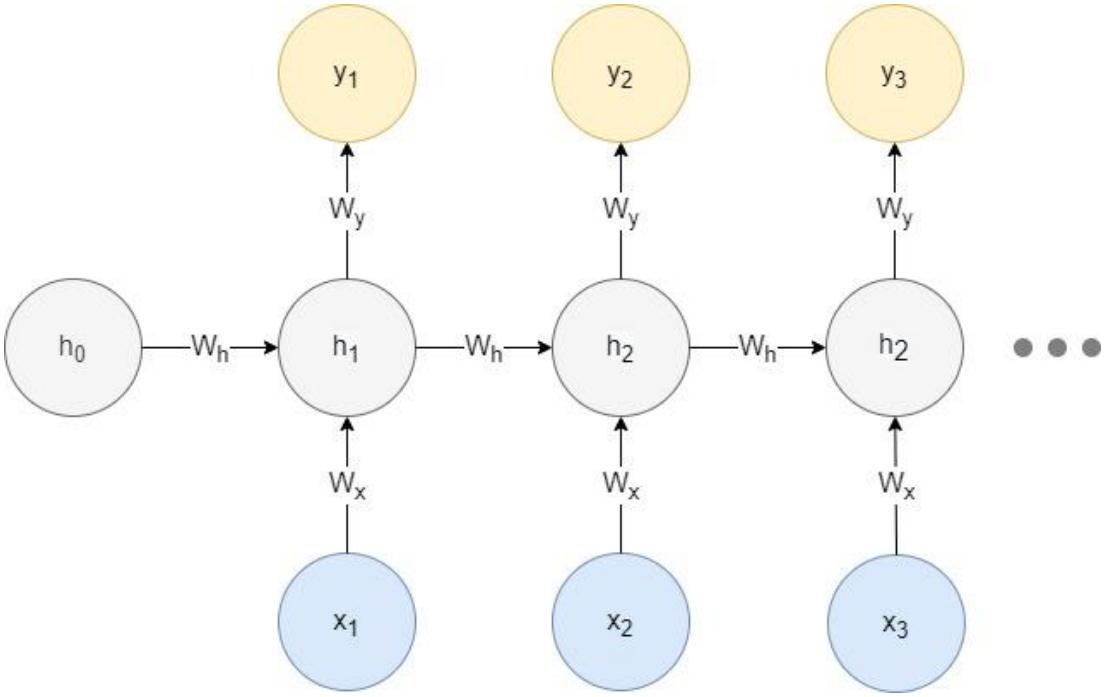


Abbildung 4: RNN, in dem der versteckte Zustand h zu nächsten Eingängen weitergegeben wird (Quelle: Venkatachalam 2019).

2.1.6 Long Short Term Memory (LSTM) Networks

LSTMs sind eine besondere Form des RNNs, dass Langzeitabhängigkeiten im Gegensatz zum RNN vermeidet (Olah 2015).

In einem RNN besteht das NN aus mehreren miteinander verketteten Komponenten. Die Struktur ist einfach gehalten mit oftmals einer Tanh-Funktion. LSTMs weisen die gleiche Kettenstruktur auf, mit dem Unterschied, dass anstelle einer einzelnen Schicht, vier verschiedene Schichten miteinander agieren (vgl. *Abbildung 5*) (vgl. zu diesem Abschnitt Olah 2015).

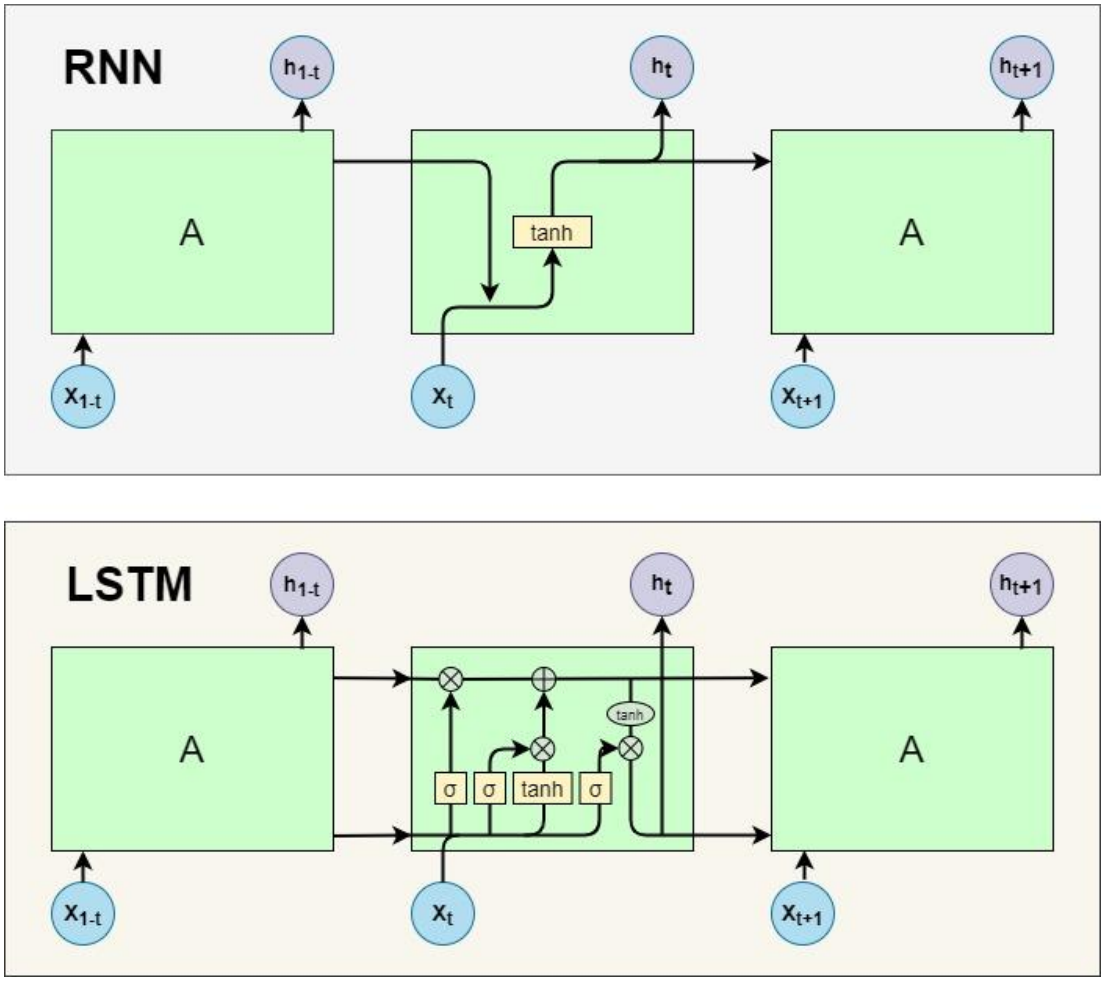


Abbildung 5: Vergleich von LSTM zu RNN (vgl. Olah 2015)

Der oberste lange horizontale Pfeil des LSTMs in *Abbildung 5* beschreibt den Zellenzustand, ihm können Informationen hinzugefügt, oder entnommen werden. Die Sigmoid-Schichten (σ in Abbildung 6) geben einen Wert zwischen 0 und 1 durch und regulieren innerhalb der Komponente, welche der ankommenden Information durchgelassen werden sollen. Bei einem Wert von 1 wird die volle Information durchgelassen, bei einem Wert von 0 wird keine der Informationen durchgelassen. In einem LSTM gibt es drei dieser Schichten. (vgl. zu diesem Abschnitt Olah 2015)

Zusammenfassend lässt sich sagen, dass sich das LSTM von dem RNN in der Hinsicht unterscheidet, dass neben dem Merken von vorherigen Schritten das LSTM durch seine Architektur Informationen gezielt filtern kann.

LSTMs lösen das Problem des RNN, bestimmte Informationen zu stark zu gewichten und andere wichtige Informationen wiederum für den breiten Kontext zu vernachlässigen. Aus diesem Grund sind LSTMs in vielen Situationen die bevorzugte rekurrente Architektur (vgl. Olah 2015; Hochreiter und Schmidhuber 1997, S. 1).

2.1.7 Autoencoder

Ein Autoencoder beschreibt ein System, welches dazu in der Lage ist, aus einem Eingang denselben Ausgang zu berechnen. Dabei gibt es eine verdeckte Schicht, dessen Code die Eingabe repräsentiert. Das neuronale Netz besteht aus einer Encoder- und einer Decoder-Funktion, woraus sich kombiniert eine Rekonstruktion ergibt. (vgl. zu diesem Abschnitt Rocca 2019)

Autoencoder eignen sich zur Reduktion von Daten. Dabei wird die Code-Dimension des Autoencoders reduziert. Das System ist somit gezwungen, die Informationen aus dem Eingang auf die wichtigsten Merkmale zu reduzieren. Diese Art von Autoencoder (vgl. Abbildung 4) nennt sich unvollständiger Autoencoder. (vgl. zu diesem Abschnitt Goodfellow 2018, S. 564–565)

Der unvollständige Autoencoder kann als Kompressionsverfahren betrachtet werden. Der Encoder reduziert den Eingang zu einem komprimierten Bereich, auch latenter Raum genannt und der Decoder dekomprimiert die Daten aus dem latenten Raum wieder in den Ausgangszustand. Ziel hierbei ist es, ein System, bestehend aus Encoder und Decoder zusammenzustellen, bei dem ein geringstmöglicher Rekonstruktionsfehler erreicht wird. Effektiv besitzt der Autoencoder das Potenzial, die Prinzipien zu erlernen auf denen die Trainingsdaten basieren. (vgl. zu diesem Abschnitt Goodfellow 2018, S. 565; Rocca 2019)

Autoencoder (vgl. *Abbildung 6*) lassen sich mit neuronalen Netzen umsetzen. Encoder und Decoder sind hierbei jeweils ein neuronales Netz. Die optimalen Parameter des latenten Raums werden während des Trainings ermittelt. Durch die Struktur des Netzwerks wird zwischen Encoder und Decoder ein Bottleneck (Engpass) erzeugt. (vgl. zu diesem Abschnitt Rocca 2019)

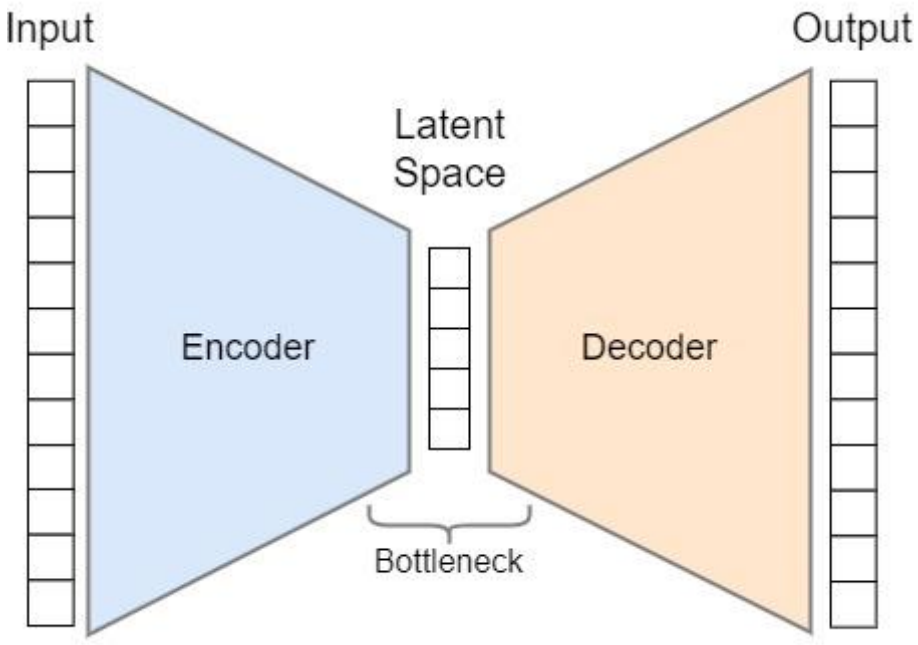


Abbildung 6: Darstellung eines Autoencoders mit Dimensions-Reduktion (vgl. Rocca 2019)

Es ist unwahrscheinlich, dass ein Netzwerk während des Trainings einen organisierten latenten Raum erstellt. Eine Problematik bei einem Autoencoder ist bspw., dass aufgrund der Freiheit eine Überanpassung (engl. Overfitting) stattfinden kann. Der latente Raum wird somit zu spezifisch auf den Trainingsdatensatz angepasst. Die Folge ist das nur bestimmte Vektoren des latenten Raums eine Bedeutung haben und seine restlichen Punkte keine nützlichen Informationen verbergen. (vgl. zu diesem Abschnitt Rocca 2019)

2.1.8 Variational Autoencoder (VAE)

Wie beim Autoencoder, ist es das Ziel, die Rekonstruktion der Eingangsdaten und damit den Rekonstruktionsfehler zu minimieren. Anders als beim gewöhnlichen Autoencoder, erlernt der VAE eine Verteilung, dessen latenter Raum organisiert ist. Zunächst wird der Eingang x im Encoder $q(z|x)$ kodiert zu einer Verteilung im latenten Raum $p_{model}(z)$ (vgl. *Abbildung 7*). Als zweiter Schritt wird eine Stichprobe z aus der Verteilung des latenten Raums gezogen. Als dritter Vorgang wird die Stichprobe im Decoder $p_{model}(x|z)$ de-

kodiert, wodurch der Ausgang $d(z)$ erhalten wird und schließlich der Rekonstruktionsfehler berechnet werden kann. Zuletzt wird der Rekonstruktionsfehler zurück durch das Netzwerk propagiert. (vgl. zu diesem Abschnitt Rocca 2019; Goodfellow 2018, S. 785)

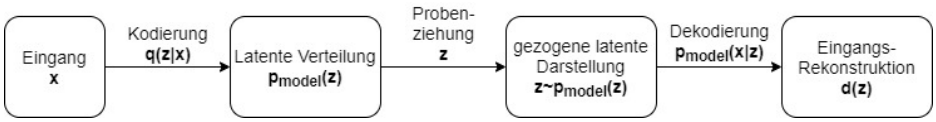


Abbildung 7: Probabilistisches Modell des VAE (vgl. Rocca 2019)

In der Anwendung wird eine Normalverteilung gewählt. Das erlaubt das Berechnen des Mittelwerts und das der Kovarianzmatrix. Hierdurch ist es möglich, die Regularisierung des latenten Raums zu kontrollieren. Durch diese Faktoren ist eine globale (Kontrolle des Mittelwerts) Regularisierung mit der „Maximum Likelihood“ Methode und eine lokale (Varianz Kontrolle) Regularisierung durch „Variational Inference“ möglich. Es wird also die Nähe und die Varianz der Verteilungspunkte kontrolliert (vgl. zu diesem Abschnitt Rocca 2019; Kingma und Welling 2013, S. 1–2)

Der VAE trainiert durch das Maximieren des ELBO (Evidence Lower Bound), welche als seine Verlustfunktion dient. Bezogen auf den Datenpunkt x ergibt sich folgende Formel für die ELBO:

$$L(q) = \mathbb{E}_{z\sim q(z|x)} \log_{p_{model}}(x|z) - D_{KL}(q(z|x)||p_{model}(z)).$$

(vgl. zu diesem Abschnitt Goodfellow 2018; Kingma und Welling 2013)

Der erste Term ist der Rekonstruktionsterm, beschrieben als Log-Likelihood. Der zweite Term ist der Regularisierungsterm, umgesetzt als KL-Divergenz zwischen der angenäherten posteriori Verteilung $q(z|x)$ und der angenäherten priori Verteilung $p_{model}(z)$. In der Praktischen Umsetzung wird $q(z|x)$ gerne als Normalverteilung und $p_{model}(z)$ als Standardnormalverteilung betrachtet. Als Folge sorgt der erste Term für die Minimierung des Rekonstruktionsfehlers und der zweite Term dafür, dass die Verteilung $q(z|x)$ sich der

Standardnormalverteilung annähert. (vgl. zu diesem Abschnitt Kingma und Welling 2013, S. 4; Goodfellow 2018, S. 785–786; Wiseodd 2016)

Regulierung des latenten Raums

Ein Problem des Autoencoders bezogen auf die Generierung von Daten ist die bereits erwähnte Überanpassung, was dazu führt, dass viele Bereiche innerhalb des latenten Raums für die Generierung unbrauchbar sind. Im VAE wird dieses Problem gelöst, indem die erlernte Verteilung der Standard Normalverteilung angeglichen wird. Diese Maßnahme gewährleistet einerseits die Nähe der Kovarianzmatrix zur Identität und andererseits die Annäherung des Mittelwerts gen Null zur Vermeidung einer zu hohen Distanz zwischen den Verteilungen. Hiermit wird vermieden, dass Verteilungen zu weit im latenten Raum auseinanderliegen und möglichst überlappen, was allerdings zu einem größeren Rekonstruktionsfehler führt. (vgl. zu diesem Abschnitt Rocca 2019)

Durch die Regularisierung (vgl. *Abbildung 8*) wird also ein Gradient der kodierten Informationen des latenten Raums erzeugt, was wiederum ermöglicht, bspw. einen Punkt zwischen zwei Verteilungen im latenten Raum auszuwählen, welcher schließlich nützliche Informationen zurückliefert (vgl. Rocca 2019).

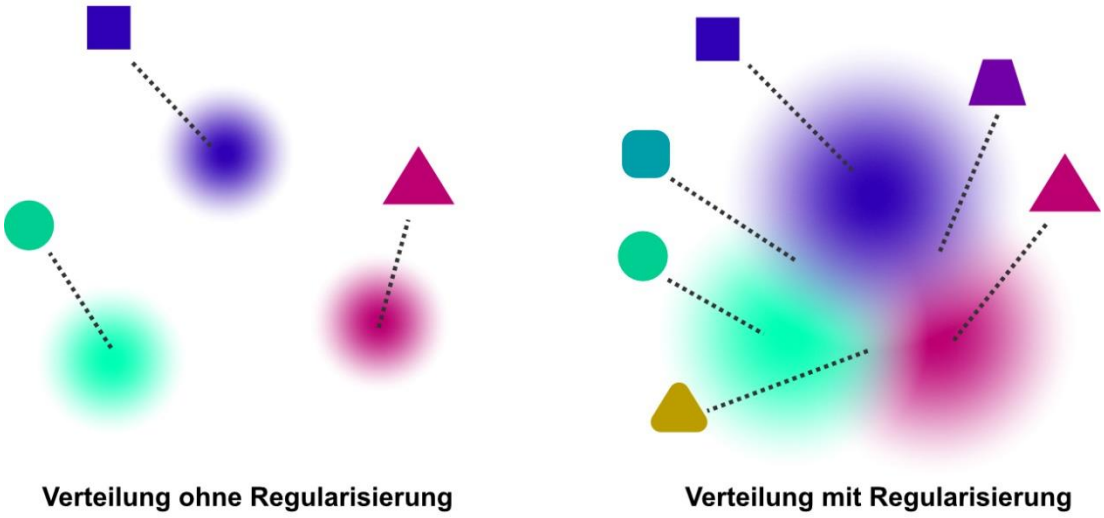


Abbildung 8: Vergleich zwischen einer Verteilung ohne Regularisierung und einer Verteilung mit Regularisierung (vgl. Rocca 2019)

Neuronaler VAE

Ein neuronales VAE kann in die drei Funktionen f, g und h unterteilt werden. Die Optimierung geschieht über die Parameter des Netzwerks. Die Funktionen-g und -h sind miteinander verbunden und teilen sich Gewichte:

$$\mathrm{g}(\mathrm{x}) = \mathrm{g2}\big(\mathrm{g1}(\mathrm{x})\big);\ \ \mathrm{h}(\mathrm{x}) = \mathrm{h2}\big(\mathrm{h1}(\mathrm{x})\big);\ \ \ \mathrm{g1}(\mathrm{x}) = \mathrm{h1}(\mathrm{x}).$$

(vgl. zu diesem Abschnitt Rocca 2019)

Um die Anzahl der Parameter zu reduzieren wird zusätzlich die Annahme gemacht, dass die Annäherung von $p_{model}(z)$ eine mehrdimensionale Normalverteilung mit einer diagonalen Kovarianzmatrix ist. Die Komponente g(x) stellt den Mittelwert der Verteilung $p_{model}(z)$ dar. Die Komponente $\mathrm{h}(\mathrm{x})$ beschreibt die Kovarianzmatrix und hat die gleiche Größe wie $\mathrm{g}(\mathrm{x})$ (vgl. *Abbildung 9*). (vgl. zu diesem Abschnitt Rocca 2019)

Das Model geht davon aus, dass $p_{model}(z)$ eine Gaußverteilung mit fester Kovarianz ist. Die Funktion f (vgl. *Abbildung 9*) ist der Decoder des VAE und wird in Form eines neuronalen Netzes realisiert (vgl. Rocca 2019).

