Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Florian Schultz

Prognose quasiperiodischer Sequenzen mit bidirektionalen LSTM-Netzwerken

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Florian Schultz

Prognose quasiperiodischer Sequenzen mit bidirektionalen LSTM-Netzwerken

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Andreas Meisel
Zweitgutachter: Prof. Dr. Wolfgang Fohl

Eingereicht am: 7. Juni 2019

Florian Schultz

Thema der Arbeit

Prognose quasiperiodischer Sequenzen mit bidirektionalen LSTM-Netzwerken

Stichworte

Long-Short-Term-Memory, LSTM, bidirektionales LSTM, BLSTM, Rekurrente Neuronale Netze, RNN, quasiperiodische Sequenzen, Musical Instrument Digital Interface, MIDI, Keras, Tensorflow

Kurzzusammenfassung

Eine Vielzahl an Problemen wird in der heutigen Zeit mithilfe komplexer Algorithmen auf immer leistungsfähigeren Computern gelöst. Probleme, die sich nicht einfach in einen solchen Algorithmus bringen lassen, müssen jedoch auf andere Weise gelöst werden. Der Mensch begegnet diesen Aufgaben, indem er seine Erfahrungen verwendet und somit auf sein Training zurückgreift. Um Computern dieses Verhalten zu ermöglichen, wurde das Konzept der künstlichen neuronalen Netze entwickelt. Vor allem Long-Short-Term-Memory-Netzwerke (LSTM-Netzwerke) sind gut geeignet, um sich Abhängigkeiten über einen gewissen zeitlichen Abstand zu merken und dementsprechend Ergebnisse zu produzieren. Hierfür werden dem Netz oftmals lange Sequenzen an Daten übergeben. Im Rahmen dieser Bachelorarbeit wird überprüft, wie gut sich LSTM-Netzwerke und insbesondere bidirektionale LSTM-Netzwerke für die Prognose quasiperiodischer Sequenzen eignen. In verschiedenen Experimenten wird getestet, wie sich die Komplexität dieser Sequenzen auf das Training der Netze auswirkt und inwieweit Fehler in den Inputsequenzen durch die Netze kompensiert werden können.

Florian Schultz

Title of the paper

Prognosis of quasiperiodic sequences with bidirectional LSTM networks

Keywords

Long-Short-Term-Memory, LSTM, bidirectional LSTM, BLSTM, Recurrent Neural Networks, RNN, Quasiperiodic Sequences, Musical Instrument Digital Interface, MIDI, Keras, Tensorflow

Abstract

A multitude of problems is being solved today with the help of complex algorithms on increasingly powerful computers. However, problems that can not be easily put into such an algorithm have to be solved in another way. Man meets these tasks by using his experience and thus accessing his training. In order to make this behavior possible for computers, the concept of artificial neural networks was developed. Especially long-short-term-memory networks (LSTM networks) are well suited to remember dependencies over a certain time interval and to produce results accordingly. For this, the network is often given long sequences of data. This bachelor thesis examines the suitability of LSTM networks and in particular bidirectional LSTM networks for the prognosis of quasiperiodic sequences. How the complexity of these sequences affects the training of the networks and to what extent errors in the input sequences can be compensated is tested in several experiments.

Inhalt

Tabellen

Abbildungen

Quelltexte

Akronyme

BLSTM bidirektionales Long short-term memory. viii–x, 1, 11, 12, 21–26, 29, 31, 33, 35, 42

BPTT Backpropagation Through Time. 4, 5

LSTM Long short-term memory. v, viii, x, 1, 2, 7–16, 21–24, 33, 42

MIDI Musical Instrument Digital Interface. viii, 16–20, 22, 24

RNN rekurrentes neuronales Netz. v, viii, 1, 3, 4, 6, 7, 11, 13, 14

TBPTT Truncated Backpropagation Through Time. 5

Glossar

Accuracy Die Präzision des Netzwerks für die Trainingsdaten korrekte Ergebnisse zu produzieren. 29

Binary Accuracy Die Übereinstimmung des berechneten und erwarteten Outputs binär verglichen. 22

Dense Eine Schicht mit vollständig verbundenen Neuronen. 21

Dropout Eine Schicht, die je nach Einstellung einen bestimmten Teil der Neuronen pro Durchlauf abschaltet, um ein auswendig lernen der Daten vom Netzwerk zu erschweren. 21, 24

Features Definiert die Anzahl an Dimensionen, die dem Netzwerk pro Zeitschritt zugeführt werden. In dieser Arbeit gibt es 27 mögliche gespielte Noten und dementsprechend auch Features. 19

Layer Eine Schicht des Netzwerks. 3, 11, 12, 14, 21, 30

Loss Der geschätzte Fehler des Outputs des Netzwerks zum erwarteten Output. 4

Multi-Hot-Encoding Das Überführen der Werte in einen binären Vektor mit der Länge der Anzahl an möglichen Werten. In dieser Arbeit wird bspw. jede gespielte Note mit einer Eins im Vektor gekennzeichnet. 19

Overfitting Das Phänomen, dass auftritt, wenn das Netzwerk die Trainingsdaten auswendig gelernt hat. Daraus folgt, dass das Netzwerk nur die Trainingsdaten beherrscht und schlecht auf unbekannte Daten reagiert. 21, 24, 26

Timesteps Definiert die Anzahl an Zeitschritten, die das Netzwerk in die Vergangenheit schauen kann und gibt somit auch die Länge des Inputs vor. 7, 19

Validation Accuracy Die Präzision des Netzwerks für die Validierungsdaten korrekte Ergebnisse zu produzieren. viii, ix, 22–26, 29, 33

1. Einleitung

1.1. Motivation

In der heutigen Zeit sind Computer in der Lage eine Vielzahl an Problemen durch Algorithmen zu lösen. Eine Voraussetzung dafür ist, dass solch ein Algorithmus existiert. Für komplexere Probleme besitzt der Mensch gegenüber einem Computer jedoch einen entscheidenden Vorteil: Er kann lernen. Um Computern diese Fähigkeit beizubringen, wurde das Konzept der künstlichen neuronalen Netze entwickelt. In jüngster Zeit werden die rekurrenten neuronalen Netzwerke (RNN) und insbesondere die Long short-term memory Netzwerke erfolgreich in verschiedenen Wettbewerben eingesetzt [12][13]. Ebenfalls setzen große Technologieunternehmen wie Google und Apple LSTM-Netzwerke ein. Hier werden sie beispielsweise für Google Translate und Siri verwendet [9][2]. Der Anwendungsbereich neuronaler Netze erstreckt sich über ein großes Themenfeld von Fehlererkennung, Klassifikation und Optimierung bis hin zu Musikgenerierung.
Eine Einführung in die Effektivität der RNN- und LSTM-Netzwerke bietet Andrej Karpathy in seinem Blogpost *The Unreasonable Effectiveness of Recurrent Neural Networks* [26].

1.2. Anwendungsfall

Ein praxisbezogenes Anwendungsgebiet für die Nutzung von neuronalen Netzen ist die Generierung von Musik. Durch den rhythmisch repetitiven Aufbau vieler Musikgenres eignen sich deren Signale als gute Vertreter für quasiperiodische Sequenzen. Außerdem kann die Richtigkeit von generierten Rhythmus-Spuren akkustisch und visuell sehr leicht beurteilt werden. In dieser Arbeit soll deshalb am Beispiel von selbst generierten Schlagzeugspuren untersucht werden, wie sich LSTM und insbesondere BLSTM für die Prognose quasiperiodischer Sequenzen nutzen lassen.
In verschiedenen Experimenten wird kontinuierlich die Komplexität dieser Inputs gesteigert, um zu testen, wie sich dies auf das Training auswirkt. Des Weiteren wird überprüft inwieweit

die Netze in der Lage sind Fehler in den Inputsequenzen durch ihr Wissen zu kompensieren. Für diese Arbeit wird ein grundlegendes Wissen über neuronale Netze vorausgesetzt.

1.3. Kapitelkurzzusammenfassung

Die Arbeit ist in folgende Kapitel gegliedert:

2 Grundlagen erklärt die verwendeten Konzepte und Netzwerkarchitekturen, sowie deren Vor- und Nachteile.

3 Stand der Technik gibt einen Überblick über die aktuelle Forschung im Bereich LSTM und im Zusammenhang mit der Musikgenerierung.

4 Erzeugung und Aufbereitung der Daten erklärt die Wahl der Trainingsdaten und die Aufbereitung dieser für die Verwendung im Training der Netzwerke.

5 Experimente beschreibt den Aufbau und die Durchführung der Experimente und stellt die jeweiligen Ergebnisse vor.

6 Auswertung diskutiert die Beobachtungen und Ergebnisse aller Experimente zusammenfassend.

7 Fazit fasst die Erkenntnisse der Arbeit zusammen und gibt einen Ausblick.

2. Grundlagen

Neuronale Netze erzeugen Ergebnisse, indem sie anhand von Trainingsdaten eine Funktion erlernen. Diese Funktion wird durch Gewichte innerhalb des Netzes eingestellt. Das Ziel ist, es die Differenz zwischen den erwarteten und den tatsächlichen Ergebnissen möglichst gering werden zu lassen.

2.1. Rekurrente neuronale Netze

Rekurrente neuronalen Netze (RNN) besitzen, im Gegensatz zu Feedforward-Netzen, Verbindungen von Neuronen eines Layers, zu Neuronen des selben oder eines vorherigen Layers.

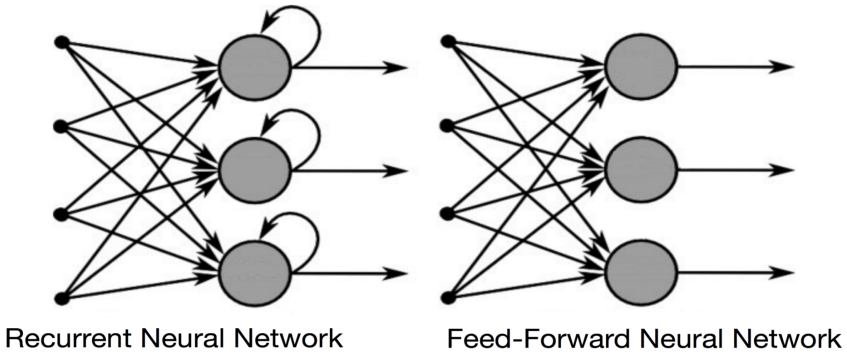


Abb. 2.1.: Rekurrentes vs. Feed-Forward Netz [24]

RNNs sind dafür gedacht, mithilfe ihrer Rückkopplungen auf Wissen vorheriger Durchläufe zurückzugreifen und somit ein Gedächtnis aufzubauen. Besonders wirksam sind diese Netze, wenn sie mit Sequenzen trainiert werden, bei denen die Inputs abhängig vom vorherigen Input sind.
Ein Beispiel dafür ist die buchstabenweise Generierung eines Textes. In diesem Fall ist es rele-

vant, welche Buchstaben zuvor generiert wurden, damit ein passender Buchstabe ausgegeben werden kann.

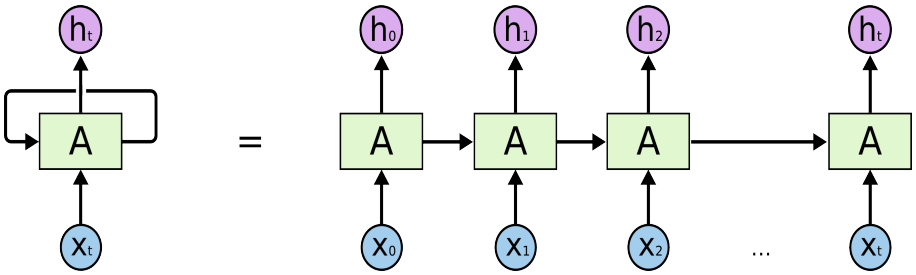


Abb. 2.2.: entrolltes rekurrentes neuronales Netz [6]

In Abbildung 2.2 ist auf der linken Seite eine vereinfachte Darstellung eines RNN-Neurons zu sehen. Auf der rechten Seite ist eine zeitlich entrollte Version des Neurons dargestellt. Im ersten Zeitschritt ist x_0 der Input und h_0 der Output. Beim nächsten Schritt bekommt das Neuron nicht mehr nur noch x_1, sondern ebenfalls den Output h_0 des vorherigen Durchlaufs als Input. Somit ist das Ergebnis vom vorherigen abhängig.

2.2. Training durch Backpropagation (Through Time)

Um die Gewichte in einem neuronalen Netz anzupassen, wird das sogenannte Backpropagation-Verfahren angewendet. Zuerst werden die Eingaben durch das Netz propagiert. Im darauf folgenden Schritt wird das errechnete Ergebnis mit dem erwarteten Ergebnis des Trainingsdatensatzes verglichen und der Fehler (Loss) berechnet. Mit diesem Fehler wird nun rückwärts durch das Netz propagiert um die Gewichte mithilfe einer Optimierungsfunktion (wie z.B. Gradient Descent), abhängig von ihrer Beteiligung am Fehler, einzustellen. Somit wird erreicht, dass die Abweichung zwischen dem tatsächlichen und dem erwarteten Ergebnis möglichst gering ist.
Für RNNs muss dieses Verfahren noch um die zeitliche Ebene erweitert werden, da der Fehler hier nicht nur vom aktuellen, sondern auch von vorherigen Inputs abhängt. Dieser Vorgang wird auch Backpropagation Through Time (BPTT) genannt. Hierfür wird dem Netz eine Sequenz an In- und Output-Paaren zugeführt. Anschließen wird es, wie in Abbildung 2.2, entrollt. Die Anzahl der entrollten Ebenen ist abhängig von der Länge der Eingabesequenz und jede Ebene stellt somit einen Zeitschritt innerhalb der Eingabe dar. Auf das entrollte Netz kann nun wieder Backpropagation angewendet werden, um den Fehler zu bestimmen und die Gewichte anzupassen.

