Bachelorarbeit

Schriftliche Prüfungsarbeit für die Bachelor-Prüfung des Studiengangs
Angewandte Informatik an der Ruhr-Universität Bochum

Long Short-Term Memory zur Generierung von Musiksequenzen

Autor:
Dada, Amin
108014234213

Abgabedatum:
15.02.2018

Betreuer:
PD Dr. Rolf P. Würtz
M.Sc. Andreas Nilkens

Überarbeitete Version, 16.03.2018

Inhaltsverzeichnis

1. Einführung

Rekurrente neuronale Netze (RNN) konnten erfolgreich auf eine Vielzahl von Problemen angewandt werden, wie z. B. Sprachübersetzung und Gesichts- und Handschrifterkennung. RNN sind künstliche neuronale Netze (KNN), die primär zur Verarbeitung sequenzieller Daten geeignet sind, da sie über ein Gedächtnis verfügen. RNN können in Vorhersagen Informationen aus vorherigen Schritten miteinbeziehen und somit kontextuelle Zusammenhänge erfassen. Eine spezielle RNN-Architektur stellt das Long Short-Term Memory Netzwerk (LSTM) dar. Im Gegensatz zu herkömmlichen RNN kann das LSTM auch Langzeitabhängigkeiten erkennen und ist somit ein häufig verwendetes RNN.

Ziel dieser Arbeit ist es, das Vermögen von LSTM zur Generierung von Musiksequenzen zu ergründen. Dabei soll Musik erzeugt werden, die bis zu einem bestimmten Grad, nicht von menschlich komponierter Musik zu unterscheiden ist.

Dazu wurde das Framework tf-seq2seq [1] eingesetzt, welches ein Encoder-Decoder Modell implementiert. Dieses Modell besteht aus zwei RNN: dem Encoder und dem Decoder. Der Encoder nimmt eine Eingabesequenz entgegen und bildet diese auf einen Vektor mit einer festen Größe ab. Der Decoder bildet anschließend den Vektor auf eine Ausgabesequenz ab. In dieser Arbeit wurden für den Encoder und den Decoder LSTM mit verschiedenen Konfigurationen verwendet.

Die verwendeten Musikstücke bestehen aus klassischer und traditioneller Klaviermusik in den Formaten MIDI, MusicXML und ABC. Die Daten wurden zunächst auf einen Notenwert quantisiert und dann in die Tonart C-Dur bzw. A-Moll transponiert. Anschließend wurden die vorbereiteten Daten in eine Textnotation überführt und in Sequenzen unterteilt. Die vorhergesagten Sequenzen des Netzwerks wurden abschließend wieder in das MIDI Format transformiert.

Ein weiterer wesentlicher Teil dieser Arbeit ist die Evaluation der generierten Musiksequenzen. Diese wurde in Form einer Umfrage durchgeführt, in der die Teilnehmer gefragt wurden, aus zwei Musiksequenzen die künstlich generierte zu bestimmen. Jeder Teilnehmer hat 5 zufällig gezogene Gegenüberstellungen von künstlich generierten und originalen Sequenzen, aus einer Sammlung von 50 Beispielen zugewiesen bekommen. Die generierten Sequenzen wurden zuvor händisch aus den Ergebnisses des Netzwerks ausgewählt.

2. Grundlagen

Das folgende Kapitel gibt eine kurze Einführung in die Entwicklung von RNN und geht anschließend auf die Architektur von LSTM ein. Zuletzt werden gängige Verfahren in der Anwendung von LSTM beschrieben. Es werden grundlegende Kenntnisse über feedforward-Netzwerke vorausgesetzt.

2.1 Rekurrente Neuronale Netze

Feedforward-Netzwerke sind in der Regel zur Verarbeitung von sequenziellen Daten ungeeignet, da diese jeden Berechnungsschritt unabhängig von dem vorherigen durchführen. Dadurch können keine Abhängigkeiten zwischen zeitlich versetzten Eingaben in Sequenzen erkannt werden. RNN wurden entwickelt, um dieses Hindernis zu überwinden.
Unter einer Eingabesequenz wird im Weiteren eine Folge $X = (x_{1,\dots,}x_n)$ und unter einer Ausgabesequenz eine Folge $Y = (y_{1,\dots,}y_n)$ verstanden. Der Index $t \in \{1, \dots, n\}$ gibt den Zeitpunkt innerhalb der Sequenz an. Des Weiteren werden Gewichtungsmatrizen, die die Aktivierung einer Zelle i in die Zelle j gewichten als W_{ij} notiert. Die folgende Formel gibt die Aktivierung einer Zelle eines simplen RNN zum Zeitpunkt t an:

$$h_t = \sigma_h(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \tag{1}$$

Der Aktivierungsfunktion σ_h wird die Eingabe x_t und die vergangene Aktivierung h_{t-1} übergeben. W_{xh} und W_{hh} sind die Gewichtungsmatrizen der Eingabe und der vergangenen Aktivierung. b_h ist das Bias, ein zusätzlicher Parameter zur Verschiebung der Aktivierungsfunktion.
Die Ausgabe des Netzwerks für den Zeitpunkt t berechnet sich durch:

$$y_t = \sigma_y(W_{hy}h_{t-1} + b_y) \tag{2}$$

Formel 2 zeigt, dass die Berechnung eines Zeitschritts der Ausgabesequenz rekursiv über alle vorherigen Eingaben und Aktivierungen erfolgt. Dadurch sollen kontextuell relevante Inhalte aus der Vergangenheit in Berechnungsschritten berücksichtig werden können.

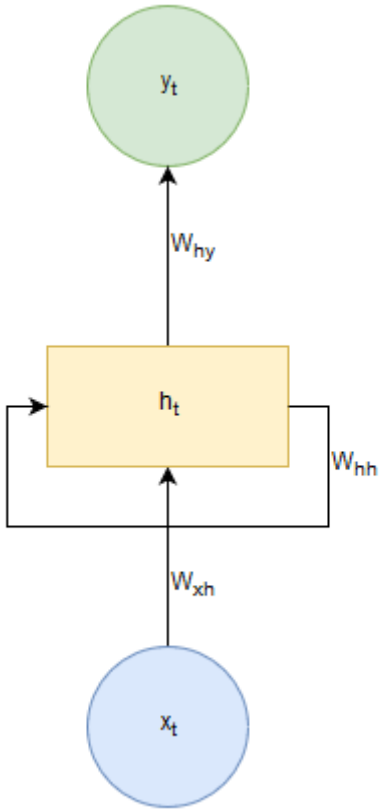

Abb. 1: RNN Zelle C mit Eingabe x und Ausgabe y zum Zeitpunkt t

Abbildung 1 visualisiert diese Berechnung. Neben den Verbindungen zwischen der Ein- und Ausgabe ist zusätzlich eine Schleife zu sehen, wodurch $\boldsymbol{y_t}$ in Abhängigkeit von vorherigen Schritten berechnet wird. Dies wird auch als Gedächtnis einer RNN-Zelle bezeichnet.
Analog zu feedforward-Netzen können RNN auch über mehrere Schichten zu tiefe RNN kombiniert werden. Als tiefe RNN werden im Folgenden RNN bezeichnet, die aus mehr als einer versteckten Schicht bestehen. Flache RNN verfügen somit über nur eine verstecken Schicht. Im Allgemeinen erzielen tiefe RNN bessere Ergebnisse als Flache [4].

2.2 Backpropagation Through Time

Alternativ zu der Darstellung in Abbildung 1, lässt sich dieselbe Zelle auch als feedforward-Netz mit einer versteckten Schicht für jede Eingabe darstellen. Diese Darstellung wird auch aufgefaltetes RNN genannt und zeigt, dass es sich bei RNN um tiefe feedforward-Netze handelt, die abhängig von der Länge der Eingabesequenz gebildet werden. Jeder versteckte Zustand des aufgefalteten RNN entspricht einem Zeitschritt innerhalb der Eingabe.

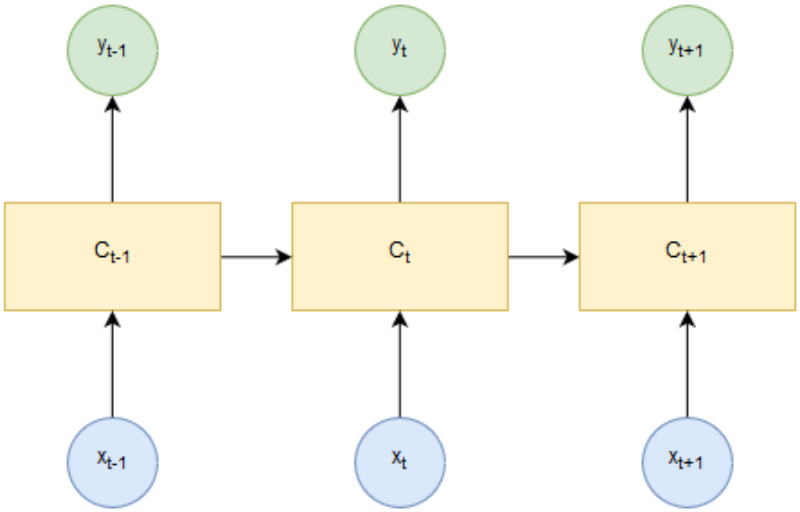

Abb. 2: Aufgefaltetes RNN

Auf ein aufgefaltetes RNN kann wie bei feedforward-Netzen der Backpropagation Algorithmus zur Minimierung der Fehlerfunktion angewendet werden. Dieser wird im Kontext von RNN Backpropagation Through Time (BPTT) genannt. Zunächst wird für jede Ausgabe des Netzwerks der Fehler über die Fehlerfunktion berechnet. Anschließend wird über die partiellen Ableitungen der Fehlerfunktion ihr Gradient berechnet. Mittels des Gradienten lässt sich dann eine Anpassung der Gewichtungsmatrizen W_{xh}, W_{hh} und W_{hy} ermitteln.

