令和 2 年度

修士学位論文

Transformer Decoder を用いたポピュラー音楽におけるメロディからの伴奏生成

Popular Music Accompaniment Generation from Melody Using Transformer Decoder

1235057　　沖　貴司

指導教員　　高田　喜朗

2021 年 3 月 1 日

高知工科大学大学院 工学研究科 基盤工学専攻

情報学コース

要 旨

Transformer Decoder を用いた ポピュラー音楽におけるメロディからの 伴奏生成

沖 貴司

近年，音楽配信サービスや動画サービスの発展により，音楽が身近なものとなっている．また，作曲用ソフトウェア等も発展しており，作曲に関する高度な専門知識が無い人が，音楽鑑賞だけでなく作曲にも取り組む機会が増えている．

作曲では，多くの場合にメロディと伴奏を制作する必要があるが，伴奏の制作は高度な専門知識や経験が無ければ困難である．よって，コンピュータによる伴奏生成に関する多くの試みがなされている．近年は，ニューラルネットワークの 1 つである Transformer を用いることが有効なアプローチとして多くの研究がなされている．一方，近年の自然言語処理タスクでは，Transformer の一部の構造を用いる多くのモデルが優れた結果を出している．入力を伴う生成に関しても，Transformer の一部である Transformer Decoder を用いて優れた結果を出すことができる．

本研究では，Transformer Decoder を用いたポピュラー音楽におけるメロディからの伴奏生成手法を提案する．この手法の性能を測るために，Transformer を用いた生成手法との比較実験を行った．比較実験は，多くの音楽自動生成に関する研究で用いられるリスニングテストによる主観的評価で行った．その結果，Transformer Decoder を用いることで優れた品質の伴奏の生成が可能であることが示された．

キーワード　自動伴奏生成，Transformer，深層学習（ディープラーニング）

Abstract

Popular Music Accompaniment Generation from Melody Using Transformer Decoder

OKI, Takashi

Recently, music has becoming familiar with the development of music and video streaming services. Also, thanks to the development of software for composing music, non-professionals are not only listening to music but also composing music.

In most cases of composition, melody and accompaniment are created, but the creation of accompaniment is difficult without a high level of expertise and experience. Therefore, many attempts have been made to generate accompaniment by computer. In recent years, the use of the Transformer, a neural network model, has been studied as an effective approach. On the other hand, in recent years, many models that use parts of the transformer structure (e.g., encoder and decoder) have shown excellent results in natural language processing tasks. Also, the decoder of the Transformer can provide excellent results in conditional generation.

We propose a method for generating accompaniment from melody in popular music using the decoder of the Transformer. In order to evaluate the performance of our method, we compared this with the Transformer-based method. This is done by subjective evaluation using a listening test, which has been used in many studies on automatic music generation. The results show that the effectiveness of the decoder of the Transformer for accompaniment generation.

key words Automatic Music Composition, Transformer, Deep Learning

目次

目次

図目次

表目次

第 1 章

はじめに

近年，音楽配信サービスや動画サービスの発展により，音楽が身近なものとなっている．また，作曲用ソフトウェア等も発展しており，作曲に関する高度な専門知識が無い人が，音楽鑑賞だけでなく作曲にも取り組む機会が増えている．

作曲では，多くの場合にメロディと伴奏を制作する必要がある．メロディの制作は，普段から音楽を聴く中で身につけた感性によって行われる場合も多く，音楽的な知識があまり無い人でも充分に制作は可能である．一方，伴奏の制作は，ハーモニーやリズムパターン等を考慮する必要があり，高度な専門知識や経験が無ければ困難である．よって，コンピュータによる伴奏生成に関する多くの試みがなされている．

音楽を自動生成するアプローチとして，ルールに基づく手法や機械学習に基づく手法等が用いられる．ルールに基づく手法では，あらかじめ人間がルールを記述する必要があり，そのルールに従って生成する．一方，機械学習による手法では，既存曲の学習からコンピュータが自動的にルールを見つける．音楽におけるルールは，音楽理論等によって明示されているものもあるが，多くは簡単に言葉や式で表現することのできない暗黙的なものである．よって，機械学習を用いる多くの研究がなされている．

近年の機械学習を用いた音楽自動生成の研究では，ニューラルネットワークによるシーケンスモデルの一つである Transformer[19] を用いることが有効なアプローチとして注目されている．Transformer は，入力シーケンスを解釈するエンコーダーと，出力シーケンスを構成するデコーダーから構成される．Transformer を用いたメロディからの伴奏生成 [9, 17] では，メロディをエンコーダー，過去の伴奏をデコーダーに入力して生成が行われている．

一方，近年では Transformer のエンコーダーもしくはデコーダーのみを用いた多くのモ

デルが提案され，様々な自然言語処理タスクで最先端の結果を残している．入力を伴う生成でも，Transformer のデコーダー（以下，Transformer Decoder と呼ぶ）を用いて優れた性能を出している [3]．この結果から，シーケンスを扱うタスクにおいて，Transformer より，その一部の構造を用いる方が優れた結果を出せることが考えられる．よって，メロディからの伴奏生成においても，Transformer Decoder を用いることで優れた結果を出す可能性が考えられるが，現在その研究はなされていない．

そこで，本研究では Transformer Decoder のみを用いた生成手法を実装し，生成された楽曲の品質が Transformer を用いた場合よりも優れている可能性を発見した．この手法の有効性を示すために主観的評価による比較実験を行った．

本論文は，本章を含めて 7 章からなる．第 2 章では，背景技術として MIDI やニューラルネットワークについて述べる．第 3 章では，音楽情報を機械学習で用いるためのデータ表現について述べる．第 4 章では，伴奏の生成モデルについて述べる．第 5 章では，比較実験のためのセットアップとして，使用するデータセットや生成モデルの設定について述べる．第 6 章では，比較実験の方法と結果について述べる．第 7 章で，本研究のまとめを述べる．

第 2 章

背景技術

2.1 MIDI（Musical Instrument Digital Interface）

MIDI は，楽曲の演奏情報を電子データとして伝達するための規格である．この規格に沿って作られたデータのことを MIDI データといい，作曲用ソフトウェア等で扱うことができる．MIDI データが送信するのは，音の波形データではなく，以下のような演奏情報である．

- ノートオン（音の始まり）
- ノートオフ（音の終わり）
- ベロシティ（音の強さ）
- ノートナンバー（音の高さ）
- プログラムチェンジ（楽器音）
- テンポ（BPM）
- 拍子

これらの演奏情報を含んだ MIDI データを MIDI 音源（コンピュータで使用するための音源）等に送信することで，実際に音を再生することができる．MIDI データを送信する際は，経路 1 本につき 16 の「チャンネル」が存在する．一般的には，1 つのパートの演奏情報は 1 つのチャンネルを使用して送信される．MIDI 音源側は，受信したいチャンネルを指定することで，どのチャンネルの演奏情報を使用するかを選択できる．MIDI データで扱う演奏情報として，演奏する音の高さをノートナンバーという．ノートナンバーは，半音刻みで，最

も低い音を 0，最も高い音を 127 に割り当てる．音域の広い楽器として 88 鍵盤のグランドピアノがあるが，この楽器の音域はノートナンバー 21〜108 に割り当てられる．ノートナンバーは，これよりも広い音域をカバーしている．

現在，最も普及している MIDI データの保存形式として，SMF（スタンダード MIDI ファイル）がある．SMF は，メーカーごとの電子機器に関係なく使用できる共通のファイルフォーマットであり，拡張子は「.mid」である．SMF を構成するデータの中には，トラックと呼ばれる部分があり，ここに実際の演奏データが格納される．SMF には，Format0, Format1，Format2 の 3 つのフォーマットがある．Format0 は，一つのトラックに 1〜16 チャンネルの MIDI データをまとめて記録する形式である．Format1 は，複数のトラックを持ち，それぞれのトラックに複数のチャンネルの MIDI データを記録する形式である．Format2 は，通常は音楽情報を記録するために用いられず，普及もしていないフォーマットである．一般的には，楽器ごとにトラックを分けた方が編集しやすいため，Format1 が使用されることが多い．

SMF では，演奏される楽器音が各メーカーや機種で異ならないように，プログラム・チェンジというものが定義されている．これは，楽器音の名前を 1〜128 の番号に対応付けたものである．この番号をプログラム・チェンジ番号という．このプログラム・チェンジ番号に対応した楽器音の中にパーカッションは含まれていない．その代わり，SMF の 10 番目のチャンネルがパーカッションのために予約されており，このチャンネルはどのプログラム・チェンジ番号が指定されていてもパーカッションパートとなる．

2.2　ニューラルネットワーク

コンピュータに人間と同じような知能を持たせることを目的として，人工知能の研究が行われている．この研究で重要となるのは，人間の持つ知識をいかにしてコンピュータ上で表現するかである．この方法の一つとして，人間の知識を論理式等の何らかの形で明示的に表現することが考えられる．しかし，人間の知識には暗黙的なものも多く，明示的に表現する