Wenn Encoder und Decoder miteinander verkettet werden, bleibt ein letztes Problem bestehen. Im Netzwerk ist in diesem Zustand keine Backpropagation möglich. Der Umparametrisierungstrick (engl. Reparameterization Trick) wird angewendet, um das Propagieren der Verlustfunktion zurück durch das Netzwerk zu ermöglichen. (vgl. zu diesem Abschnitt Rocca 2019)

Der Umparametrisierungstrick basiert darauf, dass wenn ein z aus einer Standardnormalverteilung ($N(0,1)$) stammt, mit einem Mittelwert (μ) 0 und einer Standardabweichung (σ) 1, der Probenziehungsprozess als $z = \mu + \sigma^{1/2}\epsilon$ beschrieben werden kann, wenn $\epsilon \sim N(0,1)$. In dieser Form kann die Verlustfunktion, bestehend aus einem Rekonstruktions-, einem Regularisierungsterm und einer Konstanten, welche die relativen Gewichte beider Terme definiert, zurück propagiert werden. (vgl. zu diesem Abschnitt Rocca 2019; Wiseodd 2016)

Schlussendlich ergibt sich der in *Abbildung 9* gezeigte Aufbau des neuronalen Netzes.

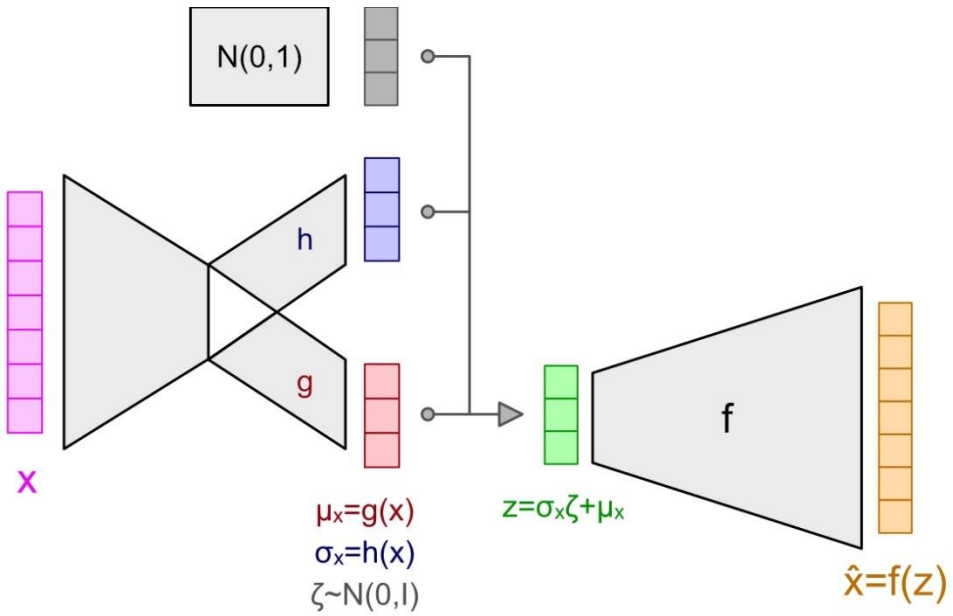


Abbildung 9: Aufbau eines VAE (Quelle: Rocca 2019)

2.1.9 Generative Adversarial Networks (GANs)

Die zweite generative Architektur, welche im Rahmen dieser Arbeit betrachtet wird, ist das GAN. Dieses System setzt sich aus zwei trainierbaren Modellen zusammen, einem Generator (G) welche über den Trainingsdatensatz die Verteilung der Daten trainiert und dem Diskriminator (D) der die Wahrscheinlichkeit abschätzt, dass die Probe aus dem Trainingsdatensatz stammt und nicht von G erstellt wurde. (vgl. *Abbildung 10*). (vgl. zu diesem Abschnitt Goodfellow et al. 2014, S. 1)

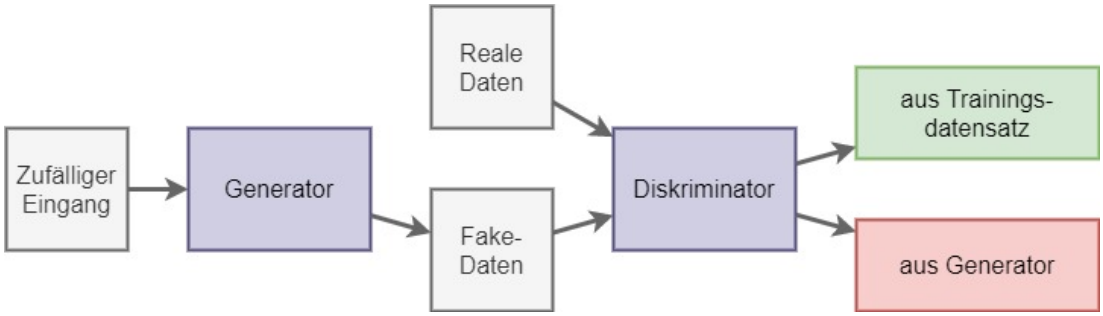


Abbildung 10: Aufbau eines GAN (vgl. Google Developers 2019)

Beim Lernprozess erzeugt G die Stichprobe $x = g(z; \theta^{(g)})$. Bei der Abwägung von D zwischen einer Stichprobe aus dem Trainingsdatensatz und einer Stichprobe, die von G erstellt wurde, gibt D einen Wahrscheinlichkeitswert $d(x; \theta)$ zurück, der aussagt, mit welcher Wahrscheinlichkeit x aus dem Trainingsdatensatz entstammt. Es findet ein Zwei-Personen-Nullsummenspiel zwischen G und D statt, mit einer Funktion $V(G, D)$. Es gilt:

$$\min_G \max_D V(G, D) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}\big[\log(1 - D\big(G(z)\big))\big].$$

(vgl. zu diesem Abschnitt Goodfellow 2018, S. 789–790; Goodfellow et al. 2014, S. 2–3)

Diese Verknüpfung führt dazu, dass D versucht, die Stichproben richtig einzuordnen, während G versucht, D mit den selbsterzeugten Stichproben reinzulegen. Bei der Konvergenz dieser Funktion sind generierte Stichproben und Datensatz-Stichproben für D nicht mehr unterscheidbar und D gibt für

alle Proben den Wahrscheinlichkeitswert $^1/_2$ zurück. Sobald dieser Fall eintritt, kann D ignoriert werden. (vgl. zu diesem Abschnitt Goodfellow 2018, S. 790)

Großer Vorteil eines GANs ist, dass keine Näherungs-Algorithmen benötigt werden. Wenn G und D genug Kapazität aufweisen und D in jedem Schritt die Möglichkeit hat, sein Optimum zu erreichen, wird das Verfahren mit Garantie konvergieren und ist somit asymptotisch konsistent. (vgl. zu diesem Abschnitt Goodfellow 2018, S. 790; Goodfellow et al. 2014, S. 5)

Im Gegenzug kann der Lernprozess wenn G und D NNs sind Schwierigkeiten bereiten. Bspw. kann das GAN unterangepasst (engl. underfitted) werden, wenn G und D nicht miteinander konvergieren. Ebenfalls ist nicht immer gegeben, dass Gs und Ds Gradientenabstieg mit einem Gleichgewicht der beiden Komponenten endet, wenn bspw. eine gewisse Abhängigkeit zwischen dem Aufwand von G und dem von D besteht. (vgl. zu diesem Abschnitt Goodfellow 2018, S. 790)

Bei Experimenten mit verschiedenen GAN-Szenarien stach ein Ansatz heraus. Anstelle das G versucht die Log-Wahrscheinlichkeit zu verringern, dass D korrekt kategorisiert, versucht G die Log-Wahrscheinlichkeit zu erhöhen, dass D fehlerhaft kategorisiert. Die Kostenfunktion von G bleibt somit auch dann groß, wenn der Diskriminator alle Stichproben des Generators als generierte Daten identifiziert. (vgl. zu diesem Abschnitt Goodfellow 2018, S. 791)

Das stabile Trainieren eines GANs ist eine herausfordernde Aufgabe und funktioniert nur sorgenfrei bei einer sorgfältigen Auswahl der Modellarchitektur und Hyperparameter. Das Lernproblem des GAN lässt sich verbessern durch die Aufteilung des Generierungsprozesses in mehrere Detailstufen. Es hat sich bspw. gezeigt, dass die Möglichkeit besteht, bedingte GANs (Mirza und Osindero 2014) zu trainieren, welche anstelle aus einer Randverteilung $p(x)$ aus einer Verteilung $p(x|y)$ Stichproben ziehen. (vgl. zu diesem Abschnitt Goodfellow 2018, S. 791)

2.2 Datentypen und Formate

2.2.1 Unterscheidung zwischen Audio und symbolischer Darstellung

Bei der Wahl des Formats besteht die Entscheidung zwischen Audio und symbolhafter Darstellung. Die Struktur und die Zielgruppen dieser Dateitypen unterscheiden sich. Dennoch ist die Verarbeitung der Daten-Formate innerhalb eines neuronalen Netzes nahezu gleich und beide Formate erweisen sich als tauglich zur originären Komposition musikalischer Inhalte.

Es gibt verschiedene Gründe für die Tendenz zu einer symbolhaften Darstellung.

- Viele der aktuellen Forschungen im Bereich der DL-Musikgenerierung verwenden symbolhafte Formate.
- Symbolhafte Darstellungen bieten für den menschlichen Nutzer eine Abstraktion, aus der harmonische Strukturen der Musik schnell ersichtlich sind und welche beim Verständnis der Vorgänge im System helfen kann.
- Das Preprocessing von Audio-Formaten unterscheidet sich, zu dem der symbolhaften Formaten und erfordert separates Fachwissen.

(vgl. zu diesem Kapitel Briot et al. 2020, S. 20–21)

2.2.2 Symbolhafte Formate

Symbolische Darstellungen dienen zur Interpretation musikalischer Konzepte durch den Menschen oder Computer. Hierbei werden Konzepte wie z. B. Noten, Notenlängen und Akkorde behandelt. (vgl zu diesem Kapitel Briot et al. 2020, S. 24)

2.2.3 MIDI

Musical Instrument Digital Interface (MIDI) bezeichnet einen Standard für digitale Schnittstellen für die Kommunikation mit bzw. zwischen verschiede-

nen elektronischen Instrumenten. MIDI transportiert Nachrichten (vgl. *Abbildung 11*) welche die Ausführung einer Note in Echtzeit beschreiben.

Note-on

Die Note-on Nachricht signalisiert das eine Note gespielt wird. Sie enthält folgende Informationen.

- Die Kanal-Nummer des abgespielten Tracks bzw. Instruments. $\{x \in \mathbb{Z} \mid 0 \leq x \leq 15\}$
- Die MIDI-Note, welche die Frequenz beschreibt, die abgespielt wird. $\{x \in \mathbb{Z} \mid 0 \leq x \leq 127\}$
- Die Dynamik (engl. Velocity), welche die Lautstärke der abgespielten Note festlegt. $\{x \in \mathbb{Z} \mid 0 \leq x \leq 127\}$

Note-off

Die Note-off Nachricht signalisiert das Ende einer Note. Die Dynamik beschreibt hierbei, wie schnell die Note gestoppt werden soll.

Jede Note wird in ein Spurstück eingebettet, welches die Deltazeit enthält und den Zeitpunkt der Aktion, welcher relativ oder absolut definiert sein kann.

Ein Problem dieses Formats ist, dass Informationen über gleichzeitig gespielte Noten nicht übertragen werden. Für ein Modell, welches mit MIDI-Daten arbeitet, kann es schwer sein, Mehrstimmigkeit zu lernen. (vgl. zu diesem Kapitel Briot et al. 2020, S. 29–30)

```
note_on channel=2 note=91 velocity=0 time=30
note_on channel=2 note=91 velocity=30 time=0
note_on channel=2 note=91 velocity=0 time=30
note_on channel=2 note=80 velocity=55 time=180
note_on channel=2 note=80 velocity=0 time=60
```

Abbildung 11: Beispiele von MIDI-Nachrichten

2.2.4 Piano-Roll

Das Piano Roll Format wird in verschiedenen Musikumgebungen wie Digital Audio Workstations (DAW) zur Visualisierung der Musik verwendet (vgl. *Abbildung 12*), zumal es meist leichter verständlich ist, verglichen mit z. B. gewöhnlicher Musiknotation. Die y-Achse, oft gekennzeichnet durch Klaviertaste gibt Informationen über die Frequenz der Note. Die x-Achse beschreibt den Zeitpunkt und die Dauer der Note.

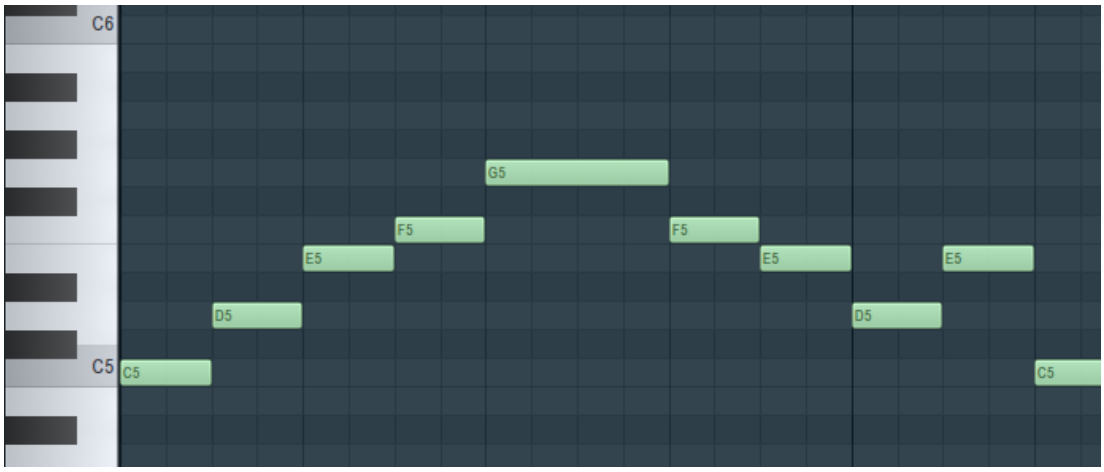


Abbildung 12: Beispiel einer Piano Roll in der DAW FL-Studio

Das Piano Roll-Format ist eines der meistverbreiteten Darstellungen. Trotz dessen gibt es Limitierungen, wie das es keine Note-Of Information gibt. Daraus ergibt sich die Problematik, dass nicht zwischen einer langen Note und mehreren kurzen Noten hintereinander unterschieden werden kann. (vgl. zu diesem Kapitel Briot et al. 2020, S. 29–32)

2.3 Musik in Videospielen

Musik ist in Videospielen einer der wichtigsten Aspekte, welcher den Spielern ermöglicht, in die Spielwelt einzutauchen. Das aus der Literatur bekannte Konzept der willentlichen Aussetzung des Unglaublichen ist auf Videospiele übertragbar. Das Konzept beschreibt wie Menschen sich auf fiktive Welten in z. B. Romanen einlassen, selbst wenn normalerweise Aspekte der Handlung fantastisch oder unmöglich erscheinen würden. Ziel des Autors ist hierbei

das Kreieren einer Illusion, in der alle Handlungsstränge der Erzählung im Rahmen der fiktiven Welt als möglich erscheinen. (vgl. zu diesem Abschnitt Phillips 2014, S. 35–37)

2.3.1 Immersion

Die Immersion ist nahe verknüpft mit der Theorie der Willentlichen Aussetzung des Unglaublichen. Die Immersion beschreibt einen Zustand, bei dem der Rezipient so sehr in die fiktive Welt vertieft ist, dass Vorgänge außerhalb der fiktiven Welt, wie z. B. der Alltag entfernt erscheinen, bis komplett vergessen werden. Hinzu kommt im Fall von Videospielen, dass die Immersion aktiv ist anstelle von passiv wie in klassischen Medien. Man spricht im Bereich der Videospiele oft vom Spielfluss, ein Zustand, in dem Vorgänge automatisch bzw. natürlich erscheinen und der Spieler für einen gewissen Zeitraum außerhalb seines Bewusstseins agiert. Dieser Verlust des Bewusstseins wird von vielen Videospielern positiv empfunden und kann somit das Spielerlebnis verbessern. (vgl. zu diesem Abschnitt Phillips 2014, S. 37–38)

Die Immersion lässt sich in drei Stufen unterteilen. Diese sind geordnet nach aufsteigenden Immersionsgrad „Engagement“, „Vertiefung“ und „totale Immersion“ (vgl. Phillips 2014, S. 39–40).

Das „Engagement“ beschreibt zunächst die initiale Motivation des Spielers, Zeit in ein Spiel investieren zu wollen. Musik kann hierzu beitragen und Aufregung erzeugen. Dabei sollte der Spieler auf keinen Fall im falschen Moment überfordert werden. Wichtige Funktion für diese Immersionsstufe ist der Zweck der Musik als Wegweiser. (vgl. zu diesem Abschnitt Phillips 2014, S. 40–44)

In der nächsten Immersionsstufe, der „Vertiefung“, folgt die emotionale Auseinandersetzung des Spielers mit der Spielwelt. Emotionalität wird vor allem durch intensive visuelle Einflüsse und fesselnde Geschichten gefördert, wobei Musik erheblich zur Verstärkung dieser Einflüsse mit beiträgt. Wenn der Spieler vor neue Herausforderungen gestellt wird, spielt Musik eine entscheidende Rolle diese herausfordernden Aufgaben unterhaltsam zu gestalten. (vgl. zu diesem Abschnitt Phillips 2014, S. 44–51).

Die letzte Immersionsstufe ist die „totale Immersion“. Ab dieser Stufe ist der Spieler voll und ganz in die Spielwelt eingetaucht. Nun kann die Spielwelt vom Spieler maximal wahrgenommen werden. Das Wecken von Empathie kann einen Weg zur totalen Immersion darstellen, wobei die Musik ein einflussreiches Mittel ist. (vgl. zu diesem Abschnitt Phillips 2014, S. 51–54).

2.3.2 Abgrenzung der Videospielmusik zur gewöhnlichen Musik

Gewöhnliche Musik hat die Eigenschaft, eine Melodie zu besitzen. Eine Melodie in einem Song soll einprägsam sein und im Kopf des Rezipienten verweilen (Ohrwurm). Dasselbe Prinzip gilt auch für eine Videospielmelodie, allerdings besteht in Videospielen das Problem, dass Stücke oft wiederholt werden aufgrund der langen und unbestimmten Laufzeit bestimmter Spielaufgaben. Die Chance, dass eine Melodie für den Spieler nach einer Zeit störend wirkt, ist sehr groß. Wenn allerdings die Melodie komplett weggelassen wird, geht das einprägsame Element verloren. Ständiges Ziel der Videospielmusik ist es, den Kompromiss zwischen einprägsamer Melodie und keiner Melodie abzuwägen (vgl. zu diesem Kapitel Phillips 2014, S. 66–67).

