令和２年度　卒業研究

AIを用いた作曲支援手法に関する研究

Research on Musical Composition Support Method using AI

函館工業高等専門学校
生産システム工学科　情報コース　５年
釜石健太郎
指導教員　東海林智也

目次

第 1 章　はじめに　2
- 1.1　概要 (Abstract)　2
- 1.2　背景　2
- 1.3　目的　2

第 2 章　使用技術　3
- 2.1　開発環境　3
- 2.2　LSTM　3

第 3 章　提案手法　5
- 3.1　手法 1：主旋律の自動生成　5
  - 3.1.1　概要　5
  - 3.1.2　ニューラルネットワークの仕様　5
  - 3.1.3　手順　6
- 3.2　手法 2：伴奏の自動生成　9
  - 3.2.1　概要　9
  - 3.2.2　ニューラルネットワークの仕様　9
  - 3.2.3　手順　9
- 3.3　手法 3：主旋律への伴奏付け　11
  - 3.3.1　概要　11
  - 3.3.2　ニューラルネットワークの仕様　11
  - 3.3.3　手順　12

第 4 章　結果と考察　14
- 4.1　主旋律の自動生成　14
- 4.2　伴奏の自動生成　15
- 4.3　主旋律への伴奏付け　16

第 5 章　まとめ　17

付録 A　ソースコード　18

参考文献　19

1　はじめに

1.1　概要 (Abstract)

The study aims to devise some methods to support composition by AI. Eventually, I'd like to find the way how to add accompaniment to a melody line given by the user. This is because I thought that many people are interested in composing music, even if they cannot play an instrument, but only have an idea of the melody. In this research, I mainly used deep learning with LSTM, which can learn the relationship by time. As a result, I was successful in implementing all methods I expected. However, they have problems with the quality of the melodies and accompaniments they generate.
Key words：Composition, Accompaniment, AI, LSTM

1.2　背景

作曲用ソフトである DAW (Digital Audio Workstation) や、ボーカロイドの普及によって、一般人でも作曲が気軽にできる環境が整いつつある [1]。 特に、考えたメロディを鼻歌で口ずさんだり、頭の中にメロディが浮かんだりすることは多くの人にあると思う。しかし、作曲するには音楽理論や音色に関する前提知識と、楽器、特にキーボードの演奏スキルが必要となるため、多くの人が曲を仕上げるには以下のような問題がある。

- 思いついたメロディを一曲分の長さに仕上げられない
- メロディに伴奏をつけられない

前提知識を持っていても楽器が弾けない場合は、マウスで一つずつ音符を入力することになるので、上のことはできても時間がかかってしまう。

一方で、近年流行りのディープラーニングには、短期間及び長期間にわたる時間系列の関係を学習できるアーキテクチャ「LSTM」(Long Short-Term Memory) [2] があり、それを使った音楽生成の研究は頻繁に行われている。しかし、それらの多くは曲全体を AI が生成してしまうものであり、メロディが与えられた前提のものはほとんどない。

1.3　目的

これらの背景を受け、ユーザが与えたメロディに自動で伴奏を付けるニューラルネットワークの構築を行うことを当研究の目的とする。ディープラーニング、特に LSTM を用いて、音楽の持つ時間系列の関係を学習することで、段階的にいくつかの作曲手法を考案し、前述の問題の解決を試みる。

2　使用技術

2.1　開発環境

OS　Windows10

CPU　AMD Ryzen 5 4600H with Radeon Graphics 3.00GHz

RAM　16.0GB

使用言語　Python(Jupyter Notebook)

主な使用ライブラリ　Keras[3], music21[4], numpy

入出力　MIDI ファイル

学習データ　著作権フリーの童謡の MIDI ファイル 34 個

2.2　LSTM

LSTM (Long Short-Term Memory) [2] とは、RNN (Recurrent Neural Network) を拡張したニューラルネットワークアーキテクチャの一種である。そのため、初めに RNN について説明する。

RNN は、再帰的な構造をもったニューラルネットワークである (図 2.1)。t 番目の入力データに対する隠れ層の出力 h_t が、次の $t+1$ 番目の入力データと共に入力されるという特徴を持つ。前の出力によって次の出力が決まるため、時系列データを学習するのに適している。しかしながら、RNN には正しく学習できるのは短期間の情報のみで、長期依存のものは学習できないという欠点がある。

