AIを用いた楽曲制作に関する検討

1532117　秋場　　翼
1532151　松元　孝樹
指導教員：中村 直人 教授

平成31年度

目 次

図 目 次

表 目 次

第1章　序論

1.1　研究の背景と目的

AI の概念の起源は, コンピュータ開発の父, アラン・チューリング氏が 1950 年に提起した「機械は思考できるのか」という問いであるとされている. チューリング氏は, 機械が思考したかどうかは, 人との会話が成立したかどうかで判断する, とした. 機械との会話が成立したかどうかの判断は次のように行う. 人の審査員が, 相手が見えない状況で「人」と「コンピュータ」と対話して, 対話した相手が人なのかプログラムなのか言い当てる. 審査員がプログラムと対話した後に「『人と』対話した」と間違った答えを出せば,「機械は思考した」ことになる. これが世にいう「チューリングテスト」である. ただチューリング氏はこのとき「AI」という言葉を使ったわけではない. チューリングテストに初めて合格したのは,2014 年に発表された, ウクライナ在住の 13 歳の少年が開発したプログラムだった.1950 年から実に 64 年の時間をかけてコンピュータ技術者は AI のスタートラインに立ったわけである.

1.1.1　第 1 次 AI ブーム

その後, 次の AI ブームは 1950 年代後半から 1960 年代に起きた. この時代の AI の出来事として, ダートマス会議と人工対話システムである「エリーザ」が大きく挙げられる. ダートマス会議は 1956 年に米ニューハンプシャー州のダートマス大学で開催された, コンピュータ研究者たちの勉強会である. ダートマス会議が AI 史に名を残しているのは, 同会議の発起人であるジョン・マッカーシー氏が AI（Artificial Intelligence, 人工知能）という言葉を初めて使ったからだ. この会議で行われたデモンストレーションは, 数学原理をコンピュータで証明することだった. 当時のコンピュータはせいぜい四則演算が限界だったため, これは画期的な成果といえた. ただ,「コンピュータが自ら学ぶ」と理解されている現代の AI からするとまったく「AI らしくない」. 人との対話チューリングテストも, 到底クリアできるレベルではない. 人工対話システム「イライザ」（1966 年） イライザ（ELIZA）は, マサチューセッツ工科大学（MIT）のジョセフ・ワイゼンバウム氏が 1966 年に作成した人工対話システムである. イライザによって人類は初めてコンピュータと会話したのである. ただイライザは「考えて回答」しているわけではなかった. 例えば人がイライザに「腹が痛い」と言えば, イライザは「な

ぜ腹が痛いのか？」と返す. これは確かに会話だが,「からくり」があった. イライザに多数の会話パターンを仕込んでおいたのだ. よって, 仕込んだパターン以外の質問をイライザにすると, イライザは回答できなくなる. しかしイライザと会話している人が偶然, ワイゼンバウム氏が想定したパターンに則した質問を続けると, 見事に会話が成立した. そのため多くの人が「イライザには知性がある」と信じた. さて, 第 1 次 AI ブームがしぼんだ理由は諸説あるが, 最も説得的なものは「当時の技術ではブレークスルーできなかった」というものだろう.AI の理論も,AI の理論を支える技術も, そして AI の可能性を信じてお金を出す投資家もいなかったわけである.

1.1.2　第 2 次 AI ブーム

1980 年代に入り, 家庭にコンピュータが普及したことにより第二次ブームが発生しました. 第二次 AI ブームの特徴として「エキスパートシステム」が挙げられます.「エキスパートシステム」とは専門家の知識をコンピュータに教え込みことで現実の複雑な問題を人工知能に解かせることを試みたシステムです. 第一次ブームと比較してコンピューターの小型化・性能が高まっており, ある程度はこれらの試みは成功しましたが, 知識を教え込む作業が非常に煩雑であること, 例外処理や矛盾したルールに柔軟に対応することが出来ませんでした. 日常世界を見渡してみると, これらの例外処理や矛盾したルールは非常に多く, 知識を教え込む作業が非常に困難なことから, 第二次 AI ブームは自然に消滅へと向かってしまいました. その後,1990 年代半ばに Windows95 の登場, インターネットの普及, 検索エンジンの高性能化が進み, 世界中にいる誰もが簡単に大量のデータを扱える時代に突入しました.