ことは困難である．このため，人間の暗黙的な知識を，明示的に表現するのではなく，コンピュータに学習させる「機械学習」という研究が始まった．

機械学習の研究の中には，人間や生物の脳に注目した分野がある．人間や生物の脳は，多数の神経細胞（ニューロン）が結合部位（シナプス）を介して接触することで，巨大なネットワークを形成している．このネットワークの働きを模倣し，数理的にモデル化したものをニューラルネットワークという．

2.2.1　順伝播型ニューラルネットワーク

最も基本的なニューラルネットワークの一つとして，順伝播型ニューラルネットワークがある．順伝播型ニューラルネットワークは，情報が入力側から出力側に一方向に伝播するニューラルネットワークである．このネットワークを構成するユニットは，複数の入力から 1 つの出力を計算する．図 2.1(a) に示すように，3 つの入力 x_1, x_2, x_3 をユニットが受け取る場合，総入力 u は，

$$u = w_1x_1 + w_2x_2 + w_3x_3 + b$$

となる．これは，各入力にそれぞれ異なる重み w_1, w_2, w_3 を掛けたものと，バイアスと呼ばれる値 b を全て加算したものである．このユニットの出力 z は，u に対する関数 $f(u)$ の値となる．

$$z = f(u)$$

このとき，関数 $f(\cdot)$ のことを活性化関数という．

順伝播型ニューラルネットワークでは，図 2.1(b) に示すように，ユニットが層状に並べられ，情報は左の層から右の層に伝わる．左の層のユニットを $i = 1, ..., I$，右の層のユニットを $j = 1, ..., J$ とすると，右の層の出力が決まるまでの計算は，次のように一般化される．

$$u_j = \sum_{i=1}^{I} w_{ji}x_i + b_j$$
$$z_j = f(u_j)$$

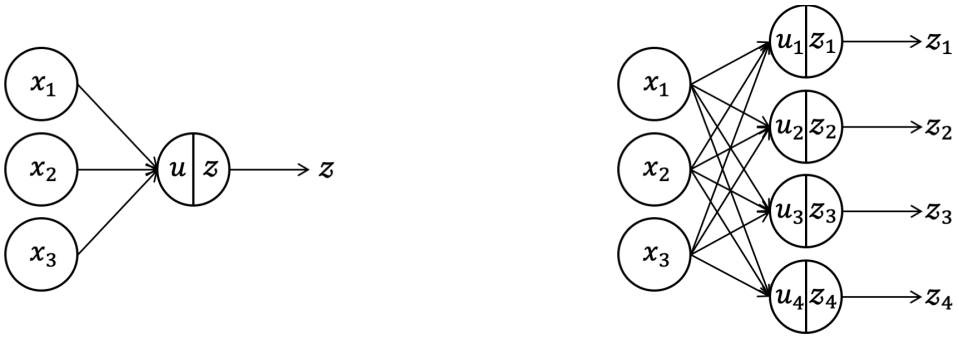
また，これは，ベクトルと行列を用いて，

$$\mathbf{u} = \mathbf{W}\mathbf{x} + \mathbf{b}$$
$$\mathbf{z} = \mathbf{f}(\mathbf{u})$$

と表現できる．ただし，各ベクトルと行列は次のように定義する．

$$\mathbf{u} = \begin{bmatrix} u_1 \\ \vdots \\ u_J \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_I \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_J \end{bmatrix}, \quad \mathbf{z} = \begin{bmatrix} z_1 \\ \vdots \\ z_J \end{bmatrix},$$

$$\mathbf{W} = \begin{bmatrix} w_{11} & \dots & w_{1I} \\ \vdots & \ddots & \vdots \\ w_{J1} & \dots & w_{JI} \end{bmatrix}, \quad \mathbf{f}(\mathbf{u}) = \begin{bmatrix} f(u_1) \\ \vdots \\ f(u_J) \end{bmatrix}$$



(a)　　(b)

図 2.1　(a) ユニットの入出力 (b) 複数のユニットを持つネットワーク

また，順伝播型ニューラルネットワークは，図 2.2 に示すように層を追加することもできる．このネットワークにおいて，一番左の層を入力層，一番右の層を出力層，それ以外の層を隠れ層（あるいは中間層）という．各層のユニットの入出力を区別するために，各層を $l = 1, ..., L$，各変数を $\mathbf{u}^{(l)}$ や $\mathbf{z}^{(l)}$ のように表記することにする．この場合，$l = 2$ の層のユニットの出力は次のようになる．

$$\mathbf{u}^{(2)} = \mathbf{W}^{(2)}\mathbf{x} + \mathbf{b}^{(2)}$$
$$\mathbf{z}^{(2)} = \mathbf{f}(\mathbf{u}^{(2)})$$

また，$l = 3$ の層のユニットの出力は，$\mathbf{x}$ を $\mathbf{z}^{(2)}$ に置き換えて，

$$\mathbf{u}^{(3)} = \mathbf{W}^{(3)}\mathbf{z}^{(2)} + \mathbf{b}^{(3)}$$
$$\mathbf{z}^{(3)} = \mathbf{f}(\mathbf{u}^{(3)})$$

となる．

以上の計算を任意の層数 L のネットワークに一般化すると次のようになる．

$$\mathbf{u}^{(l+1)} = \mathbf{W}^{(l+1)}\mathbf{z}^{(l)} + \mathbf{b}^{(l+1)}$$
$$\mathbf{z}^{(l+1)} = \mathbf{f}(\mathbf{u}^{(l+1)})$$

順伝播型ニューラルネットワークは，左の層から順番に出力が計算されるので，層 $l = 1$ のユニットの出力を $\mathbf{z}^{(1)} = \mathbf{x}$ とすると，各層の出力 $\mathbf{z}^{(2)}, ..., \mathbf{z}^{(L)}$ は，この順番に決定される．このネットワークの最終的な出力を $\mathbf{y}$ とすると，これは，

$$\mathbf{y} \equiv \mathbf{z}^{(L)}$$

となる．なお，各ユニットの持つ活性化関数は，各層で異なるものを使用することができる．

以降，各層の結合重み $\mathbf{W}^{(l)}(l = 2, ..., L)$ とユニットのバイアス $\mathbf{b}^{(l)}(l = 2, ..., L)$ をすべてまとめたものを $\mathbf{w}$ と表す．

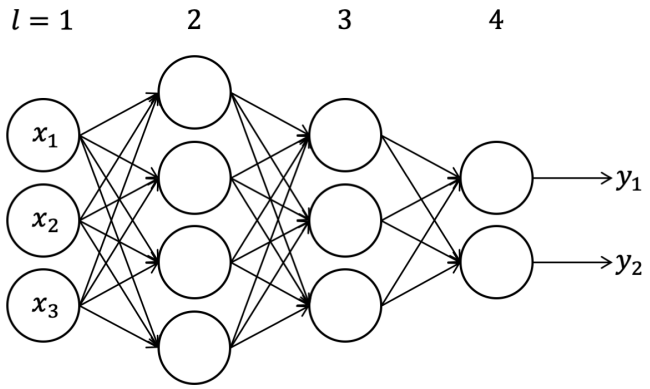


図 2.2　3 層のネットワーク

2.2.2　活性化関数

通常，ユニットが持つ活性化関数には非線形関数が使用される．古くからよく使われるものとしてシグモイド関数（ロジスティック関数）がある．

$$f(u) = \frac{1}{1 + e^{-u}}$$

また，この関数の代わりに，類似した関数である双曲線正接関数を使うこともある．

$$f(u) = \tanh(u)$$

これらの他にも，クラス分類の問題では出力層にソフトマックス関数がよく使われる．クラス分類とは，入力 $\mathbf{x}$ を有限個のクラスに分類する問題のことである．この場合，ネットワークの出力層には分類したいクラス数 K と同数のユニットを並べる．これらのユニットへの入力を $u_j (j = 1, ..., K)$ とすると．ソフトマックス関数は，

$$p_j = \frac{e^{u_j}}{\sum_{k=1}^{K} e^{u_k}}$$

を計算する．そして，この p_j が最大値をとるユニットのインデックス $\hat{j} = \mathrm{argmax}_j \, p_j$ を推定クラスとする．

2.2.3　損失関数

順伝播型ニューラルネットワークを学習させる方法の一つに教師あり学習がある．教師あり学習では，入力 $\mathbf{x}$ と正解 $\mathbf{d}$ のペアの集合 $\{(\mathbf{x}_1, \mathbf{d}_1), ..., (\mathbf{x}_N, \mathbf{d}_N)\}$ を訓練データとして，ネットワークがどの入力に対しても正解に近い出力をするように，各層の結合重み $\mathbf{w}$ を修正する．具体的には，訓練データの入力ごとに，ネットワークの出力が正解と一致しない場合に重みを修正することを繰り返す．これには，ネットワークの出力 $\mathbf{y}$ と正解 $\mathbf{d}$ の近さの尺度が必要となる．この尺度のことを損失関数（あるいは誤差関数）という．