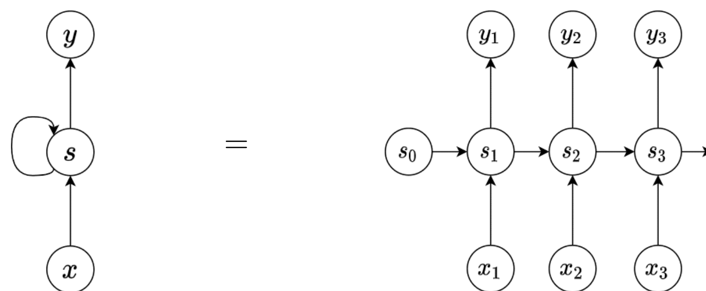

図 2.1: RNN

その解決のために開発されたのが LSTM である (図 2.2)。

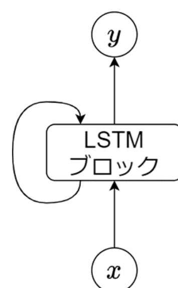

図 2.2: LSTM

LSTM では短期間だけでなく、長期間の時間依存関係も学習できる。LSTM では、RNN の中間層のユニットが LSTM Block というブロックに置き換えられている。LSTM Block は次の 3 種類のゲートを持っており、これらが長期記憶を可能にしている。

入力ゲート　入力を取り込むかを決める

出力ゲート　次の時刻にどの程度情報を伝えるかを決める

忘却ゲート　保持している情報をリセットするかを決める

3　提案手法

背景で示した「 思いついたメロディを一曲分の長さに仕上げられない」「メロディに伴奏をつけられない」という問題を解決するために、本研究では「主旋律の自動生成」「伴奏の自動生成」「主旋律への伴奏付け」 という 3 つの手法について検討を行った。この章ではそれら 3 つの手法の詳細について説明する。

3.1　手法 1：主旋律の自動生成

3.1.1　概要

はじめに、背景で示した「思いついたメロディを一曲分の長さに仕上げられない」問題を解決するために、主旋律の初めの数音を与えるとその続きを生成するモデルを作成した。主旋律とは曲の中心となるメロディのことである。ポップスでいうと、ボーカルが歌うメロディラインに相当する。

3.1.2　ニューラルネットワークの仕様

今回構成するネットワークとして 2 つのモデルを考えた。モデル 1 を図 3.1 に示す。エポック数は 100 とした。

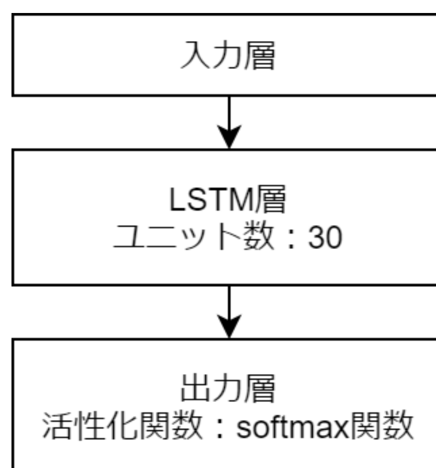

図 3.1: モデル 1

またモデル 2 を図 3.2 に示す。こちらのエポック数は 500 とした。モデル 1 との違いは、過学習対策としてドロップアウト層の追加と L2 正則化を施した点である。

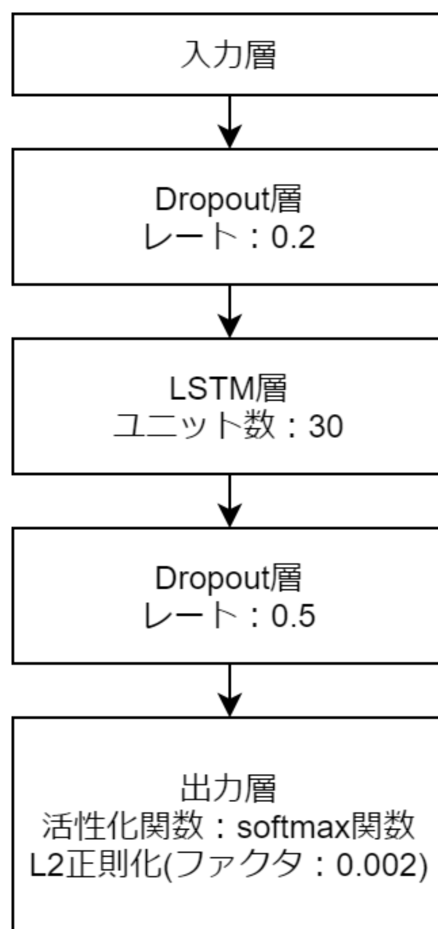

図 3.2: モデル 2（過学習対策版

どちらのネットワークも損失関数は交差エントロピー、optimizer は RMSprop(学習率=0.01) を用いた。また validation_split（訓練データのうち何割を検証データとして使うか）は 0.2、バッチサイズ は 256 とした。文章数は 6822、文章長は 16 とした (用語については次節で説明する)。学習データである童謡の MIDI ファイルは 34 個全て用いた。