Ein Problem von BPTT ist jedoch, dass ein Update sehr kostenintensiv sein kann. Da für jeden Schritt der Eingabesequenz eine Kopie des Netzes beim Entrollen erstellt wird, ist es nicht praktikabel dieses Verfahren bei langen Sequenzen anzuwenden.
Truncated Backpropagation Through Time (TBPTT) wirkt diesem Verhalten entgegen. Dabei wird durch Parameter festgelegt, wie viele Timesteps dem Netzwerk zugeführt werden, bis ein Update der Gewichte durchgeführt wird und wie viele vergangene Timesteps für die Berechnung berücksichtigt werden. Somit werden lange Sequenzen in leichter zu verarbeitende Sequenzen aufgeteilt. Auch hier gibt es jedoch Einschränkungen, da die zu erlernenden Informationen innerhalb einer Teilsequenz liegen müssen. Für eine Sequenz, wie beispielsweise die Sinusfunktion, ist dies nicht sehr relevant, da jede Stelle der Funktion mit wenigen Timesteps eindeutig zu beschreiben ist. Sobald es jedoch Abschnitte der Sequenz gibt, die mehr Timesteps brauchen, als die Teilsequenzen bieten, um eindeutig identifiziert zu werden, läuft man Gefahr, die Langzeitabhängigkeiten nicht zu erlernen.

2.3. Verschwindende und explodierende Gradienten

Wenn ein Update der Gewichte im Netzwerk durchgeführt wird, muss der Fehler wieder komplett durch das Netz propagiert werden. Dies betrifft alle Neuronen, die an diesem Output beteiligt waren und somit auch die zeitlich weit zurückliegenden.
Ein Problem stellen hier die Gewichte dar, die die zeitlich entrollten Neuronen miteinander verbinden. In Abbildung 2.3 ist ein solcher Vorgang abgebildet und diese Gewichte sind mit W_{rec} gekennzeichnet. Die Informationen, die durch das Netz fließen, werden so vielfach mit diesen Gewichten multipliziert. Je länger eine Information im Netz vorhanden ist, desto öfter wird sie multipliziert. Wenn dabei jedes Mal mit einem Wert größer als Eins multipliziert wird, wird der Gradient auf lange Sicht unendlich groß und es kommt zu dem sogenannten explodierenden Gradienten. Ebenfalls kann es auch zu einem verschwindenden Gradienten kommen, wenn der Faktor kleiner als Eins ist.
Für das Problem des explodierenden Gradienten gibt es mehrere Lösungsansätze. So ist es z.B. möglich, Backpropagation nur bis zu einem bestimmten Punkt durchzuführen, was jedoch nicht optimal ist, da nicht alle Gewichte aktualisiert werden. Eine weitere Option ist es, die Gradienten durch ein Maximum zu limitieren.
Für das Problem der verschwindenden Gradienten ist es deutlich schwieriger, eine Lösung zu finden. Um das Problem etwas abzuschwächen, können die Gewichte manuell mit Werten initialisiert werden, die nicht sehr viel kleiner als Eins sind.

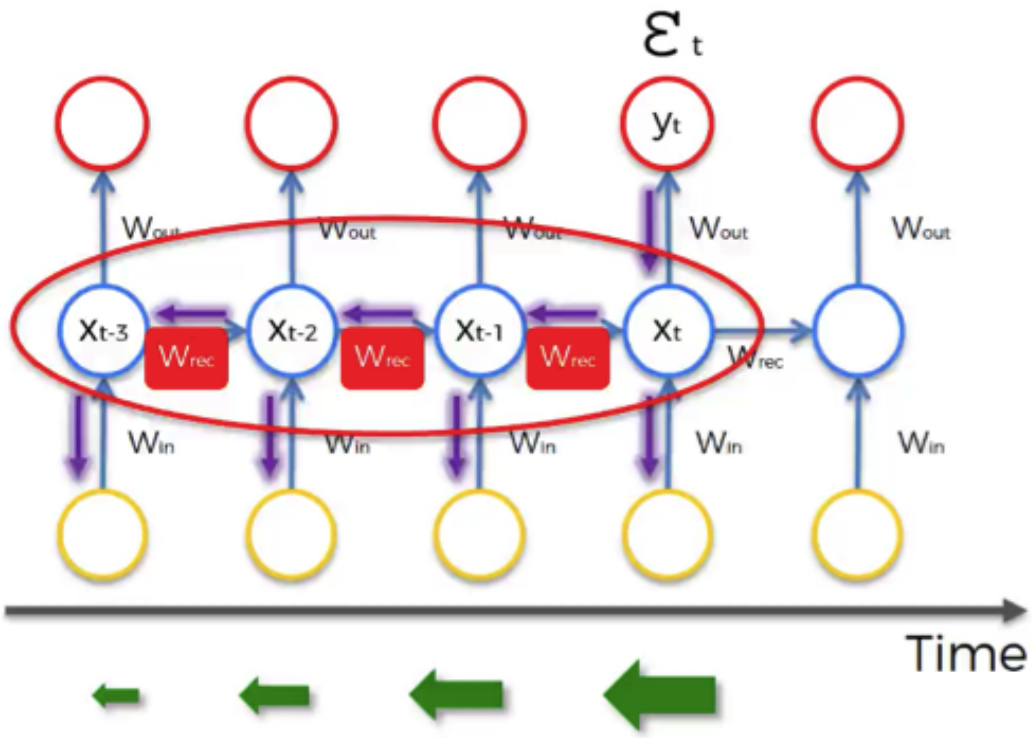


Abb. 2.3.: Darstellung von verschwindenden und explodierenden Gradienten [23]

2.4. Langzeitabhängigkeiten

Ein Problem von herkömmlichen RNNs ist, dass sie sich Zusammenhänge nur in einem recht kurzen zeitlichen Abstand merken können. Ist zwischen den Inputs ein größerer zeitlicher Abstand vorhanden, kann der Kontext verloren gehen.
Christopher Olah benutzt in seinem Blogeintrag von 2015 die folgenden Beispiele um das Problem zu verdeutlichen: Das letzte Wort in dem Satz „Die Wolken sind im Himmel“ vorauszusagen, ist recht einfach. Es ist kein weiterer Kontext notwendig, um von „Wolken“ auf „Himmel“ zu schließen. Ist der Abstand der relevanten Informationen jedoch größer, wie z.B. im Fall „Ich bin in Frankreich aufgewachsen. (...) Ich spreche fließend Französisch.“, wird es schwierig, das korrekte Wort zu wählen. Es kann zwar anhand der letzten Informationen eingegrenzt werden, dass das nächste Wort womöglich eine Sprache ist, aber der Kontext Frankreich liegt möglicherweise zu weit zurück. Wenn dieser Abstand zu groß wird, ist es dem RNN nicht mehr

möglich diesen Zusammenhang zu lernen.[6]
Für genau diesen Anwendungsfall sind Long short-term memory (LSTM) Netzwerke entwickelt worden. Diese sind durch ihren Aufbau besonders gut dafür geeignet, Informationen lange zu speichern und haben außerdem nicht das Problem der verschwindenden und explodierenden Gradienten.

2.5. LSTM-Netzwerke

Das LSTM-Netzwerk ist ein spezielles RNN, das besonders gut mit Langzeitabhängigkeiten zurechtkommt. Erstmals beschrieben wurden die LSTM-Netzwerke 1997 von Sepp Hochreiter und Jürgen Schmidhuber und seitdem stetig weiterentwickelt [10]. Eine Übersicht über die Entwicklungen und Forschungsfragen im Bereich RNN und insbesondere LSTM bietet die Ausarbeitung „Recent Advances in Recurrent Neural Networks“ von Salehinejad, Baarbe, Sankar u. a. [20].
Der Aufbau der LSTM-Zelle ist im Gegensatz zu einer Standard-RNN-Zelle deutlich komplexer.

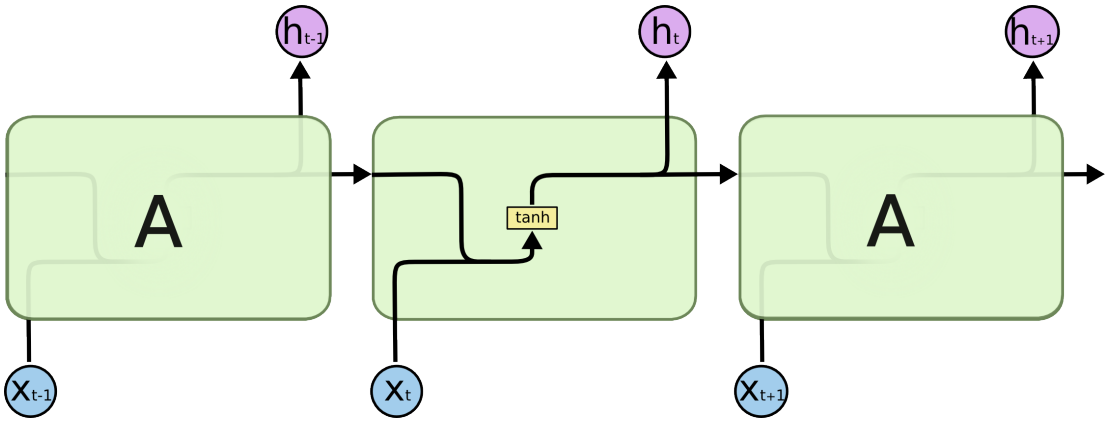


Abb. 2.4.: Darstellung einer einfachen RNN-Zelle [6]

Eine einfache RNN-Zelle ist in Abbildung 2.4 über drei Timesteps dargestellt. Im Inneren besitzt sie nur eine Funktion zur Berechnung und Weitergabe von Ergebnissen.

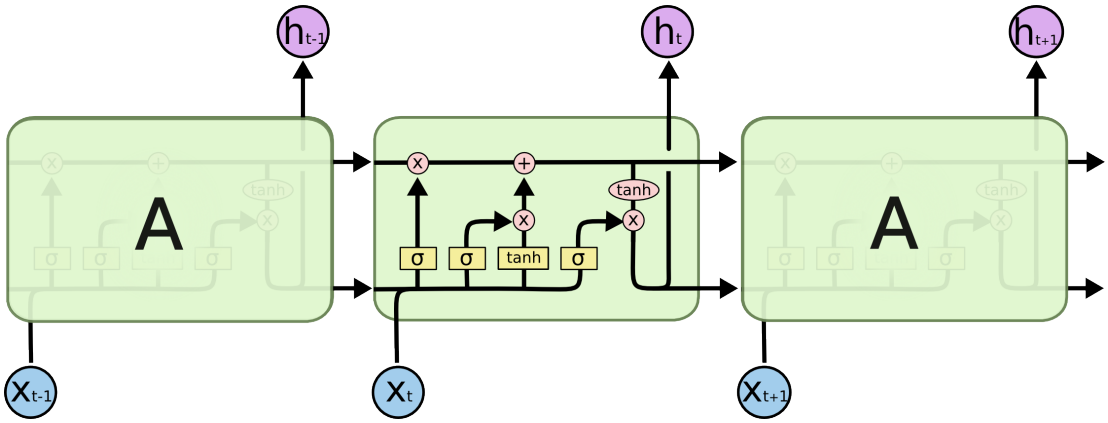


Abb. 2.5.: Darstellung einer LSTM-Zelle [6]

Im Vergleich dazu ist in Abbildung 2.5 eine LSTM-Zelle zu sehen. Diese besitzt vier Funktionen, die in Zusammenarbeit für ein kontrolliertes Speichern und Vergessen von Informationen sorgen. Es gibt viele verschiedene Arten von LSTM-Zellen, die leicht voneinander abweichen. Die Funktionsweise einer einfachen Zelle wird im folgenden Abschnitt erläutert.

2.5.1. Aufbau einer LSTM-Zelle

Zellzustand

Der wichtigste Teil der Zelle ist der Zellzustand, der hauptsächlich die zu haltenden Informationen der Zelle als Input bekommt und auch wieder ausgibt (siehe Abbildung 2.6). Die LSTM-Zelle besitzt drei sogenannte Gates um die gehaltenen Informationen zu schützen und zu verändern.

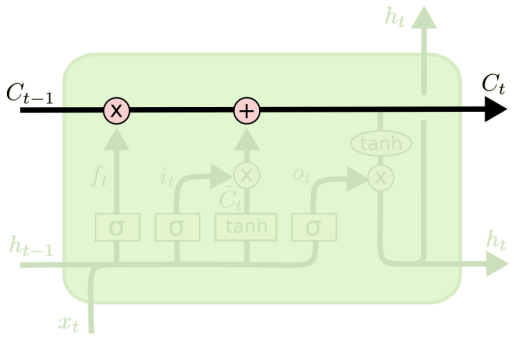
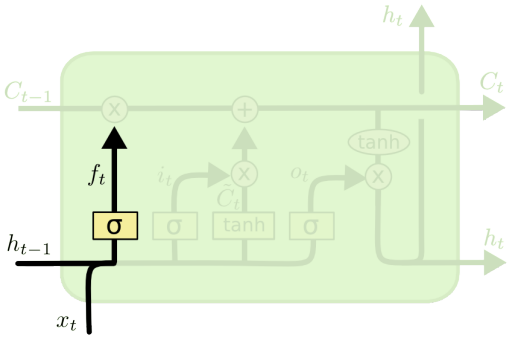


Abb. 2.6.: Zellzustand in einer LSTM-Zelle [6]

In Abbildung 2.6 ist der Zellzustand hervorgehoben. C_{t-1} ist der Zellzustand des vorherigen Zeitschritts. Innerhalb der Zelle können durch zwei Operatoren Informationen gelöscht und hinzugefügt werden.