クラス分類を行う場合，損失関数として交差エントロピーが使用される．2.2.2 小節より，K 個のクラスに分類するネットワークは，対応するクラスに対する確率 $p_1, ..., p_K$ を出力するものであった．交差エントロピーは，訓練データの n 番目の組に対して，目標となる出力

を $d_{n1}, ..., d_{nK}$，実際の出力を $p_{n1}, ..., p_{nK}$ とすると，

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{j=1}^{K} d_{nj} \log p_{nj}$$

を計算する．ただし，目標出力 $d_{n1}, ..., d_{nK}$ は，正解クラス j に対応する d_{nj} のみ 1 をとり，それ以外は 0 をとるようにする．

2.2.4　勾配降下法

順伝播型ネットワークの学習では，訓練データ $\{(\mathbf{x}_1, \mathbf{d}_1), ..., (\mathbf{x}_N, \mathbf{d}_N)\}$ が与えられたときに，損失関数 $E(\mathbf{w})$ の値を $\mathbf{w}$ について最小にする最適化を行う．この最適化には勾配降下法がよく利用される．勾配降下法では，現在の重み $\mathbf{w}^{(t)}$ における勾配 $\nabla E \equiv \frac{\partial E(\mathbf{w}^{(t)})}{\partial \mathbf{w}^{(t)}}$ を求め，

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \epsilon \nabla E \qquad (\epsilon > 0)$$

のようにして重みを更新していく．

以上は，訓練データ全体に対して計算される損失関数 $E(\mathbf{w})$ を用いた最適化である．このように，1 回の重みの更新に，訓練データ全体を用いた学習のことをバッチ学習という．これに対して，訓練データを少数のデータの集合に分割し，その単位で重みを更新するミニバッチ学習がある．

第 3 章

音楽情報のデータ表現

シーケンスモデルを用いたアプローチでは，音楽情報を “音を開始するイベント” や “音を終了するイベント” 等のイベントで構成されるシーケンスに変換する．そして，モデルの学習によって，このイベントシーケンスの確率分布モデルを作成する．このアプローチにおいて，音楽情報のデータ表現は生成される楽曲の品質に影響を与える重要な要素である．

本章では，まず Transformer を用いた音楽自動生成で用いられる 2 種類のデータ表現を紹介する．1 つ目は，MIDI の演奏データに基づいた従来のデータ表現であり，クラシックピアノ曲の生成等に用いられる．2 つ目は，ポピュラー音楽の生成のために，規則的なリズムパターンを捉えられるように改良された REMI[10] というデータ表現である．最後に，これらを基にしたメロディからの伴奏生成のためのデータ表現について述べる．

3.1 既存のデータ表現 1：MIDI-like

音楽自動生成に関する多くの先行研究 [18, 4, 12, 9] では，MIDI が演奏データを扱うために用いるノートオン，ノートオフインベントに基づいた MIDI-like なデータ表現を用いている．これは，コンピュータで音楽を制作する際に一般的に用いられる MIDI を踏襲しているため，様々な音楽を表現することができる．[18, 9] は，このデータ表現を用いてクラシックピアノ曲の自動生成を行っている．

このデータ表現では，以下のトークンを導入している．

- NOTE_ON<p> — p 番目のピッチの音の演奏を開始する．p は 0〜127 の MIDI ノートナンバーで指定する．

- NOTE_OFF<p> — p 番目のピッチで演奏されている音を終了する．p は 0〜127 の MIDI ノートナンバーで指定する．
- SET_VELOCITY<v> — これ以降のトークンによって演奏される音のベロシティを指定する．v は 32 段階に量子化された 0〜127 のベロシティ（音の強弱）を表す．
- TIME_SHIFT<t> — 時間の経過を表現するために，上記のトークンによって表される演奏イベント間のタイムシフト量を指定する．t は 10 ミリ秒から 1 秒までの範囲で指定する．
  - 例：NOTE_ON<60>，TIME_SHIFT<500>，NOTE_ON<64>は，60 番目の MIDI ノートナンバーの音を立ち上げた 500 ミリ秒後に，64 番目の MIDI ノートナンバーの音を立ち上げる．

図 3.1 に楽譜から MIDI-like なデータ表現への変換例を示す．

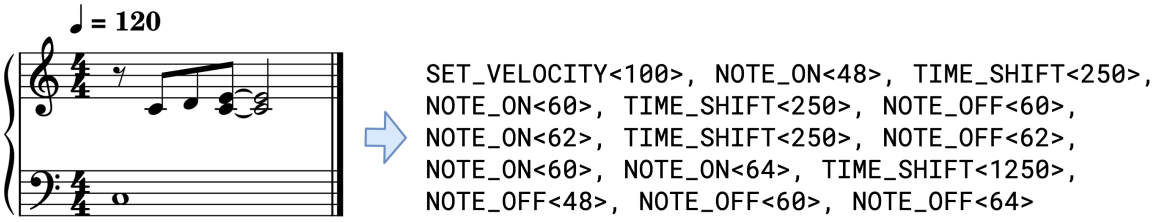


図 3.1　楽譜から MIDI-like なデータ表現への変換例

3.2　既存のデータ表現 2：REMI

3.1 節の MIDI-like なデータ表現をポピュラー音楽で用いる場合，生成した曲の品質に関する以下の問題点がある．

- 多くのポピュラー音楽は，テンポやビートに基づく規則的なリズムパターンで構成されている．しかし，MIDI-like な表現にはテンポやビート等の情報は含まれておらず，時

間の経過をミリ秒単位の TIME_SHIFT トークンのみで扱っているため，シーケンスモデルが規則的なリズムパターンを学習することは困難である．

- 1 つの音符の長さを表現するために TIME_SHIFT トークンによるタイムシフト量を累積しなければならない場合がある．図 3.1 の例でも，1 拍目から始まるドの音符の長さを表現するためにタイムシフト量の累積が起きている．
- TIME_SHIFT トークンを用いた時間表現は誤差が累積するため，安定したビートを維持することが困難である．
- NOTE_ON トークンとそれに対応する NOTE_OFF トークンがペアで出現しなければならないが，これをシーケンスモデルが学習することは困難である．これは，図 3.1 でも分かるように，ペアとなる 2 つのトークン間に TIME_SHIFT トークン等の 1 つ以上のトークンが存在し，間隔が空いてしまうためである．よって，生成された楽曲にペアとなる NOTE_OFF トークンが出現しない問題が多く発生してしまう．

Huang ら [10] は，これらの問題を解決するために REMI と呼ばれるデータ表現を提案している．REMI では，人間が作曲を行う際に考慮される小節やビート，テンポ等の情報をシーケンスに含めることで，ポピュラー音楽における規則的なリズムパターンを容易に学習することができるようにしている．

このデータ表現では，以下のトークンを導入している [*1]．

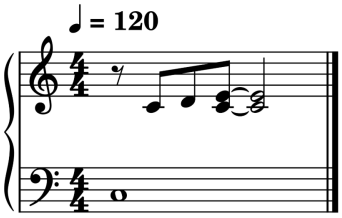
- NOTE_ON<p> — MIDI-like なデータ表現と同様．
- NOTE_DURATION<t> — NOTE_ON トークンの直後に現れる．この NOTE_ON トークンによって立ち上がる音を長さ t だけ持続することを表す．
- POSITION<t/T> — NOTE_ON トークンの直前に現れる．この NOTE_ON トークンによって音を立ち上げるタイミングが，小節内での位置 t（$1 \leq t \leq T$）であることを表す．ここで，T は 1 小節の時間分解能を示す．
- BAR — 小節の始まりを表す．
- TEMPO_CLASS，TEMPO_VALUE — テンポを表す．

[*1] 比較がしやすいように，トークンの表記は 3.1 節のデータ表現に揃えている．

- CHORD — コードを表す.

REMI は，NOTE_OFF トークンの代わりに NOTE_DURATION トークンを用いることで，1 つの音符の長さを 1 つのトークンで表現することができ，NOTE_ON トークンの直後に出現することから，モデルがペアの出現を学習することが容易となる．また，TIME_SHIFT トークンの代わりに POSITION トークンと BAR トークンを用いることで，テンポと拍子に基づいた音楽イベントの時間配置が可能となる.