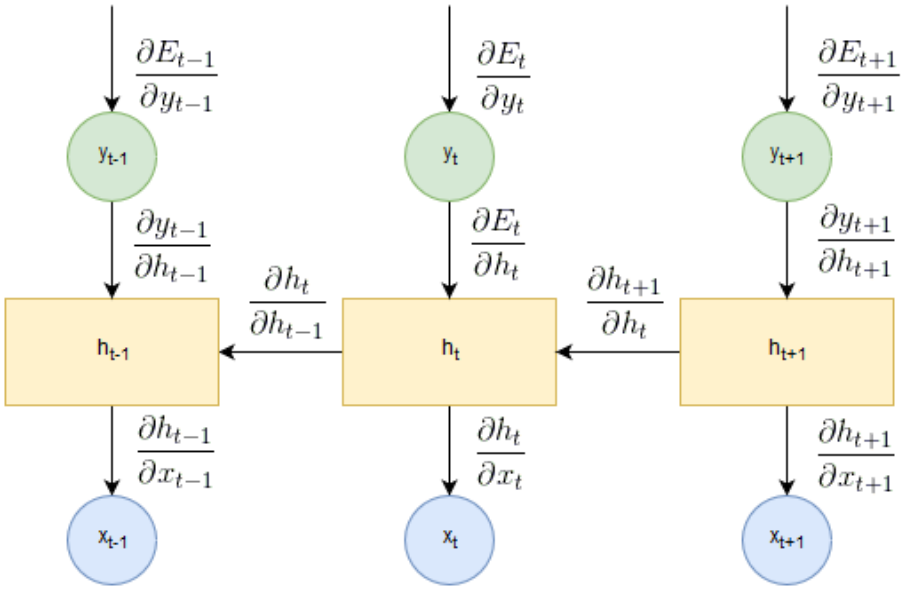

Abb. 3 Backpropagation Through Time

Im Gegensatz zu feedforward-Netzen teilen alle verstecken Schichten des aufgefalteten RNN dieselben Gewichtungsmatrizen, wodurch sich die Anzahl der zu berechnenden partiellen Ableitungen verringert. Allerdings ist die Größte des Netzwerks auch abhängig von der Länge der Eingabe. Dadurch kann es bei sehr langen Eingaben zu rechenintensiven Berechnungen des Gradienten kommen.

2.3 Vanishing/Exploding Gradient

Wie im vergangenen Abschnitt gezeigt, steigt die Tiefe eines aufgefalteten RNNs mit Länge der Eingabesequenzen. Zwei Probleme, die dadurch bei langen Eingaben auftreten, sind verschwindende und explodierende Gradienten.
Verschwindende Gradienten treten auf, wenn viele der im BPTT berechneten partiellen Ableitungen kleiner als 1 sind. Da der Gradient mittels der Kettenregel aus dem Produkt der partiellen Ableitungen berechnet wird, nimmt dieser im Verlauf der Propagation immer kleinere Werte an. Das Training von RNN wird dadurch ineffizient und ist stark von den initialen Gewichtungen abhängig. [6]
Explodierende Gradienten hingegen treten auf, wenn die partiellen Ableitungen verhältnismäßig hohe Werte annehmen. Infolgedessen wächst der Gradient sehr schnell und es werden die Gewichte innerhalb von einzelnen Schritten stark verändert. Dies führt zu instabilen Netzwerken, auf denen kein zielführendes Training mehr möglich ist. Zudem können Werte berechnet werden, die einen Überlauf verursachen, also die Kapazität des verwendeten Datentypens überschreiten. [7]
Eine Möglichkeit explodierenden Gradienten entgegenzuwirken wird im Abschnitt 2.5 vorgestellt. Verschwindende Gradienten sind in der Regel schwerer zu umgehen, da es oft schwierig ist zu bewerten, ob ein solcher Gradient entgegen dem optimalen Verhalten steht. So könnte es aufgrund der Beschaffenheit einer Fehlerfunktion von Vorteil sein, mit einem sehr kleinen Gradienten zu rechnen.
Beide Probleme nehmen dem Netzwerk die Fähigkeit, effizient gute Ergebnisse auf langen Sequenzen zu erzeugen. Für die Voraussage von Musiksequenzen sind diese Abhängigkeiten allerdings entscheidend. Zwei grundlegende musikalische Aspekte sind beispielsweise rhythmische Figuren und wiederkehrende Motive. Beide können als Langzeitabhängigkeiten betrachtet werden, da sie sich oft über mehrere Takte strecken. Ein Modell, welches diese Abhängigkeiten nicht lernen kann, wird vermutlich nicht in der Lage sein für Menschen überzeugende Musik

zu generieren. Vor diesem Hintergrund, wird im folgenden Abschnitt eine Netzwerkarchitektur vorstellt, die diesen Problemen entgegenwirkt.

2.4 Long Short-Term Memory

Long Short-Term Memory Netzwerke (LSTM) sind RNN, die durch ihre Architektur das Verschwinden des Gradienten verhindern und in der Lage sind Langzeitabhängigkeiten zu erfassen [7]. Im Gegensatz zu den bisher vorgestellten RNN-Zellen, verwenden LSTM Zustandszellen. Diese können mit ihrer wesentlich komplexeren Struktur den Informationsfluss regulieren. Die Zustandszellen von LSTM gewährleisten das Bestehen von Informationen über mehrere Zeitschritte hinweg. Der Zustand einer Zelle wird über zwei Gates beeinflusst: dem Forget Gate und dem Input Gate. Das Forget Gate ermöglicht es einerseits nicht mehr benötigte Informationen zu löschen und andererseits wichtige Informationen unverändert bestehen zu lassen. Das Input Gate ermöglicht es neue Informationen einzuspeichern. Ein drittes Gate ist das Output Gate, welches die Aktivierung der LSTM Zelle unter Berücksichtigung des Zellzustands ermittelt.

Abbildung 4 skizziert eine LSTM Zelle zum Zeitpunkt t. Im Gegensatz zu ursprünglichen RNN-Zellen erhalten die LSTM-Zellen, neben der Eingabe x_t und der Aktivierung aus dem vorherigen Zeitschritt h_{t-1}, auch den Zellzustand c_{t-1}, der als Gedächtnis der Einheit fungiert. Mittels der Gates und mehreren Aktivierungsfunktionen werden anschließend sowohl der neue Zellzustand c_t, als auch die aktuelle Aktivierung h_t berechnet. Im Folgenden werden zunächst die Operationen im separat beschrieben und anschließend in zwei Berechnungsschritte zusammengeführt.

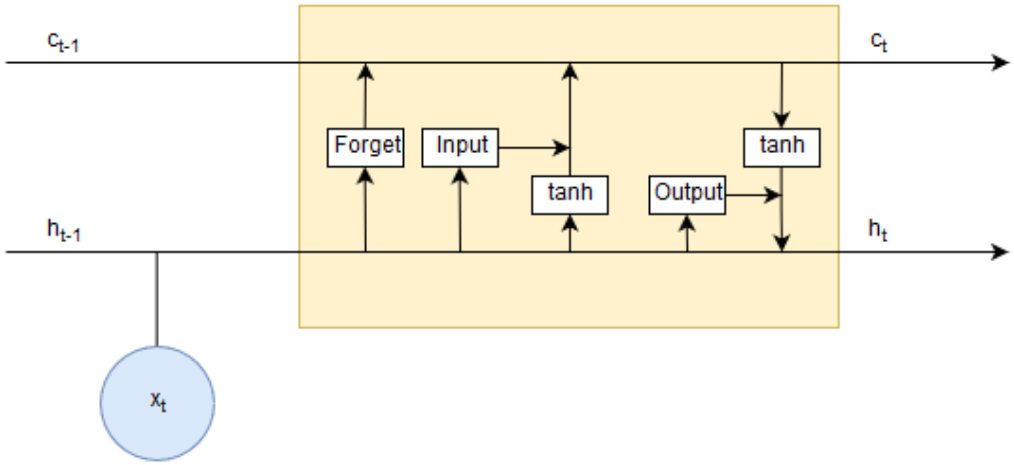

Abb. 4 LSTM Zelle

Zu Beginn einer Iteration wird dem Forget Gate die aktuelle Eingabe x_t und die Aktivierung h_{t-1} übergeben. Dieses bildet anschließend die Eingabe mittels der Sigmoid Funktion auf einen Vektor f_t mit Werten zwischen 0 und 1 ab. Der Vektor wird dann punktweise mit dem aktuellen Zellzustand multipliziert. Werte, die gegen 0 gehen bewirken, dass Informationen aus dem Zellzustand vergessen werden. Durch Werte, die gegen 1 gehen, bleiben Informationen erhalten. Formel 3 gibt die Berechnung von f_t an:

$$f_t = \sigma\left(W_{xf}\, x_t \; + W_{hf} h_{t-1}\right) \tag{3}$$

Anschließend wird der neue Zellzustand $\tilde{c}_t$ mittels der hyperbolischen Tangens Funktion (tanh) berechnet. $\tilde{c}_t$ kann als Vektor mit Kandidaten für den Zellzustand betrachtet werden. Das Input Gate bildet nun einen Vektor i_t aus Werten zwischen 0 und 1, der den Informationsfluss von Kandidaten in den Zellzustand beeinflusst. Werte, die gegen 0 gehen, verhindert den Zufluss von Informationen. Werte, die gegen 1 gehen, ermöglichen den Zufluss neuer Informationen.