2.3.3 Videospielmarkt

Ein Komponist wird für gewöhnlich in der späteren Phase der Videospielentwicklung angestellt. Die Zielgruppe steht zu diesem Zeitpunkt schon fest. Wenn die Zielgruppe noch nicht fest bestimmt wäre, stellt das eine große Hürde für das gesamte Projekt dar. Die Zielgruppe hat erheblichen Einfluss auf die spätere Komposition der Musik.

Zur Ermittlung der Zielgruppe kann z. B. „Focus Testing“ angewendet werden, falls das Spiel sich bereits in einem spielbaren Zustand befindet. Des Weiteren gibt es verschiedene wissenschaftliche Modelle wie bspw. das „Demographic Game Design Model“ (DGD1) (Bateman und Boon 2006), die zur Ermittlung der Zielgruppe beitragen. (vgl. zu diesem Kapitel Phillips 2014, S. 78–79)

2.3.4 Musik in Rollenspielen

Rollenspiele (engl. Roleplay Game) ermöglichen dem Spieler das individuelle Gestalten eines Charakters oder einer Gruppe von Charakteren. Zentral gibt es eine Story mit einer übergreifenden Geschichte mit Hauptmissionen. Neben den Hauptmissionen bestehen optionale Nebenmissionen. Die Charaktere sind dazu in der Lage, Level aufzusteigen, wodurch der Spielbare Charakter neue Fähigkeiten bzw. Gegenstände erhält und stärker wird.

Rollenspiele lassen sich kulturell in westliche Rollenspiele (Rollenspiele der westlichen Hemisphäre) und nordostasiatische Rollenspiele unterteilen. Während westliche Rollenspiele eher eine dunkle Atmosphäre und viel Spielfreiheit bieten, verfügen nordostasiatische Rollenspiel über wesentlich hellere farbenfrohe Atmosphären und haben meist lineare Handlungsstränge.

Am besten passen zum Rollenspiel die Musikgenres Rock, orchestrale Musik und Jazz. Die Praxis zeigt, dass westliche Rollenspiele sich lieber der Orchestrierung bedienen und manchmal Rockelemente einfließen lassen. Bei nordostasiatischen Rollenspielen wird gerne zwischen Rock und Orchestraler Musik variiert, wobei ab und zu Techno und Pop Elemente einwirken. (vgl. zu diesem Kapitel Phillips 2014, S. 87–88)

2.3.5 Rollenspielszenario Kampf

Wenn der Spieler einen Gegner angreift oder von einem Gegner angegriffen wird, wechselt das Musikstück von friedvoller Erkundungsmusik zu Adrenalin bepackter Kampfmusik.

Ziel der Kampfmusik ist die Unterstützung der Spieler beim Kampf gegen den Widersacher. Die Musik soll das Tempo des Spielers bestimmen und seine Konzentrationsfähigkeit steigern, um dem Spieler das präzise Ausführen von Eingabeaktionen zu ermöglichen. Ziel des Spielers ist das Besiegen des Gegners, damit Erfahrungspunkte gesammelt werden können und der Spielcharakter Level aufsteigen kann.

Verglichen mit Musik in anderen Szenarien ist Kampfmusik meist dissonanter (unharmonischer). Musikstücke werden des Öfteren in einer Moll-Tonart

komponiert. Die Stücke beinhalten Chromatik (Tonwahl außerhalb der Tonleiter) mit nicht diatonischen (Dur-/Moll-System) Tonleitern und dissonanten Akkorden. Diese zuvor genannten Stilmittel dienen zur Spannungserzeugung beim Spieler. Bei einem rundenbasierten Kampfsystem kann zusätzlich mit diesen Komponenten kompensiert werden, dass die Entscheidungsfreiheit des Spielers gerade beschränkt wird.

In modernen Rollenspielen werden häufig Komponenten populärer Musik übernommen, sodass Synthesizer, E-Gitarre und Schlagzeug aus Rock und Pop-Musik häufig genutzte Klangfarben sind.

In vielen Fällen werden verschiedene Kampfmusik-Stücke komponiert, um die Kämpfe nach Relevanz für den Spieler zu unterteilen. So wird für seltenere, aber einflussreichere Bosskämpfe ein anderes Musikstück eingesetzt als häufig eintreffende Kämpfe, welche wenig Einfluss auf den Fortschritt des Spielers haben. (vgl. zu diesem Kapitel Rossetti 2020, S. 62–63)

2.3.6 Rollenspielszenario Überwelt

Das Überwelt-Musikstück dient dazu, die musikalische Sprache und Soundpalette des Videospiels dem Spieler zu kommunizieren. Diese Art von Musikstück wird beim Reisen durch die Spielwelt abgespielt. Anknüpfend dienen Überwelt-Musikstücke als Lückenfüller zwischen Kämpfen.

Zu den Funktionen der Überweltmusik gehören das „world building“, das Beeinflussen des Geisteszustands des Spielers hin zu einem für eine aus Sichte des Spiels vorteilhaften Zustand, die Markenentwicklung des Spiels, als auch die Abgrenzung zu anderen Spielen.

Während Rhythmus und Tempo den Geisteszustand beeinflussen, trägt die Melodie zum Branding bei. Meist fungiert die Überwelt im Spiel als sicherer Bereich für den Spieler, in der keine Kämpfe geschehen können.

Überwelt-Musikstücke betonen häufig Thematiken wie Heldentum, Krieg, Böses und Sicherheit. Themen die in Fantasy-Rollenspielen möglichst häufig thematisiert werden.

Bestimmte Stilmittel, wie das Mischen zwischen Tonleiter-Modi, dienen, um Mehrdeutigkeit bzw. Instabilität zu schaffen. Musikstücke habe für gewöhnlich Song-artige Strukturen, um wiedererkennbare Muster aufzuweisen. Ein Gefühl der Familiarität wird geschaffen durch einen stabilen Rhythmus und stabile Tempi, zur Anregung des Spielers zur Erkundung der Welt. Mit bestimmten Klangfarben realisiert durch z. B. Instrumentenwahl wird Information über die Landschaft kommuniziert. (vgl. Rossetti 2020, S. 12–13)

3 Analyse generativer DL-Modelle der Musik

Im vorherigen Kapitel wurde bereits die Problematik der generativen Modelle betrachtet und vorgestellt. In diesem Kapitel werden Systeme, bzw. Umsetzungen musikalisch eingesetzter DL Modelle betrachtet. Anschließen werden diese Lösungen im Kontext des Projekts evaluiert.

3.1 MuseGAN

MuseGAN ist eine Umsetzung einer GAN Architektur, die dazu in der Lage ist, symbolische (vgl. Kap. 2.2.2) Darstellungen zu generieren. Im Rahmen dieses Projekts wurden drei Modelle trainiert, eins mit dem Ziel der Improvisation, das zweite mit Fokus auf Komposition und das letzte Modell kombiniert Komposition und Improvisation. (vgl. zu diesem Abschnitt Dong et al. 2018, S. 1)

Trainiert wurden diese Modelle auf Basis eines Datensatzes mit über hunderttausend Takten, bestehend aus Rockmusik. Das Modell generiert fünfspurige Piano-Rolls mit den Instrumenten Bass, Schlagzeug, Gitarre, Klavier und Streicher. Zusätzlich wurden objektive Metriken erstellt zur Auswertung der Ergebnisse. Es ist zusätzlich möglich, eine Eingangsspur mitzugeben, zu der 4 zusätzliche Spuren generiert werden. (vgl. zu diesem Abschnitt Dong et al. 2018, S. 34)

3.1.1 Architektur

Das System ist nach Prinzip eines GANs (vgl. Kap. 2.4.5) mit einem Generator und einem Diskriminator aufbebaut. Generator und Diskriminator wurden jeweils als „Deep Convolutional Neuronal Networks“ umgesetzt. Ergänzend beruft sich das Projekt auf eine Forschungsstudie (vgl. Arjovsky et al. 2017) und ersetzt die gewöhnlich verwendete Jensen-Shannon-Divergenz mit der Wasserstein-Metrik, mit Ziel, eines reibungslosen Trainings und der Vermeidung eines Mode Collapses. (vgl. zu diesem Abschnitt Dong et al. 2018, S. 35)

Zur Durchsetzung einer Lipschitzstetigkeit wird Gewichtsbeschneidung (engl. weight clipping) in einem Wasserstein-GAN angewendet. Zusätzlich gibt es einen Gradienten-Strafterm für den Diskriminator. Die Funktion für den Diskriminator lautet folglich

$$\mathbb{E}_{\hat{x}\sim\mathbb{P}d}\big(D(x)\big) - \mathbb{E}_{z\sim\mathbb{P}z}\Big(D\big(G(x)\big)\Big) + \mathbb{E}_{\hat{x}\sim\mathbb{P}\hat{x}}\Big(\big(\nabla_{\hat{x}}\big|\big|\hat{x}\big|\big| - 1\big)^2\Big).$$

$\mathbb{P}\hat{x}$ beschreibt das gleichmäßige ziehen von Stichproben entlang einer geraden Linie zwischen Paaren von Punkten der Verteilung $\mathbb{P}d$ und der Verteilung P $\mathbb{P}g$ (vgl. Gulrajani et al. 2017). Es entsteht ein WGAN-GP Modell, welches den Vorteil hat, dass es schneller und zu besseren Optima konvergiert und weniger Parameteranpassungen erfordert. (vgl. zu diesem Abschnitt Dong et al. 2018, S. 35)

3.1.2 Datenformat

Das Modell betrachtet einen Takt als grundlegende Einheit unter der Annahme, dass die meisten harmonischen Wechsel, mit dem Taktschlag stattfinden (vgl. Dong et al. 2018, S. 35).

Es wird das Piano-Roll-Format (vgl. Kap. 2.2.4) eingesetzt. Für jede Spur ergibt sich pro Takt eine separate Piano-Roll-Matrix, welche das Format $x \in \{0,1\}^{R*S*M}$ hat. R beschreibt die Anzahl an Zeitschritten, S die Anzahl der möglichen Noten und M die Anzahl der Spuren. (vgl. zu diesem KapitelDong et al. 2018, S. 35)

3.1.3 Modelle

Das erste trainierte Modell ist dazu in der Lage zu improvisieren (vgl. *Abbildung 13* (a)). Dieses Modell hat mehrere Generatoren (einen pro Spur), welche unabhängig voneinander eigene zufällige Vektoren bekommen und separat von einem jeweiligen Diskriminator (einen pro Spur) ausgewertet werden.

Das zweite Modell wurde zur Komposition trainiert (vgl. *Abbildung 13* (b)). Dieses Modell besitzt einen Generator und bekommt einen gemeinsamen

zufälligen Vektor, der mehrere Spuren kreiert. Analysiert werden diese Spuren von einem einzelnen Diskriminator.

Das letzte Modell kombiniert Modell eins und Modell zwei miteinander (vgl. *Abbildung 13* (c)). Es gibt mehrere Generatoren, die jeweils einen zufälligen Spuren-internen Vektor einen zufälligen Spuren-übergreifenden Vektor bekommen. Die generierten Spuren werden von einem Diskriminator entgegengenommen. (vgl. zu diesem Kapitel Dong et al. 2018, S. 36)

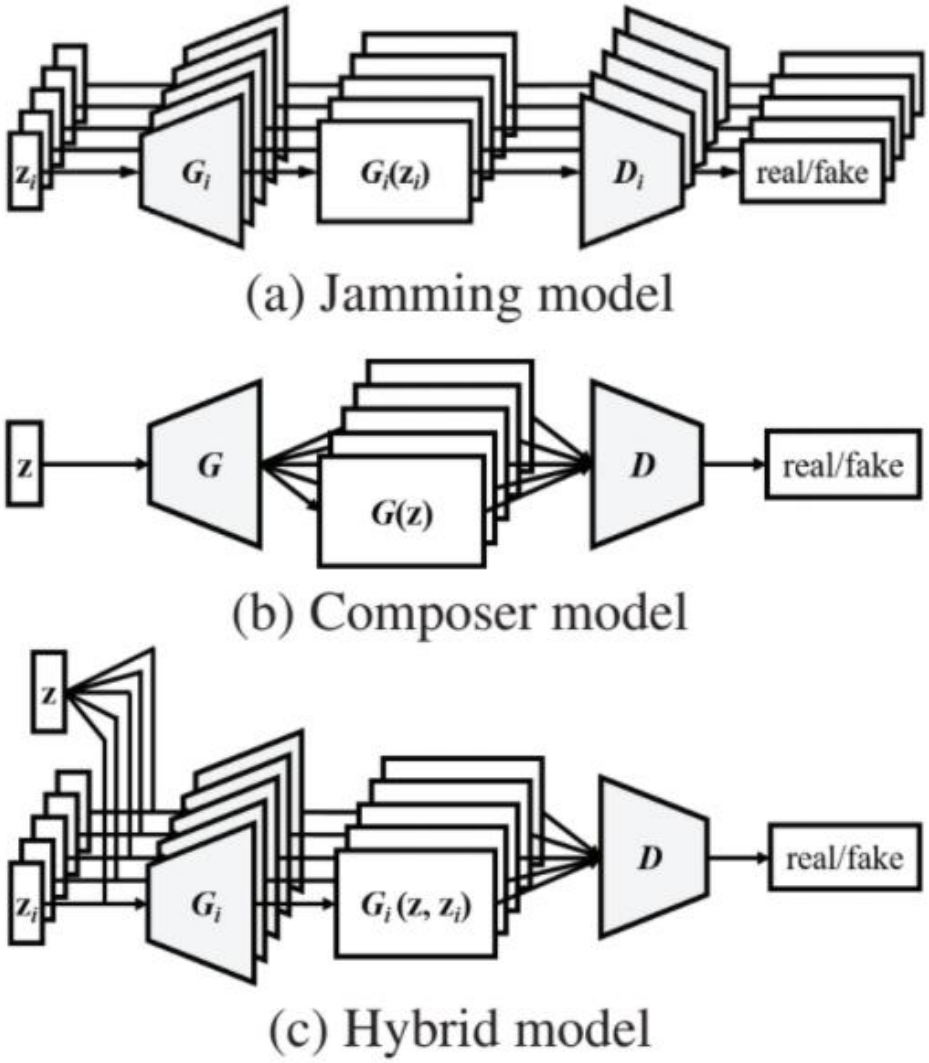


Abbildung 13: Veranschaulichung der drei MuseGAN-Modelle (Quelle: Dong et al. 2018, S. 36)

Die bisher gezeigten Modell sind dazu in der Lage mehrspurige Musik taktweise zu generieren. Um ein Musikstück bestehen aus mehreren Takten zu generieren, muss die Struktur erweitert werden.

3.1.4 Lösung der zeitlichen Struktur

Eine Möglichkeit der Erweiterung besteht darin, dass ein Stück von Grund auf mit einer festen Taktanzahl generiert wird. Die Taktanzahl kann also als weitere Dimension des Netzwerks betrachtet werden. Der Generator ist schlussfolgernd hierarchisch aufgebaut, mit einem Generator G_{temp} für die gesamte Songstruktur und einem Taktgenerator G_{bar}.

Als Nächstes wird ein zweiter möglicher Ansatz behandelt, indem die Spur eines einzelnen Taktes y dem Netzwerk vorgegeben wird. Ziel des Netzwerks ist die Erlernung der zeitlichen Struktur, welche die Spur voraussetzt, um darauffolgend das Musikstück zu vervollständigen. Die Takte werden nacheinander durch den Takt Generator G_{bar}° erstellt, welcher als Eingang die bedingte Spur und einen zufälligen Vektor bekommt. Um dementsprechend eine bedingte Generierung mit hochdimensionalen Konditionen zu ermöglichen wird ein Encoder trainiert, um diese auf einen Raum z abzubilden. (vgl. zu diesem Kapitel Dong et al. 2018, S. 36–37).

3.1.5 Endgültiges Modell

Das aus den vorherigen Schritten resultierende Modell kann in *Abbildung 14* betrachtet werden.

Der Input des MuseGANs $\bar{z}$ besteht aus 4 Teilen.

- Ein inter Spur zeitunabhängiger zufälliger Vektor – z,
- Ein intra Spur zeitunabhängiger zufälliger Vektor - z_i,
- Ein inter Spur zeitabhängiger zufälliger Vektor - z_t und
- Ein intra Spur zeitabhängiger zufälliger Vektor – $z_{i,t}$.

Für Spur i $(i = 1 \dots M)$ bekommt der gemeinsame Songstrukturgenerator G_{temp} und der private Songstrukturgenerator $G_{temp,i}$ die zeitabhängigen Vek-

toren z_t und $z_{i,t}$. Jeder Eingang und deren Ausgang enthält einen latenten Vektor mit einer Inter- und Intra-Spur-Zeitinformationen. Die latenten Ausgangsvektoren werden mit den zeitunabhängigen zufälligen Vektoren z und z_i verkettet und schließlich dem Takt-Generator G_{bar} übergeben, welcher letztlich die Piano-Rolls generiert. (vgl. zu diesem Abschnitt Dong et al. 2018, S. 37)

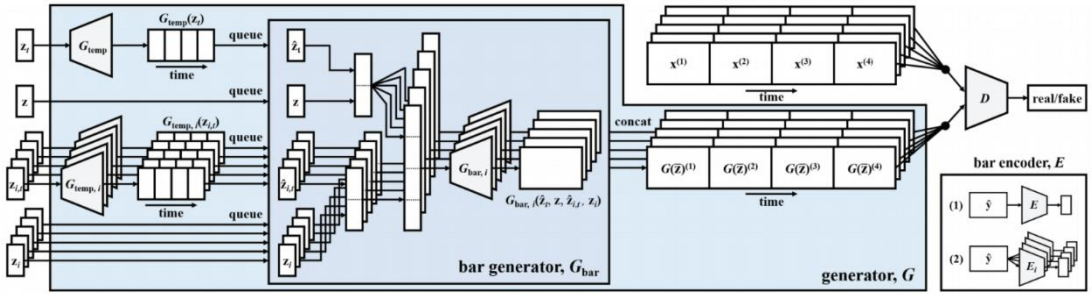


Abbildung 14: Aufbau des MuseGAN (Dong et al. 2018, S. 37)

3.1.6 Daten

Die Trainingsdaten stammen aus dem Lakh Pianoroll Dataset (LPD), bestehend aus 174.154 einzigartigen mehrspurigen Piano-Rolls, dem Lakh MIDI Dataset (LMD) (Raffel 2016) entnommen. Nach Bereinigung bleiben 21.425 Piano-Rolls bestehen.

Um den Datensatz zu bereinigen wurden mehrere Bearbeitungsschritte durchgeführt. Da manche Spuren nur wenige Noten enthalten, wurden Spuren mit ähnlichen Instrumenten zusammengefügt. Übrig bleiben die fünf Spuren Bass, Schlagzeug, Klavier, Gitarre und Streichinstrument. Zusätzlich wurden die Songs mit dem Million Song Dataset (MSD) (Bertin-Mahieux et al. 2011) nach Rock-Songs und Songs im 4/4 Takt gefiltert. Es werden 4 Takte als eine Phrase betrachtet. Möglich sind Noten von C1-C8 was insgesamt 84 mögliche Noten beträgt. Die Ausgangsdimension beträgt somit 4 (Takte)* 96 (Zeitschritte)* 84 (Noten)* 5 (Spuren). (vgl. zu diesem Kapitel Dong et al. 2018, S. 37–38)

3.2 MusicVAE

MusicVAE ist eine Umsetzung eines VAE (vgl. Kap. 2.4.4) zur Erlernung eines latenten Bereichs für die Rekonstruktion sequenzieller Musik. Das Projekt realisiert einen rekurrenten VAE mit einem hierarchisch aufgebauten Decoder. (vgl. zu diesem Abschnitt Roberts et al. 2018)

3.2.1 Recurrent VAE

Der Encoder $q_\lambda(z|x)$, ist ein RNN, mit der Eingangssequenz x und erstellt einen latenten Zustand h. Die Parameter der Verteilung über den latenten Raum z wird bezeichnet als die Funktion h_T. Der Decoder $p_\theta(x|z)$ zieht aus dem Stichprobenvektor z den initialen Zustand des Decoder RNNs, welcher schließlich autoregressiv die Ausgangssequenz y erstellt.