Forget Gate

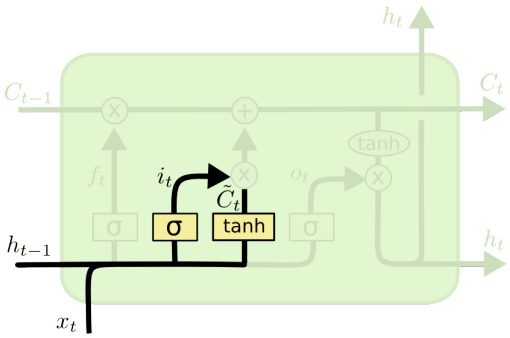


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Abb. 2.7.: Forget Gate einer LSTM-Zelle [6]

In diesem Teil der Zelle, dem Forget Gate, wird entschieden welche Informationen vergessen werden. Dazu guckt sich die Zelle den Output des vorherigen Zeitschritts h_{t-1} und den neuen Input x_t an und berechnet für jeden Eintrag in C_{t-1} einen Wert zwischen Null und Eins. Die Eins steht in diesem Fall dafür, den Eintrag komplett zu übernehmen, während er bei einer Null komplett vergessen wird.

Input Gate

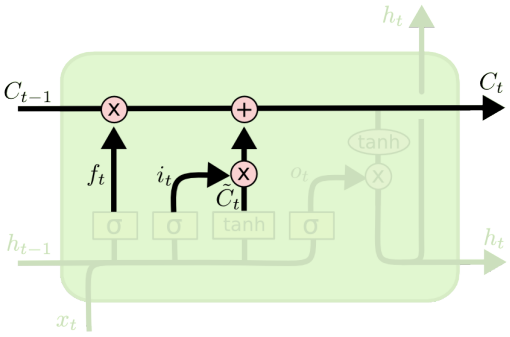


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Abb. 2.8.: Input Gate einer LSTM-Zelle [6]

Der in Abbildung 2.8 hervorgehobene Abschnitt der LSTM-Zelle ist in zwei Teile gegliedert. Der erste Teil, das Input Gate, entscheidet darüber, welche Werte aktualisiert werden (i_t). Danach wird ein Vektor $\tilde{C}_t$ mit Informationen erstellt, mit dem der Zellzustand aktualisiert werden kann.

Update

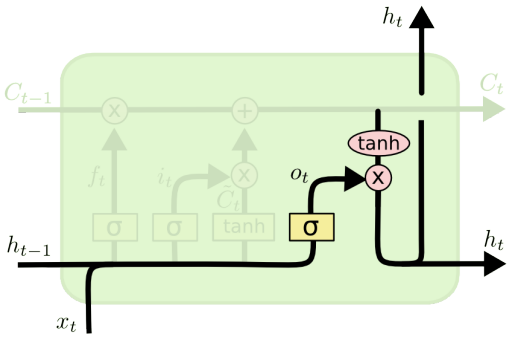


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Abb. 2.9.: Zustandsupdate in einer LSTM-Zelle [6]

Der Zellzustand wird jetzt mit den zuvor bestimmten Werten aktualisiert. Zuerst wird der alte Zustand mit f_t multipliziert, um die ausgewählten Informationen zu vergessen. Das Produkt aus i_t und $\tilde{C}_t$ bildet das Update und wird zum Zustand addiert.

Output



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Abb. 2.10.: Output einer LSTM-Zelle [6]

Als Letztes wird entschieden, was der Output der Zelle sein soll. Ähnlich wie bei dem Forget Gate, wo ein bestimmter Teil gelöscht wurde, wird hier bestimmt, welche Informationen des Zellzustands für den Output relevant sind (o_t). Die Werte des Zustands werden mithilfe der tanh-Funktion auf einen Bereich zwischen -1 und 1 gebracht, bevor sie mit o_t multipliziert werden um den Output h_t zu bilden.

2.6. Bidirektionale LSTM-Netzwerke

Eine Erweiterung der LSTM-Netzwerke stellen die bidirektionalen LSTM-Netzwerke (kurz: BLSTM-Netzwerk) dar. Die Idee der bidirektionalen RNN wurde 1997 von Schuster und Paliwal entwickelt [21]. Ein bidirektionaler LSTM-Layer besteht aus zwei LSTM-Layern.

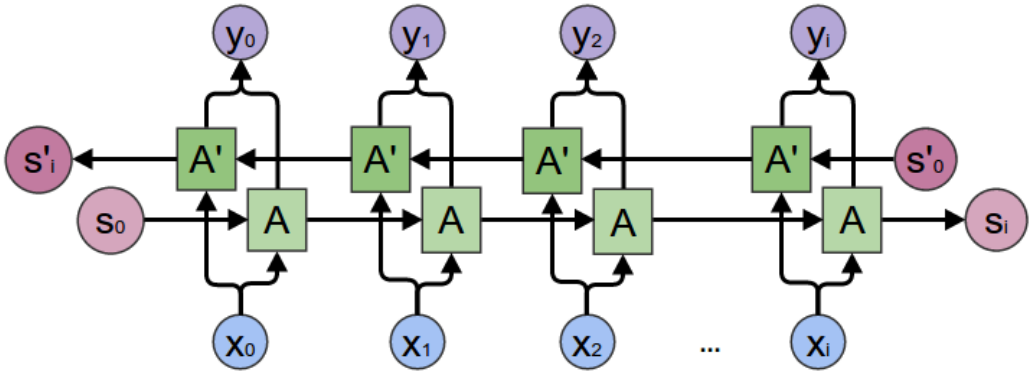


Abb. 2.11.: Output einer LSTM-Zelle [7]

Wie in Abbildung 2.11 zu sehen, wird die Input-Sequenz dem einen Layer in normaler Abfolge gegeben. Dem zweiten Layer wird die Sequenz in zeitlich umgekehrter Reihenfolge zugeführt. Der Output des bidirektionalen Layers wird dann aus den Ouputs der beiden LSTM-Layer zusammengesetzt.
Die Outputs der beiden Netze können auf verschiedene Weise miteinander kombiniert werden. Sie können aufsummiert, multipliziert, konkateniert werden oder es kann ein Mittelwert gebildet werden. Üblicherweise werden die Outputs konkateniert, da so die Outputs beider Layer unverändert weitergegeben werden und so am wenigsten Informationen verloren gehen.
Warum ein BLSTM-Netzwerk von Vorteil sein kann, lässt sich gut anhand eines Beispiels verdeutlichen: Normale LSTM-Netzwerke haben nur Zugriff auf die Informationen der Vergangenheit. Dies kann zu Problemen führen, wenn die Ausgabe des korrekten Ergebnisses von Informationen aus der Zukunft abhängt. Wenn das Netzwerk z.B. das fehlende Wort im Satz „Ich bin ... Student“ finden soll, bekommt das Standard-Netzwerk nur den Input „Ich bin“. Somit sind mehrere korrekte Optionen, wie z.B. „groß“, vorhanden.
Das bidirektionale Netz hingegen hat auch Informationen aus der Zukunft und weiß demnach, dass der Satz mit „Student“ beendet wird. Damit wird die Auswahl des richtigen Wortes, in diesem Fall „ein“, deutlich erleichtert.
BLSTM-Netzwerke haben jedoch nicht nur Vorteile. Im Vorfeld sollte genau überlegt werden, ob sich ein solches Netzwerk für das gegebene Problem eignet. Beispielsweise ist es dem BLSTM-Netzwerk nicht möglich, Inputdaten in Echtzeit zu verarbeiten. Damit die Sequenzen auch in der umgekehrten Reihenfolge dem Netzwerk übergeben werden können, müssen sie zu diesem Zeitpunkt bereits vollständig vorliegen.

3. Stand der Technik

Obwohl die Long short-term memory-Netzwerke bereits Mitte der 90er Jahre beschrieben wurden, erfreuen sie sich gerade in den letzten Jahren besonders großer Beliebtheit. In diesem Kapitel gibt es eine kurze Übersicht über die aktuelle Forschung im Zusammenhang mit LSTM-Netzwerken und ihre Zukunft, sowie einen Einblick in aktuelle Projekte zur Musikgenerierung.

3.1. LSTM/RNN-Forschung

3.1.1. Word Sense Disambiguation

Word Sense Disambiguation beschreibt eine Aufgabe im Bereich des Natural Language Processing (NLP). Deren Ziel ist es, Wörter in einem Text, die mehrere Bedeutungen haben können, mit der richtigen Bedeutung aus einer lexikalischen Datenbank zu versehen.
Beispielsweise hat das Wort „Bank“ in den beiden Sätzen „Die Bank ist am Samstag geschlossen.“ und „Im Park steht eine Bank.“ zwei verschiedene Bedeutungen. Diese gilt es zu erkennen und richtig zuzuordnen.
Bereits 2016 wurden mithilfe eines LSTM-Netzwerkes Bestwerte für diese Aufgabe, besonders im Zusammenhang mit Verben, erzielt [28]. Der Quellcode und auch der, eigens für das Experiment erstellte, große Trainingsdatensatz sind jedoch nicht frei zugänglich. Aus diesem Grund wurde eine reproduktive Studie durchgeführt, die mit einem deutlich kleinerem Datensatz vergleichbare Ergebnisse erreicht und frei zugänglich ist [27].

3.1.2. Alternative zu LSTM-Netzwerken

Eugenio Culurciello schreibt in seinem Blogeintrag *The fall of RNN / LSTM*: „Drop your RNN and LSTM, they are no good!“[11]. Ein Grund dafür ist, laut Culurciello, neben den verschwindenden und explodierenden Gradienten, dass das Training dieser Netze sehr ressourcenhungrig ist [8]. Außerdem sind laut Culurciello LSTM-Netzwerke in ihrer Fähigkeit, längere Sequenzen zu erlernen, auch noch recht begrenzt.
Als Beispiel für die Probleme der herkömmlichen RNN führt er die Arbeit mit Sequenzen an, bei denen weit in die Vergangenheit oder Zukunft geschaut werden muss, um das Ergebnis zu

bestimmen. Auch, wenn ein LSTM-Netzwerk hierfür zuerst geeignet erscheint, muss hier der Pfad von alten zu neuen Zellen sequentiell durchschritten werden. Dies führt laut Culurciello dazu, dass sich diese Netzwerke zwar Sequenzen mit einer Länge im Bereich der 100er merken können, jedoch nicht, wenn diese 1000 oder gar 10000 überschreitet.
Dieses Vorgehen kann durch neuronale Attention-Module [3] umgesetzt werden. Diese Module stellen eine Funktion dar, bei der durch eine Gewichtsmatrix eingestellt werden kann, welche Teile des Inputvektors durchgelassen werden sollen. Somit werden die Information, die nicht relevant sind, herausgefiltert und der Fokus auf die wichtigen Teile des Vektors gelenkt. Dabei wird unterschieden zwischen soft attention, wenn die Gewichte reelle Werte besitzen oder hard attention, wenn durch binäre Werte entweder alles (Eins) oder nichts (Null) durchgelassen wird.
In dem „hierarchical neural attention encoder“ (siehe Abbildung 3.1) werden die vergangenen Vektoren mithilfe solcher Module zu einem Kontextvektor C_t zusammengefasst. Angenommen ein Modul kann einen kleinen Teil der Vergangenheit, z.B. 100 Vektoren, sehen, so ist es durch den hierarchischen Aufbau möglich, weit in die Vergangenheit zu sehen. Das oberste Modul würde so 100 der Module sehen, die jeweils 100 Vektoren halten und so Informationen über $100 * 100$ Vektoren der Vergangenheit haben.
Ebenfalls ist es durch diesen Aufbau bedeutend schneller, neuen Input durch das Netzwerk zu propagieren. Die Anzahl der Schritte ist hier abhängig von der Anzahl der vorhandenen Layer, im Gegensatz zu einem normalen RNN, wo die Schrittzahl von der Länge der zu lernenden Sequenz abhängt und damit tendenziell höher ist.
Ein Problem dieses Aufbaus ist jedoch, dass jegliche Informationen im Speicher abgelegt werden, was ineffizient sein kann: Für die Analyse von Videomaterial ist es beispielsweise nicht notwendig jedes einzelne Bild im Speicher zu halten, wenn es zwischen zwei Bildern nur geringfügige Änderungen gibt. Eine Lösung hierfür ist, das Netz selber entscheiden zu lassen welche Vektoren relevant genug sind, um behalten zu werden – das umzusetzen ist Bestand der aktuellen Forschung.

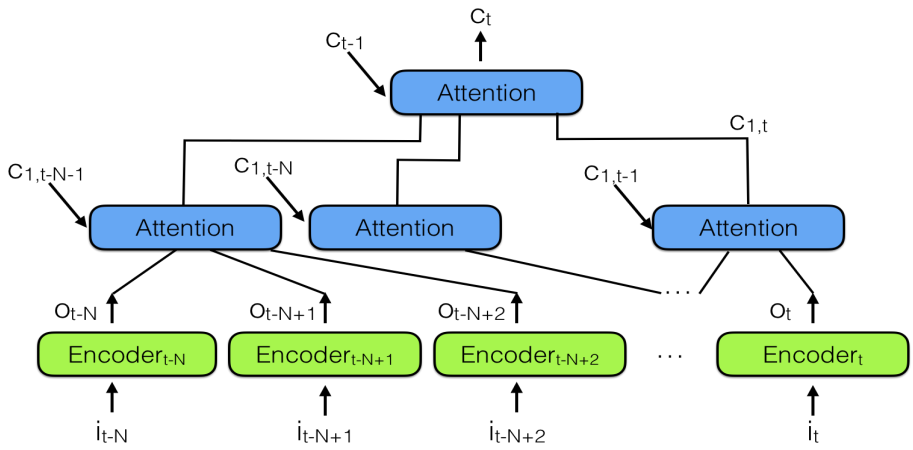


Abb. 3.1.: Hierarchical neural attention encoder [11]

3.2. Musikerzeugung

3.2.1. BachBot

BachBot [4] ist ein Projekt der Universität Cambridge. Mithilfe eines, anhand von Bach-Kompositionen trainierten, LSTM-Netzwerkes wird neue Musik generiert. Das Ziel ist es, dass die neuen Kompositionen möglichst nicht von echten Bach-Stücken zu unterscheiden sind. Die von BachBot generierten Stücke konnten von Testpersonen nur in 51% der Fälle von echten Bach-Kompositionen unterschieden werden, was nur 1% über dem Ergebnis bei zufälligem Raten liegt. Die Website des BachBot bietet einen Test an, bei dem man selber überprüfen kann, wie gut man die Stücke des BachBot von echten Stücken unterscheiden kann[1].
Der BachBot kann mit mehrstimmiger Musik umgehen und arbeitet am Besten, wenn eine Stimme fixiert ist und er die Begleitstimmen erzeugt [16].