$$i_t = \sigma(W_{xi}\, x_t \; + W_{hi} h_{t-1}) \tag{4}$$

$$\tilde{c}_t = tanh(W_{xc}\, x_t \; + W_{hc} h_{t-1}) \tag{5}$$

Die Formeln 4 und 5 beschreiben die Berechnung von den Vektoren i_t und $\tilde{c}_t$. Der Auswahlvektor i_t wird wie beim Forget Gate mit der Sigmoid Funktion σ aus der gewichteten Eingabe

x_t und der gewichteten vergangenen Aktivierung h_{t-1} berechnet. Der Eingangsvektor für den Zellzustand wird mit denselben Eingaben und seinen eigenen Gewichtungen berechnet.
Zuletzt wird der neue Zellzustand c_t und die neue Aktivierung h_t der Zelle berechnet. Dazu wird der neue Zellzustand in eine tanh Funktion überführt. Die damit berechnete Aktivierung wird durch das Output Gate geleitet. Dieses funktioniert analog zu den zuvor gezeigten Gates und wählt die Informationen aus, die in die Aktivierung einfließen sollen:

$$o_t = \sigma(W_{xo}\, x\ + W_{ho} h_{t-1}) \tag{6}$$

Die Formel 6 beschreibt die Berechnung des Output Gates. Mittels der zuvor aufgestellten Formeln lassen sich der neue Zellzustand und die Aktivierung ermitteln:

$$c_t = c_{t-1} \cdot f_t\ +\ \tilde{c}_t \cdot i_t \tag{7}$$

$$h_t = tanh(c_t) \cdot o_t \tag{8}$$

Der aktuelle Zellzustand wird aus dem vergangenen Zellzustand und der Aktivierung erstellt. Mittels dieser Operationen wird gewährleistet, dass nicht mehr benötigte Informationen vergessen werden und relevante, neue Informationen zum Gedächtnis hinzugefügt werden. Aus dem aktualisierten Zellzustand wird anschließend die Aktivierung der gesamten LSTM Zelle errechnet. Dabei wird der Informationsfluss durch das Output Gate reguliert. Dadurch wird entschieden, ob Informationen über die neue Aktivierung weitergegeben sollen.
Die Architektur eines LSTM verhindert das Auftreten von verschwindenden Gradienten aufgrund der Vorgehensweise beim Verändern des Zellzustands. Die Änderungen werden ausschließlich durch das Forget Gate und das Input Gate vorgenommen.
Das Forget Gate liefert den einzigen Vektor, mit dem der Zellzustand multipliziert wird. Der Vektor kann auch als Gewichtung des Zellzustands betrachtet werden, da die Informationen nicht durch eine nicht lineare Aktivierungsfunktion wie etwa der tanh Funktion manipuliert werden, sondern nur abhängig von ihrer Relevanz bewertet werden. Geht der Wert des Forget Gates gegen 1, wird die entsprechende Information gar nicht verändert, somit ist die in diesem Fall vorliegende Aktivierungsfunktion die Identitätsfunktion $f(x) = x$. Somit bewertete Informationen können also unverändert über mehrere Zeitschritte im Zellzustand bestehen bleiben. Zusätzlich ist zu beachten, dass $f'(x) = 1$ gilt und somit auch in diesem Kontext das zuvor

beschriebene Problem von Gradienten, die kleinere oder größere Werte als 1 annehmen und aufgrund dessen im Verlauf der BPTT verschwinden oder explodieren, direkt adressiert wird. Das Input Gate ermöglicht den Informationszufluss im Zellzustand. Aus der Formel 7 ist zu entnehmen, dass die zuvor vorberechnete Aktivierung $\tilde{c}_t$ auf den Zellzustand addiert wird. Somit fällt dieser Teil der Gleichung beim Ableiten weg.

2.5 Gradient Clipping

Im letzten Unterkapitel wurde die Funktionsweise von LSTM-Zellen und die Aussage getroffen, dass sie durch ihre Architektur nicht durch verschwindende Gradienten eingeschränkt werden. Explodierende Gradienten stellen allerdings auch bei LSTM ein Problem dar. Ein dafür verwendeter Lösungsansatz ist das Gradient Clipping [9]. Hierbei wird der Gradient, sobald er einen Schwellenwert überschreitet, auf einen kleineren Wert skaliert. Dies bewirkt eine Stabilisierung des Trainings, da große Schwankungen nach oben ausgeglichen werden.

2.6 Teacher Forcing

Eine weitere häufig verwendete Vorgehensweise beim Training von RNN ist Teacher Forcing [10]. Zur Unterscheidung zwischen der Vorhersage des Netzwerks und des Zielergebnisses zu einem Zeitpunkt t, wird im Folgenden die Notation $\tilde{y}_t$ für die Vorhersage und y für das Zielergebnis verwendet. Beim Teacher Forcing wird nach der Berechnung von $\tilde{y}_t$ im nächsten Zeitschritt als Eingabe für das Netz y verwendet, auch wenn die beiden Vektoren sich unterscheiden. Dadurch wird vor allem zu Beginn des Trainings der Fortschritt des RNN stabilisiert, da die Vorhersagen zu diesem Zeitpunkt nicht akkurat sind. Ein Problem, welches hierdurch auftreten kann, ist eine schlechte Performance des RNN bei der Validierung oder dem Testing, da ohne die Korrekturen keine zum Training vergleichbaren Ergebnisse erzielt werden können. Eine Erweiterung des Teacher Forcing ist das Professor Forcing [10], welches diesem Problem entgegenwirkt.

.

3. Daten

Im vergangen Kapitel wurde eine Einführung in die verwendeten Konzepte aus der Informatik und der Mathematik gegeben. Dieses Kapitel setzt einen Fokus auf die musikalischen Grundlagen. Zur Verarbeitung sämtlicher musikalischer Daten wurde das Framework music21 vom Massachusetts Institut of Technology verwendet [11].

3.1 Formate

Musical Instrument Digital Interface (MIDI) ist ein Protokoll zur Übermittlung von Musikdaten. Im Gegensatz zu Formaten wie beispielsweise MP3 werden in MIDI keine Audiodaten in Form von Frequenzen gespeichert, sondern Informationen über Noten und Instrumente. Diese Informationen werden als 3 Byte große Nachrichten, den MIDI Events, gespeichert. Für die Vorverarbeitung der Musikstücke sind Note on und Note off Events relevant, die den Start- und Endzeitpunkt von Noten angeben.

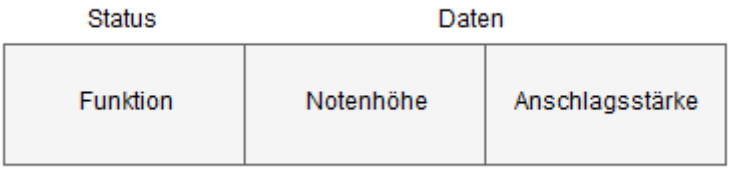

Abb. 5 MIDI Noten Event

Abbildung 5 zeigt ein solches Event. Das erste Byte gibt die Auskunft darüber, worauf sich das Event bezieht. In diesem Beispiel wird der Anfang oder das Ende einer Note dargestellt. Die letzten beiden Bytes geben die Tonhöhe und die Anschlagsstärke der Note an. Bei den Bytes wird zwischen Statusbytes und Datenbytes unterscheiden. Statusbytes geben Auskunft darüber, um was für einen Datensatz es sich handelt. Datenbytes enthalten die dazugehörigen Werte. Jedes Byte enthält ein Bit zur Unterscheidung zwischen Status- und Datenbytes. Daher verbleiben 7 Bits zur Darstellung der Notenhöhe und der Anschlagsstärke. Beide Angaben sind daher Werte zwischen 0 und 127. Die Notenhöhe 60 gibt ein C5 an und somit erstreckt sich das Intervall der möglichen Notenwerte von C0 bis G10. Eine Standardanschlagstärke hat einen Wert von 60 und wird stärker bei höheren und schwächer bei niedrigeren Werten.

Weitere Formate, in denen die verwendeten Daten vorliegen, sind ABC und MusicXML. Diese wurden zunächst mithilfe von music21 in MIDI konvertiert und werden daher im Folgenden nicht näher erläutert.

3.2 Datenquellen

Die verwendeten Musikstücke sind aus dem Bach-Werke-Verzeichnis (BWV) entnommen. Das Framework music21 bietet einen umfassenden Korpus[1] von Musikstücken verschiedener Komponisten an.

