1 Konvertierung von Musik in Maschinenlesbare Daten

Bevor wir mit der Klassifikation und Generierung von Musikstücken beginnen können, müssen wir diese in ein Format umwandeln, welches von Maschinen gelesen und verarbeitet werden kann. Hierfür verwenden wir das MIDI-Format (Musical Instrument Digital Interface). MIDI-Dateien enthalten Informationen über Noten, Lautstärke, Tempo und andere musikalische Parameter, jedoch keine Audiodaten selbst. Siehe (Swift, 1997) für eine ausführliche Beschreibung des MIDI-Formats und seine Struktur.

Im Rahmen dieses Projekts sind wir nur an den Notenwerten interessiert, sprich die Tonhöhe. Dabei wird jede Note durch eine (ganze) Zahl zwischen 0 und 127 dargestellt, wobei 60 dem mittleren C entspricht. Das bedeutet natürlich, dass wir keine Informationen über die Klangfarbe, das Instrument, die Lautstärke oder die Dauer (Rhythmus) der Noten haben. Insofern führen wir hier also eine Art Dimensionsreduktion durch und "projezieren" die Musikstücke auf die Tonhöhe der Noten. Das würde dann in etwa so aussehen:

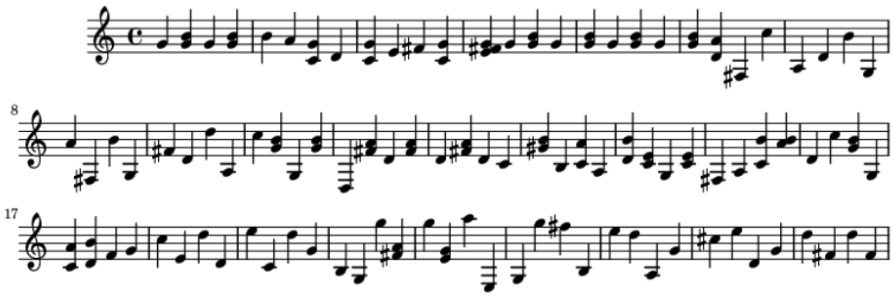

Figure 1: Musikstück nach Reduktion auf Tonhöhe der Noten

Man sieht hier noch eine weitere Diskrepanz: Die Noten sind nicht immer einzeln nacheinander gespielt, sondern teilweise auch gleichzeitig (Akkorde). Daher führen wir noch eine weitere Vereinfachung durch und ordnen die Noten in einen Vektor ein. Dabei geht natürlich wieder Information verloren. Wie genau das Preprocessing durchgeführt wird, ist für die beiden Aufgabenstellungen (Klassifikation und Generierung) unterschiedlich und wird in den jeweiligen Abschnitten beschrieben.

2 Musikgenerierung mit Recurrent Neural Networks

2.1 Preprocessing

Für die Musikgenerierung verwenden wir das Bach-Chorales Dataset, welches ca. 400 Choräle von Johann Sebastian Bach enthält. Diese wurden aus MIDI-Dateien extrahiert und in ein geeignetes Format umgewandelt, welches sich an den vier Stimmen (Sopran, Alt, Tenor, Bass) orientiert. Jede Stimme wird dabei als eine Sequenz von Notenwerten (ganzen Zahlen) dargestellt, siehe das unten stehende Beispiel.

# note0	# note1	# note2	# note3
65	60	57	53
65	60	57	53
65	60	57	53
65	60	57	53
72	60	55	52
72	60	55	52
70	60	55	52
70	60	55	52

Figure 2: Repräsentation eines Chorals im Bach-Chorales Dataset

Wie bereits im ersten Abschnitt erwähnt, wollen wir die Notenwerte in einem Vektor darstellen, was wir in unserem Preporcessing dann auch tun. Somit wir jedes Stück nun als eine Abfolge von ganzen Zahlen in $[0, 127]_{\mathbb{Z}}$ dargestellt.

