AI（機械学習）音楽プログラミング

第９回

Magentaで学ぶ
機械学習開発

Magenta
独自AI作曲モデルの作成

最終課題となりますので
作成＆要提出

最終課題

提出ファイル

・melody_rnn_model.py

ご自身のモデルを記述した（コピーして提出。オリジナルを削除しない事）

・ご自身の名前.mag

ご自身の名前をファイル名に使用した学習済みデータ(.magファイル）

の２ファイルです。

Magenta
のプログラム解説

Magenta 学習データ作成の流れ=使用しているMagenta Pythonファイルの流れ

１・midiファイルの用意

midiファイルを作成し任意のフォルダーへ。配布したmidiデータが入ったフォルダーを使用してください。

２・NoteSequence（tfrecord）の作成

midiをMagentaで扱える様にNoteSequenceというデータ（TensorFlowのレコードファイル）に変換します。

３・トレーニングデータとテストデータ の作成

任意の割合でトレーニングデータとテストデータ を作成します。

４・トレーニング

音楽データを学習させます。

５・テスト

学習検証のためテストデータでテストをします（解説しますが、現在テストはできない様です）

６・tensorboardで学習の確認

学習の結果を視覚的に確認します。

７・音楽生成

学習させたデータを使用して音楽生成を行なってみましょう。

８・Bundleファイル（.magファイル）作成

学習済みデータとしてBundleファイルを作成します。

Magenta 学習データ作成の流れ=使用しているMagenta Pythonファイルの流れ

Melody RNNを例に使用されるPythonファイルを確認

３・トレーニングデータとテストデータ の作成 melody_rnn_create_dataset.py

任意の割合でトレーニングデータとテストデータ を作成します。

４・トレーニング melody_rnn_train.py

音楽データを学習させます。

５・テスト melody_rnn_train.py

学習検証のためテストデータでテストをします。

７・音楽生成 melody_rnn_generate.py

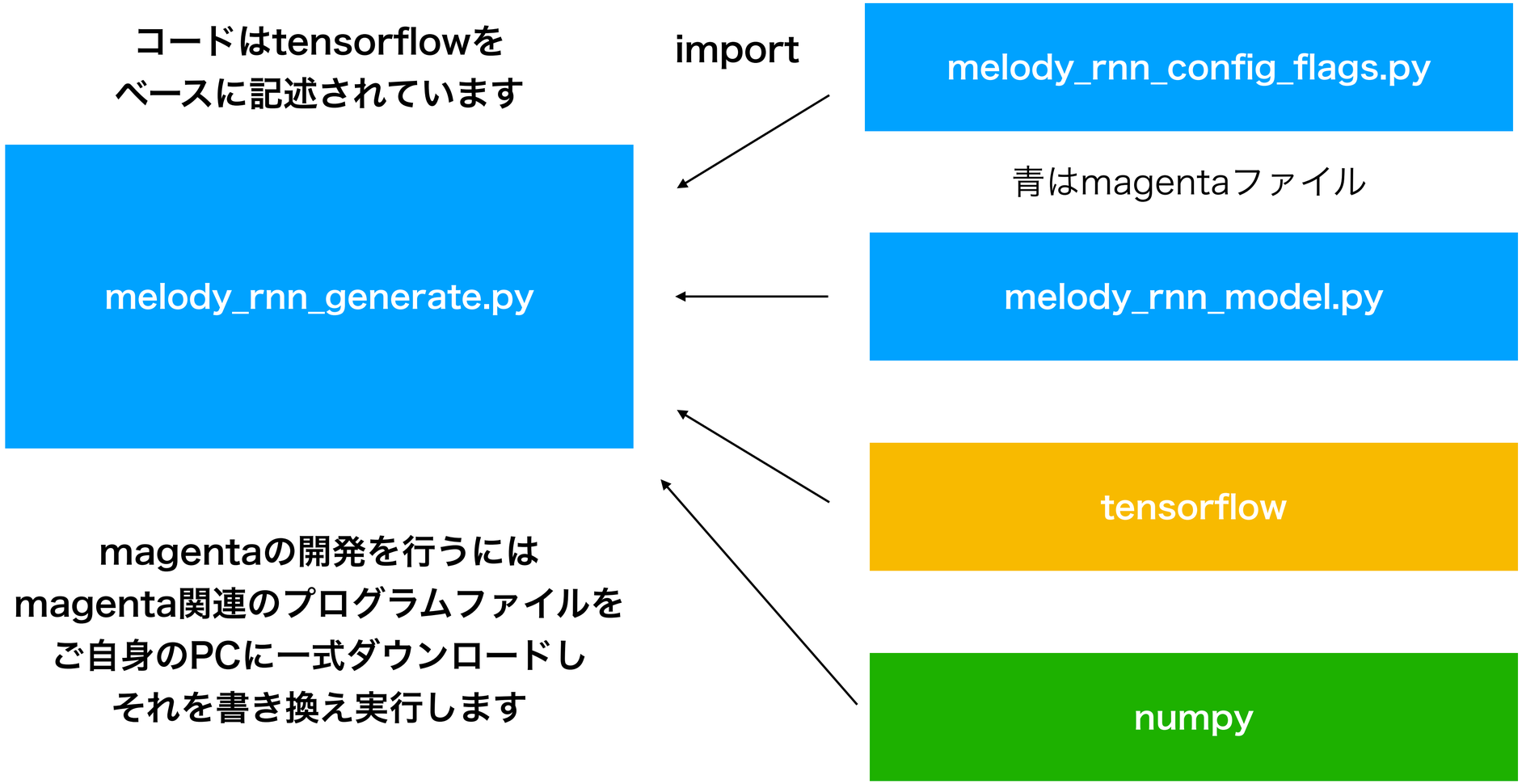
学習させたデータを使用して音楽生成を行なってみましょう。

８・Bundleファイル（.magファイル）作成 melody_rnn_generate.py

学習済みデータとしてBundleファイルを作成します。

それぞれのMagenta Pythonファイルも
その他のMagentaやTensorflowなどをライブラリー・モジュール
として読み込んで使用しています

コードはtensorflowを
ベースに記述されています



magentaの開発を行うには
magenta関連のプログラムファイルを
ご自身のPCに一式ダウンロードし
それを書き換え実行します

melody_rnn_generate.py
を例に個別にコードを詳細解説

melody_rnn_generate.pyを例に個別にコードを詳細解説

generate.pyはmagentaで音楽生成を行う処理が記述されているプログラムです。
melody rnnだけではなく、music vaeやperformance rnnでも個別に用意されています。

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import ast
import os
import time

import magenta
from magenta.models.melody_rnn import melody_rnn_config_flags
from magenta.models.melody_rnn import melody_rnn_model
from magenta.models.melody_rnn import melody_rnn_sequence_generator
from magenta.models.shared import sequence_generator
from magenta.models.shared import sequence_generator_bundle
from magenta.music.protobuf import generator_pb2
from magenta.music.protobuf import music_pb2
import tensorflow.compat.v1 as tf
```

使用するライブラリー・モジュールのインポート

melody_rnn_generate.pyを例に個別にコードを詳細解説

```
FLAGS = tf.app.flags.FLAGS
tf.app.flags.DEFINE_string(
    'run_dir', None,
    'Path to the directory where the latest checkpoint will be loaded from.')
tf.app.flags.DEFINE_string(
    'checkpoint_file', None,
    'Path to the checkpoint file. run_dir will take priority over this flag.')
tf.app.flags.DEFINE_string(
    'bundle_file', None,
    'Path to the bundle file. If specified, this will take priority over '
    'run_dir and checkpoint_file, unless save_generator_bundle is True, in '
    'which case both this flag and either run_dir or checkpoint_file are '
    'required')
tf.app.flags.DEFINE_boolean(
    'save_generator_bundle', False,
    'If true, instead of generating a sequence, will save this generator as a '
    'bundle file in the location specified by the bundle_file flag')
tf.app.flags.DEFINE_string(
    'bundle_description', None,
    'A short, human-readable text description of the bundle (e.g., training '
    'data, hyper parameters, etc.).')
```

tensorflowのapp.flagsを使用してコマンドをパラメーター初期値と組み合わせ定義する。
実行時に、コマンドラインからそのコマンドに該当したパラメーターを与えることができる。

tf.app.flagsがどんなコードなのかなど調べる場合はTensorflowのサイト
でリファレンスを参照、もしくはgithubレポジトリでコードを調べる。
全てのライブラリー、モジュールが同様です。

melody_rnn_generate.pyを例に個別にコードを詳細解説

```
tf.app.flags.DEFINE_integer(
    'num_outputs', 10,
    'The number of melodies to generate. One MIDI file will be created for '
    'each.')
tf.app.flags.DEFINE_integer(
    'num_steps', 128,
    'The total number of steps the generated melodies should be, priming '
    'melody length + generated steps. Each step is a 16th of a bar.')
tf.app.flags.DEFINE_string(
    'primer_melody', '',
    'A string representation of a Python list of '
    'magenta.music.Melody event values. For example: '
    '"[60, -2, 60, -2, 67, -2, 67, -2]". If specified, this melody will be '
    'used as the priming melody. If a priming melody is not specified, '
    'melodies will be generated from scratch.')
tf.app.flags.DEFINE_string(
    'primer_midi', '',
    'The path to a MIDI file containing a melody that will be used as a '
    'priming melody. If a primer melody is not specified, melodies will be '
    'generated from scratch.')
```

コマンド名num_outputs
初期値は整数型の１０

コマンド名num_steps
初期値は整数型の１28

コマンド名primer_melody
初期値は文字列型の" "（空）など

melody_rnn_generate.pyを例に個別にコードを詳細解説

get_checkpoint関数の作成

```
def get_checkpoint():
  """Get the training dir or checkpoint path to be used by the model."""
  if ((FLAGS.run_dir or FLAGS.checkpoint_file) and
      FLAGS.bundle_file and not FLAGS.save_generator_bundle):
    raise magenta.music.SequenceGeneratorError(
        'Cannot specify both bundle_file and run_dir or checkpoint_file')
  if FLAGS.run_dir:
    train_dir = os.path.join(os.path.expanduser(FLAGS.run_dir), 'train')
    return train_dir
  elif FLAGS.checkpoint_file:
    return os.path.expanduser(FLAGS.checkpoint_file)
  else:
    return None
```

checkpointファイルと
bundleファイルの両方が指定されると
エラーメッセージを返す

入力された
checkpointファイルのあるディレクトリーか
checkpointファイルへのパスを返す

melody_rnn_generate.pyを例に個別にコードを詳細解説

get_bundle関数の作成

```
def get_bundle():
  """Returns a generator_pb2.GeneratorBundle object based read from bundle_file.

  Returns:
    Either a generator_pb2.GeneratorBundle or None if the bundle_file flag is
    not set or the save_generator_bundle flag is set.
  """
  if FLAGS.save_generator_bundle:
    return None
  if FLAGS.bundle_file is None:
    return None
  bundle_file = os.path.expanduser(FLAGS.bundle_file)
  return magenta.music.read_bundle_file(bundle_file)
```

save_generator_bundleが指定されているか
bundleファイルが指定されていない時
Noneを返す

入力された
bundleファイルへのパスを返す

melody_rnn_generate.pyを例に個別にコードを詳細解説

run_with_flags関数の作成

```
def run_with_flags(generator):
  """Generates melodies and saves them as MIDI files.

