修士論文

ニューラルネットワークを用いた
旋律概形による作曲支援システムの研究

1610221 渡邉貴也

主指導教員	東条 敏
審査委員主査	東条 敏
審査委員	白井 清昭
	NGUYEN, Minh Le
	飯田 弘之

北陸先端科学技術大学院大学

先端科学技術研究科　情報科学

平成 30 年 8 月

概要

自動作曲システムは近年、作曲のスキルを問わずに手軽に曲を作成する手段として注目を集めている。コンピュータの普及に伴いより多くのユーザが様々な場面で目的によって異なった曲を欲しいという需要がある。しかし、ゲーム上で使われる BGM を作成するなどの音楽を作曲する必要性はあるが高度な音楽の知識を持たないユーザが実際に期待する曲の詳細を作曲システムに伝え、そのユーザの満足する音楽が得られることはごく稀である。また、そのようなユーザは「ここで音程を上げる」や、「段々早くする」などの曲の大まかなイメージを持っている場合があると考えられる。作曲のスキルのないユーザが作曲を行うには音符単位での表現ができずとも曲の大まかなイメージからから作曲するシステムが必要である。

本研究では既存の楽譜上に存在する旋律を用いることで作曲の知識のないユーザでも既存の楽譜のような作曲が行えるシステムを構築する。また、今回は旋律を大まかな曲線として表す旋律概形と呼ばれる旋律表現を用いる。旋律概形は先行研究[1]で用いられている表現である。楽譜上で連続する音階をつなぎ合わせた時間の経過で音階の推移を大まかな曲線、直線で表現したものである。先行研究[2]では旋律概形に加え HMM(隠れマルコフモデル)を用いて音楽的に適切な音符列を楽譜上から学習した結果を用いて推定しているが、先行研究[2]の手法では和声を考慮していないという課題があった。

この課題を解決する手法としてデータの時系列を学習、推定可能なニューラルネットワークを用い、旋律概形から実在する楽譜の旋律の一部を出力するシステムを構築する。ニューラルネットワークは用意した入力データと教師データで学習を行う。学習の終わったニューラルネットワークは既知のデータが入力された場合、学習時にその入力とペアとなっている出力を行う。また、未知のデータが入力された場合は学習した出力の中で尤もらしい出力を行う。しかしながら本研究では旋律概形と旋律はペアで与えられるとは限らない。近年の研究で DiscoGAN というものがある。これは入力データとペアの教師データが存在せずとも入力データの特徴をとらえた教師データの出力を行うことができる。このモデルは 2 つ

の生成器(以下 Generator)と 2 つの認識器(以下、Discriminator)を用いる。それぞれ Generator AB, BA、Discriminator A, B とした場合、本研究システムでは Generator AB は旋律概形を入力として旋律を推定する。Generator BA は旋律を入力として旋律概形を推定する。また、Discriminator B は旋律が生成されたものか既存の楽譜のものであるかを判別する。Discriminator B は旋律概形が Generator AB から推定されたものか事前に用意されたものを判定する。Generator AB から推定された旋律は Generator BA へ渡され旋律概形に再変換し Generator AB に入力された旋律概形に近づくように学習を行う。また、Generator AB より生成された旋律は Discriminator B に渡され推定された旋律であるか既存の楽譜上の旋律かを判定するこのとき Discriminator B に既存楽譜上の旋律であるよう判定させるように Generator AB を学習させる。また、既存楽譜上の旋律を入力とする Generator BA から旋律概形を推定し Generator AB に再度旋律に変換しなおす。推定された旋律が近づくように学習し、Generator BA によって推定された旋律概形は Discriminator A に渡され判別させる。このとき Discriminator A で用意された旋律概形であるように判定をさせるように Generator BA を学習する。この一連の処理を行うことにより Generator AB は旋律概形から旋律をより既存楽譜上の旋律に近しいものを推定するようになり、Generator BA はより尤もらしい用意された旋律概形を推定する。Discriminator A は旋律概形が生成されたものであるかを判別することに特化し、Discriminator B は旋律が生成されたものであるかを判別することに特化する。Generator AB では旋律概形から尤もらしい旋律を推定することができると思われる。

本研究ではこれを適応し、旋律概形の特徴をとらえた尤もらしい既存の旋律を推定することを目指す。また、ニューラルネットワークが旋律概形から旋律を推定する上で有効な手法かを検証する。

また、生成手法として DiscoGAN は画像の生成でしばしば使われる。しかしながら、時系列データ処理ではあまり使われてはいない。本研究では一度、画像へ変換するものの、時系列のデータを使用するため C-RNN-GAN を用いる。この手法は GAN に時系列データの処理を得意とする LSTM を組み合わせた時系列データの生成を得意とする手法である。本研究では DiscoGAN と C-RNN-GAN を組み合わせ、時系列データである旋律概形と旋律のペアが明示されていないデータセットに対し旋律概形から旋律の推定が可能であるかを検証する。

目次

第１章　　はじめに

1.1 研究背景

コンピュータの普及に伴い、コンピュータを用いた作曲への関心は高まっている。コンピュータ上で動作する作曲支援システムはユーザが作曲を行う上で作曲を行いやすいよう様々な機能を有している。また、ゲームの BGM でオリジナルの曲を使用したいなど作曲を行いたいがそれに労力をかけたくない場合や作曲の知識を持っては居ないがオリジナルな曲を欲しいなどの要望がある。しかしながら、作曲はそれに関する専門の知識を必要とする。そのような需要の為にも作曲支援システムというものがある。近年、そのような理由から作曲支援システムは誰でも手軽に作曲を行うことのできるツールとして注目を集めている。また、ユーザは「ここで音程を上げる」や、「段々早くする」などの曲の大まかなイメージを持っている場合があると考えられる。作曲のスキルのないユーザが作曲を行うには音符単位での表現ができずとも曲の大まかなイメージからから作曲するシステムが必要である。しかしながら、そのような大まかなイメージを作曲支援システムにユーザの意図した音符の出力を行わせることは難しい。また、ユーザの満足する旋律を得られるとは限らない。本研究では実際の楽譜に登場する旋律を用いれば聞き馴染みのあり、作曲家によって旋律上に含まれる作曲に関する知識を反映した旋律を得られるのではないかと考えた。また、作曲をあまり行わないユーザが要求する旋律は既存の楽譜内に存在する旋律の組み合わせであると思われる。ユーザの持つ大まかなイメージから作曲済みの楽譜上の旋律の一部に変換することによりユーザの要求に近い曲を得られるのではないかと思われる。

1.2　研究目的

作曲に関する知識を持たないが作曲を行いたいというユーザは大まかな曲のイメージを持っていることが多い。本研究では五線譜上に曲線として記述したユーザの持つ旋律のイメージを旋律概形に変換し、そこから既存の楽譜に登場する旋律に変換を行うことを目指す。

1.3　本論文の構成

本論文の構成は第 1 章で研究背景、研究目的を述べる。第 2 章では先行研究と本研究で用いる旋律概形について述べる。第 3 章では本研究システムに適応する諸手法の解説と本研究システムの手法を述べる。第 4 章では実験について述べる。第 5 章でおわりに本研究のまとめと今後について述べる。

第２章　　準備

2.1 先行研究

先行研究[1]では旋律概形という概念を導入している。旋律概形とは楽譜上の一連の旋律を連続する同じ音程を一つのブロックとしてつなぎ合わせたものである。

旋律概形は音符列から自動的に得られ、ユーザが書き出した曲線を旋律概形にあてはめ再度音符列に戻すことを目的としている。よって、ユーザの編集後に音符列に戻すことが必要である。

このことより先行研究では以下の 4 つの条件を満たす。

(1) 各音符の音高や音価が陽には表現されない。

(2) 音符表現を旋律概形に変化し、編集せずに音符表現に再変換すると元の音符表現に戻る。

(3) 旋律概形上で編集を行って音符表現に変換した場合、音楽的に不適切な音は避ける。

(4) 旋律概形に旋律のどの程度細かな動きが表現されるかはユーザが制御できる。

ユーザは旋律概形を用いたシステムで旋律をユーザが満足のいくものとなるまで編集する。

音符列から旋律概形への変換は入力された MIDI シーケンスを音高の時系列に変換される。ここでの音高の時系列とは旋律の音の高さを時間の流れに沿って並べた系列のことである。

音高が対数で表現され、中央の C の音が 60.0、半音が 1.0 となる。休符については、直前の音が継続しているものとして扱う。

音高の時系列を周期信号とみなし信号全体に対してフーリエ変換を行う。高次のフーリエ係数は旋律の細かな特徴を表現するのに対して低次のフーリエ級数は旋律の大まかな動きを表現するものである。得られたフーリエ級数のうち、低次のもののみを取り出し逆フーリエ変換を行う。この処理によって音高の時系列から旋律の大まかな成分のみを取り出すことができる。

また、旋律概形から音符列を生成する手法として編集後の旋律概形をフーリエ変換して得られたフーリエ級数と、旋律概形抽出時に導出した高次のフーリエ級数を結合して逆フーリエ変換を行う。これにより、旋律の大まかな流れをユーザによる操作を反映させながら、旋律の細かな動きに元の旋律の特徴を含んだ音高の時系列が得られる。

得られた音高の時系列から音符列に変換する。

単純化するため編集対象を音高ののみとし、各音符の発音時刻や消音時刻をもとの旋律と等しいのもとする。その場合、ノートナンバを決定する必要がある。このとき、各音符の発音区間における音高の時系列の中央値とする。しかし、編集した箇所を音高の時系列に変換するとゆがみが生じる場合がある。音高の時系列からスケールに即した音符列を推定する。

HMM を用いてゆがんだ音高の時系列を編集する。ゆがんだ音高の時系列は音楽的に正しいノートナンバの系列にノイズが加わったものと考える。隣り合う音符同士の遷移特徴を用いて修正を行う。たとえば、メジャースケールの音に対応する状態への遷移確率が高く設定されている場合、推定される音符はメジャースケールのものになりやすくなる。適切に遷移確率を設定することでスケールに即していない音高が出力されることを抑える。単純化のため、ハ長調である場合を仮定している。

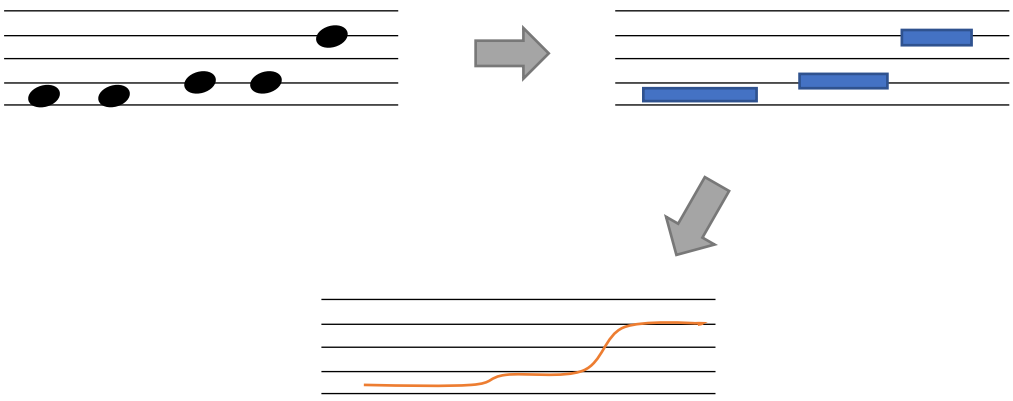


図 2.1 旋律から旋律概形への変換

2.2 本研究の旋律への変換

本研究ではユーザの入力が曲線で得られると仮定し、0~31 の背景の黒い画像内に白い曲線を旋律へ変換することを考える。入力として得られた画像を時間ごとに C major C4 から A5 の 13 音階に分類する。分類の方法は得られた画像を横軸に時間とし横軸に輪切りにした 1 次元のデータを C4 から A5 に収まるように五線譜上に展開した場合、白点が最も近い音階に分類する。この処理を入力された曲線の視点から終点まで行い、分類した時間ごとの音階を再度、縦 0~31 のサイズの画像に連続する同じ音階を直線で繋げた画像として出力する。それを本研究システムに入力し旋律へ変換を行う。

第３章　　手法

3.1 ニューラルネットワーク

本研究では人工ニューラルネットワークを用いた手法を使用する。

人工ニューラルネットワークとは生物の神経網を工学的に応用しやすいように単純化したものである。以下、本論文では人工ニューラルネットワークを単にニューラルネットワークと呼びこととする。学習には誤差逆伝播法を用いる。この手法はニューラルネットワークから出力された値と教師データの値の誤差を最小化するよう学習を行う。

伝播時

$$u_i^l = \sum^j w_{ij}^l x_j^{l-1} + b_i^l \quad (3.1.1)$$

$$x_i^l = \sigma(u_i^l) \quad (3.1.2)$$

誤差逆伝播法

確率降下勾配法の場合には以下の更新を行う。

誤差計算

$$d_i^l = \begin{cases} t_i - x_i \\ \left(\sum^j w_{ij}^{l+1} d_j^{l+1}\right) \sigma'(u_i^l) \end{cases} \quad (3.1.3)$$

重みの更新

$$w_{ij}^l = \eta\, d_i^l\, x_j^{l-1} \quad (3.1.4)$$

l　層番号

j^{l-1}入力ニューロンの値

w_{ij}^l ニューロン間の重み

b_i^l 閾値

σ 活性化関数

d_j^{l+1} 誤差

t_i　教師データ

η　学習率

3.2 GAN

GAN(Generative Adversarial Networks)[4]とは Generator, Discriminator の 2 つのモデルを使い、Discriminator はあらかじめ用意しておいたデータ(Prepared data)と Generator が生成したデータか、判別を行い、Generator は入力からあらかじめ用意しておいたデータに類似するものの生成を行う。本研究ではニューラルネットワークを適応する。