Nun stellt sich die Frage, wie wir diese Daten unserem RNN zuführen. Konkret müssen wir 3 Hyperparameter festlegen:

- N: Wie viele Noten sollen vorhergesagt werden?
- L: Wie viele vorherige Noten sollen betrachtet werden, um die nächsten Noten vorherzusagen?
- O: Das Offset, also wie viele Noten wir bei der nächsten Vorhersage überspringen.

Um das genauer zu erklären: Sei $S = [n_1, n_2, \ldots, n_k]$ eine Sequenz von Notenwerten. Dann wäre unser erstes Input-Output-Paar für das Training:

$$[n_1, \ldots, n_L] \to [n_{L+1}, \ldots, n_{L+N}]$$

und unser zweites Paar:

$$[n_{1+O}, \ldots, n_{L+O}] \to [n_{L+1+O}, \ldots, n_{L+N+O}]$$

und so weiter.

2.2 Musikgenerierung

Bevor wir unsere verschiedenen Versuche vorstellen, wollen wir kurz erklären, wie die Musikgenerierung mit dem trainierten Modell funktioniert. Gegeben ein Anfangssequenz von Noten $[n_1, \ldots, n_L]$, können wir das Modell verwenden, um die nächsten N Noten vorherzusagen. Dabei erweist es sich als nützlich, nicht deterministisch zu arbeiten, sondern den Output der softmax Schicht als Wahrscheinlichkeitsverteilung zu interpretieren und daraus eine Note zu sampeln, also NICHT immer die Note mit der höchsten Wahrscheinlichkeit zu wählen.

Diese vorhergesagten Noten werden dann an das Ende der Sequenz angehängt, und wir nehmen wieder die letzten L Noten, um die nächsten N Noten vorherzusagen. Diesen Prozess wiederholen wir, bis wir die gewünschte Länge des generierten Musikstücks erreicht haben.

Da es in Latex sehr schwierig ist, Audiodateien einzubetten, findet sich hier ein Link zu einem Kaggle Notebook, welches alle generierten Musikstücke mit Audiodateien enthält: https://www.kaggle.com/code/kristiankelly/audiodateien-fuer-ausarbeitung

2.3 Ein erster Versuch

Gerade stehen wir ganz am Anfang, wir können also nur raten, wie wir N, L, O wählen und welche Architektur wir verwenden sollen. Wir haben auf gut Glück $N = 31$ gesetzt (damit wir nicht immer auf einer Bass-Note enden). Für L scheint 4 eine gute Wahl zu sein, da wir ja 4 Stimmen haben. Für O haben wir uns für 1 entschieden, also immer eine Note weiter. Unsere Architektur ist ein einfaches RNN, welches wie strukturell für den Rest des Projekts verwenden werden, siehe Abbildung 3.

Wir heben folgende Dinge hervor:

- Wir verwenden eine Embedding-Schicht, um die Notenwerte in einen dichteren Vektorraum (in diesem Fall $\mathbb{R}^{64}$) abzubilden, was dem Netzwerk helfen soll, siehe z.B. (Chollet & Chollet, 2021) Kapitel 14. Es ist nämlich bei Noten wie bei Sprachen so, dass Noten, deren Abstände sehr kurz sind (z.B. C und C#) nicht unbedingt eine ähnliche Bedeutungen/Wirkung haben. Somit muss das Netzwerk diese Beziehungen selbst lernen.

- Wir verwenden zwei LSTM (Long Short-Term Memory) Zellen, welche sehr vereinfacht gesagt in der Lage sind, Langzeitabhängigkeiten in Sequenzen zu lernen, siehe z.B. (Chollet & Chollet, 2021) Kapitel 13 für eine ausführliche Beschreibung.