3.2.2 Encoder

Der Encoder (vgl. *Abbildung 15: Aufbau des MusicVAE (Quelle: Roberts et al. 2018)*) wurde als zweischichtiges bidirektionales LSTM-Netzwerk umgesetzt. Die Eingangssequenz x wird verarbeitet, um schließlich in der zweiten Schicht die Vektoren $\vec{h}_T, \overleftarrow{h}_T$ zu erhalten, welche danach miteinander verkettet werden und sich h_T ergibt. Dieser Parameter wird an zwei Dichte-Schichten weitergegeben, worüber die Parameter der Verteilungen des latenten Codes bestimmt werden können. Die Größe der LSTM-Schicht beträgt 2048 und die Anzahl der latenten Dimensionen 512. (vgl. zu diesem Kapitel Roberts et al. 2018)

3.2.3 Decoder

Der Decoder (vgl. *Abbildung 15: Aufbau des MusicVAE (Quelle: Roberts et al. 2018)*) ist hierarchischer Natur. Die Eingangssequenz x lässt sich in nicht überlappende Subsequenzen y_U unterteilen mit den Endpunkten i_U und einem separat betrachteten Fall $i_{U+1} = T$. Der Vektor z wird als Eingang einer Dichte-Schicht übergeben. Darauf folgt eine Tanh-Aktivierungsfunktion, um den initialen Zustand einer Dirigenten-RNN (engl. Conductor RNN) zu erhal-

ten. Das Dirigenten-RNN gibt eingebettete Vektoren für die Untersequenz zurück und ist ein zweilagiges unidirektionales LSTM mit einer Größe von 1024 und einer Ausgangsdimension von 512. Die eingebetteten Vektoren werden jeweils durch eine Dichte-Schicht geführt, gefolgt von einer Tanh-Aktivierungsfunktion. Nach der Aktivierungsfunktion bleiben schließlich die initialen Zustände erhalten, die ein letztes RNN verarbeitet und eine Sequenz an Verteilungen der Ausgangs-Token über eine Softmax-Ausgangsschicht generiert. Nach jedem Schritt der letzten Decoder-Stufe wird der aktuelle Einbettungsvektor mit dem vorherigen Ausgangs-Token verkettet und als nächster Eingang verwendet. Das letzte RNN ist ein zweischichtiges LSTM mit 1024 Einheiten pro Schicht. (vgl. zu diesem Abschnitt Roberts et al. 2018)

Die Chance eines „posterior collapse“, wobei das Decoder-RNN lernt, den latenten Bereich zu ignorieren, wird reduziert, indem der Sichtradius der untersten RNN Schicht reduziert wird. Jede Sequenz wird mit der Einbettung der Dirigenten RNN eingeleitet, wodurch das Netzwerk Langzeitstrukturen nur aus dem eingebetteten Vektor erkennen kann, der wiederum aus dem latenten Raum resultiert. (vgl. zu diesem Abschnitt Roberts et al. 2018)

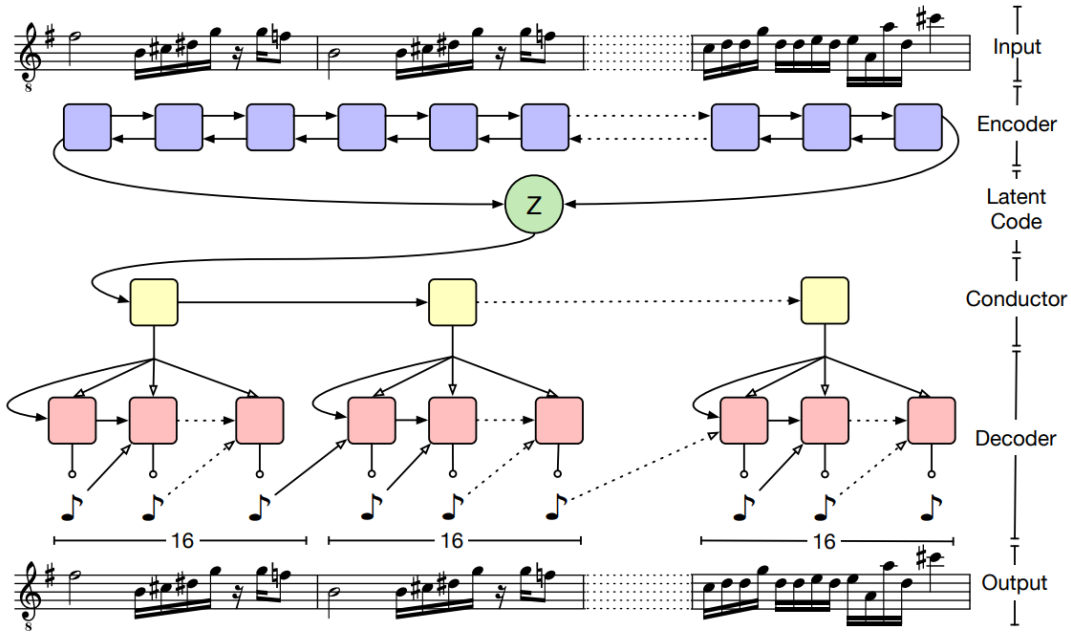


Abbildung 15: Aufbau des MusicVAE (Quelle: Roberts et al. 2018)

3.2.4 Daten

Das Neuronale Netzwerk verwendet MIDI-Daten. Aus den Trainingsdaten wurden 2-16 Takt Melodien, 2-16 Takt Schlagzeugmuster und 16 Takt Trio Sequenzen bestehend aus Melodie, Bass und Schlagzeugmuster entnommen. Die Wiederherstellungsqualität wurde zusätzlich mit dem Lakh MIDI Datensatz (LMD) (Raffel 2016) getestet. (vgl. zu diesem Kapitel Roberts et al. 2018)

3.3 Gegenüberstellung von GAN und VAE

Bei der Gegenüberstellung der beiden generativen Modelle GAN und VAE lassen sich jeweils Vorteile und Nachteile erkennen, welche die Modelle voneinander abgrenzen.

3.3.1 Pro und kontra GAN

Es hat sich in der Praxis gezeigt, dass GANs verglichen mit anderen generativen Modellen die qualitativ besten Ergebnisse zu liefern scheinen. Dagegen kann der Prozess zu einem hochwertigen Modell kompliziert sein. (vgl. zu diesem Abschnitt Goodfellow 2016, S. 17)

Eine Problematik, die auftreten kann, ist die fehlende Konvergenz des Modells. Da das GAN aus zwei neuronalen Netzen bestehen, welche in einer Zweispieler-Partie gegeneinander antreten, kann passieren, dass die Akteure gegeneinander arbeiten. Im Fall, dass z. B. Spieler-1 seine Kostenfunktion reduziert und dies dazu führt, dass Spieler-2 seine Kostenfunktion erhöht, kann das zu einer Nichtkonvergenz führen. (vgl. zu diesem Abschnitt Goodfellow 2016, S. 34)

Die Stabilisierung des Lernprozesses kann ebenfalls eine Herausforderung sein. Vor dem Training sollten die Modellstruktur und die Hyperparameter mit Bedacht ausgewählt werden, um Komplikationen zu vermeiden. (vgl. zu diesem Abschnitt Goodfellow 2018, S. 791)

Ein weiteres Problem was bei GANs auftreten kann ist der „Mode Collapse“ wobei der Generator mehrere verschiedene z Eingänge demselben Ausgang zuweist. Für gewöhnlich trifft teilweise „Mode Collapse“ ein, wobei z. B. der Generator mehreren Bildern die gleiche Textur zuweist oder mehrere Bilder das gleiche Thema beinhalten. Um dies zu vermeiden, sollten Generator und Diskriminator gut aufeinander abgestimmt sein. (vgl. zu diesem Abschnitt Goodfellow 2016, S. 34–35; Goodfellow et al. 2014, S. 6)

In manchen Fällen haben GANs mit schwindenden Gradienten (engl. vanishing gradients) zu kämpfen, was eintrifft, wenn der Diskriminator zu gut in seiner Aufgabe wird Daten zu unterscheiden. Wenn das der Fall ist, fehlt dem Generator der Freiraum, sich zu verbessern und der Generator hat keine Möglichkeit Fortschritte zu machen (vgl. zu diesem Abschnitt Arjovsky und Bottou 2017).

Eine Möglichkeit die zuvor genannten Probleme wie das der schwindenden Gradienten und den instabilen Lernprozess zu verbessern ist bspw. die Verwendung eines WGANs (Arjovsky et al. 2017) anstelle eines normalen GANs, wie im zuvor vorgestellten Modell MuseGAN (vgl. Kap. 3.1) beschrieben.

3.3.2 Pro und kontra VAE

Ein VAE kann hervorragende Ergebnisse liefern. In der Praxis zeigt sich jedoch, dass in den Ergebnissen oft Details fehlen. Somit sind generierte Bilder eines VAE oft etwas unscharf. Ein Grund für dieses Phänomen ist die Verwendung einer Normalverteilung für das Model. Ähnlich wie beim Autoencoder werden dadurch gewisse Merkmale der Eingabe ignoriert. (vgl. zu diesem Abschnitt Goodfellow 2018, S. 786–787)

Eine weitere Problematik, die VAEs aufweisen ist, dass meist nur ein Teil des latenten Raums z genutzt wird, obwohl theoretisch mehr Kapazität vorhanden wäre. (vgl. zu diesem Abschnitt Goodfellow 2018, S. 787)

In Kombination mit einem RNN wurde bereits in Kap. 3.2.3 der Fall angesprochen, dass ein „posterior collapse“ auftreten kann, wobei der latente Be-

reich vom RNN missachtet wird, was allerdings mit einem reduzierten RNN-Anwendungsbereich vermieden werden kann.

Von Vorteil beim VAE ist das die Architektur flexibel ist in der Hinsicht, mit welchen anderen Modellarchitekturen sie kombiniert werden kann. Angesprochen wurde z.B. schon die Kombination mit einem RNN, Genauso gut funktioniert die Kombination mit verschiedenen CNN Architekturen. (vgl. zu diesem Abschnitt Goodfellow 2018, S. 787)

Zusätzlich von Vorteil ist das ein VAE eine Verteilung explizit erlernt, während bspw. ein GAN eine Verteilung nur implizit erlernen kann. Schlussfolgernd wird ein vorhersagbares Koordinatensystem erlernt, aus dem in der Theorie Schlüsse über den erlernten Datensatz gezogen werden können. (vgl. zu diesem Abschnitt Goodfellow 2018, S. 788)

3.4 Metriken zur objektiven Bewertung

Im Rahmen der Bachelorarbeit wurden Metriken erarbeitet welche zur objektiven Bewertung von Musikstichproben dienen sollen. Im Rahmen des Modells MuseGAN (vgl. Kap. 3.1) wurden Metriken erstellt, welche im Kontext dieser Arbeit in Betracht gezogen werden. Abgesehen von der Tonal Distance, die eine Inter-Spur-Metrik ist, sind alle Metriken Intra-Spur-Metriken (vgl. Dong et al. 2018, S. 38).

3.4.1 Empty Bars (EB)

Diese Metrik analysiert das Verhältnis leerer Takte. Die Metrik ergibt sich aus der Anzahl leerer Takte geteilt durch die Anzahl aller Takte $\{0 \leq x \leq 1\}$ (vgl. zu diesem Abschnitt Dong et al. 2018, S. 38).

3.4.2 Used Pitch Classes (UPC)

Diese Metrik besagt, wie viele verschiedene Noten-Klassen pro Takt verwendet werden. Hierzu werden die Noten zunächst in 12 Noten-Klassen unterteilt. Das hat zur Folge, dass die Oktave einer Note als die gleiche Note betrachtet wird $\{0 \leq x \leq 12\}$ (vgl. Dong et al. 2018, S. 38).

3.4.3 Qualified Notes (QN)

Für diese Metrik wird der Anteil qualifizierter Noten berechnet. Eine Note wird als qualifizierte Note gesehen, wenn sie mindestens die ist. Diese Metrik soll Aussagen darüber treffen können, wie fragmentiert die Ergebnisse sind. Der Messwert ergibt sich aus der Anzahl der qualifizierten Noten geteilt durch die Anzahl aller Noten $\{0 \leq x \leq 1\}$. (vgl. zu diesem Abschnitt Dong et al. 2018)

3.4.4 Drum Pattern (DP)

Dieser Wert trifft Aussagen über den Rhythmus der Noten. Hierbei wird explizit betrachtet wie viele Noten in einem 8 oder 16tel Rhythmus vorkommen, ein Muster gewöhnlich für Rockmusik $\{0 \leq x \leq 1\}$. (vgl. zu diesem Abschnitt Dong et al. 2018, S. 38).

3.4.5 Tonal Distance (TD)

Für die TD (Harte et al. 2006) wird die harmonische Differenz zwischen 2 Spuren berechnet. Eine große TD sagt aus das 2 Spuren harmonisch weit auseinander liegen. (vgl. zu diesem Abschnitt Dong et al. 2018, S. 38)

3.4.6 Polyphonicity (PP)

Die Metrik beschreibt das Verhältnis von Zeitschritten, zu denen mehr als eine Note gespielt wird, zu allen Zeitschritten $\{0 \leq x \leq 1\}$ (vgl. Dong und Yang 2018).

4 Umsetzung

4.1 Trainings Umgebung

Als Trainingsumgebung wurde die open Source Distribution „Anaconda“ (Anaconda 2012) eingesetzt und eine Python Umgebung mit der Machine Learning Library „Tensorflow“ (2015) instanziiert. In Kombination wurde die Machine Learning API „Keras“ (2015) eingesetzt.

Für die Umsetzung der Architektur des neuronalen Netzes wurde sich an einem bereits bestehendes Projekt orientiert, welches ähnliche Ansätze verfolgte (vgl. CodeParade 2018; HackerPoet 2018). Das Repository bietet zusätzlich einige Methoden zum Pre- und Postprocessing der Daten. Außerdem sind bereits Hilfsmethoden für z. B. die grafische Darstellung der Daten vorhanden.

4.2 Trainings Datensatz

Für das Training wurden sequenzierte MIDI-Daten des Online-Musik-Archivs VGMusic.com (1996) verwendet. Die MIDI-Daten wurden separiert in zwei Kategorien. Die erste Kategorie fokussiert sich auf das Thema Kampf (vgl. Kap. 2.3.5) und die zweite Kategorie auf das Thema Überwelt (vgl. Kap. 2.3.6) in Videospielen. Alle verwendeten MIDI-Dateien wurden in einer Excel Tabelle dokumentiert (vgl. Anhang). Die MIDI-Daten werden für das neuronale Netz zunächst in das Piano-Roll-Format (vgl. Kap. 2.2.4) formatiert und in 16 Takt Blöcke aufgeteilt.

Im Beispiel eines Klassifizierungsproblems werden meist zwei Datensätze, ein X-Datensatz, der die ursprünglichen Daten enthält und ein Y-Datensatz, mit den Wahrheitswert der Daten, benötigt. Der Y-Eintrag ist hierbei ein Zahlen-Code, der die Klasse des X-Datensatz-Eintrags beschreibt. In einem Autoencoder-Netzwerk ist dies nicht der Fall bzw. der X-Datensatz ist gleichzusetzen mit dem Y-Datensatz, da das Trainingsziel grundsätzlich eine Rekon-

struktion der Ursprungsdaten ist. Schlussfolgernd wird ein Datensatz pro Modell benötigt. (vgl. zu diesem Abschnitt Goodfellow 2018, S. 109)

Es wurde zwei Datensätze erstellt. Der erste Datensatz dient zum Training eines Modells mit Fokus auf das Thema Videospiel „Kampfmusik“ (vgl. Kap. 2.3.5) und besteht nach Daten Augmentierung aus 2652 Proben mit jeweils 16 Takten. Der zweite Datensatz dient zum Training eines Modells mit Fokus auf das Thema Videospiel Überweltmusik (vgl. Kap. 2.3.6) und besteht nach Daten Augmentierung aus 3915 Proben mit jeweils 16 Takten.

4.2.1 Preprocessing

Zunächst müssen die Daten in ein für das neuronale Netz verarbeitbares Format gebracht werden. MIDI kann zwar auch von einem Netzwerk verarbeitet werden, wie vorgestellt am Beispiel MusicVAE (vgl. Kap. 3.2), für diese Arbeit wurde sich aber für eine Konvertierung in das Piano-Roll Format entschieden. Da für die Verarbeitung innerhalb des Netzwerks Dichte-Schichten verwendet werden, welches sich für Matrixmultiplikationen eignen, ist das matrixförmige Piano-Roll-Format sinnvoll. Schlussfolgernd wird aus einer Reihe an Nachrichten in MIDI (vgl. Kap. 2.2.3) eine Matrix bestehend aus Einsen und Nullen im Piano Roll-Format (vgl. Kap. 2.2.4).

Das Einlesen der MIDI Dateien findet in der Datei „Projekt Verzeichnis/src/loadMidis.py“ statt. Jede einzelne MIDI-Datei wird zunächst einzeln konvertiert durch die Funktion „Projekt Verzeichnis/src/utility/midi.py/midi_to_samples(fname)“. Dieser Prozess findet in mehreren Schritten statt (vgl. *Abbildung 16*).

Zunächst wird geprüft, ob eine MIDI-Datei mehrere Taktarten aufweist, falls ja, wird, zum Vermeiden weiterer Komplikationen im Trainingsvorgang, die Datei ignoriert. Als Nächstes soll die Taktart vereinheitlicht werden. Jeder Takt besteht, inspiriert vom Modell MuseGAN, aus 96 Zeitschritten (vgl. 3.1.6). Es stellt sich heraus das mit dieser Auflösung von Zeitschritten die meisten gängigen Taktarten möglich sind. Da die Zahl 96 viele Teiler besitz, sind bspw. die gängigen Taktarten $^{3}/_{4}$, $^{4}/_{4}$, $^{6}/_{8}$ und einige mehr umsetzbar.

Insgesamt wurde die Anzahl möglicher Noten, orientier an der Tastenanzahl eines Klaviers, auf 88 beschränkt, womit eine breite Frequenzvielfalt abgedeckt ist.

Im nächsten Schritt werden aus jeder Spur die MIDI-Nachrichten ausgelesen und zu jeder Note, Start und Endzeitpunkte gespeichert. Gleichzeitig werden MIDI-Dateien bestehend aus mehreren Spuren zu einer Spur zusammengelegt. Es gibt innerhalb des Modells keine separate Instrumentenunterscheidung. Alle Spuren werden als ein einzelnes musikalisches Instrument betrachtet. Zuletzt werden die Start- und Endinformationen der Noten ausgelesen und in jeweils 96(Zeitschritte)* 88*(Noten) Matrix Blöcke umgewandelt. Hiermit ist die Konvertierung in das Piano-Roll-Format vollendet.

Im Laufe des Trainings hat sich herausgestellt, dass, in der aktuellen Umsetzung der neuronalen Netzarchitektur, es für die trainierten Modellen schwer ist, die Dauer der Noten zu beachten und zufriedenstellende Ergebnisse zu liefern. Es wurde deshalb die Designentscheidung getroffen, die Notendauer der Noten zu ignorieren und somit bei der Konvertierung in das Piano-Roll-Format, die Notendauer auf jeweils einen Zeitschritt zu beschränken. Dadurch konnten bessere Ergebnisse beim Training der Modelle erzielt werden.