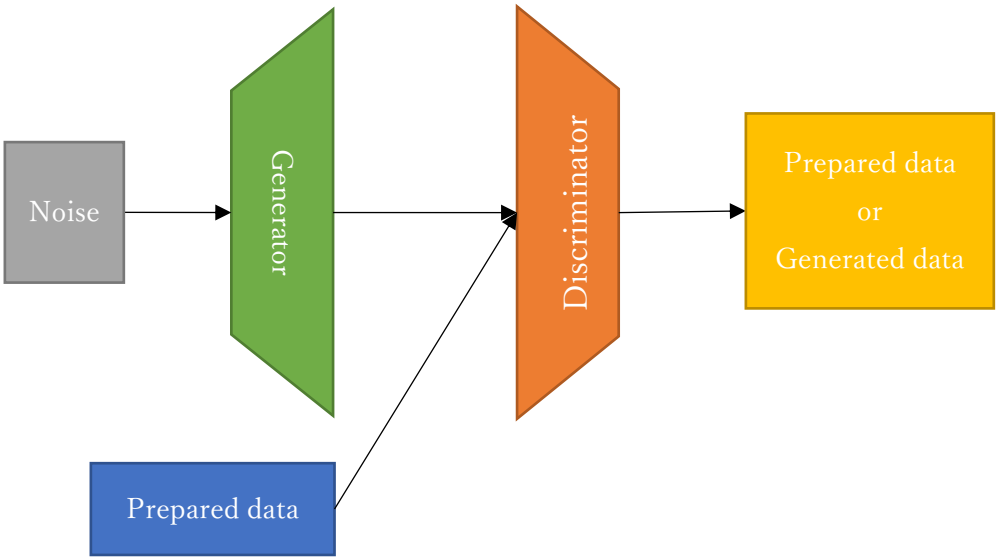


図 3.2.1 一般的な GAN の全体像

Generator は Discriminator にあらかじめ用意された画像と判断されるよう学習し、Discriminator はあらかじめ用意された画像もしくは Generator により生成されたデータであるか判定の精度を向上していくよう学習を行う。

以上を反復することにより、Generator はあらかじめ用意しておいたデータに内在する特徴をとらえた出力を行うよう振る舞う。

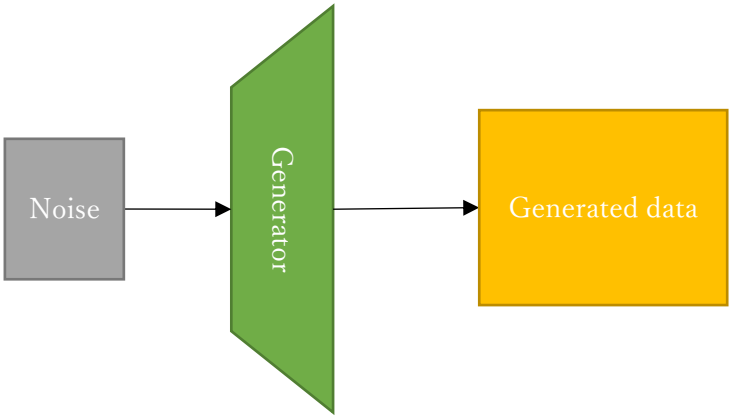


図 3.2.2 生成を行う場合のモデル

3.3 LSTM (Long Short-Term Memory) [5]

時系列信号を扱うことのできる手法である。

長期間の信号の保存と不要な信号の忘却を行うネットワークである。

ある時刻の入力 x^t に重み W_z をかけた値と前回の出力 y^{t-1}に再帰データに関して重み R_z を掛けた値を足し合わせ入力しシグモイド関数 σ に入力する。シグモイド関数σから得られた値をz^t とする。前回の状態と現在の状態で出力を表現する上で必要な要素を保持し不要なもの要素を除去するためx^t に重みW_{in} と掛けたものとy^{t-1}重み R_{in}を掛けたものをシグモイド関数σに渡した値を足し合わせ、z^t にかけるまた、忘却処理として、$f^t = \sigma(w_f x^t + R_f y^{t-1} + b_f)$を実行し前回の状態である$c^{t-1}$を掛ける。これは$f^t$が前回の状態を忘却すべき要素を弱め、必要な要素を強めるためである。その後に$out^t = \sigma(w_{out} x^t + R_{out} y^{t-1} + b_{out})$と$tanh(c^t)$を掛け合わせ、出力とする。

$$z^t = \sigma(w_z x^t + R_z y^{t-1} + b_z) \tag{3.3.1}$$
$$in^t = \sigma(w_{in} x^t + R_{in} y^{t-1} + b_{in}) \tag{3.3.2}$$
$$f^t = \sigma(w_f x^t + R_f y^{t-1} + b_f) \tag{3.3.3}$$
$$c^t = f^t c^{t-1} + z^t i^t \tag{3.3.4}$$
$$out^t = \sigma(w_{out} x^t + R_{out} y^{t-1} + b_{out}) \tag{3.3.5}$$
$$y^t = out^t \, tanh(c^t) \tag{3.3.6}$$

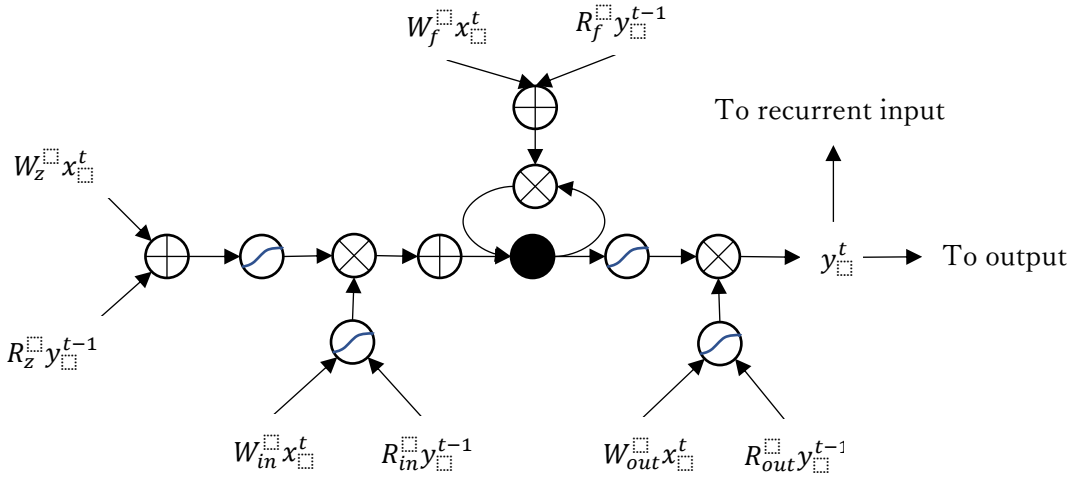


図 3.3 LSTM フロー

3.4 DiscoGAN [6]

DiscoGAN (Discover Cross-Domain Relations with Generative Adversarial Networks) は通常のフィードフォワードニューラルネットワークには入力データとペアとなる教師データが必要である。しかしながら、入力データをペアとなる教師データが存在しない場合がある。その場合に DiscoGAN は有用である。入力データと教師データのペアを有さないデータセットに対し入力データから GeneratorAB(G ab)で教師データに類するデータに変換し、再度 GeneratorBA(G ba)により入力データに再変換することにより入力データに対しユニークな教師データに類する出力を行う。また、教師データを GeneratorBA(G ba)により入力データに類するデータに変換し、再度 GeneratorAB(G ab)を用いて教師データに近しい出力を行うよう学習を行う。このとき、Discriminator A,B の 2 種を用意する。Discriminator A は入力データに類するデータを判別し、Discriminator B は教師データに類する教師データを判別する。それぞれの Discriminator は入力データに内在するの特徴、教師データに内在するの特徴での判別を行わせないよう Discriminator A は入力データ、もしくは教師データから生成された入力データに類するデータ、Discriminator B は教師データ、もしくは入力データから生成された教師データに類するデータの判別のみを行うように学習する。

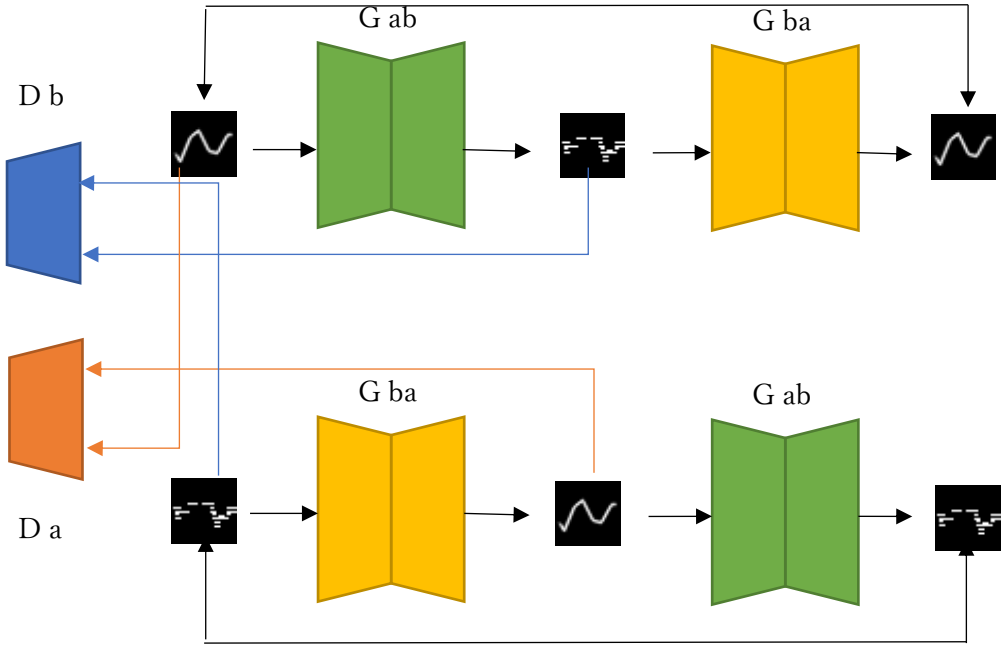


図 3.4 DiscoGAN モデル

3.5 C-RNN-GAN [7]

GAN の Generator, Discriminator を LSTM 層と全結合層を用いたモデルで時系列信号の生成を行うことを得意とする手法である。

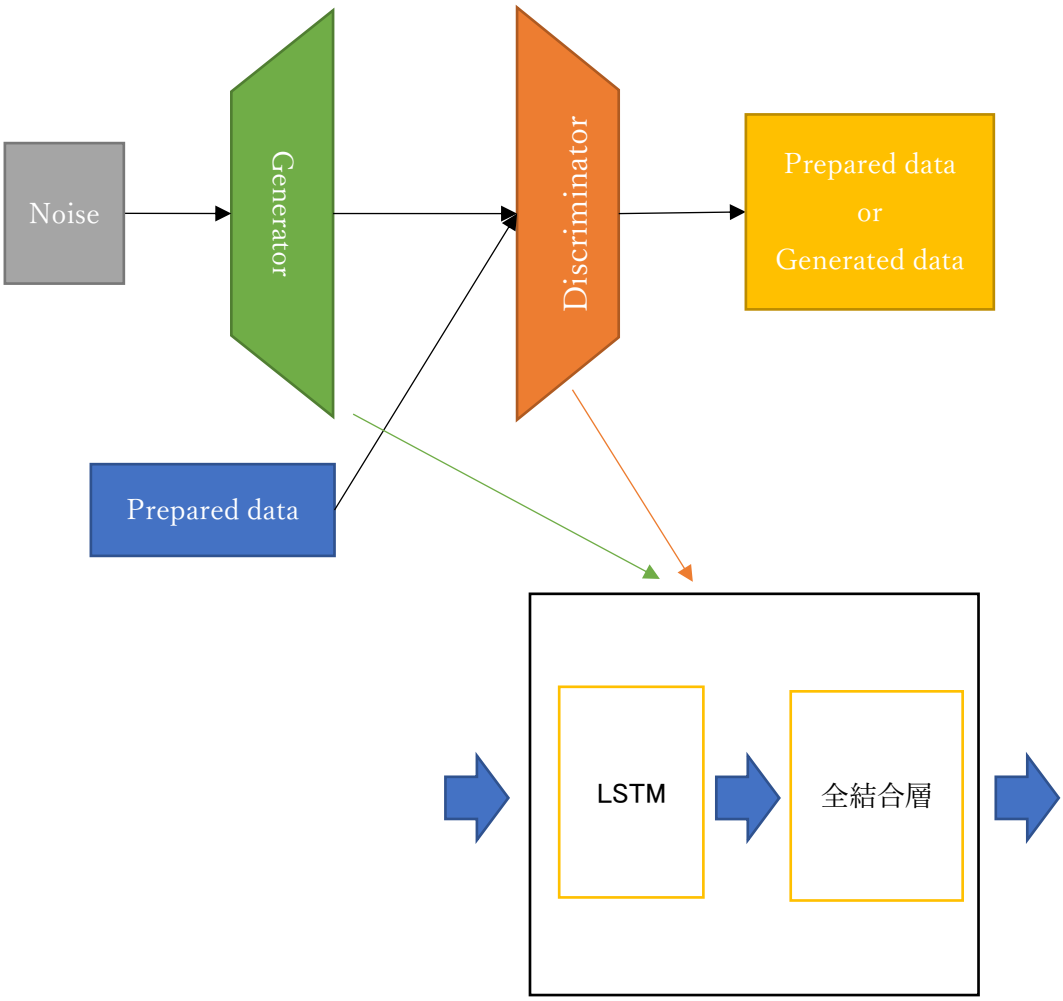


図 3.5 C-RNN-GAN

3.6 本研究手法

本研究では DiscoGAN 内の Generator, Discriminator を LSTM と全結合層に置き換え、時系列信号処理と旋律概形と旋律のペア関係を明示せずに生成を行う事のできる Generator を学習から得る。また、Generator, Discriminator への入力は LSTM, 全結合層の順でモデルを構成する。