図 3.2 に楽譜から REMI への変換例を示す.




```
BAR,
POSITION<1/32>, CHORD<Cmajor>,
POSITION<1/32>, TEMPO_CLASS<mid>, TEMPO_VALUE<30>,
POSITION<1/32>, NOTE_VELOCITY<25>,
NOTE_ON<48>, NOTE_DURATION<32>,
POSITION<5/32>, NOTE_VELOCITY<25>,
NOTE_ON<60>, NOTE_DURATION<4>,
POSITION<9/32>, TEMPO_CLASS<mid>, TEMPO_VALUE<30>,
POSITION<9/32>, NOTE_VELOCITY<25>,
NOTE_ON<62>, NOTE_DURATION<4>,
POSITION<13/32>, NOTE_VELOCITY<25>,
NOTE_ON<60>, NOTE_DURATION<20>,
POSITION<13/32>, NOTE_VELOCITY<25>,
NOTE_ON<64>, NOTE_DURATION<20>,
POSITION<17/32>, TEMPO_CLASS<mid>, TEMPO_VALUE<30>,
POSITION<25/32>, TEMPO_CLASS<mid>, TEMPO_VALUE<30>,
BAR
```

図 3.2　楽譜から REMI への変換例

3.3　メロディからの伴奏生成のためのデータ表現

これらを踏まえて，メロディからの伴奏生成では，規則的なリズムパターンを捉えるために 3.2 節で紹介した REMI を用いる．しかし，このまま用いるためには問題があるため，以下のような拡張を行う.

- REMI には楽器を区別するための情報は含まれない．そこで，NOTE_ON<Melody:60>, NOTE_ON<Piano:60>のように，NOTE_ON トークンに楽器名を含める．音符の開始や終

了イベントを表すトークンに楽器名を含めることは，複数の楽器を用いた楽曲を生成する MuseNet[14] でも用いられている．

- REMI は，MIDI-like なデータ表現と比べてシーケンス長が長くなる傾向がある．複数の楽器で構成される伴奏に用いる場合，一般的な GPU 環境ではメモリ不足を引き起こす程のシーケンス長になってしまう．よって，この問題を解決するために以下の拡張を行う．
  - `POSITION` トークンは，音楽イベントが発生する度に出現する．よって，[17] でも行われているように，各時刻の最初に 1 つの `POSITION` トークンを付与するように変更する．これにより，同時に多くの音楽イベントが発生する伴奏生成においてシーケンス長を節約することができる．
  - 使用するトークンを，音楽フレーズが生成可能な必要最低限のものに限定する．具体的には，テンポやコードを扱うトークンと `VELOCITY` トークンを除外する．

図 3.3 に楽譜から伴奏生成のためのデータ表現への変換例を示す．

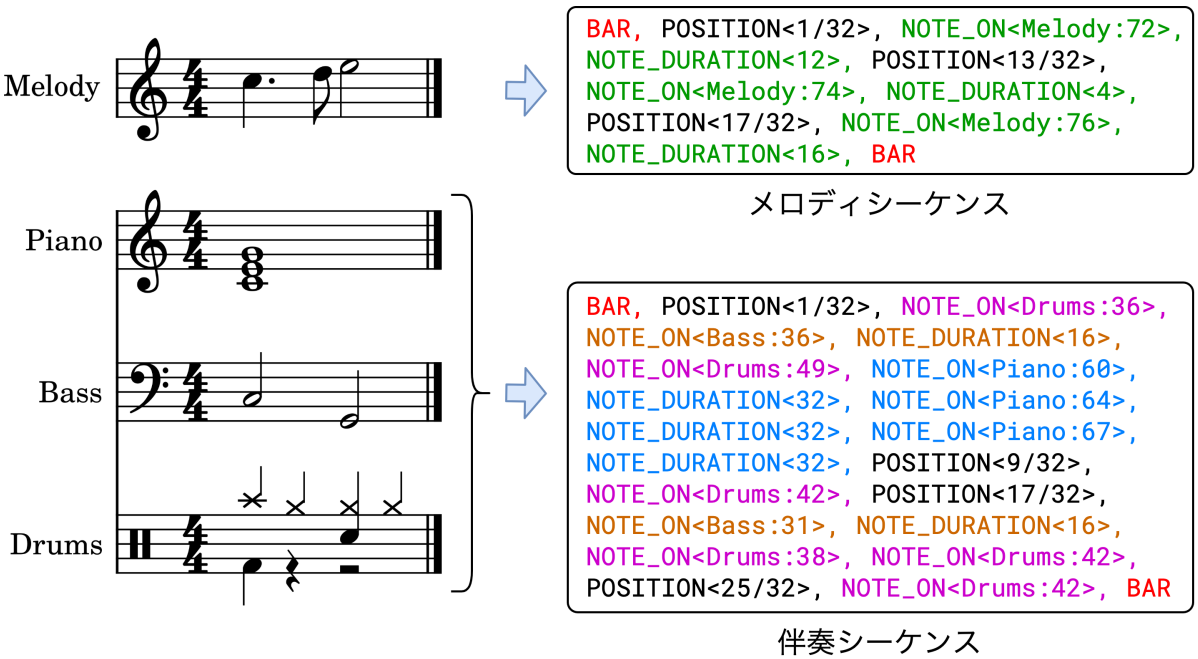


図 3.3　楽譜から伴奏生成のためのデータ表現への変換例

第 4 章

モデル

本章では，メロディからの伴奏生成手法として，Transformer Decoder を用いた手法を提案する．4.1 節では，従来の Transformer を用いた伴奏の生成手法について述べる．4.2 節では，Transformer Decoder を用いた伴奏の生成手法について述べる．

4.1 Transformer を用いた生成手法

4.1.1 Transformer の構造

Transformer は，エンコーダー・デコーダー構造のニューラルシーケンスモデルの 1 つである．図 4.1 に Transformer の全体像を示す（図中の左側にエンコーダー，右側にデコーダーが構成される）．エンコーダーは，入力シーケンス $X = (x_1, \ldots, x_n)$ を解釈し，最終的な中間状態をデコーダーに渡す．デコーダーは，エンコーダーから渡された情報を基に出力シーケンス $Y = (y_1, \ldots, y_m)$ を出力する．出力シーケンス Y に含まれるトークンは，前から順番に 1 つずつ生成される．次のトークンの出力には，エンコーダーの中間状態の他に，過去のステップで生成されたトークンが用いられる（例えば，時刻 t のトークン y_t の出力には $(y_1, \ldots, y_{t-1})$ が用いられる）．図 4.2 に Transformer を用いたシーケンス生成の様子を示す．また，Transformer の学習は，最初の時刻から y_1，y_2，$\ldots$ と逐次的に行われるのではなく，図 4.3 に示す形で，すべての時刻のトークンについて並列で行われる．