  Uses the options specified by the flags defined in this module.

  Args:
    generator: The MelodyRnnSequenceGenerator to use for generation.
  """
```

メロディーを生成しMIDIファイルを作成する

生成にはMelodyRnnSequenceGeneratorクラス

（melody_rnn_sequence_generator.pyで定義）を使用

melody_rnn_generate.pyを例に個別にコードを詳細解説

```
if not FLAGS.output_dir:
  tf.logging.fatal('--output_dir required')
  return
FLAGS.output_dir = os.path.expanduser(FLAGS.output_dir)

primer_midi = None
if FLAGS.primer_midi:
  primer_midi = os.path.expanduser(FLAGS.primer_midi)

if not tf.gfile.Exists(FLAGS.output_dir):
  tf.gfile.MakeDirs(FLAGS.output_dir)
```

output_dirが指定されていない場合
tensorflowのlogging.fatal関数で
エラーメッセージを表示

指定された出力ディレクトリーの取得

指定されたprimer midiへのパスを取得

もしも出力ディレクトリーがない場合
tensorflowのgfile.MakeDirs関数で作成

melody_rnn_generate.pyを例に個別にコードを詳細解説

テンポの指定。指定がなければ初期値（１２０）を使用する

```
primer_sequence = None
qpm = FLAGS.qpm if FLAGS.qpm else magenta.music.DEFAULT_QUARTERS_PER_MINUTE
if FLAGS.primer_melody:
  primer_melody = magenta.music.Melody(ast.literal_eval(FLAGS.primer_melody))
  primer_sequence = primer_melody.to_sequence(qpm=qpm)
elif primer_midi:
  primer_sequence = magenta.music.midi_file_to_sequence_proto(primer_midi)
  if primer_sequence.tempos and primer_sequence.tempos[0].qpm:
    qpm = primer_sequence.tempos[0].qpm
else:
  tf.logging.warning(
      'No priming sequence specified. Defaulting to a single middle C.')
  primer_melody = magenta.music.Melody([60])
  primer_sequence = primer_melody.to_sequence(qpm=qpm)
```

ast.literal_evalでprimer_melodyをリストにして取得

primer_midiにテンポ情報がない場合はqpmを使用

primer_melodyもprimer_midiもない場合は
60(C3)と規定のテンポを使用

melody_rnn_generate.pyを例に個別にコードを詳細解説

```
# Derive the total number of seconds to generate based on the QPM of the
# priming sequence and the num_steps flag.
seconds_per_step = 60.0 / qpm / generator.steps_per_quarter
total_seconds = FLAGS.num_steps * seconds_per_step
```

全体の曲の長さ
num_stepsで指定した指定したステップ数
かけるステップごとの秒数

ステップごとの秒数
60/qpmで指定したテンポ/４分音符のステップ数

melody_rnn_generate.pyを例に個別にコードを詳細解説

```
# Specify start/stop time for generation based on starting generation at the
# end of the priming sequence and continuing until the sequence is num_steps
# long.
generator_options = generator_pb2.GeneratorOptions()
if primer_sequence:
  input_sequence = primer_sequence
  # Set the start time to begin on the next step after the last note ends.
  if primer_sequence.notes:
    last_end_time = max(n.end_time for n in primer_sequence.notes)
  else:
    last_end_time = 0
  generate_section = generator_options.generate_sections.add(
      start_time=last_end_time + seconds_per_step,
      end_time=total_seconds)
```

primer_sequence（メロディーかMIDI)
が指定された場合の生成曲の長さの調整

melody_rnn_generate.pyを例に個別にコードを詳細解説

```
if generate_section.start_time >= generate_section.end_time:
  tf.logging.fatal(
      'Priming sequence is longer than the total number of steps '
      'requested: Priming sequence length: %s, Generation length '
      'requested: %s',
      generate_section.start_time, total_seconds)
  return
```

生成曲のスタートタイムが全体の長さよりも長く（後に）なってしまう場合、
tensorflowのlogging.fatal関数でエラーメッセージを表示

melody_rnn_generate.pyを例に個別にコードを詳細解説

input_sequenceがない場合は0から開始し指定した長さで終わる

```
else:
  input_sequence = music_pb2.NoteSequence()
  input_sequence.tempos.add().qpm = qpm
  generate_section = generator_options.generate_sections.add(
      start_time=0,
      end_time=total_seconds)
generator_options.args['temperature'].float_value = FLAGS.temperature
generator_options.args['beam_size'].int_value = FLAGS.beam_size
generator_options.args['branch_factor'].int_value = FLAGS.branch_factor
generator_options.args[
    'steps_per_iteration'].int_value = FLAGS.steps_per_iteration
tf.logging.debug('input_sequence: %s', input_sequence)
tf.logging.debug('generator_options: %s', generator_options)
```

tf.app.flagsで取得した
各パラメーターを
generator_optionの引数に指定

melody_rnn_generate.pyを例に個別にコードを詳細解説

```
# Make the generate request num_outputs times and save the output as midi
# files.
date_and_time = time.strftime('%Y-%m-%d_%H%M%S')
digits = len(str(FLAGS.num_outputs))
for i in range(FLAGS.num_outputs):
  generated_sequence = generator.generate(input_sequence, generator_options)

  midi_filename = '%s_%s.mid' % (date_and_time, str(i + 1).zfill(digits))
  midi_path = os.path.join(FLAGS.output_dir, midi_filename)
  magenta.music.sequence_proto_to_midi_file(generated_sequence, midi_path)

tf.logging.info('Wrote %d MIDI files to %s',
                FLAGS.num_outputs, FLAGS.output_dir)
```

生成の日時の取得

num_outputsで指定した回数
生成を繰り返す

midiのファイル名と
生成ディレクトリーの取得

melody_rnn_generate.pyを例に個別にコードを詳細解説

```
def main(unused_argv):
  """Saves bundle or runs generator based on flags."""
  tf.logging.set_verbosity(FLAGS.log)

  bundle = get_bundle()

  if bundle:
    config_id = bundle.generator_details.id
    config = melody_rnn_model.default_configs[config_id]
    config.hparams.parse(FLAGS.hparams)
  else:
    config = melody_rnn_config_flags.config_from_flags()

  generator = melody_rnn_sequence_generator.MelodyRnnSequenceGenerator(
      model=melody_rnn_model.MelodyRnnModel(config),
      details=config.details,
      steps_per_quarter=config.steps_per_quarter,
      checkpoint=get_checkpoint(),
      bundle=bundle)
```

main関数の作成

bundleファイルの取得

bundleファイルの
処理命令

生成の際の各パラメーターの取得

melody_rnn_generate.pyを例に個別にコードを詳細解説

```
  if FLAGS.save_generator_bundle:
    bundle_filename = os.path.expanduser(FLAGS.bundle_file)
    if FLAGS.bundle_description is None:
      tf.logging.warning('No bundle description provided.')
    tf.logging.info('Saving generator bundle to %s', bundle_filename)
    generator.create_bundle_file(bundle_filename, FLAGS.bundle_description)
  else:
    run_with_flags(generator)
```

save_generator_bundleに関する処理

```
def console_entry_point():
  tf.app.run(main)
```

console_entry_point関数
tf.app.runでmain関数を実行

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

console_entry_point関数の実行

Magenta
独自AI作曲モデル作成
開発の流れ

Magenta 独自AI作曲モデル作成 開発の流れ

１・Magenta開発用の仮想環境作成

Magenta開発用の仮想環境を作成します。

２・Magentaレポジトリのクローン

githubからMagentaレポジトリーのクローン（Magenatファイル一式のダウンロード）を行います。

３・Magenta開発用環境構築

クローンしたレポジトリを使用してMagenta開発用の環境構築を行います。

４・独自モデルプログラミング

Magentaのプログラムコードを記述し独自モデルの作成を行います。

５・独自モデルで学習

作成した独自モデルで音楽データを学習させます。

６・独自モデルのBundleファイル（.magファイル）作成

学習済みデータとしてBundleファイルを作成し、ご自身のモデルで音楽生成を行います。

１・Magenta開発用の仮想環境作成

1・Magenta開発用 仮想環境作成

コマンドプロンプト(Windows)、ターミナル(Mac)でcondaコマンドにて仮想環境を作成します。

Anacondaで仮想環境作成

conda コマンド

Magenta用の環境名

devmagenta（以外でも良いです）

Python
バージョン

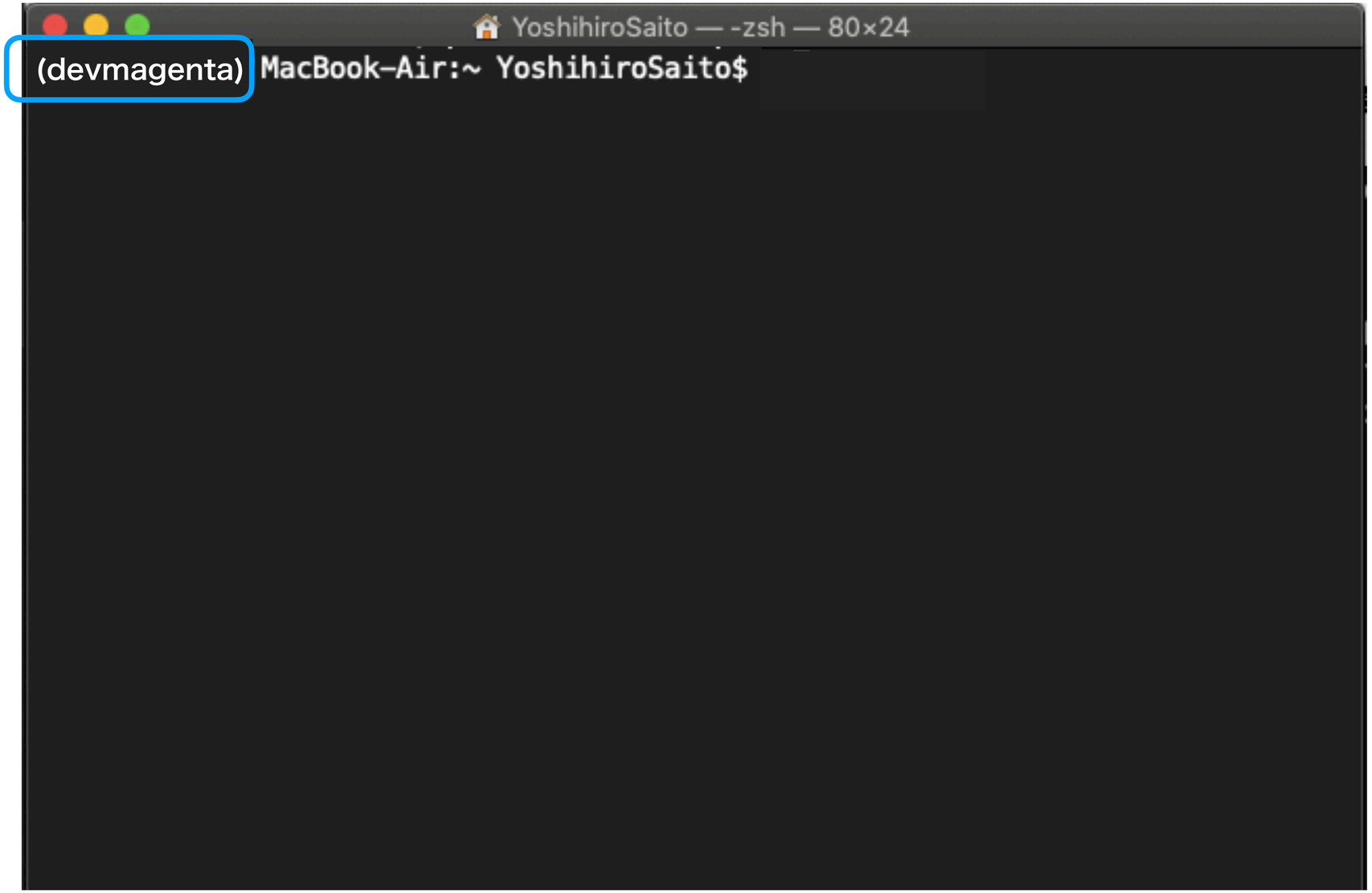
```
conda create -n devmagenta python=3.7
```

作成後有効化してください