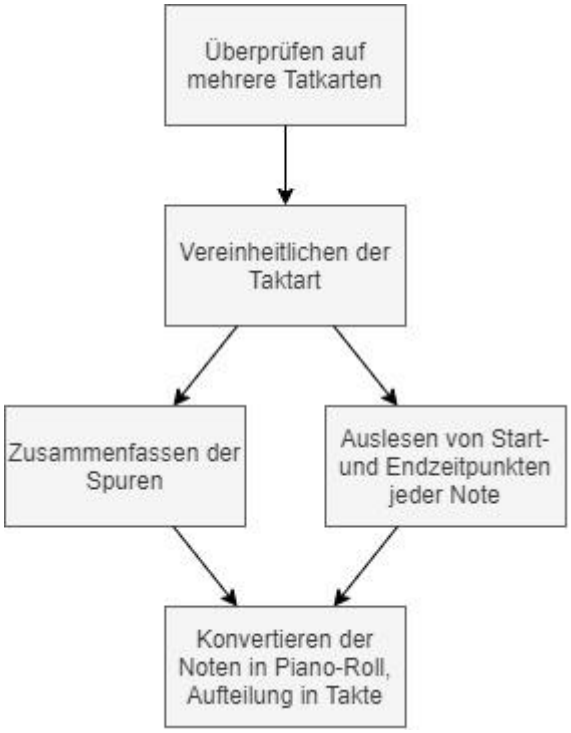


Abbildung 16: MIDI Konvertierungs-Prozess

Nach dem Auslesen und Konvertieren der MIDI-Dateien in Taktblöcke werden schließlich diese in mindestens achter-Blöcke unterteilt und in Form eines Arrays für die spätere Abrufung im neuronalen Netz zwischengespeichert.

Zuletzt dient „Projekt Verzeichnis>src>network>data.py“ dazu das die Daten in 16er Taktblöcke aufgeteilt werden. Damit ist die letztendliche Dimension der Datensatzeinträge für das neuronale Netz $16 \times 96 \times 88$.

4.2.2 Daten Augmentation

Daten Augmentation ist eine beliebte Technik, bei der die Datenmenge künstlich vergrößert wird mit dem Ziel besserer Trainingsergebnisse zu erzielen. Eine mögliche Technik der Daten Augmentation in der Musikproduktion ist das Transponieren (wechseln der Tonart) (vgl. Briot et al. 2020, 47-48 vgl.). Im Rahmen der Arbeit wurde diese Technik auf die Datensätze angewendet. Jedes Stück wird 6 Mal orientiert am Zentrum transponiert, wodurch

die ursprüngliche Datenmenge versechsfacht wird. Im Rahmen dieser Arbeit führte die Maßnahme zu einer verbesserten Qualität der Trainingsproben.

4.3 Modell

Letztendlich fiel die Entscheidung über die Architektur angesichts der in Kapitel 3.3 genannten Gründe auf den VAE. In diesem Unterkapitel wird erklärt wie der VAE, und dessen Komponenten, der Encoder und der Decoder, aufgebaut sind.

Für die Wahl der Schichten wurde eine alternative Lösung gewählt gegenüber zu den bereits vorgestellten CNN und RNN Lösungen. Es wurden Zeitverteilte-Schichten (engl. Time Distributed Layer) verwendet in Kombination mit Dichte-Schichten. Die Zeitverteilte-Schicht erlaubt die Anwendung einer Schicht (in diesem Fall ein Dichte Schicht) auf eine chronologisch aufgeteilte Sequenz von Daten. Wenn bspw. im Fall dieser Arbeit der Eingang aus 16 Sequenzen besteht, so wird die ausgewählte Schicht auf diese 16 Sequenzen angewendet und erzeugt für jede der Sequenzen einen Ausgang. Diese Art von Schicht kann z. B. auf die zeitlich hintereinander abgespielten Bilder eines Videos angewendet werden, lässt sich aber auch auf die zeitlich hintereinander abgespielten Takte eines Musikstücks übertragen. (vgl. zu diesem Abschnitt Patrice Ferlet 2019)

Die im Aufbau beschriebenen Parameter-Größen ergeben sich aus dem durchgeführten Parameter-Tuning, welches in Kap. 4.4 vorgestellt wird. Insgesamt besteht das neuronale Netz aus 54.040.582 Parametern, von denen 54.031.872 Parameter trainierbar und 8.710 nicht trainierbar sind. Abgesehen von der latenten-Schicht und der Ausgangsschicht wird auf die verdeckten Schichten die übliche „ReLU“-Aktivierungsfunktion angewendet (vgl. Goodfellow 2018, S. 192). Eine genaue Abbildung des Encoders und Decoders sind jeweils der Arbeit angehängt.

4.3.1 Encoder

Die Umsetzung des Encoders ist in der Datei „Projekt Verzeichnis/src/network/musicVAE.py“ aufzufinden.

Die Dimension des Encoder-Eingangs ist $16 \times 96 \times 88$. Eine einzelne Probe besteht aus 16 Takten. Zunächst werden die Takte des Eingangs zu einer einzelnen Dimension konvertiert (aus 96×88 wird 8448) mit einer Umform-Schicht (engl. Reshape Layer). Anschließend wird der Ausgang von einer Dichte-Schicht verarbeitet, welche eindimensionale Eingänge entgegennimmt. Als Nächstes folgen zwei Zeitverteile-Schichten mit zunächst einer Dichte-Schicht mit 2048 Einheiten im ersten und anschließend einer Dichte-Schicht bestehend aus 256 Einheiten in der zweiten Zeitverteilten-Schicht. Genauer beschrieben wird auf jedes der 16 Takte hintereinander zwei Dichte-Schichten angewendet. Danach folgt eine Abflachungs-Schicht (engl. Flatten-Layer), welche die Daten zu einer einzelnen Dimension abflacht und darauffolgend eine weiter Dichte-Schicht der Einheitsgröße 2048.

Zuletzt folgt die Umsetzung des latenten Bereichs (vgl. Kap. 2.2.6). Dafür werden zunächst zwei Schichten erstellt, eine dient als Mittelwert „mu“ und die zweite Schicht als Standardabweichung „sigma“ des Encoders (vgl. Versloot 2019). Mit Hilfe dieser Schichten ist es möglich, einen zufälligen Punkt als Stichprobe aus dem latenten Bereich zu ziehen. Für die Größe des latenten Raums wurde die Dimensionsgröße 256 festgelegt.

4.3.2 VAE-Sampling

Der in VAEs angewendete Umparametrisierungstrick kommt in der Klasse „Projekt Verzeichnis>src>network>musicVAE.py>Sampling“ zum Einsatz. Als Argumente werden dieser Funktion zum einen der Parameter „mu“ und zum anderen der Parameter „sigma“ übergeben. „Epsilon“ ist eine Normalverteilung in der Dimension des übergebenen Parameters „mu.“ Die Stichprobe wird aus dem latenten Raum gezogen über die Umparametrisierung der Probe. Die Klasse zur Realisierung des Samplings wird in *Abbildung 17* gezeigt.

```
class Sampling(layers.Layer):
```

```
    def call(self, inputs):
        mu, sigma = inputs
        epsilon = K.random_normal(shape=K.shape(mu))
        return mu + tf.exp(0.5 * sigma) * epsilon
```

Abbildung 17: Sampling-Schicht-Klasse des VAE

4.3.3 Decoder

Der Decoder ist umgesetzt in der Datei „Projekt Verzeichnis>src>network> musicVAE.py“

Die Eingangsdimension entspricht der Dimension des Latenten-Raums. Zunächst folgt gespiegelt, betrachtet zum Encoder, eine Dichte-Schicht der Einheitsgröße 2048. Anschließend folgt eine Dichte-Schicht mit der Größe $\text{max\_length} * 256$ bzw. im Anwendungsfall der Arbeit $16 * 256$. Darauf folgt eine Umform-Schicht, die den Eingang in die Dimension 16×256 formatiert. Als nächstes wird an die aktuelle Kette eine Zeitverteilungs-Schicht angehängt, die einen Dichte-Schicht der Größe 2048 auf die aktuellen 16 Dimensionen anwendet. Die Ausgangsschicht besteht aus einer Zeitverteilungs-Schicht mit einer Dichte-Schicht der Größe 96*88 und einer Umform-Schicht, der den Ausgang in das passende Ausgangs-Format $16 \times 96 \times 88$ (vgl. Kap. 3.2.2) konvertiert. Die Ausgangsschicht verwendet eine Sigmoid-Aktivierungsfunktion, wodurch für jede Stelle ein Wert zwischen 0 und 1 zurückgegeben wird.

Zwischen den Schichten wird Batchnormalisierung (Ioffe und Szegedy 2015) auf die Dimensionen angewendet, wodurch die Leistung des neuronalen Netz erheblich verbessert wird. Ebenfalls werden Ausfall-Schichten (engl. Dropout-Layer) (Nitish Srivastava et al. 2014) mit einer Ausfallrate von 0.1 eingesetzt um eine Überanpassung des Netzwerks zu vermeiden.

4.3.4 Verlust-Funktion

Die Verlustberechnung ist gegeben in „Projekt Verzeichnis/src/network/musicVAE.py/VAE.train_step(self, data)“

Die Verlustfunktion des VAE, auch ELBO genannt, setzt sich aus dem Rekonstruktionsterm, im Anwendungsfall die binäre Kreuzentropie und der KL-Divergenz zusammen, welche wie in Kap. 2.2.6 beschrieben dafür sorgt, dass der latente Raum reguliert ist, also kontinuierlich und vollständig bleibt.

Zusätzlich wird das Modell zu einem β-VAE (Higgins et al. 2016) erweitert. Das hat zur Folge, dass zur gewöhnlichen Verlustfunktion

$$\mathrm{L(q)} = \mathbb{E}_{z\sim q(z|x)} \log p_{model}(x|z) - D_{KL}(q(z|x)||p_{model}(z))$$

ein Hyperparameter β hinzugefügt wird, welcher dafür sorgen soll, das mehr des latenten Raums ausgenutzt wird. Für die Verlustfunktion gilt hiermit die Formel

$$\mathrm{L(q)} = \mathbb{E}_{z\sim q(z|x)} \log p_{model}(x|z) - \beta\, D_{KL}(q(z|x)\big|\big|p_{model}(z)\big).$$

(vgl. zu diesem Abschnitt Higgins et al. 2016)

Im Fall das β>1 führt diese Änderung zu einer Entwirrung des latenten Raums, wodurch effektiver Informationen aus dem Bereich gezogen werden können, unter eventuellen Einbußen bei der Rekonstruktion der Daten (vgl. Higgins et al. 2016).

Da der Fokus der Arbeit auf der Produktion neuer Daten und nicht auf der Rekonstruktion bestehender Daten liegt, ist diese Anpassung zielführender verglichen mit der gewöhnlichen ELBO. Es wurde ein Modell mit dem β-Wert 5 und ein Modell mit dem β-Wert 10 trainiert, wodurch Verbesserungen in den generierten Proben festgestellt werden konnten. *Abbildung 18* zeigt die technische Umsetzung der Verlustberechnung.

```
1. reconstruction_loss = tf.reduce_mean(
2.   tf.reduce_sum(
3.           keras.losses.binary_crossentropy(data, reconstruc-
   tion), axis=(1, 2, 3)
4.           )
5. )
6. kl_loss = -0.5 * (1 + sigma - tf.square(mu) - tf.exp(sigma))
7. kl_loss = tf.reduce_mean(tf.reduce_sum(kl_loss, axis=1))
8. total_loss = reconstruction_loss + (self.beta * kl_loss)
```

Abbildung 18: Code-Schnipse der Verlustberechnung

4.4 Hyperparameter-Tuning

Für die Wahl, passender Hyperparameter wurden verschiedene Kombinationen ausgesucht und getestet.

Jedes Modell wurde für 50 Epochen trainiert und deren Trainingsverlauf mit dem Tool TensorBoard (TensorFlow 2015) dokumentiert. Getestet wurde mit dem „Kampfmusik“-Datensatz (vgl. Kap. 4.2). Logdateien befinden sich im Ordner „Projekt Verzeichnis/parameter_tuning“. Die Werte in den späteren Tabellen werden gerundet.

Es wurden Folgende Parameter variiert,

- Batchgröße (B) – Die Größe der Teilmenge, welche jeweils durch das Netzwerk propagiert wird (vgl. Goodfellow 2018, S. 168).
- Lernrate (LR) – Die Schrittweiter, mit der das NN trainiert (vgl. Goodfellow 2018, S. 93).
- Latenter Raum Dimensionsgröße (LD) – Die Dimensionsgröße des latenten Raums.
- β – Der β-Wert für die ELBO

Zur Bewertung der getesteten Parameterpaare wurden folgende Werte in Betrachtung gezogen.

- KL-Divergenz (KLD) – Die Divergenz zwischen der erlernten Verteilung zur Standardnormalverteilung (vgl. Kap. 2.1.8).
- Rekonstruktions-Fehler (RF) – Der Fehler zwischen Eingang und rekonstruierten Ausgang.
- Verlust (V) – Die ELBO des VAE.

4.4.1 Dimensionspaare

Für das Hyperparameter Testen wurden die Dimensionspaare (DP) D1 [2000, 200, 1600], D2 [2048, 256, 2048] gewählt, wobei jede Dimension eine Hierarchiestufe des Netzwerks beschreibt

4.4.2 Bestimmung der Batchgröße

Zunächst wurden verschiedene Batchgrößen für ein Modell mit den Beispielparametern LR= 0.001, LD= 200, β=1 und DP= D1 getestet (vgl. *Tabelle 1*). Neben den zuvor beschriebenen Messwerten wurde zusätzlich die Trainingsdauer (D) in Minuten betrachtet.

[Tabelle mit Batch = 100, 128, 200, 256]

Modell Nr.	*B*	*KLD*	*RF*	*V*	*D*
1	100	48,56	2622,9	2671,45	8:10
2	128	51,62	3591,57	3643,19	8:42
3	200	61,1	5902,51	5963,60	7:51
4	256	69,69	7522,73	7592,43	7:56

Tabelle 1: Testen der Batchgröße

Nach der Auswertung der Werte wurden mit der Batchgröße von 128 zufriedenstellende Werte erreicht, ohne dabei in der Trainingsdauer großartige Verluste zu aufzuweisen. Auf Grundlage dieser Erkenntnisse wurde in den restlichen Testdurchgängen sich die für die Batchgröße 128 entschieden.

4.4.3 Bestimmung restlicher Parameter

Die Übersicht der Trainierten Modelle wird in *Tabelle 2* gezeigt.

Modell Nr.	*LR*	*LD*	*β*	*DP*	*KLD*	*RF*	*V*
1	0,001	200	5	D1	14,07	3778,41	3848,75
2	0,00075	200	5	D1	14,45	3800,81	3873,04
3	0,0005	200	5	D1	10,93	3982	4036,63
4	0,001	128	5	D1	11,96	3824,36	3884,18
5	0,001	256	5	D1	14,14	3809,99	3880,71
6	0,00075	256	5	D1	13,58	3846,57	3914,57
7	0,001	256	5	D2	15,67	3719,56	3797,9

8	0,00075	256	5	D2	14,6	3816,54	3889,53
9	0,001	256	10	D2	9,19	3911,53	4003,45
10	0,00075	256	10	D2	8,67	3922,11	4008,85
11	0,001	512	10	D2	9,45	3902,34	3996,86
12	0,00075	512	10	D2	9,21	3977,62	4069,73

Tabelle 2: Testergebnisse Parametersuche

Nach einer zusätzlichen Betrachtung der Trainingsverlauf-Graphen der Modelle mit dem Tool TensorBoard fiel die Entscheidung für die Hyperparameter auf die von Modell Nr. 10. Das getestete Modells schien gut mit einer Standardnormalverteilung zu konvergieren und dabei akzeptable Rekonstruktionsfehler-Wert zu erzielen.

4.5 Training

Für das Training der Modelle wurde die beste Parameterkombinationen aus dem Hyperparameter-Tuning ausgewählt. Mit diesen Parametern wurden Modelle trainiert auf Basis der beiden zusammengestellten Datensätze. Die endgültigen Parameter sind B= 128; LR= 0.00075; LD= 256 und DP= [2048, 256, 2048].

Um den Nutzen des β-Parameters abzuwägen wurde für jeden Datensatz jeweils einmal mit dem β-Wert 5 und einmal mit dem β-Wert 10 trainiert.

Die Modelle wurden jeweils für 2000 Epochen trainiert. Das Training des NN wurde in der Datei „Projekt Verzeichnis/src/train.py“ umgesetzt. Zu bestimmten Epochen-Intervallen werden zehn Stichproben genommen, somit ist der Lernprozess des NN besser nachvollziehbar. Zusätzlich werden die Mittelwerte, die Standardabweichungen und die Auswertung des latenten Raums, mit den nach Wichtigkeit sortierten Eigenschaften, dokumentiert.

Darstellungen von generierten Proben sind angehängt.

4.6 Umsetzung der Metriken zur Auswertung

Im Kapitel 3.2 wurden bereits die Metriken vorgestellt, mit denen das Modell MuseGAN ausgewertet wurde. Im Rahmen der Auswertung wurde sich an diesen vorgestellten Metriken orientiert. Dieses Unterkapitel legt näher, welche objektiven Metriken für diese Arbeit verwendet werden und wie diese umgesetzt wurden. Die Funktionen für die Berechnung der Metriken befinden sich in „Projekt Verzeichnis/src/utility/metrics.py."

Die meisten Funktionen der Auswertung erfordern die Übergabe eines Grenzwerts (engl. threshhold) welcher aussagt wie sicher sich das Netwerk für diese Stelle sein muss, dass die Stelle als eine gespielte Note erkannt wird. Der Wert 0,25 erwies sich als geeignet für eine qualitative Auswertung der Proben.

Aufgrund der Architektur des Netzwerks wird die QN Metrik (vgl. Kap. 3.2.3) bei der Umsetzung ausgeklammert, da innerhalb des Netzwerks keine Notenlängen betrachtet werden, ist diese Metrik nicht auf das Netzwerk übertragbar.

4.6.1 Average Empty Bars (AEB)

Die Funktion „evaluate_eb(data_set, thresh)" ruft für jedes Musikstück die Funktion „empty_bars(sample, thresh)" auf welche für ein Stück das Verhältnis der leeren Takte auswertet. Ein Takt gilt dann als leer, wenn alle seine Stellenwerte unterhalb des Grenzwerts liegen. Aus den Resultaten aller Musikstücke wird der Mittelwert berechnet und als Rückgabewert zurückgegeben. Die Metrik liegt im Intervall $\{0 \leq x \leq 1\}$.

4.6.2 Average Drum Pattern (ADP)

Die Funktion „evaluate_dp(data_set, thresh)" berechnet das durchschnittliche Verhältnis aller Noten, welche in einem $1/8$ oder $1/16$ Rhythmus vorkommen, zum Verhältnis aller gespielten Noten, eines Datensatzes. Zur Berechnung wird jeweils pro Musikstück die Funktion „drum_pattern(sample, thresh)" auf-

gerufen, welche alle Noten in einem Drum Pattern und alle erkannten Noten eines Musikstücks zurückgibt.

Da ein Takt aus 96 Zeitschritten besteht, können alle Noten zum Zeitschritt i als Teil eines Drum Patterns betrachtet werden, wenn $i\%6 = 0$ gilt. Zuletzt ergibt sich die Metrik aus dem Verhältnis aller Drum-Pattern-Noten zum Verhältnis aller erkannten Noten. Der zurückgegebene Wert liegt im Intervall $\{0 \leq x \leq 1\}$.

4.6.3 Average Polyphonicity (APP)

Das durchschnittliche Verhältnis von Zeitschritten zu denen mehr als eine Note gespielt wird zu Zeitschritten zu denen mind. eine Note gespielt wird, berechnet die Funktion „evaluate_polyphonicity(data_set, thresh)“. Hierzu wird für jedes Element des Datensatzes die Funktion „polyphonicity(sample, thresh)“ aufgerufen. Die Funktion gibt für jeweiliges Element die Anzahl an Zeitschritten mit mehr als einer Note und die Anzahl an Zeitschritten mit mind. einer Note zurück. Das Ergebnis der gesamten Berechnung ist die APP.