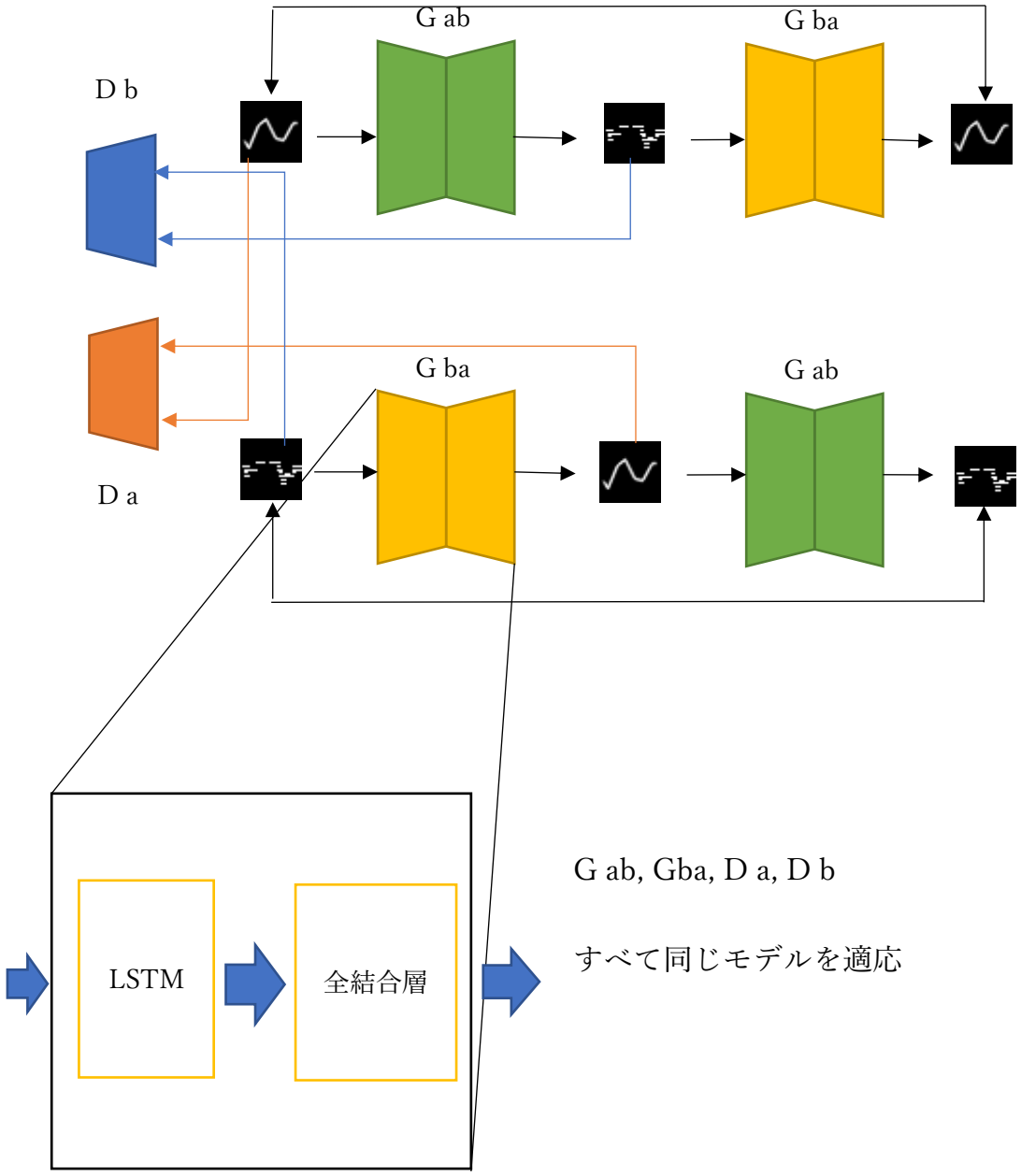


図 3.5 本研究モデル

第4章　　実験

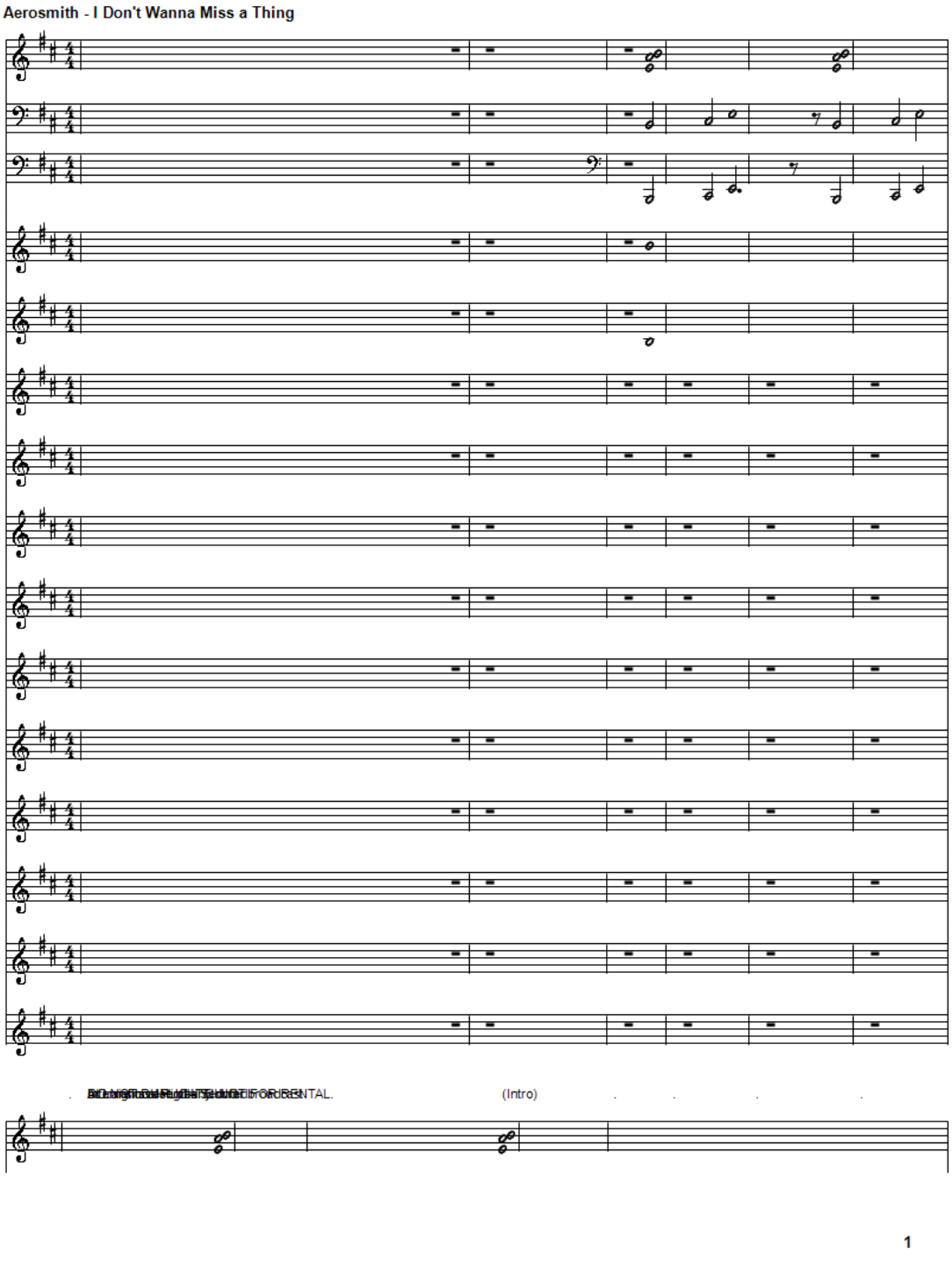
4.1 実験内容と評価基準

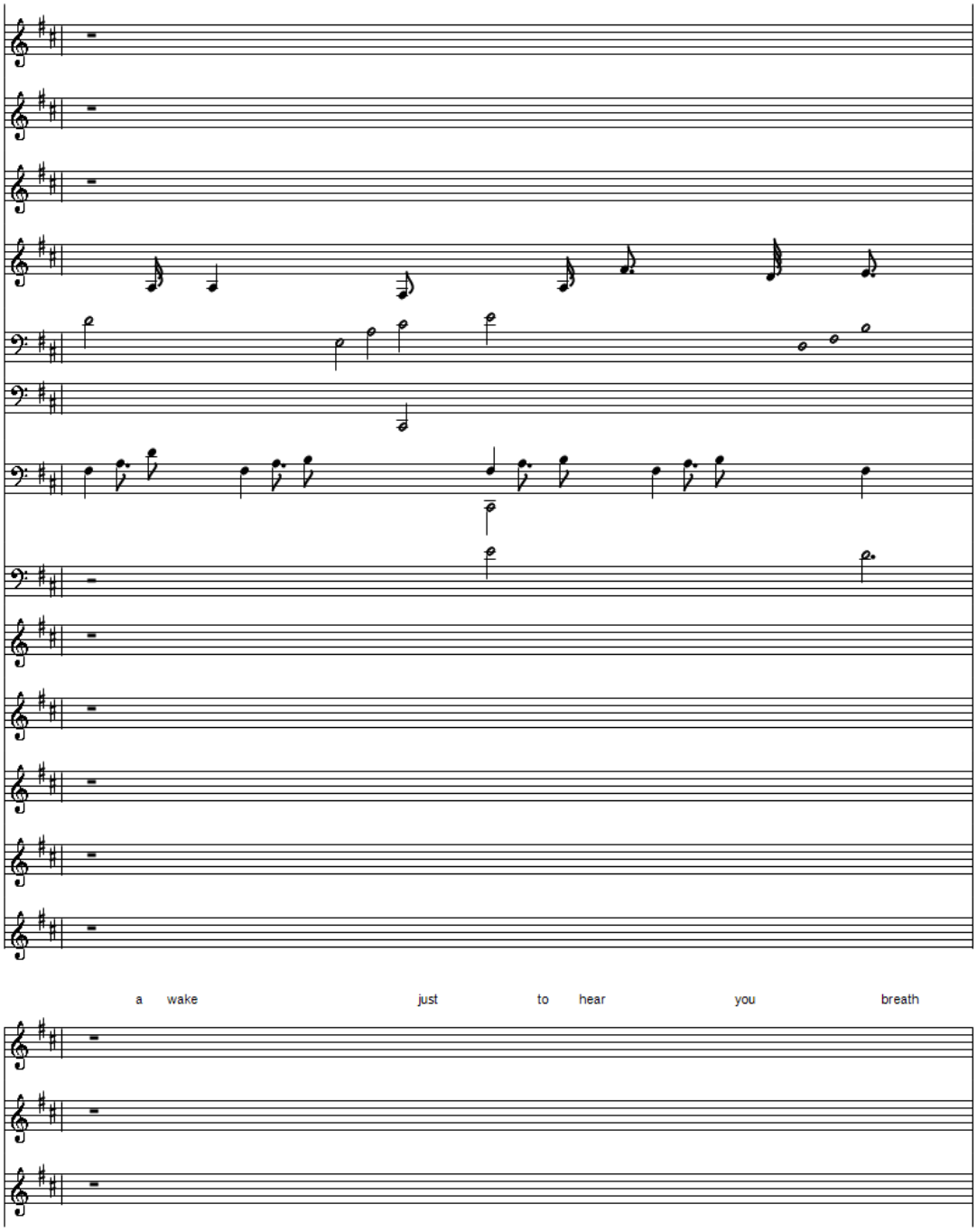
MIDI ファイルをウェブサイト midiworld.com[8]から 50 楽曲を用意した。
ジャンルは特に指定していない。
使用楽曲は以下のとおりである。

1. ALVIN_-_.mid
2. ATB_-_Sunset_Girl_(by_Zalan)_-_.mid
3. Abdelmoine_Alfa_-_Abdelmoine's_1st_symphony.mid
4. Abdelmoine_Alfa_-_Worthless.mid
5. Ace_of_Base_-_Xray.mid
6. Adson_John_-_Courtly_Masquing_Ayres.mid
7. Aerosmith_-_I_Don't_Wanna_Miss_a_Thing.mid
8. Albinoni_Tommaso_-_Adagio_in_G_min.mid
9. Alexandros_Mousafeiris_-_Sanchit's_Tune.mid
10. Allwood_Richard_-_Claro_Pascali_Gaudio.mid
11. Allwood_Richard_-_In_Nomine.mid
12. Animotion_-_Synth.mid
13. Arbeau_Thoinot_-_Branle_des_cheveaux.mid
14. Arbeau_Thoinot_-_Pavana.mid
15. Arcadelt_Jacob_-_Ave_Maria.mid
16. Arcadelt_Jacob_-_Io_dico_fra_noi.mid
17. Arcadelt_Jacob_-_La_Ingratitud.mid
18. Billingsley_-_Orchestration_Assignment.mid
19. Bomfunk_MCs_-_Uprocking_Beats.mid
20. Bon_Jovi_-_.mid
21. Casey_-_The_Wanderer.mid
22. Chicago_-_.mid
23. Chuck_Berry_-_.mid
24. Classics_IV_-_Spooky.mid
25. Classics_IV_-_Stormy.mid
26. Classics_IV_-_Traces_of_Love.mid

27. Craft_-_Windows_Startup_midi.mid
28. Debussy_-_Project.mid
29. Delmore_Brothers_-_.mid
30. Dvorak_midi-seq-by-Ong_Cmu_-_Humoresque_Devorak.mid
31. ELO_-_ELO_-_Alright.mid
32. Ensemble-_Isaac_Piano_-_Ensemble-_Isaac_Piano.mid
33. Ensemble-_Isaac_Piano_Week_4_with_structure_(Piano_Modified_110_Bpm)_-_(Ensemble-_Geo_Drum)_.mid
34. Ensemble-_Isaac_Piano_Week_4_with_structure_(Piano_Modified_110_Bpm)_-_(Ensemble-_Isaac_Piano).mid
35. Enya_-_Lothlorien.mid
36. Friedrich_Kiel_-_Bolero.mid
37. George_stone_of_Passport_Designs_-_Trip_through_the_grand_canyon(on_some_windows_pcs).mid
38. Hernan_Gomez_-_Es_Navidad.mid
39. IDC_-_IDC.mid
40. Joan_Jett_-_i_s_been_so_long.mid
41. K-On!!_-_lyntor.mid
42. Kaito_-_Cantarella.mid
43. Karuro_-_MySong3.mid
44. chinami_-_Unfinished.mid
45. chinami_-_[Free-scores.com]_haydn-joseph-string-quartet-major-viola-328.midi.mid
46. dax_-_FDAX.mid
47. gandija_-_fanfarria.mid
48. kemal_malovcic_-_.mid
49. kenneth_-_Project2-2-2.mid
50. kenneth_-_Project2-2.mid