3.1.3　手順

入力の音符と休符を文字列に変換することで自動文章生成の問題に帰着させる。最後に出来上がった文章を音符または休符に戻し、MIDI ファイルとして出力する。具体的には次のような手順になる。

1　準備

各 MIDI ファイルについて、MIDI ファイルのある場所 (パス) と主旋律のトラック番号を指定したテキストファイルを作成する。今回は次のような形式で保存した。

(MIDI ファイル 1 のパス) = (MIDI ファイル 1 の主旋律のトラック番号)
(MIDI ファイル 2 のパス) = (MIDI ファイル 2 の主旋律のトラック番号)
・・・

2　音符・休符の文字列への変換

上で作成したテキストファイルをもとに各 MIDI ファイルから主旋律のトラックを抜き出し、ライブラリ music21 を使ってトラックをハ長調に移調してから音符と休符のリストを取得する。その後、リストの各要素を次の形式で文字列に変換した。

音符 ‘（アルファベット音名）_（長さ）‘

休符 ‘Rest_（長さ）‘

ここでの長さは 4 分音符を 1 とする。また、主旋律では同時に複数の音が鳴らないと仮定している。

3 辞書の作成

上で変換した文字列を、変換した順に一つのリスト chars へ格納する。chars の各要素は自動文章生成になぞらえて「文字」と呼ぶことにする。

さらに chars の各要素からインデックスを特定する辞書 char_indices、およびその逆引き辞書 indices_char を作る。

具体的には chars の文字を先頭から見ていき、もし char_indices に含まれていなかったらその文字を char_indices にインデックスを付けて追加する。そのため、同じ文字は先に出た方のインデックスが使われる。逆引き辞書は char_indices のキーと値を逆にして作成する。

参考として表 3.1 に、ある chars が与えられた際の辞書の作成例を示す。

表 3.1: 辞書の作成例

chars	[‘C_0.5 ’, ‘Rest_1.5 ’, ‘D_0.5 ’, ‘C_0.5 ’, ...]
辞書 (char_indices)	{‘C_0.5 ’:0, ‘Rest_1.5 ’:1, ‘D_0.5 ’:2, ...}
逆引き辞書 (indices_char)	{0:‘C_0.5 ’, 1:‘Rest_1.5 ’, 2:‘D_0.5 ’, ...}

4 文章の作成

次に chars を先頭から長さ maxlen ごとに切り出し、リスト sentences に格納する。つまり、

```
sentences[i] = [chars[i], chars[i+1], ..., chars[i+maxlen-1]]
```

とする。ここで sentences の各要素を「文章」と呼ぶことにする。

また i 番目の文章 sentences[i] を LSTM に与えた時の出力を、その文章の次の文字である chars[i+maxlen] とする。この文字のことを「正解文字」と呼ぶことにし、正解文字のリスト next_char を作る。つまり

```
next_char[i] = chars[i+maxlen]
```

とする。

5 One-Hot ベクトル化

このようにして作成したリスト sentences と next_char に含まれる文章や正解文字を辞書 char_indices を用いて One-Hot ベクトル化し、それぞれ入力リスト input_data、正解リスト output_data とする。

One-Hot ベクトルとはある要素だけが 1 で、他はすべて 0 のベクトル (リスト) のことである。今回の One-Hot ベクトルは長さ len(char_indices)(= char_indices の要素数) の 1 次元リストとなる。One-Hot ベクトル v にしたい文字を c とすると、v は v[char_indices[c]]=1 で、それ以外が 0 となる。

結果的に、input_data は 3 次元 (i, 時刻,v)、output_data は 2 次元のリスト (i,v) になる。

6 学習

input_data を学習データ、output_data を正解データとしてモデルに渡し、学習を行う。ネットワークの仕様は 3.1.2 で示した通りである。

7　主旋律の自動生成

学習済みのモデルに主旋律の初めの数音を与え、次の文字の予測を繰り返すことで主旋律の続きが文字列として自動生成される。この際にモデル出力は各文字が出力される確率分布と見なせるが、この分布の分散を大きくすることで、モデルからの出力にバラつきを持たせることが出来る。

このようにして生成された文章を `generated` とする。

8　出力

生成された文章 `generated` を文字ごとに区切る。それらから順に音名と長さの情報を取り出し、music21 を使って MIDI ファイルとして出力する。