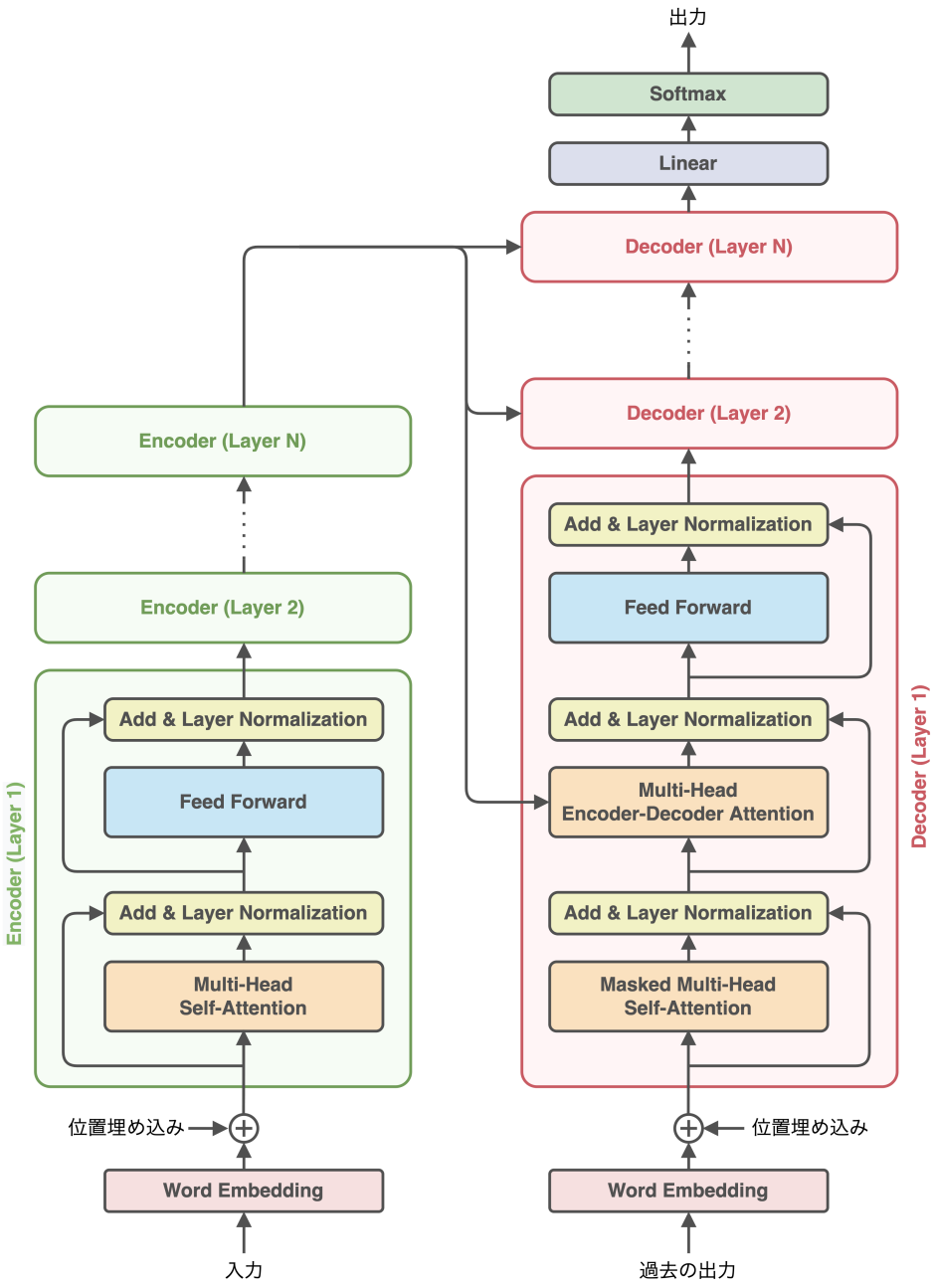


図 4.1　Transformer の全体像

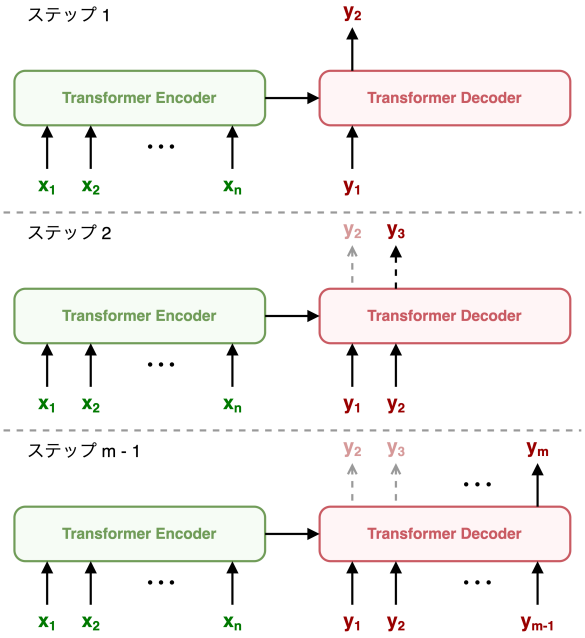


図 4.2　Transformer の生成の様子

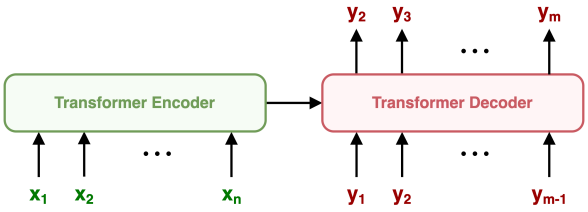


図 4.3　Transformer の学習の様子

以下で，Transformer の構造の詳細を述べる．

エンコーダー及びデコーダーブロック

Transformer のエンコーダー及びデコーダーは，それぞれ同じ構造を持つ複数のブロックが積み重なっている．エンコーダーブロックは，マルチヘッド自己注意（Multi-Head Self-Attention）層と全結合（Feed Forward）層で構成される．デコーダーブロックは，マスクされたマルチヘッド自己注意（Masked Multi-Head Self-Attention）層，マルチヘッドエンコーダー・デコーダー注意（Multi-Head Encoder-Decoder Attention）層及び全結合層で構成される．また，これらの層の前後で残差接続 [7] を行い，その後 Layer Normalization[2] を行う．

注意層

Transformer の構造の中で最も重要となるのが，注意（Attention）と呼ばれる仕組みである．注意は，シーケンス X に含まれるトークン x_i に対して，シーケンス Y に含まれる各トークン $y_1, \ldots, y_m$ がどれくらい関係しているかを計算することで，トークン間の依存関係を捉えるために用いられる．注意の入力には，シーケンス X に基づく

query と，シーケンス Y に基づく key，value が用いられる．注意は，query を入力すると，key と value を用いて各トークンの依存関係に関する情報を引き出す．この処理は，以下のような計算によって行われる．

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

ここで，Q, K, V はそれぞれ注意層への入力，d_k は K の次元数を表す．Q は query，K は key，V は value として用いられる．

ただし，Transformer では，1 つの注意層において 1 つの query，key，value の組を用いるこの処理は行われていない．その代わりに，異なる複数の query，key，value の組を用いるマルチヘッド注意（Multi-Head Attention）と呼ばれる処理が有効であるとして用いられている．マルチヘッド注意では，ヘッドごとに query，key，value の組を作成し，それぞれに対して関数 Attention(·) を計算する．これらの結果は連結されて，最終的な値となる．この処理は，以下のような計算によって行われる．

$$\text{MultiHeadAttention}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$$

$$\text{where} \quad \text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

ここで，Concat(·) は各ヘッド head_i を連結する関数を表す．また，Q, K, V はそれぞれ注意層への入力，W^O, W_i^Q, W_i^K, W_i^V はそれぞれ重みを表す．各ヘッドにおいて，QW_i^Q は query，KW_i^K は key，VW_i^V は value として用いられる．

Transformer は，上記のマルチヘッド注意を以下の 2 種類の方法で使用している．

- エンコーダーとデコーダーの両方で自己注意（Self-Attention）が使用されている．自己注意は，1 つのシーケンス内での各トークンの依存関係を捉えるために，Q, K, V に同じ値を用いる．実際には，自己注意が各エンコーダー・デコーダーブロックの最初に置かれ，1 つ前のブロックの出力が Q, K, V として用いられる．また，デコーダーで用いる自己注意では，デコーダーの自己回帰性を保つために，計算中に未来のトークンの情報を表現した部分をマスクする処理を行っている．
- デコーダーで Encoder-Decoder Attention が使用されている．この注意では，Q

に 1 つ前のデコーダーブロックの出力，K, V にエンコーダーの最終的な出力が用いられる．これによって，デコーダー内で，エンコーダーの入力シーケンスへの依存関係を扱うことができる．

全結合層

エンコーダ及びデコーダブロックの全結合層は，2 つの全結合ネットワークで構成される．この処理は，以下のような計算によって行われる．

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

ここで，W_1, b_1 はそれぞれ 1 つ目の全結合ネットワークの重みとバイアス，W_2, b_2 はそれぞれ 2 つ目の全結合ネットワークの重みとバイアスを表す．ただし，これらのパラメータは，x と $\mathrm{FFN}(x)$ の次元数が同じになるように設定する．

入力の埋め込み

Transformer では，単語埋め込み（Word Embedding）を用いて，入力シーケンスに含まれる各トークンをベクトルに変換して扱う．単語埋め込みは，似た意味を持つトークンを似たベクトルに変換する方法である．この変換方法は，モデルの学習中に一緒に学習される．また，このままでは Transformer がシーケンスの順序を学習することができない．そのため，単語埋め込みによる変換後，シーケンス内での各トークンの位置情報を付与する位置埋め込み（Positional Embedding）を行う必要がある．本研究では，近年多くのモデルで用いられる単語埋め込みによる方法を用いる．この方法は，単語埋め込みを用いて連番のシーケンス $(1, 2, 3, \dots)$ を変換し，入力の埋め込みに加算する．シーケンス $X = (x_1, \dots, x_n)$ の埋め込みは，以下の計算によって行われる．

$$\mathrm{Embedding}(X) = \mathrm{WordEmbedding}(X) + \mathrm{PositionalEmbedding}(X)$$

$$\mathrm{PositionalEmbedding}(X) = \mathrm{WordEmbedding}(1, \dots, n)$$

ここで，$\mathrm{WordEmbedding}(\cdot)$ は単語埋め込みを行う関数を表す．

4.1.2 メロディからの伴奏生成

メロディからの伴奏生成では，メロディと伴奏をそれぞれ 3.3 節のデータ表現へ変換し，メロディシーケンス $X = (\texttt{BOS}, x_1, \ldots, x_n, \texttt{EOS})$ と伴奏シーケンス $Y = (\texttt{BOS}, y_1, \ldots, y_m, \texttt{EOS})$ を作成する．ここで，BOS トークンと EOS トークンは，モデルがシーケンスの先頭と末尾を理解するために用いる．モデルの学習は，メロディシーケンス X から伴奏シーケンス Y を出力するように行う．伴奏シーケンスの生成は，エンコーダーにメロディシーケンス X，デコーダーに BOS トークンを入力して，それに続くトークンを 1 つずつ出力することで行う．

4.2 Transformer Decoder を用いた生成手法

Transformer Decoder は，図 4.1 に示す Transformer のデコーダーから，エンコーダーからの入力を受け取る Encoder-Decoder Attention を取り除いたものである．図 4.4 に Transformer Decoder の全体像を示す．Transformer Decoder を用いて入力を伴う生成を行う場合は，入力シーケンス $(x_1, \ldots, x_n)$ をモデルに入力し，これに続くトークンを，$y_1, y_2 \ldots$ という流れで 1 つずつ生成する．そして，$(y_1, \ldots, y_m)$ を出力シーケンスとして抽出する．図 4.5 に Transformer Decoder を用いたシーケンス生成の様子を示す．