Die automatische Generierung von Musik im Stile von Johann Sebastian Bach stellt eine große Herausforderung der letzten Jahrzehnte dar [20]. Ein großer Vorteil der Stücke von J. S. Bach, ist ihre Homogenität [20] und die große Anzahl an Stücken, die durch das BWV verfügbar sind. Die verwendeten Stücke wurden auf die vierstimmigen Choräle Bachs eingeschränkt, um die Homogenität der Daten zu erhöhen. Die Choräle bestehen grundsätzlich aus vier Stimmen: Sopran, Alt, Tenor und Bass. Die Herausforderung bei der Generierung neuer Musik besteht darin, die Charakteristiken dieser Stimmen wiederzugeben. Dabei spielen einzelne Stimmen teils längere Phrasen, die im Gesamtbild mit den anderen Stimmen harmonieren müssen.

Das hier vorgestellte Modell ist jedoch nicht speziell auf die Choräle zugeschnitten und kann somit auch auf beliebige andere Musikstücke in den zuvor angegebenen Formaten angewendet werden. Aus zeitlichen Gründen wurden in dieser Arbeit allerdings ausschließlich die Choräle mittels des Modells untersucht.

3.3 Vorverarbeitung der Daten

Zur Vorbereitung der Daten wurden diese quantisiert und in eine einheitliche Tonart transponiert. Beide Verfahren haben bereits die Ergebnisse vorangegangener Arbeiten positiv beeinflusst [12,13].

Unter Quantisierung wird im musikalischen Kontext das Anordnen von Noten verstanden, sodass jede Note durch einen festgelegten Notenwert ausgedrückt werden kann. Wird ein Stück beispielweise auf Achtelnoten quantisiert, bleiben die Achtelnoten und alle größeren Notenwerte des Stücks unverändert. Alle kleineren Notenwerte hingegen werden jeweils auf die nächst näheren Achtelnoten abgebildet.

[1] http://web.mit.edu/music21/doc/about/referenceCorpus.html

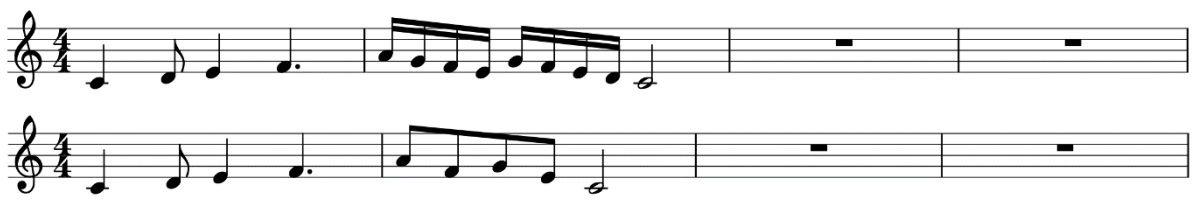

Abb. 6 Quantisierung auf Achtel

Die Abbildung 6 zeigt eine Quantisierung auf Achtelnoten. Die Noten im ersten Takt bleiben unverändert, da eine Viertelnote zwei Achtelnoten und eine punktierte Viertelnote drei Achtelnoten entsprechen. Der zweite Takt hingegen enthält Sechzehntelnoten, die durch Achtelnoten, nicht darstellt werden können. Als Folge dessen wird nur jede zweite Sechzehntelnote als Achtelnote in die quantisierte Version übernommen. In dieser Arbeit wurden die untersuchten Choräle auf Sechzehntelnoten transponiert.

Transposition ist das Erhöhen oder Absenken von allen Tonhöhen eines Stückes um ein definiertes Intervall. Dadurch kann die Tonart des Stückes verändert werden. Durch die heutige Verwendung der gleichstufigen Stimmung klingen die Intervalle in allen Tonarten gleich und somit ist eine Transposition, ohne eine charakterliche Veränderung eines Musikstücks möglich [14]. Damit bestimmt werden kann, um welches Intervall ein Stück transponiert muss, damit es in C-Dur vorliegt, muss zunächst die Tonart ermittelt werden. music21 verwendet hierzu den Krumhansl-Schmuckler Algorithmus [15], der im folgenden Kapitel kurz erläutert wird.

3.4 Krumhansl-Schmuckler Algorithmus

Mit dem Krumhansl-Schmuckler Algorithmus kann die Tonart von Musikstücken geschätzt werden. Dazu benötigt der Algorithmus die Dauer aller 12 Tonhöhen innerhalb des Musikstücks. Zudem werden zwei Profile verwendet, die Dur und Moll repräsentieren. Jedes Profil enthält 12 Einträge, die das Verhältnis der Anzahl der auftretenden Töne in einer Tonart angeben. Die folgenden Tabellen geben die Profile an:

Tabelle 1: Dur-Profil

0	1	2	3	4	5	6	7	8	9	10	11
6.35	2.23	3.48	2.33	4.38	4.09	2.52	5.19	2.39	3.66	2.29	2.88

Tabelle 2: Moll-Profil

0	1	2	3	4	5	6	7	8	9	10	11

6.33	2.68	3.52	5.38	2.6	3.53	2.54	4.75	3.98	2.69	3.34	3.17

Die erste Zeile gibt das jeweilige Intervall an, wobei sich 0 auf den Grundton bezieht, 1 auf die kleine Sekunde usw. Es lässt sich erkennen, dass in beiden Profilen der Grundton den höchsten und die Quinte (7) den zweithöchsten Wert tragen. Im Dur-Profil hat die große Terz (4) den dritthöchsten Wert und im Moll-Profil die kleine Terz (3). Das lässt sich damit erklären, dass diese Intervalle in Dur und Moll häufig verwendet werden. Zudem werden Dreiklänge aus diesen Intervallen gebildet.

Der Algorithmus vergleicht die lineare Abhängigkeit zwischen den Profilen und der Anzahl der im Musikstück vorkommenden Töne. Zur Ermittlung der linearen Abhängigkeit wird der Korrelationskoeffizient gebildet.

$$\mathrm{Kor}(\mathrm{X},\mathrm{Y}) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{9}$$

Formel 9 gibt den Korrelationskoeffizienten für zwei Mengen $\mathrm{X} = \{x_1, \dots, x_n\}$ und $\mathrm{Y} = \{y_1, \dots, y_n\}$ mit $x_i, y_i \in \mathbb{R}$ an. Der Korrelationskoeffizienten nimmt Werte zwischen -1 und 1 an, wobei -1 und 1 vollständige lineare Abhängigkeiten zwischen den betrachteten Elementen ausdrücken und 0 keine lineare Abhängigkeit.

Im Laufe des Algorithmus werden für jeden möglichen Grundton und für beide Profile der Korrelationskoeffizient aus den Werten der Profile und den im Stück gezählten Tönen gebildet. Aus den 24 Korrelationskoeffizienten wird dann der im Betrag maximale ausgewählt. Die so bestimmte Tonart wird anschließend von dem Algorithmus zurückgegeben.

Tabelle 3: Abgleich auf C-Dur

Dur-Profil	Anzahl Noten
6.35	C: 24
2.23	C#: 0
3.48	D: 111
2.33	D#: 0
4.38	E: 12
4.09	F: 0
2.52	F#: 18

Tabelle 4: Abgleich auf G-Dur

Dur-Profil	Anzahl Noten
6.35	G: 193
2.23	G#: 0
3.48	A: 22
2.33	B: 0
4.38	H: 66
4.09	C: 24
2.52	C#: 0

5.19	G: 193
2.39	G#: 0
3.66	A: 22
2.29	B: 0
2.88	H: 66
Kor(X, Y) = 0.4123	

5.19	D: 111
2.39	D#: 0
3.66	E: 12
2.29	F: 0
2.88	F#: 18
Kor(X, Y) = 0.9365	

Die Tabellen 3 und 4 zeigen beispielhaft die Berechnung des Korrelationskoeffizienten für die Tonarten C-Dur und G-Dur für das Stück Amazing Grace.

3.5 Repräsentation als Text

Um die Daten mittels des seq2seq Framework zu verarbeiten, müssen diese als Textsequenzen dargestellt werden. Das Netzwerk kann dann lernen, die einzelnen Sequenzen aufeinander abzubilden. Hierzu wurden die verwendeten Musikstücke in Sequenzen einer festgelegten Länge aufgeteilt. Anschließend wurden die so gebildeten Sequenzen abwechselnd als Eingabe- und Ausgabesequenzen verwendet. Genauer bedeutet dies, dass jede Ausgabesequenz eines Stücks, mit Ausnahme der Letzten, auch eine Eingabesequenz ist und jede Eingabesequenz, mit Ausnahme der Ersten auch eine Ausgabesequenz ist. Abbildung 7 bildet die Unterteilung eines Musikstücks ab.