3.2 手法 2：伴奏の自動生成

3.2.1 概要

次に「メロディに伴奏をつけられない」という問題を解決するために、伴奏として「コード」または「対旋律」の初めの数音を与えるとその続きを生成するモデルを作成した。

コードとは、高さの異なる音を同時に響かせたもののことで、音楽の三要素の「ハーモニー」に該当する。コードの変化をコード進行といい、コード進行が曲の雰囲気を決める。

対旋律とは、主旋律を引き立たせるために、同時に演奏されるメロディのことである。

3.2.2 ニューラルネットワークの仕様

ネットワーク構成は LSTM のユニット数を 64 に変更した以外は、主旋律のモデル 1(図 3.1) と同じである。損失関数や optimizer も同じだが、RMSprop の学習率を 0.001 にした。パラメータはエポック数が 200、validation_split が 0.2 である。文章数は 1516 で、文章長は 10 である。学習データである童謡の MIDI ファイルは全 34 個のうち、トラック指定が容易だった 5 個だけを用いた。

3.2.3 手順

1 準備

学習対象の MIDI ファイルから伴奏 (コードまたは対旋律) のトラックを取り出す。

2 音符の文字列への変換

主旋律のときと同様に、伴奏 (コードまたは対旋律) も文字列へ変換する。文字への変換方式手順は主旋律のときとほぼ同様であるが、今回は以下の様な問題がある。

問題 1. コードと対旋律は一つのトラックからなるとは限らない
問題 2. 単音が順番に鳴っているのか、複数の音が同時に鳴っているのかを区別できない

問題 1 については、コードに関しては複数トラックを統合し、1 つのトラックにすることで解決した。また対旋律に関しては各トラックを独立に扱うことで解決した。

問題 2 については、例として次の 2 つの状況を考える。

状況 1. 曲の先頭から 4 分音符ド、ミ、ソが順番に鳴る
状況 2. 曲の先頭で 4 分音符ド、ミ、ソが同時に鳴る

2 つの状況は明らかに別のものであるが、主旋律のときの方法で文字列に変換するとどちらも‘`C_1.0`‘,‘`E_1.0`‘,‘`G_1.0`‘と変換されてしまう。

そこで伴奏を文字列に変換する際に、新たに「オフセット」の概念を導入した。ある音符のオフセットをここでは「その音符の先頭が前の音符の先頭とどれくらい離れているか」によって定義する。なおオフセットに関しても四分音符を 1 とする。

さらに加えて、文字列への変換方式として主旋律からの差分を用いる形式も新たに導入する。主旋律からの差分とは、同時に鳴っている主旋律の音と何半音離れているかという値である。もし対応している主旋律がない場合は「ド」が鳴っているものとして扱う。主旋律からの差分を考えることでことで、より主旋律に沿った伴奏が生成されると考えた。

以上をまとめると、コードと対旋律の音符を文字列へ変換するために以下の 2 つの形式を導入する。

音符変換形式 1 ‘(アルファベット音名)_(長さ)_(オフセット)‘
音符変換形式 2 ‘(主旋律からの差分)_(長さ)_(オフセット)‘

オフセットを導入することで休符を扱う必要がなくなるため、音符のみを変換することにする。

例えば音符変換形式 1 を使って上の状況 1 を変換すると‘C_1.0_0.0‘，‘E_1.0_1.0‘，‘G_1.0_1.0‘となり、状況 2 を変換すると‘C_1.0_0.0‘，‘E_1.0_0.0‘，‘G_1.0_0.0‘となるので、それぞれの状況が区別される。

3 辞書の作成

主旋律のときと同様にして辞書を作成する。

4 文章の作成

主旋律のときと同様にして文章を作成する。

5 One-Hot ベクトル化

主旋律のときと同様にして文章を One-Hot ベクトル化する。

6 学習

主旋律のときと同様にして学習を行う。

7 伴奏の自動生成

主旋律のときと同様にして、学習済みのモデルに伴奏の初めの数音を与えると伴奏の続きが自動生成される。

8 出力

主旋律のときと同様にして伴奏を MIDI ファイルとして出力する。

3.3 手法 3：主旋律への伴奏付け

3.3.1 概要

手法 2 と同様に「メロディに伴奏をつけられない」という問題を解決するために、ユーザが指定した主旋律に合った伴奏 (コードまたは対旋律) を自動生成するためのモデルを作成した。手法 2 との違いは、モデルに伴奏の初めの数音ではなく、主旋律を入力する点である。

3.3.2 ニューラルネットワークの仕様

機械翻訳でよく用いられる手法である seq2seq[5] を用いる。seq2seq における訓練時と推論時の動きをそれぞれ図 3.3、図 3.4 に示す。このように、モデルへの入力は主旋律および伴奏、出力は伴奏となる。