1.1.3　第 3 次 AI ブーム

第二次 AI ブームでのエキスパートシステムが壁にぶつかった問題として, 日常世界には例外処理や矛盾したルールが非常に多く, 知識を教え込む作業が非常に困難というのがありました. これはコンピュータはプログラムと呼ばれるあらかじめ INPUT された命令を順次行っていくため,INPUT されていない例外処理や, 矛盾したルールにぶち当たった場合に柔軟に対応出来なかったためです. これらを解決する手段として「機械学習」や「ディープラーニング」にてコンピュータが自らが学んでいくという手法が第二次 AI ブームの時代から研究されていましたが, 実用化するためにはコンピュータの性能が追い付いていませんでした. しかし,2000 年代に入り, コンピューターの小型化・性能向上に加えインターネットの普及, クラウドでの膨大なデータ管理が容易となったことで実現可能なレベルとなり, 第三次 AI ブームが沸き起こりました. 第三次

AI ブームでの主な出来事 1997 年 : チェス専用のコンピューターが世界王者に勝利 2006 年 : ディープラーニングの実用方法が登場 2011 年 : IBM ワトソンがクイズ番組で人間に勝利する 2012 年 : 画像認識の向上で画像データから「猫」を特定できるようになる 2016 年 :「アルファ碁」がプロ棋士に勝利を収める 2016 年はディープラーニングを起爆剤とした AI が社会に衝撃を与え急速に発達した年であると言われています. 次の年の 2017 年が実用的なシステムも世の中に登場し始めてきており「AI 元年」と呼ばれています. このように AI ブームは 1950 年代の第一次 AI ブーム,1980 年代の第二次 AI ブームと盛り上がっては衰退していきましたが, 数々の実用的なシステムの登場により第三次 AI ブームは継続して続いていくだろうと予想されています.

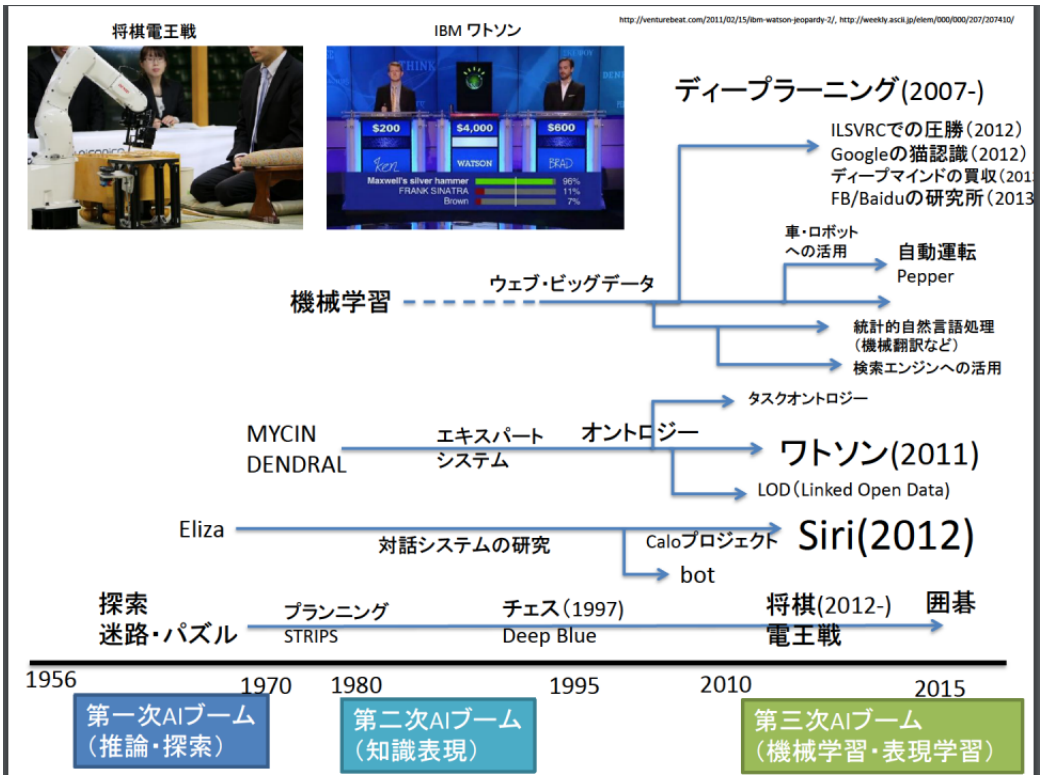

図 1.1: 第一次 AI ブームから今までの流れ(引用 : https://bit.ly/2Wc9Ykb)

スマートスピーカなどの対話型の AI が Google や Amazon,IBM によって商品化され, 現在ではスマートフォンにも Siri という AI が搭載されるなどその存在は非常に身近になっており，その種類も非常に多岐にわたる.

また囲碁や将棋, チェスなどの競技においても，プロに AI が勝利するなどその精度は以前から高いが, その AI は一つの競技でしか使用できない特化型人工知能 (AGI) でありった. しかし, 英 DeepMind が発表した AlphaZero という様々なボードゲームに対応できる汎用性を持った AI が発表され, 汎用人工知能（GAI）の成長も著しい.

図 1.2: 多種多様なスマートスピーカー

自然言語処理を用いた芸術の分野では,2012 年にスタートした人工知能を使って小説を生成するプロジェクトが「星新一賞」の第一審査を通過した. また, 絵画や音楽に関しても AI が作成した肖像画が米競売大手クリスティーズのオークションで 43 万 2500 ドル（約 4900 万円）で落札され,AI を用いて新しい作品を作るものが出回っている.
このように AI の発展は様々な分野においてその成果を上げており，今後は業務の効率化や補助だけにとどまらず，自動車の自動運転や医療の現場でも人間の手よりも高精度なものとして活躍することが期待されている.