3.2.2. Google Magenta

Magenta [14] ist ein Open-Source-Projekt von Google, bei dem der Einsatz von Machine Learning in unterschiedlichen Bereichen des kreativen Schaffens untersucht wird. Gestartet wurde das Projekt von Mitgliedern des Google-Brain-Teams und wird mittlerweile von vielen anderen mitentwickelt. Ziel ist es unter anderem, neue Tools zur Erzeugung von Bildern, Zeichnungen und auch Musik zu erstellen, die Künstler in Zukunft unterstützen sollen. Im

[1] Offizielle Website: https://www.bachbot.com/

Bereich der Musik arbeitet Magenta mit MIDI-Dateien als In- und Output. Zahlreiche andere Projekte basieren auf der Arbeit und den Interfaces von Magenta.

3.2.3. A.I. Duet

A.I. Duet [1] ist ein Experiment, dass es ermöglicht, zusammen mit einem Neuronalen Netz im Duet zu spielen. Als Grundlage dienen auch hier, neben Tensorflow, Tools des Magenta-Projekts. Das Netzwerk hat mithilfe von vielen MIDI-Beispielen musikalische Konzepte erlernt und kann so auf gespielte Noten reagieren und dazu selbst passende Noten spielen. Auf der Website des Projektes ist es möglich, auf einem virtuellen Piano Melodien zu spielen, während A.I. Duet mit leichtem Versatz dazu passende Noten generiert und ausgibt[2].

3.2.4. Music Transformer

Music Transformer [17] ist ein Projekt, was ebenfalls in Zusammenarbeit mit dem Google-Magenta-Team entstanden ist. Ziel ist es, Musik mit besonders guter Langzeitkohärenz zu erschaffen. Hierzu wird ein neuronales Netz basierend auf dem Transformer-Model [25] verwendet.
Es ist dem Netz möglich, ein gespieltes Motiv mehrfach innerhalb eines Stückes, mit leichten Variationen, zu wiederholen und einen Spannungsaufbau zu erzeugen. Durch die Verwendung von „self attention“ [22] ist es auch möglich, ein gegebenes Motiv über die Länge der Trainingsbeispiele hinaus weiter zu spielen. „Performance RNN“ [19], ein auf LSTM basierendes Model, war im Vergleich dazu nicht in der Lage, kohärente Musik zu einem vorgegebenen Motiv zu generieren.

[2] Offizielle Website: https://experiments.withgoogle.com/ai/ai-duet/

4. Erzeugung und Aufbereitung der Daten

4.1. MIDI-Dateien

Die Grundlage dieser Ausarbeitung sind MIDI-Dateien. MIDI steht für Musical Instrument Digital Interface und ist ein standardisiertes Format, welches für den Austausch musikalischer Steuerinformationen zwischen elektronischen Instrumenten entwickelt wurde, um eine einfache Kommunikation zwischen Audiogeräten zu ermöglichen.
Ein Vorteil ist, dass das Format das Editieren der Noten und Songstrukturen einfach macht, da keine Audiodaten an sich, sondern Ereignisse gespeichert werden. Diese werden dann vom jeweiligen Programm oder Instrument wieder in Sounds umgewandelt. Durch diese Vorgehensweise sind die Dateien wesentlich kleiner als andere Audiodateien.
Eine MIDI-Datei besitzt bis zu 16 Kanäle auf denen Ereignisse gesetzt sind, die z.B. für Noten stehen. Jeder Kanal steht für ein Instrument. Der Kanal mit der Nummer 10 ist dabei für das Schlagzeug reserviert. Die Ereignisse, die auf diesem Kanal gelesen werden, werden nicht auf Noten, sondern Schlagzeugteile übersetzt. Die Übersetzungstabellen für Noten (siehe A.1) und Schlagzeugspuren (siehe A.2) sind im Anhang zu finden.
Zusätzlich besitzt jedes Ereignis Informationen über seinen Start- und Endzeitpunkt. Die übergeordnete Struktur der MIDI-Datei kann unterschiedlich aufgebaut sein. Es gibt drei verschiedene Formate (0-2), die sich darin unterscheiden, wie viele verschiedene Tracks innerhalb der Datei abgelegt sind und wo Informationen, wie z.B. Tempoänderungen, zu finden sind.[15]

4.2. Wahl der Daten

Für diese Arbeit wird die Schlagzeugspur verwendet, da sie gegenüber den anderen Instrumenten und Spuren gewisse Vereinfachungen mit sich bringt. Beispielsweise muss hier nicht auf unterschiedliche Tonarten geachtet werden. Zusätzlich werden in der Regel bei einem Schlagzeug keine Noten gehalten, sondern nur angespielt. Die Notenlänge kann deshalb ebenfalls außer Acht gelassen werden.
Einen Trainingsdatensatz aus echten Musikstücken zusammenzustellen ist oft nicht praktikabel, da viele MIDI-Dateien nicht ausreichend professionell erstellt sind und folgende Probleme

aufweisen: Sie liegen in unterschiedlichen Formaten vor, haben keine Schlagzeugspur auf Kanal 10 oder sind komplett fehlerhaft und somit nicht lesbar. Aus diesem Grund besteht der Datensatz für die Experimente dieser Arbeit aus selbst generierten Dateien. Somit ist es einfach, die Dateien nach Bedarf anzupassen und die Komplexität schrittweise zu erhöhen. Ebenfalls kann so die Größe des Trainings- und Testdatensatzes beliebig vergrößert werden.

4.3. Erzeugung der Daten

Die MIDI-Dateien besitzen nur eine Schlagzeugspur und werden anhand von verschiedenen Parametern erstellt. Die gewählten Parameter für die Datensätze der einzelnen Experimente werden jeweils durch eine Komplexitätsgrafik dargestellt (Bsp. Abbildung 4.1). Es können 0-2 Soli gespielt werden, die im gewählten Solo-Intervall gespielt werden. Sind zwei Soli gewählt, werden diese jeweils abwechselnd gespielt.

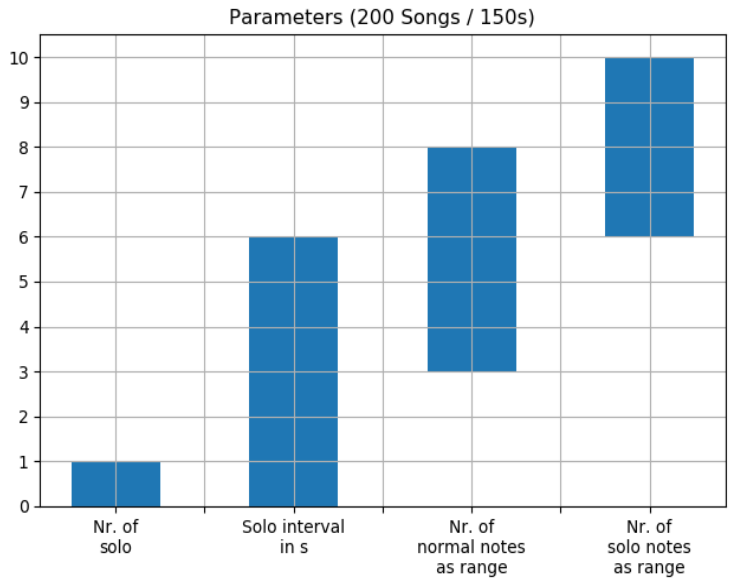


Abb. 4.1.: Beispiel für eine Komplexitätsgrafik eines Datensatzes mit 200 Songs mit einer Länge von jeweils 150 Sekunden

Die Noten der Datei wiederholen sich in Abschnitten, die jeweils einen halben Takt andauern. Ein Beispiel dafür ist zu sehen in Abbildung 4.2. Dort sind auf der X-Achse die Zeit und auf der Y-Achse die verschiedenen Tonhöhen bzw. Schlagzeugteile sichtbar.
Die Anzahl an Noten, die pro Abschnitt gespielt werden, variiert pro Datei im festgelegten

Bereich. Getrennt davon kann auch der Bereich, aus dem die Anzahl der Noten in den Soli bestimmt wird, eingestellt werden. Bei den gespielten Noten handelt es sich um $\frac{1}{16}$-Noten. Wenn mehr als drei Noten in einem Abschnitt gespielt werden, können auch mehrere Noten gleichzeitig gespielt werden.
Für diese Arbeit ist der Bereich der Noten auf die Events aus Abbildung A.2 beschränkt, deren Keys im Bereich [35, 61] liegen, um nur die wichtigsten Schlagzeugteile abzubilden.
Um im späteren Verlauf den Umgang mit fehlerbehafteten Dateien zu überprüfen, kann eine Fehlerrate in Prozent angegeben werden. Diese legt die Wahrscheinlichkeit fest, mit der Noten beim Erstellen der Datei nicht gesetzt werden.

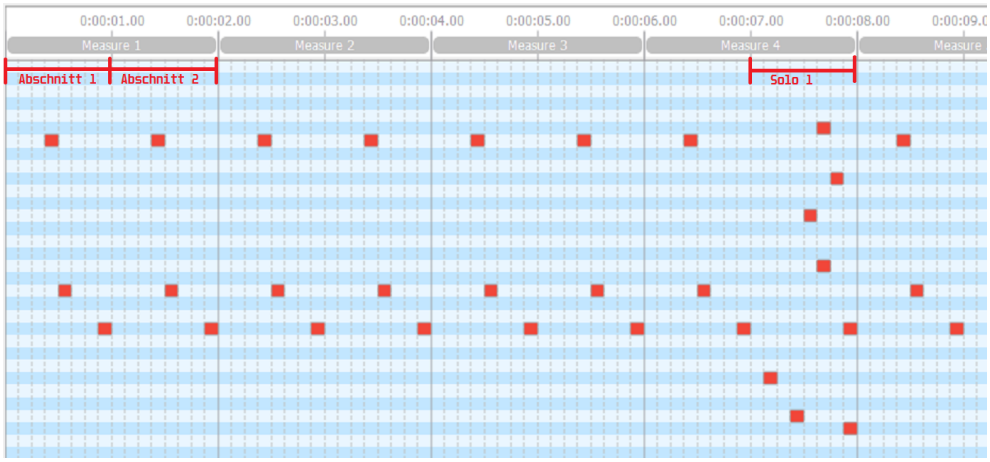


Abb. 4.2.: Beispiel für eine MIDI-Datei

4.4. Aufbereitung der Daten

In den Experimenten bekommt das Netz jeweils 20 Sekunden, um mindestens einen Durchlauf des zu lernenden Notenmusters zu erhalten. Da die generierten MIDI-Dateien nur $\frac{1}{16}$-Noten enthalten, entstehen so 320 mögliche Noten bzw. Zeitschritte pro Ausschnitt.
Die Dateien werden so Note für Note durchschritten um Input-/Output-Paare zu bilden. Als Output wird jeweils die 321. Note gesetzt. Eine Note wird durch einen Vektor der Größe 27 dargestellt, um die Schlagzeugteile, im zuvor eingegrenzten Bereich, via Multi-Hot-Encoding abzubilden.
Die Netzwerke erhalten somit einen Input mit 320 Timesteps und 27 Features.

4.5. Validierung der Netzwerke

Um die Funktion der fertig trainierten Netzwerke zu überprüfen, kann den Netzen eine MIDI-Datei als Input gegeben werden. Diese wird wie zuvor aufbereitet und ein zufälliger Teil dieser Datei mit der festgelegten Länge wird dem Netz als Input übergeben. Dieses Stück wird zusätzlich zur Verwendung im Netz als Original gespeichert.
Daraufhin generiert das Netz solange Noten, bis ein Stück der selben Länge, wie die Eingabesequenz, erstellt ist. Dabei füttert es sich immer wieder selber mit neuen Daten, indem es die generierte Note an den alten Input hängt und die vorderste abschneidet (siehe Abbildung 4.3).