メロディからの伴奏生成では，4.1 節と同様のデータ表現へ変換し，メロディシーケンス $X = (x_1, \ldots, x_n)$ と伴奏シーケンス $Y = (y_1, \ldots, y_m)$ を作成する．その後，シーケンス X，Y を連結したシーケンス $Z = (\texttt{BOS}, x_1, \ldots, x_n, \texttt{SEP}, y_1, \ldots, y_m, \texttt{EOS})$ を作成する．ここで，BOS 及び EOS は 4.1 節と同様の意図で挿入する．SEP トークンは，シーケンス X，Y の境界を生成モデルが理解するために挿入する．モデルの学習は，このシーケンス Z を用いて，図 4.6 に示すように行う．伴奏シーケンスの生成は，シーケンス Z の SEP トークンまでを入力し，それに続くトークンを 1 つずつ出力することで行う．

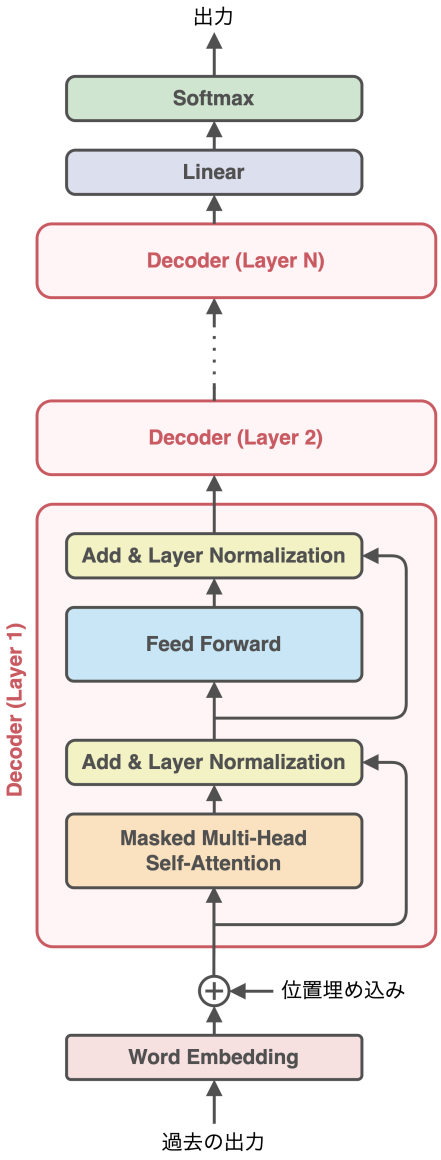


図 4.4　Transformer Decoder の全体像

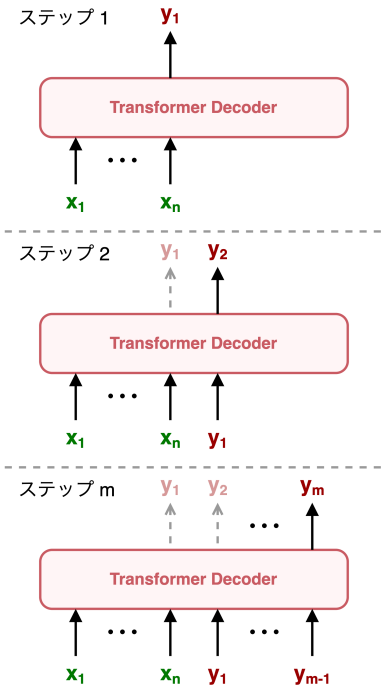


図 4.5　Transformer Decoder の入力を伴う生成の様子

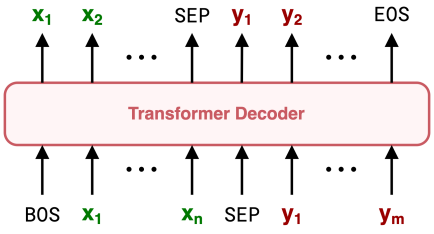


図 4.6　メロディからの伴奏生成における Transformer Decoder の学習の様子

第 5 章

比較実験のセットアップ

5.1 実験環境

生成モデルの学習に使用する GPU 環境は以下の通りである．

- GeForce GTX 1080 Ti
- NVIDIA driver 450.102.04
- CUDA 11.0

また，使用するプログラムの実装環境は以下の通りである．

- Python 3.8.5
- PyTorch 1.7.1[13] — 機械学習ライブラリ
- TensorFlow 2.4.0[1] — 機械学習ライブラリ．学習を高速化するために，訓練データのシリアライズやそのデータの読み込み等のデータに関するユーティリティのみを使用
- Transformers 3.1.0[20] — 最先端の自然言語処理モデルを実装したライブラリ
- torch-optimizer 0.1.0[*1] — PyTorch 用の最適化アルゴリズムの実装の集合．PyTorch には含まれていない RAdam[11] を用いるために使用
- pretty_midi 0.2.9[16] — MIDI データを処理するためのライブラリ

[*1] https://github.com/jettify/pytorch-optimizer

5.2　データセット

モデルの学習および比較実験には，The Lakh MIDI Dataset v0.1 の Clean MIDI subset[15] を使用する．これは，多くのポピュラー音楽を含む，約 17000 個の MIDI ファイルからなるデータセットである．

このデータセットには，インターネット上から収集されたあらゆる MIDI データが含まれているので，モデルを学習させるためには前処理を行う必要がある．ここでは，[5, 17] を基にして以下の順に前処理を行う．

メロディトラックの抽出

使用する MIDI データはインターネット上から収集されているため，あらゆるユーザによって作成されている．そのため，単に使用している楽器名や MIDI トラック名でメロディトラックかどうかを判断することは困難である．よって，MIDI トラックをメロディトラックやコードトラック等に分類する MIDI Miner[6] を使用してメロディトラックを抽出する．MIDI Miner でメロディトラックを検出できなかった場合は，フルートを楽器として使用している（プログラムチェンジ番号が 73 である）最初のトラックをメロディトラックとする．これは，メロディを演奏している楽器としてフルートが使用されているデータが多く見られたからである．フルートを使用するトラックが存在しないデータは除外する．

伴奏トラックの抽出

ピアノ，ベース及びドラムトラックを伴奏トラックとして抽出する．それぞれのトラックの抽出は以下の通りに行う．

- ピアノトラックの抽出 — 鍵盤楽器が使用されているトラックをピアノトラックとして抽出する．鍵盤楽器で演奏されるトラックかどうかは，プログラムチェンジ番号が 0〜8 または 16〜23 かどうかで判断する．プログラムチェンジ番号の 0〜7 には，アコースティックピアノやエレクトリックピアノ等のピアノ系統の楽器が並んでいる．プログラムチェンジ番号の 8 は，チェレスタという鍵盤楽器である．プロ

グラムチェンジ番号の 16〜23 には，オルガン系統の楽器が並んでいる．

- ベーストラックの抽出 — プログラムチェンジ番号のうち，ベース系統の楽器が定められているのは 32〜39 である．よって，これに該当するトラックをベーストラックとして抽出する．
- ドラムトラックの抽出 — MIDI ファイルでは，10 番のチャンネルがパーカッションの演奏用に定められている．よって，このチャンネルを使用するトラックをドラムトラックとして抽出する．

データのクオンタイズ

音符のタイミングのばらつきを補正するためにクオンタイズを行う．[10] は，16 分音符単位でクオンタイズを行っており，[17] は，32 分音符単位でクオンタイズを行っている．また，これ以上細かい単位でクオンタイズを行うと，モデルの学習が困難となることがいわれている．本実験で用いるデータセットには 16 分音符よりも細かい音符が多くみられたので，32 分音符でクオンタイズを行う．

データの分割

シーケンスモデルで扱うための現実的なシーケンス長になるように，メロディが演奏されている範囲を 8 小節ずつ切り取る．その方法として，曲の先頭から 1 小節ずつ順番に見ていき，メロディが最初に演奏する音符がある小節から，2 小節間メロディが演奏されないところまでを，メロディが演奏されている範囲とする．この時点で曲の最後まで走査していない場合，続きの小節からメロディが演奏されている新たな範囲を探す．その後，それぞれの範囲において，先頭から順番に 8 小節ずつ分割し，異なる MIDI データとして出力する．

データの圧縮

ピアノトラック及びドラムトラックが複数ある場合は，それらのトラックを結合し 1 つのトラックにする．また，ベーストラックが複数ある場合は，音数が一番多いトラックのみを残す．

データのフィルタリング

以下の条件を満たすデータを除去する．

- テンポが一定ではないデータ
- 拍子が一定ではないもしくは 4/4 拍子ではないデータ
- いずれかのトラックが存在しないデータ
- いずれかのトラックの音数が 15 以下であるデータ
- 長さ 4 小節以下のデータ