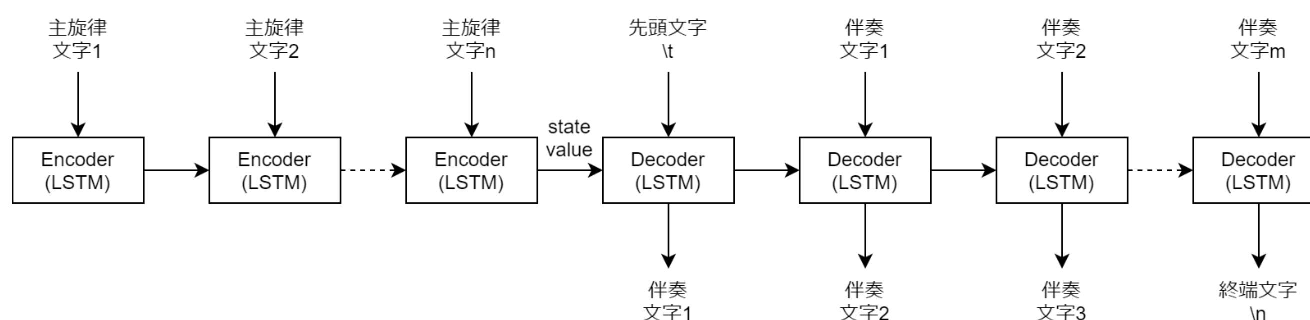

図 3.3: 訓練時

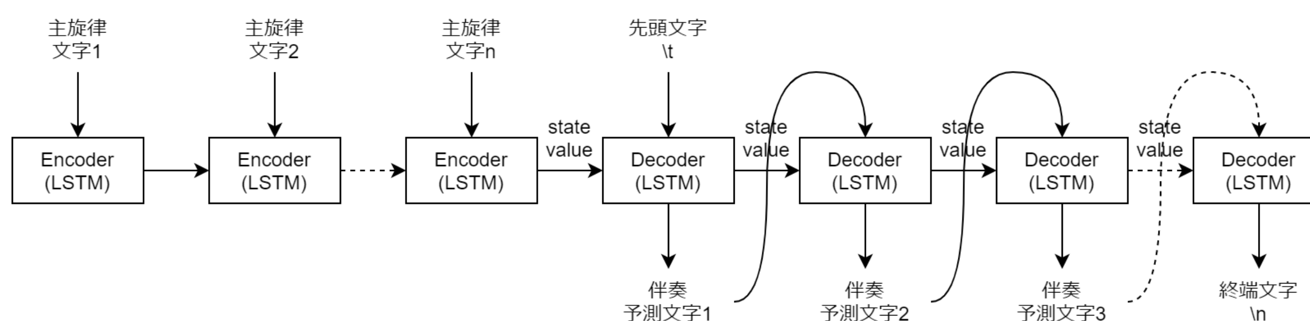

図 3.4: 推論時

訓練時は Decoder の正解が与えられているが、推論時は与えず、状態値と前の予測文字を使って進めていく。また、先頭文字と終端文字を用いることで出力される伴奏を可変長にしている。

またネットワーク構成図を図 3.5 に示す。

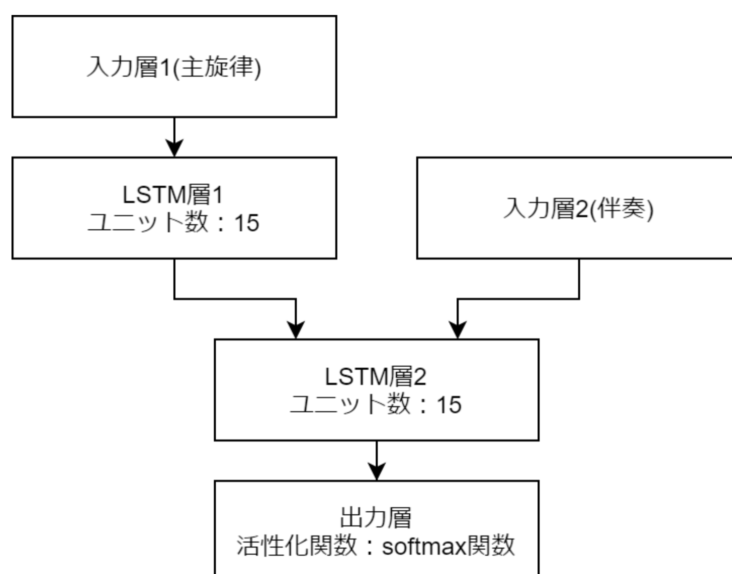

図 3.5: ネットワーク構成図

損失関数と optimizer は伴奏の自動生成のときと同じである。パラメータはエポック数が 500、バッチサイズが 1、validation_split が 0.2 である。文章数は 24 で、文章長は学習データに応じて可変である。学習データである童謡の MIDI ファイルは全 34 個のうち、1 つのみを用いた。

3.3.3 手順

1 準備

学習対象の MIDI ファイルから主旋律と伴奏のトラックを取り出す。

2 音符の文字列への変換

これまでと同様に、取り出した主旋律と伴奏のトラックそれぞれに対し音符のリストを取得し、文字列に変換する。なお文字列への変換形式として 3.2.3 の音符変換形式 1 を採用した。