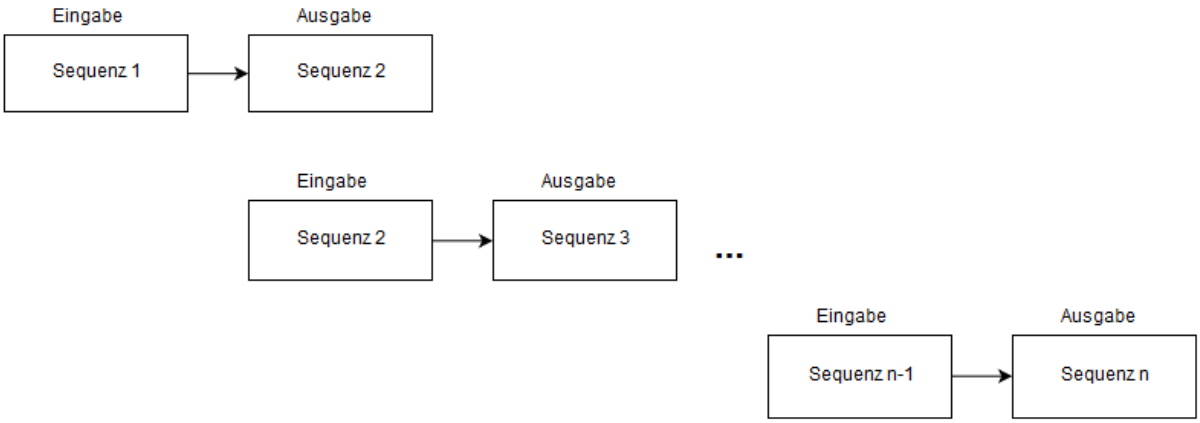

Abb. 7 Aufteilung eines Musikstücks in Sequenzen.

Das Codierungsschema zur textuellen Darstellung der Musik wurde weitestgehend aus [16] übernommen.

Eine kodierte Note besteht aus zwei Teilen. Der erste Teil gibt die Tonhöhe im MIDI Format an. Der zweite Teil gibt an, ob die Note aktuell anschlagen wird oder, ob sie bereits angeschlagen wurde und aktuell gehalten wird. Durch die vorher durchgeführte Quantisierung der Musikstücke, können die Noten in Frames unterteilt werden, die jeweils der Dauer des kleinsten Zeitschritts entsprechen. Zur Abgrenzung der Frames wird ein Senkrechter Stricht (auch Pipe) verwendet. Noten, die in einem Frame notiert sind, erklingen simultan. Vor dem ersten Frame eines Stücks wird das Wort START vorangesetzt und nach dem letzten Frame das END. Dadurch kann das Netzwerk Anfänge von Stücken in den Lernprozess miteinbeziehen und vorhergesagte Musiksequenzen beenden. Die Tabelle 5 gibt eine Übersicht über die verwendeten Wörter und Symbole.

Tabelle 5: Verwendete Codierung

START	Wird vor den Anfang eines Stücks gesetzt.
END	Wird an das Ende eines Stücks gesetzt.
60,0	Die Note mit einer Notenhöhe von 60 wird in dem aktuellen Frame angeschlagen.
65,1	Die Note mit der Notenhöhe 65 wurde in einem vergangenen Frame angeschlagen und wird im aktuellen Takt gehalten.
|	Markiert den Übergang zum nächsten Frame.

Bei der Implementierung des Schemas ist zu beachten, dass die Noten innerhalb von Frames sortiert werden sollten. Dies ist notwendig, da es sonst verschiedene Darstellungsmöglichkeiten für gleiche Akkorde gibt. Beispielsweise würde das Netzwerk zwischen den Eingaben „60,0 64,0 67,0“ und „67,0 64,0 60,0“ unterscheiden, obwohl beides einem C-Dur Akkord entspricht.

4. Modell

Das Sequence-to-sequence (seq2seq) Modell wurde 2014 in [17] und [18] vorgestellt und hat seitdem vor allem Erfolge im Bereich der maschinellen Übersetzung erzielt. Ein entscheidender Vorteil des Modells, ist die Möglichkeit zur Verarbeitung von Sequenzen verschiedener Längen. Dadurch können Sätze mit variablen Längen aufeinander abgebildet werden. Neben den Möglichkeiten, die sich dadurch bei der maschinellen Übersetzung bieten, ist dieses Modell auch bei der Anwendung auf Musiksequenzen von Vorteil, da die Längen der Sequenzen abhängig von der Anzahl der Noten in einzelnen Frames ist. Zudem kann die letzte Sequenz eines Stücks kürzer ausfallen.

4.1 Vektoreinbettung

Damit die Sequenzen vom Modell verarbeitet werden können, müssen diese zunächst in numerische Repräsentationen überführt werden. Bei Sprachmodellen werden dazu einzelne Worte und Satzzeichen in sogenannte Tokens eingeteilt. Die Tokens bilden die kleinste Einheit einer Sequenz. In dieser Arbeit wurde jedes Element der in Kapitel 3 beschriebenen Codierung (siehe Tabelle 5) als ein Token aufgefasst. Die Menge aller Tokens ist das Vokabular V. Die obere Schranke der Größe des Vokabulars ist durch $|V| \leq 128 \cdot 2 + 3 = 259$ gegeben, da das MIDI-Format 128 unterschiedliche Tonhöhen unterstützt, die in der Codierung entweder als angeschlagene oder gehaltene Note auftreten können. Zudem gibt es die Tokens ‘START‘, ‘END‘ und ‘|‘.

Aus den Tokens im Vokabular werden Vektoren einer festen Größe gebildet. Dies wird durch die erste Schicht des Netzwerks, der Einbettungsschicht, durchgeführt. Die Vektorrepräsentationen enthalten Eigenschaften von Tokens. In Sprachmodellen sind dies beispielsweise die Wortart oder eine Information darüber, dass es sich bei dem eingebetteten Wort um ein Tier handelt.

Da die Vektoreinbettung viele Vorzüge bietet, sind bereits Modelle veröffentlicht worden, die sich ausschließlich mit der Einbettung von Tokens beschäftigen [19]. Ein verbreitetes Tool ist beispielweise word2vec[2]. Mittels solcher Tools kann die Vektoreinbettung vor dem Training des Netzwerks vorgenommen werden. In dieser Arbeit wird die Einbettung durch die erste Schicht des Encoders erzeugt und somit innerhalb des Trainings gelernt.

[2] https://code.google.com/archive/p/word2vec/

4.2 Encoder-Decoder

Das verwendete Modell besteht aus zwei LSTM: dem Encoder und dem Decoder. Im Folgenden wird die Aktivierung des Encoders mit $h^{(e)}{}_t$ und die Aktivierung des Decoders mit $h^{(d)}{}_t$ bezeichnet. Zudem wird zwischen der Länge der Eingabesequenz und der Länge der Ausgabesequenz unterschieden. Die Eingabesequenz hat weiterhin eine Länge von n (siehe 2.1), wohingegen die Ausgabesequenz eine Länge von n' hat. Diese Unterscheidung ist notwendig, da sich die Längen unterscheiden können. Außerdem wird die Notation y_t für die Zielausgabe und $\tilde{y}_t$ für die vorausgesagte Ausgabe verwendet.
Die Gewichte des Encoders werden zunächst mit 0 initialisiert. Der Encoder erzeugt in der ersten versteckten Schicht, der Einbettungsschicht, die Vektorrepräsentation eines Tokens zum Zeitpunkt t und berechnet daraus seine Aktivierung $h^{(e)}{}_t$. Der Encoder liest so jedes Token der Sequenz ein. Die Ausgabesequenz des Encoders wird nicht berücksichtigt, da die gesuchte Ausgabe durch $h^{(e)}{}_n$ gegeben ist. Dies ist die Aktivierung im letzten Zeitschritt der Eingabesequenz. Das letzte Token der Eingabesequenz ist ein spezielles Token, welches das Ende der Sequenz markiert. Ist dieses Token erreicht, wird der Decoder mit der letzten Aktivierung des Encoders initialisiert.
Der Decoder erhält in seinem ersten Berechnungsschritt ein spezielles Token als Eingabe, welches den Start der Sequenz kennzeichnet. Mittels der zuvor erhaltenen Aktivierung des Encoders, gibt er das erste Ausgabetoken aus. Das Ausgabetoken wird ermittelt, indem die Aktivierung der Softmax-Funktion übergeben wird. Diese ist durch die Folgende Formel definiert:

$$\sigma(z_t)_j = \frac{e^{(z_t)_j}}{\sum_{k=1}^{K}(z_t)_k} \tag{10}$$

Die Softmax-Funktion wird vor allem als Aktivierungsfunktion von Klassifizierungsproblemen eingesetzt, bei denen es mehr als zwei Klassen gibt. Die Auswahl der Tokens ist ein solches Klassifizierungsproblem. Die Funktion bildet die Eingabe auf Werte zwischen 0 und 1 ab. Zudem dividiert sie alle Werte durch die Summe aller Eingaben, damit die Ausgabewerte in Summe 1 ergeben. Die Werte der Ausgabe geben an, mit welcher Wahrscheinlichkeit die Eingabe einer bestimmten Klasse zugewiesen wird. Die Auswahl des Tokens erfolgt demnach mittels des höchsten Werts der Softmax-Funktion. In Formel 10 wird aus Gründen der Lesbarkeit

die Aktivierung des Encoders als z_t bezeichnet. Einzelne Elemente des Vektors werden mit dem Index j adressiert. Zudem sei K als die Dimension des Vektors gegeben.
Da das in Abschnitt 2.6 beschriebene Teacher Forcing Vorgehen angewendet wird, erhält der Decoder im zweiten Schritt nicht die Ausgabe des ersten Schritts, sondern das erste Token der korrekten Zielsequenz. In den darauffolgenden Schritten sagt der Decoder so lange Tokens voraus, bis das reservierte End-Token ausgegeben wird, oder eine maximale Ausgabelänge erreicht wird. Da die Länge der Ausgabesequenz sich von der Länge der Eingabesequenz unterscheiden kann, wird im Folgenden n für die Länge der Eingabesequenz und n' für die Länge der Ausgabesequenz verwendet.