また、一楽譜の冒頭は以下の通りである。




a wake just to hear you breath
3

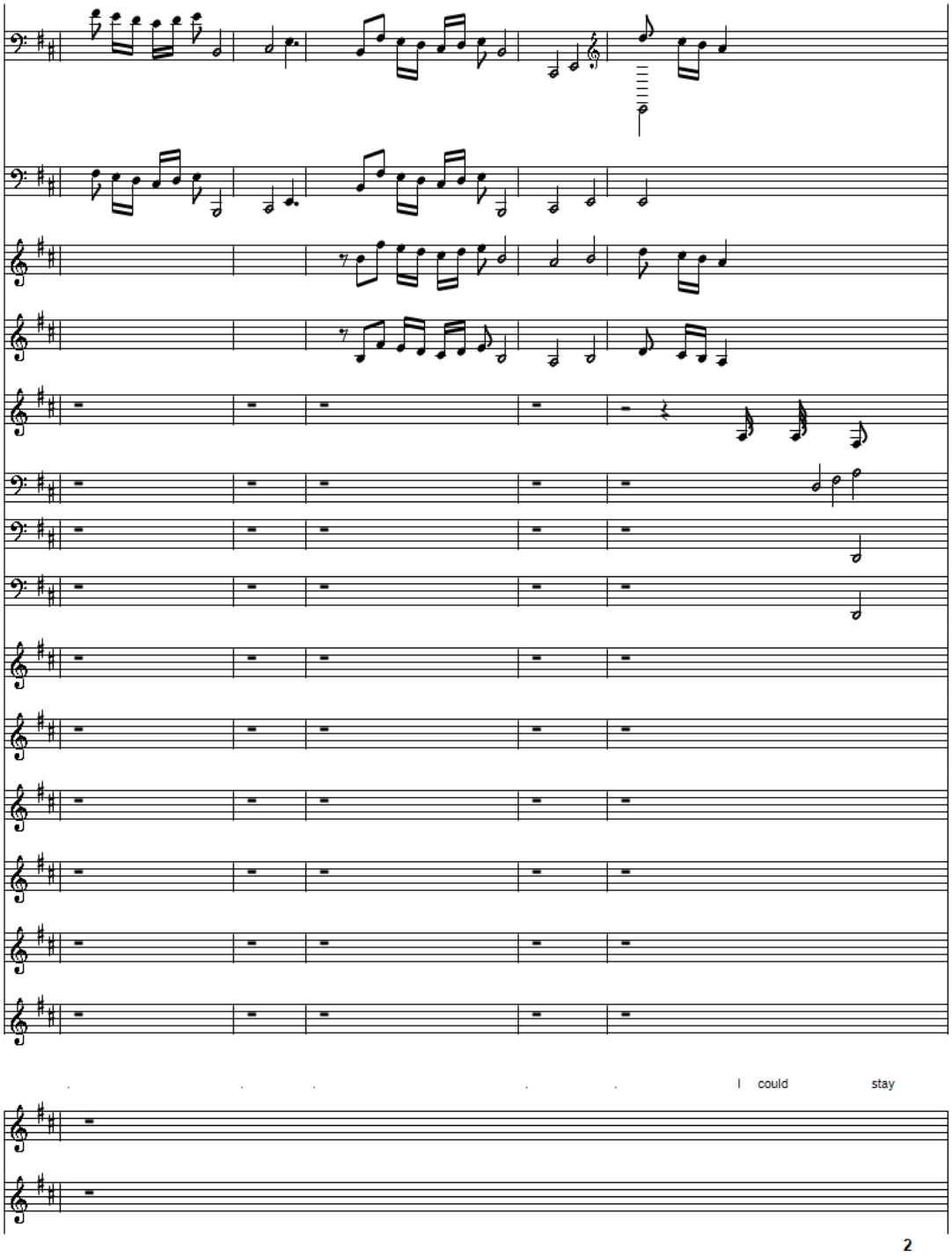


図 4.1.1 楽曲例 Aerosmith_-_I_Don't_Wanna_Miss_a_Thing.mid
（尚、画像化にはツール[9]を用いた）

ランダムに生成した曲線を背景が黒で曲線が白かつ 32x32 のサイズに収まるようにという条件のもと画像化したものを旋律概形とし、[8]より収集した、作曲された MIDI ファイルを横軸に時間、縦軸に音程とした背景が黒で一連の音符列を同じ音階ごとに横の線分で表したものを MIDI ファイルのトラック、楽器ごとに画像ファイルに変換し、それを 32x32 のサイズで切り分けたものを用意した。MIDI ファイルは 50 楽曲を用意した。MIDI ファイル 50 楽曲からは 32x32 の旋律のイメージファイルが 1453 ファイル生成された。これを用いて学習を進める。

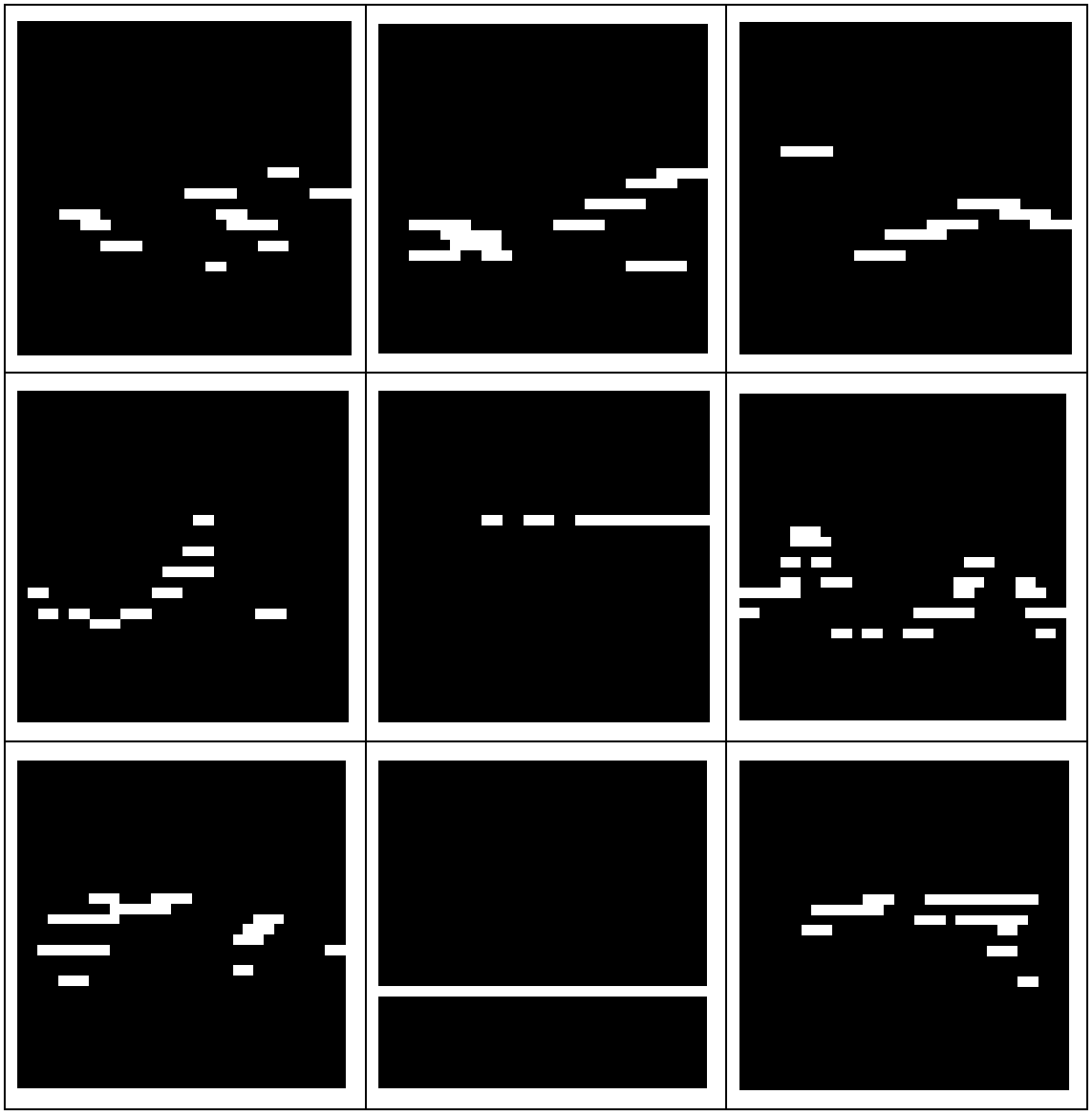


図 4.1.2 生成された MIDI ファイルの旋律イメージ一例

以上の条件で用意したランダムに生成した曲線のデータを Generator AB の入力とし実際の楽譜上の旋律を推定させる。Discriminator B には先程の推定したデータと MIDI ファイルより生成した 32x32 のサイズのデータのどちらかを入力しまた、MIDI ファイルより生成したデータの場合には教師信号に 1 を割り振る。ここでは教師信号 1 を MIDI ファイルより生成したデータという意味のラベルとする。ランダムに生成した曲線のデータにはそれぞれ教師信号に 1 を割り振る場合と Generator より生成されたデータであるラベルとして教師信号に 0 を割り振る場合のモデルを用意する。また、Generator AB により生成されたデータを Generator BA に Generator AB により推定された旋律を入力し旋律概形を出力するよう学習を行うこのときの旋律概形が Generator AB に入力された旋律概形と等しくなるよう学習を行う。この処理を行うことにより特定の旋律概形にユニークな旋律を得ることができる。また、Discriminator B は Generator AB により推定された旋律もしくは実際の楽譜上の旋律の判別のみを行うため、旋律概形が推定されたものもしくはあらかじめ用意されているランダムに生成した曲線のデータであることを判別する旋律概形のみを判別する Discriminator A を用意する。Discriminator A は Generator BA に実際の楽譜上の旋律を入力しそこから推定された旋律概形とランダムに生成した曲線のデータを入力し判別を行う。また、推定された旋律概形は入力された旋律に対してユニークである必要があるため Generator AB に推定された旋律概形を入力し推定された旋律と Generator BA に入力した旋律が等しくなるように学習をさせる。

4.2 実験環境

本研究システムでは JAIST 上の PC クラスタを使用した。
PC クラスタは

CPU ノード　48 台 : 1536Cores
GPU ノード（Tesla P100）8 台：256Cores + 16GPU
（Tesla K40）　4 台 : 40Cores + 8GPU

ノード内構成（CPU ノード）
CPU: Intel Xeon Gold 6130 2.1GHz (16Cores x2)
Memory: 64GB

ノード内構成（GPU ノード）
CPU: Intel Xeon Gold 6130 2.1GHz (16Cores x2)
GPU: NVIDIA Tesla P100 x2
Memory: 128GB

ノード内構成（GPU ノード）
CPU: Intel Xeon E5-2680v2 (10Cores x2)
GPU: NVIDIA Tesla K40 x2
Memory: 64GB

キュークラスは GPU-S を使用。
1[node]
1-2[GPU]
GPU: Tesla P100

開発言語は Python3.5
フレームワークは TensorFlow 1.4.0 (GPU)
GPGPU を扱うために cuda 9.0 を用いた。

その他、ライブラリは以下のとおりである。

MIDI ファイル取り扱い:	pretty_midi
画像への変換:	pillow
	opencv-python

旋律概形の生成には

g++ (GCC) 4.4.7 20120313 (Red Hat 4.4.7-17)

Copyright (C) 2010 Free Software Foundation, Inc.

OpenCV

を用いた。

実験パラメータは以下の通りである。

最適化手法

Adam	Learning rate	1e-4
	Beta1	0.90
	Beta2	0.999
	Epsilon	1e-8

最適化手法には確率的勾配降下法や、AdaGrad など様々存在するが収束がはやく、DCGAN などで多く用いられる Adam を今回は使用する。また、各パラメータは TensorFlow の規定値を使用している。今回は最適化手法については Adam の上位のパラメータで固定する。

Ganerator AB	LSTM Cell neuron	256	
	Full connection layer size	256	
	Input size	128 x 32	[batch size] x [pitch]
	Output size	128 x 32	
Ganerator BA	LSTM Cell neuron	256	
	Full connection layer size	256	
	Input size	128 x 32	[batch size] x [pitch]
	Output size	128 x 32	
Discriminator A	LSTM Cell neuron	256	
	Full connection layer size	256	
	Input size	128 x 32	[batch size] x [pitch]
	Output size	128	
Discriminator B	LSTM Cell neuron	256	
	Full connection layer size	256	
	Input size	128 x 32	[batch size] x [pitch]
	Output size	128	

一度の学習で入力は 128[batch size], 32[pitch], を 1 度の入力とする。これを 32 回繰り返した誤差から計算した重み誤差を合わせたものを 1 回の重みの更新とする。

この動作を繰り返す。

4.3 実験

実験では LSTM を適応した DiscoGAN にあらかじめ生成した 32x32 のサイズの曲線イメージ(以下 Data A) を変換しそれをインターネット上から収集した MIDI の旋律を連続する同音階をブロック化した画像を 32x32 に切り出したイメージ(Data B)に変換をする Generator を得ることを目標とする。(図 4.3)