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 31, 64)	5,632
lstm (LSTM)	(None, 31, 128)	98,816
dropout (Dropout)	(None, 31, 128)	0
lstm_1 (LSTM)	(None, 31, 128)	131,584
dropout_1 (Dropout)	(None, 31, 128)	0
lambda (Lambda)	(None, 4, 128)	0
dense (Dense)	(None, 4, 128)	16,512
dense_1 (Dense)	(None, 4, 88)	11,352

Total params: 263,896 (1.01 MB)

Trainable params: 263,896 (1.01 MB)

Non-trainable params: 0 (0.00 B)

Figure 3: Einfaches RNN mit $N = 31, L = 4, O = 1$

- Am Ende nehmen wir uns die letzten $N = 4$ Ausgaben und leiten diese über eine Dense-Schicht mit Softmax-Aktivierung, um eine Wahrscheinlichkeitsverteilung über die möglichen Notenwerte zu erhalten. Da die höchsten und tiefsten Notenwerte im Dataset nicht vorkommen, können wir auf 88 reduzierem, also die Tasten eines Klaviers.

Wir trainieren für 20 Epochen mit Early Stopping und erhalten eine val-accuracy von ca. 0.82 und eine test-accuracy von ca. 0.80. NICHT SCHLECHT! Ein generiertes Musikstück sieht in Abbildung 4 (die ersten zwei Takte sind vorgegeben, der Rest wurde generiert). Uns fallen zwei Dinge auf. Zum einen werden die Akkorde oft wiederholt, was jedoch zu erwarten war, da wir die Rythmus Komponente komplett ignorieren, weswegen eine ganze Note (= 4 Viertelnoten) als 4 gleiche Noten generiert wird. Zum anderen sehen (oder hören wir viel mehr), dass es dem Modell gelungen ist, gewisse Kadenzen (= typische Abfolgen von Akkorden) zu lernen, wie etwa die authentische Kadenz in Takte 6-7. Siehe Amon, 2005, Kapitel über Harmonie und die Funktionalität in Dur und Moll.

Trotzdem beinhaltet das generierte Stück viele unharmonische Übergänge, sprich Kadenzen, die Harmonietheoretisch "keinen Sinn" ergeben. (Wieder verweisen wir bzgl. der genauen Bedeutungen dieser Aussage auf Amon, 2005). Ein Grund hierfür könnte sein, dass wir immer 4 Noten auf einmal vorhersagen und so das Modell vielleicht "zu viel zu schnell" macht und daher Fehler auftauchen. Daher probieren wir im folgenden $N = 1$ aus.