1.1.4 様々な AI 楽曲作成サービス

AI を用いた楽曲作成サービスは Amper Music という「作成したい音楽ジャンル」と「曲の長さ」を指定すれば,AI がオリジナル曲を作曲してくれるサービスです. 作成した曲は, テンポを変えたり, 楽器を追加したりと後からも編集できるようにもなっている. また,Jukedeck 本研究では AI による楽曲生成についての実証実験を行う. Googole brain によって公開されている Tensorflow のライブラリである Magenta は AI Duet やそのライブラリを用いて学習データやノード数による楽曲の生成結果の違いを比較, 検証し，AI による楽曲制作が有用なものか調査する.

1.2　本論文の構成

本論文の構成は以下の通りである.

第 1 章では本論文の背景と目的について述べている.

第 2 章では本論文で利用する理論について述べている.

第 3 章では実験内容について述べている.

第 4 章では楽曲制作について述べている.

第 5 章では AI を用いた楽曲制作についての本研究の結論について述べている.

第2章　理論

2.1　AIを用いた楽曲作成

2.1.1　MIDI

MIDIとオーディオデータの形式について述べるパソコンで用いる音楽データには大きく分けて二つある. 一つ目はオーディオデータといい波形の情報を記録する形式である. 二つ目がMIDIであるAIによる曲制作では主にMIDIファイルの音楽データを使用する.MIDIファイルには実際の音ではなく音楽の演奏情報（音の高さや長さなど）である. 本研究で用いるAIはこのMIDIファイルの情報を元に学習をする. また入出力の際もこの規格を用いる.

2.1.2　Magenta

本研究で使用するMagenta[1]は音楽などをTensorFlowを使って機械学習するライブラリであり，Google BrainがGitHab上に公開されているOSSである. Magentaではまず学習させたい音楽のMIDIデータをNoteSequence（magentaが扱うファイル形式）とよばれるデータフォーマットに変更する. それを学習用データセットと評価用データセットに変換したあと学習を行う. このとき，一度に学習させるデータの数，学習を行う回数，ノード数を設定する. これをパッケージ化し，MIDIファイルとして新たに楽曲を生成するという流れである. これを図2.1に示す.

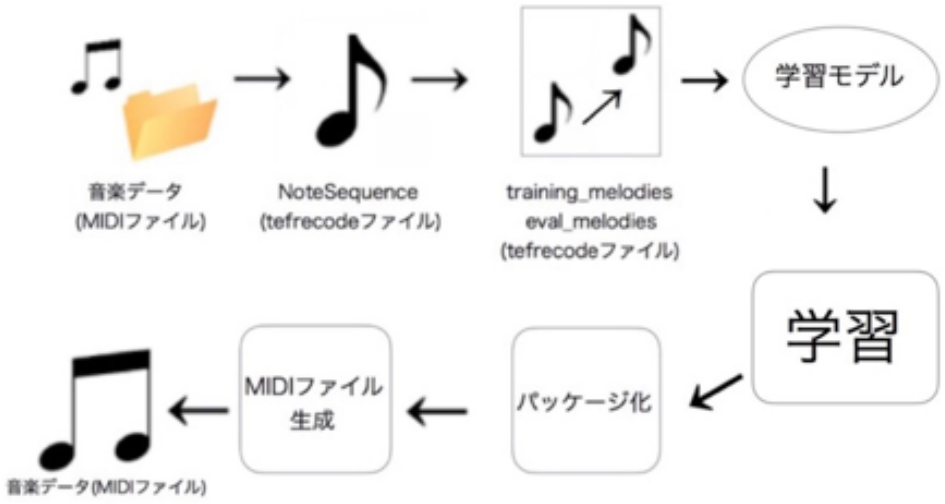

図 2.1: magenta による MIDI 音楽データ生成までのプロセス

2.1.3 MIDI

パソコンで用いる音楽データには大きく分けて二つある. 一つ目はオーディオデータといい波形の情報を記録する形式である. 二つ目が MIDI である AI による曲制作では主に MIDI ファイルの音楽データを使用する.MIDI ファイルには実際の音ではなく音楽の演奏情報（音の高さや長さなど）である. 本研究で用いる AI はこの MIDI ファイルの情報を元に学習をする. また入出力の際もこの規格を用いる.