図 4.3　Data B 例

4.3.1 実験 1

実験 1 では学習を 3 万回 128 バッチ行った。

入力データ、教師データはそれぞれ用意したものの中から無作為に 128 枚選び入力を行った。

学習開始時から学習終了時まで Data A から Data B をそれぞれの Generator, Discriminator の入力とした。

実験 1 結果

表 4.3.1

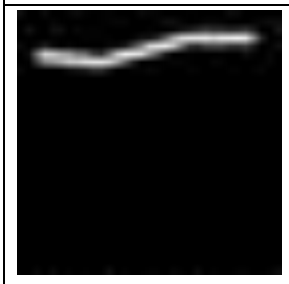


図 4.3.1.1

図 4.3.1.2

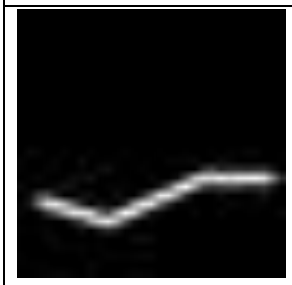


図 4.3.1.3

図 4.3.1.4

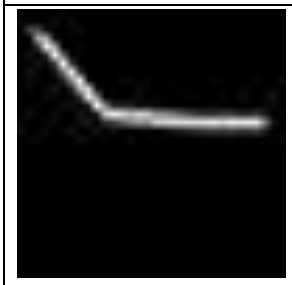


図 4.3.1.5

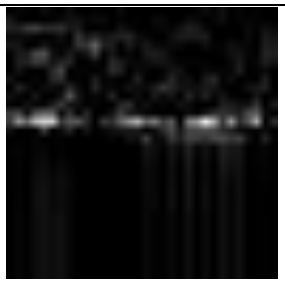


図 4.3.1.6

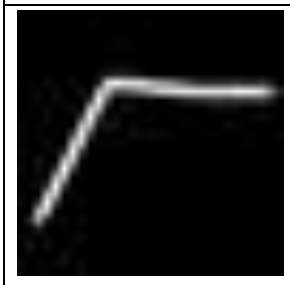


図 4.3.1.7

図 4.3.1.8

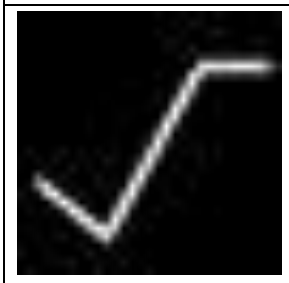


図 4.3.1.9

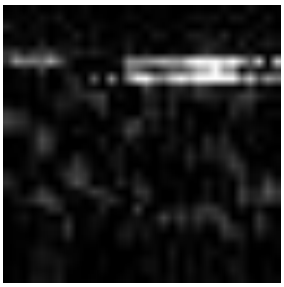


図 4.3.1.10

4.3.2 実験 1 の考察

実験 1 では Data A から Data B への変換ができていないように思われる。

考えられる理由として Data A から Data B へ変換するための特徴を Data A、Data B のみを用いた場合には発見できないと思われる。

実験 1 の改善案として学習を前半と後半の 2 部に分ける。

前半部では Data A から Data A を推定する学習を行い、Data A の特徴を取得する。

後半部は実験 1 と同じく Data A から Data B を推定する学習を行う。

実験 2 ではこの一連の処理を 3 万回行った。

前半部

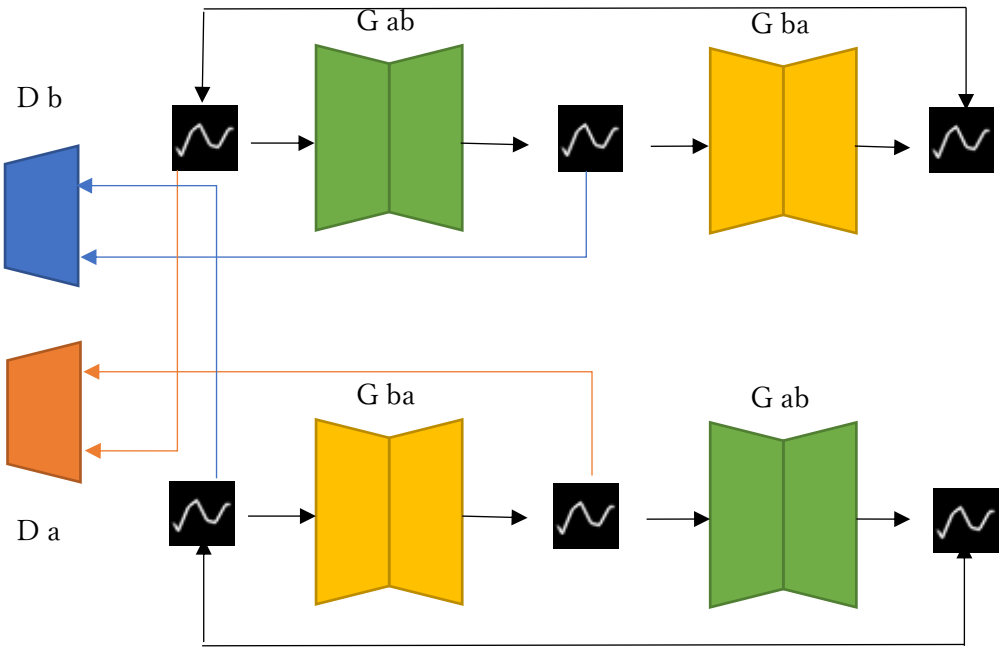


図 4.3.2.1 学習内容

後半部

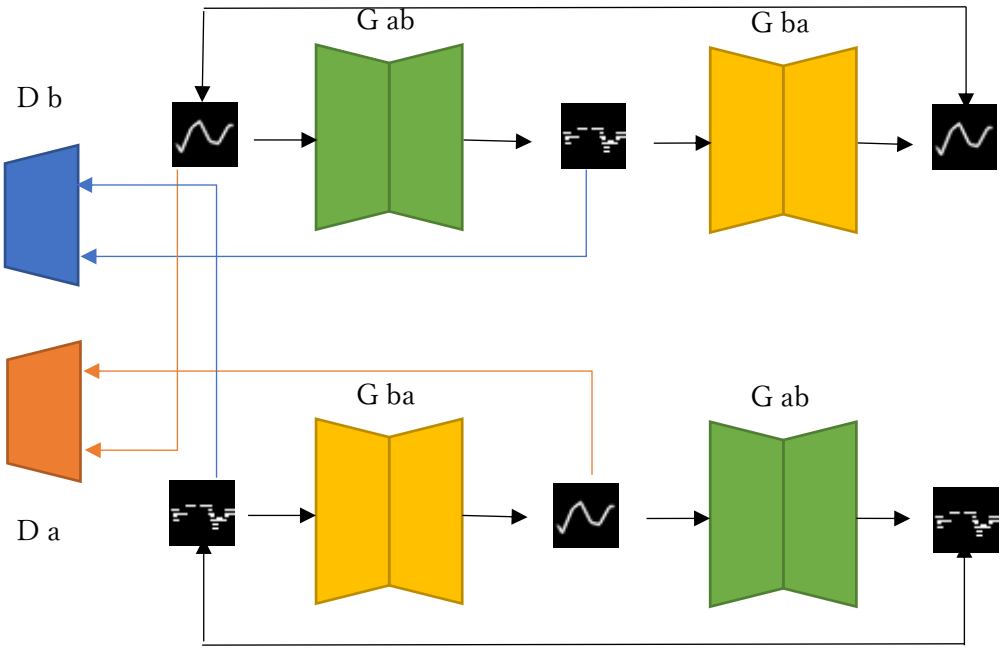
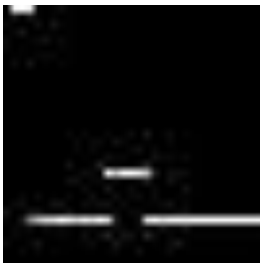
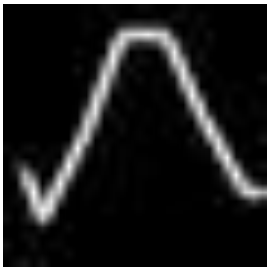
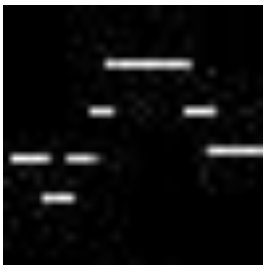


図 4.3.2.2 学習内容

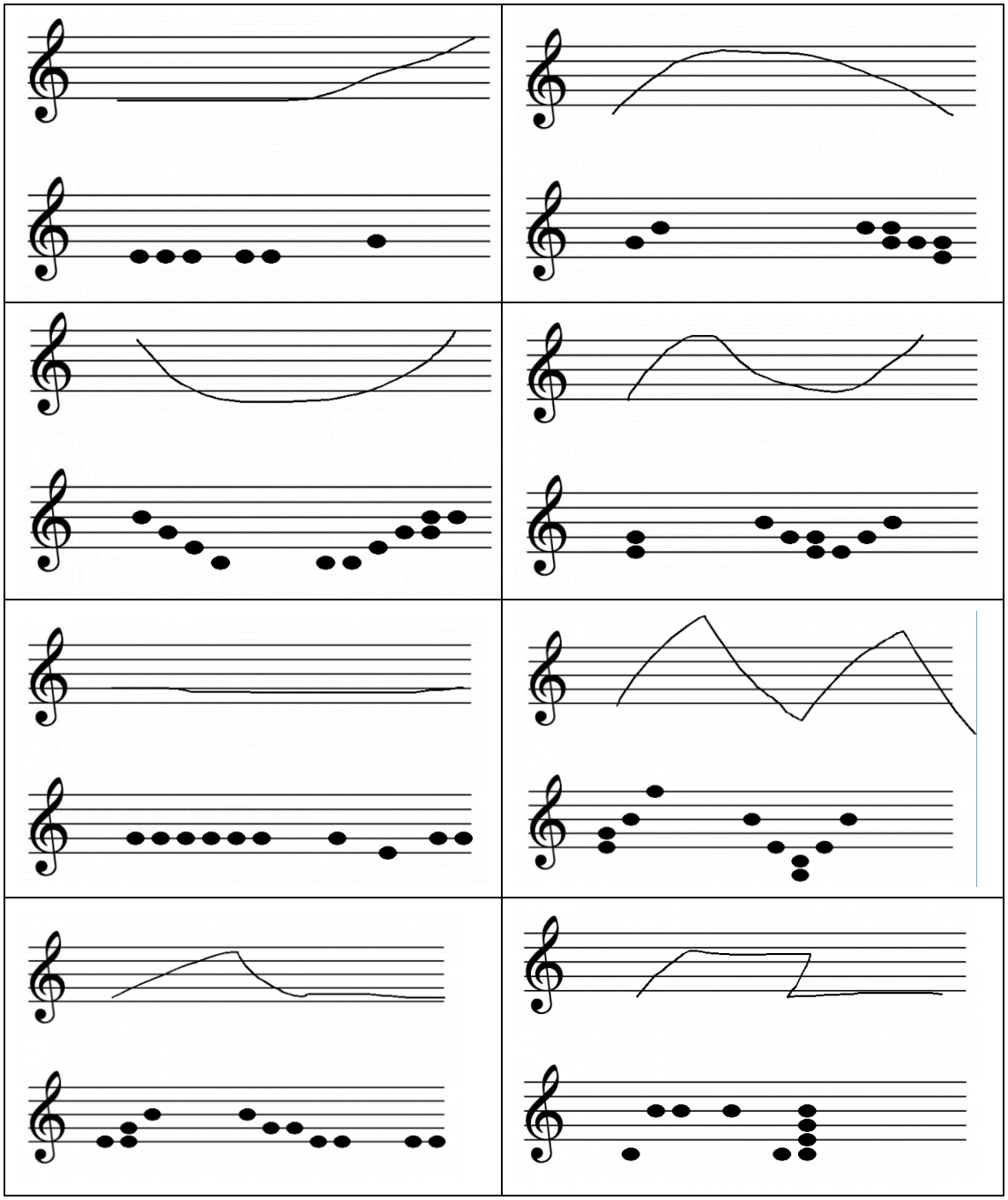
4.3.3 実験２　結果

実験２　結果

表 4.3.2

Data A	Data B
 図 4.3.2.1	 図 4.3.2.2
 図 4.3.2.3	 図 4.3.2.4
 図 4.3.2.5	 図 4.3.2.6
 図 4.3.2.7	 図 4.3.2.8
 図 4.3.2.9	 図 4.3.2.10

実行結果例



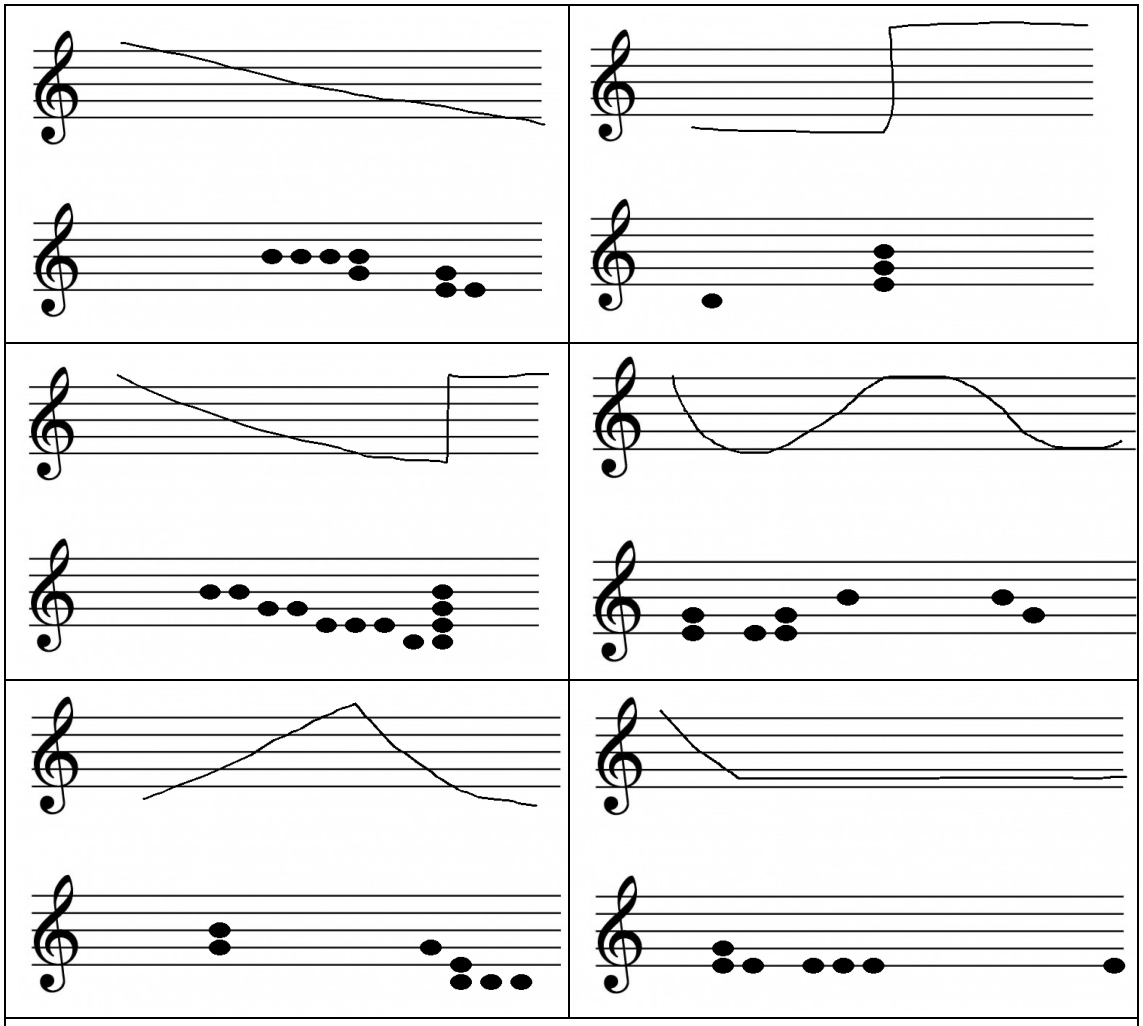


表 4.3 システム動作例 （上:旋律概形、下:推定旋律）

第5章　　おわりに

5.1 まとめ

F 値

F 値 は $\frac{2Recall \cdot Precision}{Recall+Precision}$ より導出される。

Precision(精度) は $\frac{TP}{TP+FP}$ TP はTrue positive、FP は False positive、

Recall(再現率) は $\frac{TP}{TP+FN}$ TP はTrue positive、FN は False negative である。

本研究では 1453 個の旋律の断片データを用意した[8]。そのデータを用いて評価を行う。

True positive は用意された旋律概形が用意された旋律概形と Discriminator に判別される回数で、False positive は用意された旋律概形が一度、旋律に変換され再度、旋律概形に変換したデータに対して用意された旋律概形であると判別された回数、False negative は 2 回の変換処理をされた旋律概形がそのものであると判別された回数である。

本研究では DiscoGAN を用いているため、Discriminator の判別が難しくなるような用意された旋律概形に近しい旋律概形の推定を行うことを目指す。よって F 値が 0.5 に近いほど用意された旋律概形に近しい旋律概形の推定を行えているといえる。