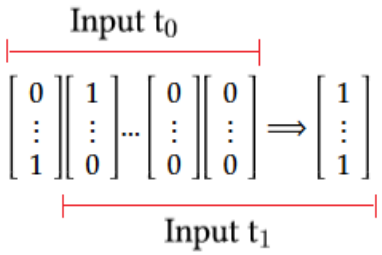


Abb. 4.3.: Input des Netzes im Laufe der Zeit

5. Experimente

5.1. Ablauf und Aufbau

Die Experimente werden in Python mithilfe der Open Source Deep-Learning-Bibliothek Keras[1] und Tensorflow[2] als Backend durchgeführt. Als Grafikkarte steht eine NVIDIA GeForce GTX 1080 Ti zur Verfügung.
Das erste Experiment dient dazu den Unterschied der Lernfähigkeit von periodischen Sequenzen bei normalen LSTM-Netzwerken und BLSTM-Netzwerken zu verdeutlichen. In den folgenden beiden Experimenten wird erst ein Solo in einem gewissen Abstand wiederholt und später zwei unterschiedliche Soli alternierend gespielt. Außerdem wird bei beiden Experimenten untersucht, inwieweit Fehler im Input bei der Prognose der Stücke kompensiert werden können. Im letzten Experiment bekommen die zuvor trainierten Netzwerke Daten als Input mit denen sie nicht trainiert wurden, um zu untersuchen, wie gut sie mit unbekannten Mustern umgehen können.
Der grundlegende Aufbau der Netzwerke ist für jedes Experiment identisch. Die genauen Parameter sind, wenn nicht im Text genannt, im Anhang zu finden (siehe: A.2). Der erste Layer stellt immer einen BLSTM-Layer dar. Als Parameter werden dem Layer für die Zeitschritte die Länge der Eingabesequenzen und als Merkmale die Anzahl der spielbaren Schlagzeugnoten übergeben. Die beiden Outputs der BLSTM-Layer werden konkateniert.
Da ein genaues bestimmen der Anzahl der Neuronen nicht möglich ist, werden diese durch Abschätzen und Testen bestimmt. Darauf folgt ein Dropout Layer, um dem Overfitting des Netzwerkes entgegenzuwirken. Als letztes wird mithilfe eines Dense Layer der Output des Netzes auf die Anzahl der spielbaren Noten verkleinert.
Da es sich bei diesen Experimenten um Multi-Label-Klassifikation handelt (es können mehrere Schlagzeugteile gleichzeitig gespielt werden), wird als Fehlerfunktion Binary Crossentropy verwendet. Als Optimierungsalgorithmus wird der „Adam“-Algorithmus verwendet, der derzeit als einer der Besten im Bereich Deep Learning gehandelt wird [18].
Um die Genauigkeit des Netzes zu überprüfen, gibt es bei jedem Experiment einen Validie-

[1] Offizielle Website: https://keras.io/
[2] Offizielle Website: https://www.tensorflow.org/

rungsdatensatz. Mit diesem wird während des Trainings, nach jedem Gewichtsupdate, mithilfe der Binary Accuracy geprüft, wie korrekt der Output des Netzes ist. Da es sich dabei um einen Vektor der Größe 27 handelt, wird so standardmäßig schon eine Trefferquote von ca. 96,3% erreicht, wenn das Ergebnis um eine Note daneben liegt. Hierbei werden bereits 26 der 27 möglichen Noten korrekt gespielt. Damit ein Netzwerk im folgenden nennenswerte Ergebnisse mit unbekannten Daten erzeugt, muss es somit eine Validation Accuracy von deutlich über 96,3% haben.
Jegliche hier dargestellte Ergebnisse werden durch wiederholte Ausführungen bestätigt.

5.2. Experiment 1: Kein Solo

Für dieses Experiment wird ein Trainingsdatensatz der Größe 200 verwendet, der mit den Parametern aus Abbildung 5.1 erstellt wird. Es wird konstant ein Notenmuster wiederholt (siehe: 5.2). Für die Validierung wird ein kleinerer Datensatz mit 20 Songs und den gleichen Parametern erstellt.

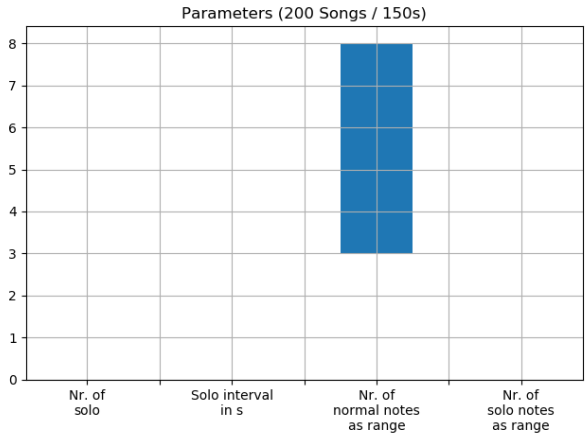


Abb. 5.1.: Parameter für den Datensatz von Experiment 1

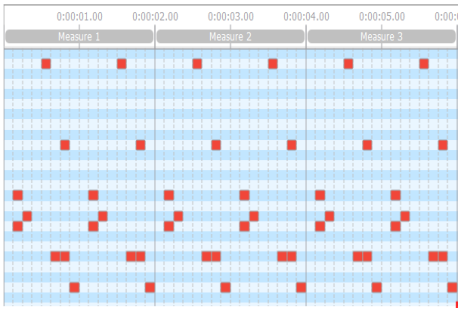


Abb. 5.2.: 6-sekündiger Auszug aus einer MIDI-Datei für Experiment 1

5.2.1. Training

In diesem Experiment wird zusätzlich zum BLSTM-Netzwerk (siehe Quelltext A.1) auch ein LSTM-Netzwerk (siehe Quelltext A.2) trainiert. Beide Netze erhalten die selben Datensätze und werden mit den selben bereits erwähnten Parametern, mit Ausnahme der Anzahl der

Neuronen, trainiert. Beiden Varianten gelingt es, eine Validation Accuracy von über 99.8% zu erreichen (siehe Abbildung 5.3 und 5.4).

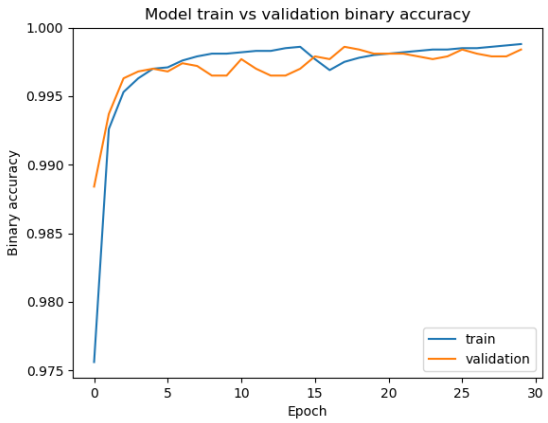


Abb. 5.3.: Validation Accuracy des BLSTM für Experiment 1

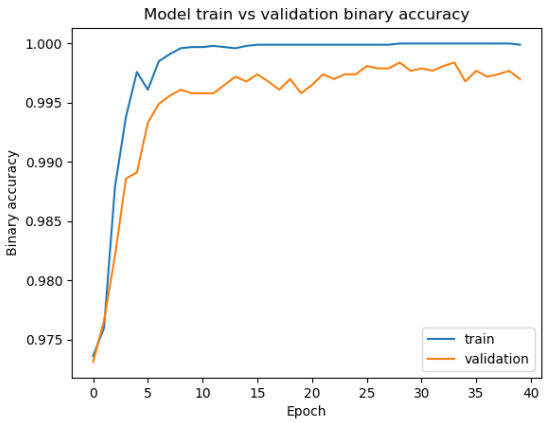


Abb. 5.4.: Validation Accuracy des LSTM für Experiment 1

5.2.2. Prognose

Um die Korrektheit der Ausgabe der Netze zu prüfen, wird beiden Netzwerken die selbe, ihnen nicht bekannte, Datei als Input gegeben. Diese Datei besitzt die selbe Komplexität wie die Test- und Trainingsdaten. Sowohl mit dem normalen als auch mit dem bidirektionalen LSTM-Netzwerk ist es möglich, dem Netz unbekannte Daten der gleichen Komplexität ohne Fehler weiterzuspielen (siehe Abbildungen 5.5, 5.6 und 5.7).

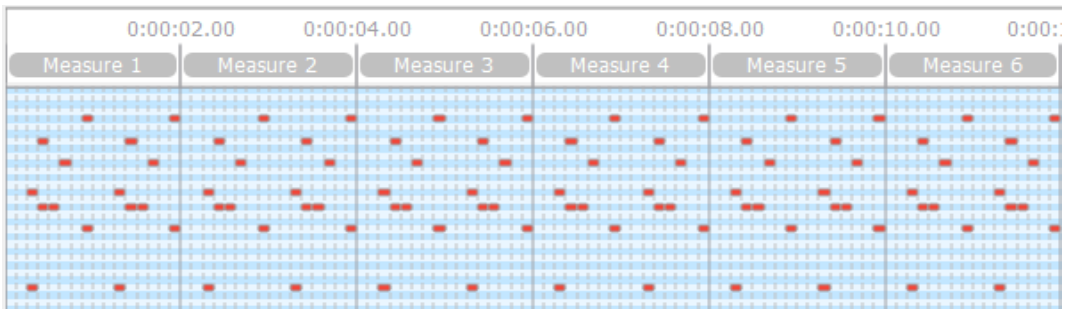


Abb. 5.5.: 12-sekündiger Auszug der Originaldatei

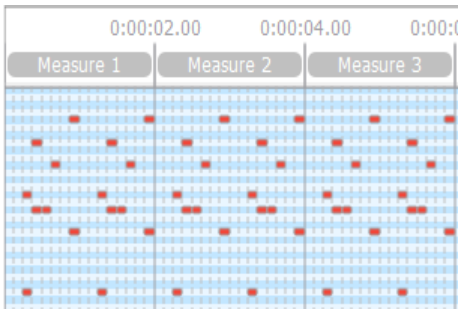


Abb. 5.6.: 6-sekündiger Auszug der vom BLSTM generierten Datei

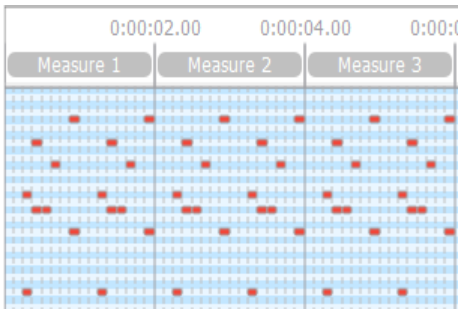


Abb. 5.7.: 6-sekündiger Auszug der vom LSTM generierten Datei

5.2.3. Ergebnis

Das BLSTM-Netzwerk ist im Vergleich zum normalen LSTM-Netzwerk deutlich schneller im Lernen des Musters. Für diese Aufgabe erreicht das BLSTM-Netz bereits nach der dritten Epoche eine Validation Accuracy von über 99.5%. Das LSTM-Netz benötigt für ähnliche Werte acht Epochen.
Ein weiterer Punkt ist, dass mehr Neuronen benötigt werden und zusätzlich dazu auch Dropout aktiv sein muss, um ein Overfitting zu verhindern. In zusätzlichen Experimenten zeigt sich, dass das BLSTM-Netzwerk nicht nur schneller lernt, sondern diese einfachen Muster auch mit einem deutlich kleinerem Datensatz und ohne Dropout-Layer lernen kann. Da die Beschaffung ausreichender Trainingsdaten in der Realität nicht zu unterschätzen ist, ist dies ein großer Vorteil. Auch, wenn beide Netze das Muster fehlerfrei wiederholen können, ist das BLSTM-Netzwerk somit besser geeignet und wird in den folgenden Experimenten verwendet.

5.3. Experiment 2: Ein Solo

Das zweite Experiment wird um eine zusätzliche Komponente erweitert. Der Datensatz besteht aus 100 MIDI-Dateien, die jeweils alle vier Sekunden ein Solo beinhalten. Später wird der Abstand zwischen den Soli auf sechs Sekunden vergrößert.
Ein Beispiel für eine solche Datei mit einem Solo-Abstand von vier Sekunden ist in Abbildung 5.8 zu sehen. Außerdem werden die trainierten Netze mit fehlerbehafteten Dateien getestet, um zu überprüfen, ob und wie gut sie Fehler kompensieren können.

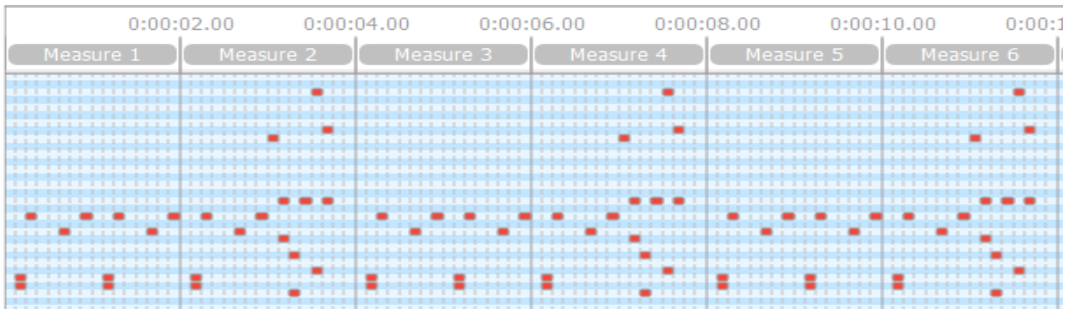


Abb. 5.8.: 12-sekündiger Auszug aus einer Trainingsdatei mit vier Sekunden Abständen zwischen den Soli für Experiment 2

5.3.1. Training mit vier Sekunden Solo-Abstand

Der Trainingsdatensatz für dieses Experiment besteht aus 100 Dateien, die mit den Parametern aus Abbildung 5.9 erstellt wurden. Als Validierungsdatensatz kommen 20 Dateien der selben Komplexität zum Einsatz. Das gewählte BLSTM-Netzwerk (siehe Quelltext A.3) verwendet die selben Parameter wie in Experiment 1. Dem Netzwerk gelingt es auch hier nach bereits 12 Epochen eine Validation Accuracy von über 99,8% zu erreichen (siehe Abbildung 5.10). Das Netzwerk aus diesem Experiment wird im folgendem Netzwerk 2a genannt.