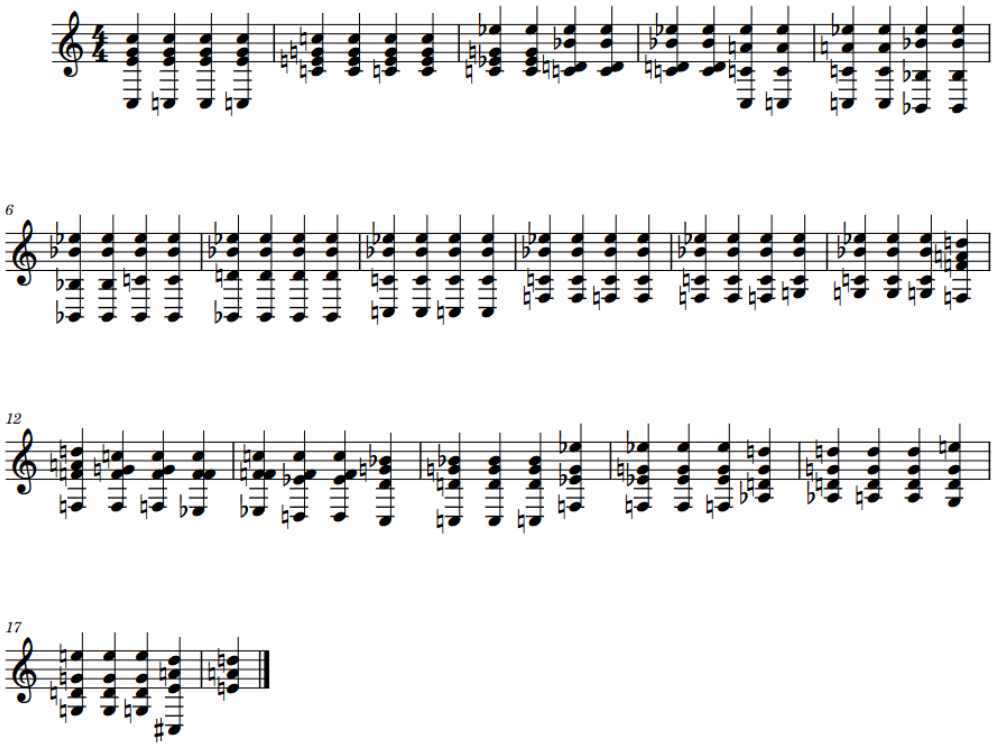

Figure 4: Generiertes Musikstück mit $N = 32, L = 4, O = 1$

2.4 Zweiter Versuch mit $N = 1$

Wir behalten alles bei wie vorhin, nur dass wir jetzt $N = 1$ wählen, also immer nur eine Note vorhersagen. Wir trainieren für 30 Epochen mit Early Stopping und erhalten eine val-accuracy von ca. 0.85 und eine test-accuracy von ca. 0.83. BESSER! Ein generiertes Musikstück sieht man in Abbildung 5 (wieder sind die ersten zwei Takte vorgegeben, der Rest wurde generiert). Dieses Stück ist harmonisch deutlich besser gelungen (es klingt nichts "falsch"). Jedoch ist dieses auch nur das beste aus 10 Versuchen, welches wir generiert haben. Die anderen hatten fast alle zumindest ein paar Stellen, die sehr dissonant klangen. Im Barock (die Epoche, in der Bach gelebt hat) waren solche unharmonischen Stellen eher unüblich und man hat sich an strikte harmonische Regeln gehalten, wie etwa den Kontrapunkt Tittel, 1959. Als nächsten Schritt wollen wir versuchen, diese Regeln irgendwie in das Modell zu integrieren, um konsistenter harmonische Stücke zu generieren.