```
conda activate devmagenta
```

1・Magenta用仮想環境作成

有効化になると
名前が表示されます



1・Magenta用仮想環境作成

TensorFlowのインストール

pipでのインストール
ターミナル、コマンドプロンプトで以下のコマンドを入力しインストール
Mganta環境では今回ライブラリーのインストールにpipを使用します。

```
pip install tensorflow==1.15
```

バージョン指定しないと2.0～がインストールされるので注意

注意：tensorflowのバージョンは1系しかMagentaは動きません。
必ず1系（1.15など）を指定してインストールしてください。例：pip install tensorflow==1.15
バージョン指定しないと2系（2.0~）がインストールされますがエラーが出て動作しませんのでご注意ください。
Anaconda Navigatorで2系しか表示されない場合、ターミナルからcondaコマンドでインストール

Magentaのインストール

pip install magentaでインストール

pipでのインストール
ターミナル、コマンドプロンプトで以下のコマンドを入力しインストール

```
pip install magenta
```

２・Magentaレポジトリのクローン

git

git

gitは、プログラムのソースコードなどの変更履歴を記録・追跡するための分散型バージョン管理システムである。
Linuxカーネルのソースコード管理に用いるためにリーナス・トーバルズによって開発され、それ以降ほかの多くのプロジェクトで採用されている。

gitでは、各ユーザのワーキングディレクトリに、全履歴を含んだリポジトリの完全な複製が作られる。
したがって、ネットワークにアクセスできないなどの理由で中心リポジトリにアクセスできない環境でも、履歴の調査や変更の記録といったほとんどの作業を行うことができる。

gitのコマンド

gitをインストールすると、バージョン管理のための色々な便利なコマンドを使用する事ができます

・git init リポジトリの作成

```
cd ＜リポジトリを作成するディレクトリ＞
git init
```

・git clone リポジトリのコピー（今回こちらを使用します）

```
cd ＜リポジトリを作成するディレクトリ＞
git clone ＜複製したいリポジトリのURL>
```

・git pull　他のリポジトリの変更点をローカル（自分の）リポジトリにマージ

```
git pull ＜変更点の取り込み元リポジトリURL＞
```

・git push　公開リポジトリにローカル（自分の）リポジトリの内容を送信

```
git push ＜送信先リポジトリ＞ ＜送信するブランチ＞:＜送信先ブランチ＞
```

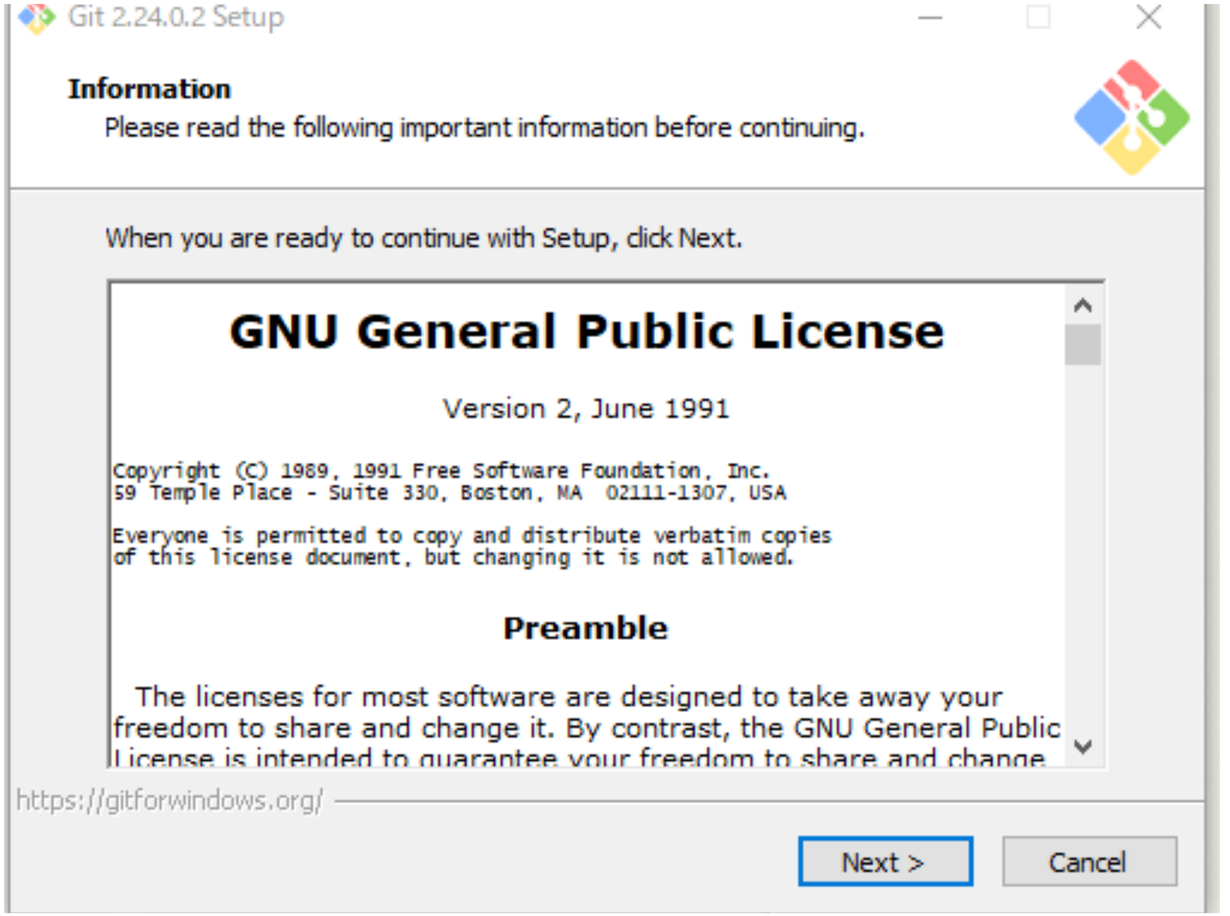
gitのインストール

Mac
https://sourceforge.net/projects/git-osx-installer/files/

Windows
https://gitforwindows.org/

通常最新版のインストールで問題ありません。

gitのインストール



通常デフォルトのままインストールを進めていただいて大丈夫です

gitコマンドで
Magentaレポジトリ
をgithubからクローン

２・Magentaレポジトリのクローン

githubからMagentaレポジトリのクローン（Magenatファイル一式のダウンロード）を行います。

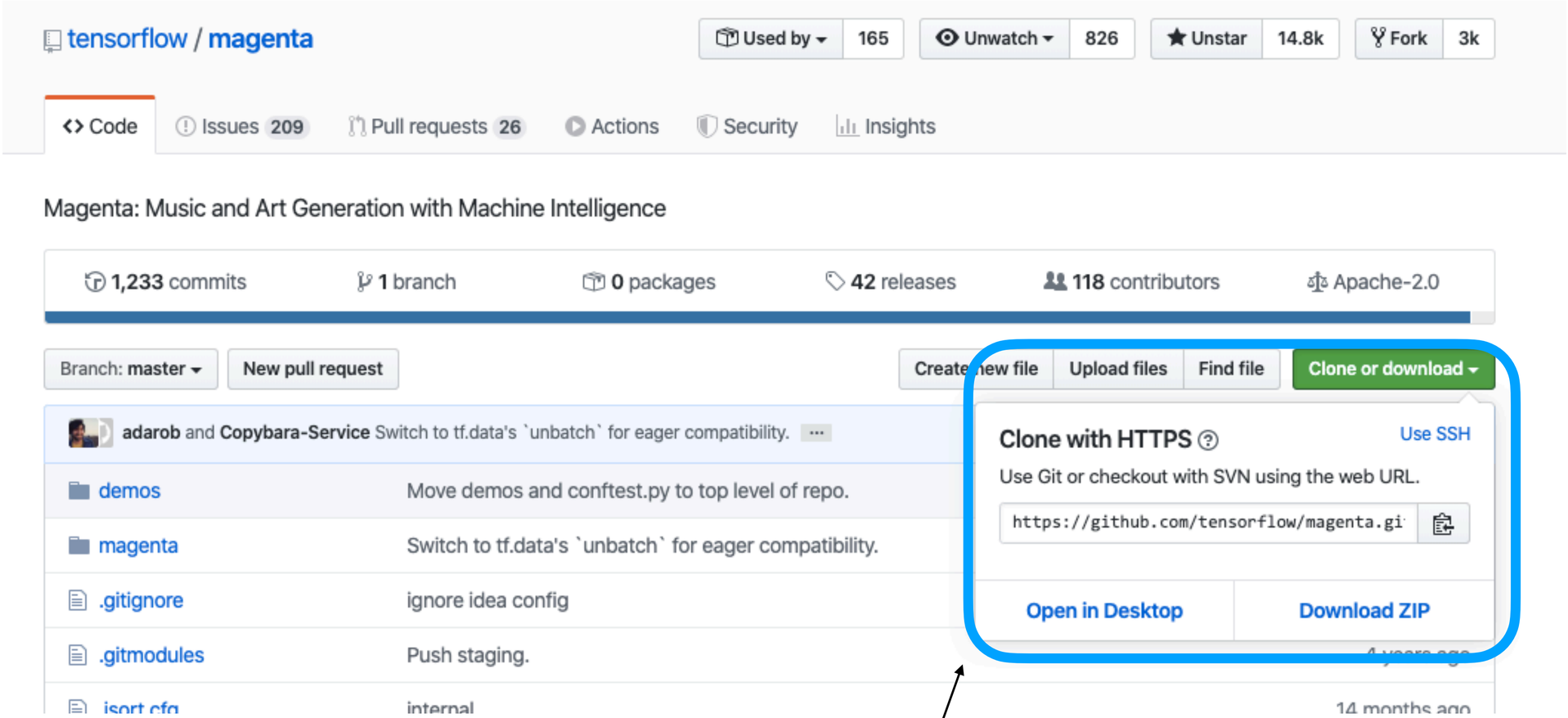
git clone https://github.com/tensorflow/magenta.git

ディレクトリパスの指定をしない場合は通常ユーザーディレクトリ直下にクローンされます。（講義ではユーザーディレクトリ直下として解説します）

githubのMagentaレポジトリからzipファイル形式でダウンロードする事も可能です。

https://github.com/tensorflow/magenta

2・Magentaレポジトリのクローン



ダウンロードおよびクローンのリンクはgithubのMagentaレポジトリ右上の方にあります

３・Magenta開発用環境構築

３・Magenta開発用環境構築

cdコマンドで作業ディレクトリをクローンしたMagentaフォルダに移動してください。
ユーザーディレクトリ直下にクローンした場合の例

cd magenta

です。（任意のディレクトリにクローンした場合はそこまでのパスを指定してください）

開発用のセットアップを行います。
magentaフォルダ内のsetup.pyを使用しますのでpythonファイルの実行です。（作業ディレクトリに注意）

python setup.py develop

仮想環境のmagentaをアンインストールします。

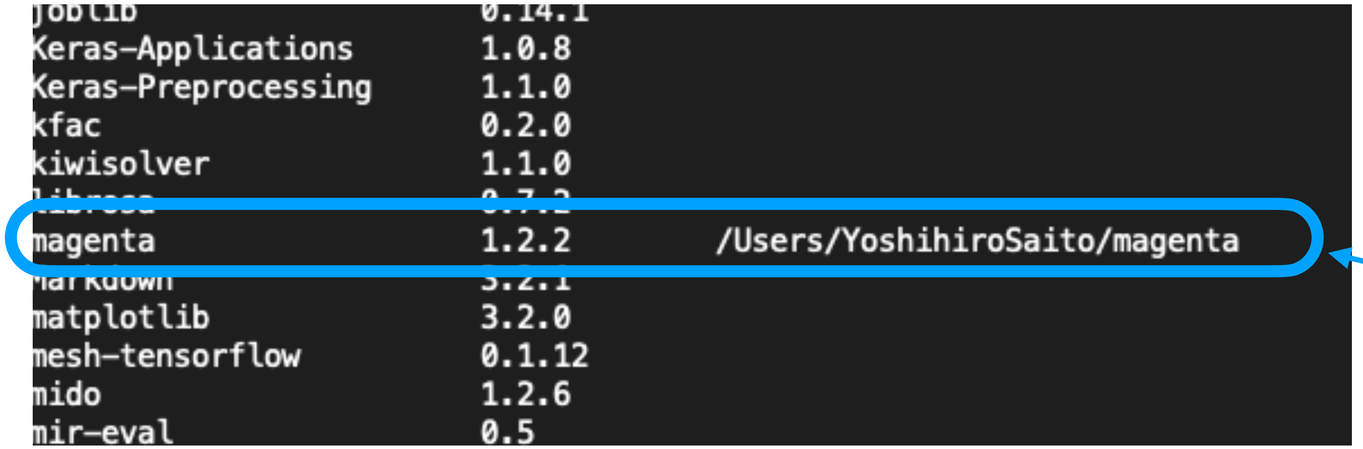
pip uninstall magenta

３・Magenta開発用環境構築

ライブラリの一覧をpip listで表示

pip list

devmagentaのPython環境ではなく、クローンしたMagentaのレポジトリが表示されます。



クローンしたMagentaが表示

以降実行はシェルスクリプトコマンドではなくPythonファイルとなります

Pythonファイルでメロディー生成実行コマンド Windows