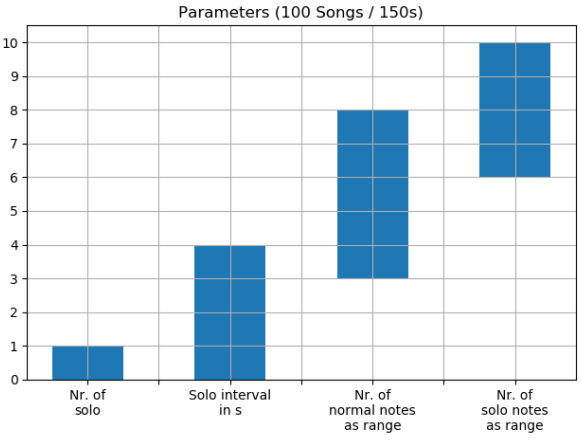


Abb. 5.9.: Parameter für den Datensatz von Experiment 2 mit vier Sekunden Solo-Abstand

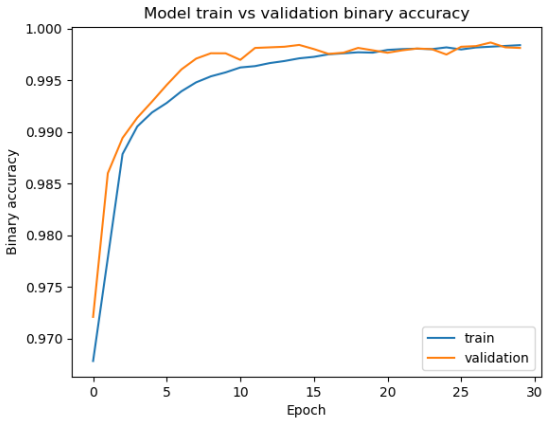


Abb. 5.10.: Validation Accuracy des BLSTM für Experiment 2 mit vier Sekunden Solo-Abstand

5.3.2. Training mit sechs Sekunden Solo-Abstand

Da dieses Experiment ein leistungsfähigeres Netzwerk erfordert, wird sowohl der Trainings- als auch der Validierungsdatensatz erhöht (250 bzw. 75 Dateien), um ein Overfitting zu erschweren. Diese Dateien besitzen die Parameter aus Abbildung 5.11. Dem Netzwerk aus dem ersten Teil dieses Experiments ist es nicht mehr möglich die Muster dieser Komplexität zu erlernen. Einem Netzwerk mit mehr Neuronen (siehe Quelltext A.4) gelingt es mit einem Solo-Abstand von sechs Sekunden erst nach 29 Epochen eine Validation Accuracy von über 99,5% zu erreichen (siehe Abbildung 5.12). Dieses Netzwerk ist im folgenden als Netzwerk 2b geführt.

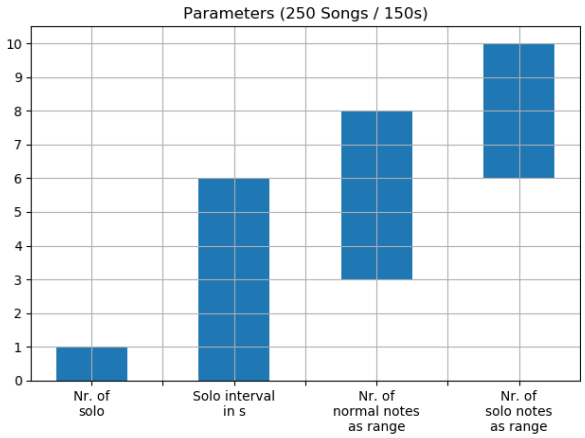


Abb. 5.11.: Parameter für den Datensatz von Experiment 2 mit sechs Sekunden Solo-Abstand

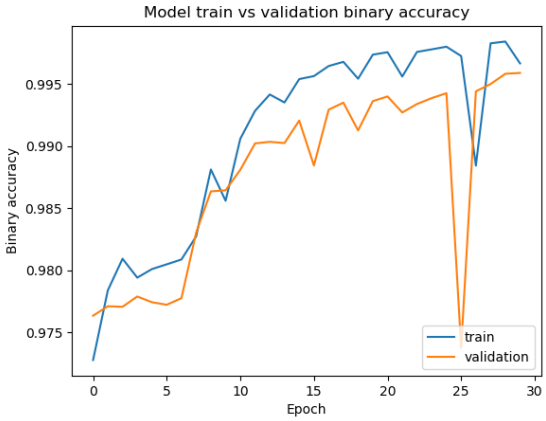


Abb. 5.12.: Validation Accuracy des BLSTM für Experiment 2 mit sechs Sekunden Solo-Abstand

5.3.3. Prognose

Bei einer stichprobenartigen Überprüfung von Netzwerk 2a ist zu sehen, dass die Stücke grundlegend korrekt weitergespielt werden. Die Takte zwischen den Soli können komplett ohne Fehler reproduziert werden. Innerhalb der Soli sind jedoch vereinzelt fehlende Noten zu erkennen.

Im Vergleich dazu fällt es Netzwerk 2b, das einen größeren Abstand zwischen den Soli zu erlernen hat, schwerer eine korrekte Weiterführung der Noten zu produzieren. Zwar können auch hier die Takte zwischen den Soli ohne Fehler gespielt werden, es konnte jedoch keine der getesteten Dateien fehlerfrei fortgeführt werden. Kein Solo kann ohne mindestens eine

fehlende Note gespielt werden. Zusätzlich werden vereinzelt Noten zwar zur richtigen Zeit gespielt, aber auf einer falschen (im Original nicht bespielten) Tonhöhe.

5.3.4. Prognose mit fehlerbehafteten Dateien

Im Folgenden soll nun untersucht werden, ob die Netzwerke in der Lage sind, fehlende Noten innerhalb des Inputs zu kompensieren. Hierzu wird den Netzwerken eine Datei in verschiedenen Varianten zugeführt. Es wird jeweils getestet wie die Datei weitergespielt werden kann, wenn 5% (10%, 15%, 20% und 25%) der in der Originaldatei gesetzten Noten fehlen. Für beide Netzwerke werden Dateien gewählt, die in ihrer Originalform möglichst fehlerfrei generiert werden können, um den grundlegenden Fehler gering zu halten.
Dem Netzwerk 2a gelingt es, die Originaldatei ohne Fehler fortzuführen. Sobald der 20-sekündige Abschnitt, der dem Netzwerk übergeben wird, fehlende Noten beinhaltet, werden diese jedoch konstant in dieser fehlerhaften Form reproduziert (siehe Abbildung 5.13 und 5.14). Dieses Verhalten zeigt sich unabhängig von dem gewählten Prozentsatz an fehlenden Noten.

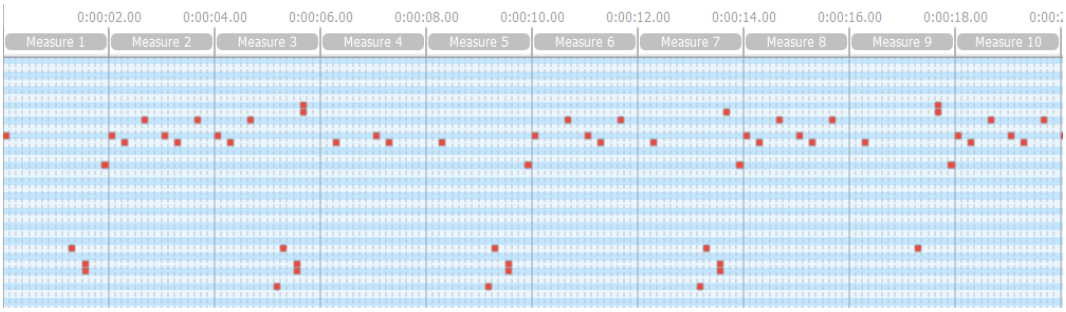


Abb. 5.13.: 20-sekündiger Auszug aus einer Originaldatei mit 25% fehlenden Noten

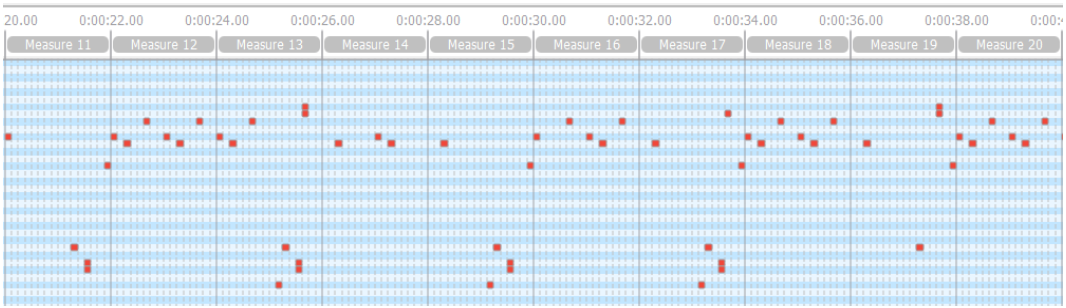


Abb. 5.14.: 20-sekündiger generierter Auszug mit der Datei aus 5.13 als Input

Ein ähnliches Verhalten zeichnet sich bei Netzwerk 2b ab. Hier kommt jedoch hinzu, dass bereits bei fehlerfreiem Input Noten im Output nicht gesetzt sind. Bei Fehlern im Input wird somit nicht nur die fehlerhafte Sequenz fortgeführt, sondern zusätzlich weitere Noten nicht gesetzt. Mit zunehmendem Prozentsatz an fehlenden Noten im Input ist dieses Phänomen verstärkt zu beobachten.

5.3.5. Ergebnis

In diesem Experiment zeigt sich, dass es dem Netzwerk 2a, mit dem selben Aufbau wie aus Experiment 1, möglich war mit einem Solo umzugehen. Sobald jedoch der Abstand zwischen den Soli vergrößert wird, muss sowohl das Netzwerk als auch die Größe des Trainingsdatensatzes angepasst werden. Netzwerk 2b kann selbst mit diesen Anpassungen keine vollständig fehlerfreie Prognose durchführen. Auffällig ist zudem, dass das Netzwerk oft nicht nur eine Note, sondern eine komplette Tonhöhe in der Prognose nicht weiter berücksichtigt.
Beiden Netzwerken ist es nicht möglich fehlende Noten im Input zu kompensieren. Sie führen die fehlerhafte Sequenzen in dieser Form fort. Das Netzwerk 2b entfernt sich bei steigendem Prozentsatz an fehlenden Noten sogar immer weiter von der korrekten Prognose.

5.4. Experiment 3: Zwei alternierende Soli

Für dieses Experiment wird der Aufbau der Daten ein weiteres Mal verändert. Es werden nicht mehr nur ein Solo, sondern zwei verschiedene im Wechsel gespielt. Somit ist die Länge des zu lernenden Musters doppelt so lange wie bei Experiment 2, da beide Soli erfasst werden müssen. Ein Auszug einer solchen Datei mit vier Sekunden Abstand zwischen den Soli ist in Abbildung 5.15 zu sehen.

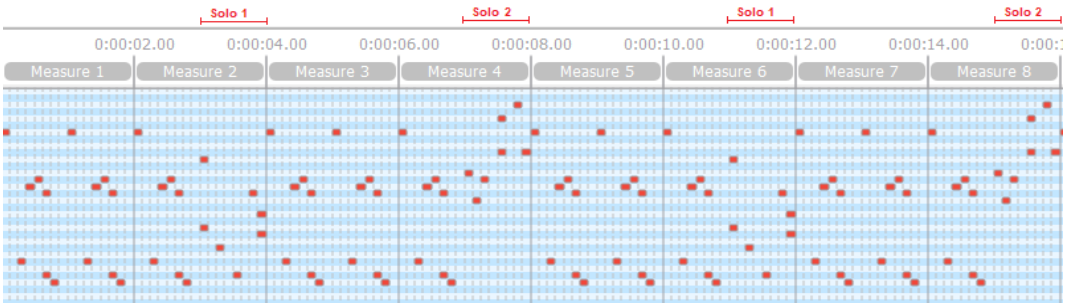


Abb. 5.15.: 16-sekündiger Auszug aus einer Trainingsdatei mit vier Sekunden Abständen zwischen den Soli für Experiment 3

5.4.1. Training

Wie bereits im vorherigen Experiment wird ein weiteres Mal sowohl der Trainigs- als auch der Validierungsdatensatz vergrößert (350 bzw. 100 Dateien). Für diese Datensätze gelten die Parameter aus Abbildung 5.16. Dem Netzwerk (siehe Quelltext A.5) gelingt es mit unterschiedlichen Konfigurationen nicht reproduzierbar, über eine Validation Accuracy von 98,5% hinauszukommen (siehe Abbildung 5.17).

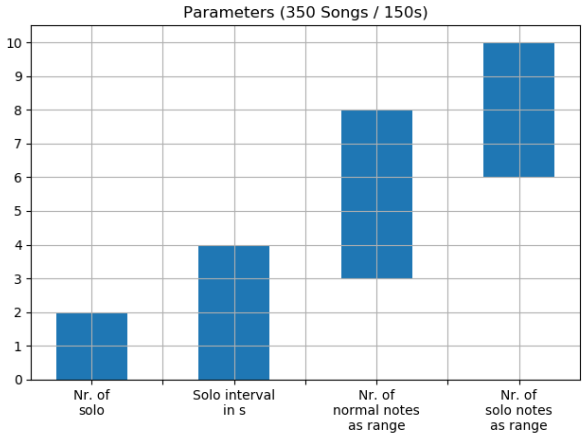


Abb. 5.16.: Parameter für den Datensatz von Experiment 3 mit vier Sekunden Solo-Abstand

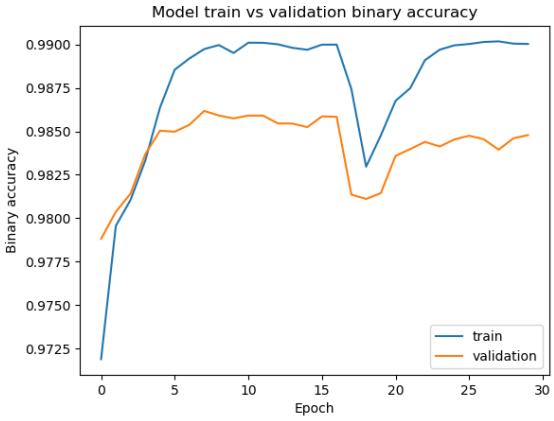


Abb. 5.17.: Validation Accuracy des BLSTM für Experiment 3

5.4.2. Prognose

Sowohl die Accuracy, als auch die Validation Accuracy sind hier deutlich geringer als in den vorangegangenen Experimenten. Dem Netzwerk ist es so nicht möglich, die übergebene Sequenz korrekt fortzuführen. Auch wenn zu Beginn der Prognose die Takte zwischen den Soli fast korrekt wiedergegeben werden, nimmt im Verlauf der Zeit die Anzahl der gespielten Noten konstant ab (siehe 5.18). Wenn über einen längeren Zeitraum generiert wird, gehen die gespielten Noten sogar gegen Null. Eine Vergrößerung des Abstands zwischen den Soli oder eine Analyse des Verhalten mit fehlerhaftem Input wird deshalb nicht vorgenommen.