2.2 機械学習に適した開発環境について

2.2.1 CUDA

CUDA というのは、NVIDIA が開発している GPU 上でプログラミングをするためのソフトウェアプラットフォームになります。含まれるものとしては、CUDA を実行形式に変えるコンパイラみたいなもの、それをサポートする SDK、ライブラリ、あとはデバッグツール群ですね。——CUDA を導入することによって、何がどう変わるのでしょうか？橋本：一般的に行われるプログラミングというのは、いわゆる順次処理に適したものです。それを並列化していくなかで、単に複数のプロセッサで動かせばいいということではなく、いかに無駄なく並列化するのか、というのが重要なのですが、これが非常に難しい問題なんですね。特に NVIDIA の GPU には非常に多くのプロセッシングユニットが含まれています。そして CUDA にはこれを無駄なく活用するためのいろいろなテクニックが組み込まれています。CUDA のプログラムを実行する

にあたって、GPU はグループ化されているんですね。GPU にはたくさんのコアがありますが、プログラムは 1 個 1 個のコアを認識するのではなくて、ある一定の数でグループ化されていて、その各々のグループに対してセットのオペレーションを振り分けていくんです。これはどういうことかと言うと、コンピュータを実行するためには計算だけではなく、メモリのリードライトが必要です。実はこのメモリのリードライトってものすごく時間かかるわけですよ。だからプログラムをコアで実行しようとすると、最初にやるのはメモリリードなんです。まず、データが来るまでにすごく待ちます。計算をしたら次のデータがほしいので、また待たないといけません。すると時間軸上でプロセッサが動いてるのはココとココ（下記の画像の、上側のオレンジの部分）だけになるんです。GPU ではテクスチャ処理などを実行するときに、ピクセルを順次に計算していくのですが、次のピクセルで読まなきゃいけないデータはある程度予測できるんです。必要なデータに事前にメモリーリクエストを出すんですよ。いわゆるスケジュールドアクセスですね。同じようなテクニックを使って CUDA でも次の処理に必要なデータを先読みしようと。そうすることによって、計算処理があって、これを 10 スレッド走らせるときに、クロックを少しずつずらして実行していくんですね。そうするとプロセッサが無駄なく回るわけですよ。こういった処理をより書きやすくしたのが CUDA というわけです。ちなみに、CUDA の形式で書けばなんでも速くなるかというと、そうではありません。各人は事前に、どこにどういうデータがあって、どういう風に割り当てて、どういう風にアクセスさせていくかということを設計しなければダメですね。CUDA の場合は設計が一番重要になってきます。

2.2.2　開発環境

前提としてディープラーニング（深層学習）などの用途で NVIDIA 社製 GPU ボードを利用する場合、 ディープラーニング用フレームワーク等のソフトウェアを導入する前に 下記の NVIDIA 社製ソフトウェアを導入する必要があります.

(1) CUDA Toolkit

(2) PU ボード用ドライバーソフトウェア

(3) 追加のライブラリ (cuDNN など

Magenda プロジェクトが推奨している開発環境の構築には二つ方法があり, 一つは Docker というコンテナ型の仮想化環境を構築できるオープンソースソフトウェアを用いる方法と, ローカル環境に Python のパッケージ管理システムである pip を用いて構築する方法の 2 つがある.
Docker を用いることで仮想化環境をクラウド上で共有できるサービスをである DockerHub を使用できるため, パッケージのインストールをおこなわずにコマンド一つで環境を構築できる. しかし, コンテナに GPU を割り当てることは現状では NVIDIA が公開している OSS の nbidia-docker という

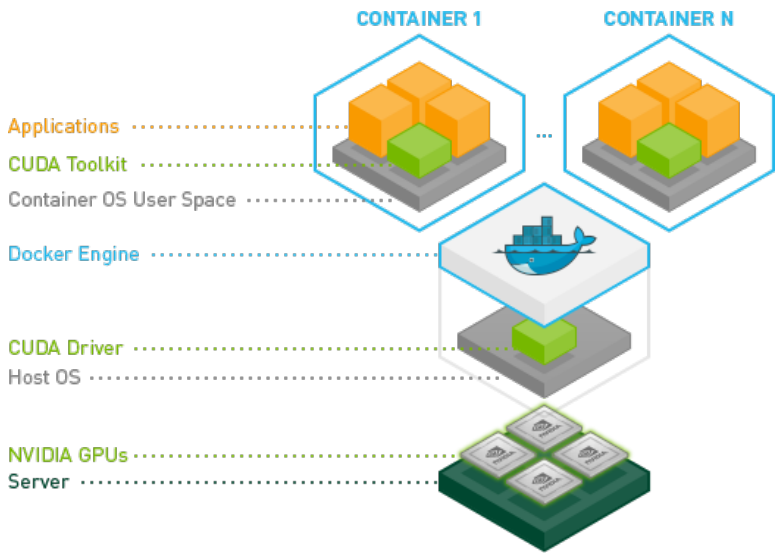

図 2.2: Docker 用 NVIDA コンテナランタイム
(引用：https://github.com/NVIDIA/nvidia-docker)

本システムの開発環境を表 2.1 に示す.

表 2.1: 開発環境

OS	Ubuntu
CPU	Intel Core i5
メモリ	8GB
GPU	GeForce GTX 1060
使用ライブラリ	TensorFlow ， magenta

本システムを使用し OS は Ubuntu を使用した．使用するライブラリとして，ニューラルネットワークを構築できる Tensorflow を利用した．

第3章　実験内容

3.1　モデルによる違い

本研究では Magenta で用意されている 2 つの学習モデルを用いた。以下に示すモデルを用いて楽曲制作をそれぞれ行い，比較，検証する.