```
python パス（絶対パス）melody_rnn_generate.py ^
--config=４種類の設定から１つ選択 ^
--bundle_file=ご自身の.magファイルへのパス ^
--output_dir=任意で生成先ディレクトリーを指定 ^
--num_outputs=10 ^
--num_steps=128 ^
--primer_melody="[60]”
```

通常は下記の様になります。
無事に音楽生成できれば成功です。
python /ユーザーディレクトリ/magenta/magenta/models/melody_rnn/melody_rnn_generate.py

Pythonファイルでメロディー生成実行コマンド　Mac

```
python パス（絶対パス）melody_rnn_generate.py \
--config=４種類の設定から１つ選択 \
--bundle_file=ご自身の.magファイルへのパス \
--output_dir=任意で生成先ディレクトリーを指定 \
--num_outputs=10 \
--num_steps=128 \
--primer_melody="[60]”
```

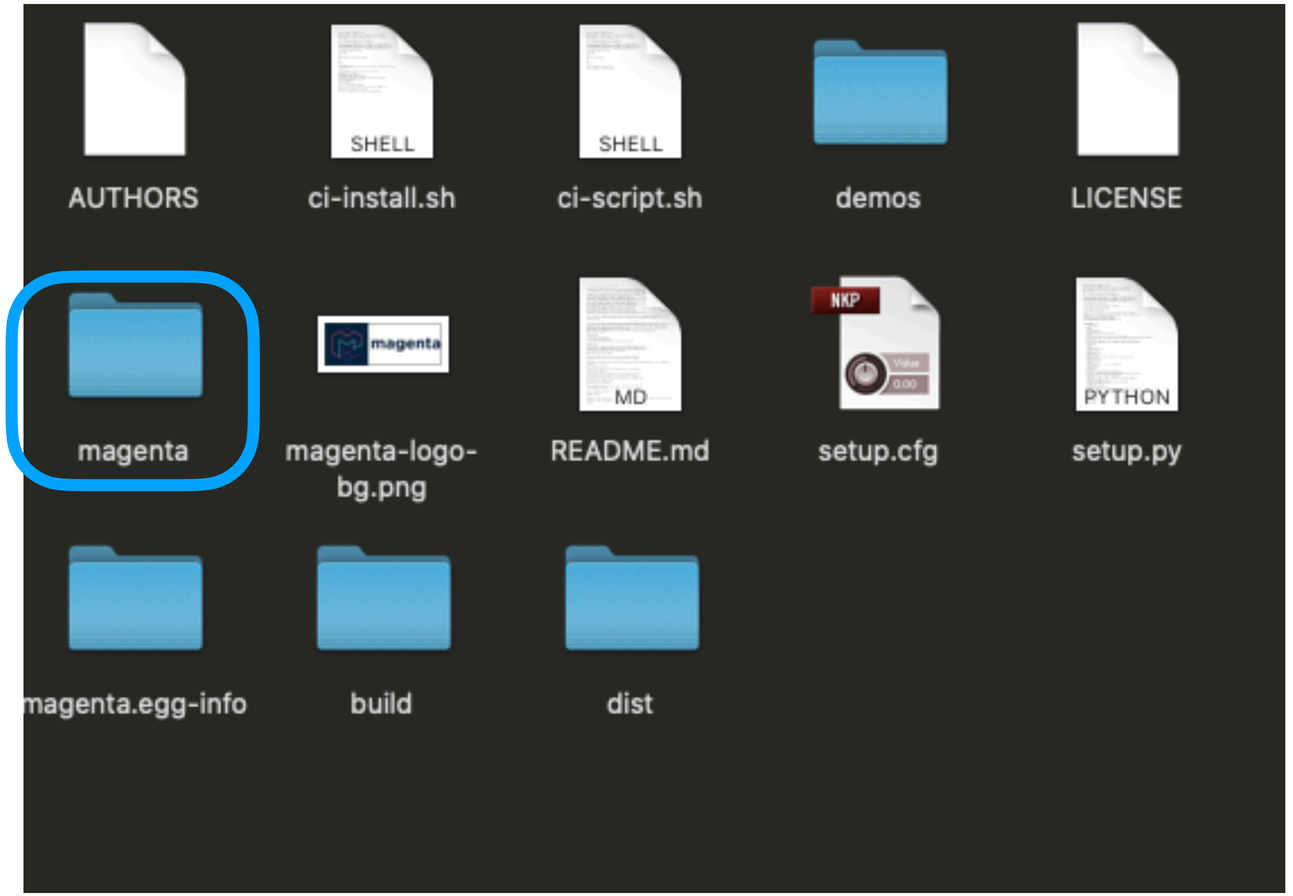
通常は下記の様になります。
無事に音楽生成できれば成功です。
python /ユーザーディレクトリ/magenta/magenta/models/melody_rnn/melody_rnn_generate.py

４・独自モデルプログラミング

４・独自モデルプログラミング

- demos
- magenta
- .gitignore
- .gitmodules
- .isort.cfg
- .pylintrc
- .travis.yml
- AUTHORS
- LICENSE
- README.md
- ci-install.sh
- ci-script.sh
- magenta-logo-bg.png
- setup.cfg
- setup.py

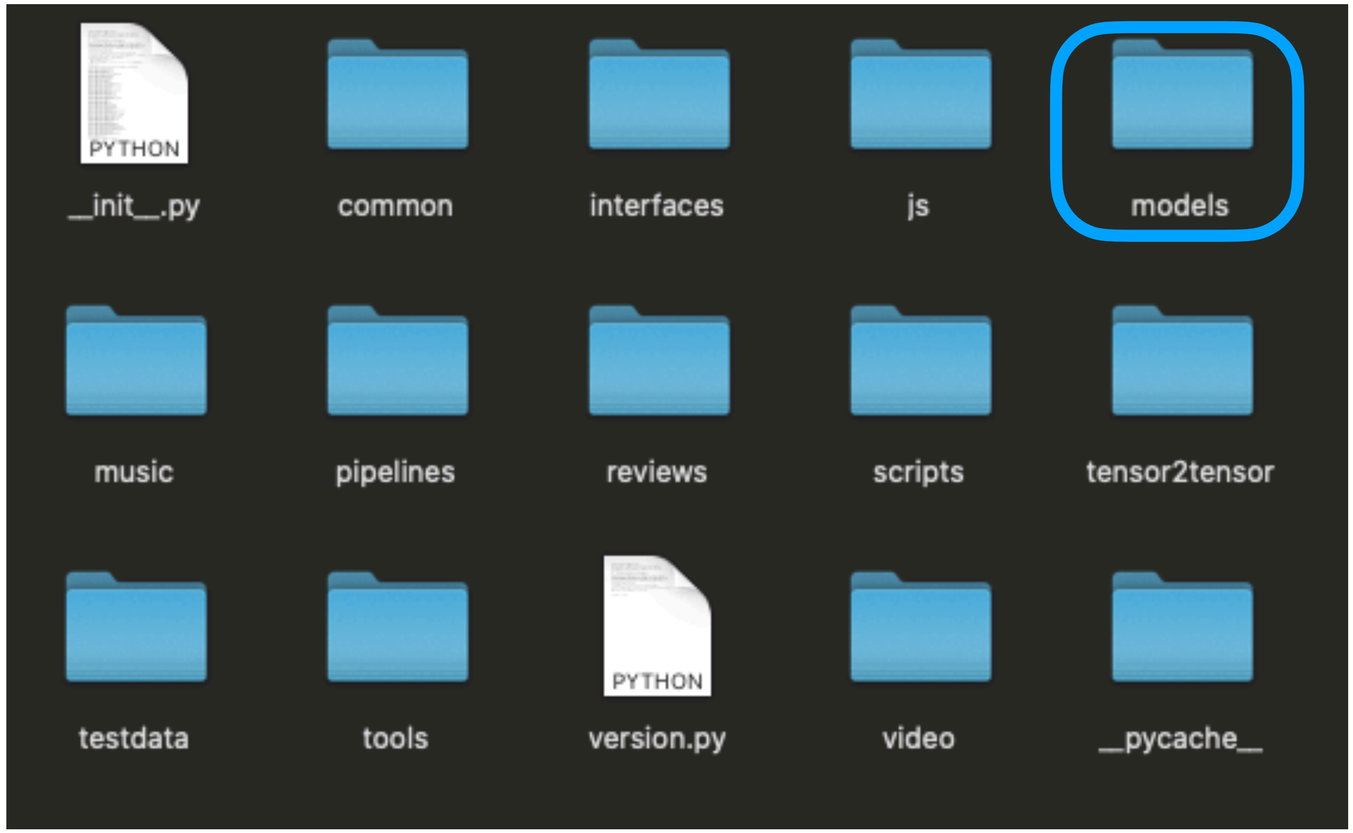
magentaレポジトリ（フォルダーに入っているMagenta Pythonプログラム一式）はこの様になっています
magenta内にmagentaがあり、ここに各モデルファイルなどがあります。



４・独自モデルプログラミング

- common
- interfaces
- js
- models
- music
- pipelines
- reviews
- scripts
- tensor2tensor
- testdata
- tools
- video
- __init__.py
- version.py