3 辞書の作成

これまでと同様に辞書も作成する。ただし主旋律と伴奏の両トラックについて辞書を作成し、それぞれを `input_token_indices`、`output_token_indices` とする。

逆引き辞書も同様に作成する。

4 文章の作成

次に主旋律と伴奏の両トラックに対し 0.5 小節ごとに文章を分けてリストへ格納し、それぞれを `InputSentences`、`OutputSentences` とする。

各リストの i 番目には、曲の先頭から見て $0.5i$ 小節〜$0.5(i+1)$ 小節区間の文章を連結して入れる。なお音符のオフセットの定義は「所属する小節の先頭からその音符の先頭がどれくらい離れているか」とする。オフセットの単位は変わらず四分音符を 1 とする。

また主旋律と伴奏の組を作れない小節の文章については `InputSentences` と `OutputSentences` から削除する。

5 One-Hot ベクトル化

手順としてはこれまでの手法とほとんど同じだが、正解データは伴奏の方だけ作成する。また、各文章の長さが異なるので、主旋律、伴奏のそれぞれの最大文章長に統一して One-Hot ベクトル化する。

主旋律、伴奏の One-Hot ベクトル化した入力リストをそれぞれ `encoder_input_data`、`decoder_input_data`、

伴奏の One-Hot ベクトル化した正解リストを `decoder_output_data` とする。

6 学習

`encoder_input_data` と `decoder_input_data` を学習データ、`decoder_output_data` を正解データとしてモデルに渡し、学習を行う。ネットワークの仕様は 3.3.2 で示した通りである。

7 主旋律への伴奏付け

学習済みのモデルに伴奏をつけたい主旋律を与え、図 3.4 で示した推論を繰り返すことで、伴奏が生成される。なお今回は 0.5 小節単位でモデルに入力を与えるので、伴奏も 0.5 小節単位で生成される。

8 出力

入力した主旋律と生成された伴奏を一つの MIDI ファイルとして出力する。

4　結果と考察

4.1　主旋律の自動生成

モデル 1(図 3.1) における学習曲線を図 4.1、モデル 2(図 3.2) における学習曲線を図 4.2 に示す。なお左側が正解率 (accuracy)、右側が損失 (loss) のグラフで、青が学習データ、オレンジが評価データの推移を示している。