3.1.1　MelodyRNN

MelodeRNN は楽曲のメロディを制作するモデルである.MelodyRNN である 3 つのモデルを以下に示す.

(1) basic_rnn

前の状態を保持し，これを記憶または忘却する．時系列を学習することにより，次の音の予測を可能にしている．Lookback_rnn と Attention_rnn はこれを基に機能を追加したものである.

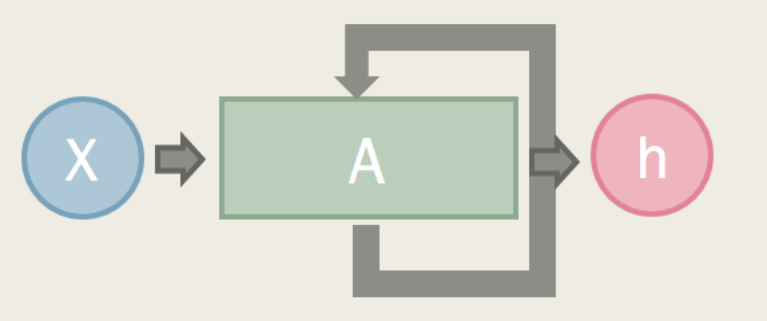

図 3.1: basic_rnn のモデル (1)

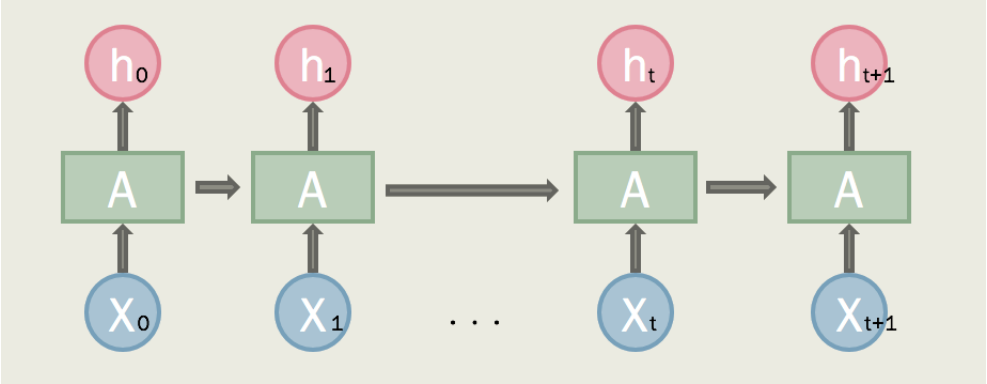

図 3.2: basic_rnn のモデル (2)