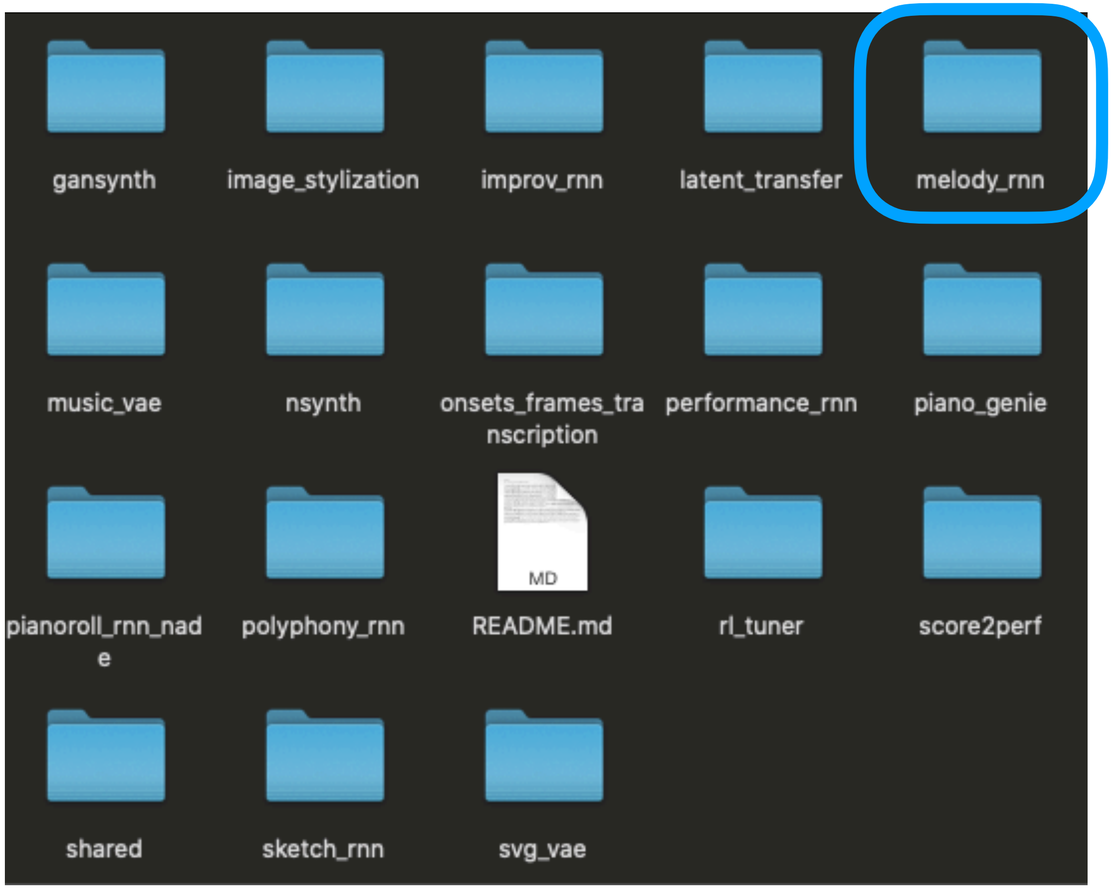
magenta/magenta
フォルダ内のmodelsにmelody rnnなど各モデルが入っています。



４・独自モデルプログラミング

- arbitrary_image_stylization
- coconet
- drums_rnn
- gansynth
- image_stylization
- improv_rnn
- latent_transfer
- melody_rnn
- music_vae
- nsynth
- onsets_frames_transcription
- performance_rnn
- piano_genie
- pianoroll_rnn_nade
- polyphony_rnn
- rl_tuner
- score2perf
- shared
- sketch_rnn

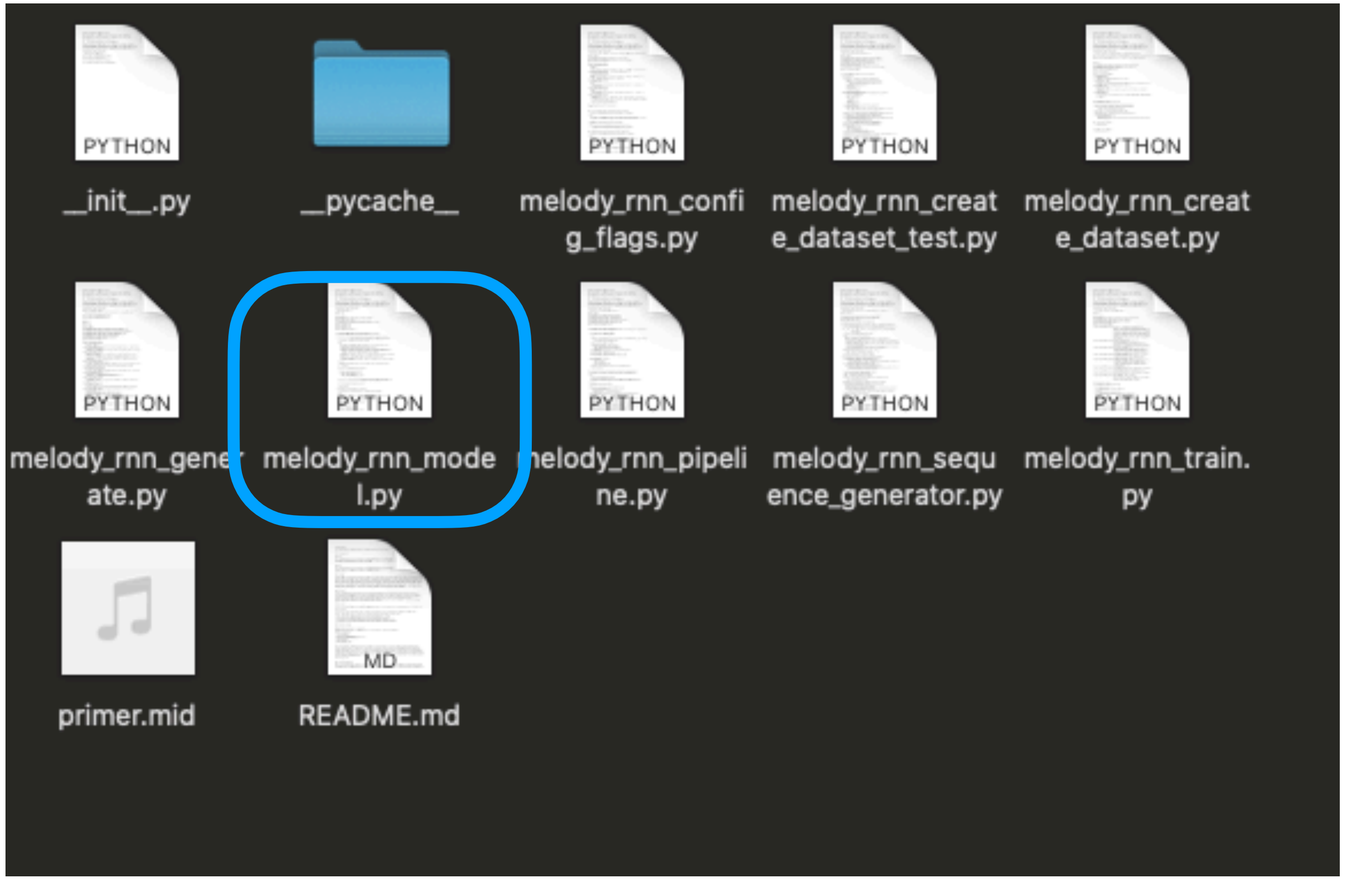
melody rnnに学習や音楽生成用のプログラムが一式入っています。
他のモデルも同様です。



４・独自モデルプログラミング

- README.md
- __init__.py
- melody_rnn_config_flags.py
- melody_rnn_create_dataset.py
- melody_rnn_create_dataset_test.py
- melody_rnn_generate.py
- melody_rnn_model.py
- melody_rnn_pipeline.py
- melody_rnn_sequence_generator.py
- melody_rnn_train.py
- primer.mid

モデルのプログラムはmelody_rnn_model.pyです。
このファイルに新たなモデルを書き加え作成します。



４・独自モデルプログラミング

melody_rnn_model.pyにコードを書き加え、ご自身の独自AI作曲モデルを作成します。
記述、作成する内容は以下です。

・モデルの名前（とid）　ご自身の名前をAI作曲モデル名にします

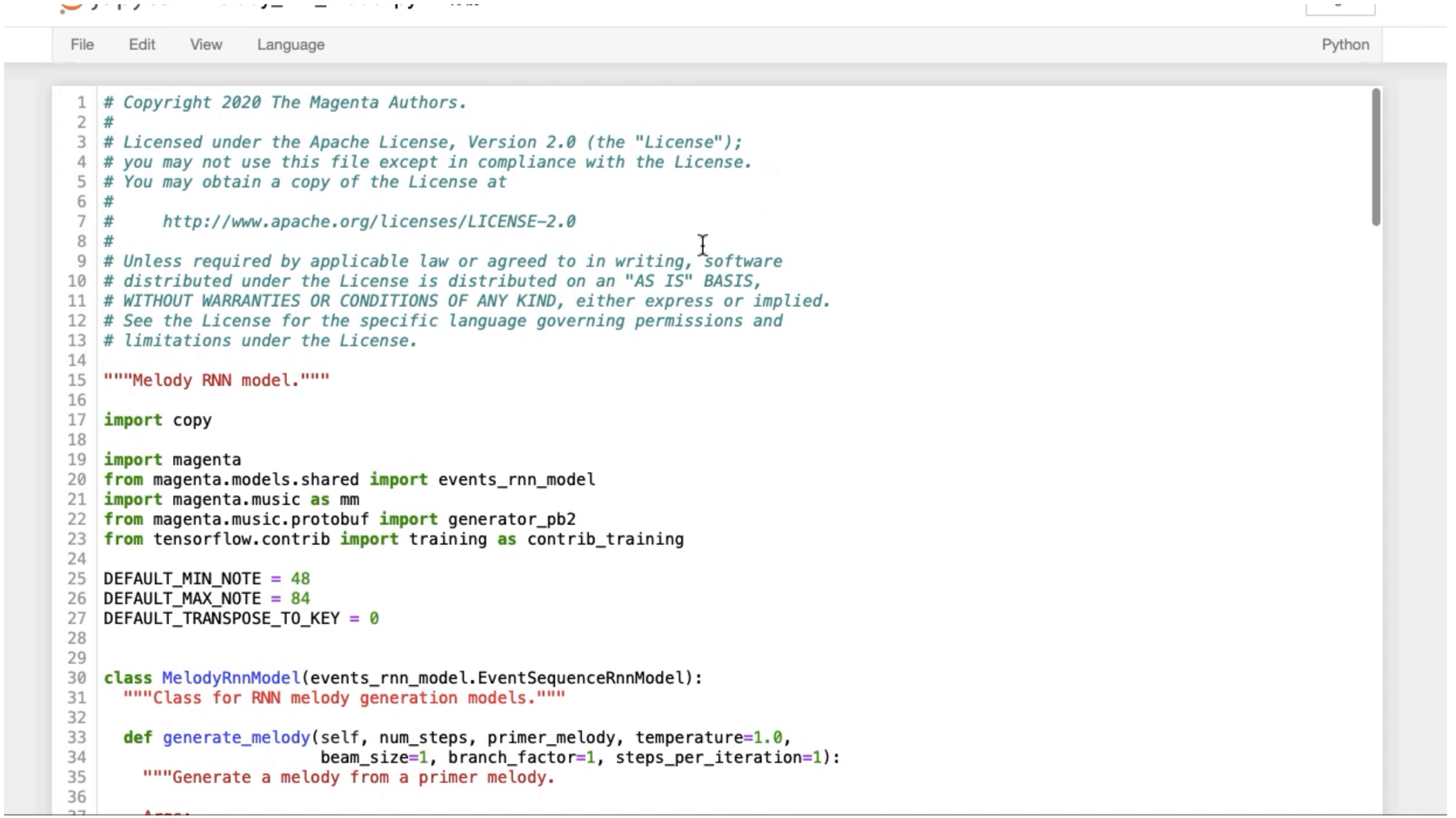
・音楽生成のアルゴリズムを３種類から選択

・最低音階、最高音階の範囲指定

・ハイパーパラメータの指定（ドロップアウト率と学習率）

４・独自モデルプログラミング

Jupyter Notebook（またはご自身使用のテキストエディター）でmelody_rnn_model.pyを開いてください。



４・独自モデルプログラミング

Jupyter Notebook（またはご自身使用のテキストエディター）でmelody_rnn_model.pyを開いてください。

basic rnn