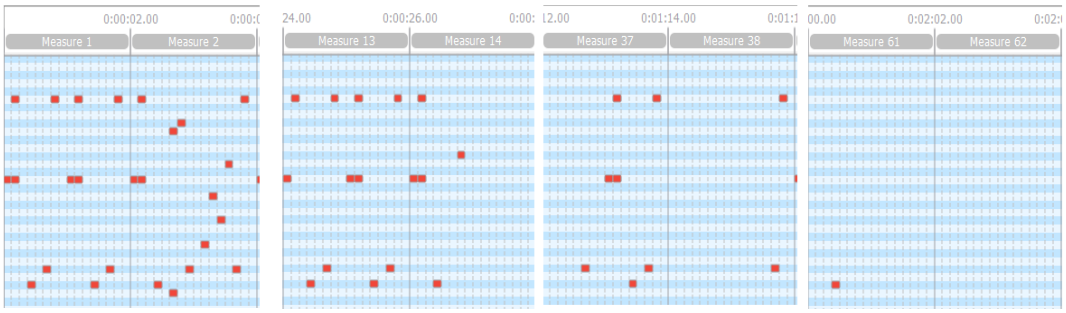


Abb. 5.18.: Auszüge aus eine generierten Datei aus Experiment 3

5.4.3. Ergebnis

Dieses Experiment zeigt, dass die Umstellung auf zwei unterschiedliche Soli deutlich höhere Anforderungen an das Netzwerk stellt. Es ist dem Netzwerk nicht möglich, reproduzierbar die Sequenzen zu erlernen. Der Verlauf des Graphen in Abbildung 5.17 ist ein Hinweis darauf, dass das Netzwerk möglicherweise nicht potent genug ist, um mit Sequenzen dieser Komplexität zu arbeiten. Ein Netzwerk mit mehr Neuronen oder Layern ist hier erforderlich. Dies wird aus Gründen der Zeit nicht weiter überprüft, da das Training pro Epoche, mit dem getesteten Aufbau und Daten, bereits über 30 Minuten dauert.

5.5. Experiment 4: Prognose mit unbekannten Daten

Im abschließenden Experiment soll nun analysiert werden, ob die trainierten Netzwerke auch mit Sequenzen umgehen können, mit denen sie nicht trainiert wurden. Hierfür werden die trainierten Netzwerken aus den vorherigen Experimenten verwendet. Diesen werden die Daten aus den jeweils anderen Experimente als Input übergeben. Eine Übersicht der Ergebnisse ist in Tabelle 5.1 zu sehen. Die Prognose ist mit „Ja“ gekennzeichnet, wenn eine fehlerfreie Prognose möglich ist. „m. E.“ steht für „mit Einschränkungen“ und bedeutet hier, dass eine Prognose zwar ohne fehlende Noten durchgeführt, aber die korrekte Struktur nicht eingehalten wird oder die Struktur korrekt ist, jedoch Noten fehlen. Felder, die mit „Nein“ beschriftet sind, stehen für eine Prognose, bei der weder die Noten korrekt ausgegeben werden, noch die Struktur eingehalten wird.

Input \ Netzwerk	Experiment 1 (BLSTM)	Experiment 2 (4 Sek.)	Experiment 2 (6 Sek.)	Experiment 3
Experiment 1	Ja	Ja	Ja	Ja
Experiment 2 (4 Sek.)	Ja	Ja	Nein	Nein
Experiment 2 (6 Sek.)	m. E.	m. E.	m. E.	Nein
Experiment 3	Nein	Nein	Nein	Nein

Tab. 5.1.: Übersicht über korrekte Prognosen der verschiedenen Netzwerke mit den Daten aus allen Experimenten

5.5.1. Prognose

Das BLSTM-Netzwerk aus Experiment 1 ist in der Lage, die Prognose von Sequenzen mit einem wiederkehrenden Solo aus Experiment 2 korrekt auszuführen.
Sobald jedoch der Abstand zwischen diesen Soli auf sechs Sekunden erhöht wird, wird in der Prognose nicht mehr die erwartete Anzahl an Takten zwischen den Soli gespielt. Das Netzwerk produziert zwar noch alle Noten korrekt, spielt diese jedoch zu falschen Zeitpunkten.
Die Dateien aus Experiment 3, in denen zwei Soli alternierend gespielt werden, können vom Netzwerk nicht mehr ohne den Verlust von Noten fortgeführt werden. Die Takte zwischen den Soli werden zwar vollständig prognostiziert, die Soli hingegen weisen Fehler auf. Ebenfalls werden diese nicht mehr im Wechsel gespielt, sondern teilweise doppelt wiederholt.

Netzwerk 2a aus Experiment 2 gelingt es, die Dateien aus Experiment 1 komplett fehlerfrei zu generieren.
Für die Daten mit größerem Abstand zwischen dem Solo zeigt sich ein ähnliches Bild, wie bereits bei dem Netzwerk aus Experiment 1. Die Noten werden korrekt gespielt, aber die Struktur des Musters mit sechs Sekunden zwischen den Soli kann nicht korrekt beibehalten werden. Teilweise wird das Solo zwei mal in Folge gespielt.
Auch für die Daten aus Experiment 3 ist das Verhalten identisch zu dem Netzwerk aus Experiment 1. Die Takte zwischen den Soli werden korrekt ausgegeben. Die Soli werden jedoch nicht mehr abwechselnd gespielt und innerhalb der Soli sind fehlende Noten zu beobachten.

Für das zweite Netzwerk, Netzwerk 2b, aus Experiment 2 gilt ebenfalls, dass die Daten aus Experiment 1 vollständig prognostiziert werden.
Sobald ein Solo mit einem Abstand von vier Sekunden gespielt werden soll, kommt es zu

Fehlern. Einzig das Solo wird durchgängig mit den korrekten Noten wiedergegeben. Dies geschieht jedoch nicht im richtigen Abstand. Die Takte zwischen den Soli variieren in ihrer Anzahl und zusätzlich fehlen Noten.
Netzwerk 2b kann die Dateien aus Experiment 3 nicht prognostizieren. Fehlende Noten treten hier nicht nur innerhalb der zwei Soli, sondern auch in den Takten dazwischen auf. Ebenfalls variieren die Takte zwischen den Soli und diese werden nicht korrekt im Wechsel gespielt.

Das Netzwerk aus Experiment 3 ist ebenfalls in der Lage die Sequenzen ohne Solo fehlerfrei zu prognostizieren.
Für die Sequenzen aus den anderen Experimenten zeigt sich ein ähnliches Bild wie bereits bei 5.18 aus Experiment 3. Die Muster werden nicht korrekt fortgeführt und im Laufe der Zeit spielt das Netzwerk immer weniger Noten, bis kein Muster mehr erkennbar ist.

5.5.2. Ergebnis

In diesem Experiment ist zu erkennen, dass jedes trainierte Netzwerk in der Lage ist eine einfache Sequenz ohne Solo zu prognostizieren. Zusätzlich ist es keinem der Netzwerke möglich Dateien mit zwei alternierenden Soli, weder ohne fehlende Noten, noch in der korrekten Struktur, fortzuführen.
Außerdem zeigt sich, dass das Netzwerk aus Experiment 1 überraschend gute Prognosen erzeugt. Das Netzwerk besitzt die gleichen Parameter wie Netzwerk 2a und ist in der Lage qualitativ identische Prognosen abzugeben. Als Trainingsgrundlage dienen hier jedoch deutlich weniger komplexe Dateien. Es zeigt sich somit, dass das Netzwerk nicht anhand eines kleinen Ausschnitts gelernt hat, dass in Experiment 1 immer die gleichen Noten wiederholt werden, sondern den kompletten Input kopiert.
Der vom Netzwerk gelernte, zu wiederholende Ausschnitt muss groß genug sein, das Solo aus Experiment 2 zu beinhalten, um dieses korrekt zu prognostizieren. Sobald der Abstand zwischen den Soli auf sechs Sekunden erhöht wird, schafft es das Netzwerk jedoch nicht mehr dieses Muster korrekt fortzuführen. Dies lässt die Vermutung zu, dass das Netzwerk nur die korrekte Prognose durchführen kann, wenn das Muster genau in dem übergebenen Abschnitt (in diesem Fall 20 Sekunden) aufgeht. Dies bestätigt die Annahme, das das Netzwerk lernt diesen komplett zu reproduzieren. Netzwerk 2a zeigt das gleiche Verhalten. Ab einer gewissen Komplexität ist es beiden Netzwerken nicht mehr möglich diese Kopie konstant zu erzeugen und die Dateien aus Experiment 3 können so nicht ohne Fehler generiert werden.
Das Netzwerk 2b und das Netzwerk aus Experiment 3 sind bis auf die Dateien ohne Solo nicht in der Lage ihnen unbekannte Muster in den Sequenzen zu prognostizieren.

6. Auswertung

Der Vergleich der beiden Netzwerke in Experiment 1 hat deutlich gezeigt, welche Vorteile ein BLSTM-Netzwerk für die Bewältigung dieser Aufgabe bietet. Nicht nur ein schnelleres Erlernen der Struktur der Dateien wurde erreicht, es konnten auch mit weniger Trainingsdaten und einem kleineren Netzwerk korrekte Prognosen getroffen werden. Durch die Anzahl der Neuronen, die notwendig ist, um mit einem LSTM-Netzwerk zu arbeiten, neigt dieses mit wenigen Trainingsdaten zum Overfitting. Bei dem BLSTM-Netzwerk wird die Aufgabe auf zwei kleinere LSTM-Layer aufgeteilt, die einzeln nicht in der Lage wären die korrekte Prognose zu treffen oder die Daten auswendig zu lernen. Zusammen haben sie jedoch ausreichend Informationen, um das Problem zu lösen. Sie eignen sich damit wesentlich besser, da das Training mit größeren Netzwerken mehr Zeit und Rechenleistung erfordert. Zusätzlich ist es bei Problemen, bei denen man sich nicht selber neue Daten nach belieben generieren kann, ein großer Vorteil, wenn weniger Daten für einen Erfolg notwendig sind.
Eine weitere Erkenntnis aus den Experimenten 1 und 2 ist, dass die Netzwerke nicht unbedingt den erwarteten Weg nehmen, um Prognosen für die Sequenzen zu treffen. Die Netzwerke haben sich nicht etwa einen kleinen Teil der Sequenzen gemerkt oder die wirkliche Struktur an sich, sondern erkennen die gesamten übergebenen 20 Sekunden als ein Muster. Dieses wird dann kontinuierlich fortgeführt.
Durch dieses Verhalten können sie jedoch keine Fehler in den Inputsequenzen kompensieren. Diese werden eins zu eins übernommen und konstant reproduziert.
Dadurch, dass das komplette Muster nur kopiert wird, können sie selbst mit den komplexen Daten aus Experiment 3 überraschend gut umgehen. Im Vergleich zu dem, eigens für dieses Problem trainierten, Netzwerk aus Experiment 3 werden sogar mehr Noten in den Takten zwischen den Soli korrekt gesetzt. Dieses Netzwerk produziert dafür jedoch die richtige Struktur der alternierenden Soli. Es ist also davon auszugehen, dass hier tatsächlich mehr als nur reines Kopieren der gesamten Sequenz erlernt wurde.
Interessant war außerdem zu sehen, dass selbst das Netzwerk mit der schlechtesten Validation Accuracy (Experiment 3) in der Lage war die einfache Sequenzen ohne Solo vollständig fehlerfrei zu generieren. Ein Grundverständnis für sich wiederholende Sequenzen ist somit auch hier schon gegeben. Sobald es zu fehlenden Noten in der Prognose kam, verfiel das Netz in

eine Spirale des Verfalls der Sequenz und spielte immer weniger Noten.
Wenn es innerhalb der trainierten Netzwerke zu nicht gesetzten Noten in der Prognose kam, war oft zu beobachten, dass die gesamte Tonhöhe nicht bespielt wurde. Das Fehlen der Noten auf dieser Tonhöhe zog sich auch durch die Prognose anderer Dateien mit diesem Netzwerk. Eine mögliche Ursache dafür ist, dass diese Tonhöhe innerhalb der Trainingsdaten überhaupt nicht oder nur selten vorkam und das Netzwerk diese deshalb für irrelevant hält.

7. Fazit

Im Verlauf der Arbeit wurde deutlich, dass BLSTM-Netzwerke definitiv für die Prognose quasiperiodische Sequenzen geeignet sind. Die Experimente haben jedoch gezeigt, dass die richtige Wahl der Parameter für die Netzwerke entscheidend sind und die erforderliche Größe der Netzwerke und Datensätze mit steigender Komplexität drastisch ansteigt. Auch die Länge der Sequenzen, die den Netzwerken übergeben werden ist relevant für den Erfolg der Prognosen, da die zu erlernenden Muster enthalten sein müssen.
Zusätzlich zeigte sich, dass eine korrekte Prognose der Sequenz nicht zwangsläufig bedeutet, dass die Sequenzen in ihrer Form gelernt wurden. Dadurch, dass die Netzwerke die Sequenzen zum Großteil nur kopiert haben, war es nicht möglich, Fehler im Input durch mögliches Wissen zu kompensieren. Die übergebenen Fehler wurden so konstant weiter prognostiziert.
Es konnte keine Konfiguration für ein Netzwerk gefunden werden, dass in annehmbarer Zeit reproduzierbar Sequenzen mit zwei alternierenden Soli prognostizieren kann. Das trainierte Netzwerk zeigte jedoch ein grundlegendes Verständnis der Sequenzen und bietet eine gute Ausgangsposition für weitere Experimente.