Lookback_rnn と Attention_rnn はこれを基に機能を追加したものである

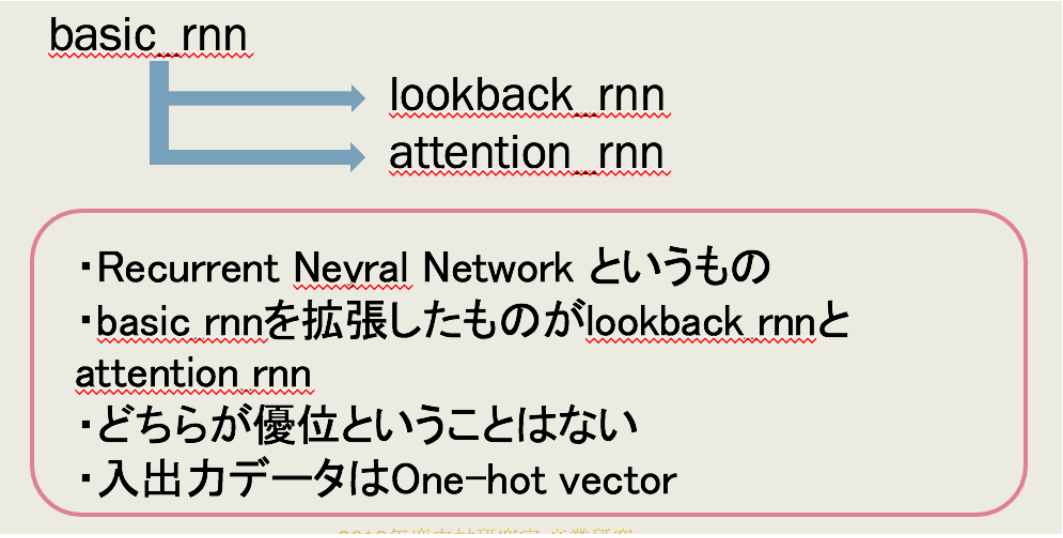

図 3.3: 三つのモデルについて

(2) lookback_rnn

Basic_rnn を基に，１小節前と２小節前の音，拍数，前の小節の繰り返しかどうかの情報を与え，音楽の流れを掴もうとするもの．

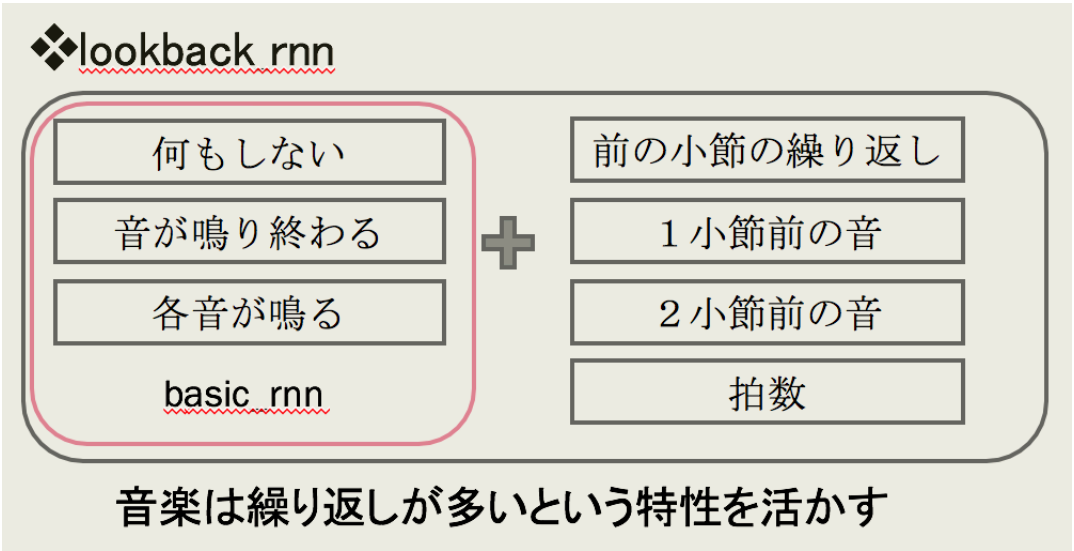

図 3.4: lookback_rnn のモデル

(3) attention_rnn

basic_rnn を基に，過去の情報を予測結果に加えてこれによる繰り返しを捉えるもの．

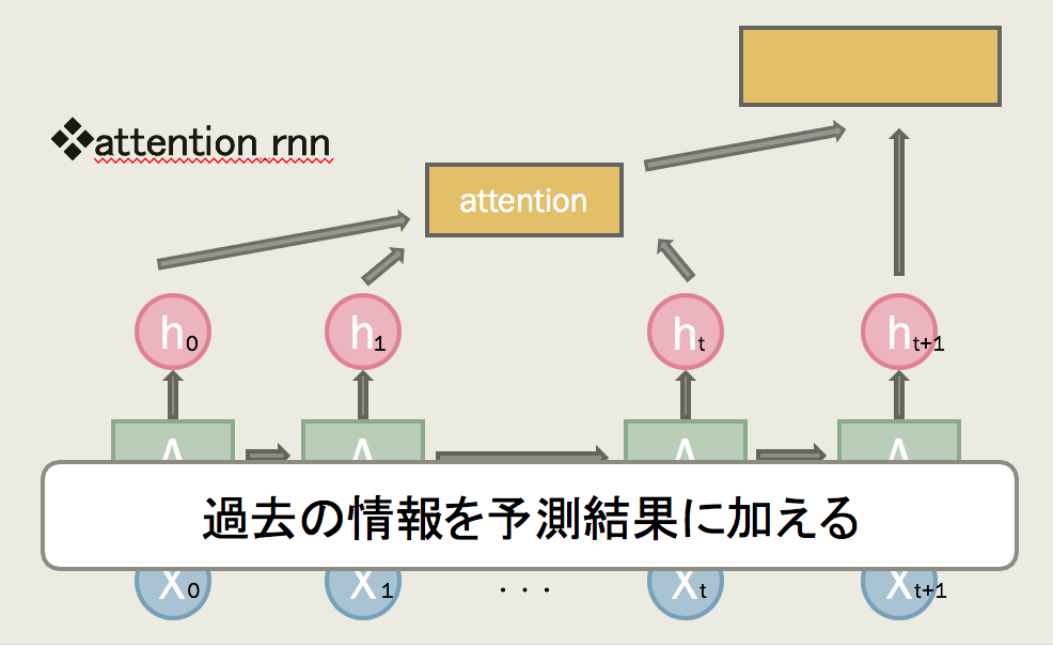

図 3.5: attention_rnn のモデル

3.1.2 PolyphonyRNN

複数の同時音のモデリングが可能になっており，複数音の響きを 1 つのかたまりとして捉えて学習しているモデルである. このモデルを使用することで，伴奏も含めた楽曲の生成が可能である