Diese Metrik unterscheidet sich zu der Ursprungsmetrik beschrieben in Kap. 3.4.6 darin, dass die Metrik relative zu Zeitschritten mit Notenvorkommen und nicht relative zur Anzahl der Zeitschritte betrachtet wird. Die APP liegt im Intervall $\{0 \leq x \leq 1\}$.

4.6.4 Average Used Pitch Classes (AUPC)

Diese Metrik wird in der Funktion „evaluate_upc_average(data_set, thresh)“ berechnet. Für jedes Musikstück des Datensatzes wird die Funktion „get_upc(samples, thresh)“ aufgerufen, welche berechnet, wie viele verschiedene Frequenzklassen im Durchschnitt innerhalb eines Taktes auftreten. Als Frequenzklassen gelten die Noten von C-H, demnach gibt es insgesamt 12 Frequenzklassen.

Aus den Berechnungen jedes Musikstücks wird der Durchschnitt des gesamten Datensatzes berechnet und als Rückgabewert der Funktion zurückgegeben. Der Rückgabewert ist im Intervall $\{0 \leq x \leq 12\}$.

4.6.5 Average Tonal Distance (ATD)

Die ATD ist eine inter-Spur Metrik, dementsprechend werden 2 Spuren gegenübergestellt und dessen harmonische Ähnlichkeit verglichen. Die Berechnung der TD zwischen 2 Spuren findet statt in der Funktion " tonal_distance(samples1, samples2, ignore_treshhold, thresh)".

Die Berechnung der Metrik baut auf der Grundlage auf, dass die Noten-Frequenz-Beziehungen innerhalb eines harmonischen Netzwerks bzw. Tonnetzes dargestellt werden können. (vgl. zu diesem Abschnitt Harte et al. 2006)

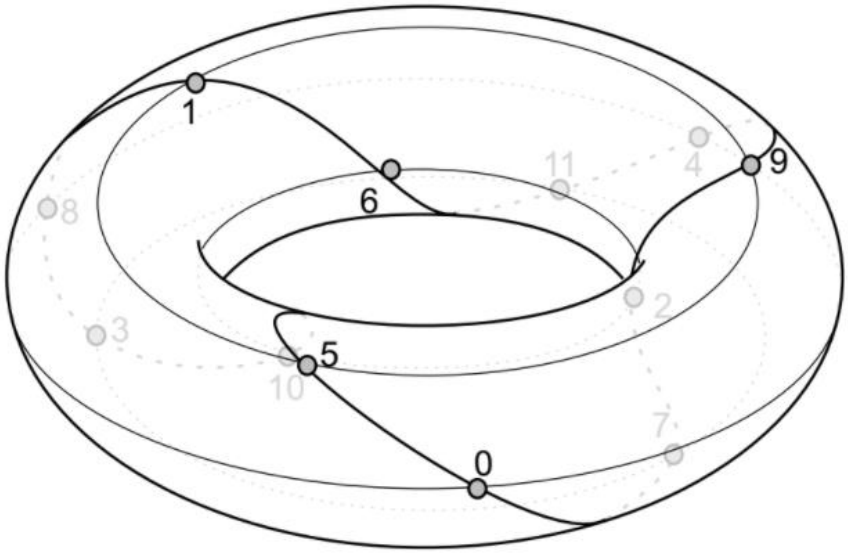


Abbildung 19: Darstellung des Tonnetz in vor eines Hypertorus (Quelle: Harte et al. 2006)

Wenn das Tonnetz in Form eines Hypertorus dargestellt wird (vgl. *Abbildung 19*), so formt der Quinten-Zirkel eine Spirale, welche sich drei Mal um die Oberfläche des Hypertorus windet. Betrachtet werden die 12 Frequenzklas-

sen der reinen Stimmung, schlussfolgernd wird die Oktave einer Note als die gleiche Note betrachtet. (vgl. zu diesem Abschnitt Harte et al. 2006)

Als Nächstes kann dieser Hypertorus in einen 6-dimensionalen Raum umgewandelt werden in, welchem es möglich ist, die Wirkungszentren von Frequenzbeziehungen darzustellen und Akkorde eindeutig zu definieren. (vgl. zu diesem Abschnitt Harte et al. 2006)

Der 6-dimensionale Raum kann in der Vorstellung als drei Kreise betrachtet werden. Der erste Kreis ist der Quinten-Zirkel, der zweite Kreis der große Terz Zirkel und der letzte Kreis der kleine Terz-Zirkel (vgl. *Abbildung 20*). Die 6-Dimensionen lassen sich in drei Koordinatenpaare unterteilen. Jedes Koordinatenpaar beschreibt jeweils einen Punkt innerhalb eines der 3 Kreise. Hiermit kann ein Akkord, besteht aus mehreren Frequenzen, eindeutig als Schwerpunkt innerhalb des 6-dimensionalen Raums eingeordnet werden. (vgl. zu diesem Abschnitt Harte et al. 2006)

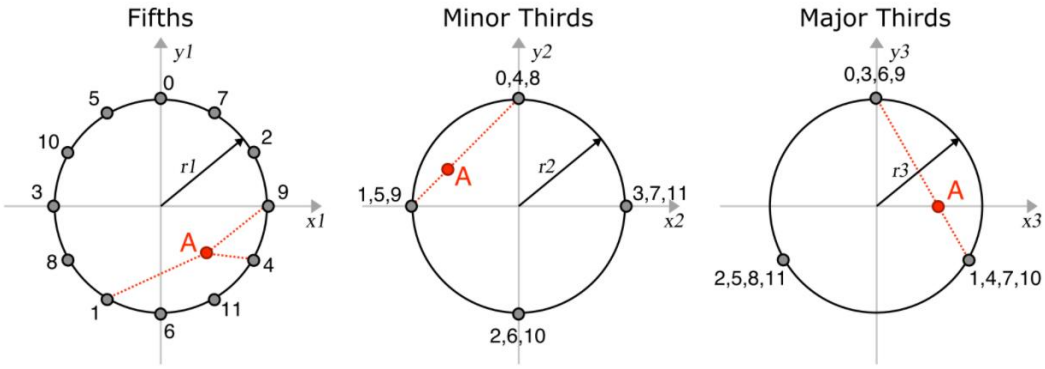


Abbildung 20: 6-D Raum dargestellt als drei Kreise. (Quelle: Harte et al. 2006)

Technische Umsetzung

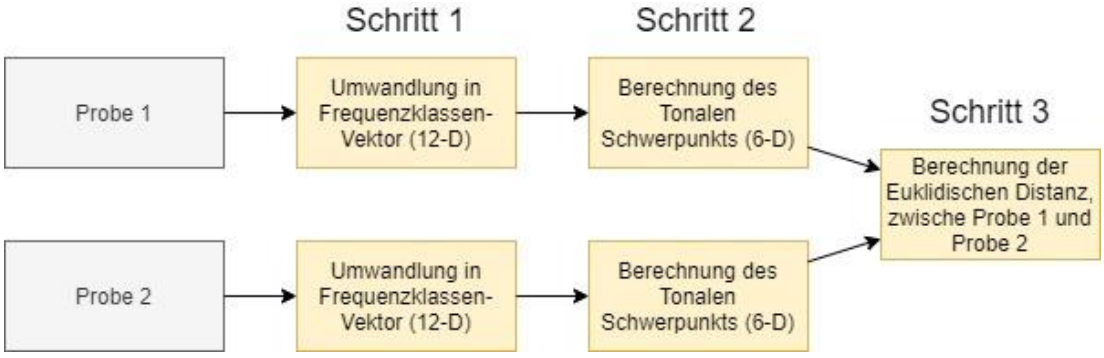


Abbildung 21: Schritte zur Berechnung der TD

Die Berechnung der Tonalen Distanz lässt sich in 3 Schritte unterteilen (vgl. *Abbildung 21*). Zunächst folgt die Umwandlung der Noten einer Probe in die Frequenzklassen durch die Funktion „samples_to_pitchclasses(samples, ignore_treshhold, thresh)“. Dementsprechend wird ein Datensatz in der Dimension $16 \times 96 \times 88$ umgeformt in die Dimension $16 \times 96 \times 12$.

Im zweiten Schritt folgt die Umwandlung in die 6-dimensionale Tonale Schwerpunkts Darstellung, von einer Dimension der Größe $16 \times 96 \times 12$ in die Dimension $16 \times 96 \times 6$. Dies Geschieht in der Funktion „get_t_c_samples(samples)“. Der Tonale Schwerpunkt ζ des Zeitschritts n wird mit der Formel

$$\zeta n(d) = \frac{1}{\|c_n\|_1} \sum_{l=0}^{11} \Phi(d,l) c_n(l)$$

$$0 \leq d \leq 5;\ \ 0 \leq l \leq 11$$

berechnet (vgl. Harte et al. 2006). C ist der zuvor berechnete Chromavektor der Dimension 12 und Φ die Transformationsmatrix

$$\Phi_l = \begin{pmatrix} r_1 \sin l\frac{7\pi}{6} \\ r_1 \cos l\frac{7\pi}{6} \\ r_2 \sin l\frac{3\pi}{2} \\ r_2 \cos l\frac{3\pi}{2} \\ r_3 \sin l\frac{2\pi}{3} \\ r_3 \cos l\frac{2\pi}{3} \end{pmatrix}; 0 \leq l \leq 11,$$

implementiert in der Funktion „get_transformation_matrix(l).“ Da der 6-D Vektor in Verbindung mit der Wahrnehmung des Menschen stehen soll, ist es wichtig, dass diese Darstellung die harmonischen Beziehungen zwischen Noten darstellt (eine Quinte ist die stärkste harmonische Beziehung, danach eine große Terz, usw.). Für $r1$, $r2$ und $r3$, werden die Werte 1, 1 und 0,5 eingesetzt. Die Herleitungen dieser Werte entstammen dem „Spiral Array Model“ von Chew (2000). (vgl. zu diesem Abschnitt Harte et al. 2006)

Nachdem die beiden Proben jeweils in ihren Tonalen Schwerpunkte umgewandelt wurden, folgt im letzten Schritt die Berechnung des Tonalen Unterschieds zwischen den Proben. Die tonale Distanz, ausgedrückt als Euklidische Distanz von Probe $S1$ zur Probe $S1$ zum Zeitpunkt n, wird mit der Formel

$$\xi_n \sqrt{\sum_{d=0}^{5} [S1_n(d) - S2_n(d)]^2}$$

Berechnet (vgl. Harte et al. 2006). Nachdem dieser Schritt für jeden Zeitschritt n angewendet wurde, werden die Ergebnisse aufsummiert und durch die Anzahl an Zeitschritten geteilt. Man erhält damit eine durchschnittliche Tonale Distanz pro Zeitschritt zwischen 2 Proben, welcher schließlich als Rückgabewert der Funktion dient.

Beim Aufruf der Funktion „evaluate_tonal_distance(data_set1, data_set2, thresh)“ gibt es zwei Möglichkeiten. Die erste Option ist, dass die Funktion

zwei Datensätze erhält. In diesem Fall wird Datensatz 1 und Datensatz 2 miteinander verglichen und die ATD ausgerechnet. Die zweite Option ist, dass ein Datensatz übergeben wird und innerhalb des Datensatzes ein Vergleich stattfindet. Neben der ATD wird zusätzlich der niedrigste Durchschnittswert, der unter allen Vergleichen erreicht wurde, zurückgegeben. Somit kann nach einem GAN ähnlichen Mode Collapse geprüft werden (vgl. Kap. 3.3.1). Der Wert der ATD liegt im Intervall $\{0 \leq x \leq 1\}$.

4.6.6 Average Note Count (ANC)

Die Funktion „evaluate_notes_per_song(data_set, thresh)“, berechnet, wie viele Noten im Durchschnitt pro Song gespielt werden. Hierbei wird die Funktion „get_note_count(song, thresh)“ aufgerufen, welche die Notenanzahl eines Songs zurückgibt

5 Auswertung

5.1 Höreindruck

In den Proben der trainierten Modelle sind klare musikalische Strukturen erkennbar. Die akustische Evaluation stützt sich auf den Konzepten die in „Deep Learning Techniques for Music Generation“ (Briot et al. 2020, S. 25–28)beschrieben sind und beruht auf eigenen Beobachtungen.

Die Proben der erstellten Modelle weisen erkennbare Rhythmen auf. In so gut wie allen generierten Beispielen sind klare Strukturen erkennbar. Auffällig ist, dass die Takte innerhalb eines einzelnen Lieds starke harmonische und rhythmische Ähnlichkeiten aufweisen. Es ist somit eine Songstruktur erkennbar, auch wenn diese meist einfach gehalten ist.

Innerhalb der „Kampfmusik“-Proben sind Tendenzen zur Dissonanz und überwiegen Moll-Tonleitern heraushörbar (vgl. Kap. 2.3.5). In den Beispielen der „Überweltmusik“-Proben hingegen Variieren die Beispiele untereinander zwischen Moll- oder Dur-Tonleiter (vgl. Kap. 2.3.6).

Die Proben weißen eine erkennbar starke thematische Struktur auf und kein Takt weicht stark vom Folgetakt ab. Diese Feststellung gilt vermindert für den Rhythmus und verstärkt für die Harmonie eines Stücks.

Varianzwechsel innerhalb einer Probe zwischen Takten sind nahezu ausschließlich rhythmischer Natur. Interessanterweise ist anzumerken, dass Rhythmuswechsel überwiegend nach zwei oder vier Taktintervallen einsetzen.

In den Proben ist eine Melodie erkennbar, welche sich meist im Taktintervall wiederholt. Es sind kein Akkordwechsel zwischen Takten hörbar wie bspw. in Pop- und Rocksongs. Eine mögliche Erklärung hierfür ist das Anfangs Melodie-, Begleitungs-, Bass- und Rhythmusspuren im Preprocessing der Datensätze (vgl. 4.2.1) zusammengelegt werden. Da Rhythmus die wohl prominenteste Eigenschaft ist, welche sich abgesehen von in den Rhythmusspu-

ren auch in Melodie- und Begleitspuren widerspiegelt, wird dieser Aspekt prominenter erlernt als bspw. ein Akkordwechsel. Ein Modell welches Rhythmus und Harmonie separat betrachtet könnte womöglich dazu in der Lage sein, einen Akkordwechsel zwischen Takten zu erlernen.

Zusammenfassend generieren die Modelle Proben, die als Musik wiedererkennbar sind und ein gewisses Niveau an Qualität aufweisen.

5.2 Rahmenbedingungen der Auswertung

Pro Modell wurden 100 zufällige Proben generiert und anhand der im Kapitel 4.6 beschriebenen Metriken ausgewertet. Zusätzlich wurden zum Vergleich dieselben Metriken für beide Trainingsdatensätze berechnet.

Aus dem Trainingsdatensatz wurden jeweils die ersten 16 Takte eines Musikstücks selektiert. Dadurch kann einerseits die Menge der Vergleichsdaten auf ein Minimum reduziert werden und andererseits vermieden werden, dass zwei Teile eines gleichen Musikstücks verglichen werden, wodurch die Chance steigt, dass die ATD Metrik irreführende Ergebnisse liefert.

Der Auswertungsprozess wird in zwei Phasen aufgeteilt.

5.2.1 Phase 1

In der ersten Phase wurden zunächst für Proben- und Trainings-Datensätze alle Metriken abgesehen von der ATD berechnet. Die Ergebnisse der jeweiligen Probe-Datensätze wurden, mit denen, der Trainingsdatensätze verglichen. Die Umsetzung der 1. Phase befindet sich in „Projekt Verzeichnis/src/evaluation_part_1.py“. Mit dem Probendatensatz, dessen Werte näher am Trainingsdatensatz liegen, wird in Phase 2 der Auswertung weiterverfahren.

5.2.2 Phase 2

In Phase 2 werden die ATDs innerhalb der jeweiligen Datensätze, zwischen Trainings- und Proben-Datensatz und zwischen den beiden Trainingsdaten-

sätzen berechnet. Die Umsetzung der 2. Phase befindet sich in „Projekt Verzeichnis/src/evaluation_part_2.py“

Da die Berechnung der ATD ein aufwendiger Prozess ist, werden innerhalb des Trainingsdatensätze die ersten 100 Einträge ausgewählt und dies miteinander verglichen. Zusätzlich wird pro Datensatz die minimal erreichte ATD (Min. ATD) hinterlegt.

Der Vergleich zwischen den beiden Trainingsdatensätzen findet in zehn Iterationen statt. Für die ersten fünf Iterationen werden vom zweiten Datensatz zehn zufällige Proben entnommen und mit dem ersten Datensatz verglichen. Anschließend werden in den nächsten fünf Iterationen zehn zufällige Proben des ersten Datensatzes entnommen und mit dem zweiten Datensatz verglichen. Anschließen wird aus allen Zwischenergebnissen der Mittelwert berechnet und das Minimum (Min. ATD) innerhalb des gesamten Durchgangs entnommen.

Im Vergleich der Trainingsdatensätze mit den Probedatensätzen werden in fünf Iterationen jeweils zehn zufällige Proben aus dem Probedatensatz entnommen und mit dem Trainingsdatensatz verglichen. Es wird das Minimum und der Mittelwert der Zwischenergebnisse entnommen.

5.3 Ergebnisse

Die Ergebnisse der Auswertung wurden im Ordner Projekt Verzeichnis/evaluation/evaluation_results“ gespeichert.

5.3.1 Ergebnisse Phase 1

In diesem Kapitel werden die Testergebnisse der ersten Auswertungsphase vorgestellt. „Modell 1“ beschreibt jeweils das Modell für den Fall β=10 und „Modell 2“ das mit dem Fall β=5.

In *Tabelle 3* werden die Ergebnisse der ersten Auswertungsphase der Probe- und Trainingsdatensätze für das Modell „Kampfmusik“ aufgelistet.

Datensatzname	*AEB*	*ADP*	*APP*	*AUPC*	*ANC*

Trainings-Datensatz	0,016	0,824	0,706	6,959	679
Proben Modell 1	0,015	0,947	0,703	7,264	583
Proben Modell 2	0,005	0,893	0,642	7,951	685

Tabelle 3: Phase 1, Kampf

In *Tabelle 4* werden die Ergebnisse der ersten Auswertungsphase der Probe- und Trainingsdatensätze für das Modell „Überweltmusik“ aufgelistet.

Datensatzname	*AEB*	*ADP*	*APP*	*AUPC*	*ANC*
Trainings-Datensatz	0,019	0,791	0,601	5,848	431
Proben Modell 1	0,016	0,902	0,578	5,621	283
Proben Modell 2	0,006	0,909	0,620	6,691	443

Tabelle 4: Phase 1, Überwelt

In beiden Modellen zeigt sich das die ADP verglichen mit dem Trainingsdatensatz erhöht ist. Eine mögliche Erklärung hierfür könnte sein, dass die ADP eine Beobachtung im Trainingsdatensatz beschreibt, welche durch seine Häufigkeit mehrere Dimensionen des latenten Raums belegt und sich somit in den Proben angehäuft als Eigenschaft widerspiegelt.

In den Ergebnissen zeigt sich, dass, abgesehen von der ANC, die Metriken der ersten Modelle überwiegend näher an den Werten der jeweiligen Trainingsdatensätze liegen.

Es ist nicht klar ersichtlich, ob die Diskrepanz zwischen der ANC und den restlichen Metriken der Modelle ein Indiz für einen erhöhten Noise-Faktor innerhalb jeweiligem „Modell 2“ sein könnte oder ob eine mögliche andere Schwäche des Modells aufgewiesen wird. Eine Hörvergleich der Modelle weist jedoch auf ersteres hin. In den Proben der 2. Modelle besteht eine grö-

ßere Chance, fragwürdige Ergebnisse zu hören als in jeweiligen „Modell 1“. Die Proben von „Modell 1“ scheinen konsistenter in ihrer Qualität zu sein.