以下、本研究の結果である。

F 値	0.6894619845148602
Precision	0.6862308382987976
Recall	0.6927237026506194

旋律概形からそれを反映した旋律を出力できたと思われる。

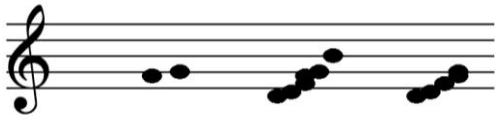
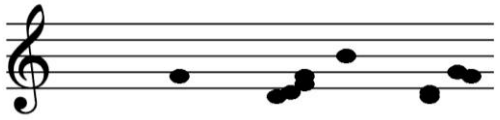
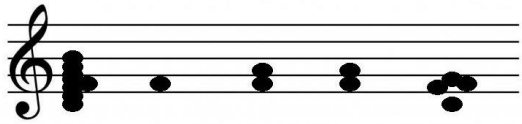
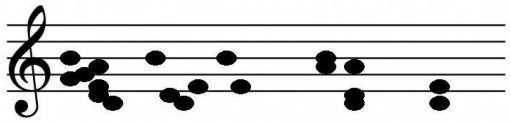
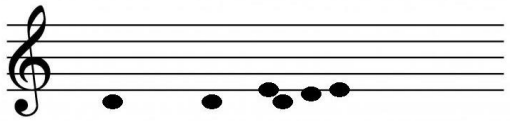
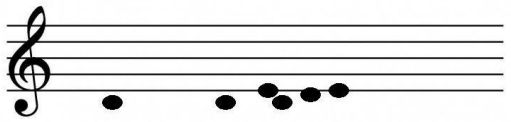
入力された旋律	推定された旋律
	
	
	

表 5.1　旋律から旋律概形を推定し旋律を再度推定しなおした実験結果

5.2 今後の課題

今回は短い旋律のみの推定を行ったが一連の旋律を推定するには短いという問題がある。

また、用意したデータセットに依存しておりユーザによってシステムの学習した旋律概形を大きく異なる旋律概形が入力された場合の推定が困難である。

謝辞

本研究を行うにあたり、未熟な私に対し、手厚くご指導いただきました東条敏教授には深く感謝致します。

また、審査員を引き受けていただきました、飯田弘之教授、白井清昭准教授、NGUYEN, Minh Le 准教授には、本研究に対し多くのご助言をいただき、深く感謝致します。

副テーマ指導教員である長谷川忍准教授には、わかりやすいご指導をしていただきました。厚く御礼申し上げます。

最後に、北陸先端科学技術大学院大学での学生生活を共に過ごし、共に切磋琢磨した友人、そして、生活面や精神面で支えてくれた家族へ心から感謝致します。

参考文献

[1] 土屋裕一 日本大学大学院総合基礎科学研究科地球情報数理科学専攻　修士論文
旋律概形を用いた作曲支援システムの研究　(2014)
[2] 北原鉄朗 音符を単位としない旋律編集のための旋律概形抽出手法 情報処理学会論文誌 Vol.54 No.4 1302-1307 (Apr. 2013)
[3] 北原鉄朗,・土屋裕一 (2014) 旋律概形を用いた作曲支援システム：ユーザビリティ実験の報告 情報処理学会 第 76 回全国大会 1R-2
[4] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio (Submitted on 10 Jun 2014)
Generative Adversarial Networks　https://arxiv.org/abs/1406.2661
[5] Alex Graves Department of Computer Science University of Toronto (5 Jun 2014)
Generating Sequences With Recurrent Neural Networks
https://arxiv.org/pdf/1308.0850.pdf
[6] Taeksoo Kim, Moonsu Cha, Hyunsoo Kim, Jung Kwon Lee, Jiwon Kim
(Submitted on 15 Mar 2017 (v1), last revised 15 May 2017 (this version, v2))
Learning to Discover Cross-Domain Relations with Generative Adversarial Networks
https://arxiv.org/abs/1703.05192
[7] Olof Mogren (Submitted on 29 Nov 2016)
C-RNN-GAN: Continuous recurrent neural networks with adversarial training
Accepted to Constructive Machine Learning Workshop (CML) at NIPS 2016 in Barcelona,
https://arxiv.org/abs/1611.09904
[8] midiworld.com
http://www.midiworld.com/files/
[9] MidiSheetMusic-2.6
https://sourceforge.net/projects/midisheetmusic/files/midisheetmusic/2.6/

付録　プログラムリスト

melodyToLine.py　　Midi からの旋律イメージファイルの生成

```
#!/usr/bin/python
#-*-coding:utf-8-*-

"""
   MIDI から旋律ブロックに変換するプログラム
   .mid => .png
"""

import pretty_midi
import sys
import os
import cv2
import numpy as np
import glob


def save_image(images, names, directory):
   if not os.path.isdir(directory):
      os.mkdir(directory)
   i = 0
   for image in images:
      image[image > 1.] = 1.
      image[image < 0.] = 0.

      image = image * 255.0
      pil_img = Image.fromarray(np.uint8(image))
      pil_img.save(directory + '/test' + str( i ) + "_" + names[i] + '.jpg')
      i += 1

def midiToBlock(file_name, pitch_max = 256):
   if not os.path.exists(file_name):
      return []
```

```
    midi_data = pretty_midi.PrettyMIDI(file_name)
    if midi_data is None:
        return None

    note_size = 0
    for track in midi_data.instruments:
        for note in track.notes:
            note_size = max(note_size, int(note.end))

    image = np.zeros((pitch_max, note_size, 3), np.uint8)
    for track in midi_data.instruments:
        # print(track)
        for note in track.notes:
            start = int(note.start)
            end = int(note.end)
            cv2.line(image, (start, note.pitch), (end + 1, note.pitch), (0xff, 0xff, 0xff), 1)
            # print(note.start, note.end, note.pitch )
            # print(note)
    return image

def midiOneTrack(file_name):
    midi_data = pretty_midi.PrettyMIDI(file_name)
    if midi_data is None:
        return None

    midi_out = pretty_midi.PrettyMIDI()
    program = pretty_midi.instrument_name_to_program('Cello')
    instrument = pretty_midi.Instrument(program=program)

    for track in midi_data.instruments:
        for note in track.notes:
            start = int(note.start)
            end = int(note.end)
            key = note.pitch
            note_out = pretty_midi.Note(velocity=127, pitch=key, start=start, end=end)
            instrument.notes.append(note_out)
```

```
            # print(start, end, key)
    midi_out.instruments.append(instrument)
    midi_out.write('sample1.mid')

# 旋律概形を切り出す
def midiToMelodyOutline(file_name, pitch_range=256, image_width=32, image_height=32,
note_scale=1, saver_directory='lines/'):

    if not os.path.isfile(file_name):
        return []

    try:
        midi_data = pretty_midi.PrettyMIDI(file_name)
        if midi_data is None:
            return []
    except OSError:
        print("Cannot read.")
        return []
    file_base_name = os.path.basename(file_name)

    octave = 12
    # pitch_scale = 1

    if not os.path.isdir(saver_directory):
        os.mkdir(saver_directory)
    save_count = 0
    result_images = []
    for track in midi_data.instruments:
        note_base_pos = 0
        line_images = []
        for _ in range(pitch_range // image_height + 1):
            line_images.append(np.zeros((image_width, image_height), np.uint8))
        for note in track.notes:
            note_start = note.start
            note_end = note.end
            note_pitch = note.pitch
            start = int(note_start * note_scale - note_base_pos)
```

```
        end = int(note_end * note_scale - note_base_pos)

        # pitch をどの領域に配置するかを計算
        pitch_level = int(note_pitch // octave)
        pitch_base = (pitch_level - 1) * octave
        note_pitch = note_pitch - pitch_base

        image = line_images[pitch_level]

        cv2.line(image, (start, note_pitch), (end + note_scale, note_pitch), (0xff, 0xff, 0xff), 1)
        if end + note_scale >= image_width:

            note_base_pos = int(note_start * note_scale)
            cv2.line(image, (start, note_pitch), (end + note_scale, note_pitch), (0xff, 0xff, 0xff),
1)
            cv2.imwrite(saver_directory + file_base_name + str(save_count) + '.png', image)
            result_images.append(image)

            save_count += 1
            line_images[pitch_level] = np.zeros((image_width, image_height), np.uint8)
            start = int(note_start * note_scale - note_base_pos)
            end = int(note_end * note_scale - note_base_pos)

        # print((start, end + note_scale, note_pitch, note_base_pos))
        cv2.line(image, (start, note_pitch), (end + note_scale, note_pitch), (0xff, 0xff, 0xff), 1)

    cv2.imwrite(saver_directory + file_base_name + str(save_count) + '.png', image)
    result_images.append(image)
    save_count += 1

  return result_images


def showMidi(file_name):
  # MIDI ファイルのロード
  midi_data = pretty_midi.PrettyMIDI(file_name)
  # トラック別で取得
```

```
    midi_tracks = midi_data.instruments
    #  トラック１のノートを取得
    notes = midi_tracks[0].notes
    for note in notes:
        # ベロシティー、ノートナンバー、
        # ノートオンタイム、ノートオフタイム
        # の順でノート情報が渡される
        print(note)

if __name__ == '__main__':
    argv = sys.argv
    if len(argv) < 2:
        exit(0)

    file_path_list = argv[1:]
    if argv[1] == '-d':
        dir_name = argv[2]
        if not '*' in argv[2]:
            dir_name = dir_name + '/*'
        file_path_list = glob.glob(dir_name)

    # showMidi(argv[1])
    for path_name in file_path_list:
        file_name = os.path.basename(path_name)
        # image = midiToBlock(file_name)
        # cv2.imwrite(file_name + '.png', image)
        print(path_name + str(' => ') + file_name)
        midiToMelodyOutline(path_name)
        # midiOneTrack(argv[1])
```

curve.cpp　　旋律概形を生成する

```
#include <stdlib.h>
#include <time.h>
#include <opencv2/opencv.hpp>


float randf( void )
{
   static int    call_counter = 0;
   if( call_counter == 0 )
      srand( ( unsigned int )clock() );
   call_counter ++;
   return ( float )rand() / RAND_MAX;
}

float randp( void )
{
   return randf();
}

// 横線生成
int randomLine( float* points_x, float* points_y, int point_num )
{
   float    x = 0;
   float    y = 0.90 * randp() + 0.05;
   float    l = 0;
   int    line_num = point_num - 1;
   float    l_scale = ( float )1 / line_num;

   for( int i = 0; i < line_num; i ++ )
   {
      //y = 0.90 * randp() + 0.05;
      y += 3 * l_scale * ( 2 * randp() - 1 );
      if( y >= 0.90 )
         y = 0.90;
      if( y <= 0.10 )
```

```
            y = 0.10;
        l = l_scale;

        points_x[ i ] = x;
        points_y[ i ] = y;

        x += l;
        if( x > 1 )
        {
            x = 0;

            i ++;
            points_x[ i ] = 1;
            points_y[ i ] = y;

            if( i >= line_num )
                break;
            i ++;
            points_x[ i ] = 0;
            points_y[ i ] = y;
        }
    }
    points_x[ line_num ] = 1;
    points_y[ line_num ] = y;

    return 0;
}

// ベジエ曲線
float BezierPont( float x1, float y1, float x2, float y2, float x )
{
    // ベジエ曲線を利用して補間する。
    // 3 次方程式は 2 分法を利用。
    const int   _loop_len_ = 8;
    float       s = 0.5f;
    float       t = 0.5f;
    float       ft = x;
```

```
    // 二分法
    for( int i = 0; i < _loop_len_; i ++ )
    {
        ft = ( 3.0f * s * s * t * x1 ) + ( 3.0f * s * t * t * x2 ) + ( t * t * t ) - x;
        if( fabs( ft ) < 1e-4f )
            break;
        if( ft < 0 )
            t += 1.0f / ( 4 << i );
        else
            t -= 1.0f / ( 4 << i );
        s = 1 - t;
    }
    return ( 3.0f * s * s * t * y1 ) + ( 3.0f * s * t * t * y2 ) + ( t * t * t );
}

int BezierCurve( float* point_x, float* point_y, int point_num, float x1, float y1, float x2, float
y2 )
{
    int       i;
    float     x;
    float     x_step = ( float )1 / point_num;
    for( i = 0, x = 0; i < point_num && x < 1; i ++, x += x_step )
    {
        point_x[ i ] = x;
        point_y[ i ] = BezierPont( x1, y1, x2, y2, x );
    }
    return 0;
}

int pointToImage( cv::Mat& image, const float* points_x, const float* points_y, int points_size,
float scale_x, float scale_y )
{
    int       cols = image.cols;
    int       rows = image.rows;
    float     x0 = 0;
    float     x1 = 0;
```

```
    float       y0 = 0;
    float       y1 = 0;

    for( int j = 0; j < rows; j ++ )
        for( int i = 0; i < cols; i ++ )
            image.data[ j * cols + i ] = 0x00;

    x0 = ( int )( scale_x * points_x[ 0 ] );
    y0 = ( int )( scale_y * points_y[ 0 ] );
    for( int i = 1; i < points_size; i ++ )
    {
        x1 = ( int )( scale_x * points_x[ i ] );
        y1 = ( int )( scale_y * points_y[ i ] );

        cv::line( image, cv::Point( x0, y0 ), cv::Point( x1, y1 ), cv::Scalar( 0xff, 0xff, 0xff ), 1, CV_AA );

        y0 = y1;
        x0 = x1;
    }

    return 0;
}

int main( int argc, char** argv )
{
    const int    picth_size     = 32;
    const int    beat_size    = 32;

    const int    image_cols    = beat_size;
    const int    image_rows     = picth_size;

    const int    point_num = 8;
    float       points_x[ point_num ] = { 0x00 };
    float       points_y[ point_num ] = { 0x00 };

    cv::Mat       image( image_cols, image_rows, CV_8U );
    // cv::Mat       gray_image( image_cols, image_rows, CV_8U );
```

```
    cv::Mat       show_image( image_cols, image_rows, CV_8U );

    srand( ( unsigned int )clock() );
    for( int i = 0; i < 0x1000; i ++ )
    {
        // ランダムな曲線
        randomLine( points_x, points_y, point_num );

        // // ベジエ曲線の生成
        // BezierCurve( points_x, points_y, point_num, randp(), randp(), randp(), randp() );

        // ベジェ曲線上の点から画像を生成
        pointToImage( image, points_x, points_y, point_num, image_cols, image_rows );

        // // 2 値化
        // cv::threshold( image, show_image, 150, 255, cv::THRESH_BINARY );

        // コピー
        image.copyTo( show_image );

        // 保存
        char    file_name[ 0x100 ] = { 0x00 };
        sprintf( file_name, "images/%d.png", i );
        cv::imwrite( file_name, show_image );

        printf( "Output: %s¥r", file_name );
    }
    printf( "¥nFinished.¥n" );

    return 0;
}

// int main( int argc, char** argv )
// {
//     const int    image_cols    = 64;
//     const int    image_rows     = 64;
//     const int    show_scale     = 4;
```

```
//     float       points[ image_cols ] = { 0x00 };

//     cv::Mat       image( image_cols, image_rows, CV_8U );
//      cv::Mat         show_image(  image_cols  *  show_scale,  image_rows  *  show_scale,
CV_8U );

//     // ベジエ曲線の生成
//     BezierCurve( points, image_cols, randp(), randp(), randp(), randp() );
//     // BezierCurve( points, image_cols, 0, 0, 1, 1 );

//     // ベジェ曲線上の点から画像を生成
//     pointToImage( image, points, image_cols, image_rows );

//     // リサイズ
//     cv::resize( image, show_image, cv::Size(), show_scale, show_scale );

//   // 表示
//     cv::namedWindow( "image1", CV_WINDOW_AUTOSIZE | CV_WINDOW_FREERATIO );
//     cv::imshow( "image1", show_image );
//     cv::waitKey( 0 );

//     return 0;
// }
```