これらの前処理を行った結果，19709 個の MIDI データが得られた．この中から無作為に 98%のデータをモデルの学習用，1%を評価実験用として抽出し，残りは学習や生成が上手くできているかどうかを確認するために使用した．

5.3　生成モデルの設定

この実験では，Transformers ライブラリで実装されているモデルを使用する．

Transformer の実装として，EncoderDecoderModel を使用する．このモデルは，Transformers ライブラリで実装されている任意のエンコーダーモデル及びデコーダーモデルを指定し，Transformer のようなエンコーダー・デコーダー構造のモデルを構築することができる．本実験では，エンコーダーモデルとして BERT の実装である BertModel，デコーダーモデルとして GPT-2 の実装である GPT2LMHeadModel を用いる．BERT 及び GPT-2 は，それぞれが Transformer のエンコーダー，デコーダーと同様の構造をしている．よって，この 2 つのモデルを組み合わせることで，Transformer を構築することができる．また，Transformer Decoder の実装として，GPT-2 の実装である GPT2LMHeadModel を使用する．

これらのモデルの構成として，エンコーダー及びデコーダーブロックの数を 4，注意層の次元数を 256，注意ヘッドの数を 4，埋め込みの次元数を 256，全結合層の中間の次元数を 1024，全結合層の活性化関数を GELU[8]，Transformer の扱うシーケンスの最大長を 512，

Transformer Decoder の扱うシーケンスの最大長を 1024（メロディ及び伴奏シーケンスを連結して扱うため，Transformer の 2 倍のシーケンス長が必要である）とした．これらの構成は，本実験環境でメモリ不足が起こらない範囲で，パラメータの数がある程度等しくなるように決定した．

モデルの学習は，本実験における GPU 環境において，バッチサイズ 16 で行う．また，最適化アルゴリズムとして，RAdam[11] を $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$ の設定で使用する．学習率を 10^{-3} とし，5 万ステップの学習を行う．5 万ステップでは損失関数の値は収束しないが，これ以上の学習を行うと，生成された伴奏の品質が下がるように感じられた．

また，伴奏シーケンスの生成は先行研究 [9, 10, 17] に倣って確率的サンプリング法を用いて行った．

第 6 章

比較実験

Transformer Decoder を用いた伴奏生成の有効性を示すために主観評価を行う．まず，実験方法を紹介し，その実験結果を報告する．

6.1 実験方法

音楽自動生成モデルの評価方法として，多くの先行研究 [5, 9, 10, 17] でリスニングテストが用いられている．よって，本研究ではリスニングテストによる比較実験を行う．

リスニングテストでは，4.2 節で作成した評価実験用データセットに含まれる MIDI ファイルのメロディトラックを使用する．これらメロディトラックから，評価対象となる 2 つのモデルで伴奏を生成する．生成した伴奏は，MIDI トラックに変換して元のメロディトラックと連結し，テンポを 120，すべての音符のベロシティを 100，メロディ・ピアノ・ベーストラックのそれぞれのプログラムチェンジ番号を 80，0，33 と設定し，MIDI データを作成する．この MIDI データは，pretty_midi ライブラリを用いて WAV ファイルに変換して書き出す．ここで，MIDI データは単なる演奏情報を記録したものである．よって，WAV ファイルに変換するには，MIDI データを波形として書き出すための音源を指定する必要がある．ここでは，その音源としてサウンドフォント形式の音源の 1 つである GeneralUser GS v1.471[*1] を使用する．この音源を選んだ理由は，ベースの音が一般的なリスニング環境でも聴き取りやすく，本実験に適していると考えたからである．図 6.1 に，生成結果から WAV ファイルへの変換の流れを示す．

[*1] http://schristiancollins.com/generaluser.php

リスニングテストでは，同じメロディに対して，評価対象となる 2 つのモデルで生成した音源を 1 セットとし，30 セットを作成した．この 30 セットは被験者全員が評価するために用いられる．被験者は，各セットの音源をランダムな順序で聴き，それぞれの音源に対して以下の質問に回答する．

- ハーモニーが心地よい
- リズムが統一されている
- まとまりがある
- 総合的に好き

なお，評価の前に，ハーモニーがよく分からない場合は不協和音が感じられるかどうかで評価するように説明した．また，回答は以下のような 5 段階のリッカート尺度で行う．

1. 全くそう思わない
2. そう思わない
3. ふつう
4. そう思う
5. 非常にそう思う

6.2　実験結果と考察

リスニングテストには，8 名の被験者から回答があった．被験者は，20 代が 7 名，40 代が 1 名であり，男性が 6 名，女性が 2 名である．表 6.1 は 5 段階尺度を 1〜5 点で点数付けした場合の平均点と Wilcoxon の符号付き順位検定の p 値，表 6.2 はセットごとの勝ち数を示す．表 6.1 からは，従来手法は “リズムが統一されている” 以外の評価軸が 3 点未満であるのに対し，提案手法はすべての評価軸で 3 点を上回っていることがわかる．表 6.2 からは，従来手法と比べて提案手法が “リズムが統一されている” については約 2 倍，それ以外は約 3 倍上回っていることが分かる．ただし，引き分けの数は少なくはなかった．また，

図 6.2 にメロディごとの平均点，図 6.3 にメロディごとの勝ち数を示す．これらの結果から，Transformer Decoder を用いて生成された伴奏の方が，より好まれていることが分かる．

2 つのモデル間にこのような差が生じた要因として，ハーモニーの心地良さの違いが考えられる．いくつかの生成結果を聴いてみると，従来手法で生成された伴奏には，メロディと合わせて聴いた場合に不協和音が生じていると感じるものが多かった．このことから，従来手法はメロディの情報を比較的上手く扱えていないと考えられる．不協和音が多いと，全体的にその楽曲の評価が低くなる可能性があるので，不協和音が “ハーモニーが心地良いか” だけでなく，それ以外の評価軸での差にも影響を与えていることが考えられる．また，2 つのモデル間に差が生じた他の要因として，従来手法を用いた場合，メロディが演奏中であるのに伴奏が途中で終了することが多かった．この問題によって，“まとまりがある” という評価軸で差が生じたと考えられる．

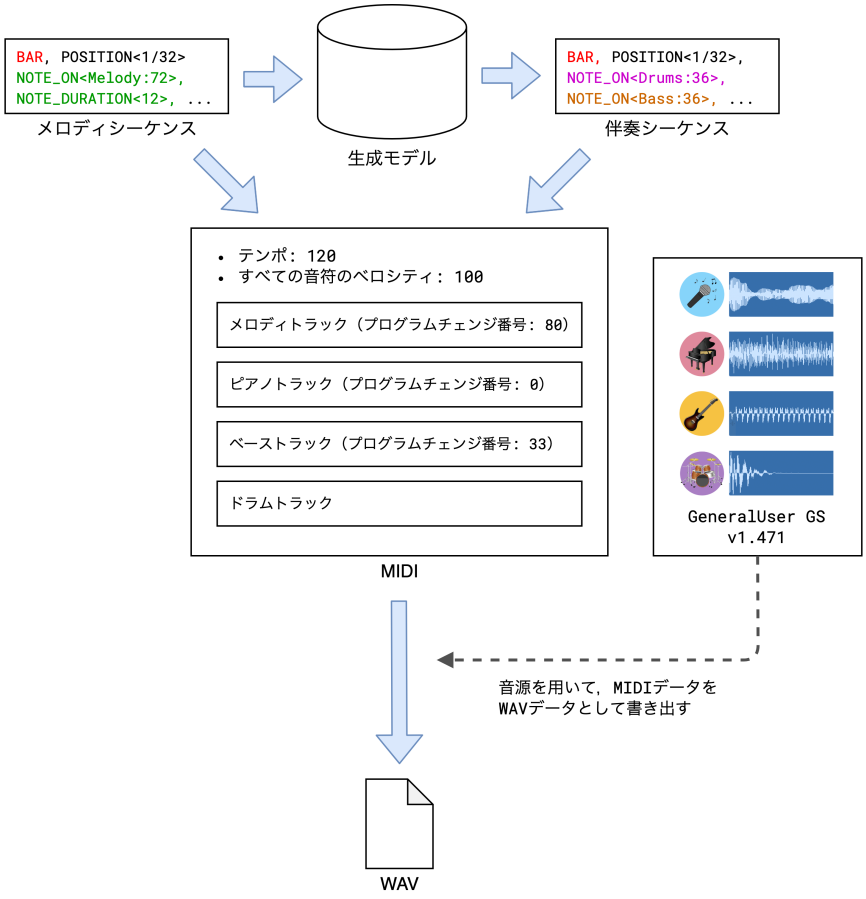


図 6.1　生成結果から WAV ファイルへの変換の流れ

表 6.1　5 段階尺度を 1〜5 点で点数付けした場合の平均点と Wilcoxon の符号付き順位検定の p 値．左の列が Transformer Decoder の平均点，中央の列が Transformer の平均点，右の列が p 値を示す．