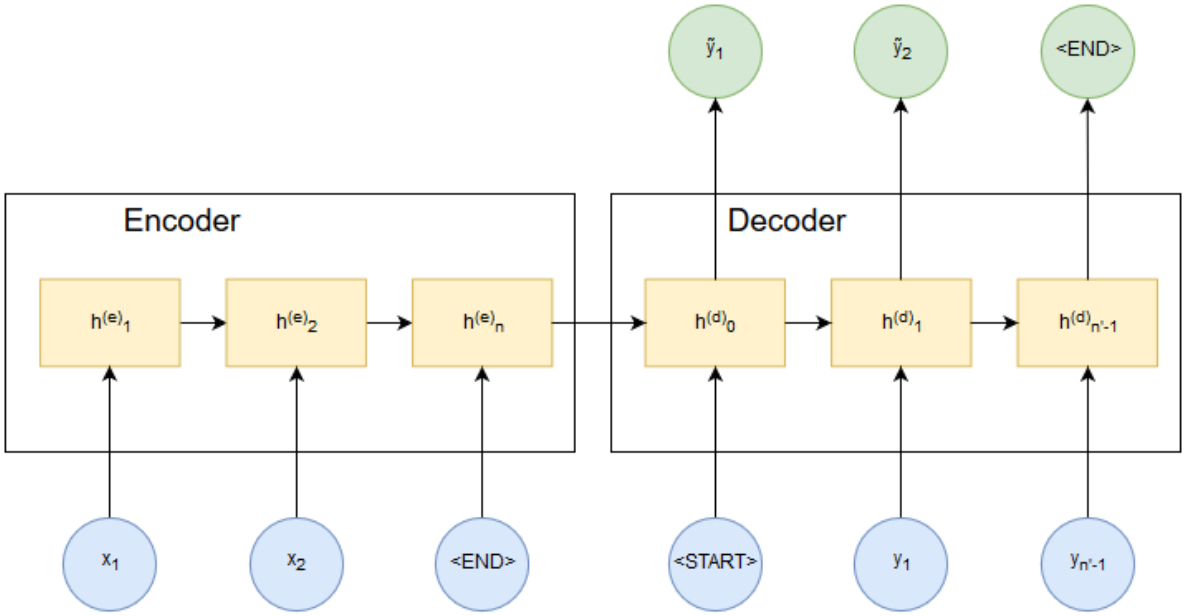

Abb. 8 Encoder-Decoder Modell

Abbildung 8 zeigt den oben beschriebenen Vorgang. Um eine bessere Übersicht zu bieten, wurde darauf verzichtet die Einbettungsschicht des Encoders und die Softmax-Schicht des Decoders darzustellen. Die übergebene Eingabesequenz ist $x_1, x_2, \ldots, < END >$. <END> markiert das Ende der Sequenz und somit den Übergang zum Decoder. Die Vorhersage der Ausgabesequenz wird mit dem Token <START> initialisiert. Anschließend wird jeweils das nächste Token der Zielsequenz dem Decoder als Eingabe übergeben. Da der Decoder mit der letzten Aktivierung des Encoders initialisiert wurde, hat er Zugriff auf alle Informationen der Eingabesequenz.

4.3 Bidirektionale RNN

In einem bidirektionalen RNN wird die Anzahl der versteckten Schichten verdoppelt. Jede Schicht des RNN wird um eine weitere Schicht ergänzt, die die Eingabe in umgekehrter Reihenfolge liest. Dieser Ansatz wurde erstmals in [21] vorgestellt. Die Intention dieser Architektur ist, dass herkömmliche RNN nicht in der Lage sind in der Zukunft liegende Informationen zu verarbeiten. Diese Einschränkung mag zunächst logisch klingen. Beobachtungen im Gebiet der Sprachverarbeitung deuten allerdings darauf hin, dass Menschen gehörte Äußerungen nicht zwingend linear interpretieren [22]. Einige Äußerungen werden vielmehr erst durch zukünftige Informationen verstanden [22]. Es liegt daher die Vermutung nahe, dass Menschen gehörte Musik ähnlich aufnehmen. Es wurden daher in dieser Arbeit auch Experimente mit einem bidirektionalen Encoder durchgeführt.

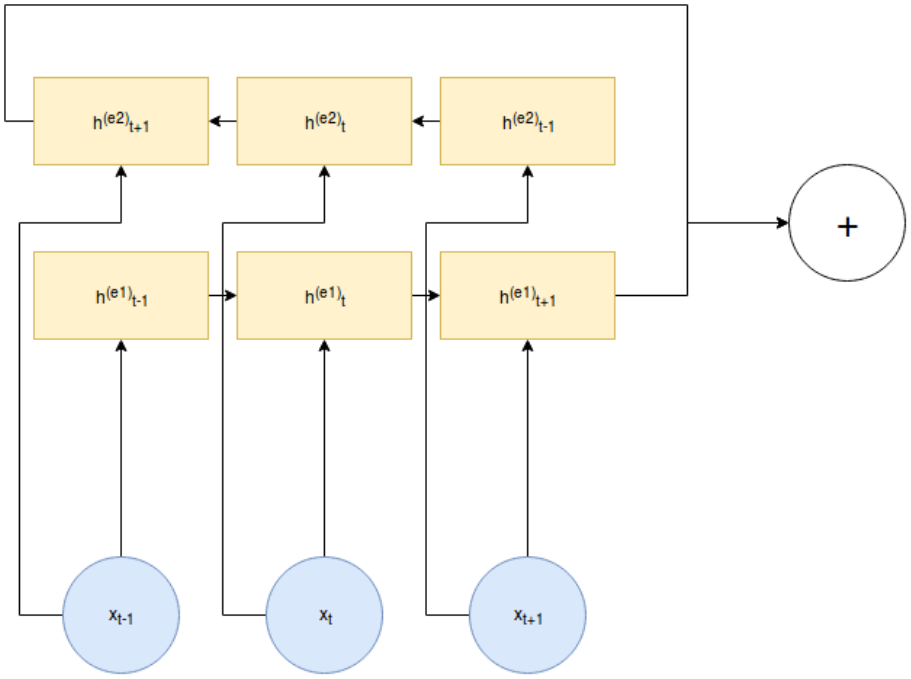

Abb. 8 Bidirektionales RNN

Abbildung 8 skizziert ein zweischichtiges bidirektionales RNN. Im Gegensatz zu RNN mit mehreren tiefen Schichten, tauschen die hier hinzugefügten Schichten keine Informationen untereinander aus. Die Schichten können vielmehr als separate Netzwerke gesehen werden, die individuell auf derselben Eingabe arbeiten. In dem hier verwendeten Encoder-Decoder Modell gibt es daher einen Encoder, der die Sequenz vorwärts einliest und einen weiteren, der die Sequenz rückwärts einliest.

Die Ausgaben der beiden Encoder werden in der Regel zu einem Vektor konkateniert oder punktweise addiert. In dieser Arbeit wurde letztere Option gewählt. Im Falle einer Konkatenation verdoppelt sich die Dimension der Ausgabe und der Decoder müsste entsprechend angepasst werden.

4.4 Dropout

Dropout ist eine Methode zur Prävention, bzw. zur Verringerung, von Overfitting. Diese Methode wird häufig zur Regulierung von RNN verwendet. Während des Trainings werden in jedem Berechnungsschritt zufällig ausgewählte Zellen deaktiviert. Dadurch wird in jedem Schritt, abhängig von der Rate nur ein Teil der Zellen des RNN zur Berechnung der Ausgabe und zur Backpropagation des Fehlers verwendet. Dropout wirkt Overfitting vor allem durch die Verhinderung von Abhängigkeiten zwischen einzelnen Zellen entgegen [23].

5. Versuche

In diesem Kapitel werden die Experimente beschrieben, die mit dem zuvor beschriebenen Modell durchgeführt wurden. Zunächst werden Parameter beschrieben, die in allen Experimenten gewählt wurden. Anschließend wird eine Ausgangskonfiguration vorgestellt. Abhängig von den Beobachtungen in der Ausgangskonfiguration werden schrittweise einzelne Parameter des Modells verändert und die Ergebnisse erneut bewertet.