上記のモデルを用いて制作を行い，それぞれの違いと有用性について検証する．

3.2 学習回数による違い

学習回数を変更して楽曲制作を行い，それぞれの違いと有用性について検証する．

3.3 ノード数による違い

ノード数を変更して楽曲制作を行い，それぞれの違いと有用性について検証する．

第4章　楽曲制作

4.1　Melody_rnn を使用して学習モデルを作成

4.1.1　NoteSequence の作成

NoteSequence とは MIDI データから作成されるプロトコルバッファである. プロトコルバッファとは Google が 2008 年にオープンソース化したバイナリベースのデータフォーマットである. 既存の技術としては XML や JSON などのテキストベースのデータフォーマットがあるが, プロトコルバッファはバイナリフォーマットであるので, アプリケーション間でデータ構造の送受信をする際に少ないデータ量ですむという特徴がある. スキーマ言語はなぜ重要なのか、そういう時代になったからだ. 我々が単一 RDB に接続する単一の Web アプリだけを書いていられる時代はとうの昔に終わった。データはあちこちのいろんなストレージ技術で保存されているかもしないし、バックエンドも単一サービスではなくて分割されているかもしれないし、クライアントは web 版, iOS 版, Android 版があってひょっとしたらそれぞれ別の言語で実装されており、外部開発者向けに API も公開しなければならない。だから、そこら中でデータをシリアライズするしデシリアライズするし、通信の両端で解釈に矛盾が無いようにすり合わせる必要がある。でも、すりあわせの目的でいちいち人間と自然言語で会話するのは苦痛なので、私たちは機械処理可能なコードで語りたい。だから、どういうデータがやってくるのかきちんと宣言的 DSL で定義しておきたい。そこでスキーマ言語だ。JSON Schema が広がりつつあるのもたぶんそういう理由だし、そもそも Protobuf 自体も Google が社内で同じ問題にぶち当たって発明されたんだったような気がする。自分の管轄領域内に閉じたデータ構造であれば、スキーマとか型宣言とか細かい縛りなしで軽量に進めるのもありだ。しかし、他人の領域との界面はしっかり定義しておいた方が良い。スキーマ定義という税金を支払わないとその分のツケはどこかで回ってきて、1 時間掛かる E2E テストがこけるとか、週次変更レビュー合同会議の発足とか、そういうやつで支払うことになる。スキーマ定義さえあれば、web クライアント開発用の, バックエンド開発用の, アプリ開発用の言語に向けて対応するデータ構造定義を自動生成して、矛盾なくすべてをメンテナンスできる。通信の両端でスキーマさえ合致していれば、シリアライゼーション形式を合わせるのはむしろ間違えづらい。だから、JSON でも Protobuf のデフォルトのそれでも、MessagePack でも XML でも好き

にすれば良い。ちなみに、Protobuf で定義したデータ構造は Protobuf 標準形式の他に JSON にもきちんとシリアライズできる。NoteSequence の作成は以下に示すコマンドで作成できる.
–input_dir で学習させる MIDI データのディレクトリの絶対パスを指定し,–output_file で Notesequence の出力先のディレクトリを指定する.

```
    convert\_dir\_to\_note\_sequences \
    --input\_dir=\$INPUT\_DIRECTORY \
    --output\_file=\$SEQUENCES\_TFRECORD \
    --recursive
```

次に作成した NoteSequence のデータセットを学習用と評価用に分割するために以下に示すのコマンドを実行する.
–config で使用する RNN を指定する.–input_dir で NoteSequence の絶対パスを指定し,–output_file で分割した Notesequence の出力先のディレクトリを指定する. –eval_ratio で Notesequence のデータを何パーセント学習用に用いるかを指定する. コマンドの場合は 10%が学習用のデータになる.

```
melody_rnn_create_dataset \
--config=<one of 'basic_rnn', 'lookback_rnn', or 'attention_rnn'> \
--input=/tmp/notesequences.tfrecord \
--output_dir=/tmp/melody_rnn/sequence_examples \
--eval_ratio=0.10
```

4.1.2 学習の開始

作成した NoteSequence から学習モデルを作成するために以下のコマンドを実行する.
–config で学習に使用する学習モデルを指定,–rundir で学習のために用意した Notesequence を指定し,–sequence_examplefile で学習モデルの出力先のディレクトリを指定する. –hparams でメモリの使用量を指定し,–rnn_layer_size で中間層のノード数を指定し,–num_trainingsteps で学習回数を設定する.

```
melody_rnn_train \
--config=attention_rnn \
--run_dir=/tmp/melody_rnn/logdir/run1 \
--sequence_example_file=/tmp/melody_rnn/training_melodies.tfrecord \
--hparams="batch_size=64,rnn_layer_sizes=[64,64]" \
--num_training_steps=20000
```

4.1.3　音楽データの作成

以下に示すコマンドで学習モデルに入力する.
–config で学習に使用する学習モデルを指定,–rundir で学習済みのモデルを指定し,–output_dir で音楽データの出力先のディレクトリを指定する. –num_outputs で生成する音楽データの個数を指定し,–num_steps で–hparams でメモリの使用量を指定し,–rnn_layer_size で中間層のノード数を指定し,–primer_melody で学習モデルに入力する最初の音程を MIDI の形式で指定する.

```
melody_rnn_generate \
--config=attention_rnn \
--run_dir=/tmp/melody_rnn/logdir/run1 \
--output_dir=/tmp/melody_rnn/generated \
--num_outputs=10 \
--num_steps=128 \
--hparams="batch_size=64,rnn_layer_sizes=[64,64]" \
--primer_melody="[60]"
```