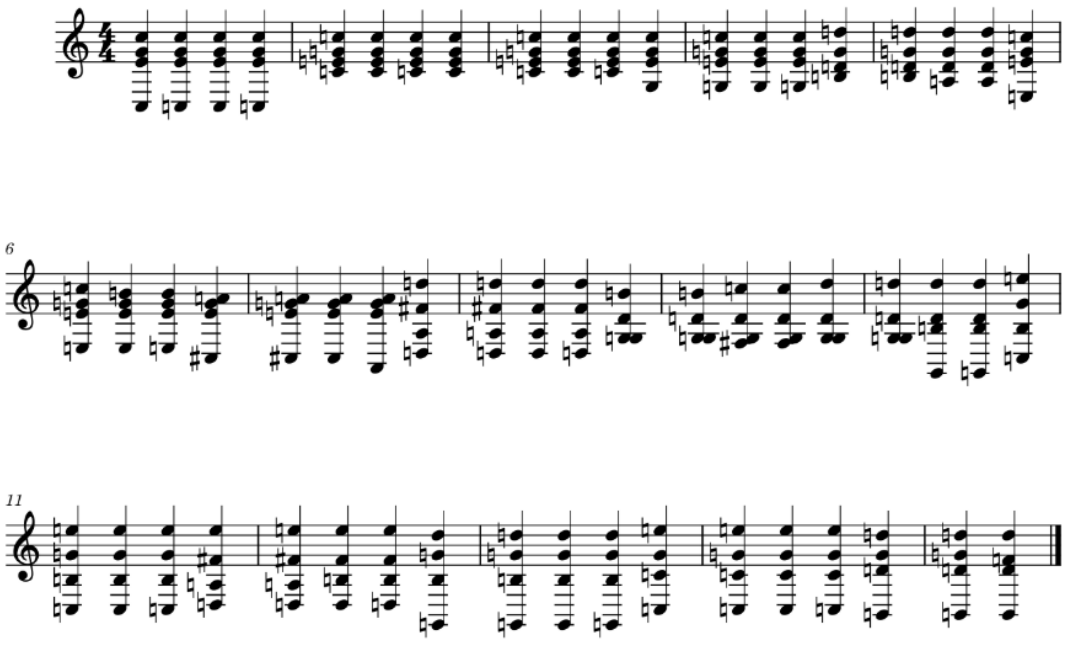

Figure 5: Generiertes Musikstück mit $N = 32, L = 1, O = 1$

2.5 Eine neue Loss Funktion

Bis jetzt haben wir immer die Sparse-Categorical-Crossentropy als Loss Funktion verwendet. Nun wollen wir dissonante Fehler mehr bestrafen als konsonante. Dafür brauchen wir zuerst ein Grundwissen über Intervalle, die Grundbausteine der Harmonielehre. Ein Intervall ist der Abstand zwischen zwei Tönen, gemessen in Halbtonschritten. Man unterscheidet zwischen perfekten, konsonanten und dissonanten Intervallen Amon, 2005. Eine Übersicht über die wichtigsten Intervalle ist in der Tabelle 1 dargestellt. Ist also $v^{(n)} \in \mathbb{R}^{88}$ unsere Vorhersage für eine richtige Note $n \in [0, 88]_{\mathbb{Z}}$, wollen wir einen Vektor $s^{(n)} \in \mathbb{R}^{88}$ definieren, welcher die Einträge von v je nach Intervall zum vorhergesagten Ton bestraft. Wir definieren $s^{(n)}$ wie folgt:

$$s_i^{(n)} = \begin{cases} 0, & |i-n| = 0 \mod 12 \text{ (Prime)} \\ w_1, & |i-n| = 5, 7 \mod 12 \text{ (perfekte Konsonanz)} \\ w_2, & |i-n| = 3, 4, 8, 9 \mod 12 \text{ (Konsonanz)} \\ w_3, & |i-n| = 1, 2, 6, 10, 11 \mod 12 \text{ (Dissonanz)} \end{cases}$$

mit $0 < w_1 < w_2 < w_3$ positiven Gewichten. Unser neuer Loss für eine Vorhersage $v^{(n)}$ mit richtiger Note n ist dann:

$$\text{Loss}(v^{(n)}, n) = \text{SparseCategoricalCrossentropy}(v^{(n)}, n) + v^{(n)} \cdot s^{(n)}$$

Intervall	Halbtöne	Beispiel (von C)	Konsonanz
Prime	0	C - C	Perfekte Konsonanz
kleine Sekunde	1	C - C#	Dissonanz
große Sekunde	2	C - D	Dissonanz
kleine Terz	3	C - D#	Konsonanz
große Terz	4	C - E	Konsonanz
reine Quarte	5	C - F	Perfekte Konsonanz
verminderte Quinte / Tritonus	6	C - F#	Dissonanz
reine Quinte	7	C - G	Perfekte Konsonanz
kleine Sexte	8	C - G#	Konsonanz
große Sexte	9	C - A	Konsonanz
kleine Septime	10	C - A#	Dissonanz
große Septime	11	C - B	Dissonanz
Oktave	12	C - C	Perfekte Konsonanz

Table 1: Musikalische Intervalle von Prime bis Oktave mit Konsonanz/Dissonanz Amon, 2005

wobei $\cdot$ das euklidische Skalarprodukt ist. Wir bemerken, dass der zweite Term genau dann 0 ist, wenn die Note korrekt vorhergesagt wurde, und umso größer wird, je "dissonanter" die Vorhersage ist, also genau dass, was wir wollten.