5.3.2 Ergebnisse Phase 2

Auf Basis der Ergebnisse in Phase 1, wurde in der zweiten Auswertungsphase für „Modell 1 Kampfmusik“ und „Modell 1 Überweltmusik“ die Proben weiter ausgewertet.

Zunächst wurden die **ATDs** und **Min ATDs** innerhalb der Datensätze berechnet (vgl. *Tabelle 5*).

Datensatz Beschreibung	*ATD*	*Min. ATD*
Trainingsdatensatz „Kampf“	0,378	0,146
Probendatensatz „Kampf“	0,266	0,029
Trainingsdatensatz „Überwelt“	0,286	0,074
Probendatensatz „Überwelt“	0,19	0,015

Tabelle 5: Phase 2, Berechnung der ATDs innerhalb der Datensätze

Aus den ersten Ergebnissen der Phase 2 lässt sich erkennen, dass die ATD-Werte der Trainingsdatensätze höher liegen als die der Probedatensätze. Schlussfolgernd sind die Stichproben untereinander ähnlicher als die Musikstücke des Trainingsdatensatzes. Allgemein sind die ATD Werte der „Kampfmusik“-Datensätze höher als die der „Überweltmusik“-Datensätze.

Der erreichte Minimalwert bezogen auf den „Überwelt“-Probendatensatz ist ebenso auffällig, wie der Minimalwert des „Kampf“-Probendatensatz. Die niedrigen Werte weisen darauf hin, dass es innerhalb der Modelle, Proben gibt, die harmonisch starke Ähnlichkeiten aufweisen. Aufgrund der Architektur des VAE, ist hierbei schwer auszusagen, ob die Ursache dafür ist, dass zwei benachbarte Vektoren innerhalb des latenten Raums selektiert wurden oder ob allgemein Punkte innerhalb des latenten Raums starke Ähnlichkeiten

nach der Decodierung aufweisen. Die hohe Diskrepanz zwischen Minimalwert und Mittelwert lässt jedoch ersteres andeuten.

In den Stichproben des „Kampfmusik“-Modells gibt, es mehr harmonische Varianz als in den Stichproben des „Überwelt“ Modells.

Nach der Gegenüberstellung der ersten Ergebnisse von Phase 2 wurden anschließend die ATDs und Min ATDs zwischen verschiedenen Datensätzen berechnet (vgl. *Tabelle 6*).

Beschreibung	*ATD*	*Min. ATD*
Vergleich zwischen Trainingsdatensatz „Kampf“ und Trainingsdatensatz „Überwelt	0,334	0,113
Vergleich zwischen Trainingsdatensatz „Kampf“ und Probendatensatz „Kampf“	0,23	0,047
Vergleich zwischen Trainingsdatensatz „Überwelt“ und Probendatensatz „Überwelt“	0,304	0,037

Tabelle 6: Berechnung der ATDs zwischen verschiedenen Datensätzen

Zwischen „Kampf“-Trainingsdatensatz und „Überwelt“-Datensatz wurde der höchst ATD-Wert erreicht. Diese Feststellung deckt sich mit der Annahme, dass zwei Trainingsdatensätze gewählt wurden, welche verschiedene Unterkategorien der Videospielmusik darstellen. Interessanterweise ist der Mittelwert innerhalb des „Kampf“-Trainingsdatensatz größer als der Mittelwert zwischen dem Trainingsdatensatz-Vergleich. Folglich weist der „Kampf“-Trainingsdatensatz selbst hohe Varianz auf.

5.3.3 Ergebnis Interpretation

Beide Modelle, das „Überweltmusik“-Modell stärker, weisen eine ATD zum Trainingsdatensatz auf, die klein genug ist, sodass die generierten Proben möglicherweise Teil des Trainingsdatensatzes sein könnten, aber groß genug, sodass die Proben Variation gegenüber dem Trainingsdatensatz auf-

weisen. Gepaart mit den Ergebnissen aus Phase 1, wo erkenntlich ist das die Werte sich dem Trainingsdatensatz annähern, werden diese Erkenntnisse der 2. Phase weiter gestützt.

Das VAE scheint seine Aufgabe der Varianz und Annäherungskontrolle (vgl. Kap. 2.1.8) der erlernten Verteilung erfolgreich erfüllt zu haben. Das Modell hat einen geordneten latenten Raum erstellt, mithilfe dessen Aussagen über die Eigenschaften des Trainingsdatensatzes getroffen werden können. Zusätzlich kann der Raum für die Generierung originärer Musik eingesetzt werden kann.

6 Ausblick und Fazit

Im Rahmen dieser Arbeit wurden zwei Modelle erstellt. Die Modelle wurden als VAE konzipiert. Das erste Modell erlernte die Verteilung eines Datensatzes mit dem Fokus auf Kampfmusik in Videospielen. Das zweite Modell erlernte die Verteilung eines Datensatzes mit Fokus auf Überweltmusik in Rollenspielen. Für beide Modelle können zufällige Punkte aus der Modellverteilung entnommen werden, die schließlich im Decoder zu Daten mit musikalischem Inhalt verarbeitet werden.

Darüber hinaus wurde ein Auswertungsverfahren konzipiert, durch das Schlüsse über die generierten Proben der beiden Modelle gezogen werden konnten.

In der Auswertung war feststellbar das die Proben untereinander eine akzeptable harmonische Distanz () aufweisen und stark genug von ihrem jeweiligen Datensatz abweichen. Trotz dessen besteht eine gewisse harmonische Nähe zwischen Trainings und Probedaten.

Die Beispiele weisen rhythmische und harmonische Strukturen auf, die als Musik wiedererkennbar sind.

Im Bereich des Deep Learnings gibt es die beiden aktuellen generativen Modelle, GAN und VAE, wobei sich im Rahmen dieser Arbeit für den VAE entschieden wurde. VAEs sind bekannt dafür mit beschränkten Umsetzungsaufwand gute Ergebnisse zu liefern. GANs hingegen können schwierig zu trainieren sein. Außerdem sind für GAN große Datenmengen von Vorteil. Im Bereich der Videospiele ist es allerdings nicht unbedingt leicht große Datensätze zusammenzustellen. Rückblickend erwies sich daher die Entscheidung für das VAE als sinnvoll. Die Modelle konnten zufriedenstellende Musikproben erstellen, welche überraschend gut klingen.

Die Ergebnisse sind gut aber verbesserungsfähig. Bspw. wurde die Erkenntnis gemacht, dass zwar Varianz zwischen verschiedenen Musikstücken besteht, die Stücke in sich aber relativ monoton klingen. Das gilt vor allem für

die harmonische Vielfalt zwischen Takten. Wenn man diese Problematik auf den Kontext der Videospiele überträgt, könnte die Musik in einer langen Spielpassage schnell eintönig werden, falls immer die gleiche Musikschleife abgespielt wird.

An diesem Problem lässt sich ansetzten. In Kap. 5.1 wurde bereits angesprochen, dass durch Hinzufügen einer zusätzlichen Hierarchiestufe, das Problem der harmonischen Vielfalt gelöst werden könnte. Die zusätzliche Hierarchiestufe könnte Harmonie und Rhythmus unterteilen. Die Schwierigkeit dabei ist allerdings das durch diese Änderung die Anzahl der Gewichte gewaltig wachsen wird. Möglicherweise wären andere Zwischenschichten wie z. B. die Faltungs-Schichten eines CNN hierbei sinnvoller. Die Lösung der qualitativen Verbesserung des Modells ist ein Projekt, welches viel Zeit in Anspruch nehmen sollte. Der Ansatz könnte allerdings zu besseren Ergebnissen führen.

Hiermit wurde die qualitative Verbesserung des Modells angesprochen. Es wäre ebenfalls möglich an der praktischen Anwendung des Modells anzusetzen. VAE sind vielseitig einsetzbar und der Fakt, dass eine Verteilung explizit erlernt wird, weckt die Frage, ob diese Verteilung auch in anderer Form nutzbar ist.

In Kap. 2.3.5 wurde angesprochen, dass Musikstücke in Kampfmusik oft nach Wichtigkeit des Kampfes unterteilt werden. Diesen Ansatz könnte man ausbauen mit der Überlegung der Echtzeitanpassung der generierten Musik. Interessant wäre herauszufinden, ob es möglich ist, durch eine Analyse der Verteilung des latenten Raums, Eigenschaften wie z. B. Rhythmen auf Punkte oder Dimensionen innerhalb des latenten Raums festzulegen. In diesem Fall wäre die Echtzeitanpassung der Musik nur noch ein einfaches Wandern auf dem Gradienten der Modellverteilung. Die Musik könnte somit dynamisch und intuitiv an das Spielverhalten des Spielers angepasst werden.

Es wird in Zukunft interessant sein zu verfolgen, welche weiteren bahnbrechenden Neuerung die KI-Forschung bieten wird.

Literaturverzeichnis

Anaconda (2012): Anaconda. Online verfügbar unter https://www.anaconda.com/, zuletzt geprüft am 16.02.2021.

Arjovsky, Martin; Bottou, Léon (2017): Towards Principled Methods for Training Generative Adversarial Networks. Online verfügbar unter http://arxiv.org/pdf/1701.04862v1.

Arjovsky, Martin; Chintala, Soumith; Bottou, Léon (2017): Wasserstein GAN. Online verfügbar unter http://arxiv.org/pdf/1701.07875v3.

Bateman, Chris Mark; Boon, Richard (2006): 21st century game design. Hingham, Mass.: Charles River Media.

Bertin-Mahieux, Thierry; Ellis, Daniel P. W.; Whitman, Brian; Lamere, Paul (2011): The Million Song Dataset. DOI: 10.7916/D8NZ8J07.

Briot, Jean-Pierre; Hadjeres, Gaëtan; Pachet, François (2020): Deep Learning Techniques for Music Generation. Cham: Springer Nature Switzerland (Computational Synthesis and Creative Systems).

Chew, Elaine (2000): Towards a mathematical model of tonality. Dissertation, Cambridge, Massachusetts. Massachusetts Institute of Technology.

CodeParade (2018): Generating Songs With Neural Networks (Neural Composer). Online verfügbar unter https://www.youtube.com/watch?v=UWxfnNXlVy8, zuletzt geprüft am 16.02.2021.

Dong, Hao-Wen; Hsiao, Wen-Yi; Yang, Li-Chia; Yang, Yi-Hsuan (2018): MuseGAN. Multi-Track Sequential Generative Adversarial Networks for Symbolic Music. The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18). Research Center for Information Technology Innovation, Academia Sinica, Taipei, Taiwan; Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan. Online verfügbar unter https://salu133445.github.io/musegan/pdf/musegan-aaai2018-paper.pdf.

Dong, Hao-Wen; Yang, Yi-Hsuan (2018): Convolutional Generative Adversarial Networks with Binary Neurons for Polyphonic Music Generation. Online verfügbar unter http://arxiv.org/pdf/1804.09399v3.

Foster, David (2020): Generatives Deep Learning. Maschinen das Malen Schreiben und Komponieren beibringen. 1. Auflage. Heidelberg: O'Reilly (Animals).

Goodfellow, Ian (2016): NIPS 2016 Tutorial: Generative Adversarial Networks. Online verfügbar unter http://arxiv.org/pdf/1701.00160v4.

Goodfellow, Ian (2018): Deep learning. Das umfassende Handbuch : Grundlagen aktuelle Verfahren und Algorithmen neue Forschungsansätze. Unter Mitarbeit von Yoshua Bengio und Aaron Courville. 1. Auflage. Frechen: mitp Verlags GmbH & Co.

Goodfellow, Ian J.; Pouget-Abadie, Jean; Mirza, Mehdi; Xu, Bing; Warde-Farley, David; Ozair, Sherjil et al. (2014): Generative Adversarial Networks. Online verfügbar unter http://arxiv.org/pdf/1406.2661v1.

Google Developers (2019): Overview of GAN Structure. Online verfügbar unter https://developers.google.com/machine-learning/gan/gan_structure, zuletzt geprüft am 10.02.2021.

Gulrajani, Ishaan; Ahmed, Faruk; Arjovsky, Martin; Dumoulin, Vincent; Courville, Aaron (2017): Improved Training of Wasserstein GANs. Online verfügbar unter http://arxiv.org/pdf/1704.00028v3.

Gupta, Tushar (2017): Deep Learning: Feedforward Neural Network. towardsdatascience. Online verfügbar unter https://towardsdatascience.com/deep-learning-feedforward-neural-network-26a6705dbdc7, zuletzt geprüft am 17.02.2021.

HackerPoet (2018): Composer. Online verfügbar unter https://github.com/HackerPoet/Composer, zuletzt geprüft am 16.02.2021.

Harte, Christopher; Sandler, Mark; Gasser, Martin (2006): Detecting harmonic change in musical audio. In: Xavier Amatriain, Elaine Chew und Jonathan Foote (Hg.): Proceedings of the 1st ACM workshop on Audio and music computing multimedia. the 1st ACM workshop. Santa Barbara, California, USA, 10/27/2006 - 10/27/2006. The 14th ACM International Conference on Multimedia 2006; ACM Special Interest Group on Multimedia; ACM Special Interest Group on Computer Graphics and Interactive Techniques. New York, NY: ACM, S. 21.

Higgins, Irina; Matthey, Loic; Pal, Arka; Burgess, Christopher; Glorot, Xavier; Botvinick, Matthew et al. (2016): β-VAE: LEARNING BASIC VISUAL CONCEPTS WITH A CONSTRAINED VARIATIONAL FRAMEWORK. Online verfügbar unter https://openreview.net/pdf?id=Sy2fzU9gl, zuletzt geprüft am 15.02.2021.

Hochreiter, S.; Schmidhuber, J. (1997): Long short-term memory. In: *Neural computation* 9 (8), S. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

Ioffe, Sergey; Szegedy, Christian (2015): Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Online verfügbar unter http://arxiv.org/pdf/1502.03167v3.

Keras (2015): Keras. Online verfügbar unter https://keras.io/, zuletzt geprüft am 16.02.2021.

Kingma, Diederik P.; Welling, Max (2013): Auto-Encoding Variational Bayes. Online verfügbar unter http://arxiv.org/pdf/1312.6114v10.

Li, Fei-Fei; Krishna, Ranjay; Xu, Danfei (2020): Convolutional Neural Networks (CNNs / ConvNets). CS231n: Convolutional Neural Networks for Visual Recognition. Stanford University. Online verfügbar unter https://cs231n.github.io/convolutional-networks/, zuletzt geprüft am 18.02.2021.

Liang, Feynman; Gotham, Mark; Johnson, Matthew; Shotton, Jamie (2017): Automatic Stylistic Composition of Bach Chorales with Deep LSTM. In: 18th International Society for Music Information Retrieval Conference. 18th International Society for Music Information Retrieval Conference. Online verfügbar unter https://www.microsoft.com/en-us/research/publication/automatic-stylistic-composition-of-bach-chorales-with-deep-lstm/.

Mirza, Mehdi; Osindero, Simon (2014): Conditional Generative Adversarial Nets. Online verfügbar unter http://arxiv.org/pdf/1411.1784v1.

Nitish Srivastava; Geoffrey Hinton; Alex Krizhevsky; Ilya Sutskever; Ruslan Salakhutdinov (2014): Dropout: A Simple Way to Prevent Neural Networks from Overfitting. In: *Journal of Machine Learning Research* 15 (56), S. 1929–1958. Online verfügbar unter http://jmlr.org/papers/v15/srivastava14a.html.

Olah, Christopher (2015): Understanding LSTM Networks. Hg. v. colah's blog. Online verfügbar unter https://colah.github.io/posts/2015-08-Understanding-LSTMs/, zuletzt geprüft am 21.02.2021.

Patrice Ferlet (2019): How to work with Time Distributed data in a neural network. Hg. v. Medium. Online verfügbar unter https://medium.com/smileinnovation/how-to-work-with-time-distributed-data-in-a-neural-network-b8b39aa4ce00, zuletzt geprüft am 16.02.2021.

Phillips, Winifred (2014): A composer's guide to game music. Cambridge, Massachusetts: The MIT Press.

Raffel, Colin (2016): Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching.

Roberts, Adam; Engel, Jesse; Raffel, Colin; Hawthorne, Curtis; Eck, Douglas (2018): A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music. Online verfügbar unter http://arxiv.org/pdf/1803.05428v5.

Rocca, Joseph (2019): Understanding Variational Autoencoders (VAEs). Building, step by step, the reasoning that leads to VAEs. Towards Data Science. Online verfügbar unter https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73, zuletzt geprüft am 16.02.2021.

Rossetti, Gregory James (2020): Overworlds, towns, and battles. how music develops the worlds of role-playing video games. Dissertation. Rutgers, The State University of New Jersey.

Saha, Sumit (2018): A Comprehensive Guide to Convolutional Neural Networks. the ELI5 way. Hg. v. towardsdatascience. Online verfügbar unter https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, zuletzt geprüft am 18.02.2021.

Sanders, Timothy; Cairns, Paul (2010): Time Perception, Immersion and Music in Videogames. In: Proceedings of the 24th BCS Interaction Specialist Group Conference. Swindon, GBR: BCS Learning & Development Ltd (BCS '10), S. 160–167.

TensorFlow (2015): TensorFlow. Online verfügbar unter https://www.tensorflow.org/, zuletzt geprüft am 16.02.2021.

Venkatachalam, Mahendran (2019): Recurrent Neural Networks. Remembering what's important. Online verfügbar unter https://towardsdatascience.com/recurrent-neural-networks-d4642c9bc7ce, zuletzt geprüft am 20.02.2021.

Versloot, Christian (2019): How to create a variational autoencoder with Keras? Hg. v. machinecurve. Online verfügbar unter https://www.machinecurve.com/index.php/2019/12/30/how-to-create-a-variational-autoencoder-with-keras/, zuletzt geprüft am 17.02.2021.

VGMusic (1996): Video Game Music Archive. Online verfügbar unter https://www.vgmusic.com/, zuletzt aktualisiert am 16.02.2021.

Wiseodd (2016): Variational Autoencoder. Intuition and Implementation. Online verfügbar unter https://wiseodd.github.io/techblog/2016/12/10/variational-autoencoder/, zuletzt aktualisiert am 22.02.2021.

Eidesstattliche Erklärung

Ich versichere, dass ich die vorstehende Arbeit selbständig verfasst und hierzu keine anderen als die angegebenen Hilfsmittel verwendet habe. Alle Stellen der Arbeit. die wörtlich oder sinngemäß aus fremden Quellen entnommen wurden, sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt oder an anderer Stelle veröffentlicht.