```
'basic_rnn': MelodyRnnConfig(
    generator_pb2.GeneratorDetails(
        id='basic_rnn',
        description='Melody RNN with one-hot encoding.'),
    magenta.music.OneHotEventSequenceEncoderDecoder(
        magenta.music.MelodyOneHotEncoding(
            min_note=DEFAULT_MIN_NOTE,
            max_note=DEFAULT_MAX_NOTE)),
    contrib_training.HParams(
        batch_size=128,
        rnn_layer_sizes=[128, 128],
        dropout_keep_prob=0.5,
        clip_norm=5,
        learning_rate=0.001)),
```

Config（設定）の名前とID

説明

学習と生成に使用するアルゴリズム

音域

ハイパーパラメーター

４・独自モデルプログラミング

Jupyter Notebook（またはご自身使用のテキストエディター）でmelody_rnn_model.pyを開いてください。

mono rnn

```
'mono_rnn': MelodyRnnConfig(
    generator_pb2.GeneratorDetails(
        id='mono_rnn',
        description='Monophonic RNN with one-hot encoding.'),
    magenta.music.OneHotEventSequenceEncoderDecoder(
        magenta.music.MelodyOneHotEncoding(
            min_note=0,
            max_note=128)),
    contrib_training.HParams(
        batch_size=128,
        rnn_layer_sizes=[128, 128],
        dropout_keep_prob=0.5,
        clip_norm=5,
        learning_rate=0.001),
    min_note=0,
    max_note=128,
    transpose_to_key=None),
```

Config（設定）の名前とID

説明

学習と生成に使用するアルゴリズム

音域

ハイパーパラメーター

４・独自モデルプログラミング

Jupyter Notebook（またはご自身使用のテキストエディター）でmelody_rnn_model.pyを開いてください。

lookback rnn

```
'lookback_rnn': MelodyRnnConfig(
    generator_pb2.GeneratorDetails(
        id='lookback_rnn',
        description='Melody RNN with lookback encoding.'),
    magenta.music.LookbackEventSequenceEncoderDecoder(
        magenta.music.MelodyOneHotEncoding(
            min_note=DEFAULT_MIN_NOTE,
            max_note=DEFAULT_MAX_NOTE)),
    contrib_training.HParams(
        batch_size=128,
        rnn_layer_sizes=[128, 128],
        dropout_keep_prob=0.5,
        clip_norm=5,
        learning_rate=0.001)),
```

Config（設定）の名前とID

説明

学習と生成に使用するアルゴリズム

音域

ハイパーパラメーター

４・独自モデルプログラミング

Jupyter Notebook（またはご自身使用のテキストエディター）でmelody_rnn_model.pyを開いてください。

attention rnn

```
'attention_rnn': MelodyRnnConfig(
    generator_pb2.GeneratorDetails(
        id='attention_rnn',
        description='Melody RNN with lookback encoding and attention.'),
    magenta.music.KeyMelodyEncoderDecoder(
        min_note=DEFAULT_MIN_NOTE,
        max_note=DEFAULT_MAX_NOTE),
    contrib_training.HParams(
        batch_size=128,
        rnn_layer_sizes=[128, 128],
        dropout_keep_prob=0.5,
        attn_length=40,
        clip_norm=3,
        learning_rate=0.001))
```

Config（設定）の名前とID

説明

学習と生成に使用するアルゴリズム

音域

ハイパーパラメーター

４・独自モデルプログラミング
ではコード記述作業をします

※難易度が高い方はコピー＆ペーストで対応できる様になっています

４・独自モデルプログラミング

```
'mono_rnn': MelodyRnnConfig(
    generator_pb2.GeneratorDetails(
        id='mono_rnn',
        description='Monophonic RNN with one-hot encoding.'),
    magenta.music.OneHotEventSequenceEncoderDecoder(
        magenta.music.MelodyOneHotEncoding(
            min_note=0,
            max_note=128)),
    contrib_training.HParams(
        batch_size=128,
        rnn_layer_sizes=[128, 128],
        dropout_keep_prob=0.5,
        clip_norm=5,
        learning_rate=0.001),
    min_note=0,
    max_note=128,
    transpose_to_key=None),

'ysaito_rnn': MelodyRnnConfig(
    generator_pb2.GeneratorDetails(
        id='ysaito_rnn',
        description='ysaito original ai music model'),
    magenta.music.LookbackEventSequenceEncoderDecoder(
        magenta.music.MelodyOneHotEncoding(
            min_note=0,
            max_note=128)),
    contrib_training.HParams(
        batch_size=128,
        rnn_layer_sizes=[128, 128],
        dropout_keep_prob=0.5,
        clip_norm=5,
        learning_rate=0.001),
    min_note=0,
    max_note=128,
    transpose_to_key=None),

'lookback_rnn': MelodyRnnConfig(
```

basic rnn（またはmono_rnn）の下に左図の様にモデル設定を記述し作成します。
（インデントに注意）

basic rnnまたはmono rnnをコピペしていただいて該当部分をご自分で書き換えでも構いません。

・モデルの名前（とid）　ご自身の名前をAI作曲モデル名にします

basic rnn（またはmono_rnn）の下に以下の様にモデル設定を記述し作成します。（インデントに注意）

```
'ysaito_rnn': MelodyRnnConfig(
    generator_pb2.GeneratorDetails(
        id='ysaito_rnn',
        description='ysaito original ai music model'),
    magenta.music.LookbackEventSequenceEncoderDecoder(
        magenta.music.MelodyOneHotEncoding(
            min_note=0,
            max_note=128)),
    contrib_training.HParams(
        batch_size=128,
        rnn_layer_sizes=[128, 128],
        dropout_keep_prob=0.5,
        clip_norm=5,
        learning_rate=0.001),
    min_note=0,
    max_note=128,
    transpose_to_key=None),
```

Config（設定）の名前とID

モデルの設定名とidを記述します。

基本的にどちらも同じ名称にしてください。

今回はご自身の名前をモデル名に。

名前＋_rnn

など

４・独自モデルプログラミング

・モデルの名前（とid）　ご自身の名前をAI作曲モデル名にします

basic rnn（またはmono_rnn）の下に以下の様にモデル設定を記述し作成します。（インデントに注意）

```
'ysaito_rnn': MelodyRnnConfig(
    generator_pb2.GeneratorDetails(
        id='ysaito_rnn',
        description='ysaito original ai music model'),
    magenta.music.LookbackEventSequenceEncoderDecoder(
        magenta.music.MelodyOneHotEncoding(
            min_note=0,
            max_note=128)),
    contrib_training.HParams(
        batch_size=128,
        rnn_layer_sizes=[128, 128],
        dropout_keep_prob=0.5,
        clip_norm=5,
        learning_rate=0.001),
    min_note=0,
    max_note=128,
    transpose_to_key=None),
```

description（説明）は学習および生成には影響はありませんが、アルゴリムの指定などで確認したい事などがある場合はここに書いてください。

４・独自モデルプログラミング

・モデルの名前（とid）　ご自身の名前をAI作曲モデル名にします

basic rnn（またはmono_rnn）の下に以下の様にモデル設定を記述し作成します。（インデントに注意）