7.1. Ausblick

Für die Fortführung dieser Arbeit wäre es denkbar weiter zu untersuchen, wie sich die Länge der Inputsequenzen auf das Lernverhalten der Netzwerke auswirkt. In Experiment 1 und 2 werden beispielsweise nicht die vollen 20 Sekunden benötigt, um die wichtigen Informationen an das Netzwerk zu übergeben. Auch das Erlernen der alternierenden Soli ist durch eine Verfeinerung der Parameter und ein längeres Training erwartbar. Falls für dieses Netzwerk tatsächlich ein Verständnis der Struktur der Sequenzen erreicht werden sollte (und nicht nur reines Kopieren), ist auch eine erneute Untersuchung des Verhaltens im Falle von fehlerhaftem Input von Interesse.
Eine Gegenüberstellung zu dem in Kapitel 3 erwähnten Transformer Model und Attention wäre zudem möglich. Somit kann verglichen werden, inwieweit die genannten Vorteile dieser Techniken bei der Prognose quasiperiodischer Sequenzen zum Tragen kommen und ob diese möglicherweise besser geeignet sind.

Literatur

[1] Yotam Mann, *Ai duet*, https://github.com/googlecreativelab/aiexperiments-ai-duet, [Online; zuletzt besucht am 06.06.2019].

[2] Amir Efrati, *Apple's machines can learn too*, https://www.theinformation.com/articles/apples-machines-can-learn-too?, [Online; zuletzt besucht am 06.06.2019].

[3] Adam Kosiorek, *Attention in neural networks and how to use it*, http://akosiorek.github.io/ml/2017/10/14/visual-attention.html, [Online; zuletzt besucht am 06.06.2019].

[4] Feynman T Liang, Mark Gotham, Matthew Johnson und Jamie Shotton, *Automatic stylistic composition of bach chorales with deep lstm*, https://ismir2017.smcnus.org/wp-content/uploads/2017/10/156_Paper.pdf. [Online; zuletzt besucht am 06.06.2019], 2017.

[5] Jason Brownlee, *Long short-term memory networks with python: Develop sequence prediction models with deep learning*, v1.4. Jason Brownlee, 2018. Adresse: https://machinelearningmastery.com/lstms-with-python/.

[6] Christopher Olah, *Understanding lstm networks*, http://colah.github.io/posts/2015-08-Understanding-LSTMs/, [Online; zuletzt besucht am 06.06.2019].

[7] Christopher Olah, *Neural networks, types, and functional programming*, http://colah.github.io/posts/2015-09-NN-Types-FP/, [Online; zuletzt besucht am 06.06.2019].

[8] Eugenio Culurciello, *Computation and memory bandwidth in deep neural networks*, https://medium.com/@culurciello/computation-and-memory-bandwidth-in-deep-neural-networks-16cbac63ebd5, [Online; zuletzt besucht am 06.06.2019].

[9] Daniil Korbut, *Machine learning translation and the google translate algorithm*, https://blog.statsbot.co/machine-learning-translation-96f0ed8f19e4, [Online; zuletzt besucht am 06.06.2019].

[10] Sepp Hochreiter und Jürgen Schmidhuber, "Long short-term memory", *Neural Computations*, Bd. 9: 1735-1780, 1997, http://www.bioinf.jku.at/publications/older/2604.pdf. [Online; zuletzt besucht am 06.06.2019].

[11] Eugenio Culurciello, *The fall of rnn / lstm*, https://towardsdatascience.com/the-fall-of-rnn-lstm-2d1594c74ce0, [Online; zuletzt besucht am 06.06.2019].

[12] Dan Ciresan, Alessandro Giusti, Luca M. Gambardella und Jürgen Schmidhuber, "Deep neural networks segment neuronal membranes in electron microscopy images", in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou und K. Q. Weinberger, Hrsg., [Online; zuletzt besucht am 06.06.2019], Curran Associates, Inc., 2012, S. 2843–2851. Adresse: http://papers.nips.cc/paper/4741-deep-neural-networks-segment-neuronal-membranes-in-electron-microscopy-images.pdf.

[13] Dan C. Cireşan, Alessandro Giusti, Luca M. Gambardella und Jürgen Schmidhuber, "Mitosis detection in breast cancer histology images with deep neural networks", in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2013*, Kensaku Mori, Ichiro Sakuma, Yoshinobu Sato, Christian Barillot und Nassir Navab, Hrsg., [Online; zuletzt besucht am 06.06.2019], Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, S. 411–418, ISBN: 978-3-642-40763-5. Adresse: http://papers.nips.cc/paper/4741-deep-neural-networks-segment-neuronal-membranes-in-electron-microscopy-images.pdf.

[14] Google Brain Team, *Magenta: Music and art generation with machine intelligence*, https://github.com/tensorflow/magenta, [Online; zuletzt besucht am 06.06.2019].

[15] McGill University, *Standard midi-file format spec. 1.1, updated*, http://www.music.mcgill.ca/~ich/classes/mumt306/StandardMIDIfileformat.html#BMA1_5, [Online; zuletzt besucht am 06.06.2019].

[16] Frank Brinkkemper, *Analyzing six deep learning tools for music generation*, http://www.asimovinstitute.org/analyzing-deep-learning-tools-music/, [Online; zuletzt besucht am 06.06.2019].

[17] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Curtis Hawthorne, Andrew M. Dai, Matthew D. Hoffman und Douglas Eck, "An improved relative self-attention mechanism for transformer with application to music generation", *CoRR*, Bd. abs/1809.04281, 2018. arXiv: 1809.04281. Adresse: http://arxiv.org/abs/1809.04281.

[18] Sebastian Ruder, "An overview of gradient descent optimization algorithms", *CoRR*, Bd. abs/1609.04747, 2016. arXiv: 1609.04747. Adresse: http://arxiv.org/abs/1609.04747.

[19] Ian Simon und Sageev Oore, *Performance rnn: Generating music with expressive timing and dynamics*, https://magenta.tensorflow.org/performance-rnn, [Online; zuletzt besucht am 06.06.2019].

[20] Hojjat Salehinejad, Julianne Baarbe, Sharan Sankar, Joseph Barfett, Errol Colak und Shahrokh Valaee, "Recent advances in recurrent neural networks", *CoRR*, Bd. abs/1801.01078, 2018. arXiv: 1801.01078. Adresse: http://arxiv.org/abs/1801.01078.

[21] Mike Schuster und Kuldip K. Paliwal, "Bidirectional recurrent neural networks", *IEEE Transactions On Signal Processing*, Bd. Vol. 45, No. 11, 1997, https://pdfs.semanticscholar.org/4b80/89bc9b49f84de43acc2eb8900035f7d492b2.pdf. [Online; zuletzt besucht am 06.06.2019].

[22] Peter Shaw, Jakob Uszkoreit und Ashish Vaswani, "Self-attention with relative position representations", *CoRR*, Bd. abs/1803.02155, 2018. arXiv: 1803.02155. Adresse: http://arxiv.org/abs/1803.02155.

[23] SuperDataScience Team, *Recurrent neural networks (rnn) - the vanishing gradient problem*, https://www.superdatascience.com/blogs/recurrent-neural-networks-rnn-the-vanishing-gradient-problem/, [Online; zuletzt besucht am 06.06.2019].

[24] Niklas Donges, *Recurrent neural networks and lstm*, https://towardsdatascience.com/recurrent-neural-networks-and-lstm-4b601dd822a5, [Online; zuletzt besucht am 06.06.2019].

[25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser und Illia Polosukhin, "Attention is all you need", *CoRR*, Bd. abs/1706.03762, 2017. arXiv: 1706.03762. Adresse: http://arxiv.org/abs/1706.03762.

[26] Andrej Karpathy, *The unreasonable effectiveness of recurrent neural networks*, http://karpathy.github.io/2015/05/21/rnn-effectiveness/, [Online; zuletzt besucht am 06.06.2019].

[27] Jacopo Urbani Minh Le Marten Postma und Piek Vossen, "A deep dive into word sense disambiguation with lstm", *Proceedings of the 27th International Conference on Computational Linguistics*, S. 354–365, 2018, https://aclweb.org/anthology/C18-1030. [Online; zuletzt besucht am 06.06.2019].

[28] Dayu Yuan, Ryan Doherty, Julian Richardson, Colin Evans und Eric Altendorf, "Word sense disambiguation with neural language models", *CoRR*, Bd. abs/1603.07012, 2016. arXiv: 1603.07012. Adresse: http://arxiv.org/abs/1603.07012.

A. Anhang

A.1. MIDI Übersetzungstabellen

Octave #	Note Numbers											
	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
-1	0	1	2	3	4	5	6	7	8	9	10	11
0	12	13	14	15	16	17	18	19	20	21	22	23
1	24	25	26	27	28	29	30	31	32	33	34	35
2	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59
4	60	61	62	63	64	65	66	67	68	69	70	71
5	72	73	74	75	76	77	78	79	80	81	82	83
6	84	85	86	87	88	89	90	91	92	93	94	95
7	96	97	98	99	100	101	102	103	104	105	106	107
8	108	109	110	111	112	113	114	115	116	117	118	119
9	120	121	122	123	124	125	126	127				

Abb. A.1.: Übersetzungstabelle für Noten [15]

Key#	Note	Drum Sound	Key#	Note	Drum Sound
35	B1	Acoustic Bass Drum	59	B3	Ride Cymbal 2
36	C2	Bass Drum 1	60	C4	Hi Bongo
37	C#2	Side Stick	61	C#4	Low Bongo
38	D2	Acoustic Snare	62	D4	Mute Hi Conga
39	D#2	Hand Clap	63	D#4	Open Hi Conga
40	E2	Electric Snare	64	E4	Low Conga
41	F2	Low Floor Tom	65	F4	High Timbale
42	F#2	Closed Hi Hat	66	F#4	Low Timbale
43	G2	High Floor Tom	67	G4	High Agogo
44	G#2	Pedal Hi-Hat	68	G#4	Low Agogo
45	A2	Low Tom	69	A4	Cabasa
46	A#2	Open Hi-Hat	70	A#4	Maracas
47	B2	Low-Mid Tom	71	B4	Short Whistle
48	C3	Hi Mid Tom	72	C5	Long Whistle
49	C#3	Crash Cymbal 1	73	C#5	Short Guiro
50	D3	High Tom	74	D5	Long Guiro
51	D#3	Ride Cymbal 1	75	D#5	Claves
52	E3	Chinese Cymbal	76	E5	Hi Wood Block
53	F3	Ride Bell	77	F5	Low Wood Block
54	F#3	Tambourine	78	F#5	Mute Cuica
55	G3	Splash Cymbal	79	G5	Open Cuica
56	G#3	Cowbell	80	G#5	Mute Triangle
57	A3	Crash Cymbal 2	81	A5	Open Triangle
58	A#3	Vibraslap			

Abb. A.2.: Übersetzungstabelle für Schlagzeugspuren [15]

A.2. Parameter der Netzwerke

A.2.1. Experiment 1

```
def define_model(X):
    timesteps = X.shape[1]
    features = X.shape[2]

    model = Sequential()
    model.add(Bidirectional(LSTM(20), input_shape=(timesteps,
        features), merge_mode='concat'))
    model.add(Dropout(0.3))
    model.add(Dense(27, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer=adam(lr=
        lr_normalizer(3, adam)), metrics=['binary_accuracy'])
    model.summary()

    return model
```

Quelltext A.1: Parameter zur Erstellung des BLSTM-Netzwerks aus Experiment 1

```
def define_model(X):
    timesteps = X.shape[1]
    features = X.shape[2]

    model = Sequential()
    model.add(LSTM(120, input_shape=(timesteps, features)))
    model.add(Dropout(0.3))
    model.add(Dense(27, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer=adam(lr=
        lr_normalizer(3, adam)), metrics=['binary_accuracy'])
    model.summary()

    return model
```

Quelltext A.2: Parameter zur Erstellung des LSTM-Netzwerks aus Experiment 1

A.2.2. Experiment 2

```
def define_model(X):
    timesteps = X.shape[1]
    features = X.shape[2]

    model = Sequential()
    model.add(Bidirectional(LSTM(20), input_shape=(timesteps,
        features), merge_mode='concat'))
    model.add(Dropout(0.3))
    model.add(Dense(27, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer=adam(lr=
        lr_normalizer(2.5, adam)), metrics=['binary_accuracy'])
    model.summary()

    return model
```

Quelltext A.3: Parameter zur Erstellung des Netzwerks 2a aus Experiment 2

```
def define_model(X):
    timesteps = X.shape[1]
    features = X.shape[2]

    model = Sequential()
    model.add(Bidirectional(LSTM(40), input_shape=(timesteps,
        features), merge_mode='concat'))
    model.add(Dropout(0.3))
    model.add(Dense(27, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer=adam(lr=
        lr_normalizer(2.5, adam)), metrics=['binary_accuracy'])
    model.summary()

    return model
```

Quelltext A.4: Parameter zur Erstellung des Netzwerks 2b aus Experiment 2

A.2.3. Experiment 3

```
def define_model(X):
    timesteps = X.shape[1]
    features = X.shape[2]

    model = Sequential()
    model.add(Bidirectional(LSTM(30, return_sequences=True),
        input_shape=(timesteps, features), merge_mode='concat'))
    model.add(Dropout(0.3))
    model.add(Bidirectional(LSTM(15)))
    model.add(Dense(27, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer=adam(lr=
        lr_normalizer(2.5, adam)), metrics=['binary_accuracy'])
    model.summary()

    return model
```

Quelltext A.5: Parameter zur Erstellung des Netzwerks aus Experiment 3

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 7. Juni 2019 Florian Schultz