5.1 Fest gewählte Parameter

In allen Versuchen wurde die Adam Optimierungsmethode verwendet [2], die eine Weiterentwicklung des stochastischen Gradientenabstiegs (SGD) ist. Es wurde eine Minibatch Größe von 32 gewählt. Die initiale Lernrate wurde auf einen Wert von 0.002 gesetzt und im Durchlauf nach jeder achten Epoche mit einem Wert von 0.5 multipliziert.
Eine weitere Entscheidung, die getroffen werden musste, ist die Wahl der Länge einer Sequenz. Diese wird durch die Anzahl der Frames einer Sequenz festgelegt. Grundsätzlich sind längere Sequenzen vorteilhaft, da dadurch auch die vorhergesagte Sequenz länger ist. Allerdings steigt der Rechenaufwand auch mit der Länge der Sequenz schnell an. Zudem sollte es sich bei der Frameanzahl um Zweierpotenz handeln, da Sequenzen sonst an unvorteilhaften Stellen abgeschnitten werden (z.B. an der fünfzehnten Sechzehntelnote eines Taktes). Eine Frameanzahl, die kleiner als 64 ist, erscheint bei einer Quantisierung auf Sechzehntelnoten wenig sinnvoll, da die Ausgabesequenzen in diesem Fall eine maximale Länge von ungefähr 4 Sekunden hätten und somit eine Bewertung kaum möglich wäre. In den Versuchen ist allerdings auch schnell aufgefallen, dass eine Frameanzahl von 128 die Kapazitäten der verwendeten Hardware überschreitet. Daher wurden alle Versuche mit einer Framezahl von 64 durchgeführt.
Der Betrag des Gradienten wurde bei gleichbleibender Richtung mittels Gradient Clipping ab einem Wert von 5 heruntergesetzt.
Als Fehlerfunktion wurde die Kreuzentropie verwendet.
Somit bleiben als festzulegende Parameter die Folgenden übrig:

- Dropout-Rate
- Anzahl der LSTM-Zellen
- Anzahl der Schichten

- Wahl zwischen unidirektionalem und bidirektionalem Encoder

5.2 Ausgangskonfiguration

Die Ausgangskonfiguration ist in der folgenden Tabelle dargestellt:

Tabelle 6: Ausgangskonfiguration

Dropout-Rate	0
Anzahl der LSTM-Zellen	128
Anzahl der Schichten	2
Unidirektional oder bidirektional	Unidirektional

Es ist zu erwarten, dass diese Konfiguration an Overfitting leiden wird, da bereits viele Arbeiten in der Vergangenheit dieses Problem bei Encoder-Decoder Modellen und RNN im Allgemeinen beschrieben haben. Zudem ist die Menge der verwendeten Daten im Gegensatz zu anderen Szenarien, wie beispielsweise der Sprachübersetzung, relativ klein. Dies erhöht die Anfälligkeit für Overfitting.

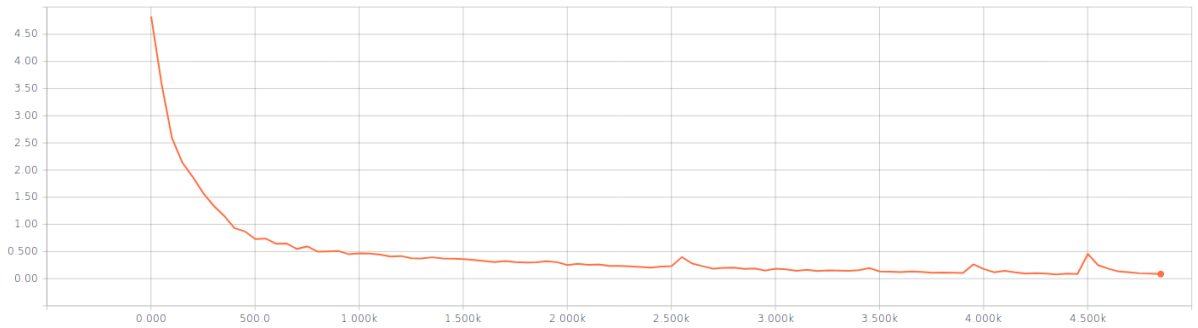

Abb. 9: Trainingsfehler der mit Ausgangskonfiguration

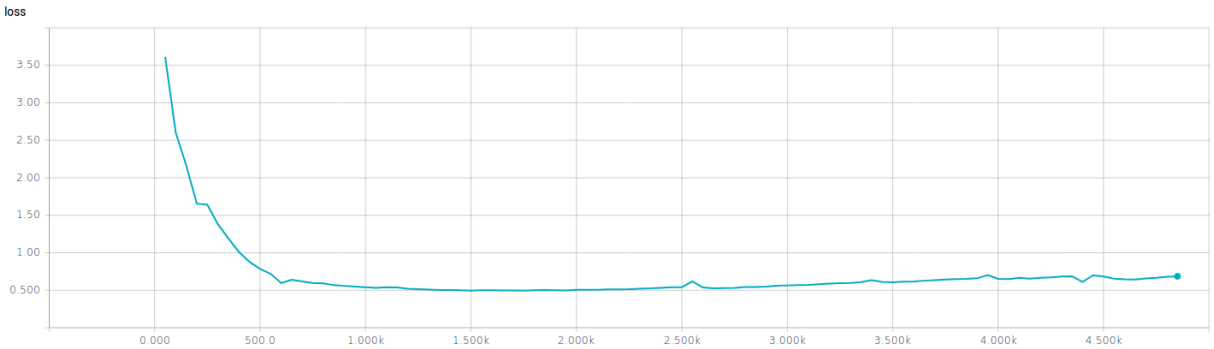

Abb. 10: Validierungsfehler mit der Ausgangskonfiguration

Aus Abbildung 10 lässt sich entnehmen, dass nach einer ersten Senkung des Fehlers auf dem Validierungsset, der Fehler wieder steigt. Parallel dazu sinkt allerdings der Trainingsfehler in Abbildung 9 weiter. Dies deutet auf Overfitting hin. Im Folgenden werden daher verschiedene Dropout-Raten untersucht.

5.3 Hinzufügen von Dropout

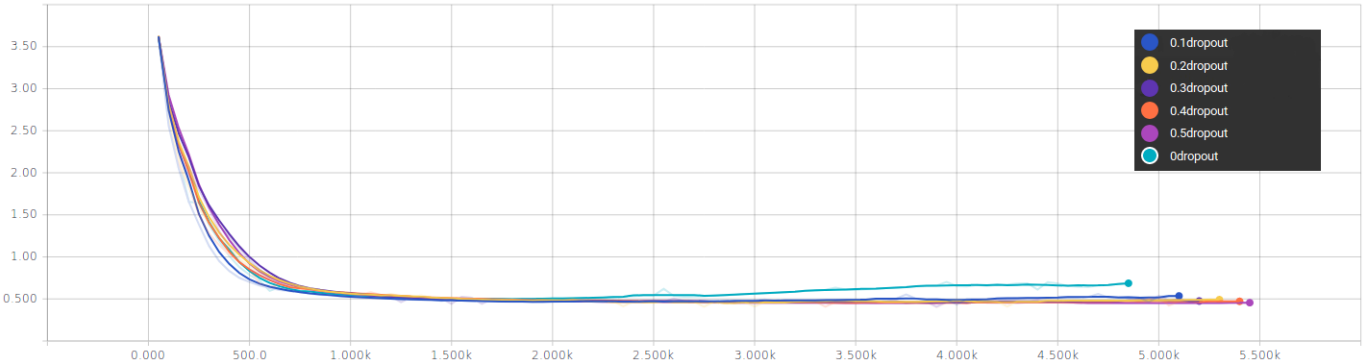

Abb. 11: Vergleich von den Dropout-Raten 0, 0.1, 0.2, 0.3, 0.4 und 0.5 anhand des Validierungsfehlers

Abbildung 11 zeigt, dass das Hinzufügen von Dropout deutlich das zuvor beobachtete Overfitting eindämmt. Da eine Dropout-Rate von 0.5 den hat den geringsten Fehler erzielt, wird dieser Wert für die weiterhin folgenden Konfigurationen verwendet.

5.4 Beste Konfiguration

Weiterhin wurden die Folgenden Konfigurationen getestet:

Konfiguration	Anzahl der LSTM-Zellen	Anzahl der Schichten	Bidirektional
1	256	1	Nein
2	256	2	Nein
3	256	3	Nein
4	512	1	Nein
5	128	1	Nein
6	128	2	Nein
7	128	3	Nein

8	128	2	Ja
9	256	2	Ja

Den geringsten Fehler in der Validierung hat Konfiguration 8 mit einem Wert von 0.4755 erzielt. Der Trainingsfehler lag bei einem Wert von 0.3630.

6. Auswertung

In diesem Kapitel wird die Erstellung und Durchführung der Umfrage zur Evaluierung der erzielten Ergebnisse durchgeführt. Zur Durchführung wurde die Plattform SoSci Survey[3] eingesetzt.

6.1 Umfrage

Ein Durchlauf über alle Testsequenzen hat 219 Sequenzen erzeugt. Zur Erstellung der Umfrage wurden davon 50 Ausgabesequenzen händisch ausgewählt. Zudem wurden die jeweiligen originalen Sequenzen ermittelt. Anschließend wurden bei jeder Durchführung 5 Sequenzpaare zufällig gezogen und dem Teilnehmer präsentiert. Der Teilnehmer sollte anschließend entscheiden, welche der beiden gehörten Sequenzen automatisch generiert wurde.