```
'ysaito_rnn': MelodyRnnConfig(
    generator_pb2.GeneratorDetails(
        id='ysaito_rnn',
        description='ysaito original ai music model'),
    magenta.music.LookbackEventSequenceEncoderDecoder(
        magenta.music.MelodyOneHotEncoding(
            min_note=0,
            max_note=128)),
    contrib_training.HParams(
        batch_size=128,
        rnn_layer_sizes=[128, 128],
        dropout_keep_prob=0.5,
        clip_norm=5,
        learning_rate=0.001),
    min_note=0,
    max_note=128,
    transpose_to_key=None),
```

学習と生成に使用するアルゴリズム

３種類のアルゴリズムから指定できます

・OneHotEventSequenceEncoderDecoder

・LookbackEventSequenceEncoderDecoder

・KeyMelodyEncoderDecoder (attention)

KeyMelodyEncoderDecoder以外は左図参考コードの様に

MelodyOneHotEncodingも記述してください。

３種類のアルゴリムは第３回学習済みデータの回に解説しています

Basic: OneHotEventSequenceEncoderDecoder
LSTMを使用した基本的な単音音楽生成モデル

ワンホットエンコーディングで学習データからメロディーを抽出し生成

Lookback: LookbackEventSequenceEncoderDecoder
音楽から、その楽曲のパターンをより簡単に抽出できる様にしたモデル

Attention: KeyMelodyEncoderDecoder
より長い時間（過去の）データにアクセスする事ができるモデルで、時間

的なメロディーの整合性をより高める事ができる

Basic: OneHotEventSequenceEncoderDecoder
用途例：通常のメロディー生成に使用

特徴：LSTMを使用した基本的な単音の音楽生成を、ワンホットエンコーディングで学習データからメロディーを抽出する事で実行します。
メロディーの音域はC２～C5（MIDIの４８～８４）に制限され、範囲外の音はC2~C5の範囲内に修正された上でメロディー生成される仕組みです。
C３～C4（MIDIの６０～７２）はピアノでいうとちょうど真ん中あたりで、最もメロディーでは活用される音域です。
それに上下１オクターブの音域を加えた範囲に限定する事で、極端に外れる音のないメロディーらしい音楽の生成を可能にしているわけです。

Lookback: LookbackEventSequenceEncoderDecoder
用途例：パターンに沿ったメロディーを生成

特徴：音楽の構造は、何かしらの共通パターンによって成り立っている事が非常に多いです。
もちろんメロディーも例外ではありません。
フレーズとフレーズが、共通のリズム・音の特徴を持った塊ごとに、周期的に分けられている事が多いはずです。（分節と呼びます）
Lookbackは１小節から２小節前のメロディーのパターンを容易に認識できる様にRNNのセル（機械学習で演算を行う部分）が工夫されています。
これによって、より音楽的なパターン構造を持ったメロディーを生成する事を目指した学習済みデータです。

Attention: KeyMelodyEncoderDecoder
用途例：時間的整合性を持ったメロディーの生成

特徴：AIによって生成されたメロディーは時間経過していくと、最初のメロディーと後のメロディーの整合性が取りづらくなり、違和感のあるメロディーになってしまう事があります。
AttentionはLookbackに加えて、より長い時間のデータにアクセスし、メロディーの依存関係を発見できる様にセルが設計されています。
これにより整合性のあるメロディー生成が行われるとの事です。

上級編

Magentaの各アルゴリズムやRNNの構造はそれぞれの外部ファイルに記述してあります。

例えばOneHotEventSequenceEncoderDecoderはmagentaのmusicディレクトリの中にあるencoder_decoder.pyで定義されています。

```
generator_pb2.GeneratorDetails(
    id='basic_rnn',
    description='Melody RNN with one-hot encoding.'),
magenta.music.OneHotEventSequenceEncoderDecoder(
    magenta.m
        min_r
        max_r
contrib_trair
    batch_siz
    rnn_layer_sizes=[128, 128],
    dropout_keep_prob=0.5,
    clip_norm=5,
    learning_rate=0.001)),
```

Definition References

Defined in magenta/music/encoder_decoder.py

```
359  class OneHotEventSequenceEncoderDecoder(EventSequenceEncoderDecoder):
```

上級編

encoder_decoder.py 音楽学習＆生成アルゴリズムの中心ともなるプラグラム

```
# Copyright 2020 The Magenta Authors.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

"""Classes for converting between event sequences and models inputs/outputs.

OneHotEncoding is an abstract class for specifying a one-hot encoding, i.e.
how to convert back and forth between an arbitrary event space and integer
indices between 0 and the number of classes.

EventSequenceEncoderDecoder is an abstract class for translating event
_sequences_, i.e. how to convert event sequences to input vectors and output
labels to be fed into a model, and how to convert from output labels back to
events.

Use EventSequenceEncoderDecoder.encode to convert an event sequence to a
tf.train.SequenceExample of inputs and labels. These SequenceExamples are fed
into the model during training and evaluation.

During generation, use EventSequenceEncoderDecoder.get_inputs_batch to convert a
list of event sequences into an inputs batch which can be fed into the model to
```

４・独自モデルプログラミング

・モデルの名前（とid）　ご自身の名前をAI作曲モデル名にします

basic rnn（またはmono_rnn）の下に以下の様にモデル設定を記述し作成します。（インデントに注意）

```
'basic_rnn': MelodyRnnConfig(
    generator_pb2.GeneratorDetails(
        id='basic_rnn',
        description='Melody RNN with one-hot encoding.'),
    magenta.music.OneHotEventSequenceEncoderDecoder(
        magenta.music.MelodyOneHotEncoding(
            min_note=DEFAULT_MIN_NOTE,
            max_note=DEFAULT_MAX_NOTE)),
    contrib_training.HParams(
        batch_size=128,
        rnn_layer_sizes=[128, 128],
        dropout_keep_prob=0.5,
        clip_norm=5,
        learning_rate=0.001)),
```

音域

basicの様に音域指定のdefault48~84
またはmonoの様に0~128の全域
のどちらかを指定します。

左記サンプルはdefault設定使用の場合
basic_rnnを参考

４・独自モデルプログラミング

・モデルの名前（とid）　ご自身の名前をAI作曲モデル名にします

basic rnn（またはmono_rnn）の下に以下の様にモデル設定を記述し作成します。（インデントに注意）

```
'ysaito_rnn': MelodyRnnConfig(
    generator_pb2.GeneratorDetails(
        id='ysaito_rnn',
        description='ysaito original ai music model'),
    magenta.music.LookbackEventSequenceEncoderDecoder(
        magenta.music.MelodyOneHotEncoding(
            min_note=0,
            max_note=128)),
    contrib_training.HParams(
        batch_size=128,
        rnn_layer_sizes=[128, 128],
        dropout_keep_prob=0.5,
        clip_norm=5,
        learning_rate=0.001),
    min_note=0,
    max_note=128,
    transpose_to_key=None),
```

音域

basicの様に音域指定のdefault48~84

またはmonoの様に0~128の全域

のどちらかを指定します。

左記サンプルの様に0~128指定の場合は

```
min_note=0,
max_note=128,
transpose_to_key=None,
```

も追記。

mono_rnnを参考

・モデルの名前（とid）　ご自身の名前をAI作曲モデル名にします

basic rnn（またはmono_rnn）の下に以下の様にモデル設定を記述し作成します。（インデントに注意）

```
'ysaito_rnn': MelodyRnnConfig(
    generator_pb2.GeneratorDetails(
        id='ysaito_rnn',
        description='ysaito original ai music model'),
    magenta.music.LookbackEventSequenceEncoderDecoder(
        magenta.music.MelodyOneHotEncoding(
            min_note=0,
            max_note=128)),
    contrib_training.HParams(
        batch_size=128,
        rnn_layer_sizes=[128, 128],
        dropout_keep_prob=0.5,
        clip_norm=5,
        learning_rate=0.001),
    min_note=0,
    max_note=128,
    transpose_to_key=None),
```

学習のためのパラメーターです。
dropout率や学習率を変えてみるとどんな変化がでるか試してみてください。

今回は特にハイパーパラメーターを変更する必要はありませんが、今後機械学習を実践していく場合にこの様に設定をしていという事は是非覚えておいてください。

ハイパーパラメーター

５・独自モデルで学習

前回解説した流れで以下の様に実践します

5-１・midiファイルの用意

mdata5を使用します。

5-２・NoteSequence（tfrecord）の作成

noteSequences.terecordファイルを作成し使用します。

5-３・トレーニングデータとテストデータ の作成

トレーニングデータとテストデータ を作成します。今回もテストデータ は作成しません。

5-４・トレーニング

音楽データを学習させます。

5-５・テスト

今回も行いません。

5-６・tensorboardで学習の確認

学習の結果を視覚的に確認します。

5-７・音楽生成

学習させたデータを使用して音楽生成を行なってみましょう。

5-８・Bundleファイル（.magファイル）作成

学習済みデータとしてBundleファイルを作成します。

ご自身のモデルを記述したmelody_rnn_model.pyと.magファイルが最終課題の提出ファイルとなります。

5-１・midiファイルの用意

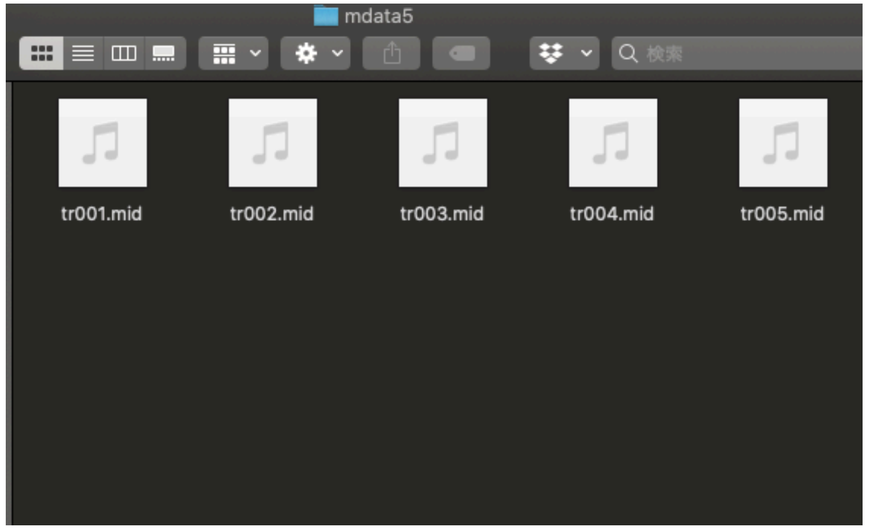
midiファイルを作成し任意のフォルダーへ。配布したmidiデータが入ったフォルダーを使用してください。

midiファイルを作成し任意のフォルダーへ。
今回は配布したmidiデータが入ったフォルダー
mdata5を使用してください。

呼び出しにパスを指定するのでわかりやすい場所へ保存する事をお勧めします。

今後ご自身で新たな音楽データ学習をする場合も同様に全ての学習させるmidiデータを同一のフォルダーに入れておきます

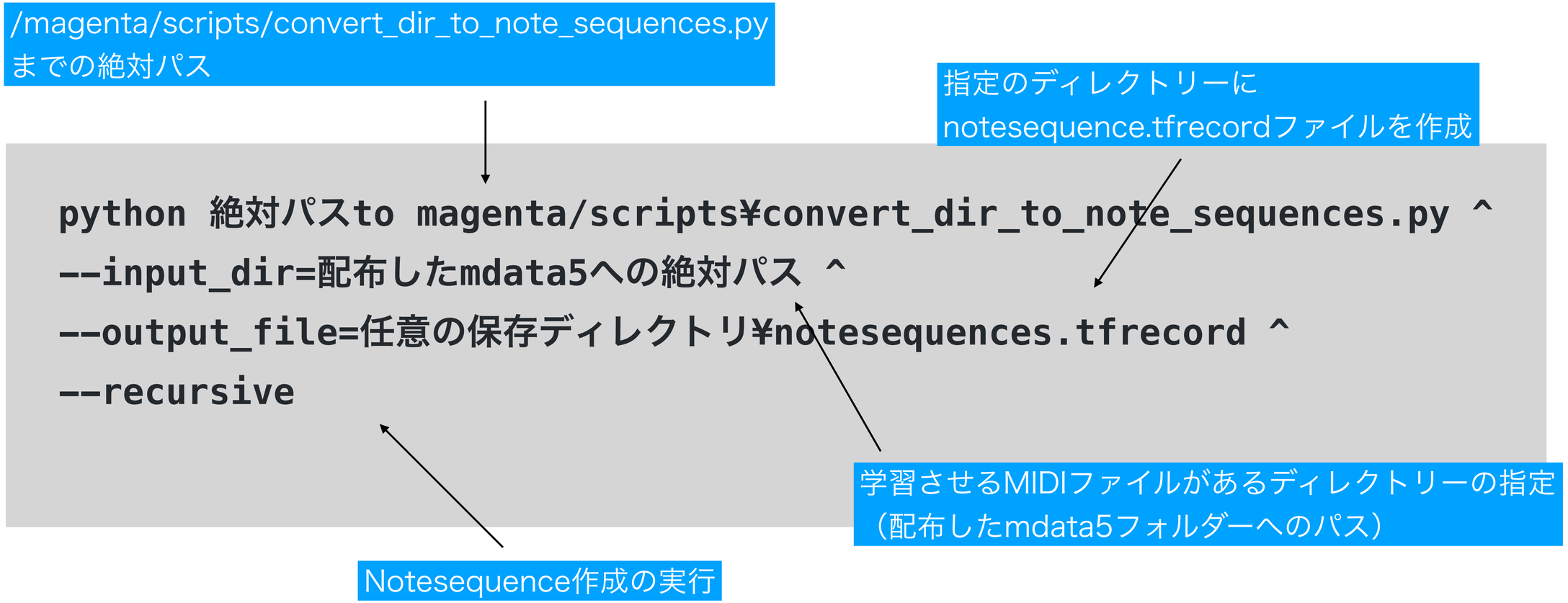
この操作はMelody RNNだけではなくMagenta各モデル共通です。



mdata5
８小節の単音メロディーデータ
が5曲入っています

5-２・NoteSequence（tfrecord）の作成 Win

midiをMagentaで扱える様にNoteSequenceというデータ（TensorFlowのレコードファイル）に変換します。



magenta開発環境では実行は全てpythonファイルになります

5-２・NoteSequence（tfrecord）の作成 Mac

midiをMagentaで扱える様にNoteSequenceというデータ（TensorFlowのレコードファイル）に変換します。

/magenta/scripts/convert_dir_to_note_sequences.py
までの絶対パス

指定のディレクトリーに
notesequence.tfrecordファイルを作成