melodyToLine.py　　曲線の旋律イメージを音階ごとにブロック化する

```
#!/usr/bin/python
#-*-coding:utf-8-*-

import numpy as np
import cv2


def melodyToLine(melodys, bias=4):
    lines = []
    for melody in melodys:
        line = np.zeros(melody.shape)

        prev_x = 0
        prev_y = 0
        for j in range(len(melody)):
            if melody[j][1] > 0.01:
                prev_y = j
        for i in range(1, len(melody[0])): #  一列検査
            y = prev_y
            for j in range(len(melody)):
                if melody[j][i] > 0.50:
                    y = j

            #  一定量を超えたので旋律概形を分離
            if abs(y - prev_y) > bias:
                cv2.line(line, (prev_x, prev_y), (i, prev_y), (0xff, 0xff, 0xff), 1)
                prev_y = y
                prev_x = i
        cv2.line(line, (prev_x, prev_y), (len(melody[0]), y), (0xff, 0xff, 0xff), 1)

        lines.append(line / 255.)
    return lines

def show():
    import DirectoryReader as dr
    curves = dr.DirectoryReader.readPictures('curves/')
```

```
    lines = melodyToLine(curves)

    index = 0
    show_image = np.c_[lines[index], curves[index]]
    cv2.imshow("MelodyLine", show_image)

    while True:
        # q が押されるまで画像を表示する
        key = cv2.waitKey(16)
        if key & 0xff == ord("q"):
            break

    cv2.destroyAllWindows()

if __name__ == '__main__':
    show()
```

lstm.py　　LSTM の実装

```
#!/usr/bin/python
#-*-coding:utf-8-*-

import tensorflow as tf
import numpy as np


class lstm:
    def __init__(self):
        pass

    def model(self, sess, inputs_size, output_size, step_size, learning_rate=0.01):
        self.sess = sess

        with tf.variable_scope("lstm_model1"):
            self.input_ph = tf.placeholder(tf.float32, [None, step_size, inputs_size])

            x2 = tf.unstack(tf.transpose(self.input_ph, perm=[1, 0, 2]))

            cell = tf.contrib.rnn.BasicLSTMCell(num_units=output_size)
            x3, states_op = tf.contrib.rnn.static_rnn(cell, x2, dtype=tf.float32)
            self.output_op = tf.transpose(tf.stack(x3), perm=[1, 0, 2])

            self.teach_ph = tf.placeholder(tf.float32, [None, step_size, output_size])

        self.loss_init()
        self.train_init(learning_rate)

        return self.output_op, self.train_op

    def model2(self, sess, input_ph, teach_ph, output_size, learning_rate=0.01):
        self.sess = sess

        with tf.variable_scope("lstm_model2"):
            self.input_ph = input_ph
```

```
        x2 = tf.unstack(tf.transpose(self.input_ph, perm=[1, 0, 2]))

        cell = tf.contrib.rnn.BasicLSTMCell(num_units=output_size)
        x3, states_op = tf.contrib.rnn.static_rnn(cell, x2, dtype=tf.float32)
        self.output_op = tf.transpose(tf.stack(x3), perm=[1, 0, 2])

        self.teach_ph = teach_ph

    if teach_ph is None:
        return self.output_op

    self.loss_init()
    self.train_init(learning_rate)

    return self.output_op, self.train_op

# output_op is inference's output_op, supervisor_ph is teaching signals.
def loss_init(self):
    output_op = self.output_op
    teach_ph  = self.teach_ph
    # Define losses.
    with tf.name_scope("lstm_loss"):
        square_error = tf.reduce_mean(tf.square(output_op - teach_ph))
        loss_op = square_error
        tf.summary.scalar("lstm_loss", loss_op)
        self.loss_op = loss_op
        return self.loss_op

def train_init(self, learning_rate):
    with tf.name_scope("lstm_training"):
        optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
        train_op = optimizer.minimize(self.loss_op)
        self.train_op = train_op
        return self.train_op

def forward(self, inputs):
```

```
        return self.sess.run(self.output_op, feed_dict = {self.input_ph: inputs})

    def train(self, inputs, teach):
        return self.sess.run(self.train_op, feed_dict =
                        {self.input_ph: inputs, self.teach_ph: teach})


def lstm_test():
    inputs_size = 32
    output_size = inputs_size
    step_size     = 32
    learning_rate = 0.01

    sign = []
    for i in range(step_size):
        m = np.zeros([inputs_size])
        m[i] = 1.
        sign.append(m)
    sign = np.reshape(np.array(sign), [-1, step_size, inputs_size])

    with tf.Session(config=tf.ConfigProto(allow_soft_placement=True)) as sess:
        l = lstm()
        l.model(sess, inputs_size, output_size, step_size, learning_rate)
        sess.run(tf.global_variables_initializer())

        outputs = []
        for i in range(100):
            output = l.forward(sign)
            outputs.append(output)

            l.train(sign, sign)

        print(outputs)

if __name__ == '__main__':
    lstm_test()
```

disco_lstm_gan.py LSTM を用いた DiscoGAN の実装

```
#!/usr/bin/python
#-*-coding:utf-8-*-

import tensorflow as tf
import lstm
import numpy as np
import os
import math
import io
import sys



class Disco_lstm_GAN:
   class DiscoGAN_lstm_Generator:
      def __init__(self, sess,
                         input_length, output_length,
                         hidden_length, sequence_length, batch_size=128, tag=''):
         self.class_name = 'DiscoGAN_lstm_Generator' + tag
         self.sess = sess

         self.input_length     = input_length
         self.output_length     = output_length
         self.hidden_length     = hidden_length
         self.batch_size     = batch_size
         self.sequence_length    = sequence_length

         self.reuse = False

      def __call__(self, inputs = None):
         input_length        = self.input_length
         output_length        = self.output_length
         hidden_length        = self.hidden_length
         # batch_size        = self.batch_size
         sequence_length       = self.sequence_length

         self.input_ph        = inputs
```

```
        with tf.variable_scope(self.class_name, reuse=self.reuse):
            if self.input_ph is None:
                self.input_ph = tf.placeholder(tf.float32,
                                    [None, sequence_length, input_length], name="inputs")
            self.lstm = lstm.lstm()
            tfv_hidden = self.lstm.model2(self.sess, self.input_ph, None, hidden_length)
            tfv_weight = tf.get_variable('weight',
                                         [hidden_length, output_length], tf.float32,
                                         tf.truncated_normal_initializer(stddev=0.02))
            tfv_bias = tf.get_variable('bias', [output_length], tf.float32, tf.zeros_initializer())
            # tfv_hidden = tf.matmul(tfv_hidden, tfv_weight)# + tfv_bias
            tfv_hidden = tf.map_fn(lambda tfv_h: tf.nn.relu(tf.matmul(tfv_h, tfv_weight) +
                                          tfv_bias), tfv_hidden)
            self.output_op = tfv_hidden

        self.variables = tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES,
                                           scope=self.class_name)
        self.reuse = True

        print(self.class_name + ' OK.')

        return self.output_op

class DiscoGAN_lstm_Discriminator:
    def __init__(self, sess, input_length, output_length, hidden_length, sequence_length,
                 batch_size=128, tag=''):
        self.class_name = 'DiscoGAN_lstm_Discriminator' + tag
        self.sess = sess

        self.input_length     = input_length
        self.output_length    = output_length
        self.hidden_length    = hidden_length
        self.batch_size       = batch_size
        self.sequence_length  = sequence_length

        self.reuse = False
```

```
    def __call__(self, inputs):
        # input_length        = self.input_length
        output_length       = self.output_length
        hidden_length       = self.hidden_length
        # batch_size        = self.batch_size
        # sequence_length       = self.sequence_length

        self.input_ph       = inputs

        with tf.variable_scope(self.class_name, reuse=self.reuse):
            self.lstm = lstm.lstm()
            tfv_hidden = self.lstm.model2(self.sess, self.input_ph, None, hidden_length)
            tfv_weight = tf.get_variable('weight', [hidden_length, output_length], tf.float32,
                                         tf.truncated_normal_initializer(stddev=0.02))
            tfv_bias = tf.get_variable('bias', [output_length], tf.float32, tf.zeros_initializer())
            # tfv_hidden = tf.matmul(tfv_hidden, tfv_weight) + tfv_bias
            tfv_hidden = tf.map_fn(lambda tfv_h: (tf.matmul(tfv_h, tfv_weight) + tfv_bias),
                                         tfv_hidden)
            self.output_op = tfv_hidden

        self.variables = tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES,
                    scope=self.class_name)
        self.reuse = True

        print(self.class_name + ' OK.')

        return self.output_op

    # GAN processes
    def __init__(self, session, input_length, generate_length, sequence_length,
learning_rate=1e-4, hidden_length=512, batch_size=128):
        self.sess = session

        self.learning_rate = learning_rate

        self.batch_size = batch_size
```

```
        self.judge_length = 1

        self.input_length = input_length
        self.generate_length = generate_length

        self.sequence_length = sequence_length

        # 旋律概形 => 旋律
        self.g_ab = self.DiscoGAN_lstm_Generator(self.sess, input_length, generate_length,
                        hidden_length, sequence_length, batch_size, tag ='_AB')

        # 旋律 => 旋律概形
        self.g_ba = self.DiscoGAN_lstm_Generator(self.sess, generate_length, input_length,
                        hidden_length, sequence_length, batch_size, tag ='_BA')

        # 旋律判定
        self.d_a = self.DiscoGAN_lstm_Discriminator(self.sess, input_length, self.judge_length,
                        hidden_length, sequence_length, batch_size, tag = '_A')

        # 旋律概形判定
        self.d_b = self.DiscoGAN_lstm_Discriminator(self.sess, generate_length,
                        self.judge_length, hidden_length, sequence_length, batch_size, tag = '_B')

        self.global_step = 0

        self.loss()

        # Saver make
        self.saver = tf.train.Saver()

        print('Ready to finish.')


    def loss(self):
        self.a_sign = tf.placeholder(tf.float32,
                shape=[self.batch_size, self.sequence_length, self.input_length], name='xg')
        self.b_sign = tf.placeholder(tf.float32,
```

```
            shape=[self.batch_size, self.sequence_length, self.generate_length], name='xr')

        # 生成器
        self.gi_ab = self.g_ab(self.a_sign)
        self.gi_ba = self.g_ba(self.b_sign)

        self.generator = self.gi_ab

        # 生成データ用
        self.dg_a = self.d_a(self.gi_ba)
        self.dg_b = self.d_b(self.gi_ab)

        # 実データ用
        self.dr_a = self.d_a(self.a_sign)
        self.dr_b = self.d_b(self.b_sign)

        # データのエンコードとデコード
        self.gaba = self.g_ba(self.gi_ab)
        self.gbab = self.g_ab(self.gi_ba)

        # 生成データを実データと認識させる
        tf.add_to_collection('g_losses_a', tf.reduce_mean(tf.nn.softplus(-self.dg_a)))
        tf.add_to_collection('g_losses_b', tf.reduce_mean(tf.nn.softplus(-self.dg_b)))

        # 実データを実データと認識させる
        tf.add_to_collection('d_losses_a', tf.reduce_mean(tf.nn.softplus(-self.dr_a)))
        tf.add_to_collection('d_losses_b', tf.reduce_mean(tf.nn.softplus(-self.dr_b)))

        # 生成データを実データと認識させない
        tf.add_to_collection('d_losses_a', tf.reduce_mean(tf.nn.softplus(self.dg_a)))
        tf.add_to_collection('d_losses_b', tf.reduce_mean(tf.nn.softplus(self.dg_b)))

        # データのエンコードとデコード
        tf.add_to_collection('g_losses_aba', tf.reduce_mean(tf.square(self.a_sign - self.gaba)))
        tf.add_to_collection('g_losses_bab', tf.reduce_mean(tf.square(self.b_sign - self.gbab)))

        g_a_loss = tf.add_n(tf.get_collection('g_losses_a'), name = 'total_g_a_loss')
```

```
g_b_loss = tf.add_n(tf.get_collection('g_losses_b'), name = 'total_g_b_loss')
d_a_loss = tf.add_n(tf.get_collection('d_losses_a'), name = 'total_d_a_loss')
d_b_loss = tf.add_n(tf.get_collection('d_losses_b'), name = 'total_d_b_loss')
gaba_loss = tf.add_n(tf.get_collection('g_losses_aba'), name = 'total_g_losses_aba')
gbab_loss = tf.add_n(tf.get_collection('g_losses_bab'), name = 'total_g_losses_bab')

# 学習する変数を集める
g_ab_vars = self.g_ab.variables
g_ba_vars = self.g_ba.variables

d_a_vars = self.d_a.variables
d_b_vars = self.d_b.variables

optimizer = tf.train.AdamOptimizer(learning_rate = self.learning_rate)

self.g_ab_optim_op = optimizer.minimize(g_b_loss,
                                var_list = [d_b_vars, g_ba_vars, g_ab_vars])
self.g_ba_optim_op = optimizer.minimize(g_a_loss,
                                var_list = [d_a_vars, g_ab_vars, g_ba_vars])

self.d_a_optim_op = optimizer.minimize(d_a_loss, var_list = [d_a_vars, g_ab_vars])
self.d_b_optim_op = optimizer.minimize(d_b_loss, var_list = [d_b_vars, g_ba_vars])

self.gaba_optim_op = optimizer.minimize(gaba_loss, var_list = [g_ab_vars, g_ba_vars])
self.gbab_optim_op = optimizer.minimize(gbab_loss, var_list = [g_ba_vars, g_ab_vars])

self.losses = {
            self.g_ab: g_a_loss, self.g_ba: g_b_loss,
            self.d_a: d_a_loss, self.d_b: d_b_loss,
            self.gaba: gaba_loss, self.gbab: gbab_loss}

ops = [self.g_ab_optim_op, self.g_ba_optim_op, self.d_a_optim_op, self.d_b_optim_op,
       self.gaba_optim_op, self.gbab_optim_op]
with tf.control_dependencies(ops):
   self.train_op = tf.no_op(name='train')

self.ops = [self.train_op, self.losses[self.g_ab], self.losses[self.g_ba],
```

```
                    self.losses[self.d_a], self.losses[self.d_b], self.losses[self.gaba],
                    self.losses[self.gbab]]
        return self.losses

    # input は inputs[バッチサイズ][シーケンスサイズ][入力サイズ]
    def forward(self, inputs):
        return self.sess.run(self.generator, feed_dict={self.a_sign: inputs})

    def train(self, inputs, real_data):
        self.global_step += 1

        param = {
            self.a_sign: inputs,
            self.b_sign: real_data
        }
        ops = self.ops
        _, g_ab_loss_value, g_ba_loss_value, d_a_loss_value, d_b_loss_value, _1, _2 =
self.sess.run(ops, feed_dict=param)
        g_loss_value = (g_ab_loss_value + g_ba_loss_value) / 2
        d_loss_value = (d_a_loss_value + d_b_loss_value) / 2
        return g_loss_value, d_loss_value

    def load(self, directory='./', name='discogan.ckpt'):
        ckpt = tf.train.get_checkpoint_state(directory)
        if ckpt:
            self.saver.restore(self.sess, directory + name)
            print(name, ' loaded')
            return True
        print('I could not load ', name)
        return False

    def save(self, directory='./', name='discogan.ckpt', global_step=None):
        if global_step is None:
            self.saver.save(self.sess, directory + name)
        else:
            self.saver.save(self.sess, directory + name, global_step=global_step)
        print('Parameters saved.')
```