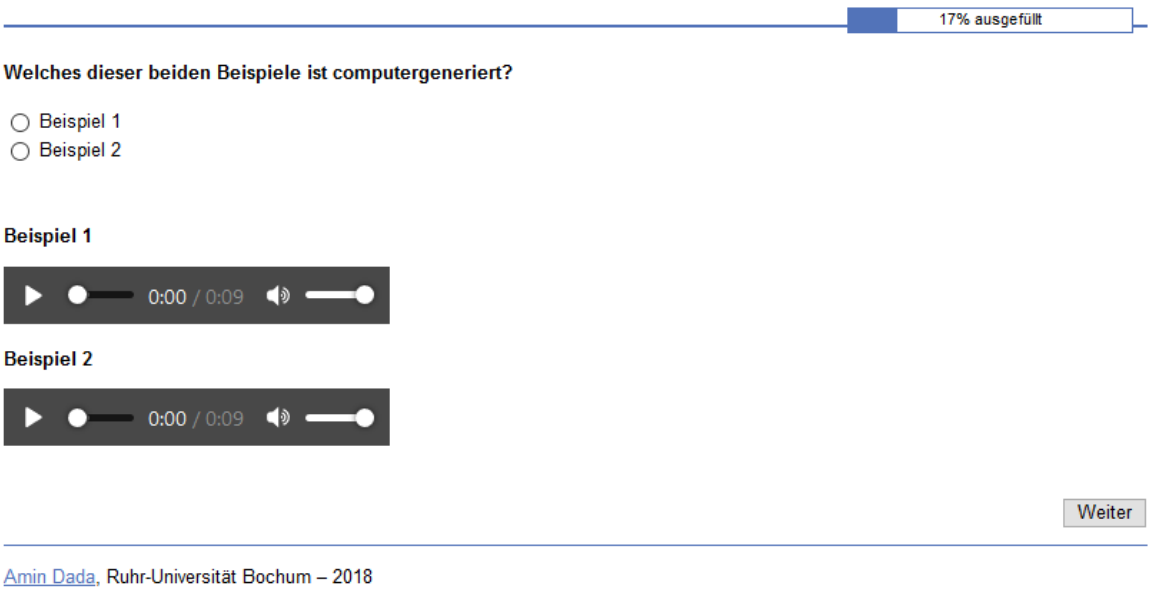

Abb. 9: Fragebogen der Umfrage

Die Teilnehmer wurden in drei Gruppen unterteilt:

- Gruppe 1: Teilnehmer, die kein Instrument spielen.
- Gruppe 2: Teilnehmer, die ein Instrument spielen.
- Gruppe 3: Teilnehmer, die Musik studiert haben.

[3] https://www.soscisurvey.de/

Eine genaue Anzahl an Teilnehmern ist nicht zu ermitteln, da die Umfrage beliebig oft von derselben Person wiederholt werden konnte. Allerdings ist durch die zufällige Ziehung die Anzahl an möglichen Kombinationen hoch: $\binom{50}{5}$ = 2118760. Dadurch können auch mehrmalige Teilnahmen derselben Person den Informationsgehalt des Ergebnisses erhöhen.
Die Umfrage wurde ausschließlich durch Veröffentlichungen in Social Media und persönliche Gespräche verbreitet.

6.2 Ergebnisse

Die Umfrage wurde innerhalb von 10 Tagen 450 Mal durchgeführt. Da die Anzahl der Teilnehmer der Gruppe 3 sehr gering ausgefallen ist, werden die Gruppen 2 und 3 im Folgenden zusammengefasst.

Tabelle 7: Prozentsatz der richtigen Antworten

Teilnehmergruppe 1	Teilnehmergruppe 2 und 3
50,69306%	51,09311%

Den Ergebnissen der Umfrage ist zu entnehmen, dass den Teilnehmern aller Gruppen eine Unterscheidung schwerfiel. Der Prozentsatz der richtigen Antworten unterscheidet sich bei beiden Gruppen nur marginal von einer zufälligen Auswahl von Antwortmöglichkeiten. Die Teilnehmer der Gruppen 2 und 3 haben einen leicht höheren Anteil an richtigen Antworten.

Literaturverzeichnis

[1] Denny Britz, Anna Goldie, Minh-Thang Loung, Quoc Le, *Massive Exploration of Neural Machine Translation Architectures*, März 2017. arXiv:1703.03906.

[2] Diederik P. Kingma, Jimmy Lei Ba, *Adam: A Method for statistical optimization*, Januar 2017. arXiv:1412.6980.

[3] Jeffrey L. Elman, *Finding Structure in Time*, Cognitive Science, 14: 179–211, März 1990. doi:10.1207/s15516709cog1402_1

[4] Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio, *How to Construct Deep Recurrent Neural Networks*, April 2014. arxiv:1312.6026

[5] Herbert Jaeger. *A tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the "echo state network" approach.* Oktober 2002. Url: http://minds.jacobs-university.de/sites/default/files/uploads/papers/ESNTutorialRev.pdf

[6] Sepp Hochreiter. *Untersuchungen zu dynamischen neuronalen Netzen*, Juni 1991. Url: http://people.idsia.ch/~juergen/SeppHochreiter1991ThesisAdvisorSchmidhuber.pdf

[7] Yoshua Bengio, Patrice Simard, Paolo Frasconi, Learning Long-Term Dependencies with Gradient Descent is Difficult. März 1994. Url: http://ai.dinfo.unifi.it/paolo/ps/tnn-94-gradient.pdf

[8] Sepp Hochreiter, Jürgen Schmidhuber, *LONG SHORT-TERM MEMORY*, in Neural Computations 9: 1735-1780, 1997. Url: http://www.bioinf.jku.at/publications/older/2604.pdf

[9] Razvan Pascanu, Tomas Mikolov, Yoshua Bengio, *On the difficulty of training Recurrent Neural Networks*, Februar 2013. arXiv:1211.5063

[10] Anirudh Goyal, Alex Lamb, Ying Zhang, Saizheng Zhang, Aaron Courville, Yoshua Bengio, *Professor Forcing: A New Algorithm for Training Recurrent Networks*, Oktober 2016. arXiv:1610.09038

[11] Michael Scott Cuthbert, Christoper Ariza, *music21: A Toolkit for Computer-Aided Musicology and Symbolic Music Data*, 11th International Society for Music Information Retrieval Conference (ISMIR 2010), August 2010. Url: http://hdl.handle.net/1721.1/84963

[12] Douglas Eck, Jürgen Schmidthuber, *Finding temporal structure in music: blues improvisation with LSTM recurrent networks, Neural Networks for Signal Processing*, 2002. Proceedings of the 2002 12th IEEE Workshop on, September 2002. DOI: 10.1109/NNSP.2002.1030094

[13] Nicolas Boulanger-Lewandowski, Yoshua Bengio, Pascal Vincent, *Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription*, Juni 2012. arXiv:1206.6392

[14] Hans Bauer, *Untersuchung über die Existenz von Tonarten-Charakteristiken im 21. Jahrhundert*, August 2016.
Url: http://habauer.de/ba_tonartencharakteristiken_21_jahrhundert.pdf

[15] Carol L. Krumhansl, *Cognitive Foundations of Musical Pitch, Oxford Psychology Series No. 17*, 1990. Url:
http://www.scar.utoronto.ca/~marksch/psyc56/Krumhansl%201990%20Ch%204.pdf

[16] Feynman Liang, BachBot: *Automatic composition in the style of Bach chorales*, August 2016. Url: http://www.mlsalt.eng.cam.ac.uk/foswiki/pub/Main/ClassOf2016/Feynman_Liang_8224771_assignsubmission_file_LiangFeynmanThesis.pdf

[17] Ilya Sutskever, Oriol Vinyals, Quoc V. Le, *Sequence to Sequence Learning with Neural Networks*, Dezember 2014. arXiv:1409.3215.

[18] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, Yoshua Bengio, *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*, September 2014. arXiv:1406.1078.

[19] Graham Neubig, Neural Machine Translation and Sequence-to-sequence Models: A Tutorial, März 2017. arXiv:1703.01619

[20] Gaëtan Hadjeres, François Pachet, Frank Nielsen, *DeepBach: a Steerable Model for Bach Chorales Generation*, Juni 2017. arXiv:1612.01010.

[21] Mike Schuster, Kuldip K. Paliwal, *Bidirectional Recurrent Neural Networks*, IEEE Transactions on Signal Processing, vol. 45, no. 11, November 1997. Url: https://max-well.ict.griffith.edu.au/spl/publications/papers/ieeesp97_schuster.pdf

[22] Alex Graves, Jürgen Schmidhuber, *Framewise Phoneme Classification with Bidirectional LSTM and Other Neural Network*, Proceedings. 2005 IEEE International Joint Conference on Neural Networks, Juli 2005. Url: ftp://ftp.idsia.ch/pub/juergen/nn_2005.pdf

[23] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, Ruslan R. Salakhutdinov, *Improving neural networks by preventing co-adaptation of feature detectors*, Juli 2012. arXiv:1207.0580

Erklärung

Ich erkläre, dass das Thema dieser Arbeit nicht identisch ist mit dem Thema einer von mir bereits für eine andere Prüfung eingereichten Arbeit.
Ich erkläre weiterhin, dass ich die Arbeit nicht bereits an einer anderen Hochschule zur Erlangung eines akademischen Grades eingereicht habe.

Ich versichere, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Die Stellen der Arbeit, die anderen Werken dem Wortlaut oder dem nach entnommen sind, habe ich unter Angabe der Quellen der Entlehnung kenntlich gemacht. Dies gilt sinngemäß auch für gelieferte Zeichnungen, Skizzen, bildliche Darstellungen und dergleichen.

_____ _____

Datum Unterschrift