Trainieren wir nun dasselbe Modell wie im vorherigen Abschnitt mit dieser neuen Loss Funktion, wobei wir $w_1 = 1, w_2 = 2, w_3 = 3$ setzen, erhalten wir nach 30 Epochen eine val-accuracy von ca. 0.85 und eine test-accuracy von ca. 0.84. Wir haben uns also dahingehend nicht wirklich verbessert. Aber wie schauen die generierten Musikstücke aus? Nunja, wir haben unser Ziel erfüllt: Die generierten Stücke sind harmonisch perfekt. Leider bedeutet "perfekt" in diesem Fall, das die ganze Zeit nur der erste Akkord (die Tonika) gespielt wird. Das gibt uns zwar das objektiv beste harmonische Stück, aber es ist natürlich unglaublich langweilig.

Wir starten einen zweiten Versuch mit $w_1 = 0.01, w_2 = 0.1, w_3 = 1$, also wir bewegen uns auf einer logarithmischen Skala. Die val-accuracy und die test-accuracy bleiben gleich, aber die generierten Stücke sind nun das, was wir uns erhofft haben: Harmonisch sinnvoll und abwechslungsreich (und das bei fast jeder Generierung). Ein Beispiel ist in Abbildung 6 dargestellt.

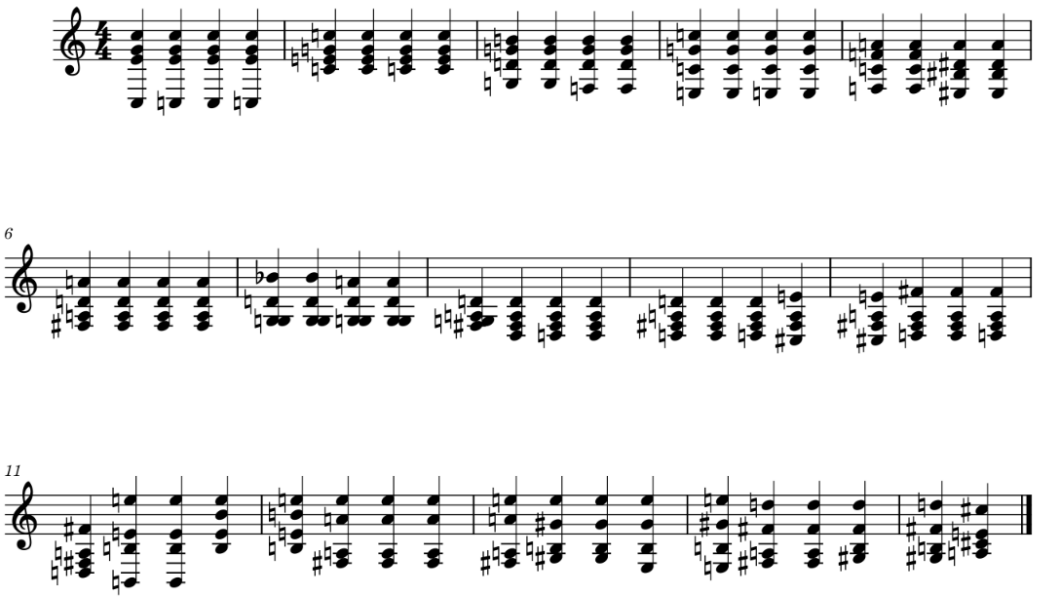

Figure 6: Generiertes Musikstück mit $N = 32, L = 1, O = 1$ und neuer Loss Funktion

References

Amon, R. (2005). *Lexikon der harmonielehre: Nachschlagewerk zur durmolltonalen harmonik mit analysechiffren für funktionen, stufen und jazzakkorde*. Springer.

Chollet, F., & Chollet, F. (2021). *Deep learning with python*. simon; schuster.

Swift, A. (1997). Standard midi file format [Accessed: 2026-01-07]. https://web.archive.org/web/20120830211425/http://www.doc.ic.ac.uk/~nd/surprise_97/journal/vol1/aps2/

Tittel, E. (1959). *Der neue gradus: Textteil* (Vol. 1). Doblinger.