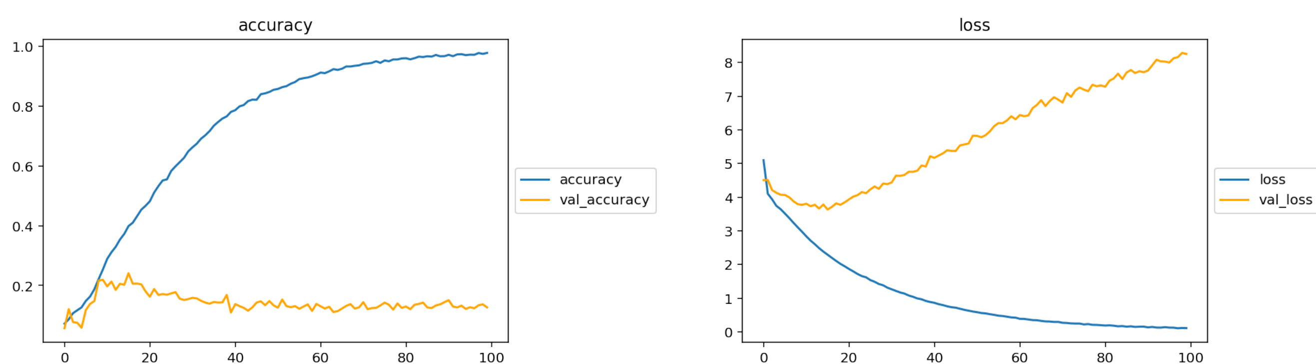

図 4.1: モデル 1 の学習曲線

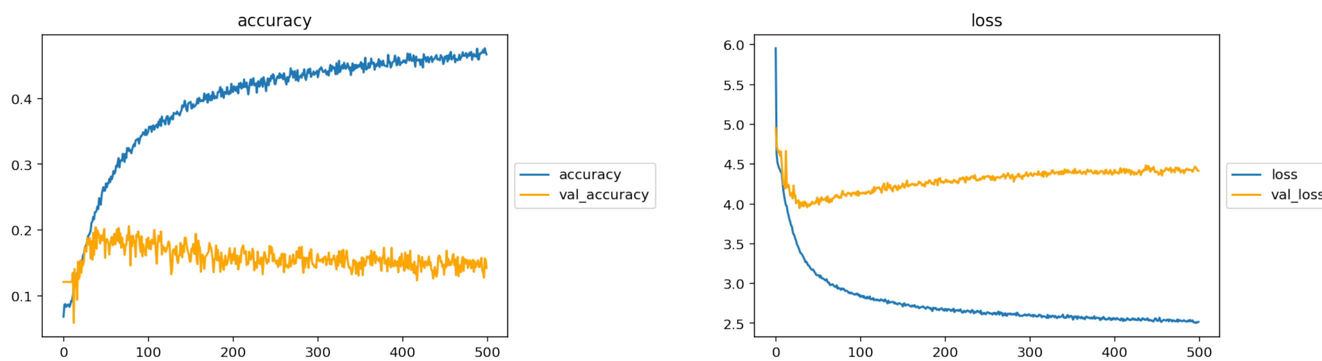

図 4.2: モデル 2（過学習対策版）の学習曲線

どちらのモデルも評価データの正解率 (accuracy) が低く、学習曲線にも揺れが見られる。よって学習率をもっと下げる必要があると考えられる。

またモデル 1 では評価データの損失 (loss) が増大していることから過学習 [6] の傾向を示している。モデル 2 でドロップアウト層の追加と正則化を行っても大きな改善は見られなかった。

なお実際に生成された主旋律の聴感としてはリズム感が不安定に感じた。音程の変化の仕方は不自然ではなかった。従って、現状では修正を前提にすればどちらのモデルも主旋律の生成手法として有用なモデルであると言える。

4.2 伴奏の自動生成

音符変換形式 1(アルファベット音名) における学習曲線を図 4.3、音符変換形式 2(主旋律からの差分) における学習曲線を図 4.4 に示す。なお左側が正解率 (accuracy)、右側が損失 (loss) のグラフで、学習データ数が少なかったので評価データの曲線は表示していない。

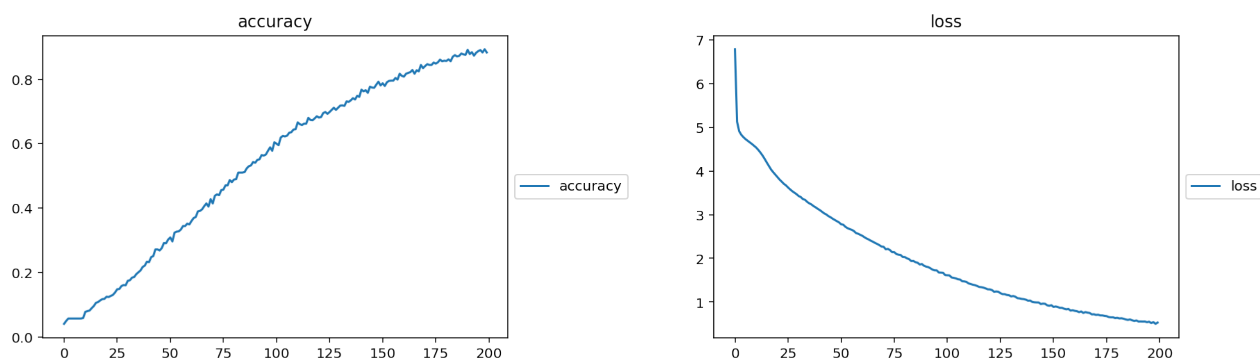

図 4.3: 音符変換形式 1 の学習曲線

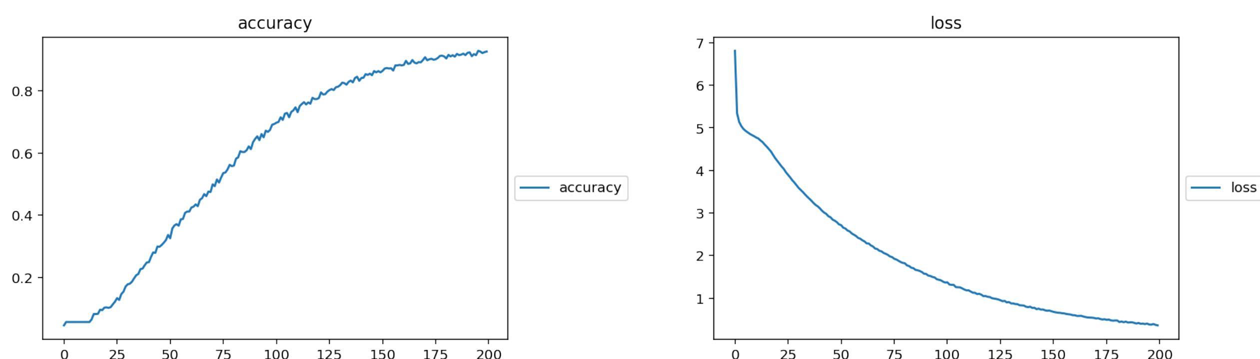

図 4.4: 音符変換形式 2 の学習曲線

実際に生成された伴奏を聞いたところ、どちらの形式とも毎回学習データに似たような伴奏が生成されていた。また同じフレーズの反復が多いようにも感じた。これは学習データが少なく、かつ過学習を起こしていることが原因であると考えられる。