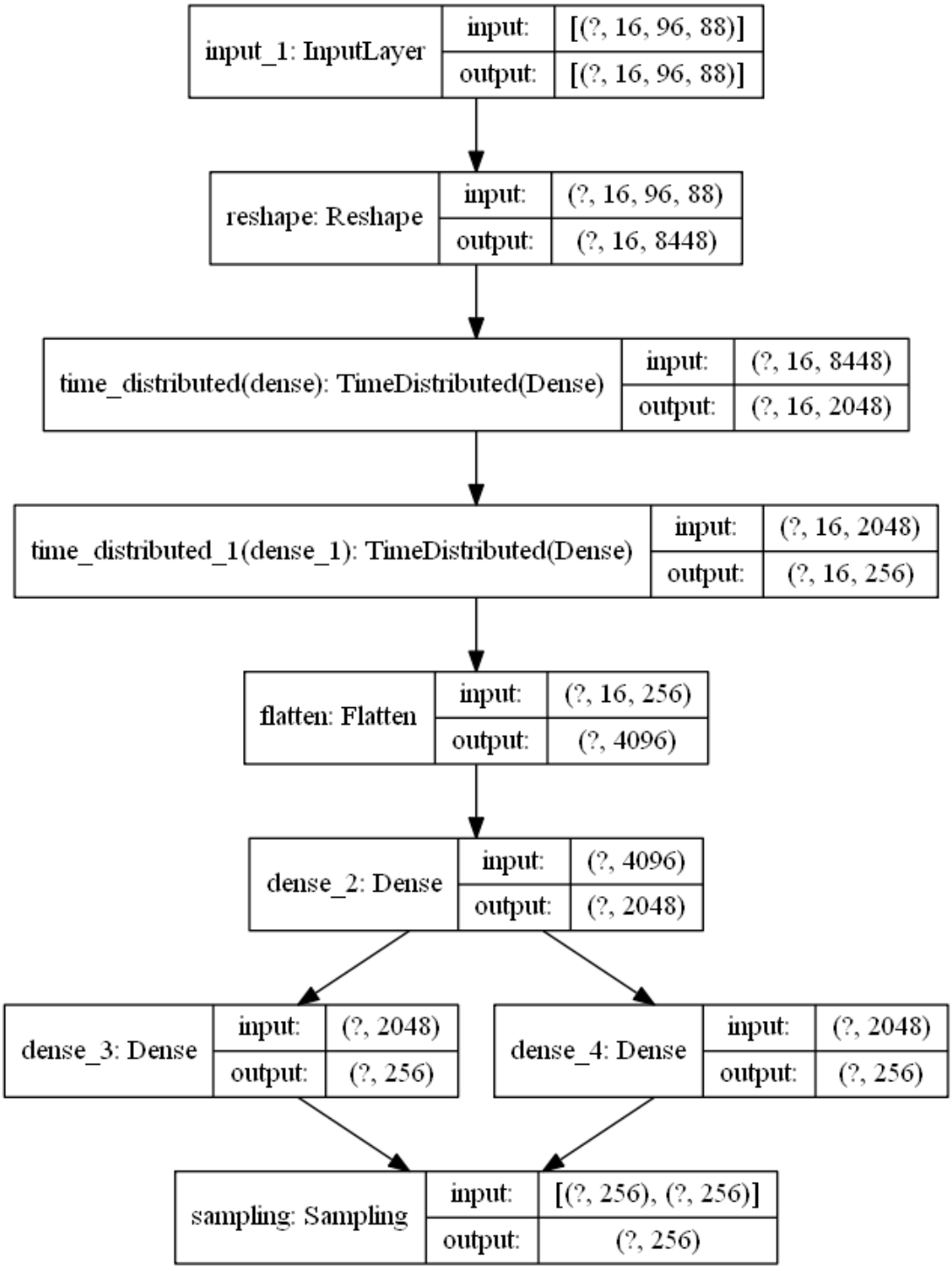
Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben kann.

_____

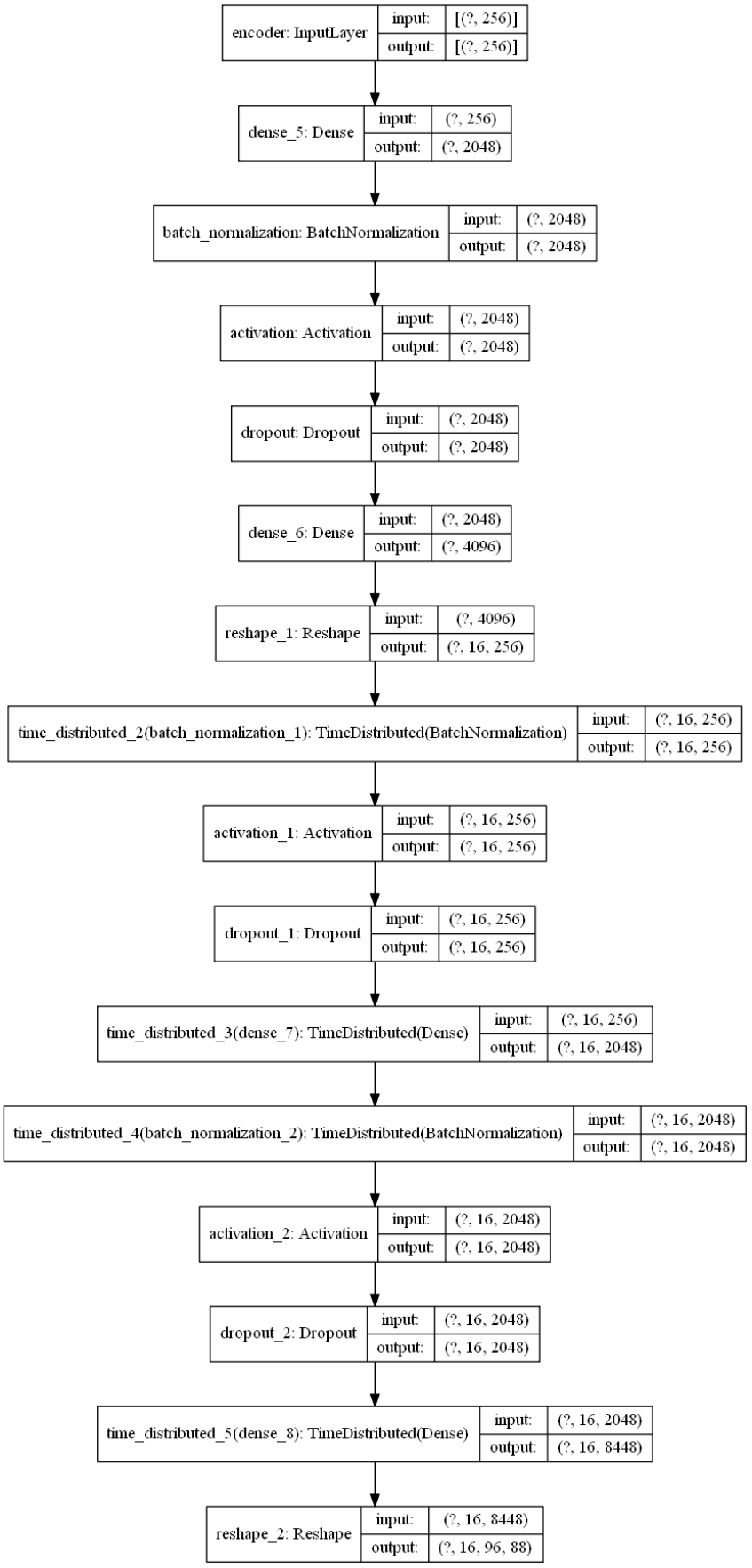
Furtwangen, 28.02.2021 Philipp Oeschger

Anhang

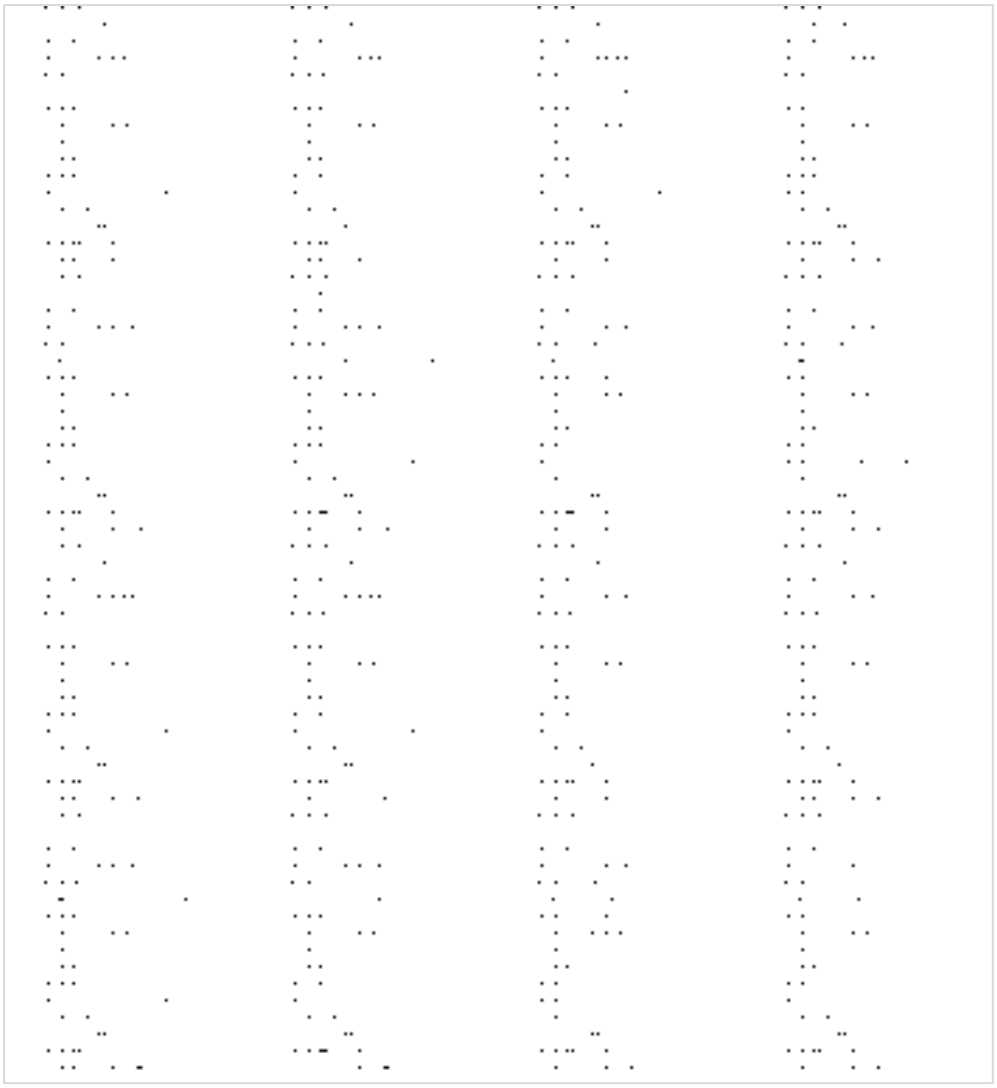
Anhang 1: Encoder Aufbau



Anhang 2: Decoder Aufbau

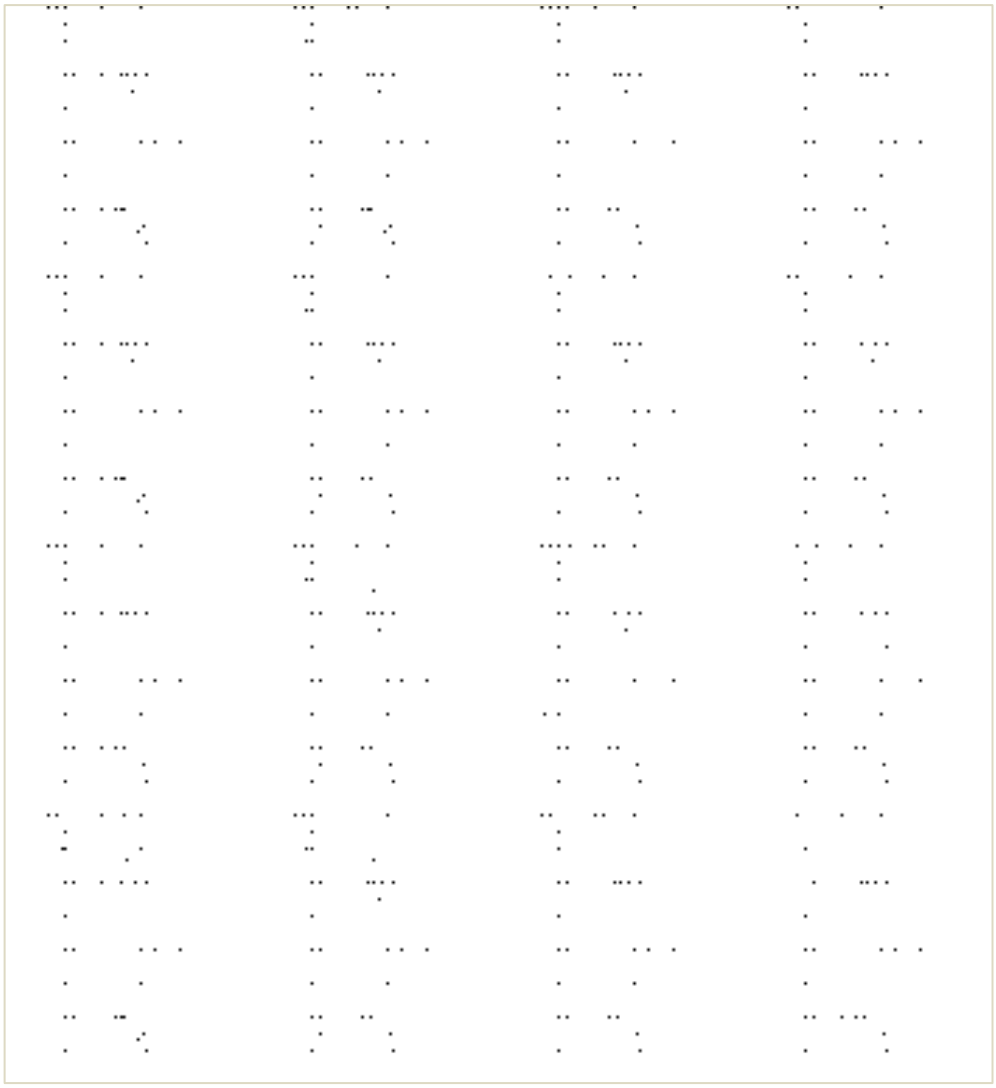


Anhang 3: Stichprobe aus "Kampf"-Modell in Piano-Roll-Format



Anmerkung: Die Leserichtung ist von oben nach unten, von links nach rechts.

Anhang 4: Stichprobe aus "Überwelt"-Modell in Piano-Roll-Format



Anmerkung: Die Leserichtung ist von oben nach unten, von links nach rechts.

Anhang 5: Ausschnitt der Songliste, Kategorie „Überweltmusik“

Nr.	*Song Name*	*Dateiname*	*Videospiel*	*Plattform*
1	Battle (on ground)	Battle-1.mid	Legend of Zelda, The The Wind Waker	GameCube
2	Ganon-dorf Batt-le	Z-WWGanondorfBattle.mid	Legend of Zelda, The The Wind Waker	GameCube
3	Helmaroc King Boss	Zelda_HelmarocBoss.mid	Legend of Zelda, The The Wind Waker	GameCube
4	Maritime Battle	Battle_At_Sea.mid	Legend of Zelda, The The Wind Waker	GameCube
5	Mini-boss	Mini-boss.mid	Legend of Zelda, The The Wind Waker	GameCube
6	Battle with Hooktail	HooktailBattle.mid	Paper Mario The Thousand-Year Door	GameCube
7	Battle with Magnus Von Grapple	PM_Ttyd_Lord_Crump.mid	Paper Mario The Thousand-Year Door	GameCube
8	Battle with Shadow Queen - First Form	PM_Ttyd_Final_Battle_Pre.mid	Paper Mario The Thousand-Year Door	GameCube
9	Battle with Shadow Queen - True Form	PM_Ttyd_Final_Battle.mid	Paper Mario The Thousand-Year Door	GameCube
10	Bowser Battle	bowserbattle2.mid	Paper Mario The Thousand-Year Door	GameCube
11	Chapter 3 Boss	PMTTYD-chap3boss.mid	Paper Mario The Thousand-Year Door	GameCube
12	Cortez Battle	pmttyd_cortezbattle.mid	Paper Mario The Thousand-Year Door	GameCube
13	Doopliss Battle	Doopliss_Battle.mid	Paper Mario The Thousand-Year Door	GameCube
14	Mini Boss Battle	PMTTYD-Mini_Boss.mid	Paper Mario The Thousand-Year Door	GameCube
15	Regular	Pa-	Paper Mario The	GameCube

	Enemy Battle	per_Mario_2_Battle_v2.mid	Thousand-Year Door	
16	Smorg Battle	PMTYD-Smorg_Battle.mid	Paper Mario The Thousand-Year Door	GameCube
17	Training Battle	Training_Battle-longer.mid	Paper Mario The Thousand-Year Door	GameCube
18	Battle with Gruntilda	Witch.mid	Banjo Kazooie	N64
19	Battle with Hag 1	Hag1.mid	Banjo-Tooie	N64
20	Battle for Lines	Bomber-man64BattleRemix.mid	Bomberman 64	N64
21	Mul-tiplayer Battle	Bombbatt.mid	Bomberman 64	N64
22	Sthertoth Battle	Sthertoth.mid	Bomberman 64 The Second Attack!	N64
23	Dance Of Illusions (Battle with Dracula's Servant)	CastleVania-_Illusions.mid	Castlevania 64	N64
24	Battle Arena	DK64_Battle_Arena-KM.mid	Donkey Kong 64	N64
25	K. Rool Battle	dk64_kingkrool.mid	Donkey Kong 64	N64
…	…	…	…	…

Anmerkung: Alle Musiktitel wurden der Quelle: „VGMusic“ (1996) entnommen.

Anhang 6: Ausschnitt der Songliste, Kategorie „Überweltmusik“

Nr.	*Song Name*	*Dateiname*	*Videospiel*	*Plattform*
1	Battle (on ground)	Battle-1.mid	Legend of Zelda, The The Wind Waker	GameCube
2	Ganon-dorf Batt-le	Z-WWGanondorfBattle.mid	Legend of Zelda, The The Wind Waker	GameCube
3	Helmaroc King Boss	Zelda_HelmarocBoss.mid	Legend of Zelda, The The Wind Waker	GameCube
4	Maritime Battle	Battle_At_Sea.mid	Legend of Zelda, The The Wind Waker	GameCube
5	Mini-boss	Mini-boss.mid	Legend of Zelda, The The Wind Waker	GameCube
6	Battle with Hooktail	HooktailBattle.mid	Paper Mario The Thousand-Year Door	GameCube
7	Battle with Magnus Von Grapple	PM_Ttyd_Lord_Crump.mid	Paper Mario The Thousand-Year Door	GameCube
8	Battle with Shadow Queen - First Form	PM_Ttyd_Final_Battle_Pre.mid	Paper Mario The Thousand-Year Door	GameCube
9	Battle with Shadow Queen - True Form	PM_Ttyd_Final_Battle.mid	Paper Mario The Thousand-Year Door	GameCube
10	Bowser Battle	bowserbattle2.mid	Paper Mario The Thousand-Year Door	GameCube
11	Chapter 3 Boss	PMTTYD-chap3boss.mid	Paper Mario The Thousand-Year Door	GameCube
12	Cortez Battle	pmttyd_cortezbattle.mid	Paper Mario The Thousand-Year Door	GameCube
13	Doopliss Battle	Doopliss_Battle.mid	Paper Mario The Thousand-Year Door	GameCube
14	Mini Boss Battle	PMTTYD-Mini_Boss.mid	Paper Mario The Thousand-Year Door	GameCube
15	Regular	Pa-	Paper Mario The	GameCube

	Enemy Battle	per_Mario_2_Battle_v2.mid	Thousand-Year Door	
16	Smorg Battle	PMTYD-Smorg_Battle.mid	Paper Mario The Thousand-Year Door	GameCube
17	Training Battle	Training_Battle-longer.mid	Paper Mario The Thousand-Year Door	GameCube
18	Battle with Gruntilda	Witch.mid	Banjo Kazooie	N64
19	Battle with Hag 1	Hag1.mid	Banjo-Tooie	N64
20	Battle for Lines	Bomber-man64BattleRemix.mid	Bomberman 64	N64
21	Mul-tiplayer Battle	Bombbatt.mid	Bomberman 64	N64
22	Sthertoth Battle	Sthertoth.mid	Bomberman 64 The Second Attack!	N64
23	Dance Of Illusions (Battle with Dracula's Servant)	CastleVania-_Illusions.mid	Castlevania 64	N64
24	Battle Arena	DK64_Battle_Arena-KM.mid	Donkey Kong 64	N64
25	K. Rool Battle	dk64_kingkrool.mid	Donkey Kong 64	N64
…	…	…	…	…

Anmerkung: Alle Musiktitel wurden der Quelle: **VGMusic** (1996) entnommen.