DirectoryReader.py　　画像ファイルとして保存した旋律、旋律概形の読み込み

```
#!/usr/bin/python
#-*-coding:utf-8-*-

import os
import cv2
import numpy as np

class DirectoryReader:
    def __init__(self):
        pass

    @staticmethod
    def readPictures(directory_name, ext = ['jpg', 'png', 'gif', 'bmp']):
        files = os.listdir(directory_name)
        pic = []
        for file in files:
            if file[-3:] in ext:
                image = cv2.imread(directory_name + file, cv2.IMREAD_GRAYSCALE)
                arr = np.array(image)
                arr = arr / 255.
                pic.append(arr)
        return pic
```

train.py　　学習ルーチン

```
#!/usr/bin/python
#-*-coding:utf-8-*-

import tensorflow as tf
import numpy as np
from PIL import Image
import os
import math
import io
import sys
import time
import glob
import gc
import DirectoryReader as dr
import random
import disco_lstm_gan
import progress_bar
import melodyToLine as m2l


def save_image(images, names, directory):
    if not os.path.isdir(directory):
        os.mkdir(directory)
    i = 0
    for image in images:
        image[image > 1.] = 1.
        image[image < 0.] = 0.

        image = image * 255.0
        pil_img = Image.fromarray(np.uint8(image))
        pil_img.save(directory + '/test' + str( i ) + "_" + names[i] + '.jpg')
        i += 1

def file_alldel(directory):
    file_list = glob.glob(directory + "/*")
    for f in file_list:
```

```
        os.remove(f)

def padding(a, length):
    pad_width = [length - len(a), 0]
    return np.pad(a, pad_width, 'constant', constant_values=0.)

def main(argv):

    image_directory = "wave_test"
    saver_directory = 'model/'

    freq_length       = 32
    input_length      = 32
    output_length     = input_length
    batch_size        = 128
    hidden_length     = 256
    # learning_rate     = 0.01
    learning_rate     = 1e-4

    loop_num = 30000
    save_num = 10000

    curves = dr.DirectoryReader.readPictures('curves/')
    lines  = dr.DirectoryReader.readPictures('lines/')

    curve_index = 0
    line_index = 0

    pgb_flag = False

    with tf.Session() as sess:
        gan = disco_lstm_gan.Disco_lstm_GAN(sess,
                input_length, output_length, freq_length,
                learning_rate=learning_rate,                    hidden_length=hidden_length,
batch_size=batch_size)

        if not gan.load(saver_directory, 'wave.ckpt'):
```

```
    sess.run(tf.global_variables_initializer())

if not os.path.isdir(saver_directory):
    os.mkdir(saver_directory)

# Pre-Train
for loop_counter in range(loop_num + 1):
    if pgb_flag:
        # Progress Bar draw.
        pgb = progress_bar.ProgressBar(save_num)
        if loop_counter % save_num > 0:
            pgb.set(loop_counter % save_num)
            pgb.draw()
        else:
            print('')

    # train
    curve = []
    for _ in range(batch_size):
        curve_index = random.randint(0, len(curves) - 1)
        curve.append(curves[curve_index])
    melody_block = m2l.melodyToLine(curve)

    # train
    g_loss_value, d_loss_value = gan.train(melody_block, melody_block)

    # test
    if loop_counter % save_num == 0:
        # Print loss
        print(
            'step:' + str(loop_counter) + '¥n' + ¥
            '¥t' + 'g_loss:' + str(g_loss_value) + '¥n' + ¥
            '¥t' + 'd_loss:' + str(d_loss_value) + '¥n' + ¥
            '¥t' + 'abs(dg)loss:' + str(abs(d_loss_value - g_loss_value))
        )

        # test
```

```
        generated = gan.forward(melody_block)

        # サンプルと生成画像を並べる
        # 画像をくっつけて入力と出力の対応関係を確認
        label_list = []
        output_images = []
        c = 0
        for i, j in zip(curve, generated):
            image_temp = np.c_[np.r_[i, j], np.r_[j, i]]
            output_images.append(image_temp)
            label_list.append(str(c))
            c += 1

        image_directory_temp = image_directory
        file_alldel(image_directory_temp)
        save_image(output_images, label_list, image_directory_temp)

        # save
        gan.save(saver_directory, 'wave.ckpt')

# Train
for loop_counter in range(loop_num + 1):
    if pgb_flag:
        # Progress Bar draw.
        pgb = progress_bar.ProgressBar(save_num)
        if loop_counter % save_num > 0:
            pgb.set(loop_counter % save_num)
            pgb.draw()
        else:
            print('')

    # train
    curve = []
    line = []
    for _ in range(batch_size):
        curve_index = random.randint(0, len(curves) - 1)
        curve.append(curves[curve_index])
```

```
        line_index = random.randint(0, len(lines) - 1)
        line.append(lines[line_index])
    melody_block = m2l.melodyToLine(curve)

    # train
    g_loss_value, d_loss_value = gan.train(melody_block, line)

    # test
    if loop_counter % save_num == 0:
        # Print loss
        print(
            'step:' + str(loop_counter) + '¥n' + ¥
            '¥t' + 'g_loss:' + str(g_loss_value) + '¥n' + ¥
            '¥t' + 'd_loss:' + str(d_loss_value) + '¥n' + ¥
            '¥t' + 'abs(dg)loss:' + str(abs(d_loss_value - g_loss_value))
        )

        # test
        generated = gan.forward(melody_block)

        # サンプルと生成画像を並べる
        # 画像をくっつけて入力と出力の対応関係を確認
        label_list = []
        output_images = []
        c = 0
        for i, j in zip(curve, generated):
            image_temp = np.c_[np.r_[i, j], np.r_[j, i]]
            output_images.append(image_temp)
            label_list.append(str(c))
            c += 1

        image_directory_temp = image_directory
        file_alldel(image_directory_temp)
        save_image(output_images, label_list, image_directory_temp)

        # save
        gan.save(saver_directory, 'wave.ckpt')
```

```
        sess.close()

if __name__ == '__main__':
    main(sys.argv)
```

gui.py　　実験用 GUI

```
#!/usr/bin/python
#-*-coding:utf-8-*-

import cv2
import numpy as np
from PIL import Image
import sys
import melody
import tensorflow as tf

def maxper(x, axis = None):
    per = np.max(x)
    if per > 0:
        zscore = x / per
    else:
        zscore = x
    return zscore

class GuiMain:
    def __init__(self):
        self.score_image = cv2.imread("melody_line.jpg")

        self.window_image = self.score_image.copy()

        # ウィンドウ
        cv2.namedWindow("Melody")
        cv2.namedWindow("debug")

        # マウスイベント時に関数 mouse_event の処理を行う
        cv2.setMouseCallback("Melody", self.mouse_event)

        # 旋律概形
        self.melody_points = []
        self.melody_point_temp = []

        # クリック
```

```
        self.lbutten = False
        self.mouse_pos = None

        # 変換器
        self.l2m_flag = True
        if self.l2m_flag:
            self.l2m = melody.LineToMelody()

        self.note_height = 23

        self.input_score_y = 100 - self.note_height * 1
        self.input_score_x = 128

        self.score_y = 338 - self.note_height * 1
        # self.score_y = 361 - self.note_height * 1

        self.l2m_iosize = (32, 32)
        self.batch_szie = 128

        self.melody_image_one = np.zeros(self.l2m_iosize)

        self.show_flag = False

    def clear(self):
        self.melody_points = []

    # マウスイベント時に処理を行う
    def mouse_event(self, event, x, y, flags, param):
        # 左クリック
        if event == cv2.EVENT_LBUTTONDOWN:
            self.lbutten = True
        elif event == cv2.EVENT_LBUTTONUP:
            self.lbutten = False

        # # 右クリック + Shift キーで緑色のテキストを生成
        # elif event == cv2.EVENT_RBUTTONUP and flags & cv2.EVENT_FLAG_SHIFTKEY:
```

```
        # 右クリック
        if event == cv2.EVENT_RBUTTONDOWN:
            pass
        elif event == cv2.EVENT_RBUTTONUP:
            pass

        # 最後のマウス位置
        self.mouse_pos = (x, y)
        # print(self.mouse_pos, end='¥r')

        # イベント
        self.event()

    def event(self):
        if self.lbutten:
            self.melody_point_temp.append(self.mouse_pos)
            self.show_flag = False
        else:
            self.melody_points.append(self.melody_point_temp)
            self.melody_point_temp = []

            min_y = self.input_score_y
            max_y = self.input_score_y + self.note_height * 7 # 5 + 2
            min_x = self.input_score_x
            max_x = self.input_score_x + 600
            self.melody_range_x = (min_x, max_x)
            self.melody_range_y = (min_y, max_y)

            self.melody_image_one = np.zeros(self.l2m_iosize)
            for points in self.melody_points:
                if len(points) == 0:
                    continue
                p0 = points[0]
                x = (p0[0] - min_x) / (max_x - min_x) * (self.l2m_iosize[0] - 1)
                y = (p0[1] - min_y) / (max_y - min_y) * (self.l2m_iosize[1] - 1)
                index_x, index_y = int(x + 0.5), int(y + 0.5)
                for p0 in points[1:]:
```

```
                index_px = index_x
                index_py = index_y
                x = (p0[0] - min_x) / (max_x - min_x) * (self.l2m_iosize[0] - 1)
                y = (p0[1] - min_y) / (max_y - min_y) * (self.l2m_iosize[1] - 1)
                index_x, index_y = int(x + 0.5), int(y + 0.5)
                cv2.line(self.melody_image_one, (index_px, index_py), (index_x, index_y), (0xff,
0xff, 0xff), 1)

            self.show_flag = True

    def showMelodyOutline(self):
        line_width = 2
        for points in self.melody_points:
            for p0, p1 in zip(points[:-1], points[1:]):
                cv2.line(self.window_image, p0, p1, (0x00, 0x00, 0x00), line_width)
        points = self.melody_point_temp
        for p0, p1 in zip(points[:-1], points[1:]):
            cv2.line(self.window_image, p0, p1, (0x00, 0x00, 0x00), line_width)

    def showNote(self, image, x, y):
        angle = 30
        line_height = self.note_height
        line_width = 20
        line_base_y = self.score_y
        # line_base_y = self.input_score_y
        line_left = self.input_score_x

        x = x * line_width    + line_left
        y = y * line_height + line_base_y
        cv2.ellipse(image, (int(x + 0.5), int(y + 0.5)), (14, 10), 0, 0 + angle, 360 + angle, (0, 0, 0),
-1)

    #  音符の描画
    def showNotes(self):
        if not self.show_flag:
            return
```

```
# image を分割するサイズ
image_step_x = 2
image_step_y = 4

# notes = self.melody_points
self.melody_image = []
melody_image_temp = self.melody_image_one.copy() / 255.
for _ in range(self.batch_szie):
    self.melody_image.append(melody_image_temp.copy())
if self.l2m_flag:
    images = self.l2m.run(self.melody_image, image_step_x, image_step_y)
    if len(images) > 0:
        images = images[0]
    images = maxper(images)
    # images = np.fliplr(images)
else:
    images = melody_image_temp.copy()
bias = 0.50
notes = []
for x in range(0, self.l2m_iosize[0], image_step_x):
    for y in range(0, self.l2m_iosize[1], image_step_y):
        p = 0
        for yt in range(image_step_y):
            p = max(images[y + yt][x], p)
        for xt in range(image_step_x):
            p = max(images[y][x + xt], p)
        if p > bias:
            notes.append((x, y / image_step_y))
# if len(list(notes)) == 0:
#     return
for p in notes:
    self.showNote(self.window_image, p[0], p[1])

# note debug
# for i in range(32):
#     self.showNote(self.window_image, i, i)
```

```
        debug_image = np.c_[melody_image_temp, images]
        debug_image = cv2.resize(debug_image, None, fx=4, fy=4)
        cv2.imshow("debug", debug_image)

    def show(self):
        while True:
            # q が押されるまで画像を表示する
            key = cv2.waitKey(16)
            if key & 0xff == ord("q"):
                break

            # c で初期化
            if key & 0xff == ord('c'):
                self.clear()

            self.window_image = self.score_image.copy()
            self.showMelodyOutline()
            self.showNotes()

            cv2.imshow("Melody", self.window_image)

        cv2.destroyAllWindows()

def main(argv):
    win = GuiMain()
    win.show()

if __name__ == '__main__':
    main(sys.argv)
```