なお形式 2 の方は不協和な伴奏が生成されていた。これは主旋律からの離れ具合だけを学習したため、文章作成中に学習データとのずれが大きくなり、音階から外れた音が増えてしまったことが原因であると考えられる。従って、現状では形式 1 の方が伴奏生成手法として有用な手法であると言える。

4.3 主旋律への伴奏付け

学習曲線を図 4.5 に示す。なお左側が正解率 (accuracy)、右側が損失 (loss) のグラフで、学習データ数が少なかったので評価データの曲線は表示していない。

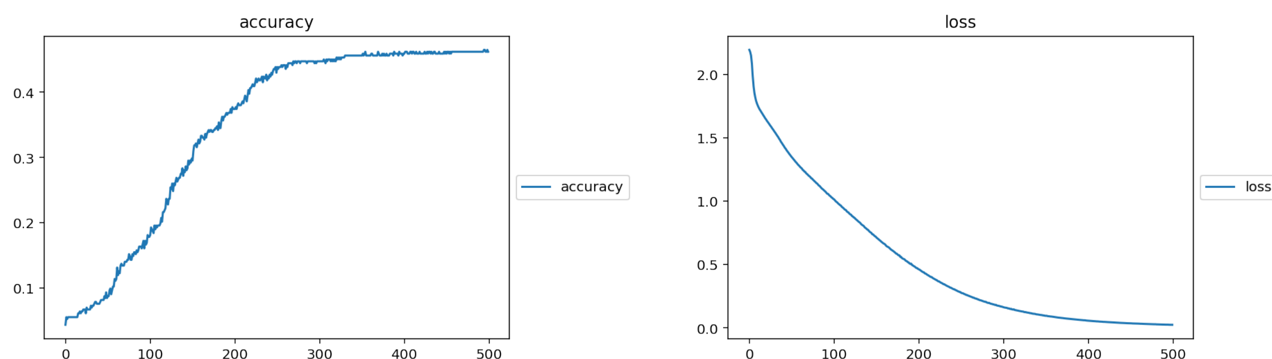

図 4.5: 学習曲線

学習データで使用した主旋律を入力として与えて生成された伴奏を聞いたところ、自然ではあるが元の伴奏と似ている伴奏が生成されていた。一方、学習データ以外の主旋律を入力として与えたときは、不協和な伴奏が生成されたが、リズム感はそれほど悪くないと感じた。これは過学習が原因であると思われるので、学習データを増やすことで、過学習が改善されてより独創的な伴奏がつくかもしれない。

従って、現状では学習データと似ている別パターンの伴奏を提案するためには有用な手法であるといえる。

5　まとめ

本研究では、ユーザが与えたメロディに自動で伴奏を付けるために「主旋律の自動生成」「伴奏の自動生成」「主旋律への伴奏付け」という 3 つの手法について検討し、実験の結果、どれも有用な手法であることが分かった。

課題としては学習データの少なさが挙げられる。学習データが少なかった理由は、歌の入った曲を生成したかったために学習データの種類を童謡に絞ったこと原因である。学習データが増えると、生成の質がより向上すると考えられる。

またユーザ入力に辞書に登録されていない文字が入った際に曲が生成されない問題も生じた。特に「主旋律への伴奏付け」ではこの問題が起こりやすかったため、さらなる学習データの収集や、GAN などのデータが少なくても実行出来る手法の検討が必要である。

さらに現状ではコマンドラインから実行しているので、ユーザが使いやすい UI を用意し、手軽に楽曲の生成を行えるようにすることが必要である。

A　ソースコード

ソースコードは `https://github.com/Kentaro-Kamaishi/AICompositionSupport` に掲載した。
本文とファイルの対応は次の通りである。

主旋律の自動生成　MelodyGeneration.ipynb
伴奏の自動生成 (音符変換形式 1)　AccompGeneration(way1).ipynb
伴奏の自動生成 (音符変換形式 2)　AccompGeneration(way2).ipynb
主旋律への伴奏付け　AccompAttach.ipynb

参考文献

[1] https://sakkyoku.info/beginner/things-necessary-composition/

[2] https://qiita.com/KojiOhki/items/89cd7b69a8a6239d67ca

[3] https://keras.io/ja/

[4] http://web.mit.edu/music21/

[5] https://qiita.com/FukuharaYohei/items/27cd247342a0f7006511

[6] https://axa.biopapyrus.jp/machine-learning/model-evaluation/learning-curves.html