4.1.4　事前に学習済のモデルを使用

また,Magenda のプロジェクトにすでに学習済のモデルが存在するのでそれを使用して音楽データを作成することもできる. 生成には mag バンドファイルが必要になるので magenda の Github に公開されているので, それをダウンロードしてくる. その後以下に示すコマンドを実行する事で生成することができる.
–config で学習に使用する学習モデルを指定,–rundir で学習済みのモデルを指定し,–output_dir で音楽データの出力先のディレクトリを指定する. –num_outputs で生成する音楽データの個数を指定し,–num_steps で–hparams でメモリの使用量を指定し,–rnn_layer_size で中間層のノード数を指定し,–primer_melody で学習モデルに入力する最初の音程を MIDI の形式で指定する.

```
melody_rnn_generate \
--config=${CONFIG} \
--bundle_file=${BUNDLE_PATH} \
--output_dir=/tmp/melody_rnn/generated \
--num_outputs=10 \
--num_steps=128 \
--primer_melody="[60]"
```

4.2 Polyphony_rnn を使用して学習モデルを作成

PolyphonyRNN を使用してする手順としては, 大まかな流れは同じであるが–config で使用する RNN を指定する必要がないという違いがある

4.2.1 NoteSequence の作成

NoteSequence の作成は以下に示すコマンドで作成できる.
–input_dir で学習させる MIDI データのディレクトリの絶対パスを指定し,–output_file で Notesequence の出力先のディレクトリを指定する.

```
    convert\_dir\_to\_note\_sequences \
    --input\_dir=\$INPUT\_DIRECTORY \
    --output\_file=\$SEQUENCES\_TFRECORD \
    --recursive
```

次に作成した NoteSequence のデータセットを学習用と評価用に分割するために以下に示すのコマンドを実行する.
–input_dir で NoteSequence の絶対パスを指定し,–output_file で分割した Notesequence の出力先のディレクトリを指定する. –eval_ratio で Notesequence のデータを何パーセント学習用に用いるかを指定する. コマンドの場合は 10%が学習用のデータになる.

```
    polyphony_rnn_create_dataset \
    --input=/tmp/notesequences.tfrecord \
    --output_dir=/tmp/polyphony_rnn/sequence_examples \
    --eval_ratio=0.10
```

4.2.2 学習の開始

作成した NoteSequence から学習モデルを作成するために以下のコマンドを実行する.
–config で学習に使用する学習モデルを指定,–rundir で学習のために用意した Notesequence を指定し,–sequence_examplefile で学習モデルの出力先のディレクトリを指定する. –hparams でメモリの使用量を指定し,–rnn_layer_size で中間層のノード数を指定し,–num_trainingsteps で学習回数を設定する.

```
polyphony_rnn_train \
--run_dir=/tmp/polyphony_rnn/logdir/run1 \
--sequence_example_file=/tmp/polyphony_rnn/training_tracks.tfrecord \
--hparams="batch_size=64,rnn_layer_sizes=[64,64]" \
--num_training_steps=20000
```

4.2.3　音楽データの作成

以下に示すコマンドで学習モデルに入力する.
–config で学習に使用する学習モデルを指定,–rundir で学習済みのモデルを指定し,–output_dir で音楽データの出力先のディレクトリを指定する. –num_outputs で生成する音楽データの個数を指定し,–num_steps で–hparams でメモリの使用量を指定し,–rnn_layer_size で中間層のノード数を指定し,–primer_melody で学習モデルに入力する最初の音程を MIDI の形式で指定する.

```
polyphony_rnn_generate \
--run_dir=/tmp/polyphony_rnn/logdir/run1 \
--hparams="batch_size=64,rnn_layer_sizes=[64,64]" \
--output_dir=/tmp/polyphony_rnn/generated \
--num_outputs=10 \
--num_steps=128 \
--primer_pitches="[67,64,60]" \
--condition_on_primer=true \
--inject_primer_during_generation=false
```

第5章　結論

5.1　今後の課題

aaa

謝辞

参考文献

[1] Google Brain チーム,”Magenda” https://github.com/tensorflow/magenta

[2] 財経新聞,”人工知能を使って執筆した小説が星新一賞一次選考を通過” https://www.zaikei.co.jp/article/20160323/299468.html

[3] YAHOO ニュース,”AI 絵画、大手オークションで初の落札 予想額の 40 倍超” https://headlines.yahoo.co.jp/hl?a=20181026-00010002-afpbbnewsv-int

[4] マイナビニュース,”XML はもう不要!? Google 製シリアライズツール「Protocol Buffer」” https://news.mynavi.jp/article/20080718-protocolbuffer/