	TD scores（提案手法）	T scores	p-value
ハーモニーが心地よい	**3.42**	2.73	5.38e−13
リズムが統一されている	**3.62**	3.14	1.58e−7
まとまりがある	**3.4**	2.66	3.78e−11
総合的に好き	**3.24**	2.56	6.79e−14

表 6.2　セットごとの勝ち数（合計セット数 = 240）．左の列が Transformer Decoder の方が評価が良かったセット数，右の列が Transformer の方が評価が良かったセット数を示す．

	TD wins（提案手法）	T wins	ties
ハーモニーが心地よい	**127**	38	75
リズムが統一されている	**111**	52	77
まとまりがある	**124**	38	78
総合的に好き	**117**	36	87

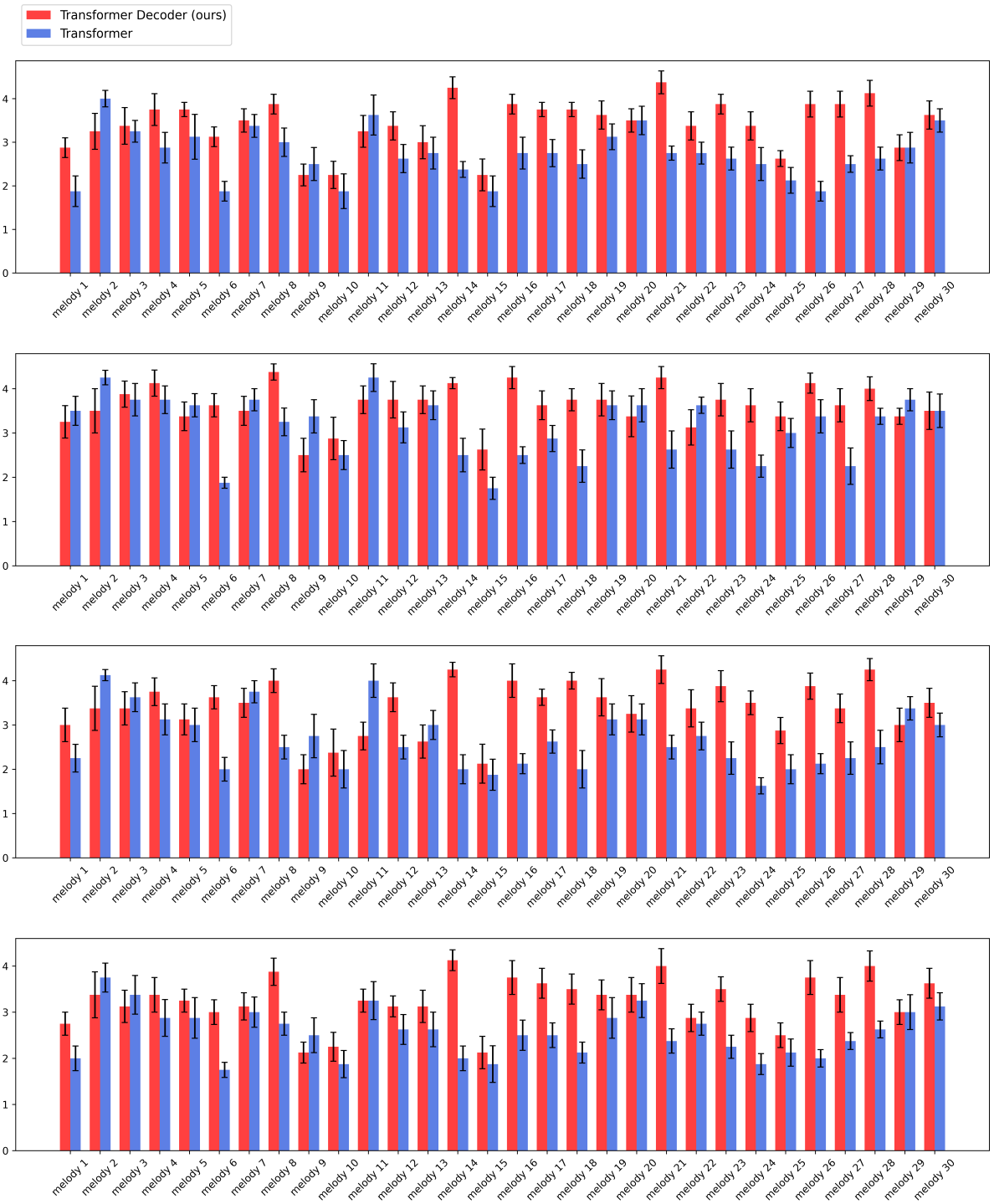


図 6.2　5 段階尺度を 1〜5 点で点数付けした場合のメロディごとの平均点（各メロディのセット数=8）．エラーバーは標準誤差を示す．上から順番に “ハーモニーが心地よい”，“リズムが統一されている”，“まとまりがある”，“総合的に好き” に関するグラフである．

図 6.3　メロディごとの勝ち数（各メロディのセット数=8）．上から順番に “ハーモニーが心地よい”，“リズムが統一されている”，“まとまりがある”，“総合的に好き” に関するグラフである．

第 7 章

まとめ

Transformer Decoder と独自に拡張した REMI を用いたポピュラー音楽におけるメロディからの伴奏の自動生成を提案した．この手法の性能を測るために，リスニングテストによる主観的評価によって，Transformer を用いた従来の生成手法との比較を行った．その結果，Transformer Decoder を用いることで優れた品質の伴奏の生成が可能であることが示された．

今後の課題として，Transformer Decoder による生成が Transformer を用いた場合よりも優れた結果を出した要因を解析することや，一般的な長さの楽曲を生成するために，さらに長期的なシーケンスの生成を行えるようにすることが考えられる．

謝辞

高知工科大学情報学群准教授の高田喜朗先生には指導教員として本研究を行う機会を与えていただき，その遂行の間にも多くのご助言を賜りました．また，研究の成果がなかなか出ない中，あたたかく見守っていただきました．厚く御礼申し上げます．

副査を引き受けていただきました，高知工科大学情報学群准教授の妻鳥貴彦先生，並びに，同学群教授の吉田真一先生に厚く御礼申し上げます．

また，実験に協力してくださった方々にも，厚く御礼申し上げます．

参考文献

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu and Xiaoqiang Zheng. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation, 2016.

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. "Layer normalization," CoRR, 2016.

[3] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever and Dario Amodei. "Language Models are Few-Shot Learners," CoRR, 2020.

[4] Kristy Choi, Curtis Hawthorne, Ian Simon, Monica Dinculescu and Jesse Engel. "Encoding Musical Style with Transformer Autoencoders," CoRR, 2019.

[5] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang and Yi-Hsuan Yang. "MuseGAN:

Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment," AAAI-18, 2018.

[6] Rui Guo, Dorien Herremans and Thor Magnusson. "Midi Miner – A Python library for tonal tension and track classification," CoRR, 2019.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp.770—778, 2016.

[8] Dan Hendrycks and Kevin Gimpel. "Gaussian Error Linear Units (GELUs),", CoRR, 2016.

[9] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew M. Dai, Matthew D. Hoffman, Monica Dinculescu and Douglas Eck. "Music Transformer," CoRR, 2018.

[10] Yu-Siang Huang and Yi-Hsuan Yang. "Pop Music Transformer: Beat-based Modeling and Generation of Expressive Pop Piano Compositions," CoRR, 2020.

[11] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao and Jiawei Han. "On the Variance of the Adaptive Learning Rate and Beyond," CoRR, 2019.

[12] Sageev Oore, Ian Simon, Sander Dieleman, Douglas Eck and Karen Simonyan. "This Time with Feeling: Learning Expressive Musical Performance," CoRR, 2018.

[13] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai and Soumith Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library," CoRR, 2019.

[14] Christine McLeavey Payne. "MuseNet," OpenAI, 2019,

https://openai.com/blog/musenet/.（2021 年 1 月 30 閲覧）.

[15] Colin Raffel. “Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching,” PhD Thesis, Columbia University, 2016.

[16] Colin Raffel, Daniel P. W. Ellis. “Intuitive Analysis, Creation and Manipulation of MIDI Data with pretty_midi,” 15th International Society for Music Information Retrieval Conference, 2014,
http://craffel.github.io/pretty-midi/.

[17] Yi Ren, Jinzheng He, Xu Tan, Tao Qin, Zhou Zhao and Tie-Yan Liu. “PopMAG: Pop Music Accompaniment Generation,” CoRR, 2020.

[18] Ian Simon, Sageev Oore. “Performance RNN: Generating Music with Expressive Timing and Dynamics,” Magenta Blog, 2017,
https://magenta.tensorflow.org/performance-rnn.（2021 年 1 月 30 閲覧）.

[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser and Illia Polosukhin. “Attention Is All You Need,” NIPS 2017, 2017.

[20] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest and Alexander M. Rush. “HuggingFace’s Transformers: State-of-the-art Natural Language Processing,” CoRR, 2019.