```
python 絶対パスto magenta/scripts/convert_dir_to_note_sequences.py \
--input_dir=配布したmdata5への絶対パス \
--output_file=任意の保存ディレクトリ/notesequences.tfrecord \
--recursive
```

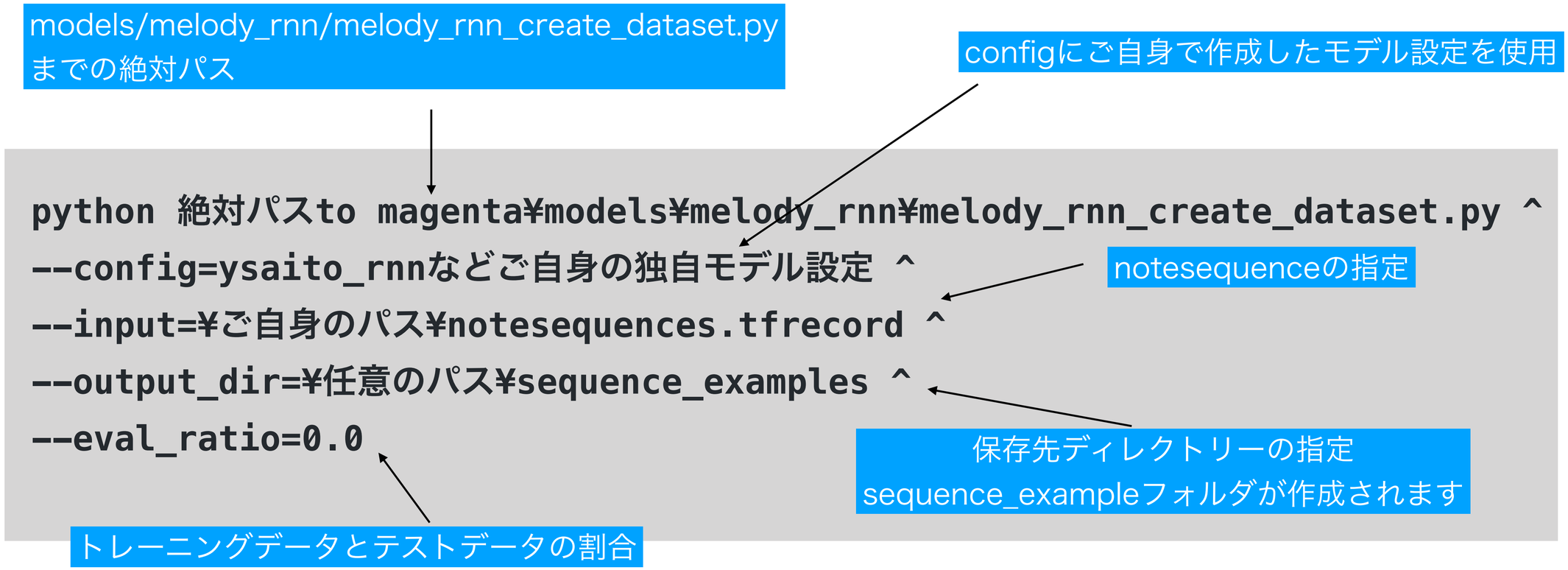
学習させるMIDIファイルがあるディレクトリーの指定
（配布したmdata5フォルダーへのパス）

Notesequence作成の実行

magenta開発環境では実行は全てpythonファイルになります

5-３・トレーニングデータとテストデータ の作成 Windows

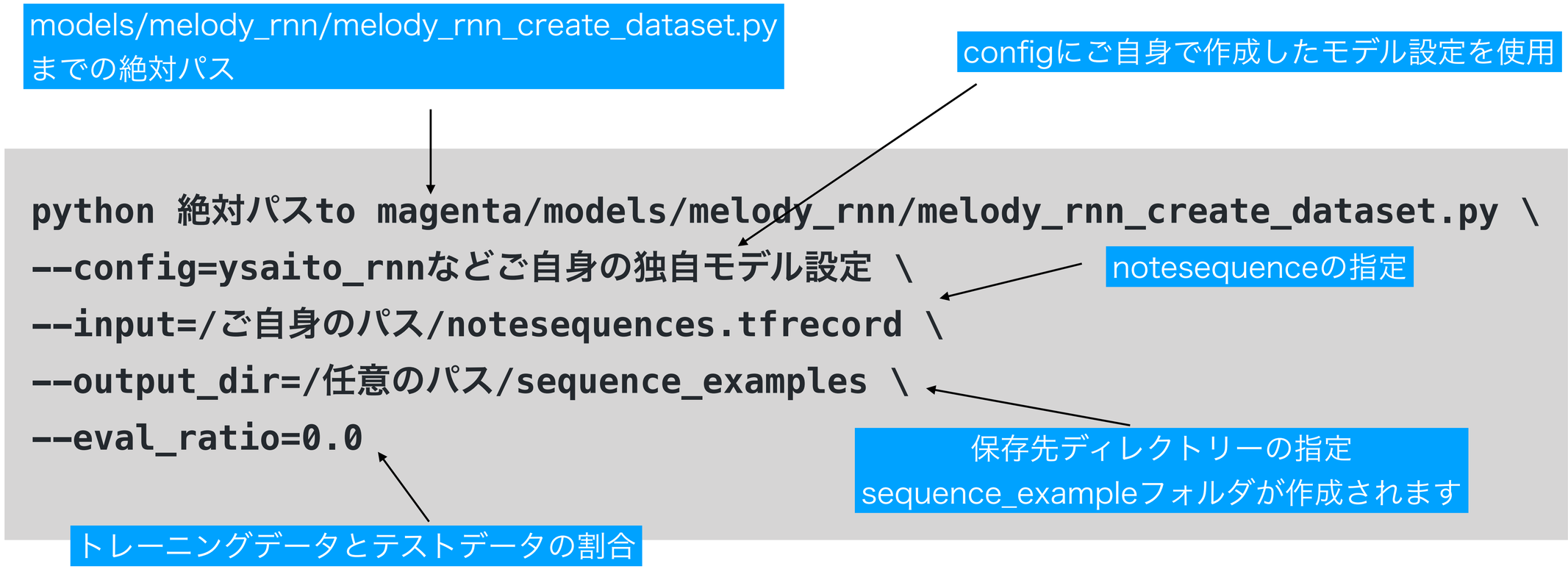
任意の割合でトレーニングデータとテストデータ を作成します。



magenta開発環境では実行は全てpythonファイルになります

5-３・トレーニングデータとテストデータ の作成 Mac

任意の割合でトレーニングデータとテストデータ を作成します。



magenta開発環境では実行は全てpythonファイルになります

5-４・トレーニング　Windows

音楽データを学習させます。

/magenta/models/melody_rnn/melody_rnn_train.py
までの絶対パス

configはご自身の独自モデル設定の名前を指定

```
python 絶対パスto ¥magenta¥models¥melody_rnn¥melody_rnn_train.py ^
--config=ysaito_rnnなどご自身の独自モデル設定 ^
--run_dir=¥任意のパスを指定¥logdir¥run1 ^
--sequence_example_file=¥ご自身のパス¥sequence_examples¥training_melodies.tfrecord ^
--hparams="batch_size=5,rnn_layer_sizes=[64,64]" ^
--num_training_steps=500
```

実行ディレクトリーの指定

トレーニングデータの指定

ハイパーパラメーターの指定
バッチサイズはtfrecordのファイル数よりも少なく指定する
今回は5

学習回数の指定
後から回数を増やせます
最初は５００回

magenta開発環境では実行は全てpythonファイルになります

5-４・トレーニング　Mac

音楽データを学習させます。

/magenta/models/melody_rnn/melody_rnn_train.py
までの絶対パス

configはご自身の独自モデル設定の名前を指定

```
python 絶対パスto /magenta/models/melody_rnn¥melody_rnn_train.py \
--config=ysaito_rnnなどご自身の独自モデル設定 \
--run_dir=/任意のパスを指定/logdir/run1 \
--sequence_example_file=/ご自身のパス/sequence_examples¥training_melodies.tfrecord \
--hparams="batch_size=5,rnn_layer_sizes=[64,64]" \
--num_training_steps=500
```

実行ディレクトリーの指定

トレーニングデータの指定

ハイパーパラメーターの指定
バッチサイズはtfrecordのファイル数よりも少なく指定する
今回は5

学習回数の指定
後から回数を増やせます
最初は５００回

magenta開発環境では実行は全てpythonファイルになります

5-６・tensorboardで学習の確認

学習の結果を視覚的に確認します。

TensorBoardで学習結果を確認できます

```
tensorboard --logdir=/ご自身のパス/logdir
```

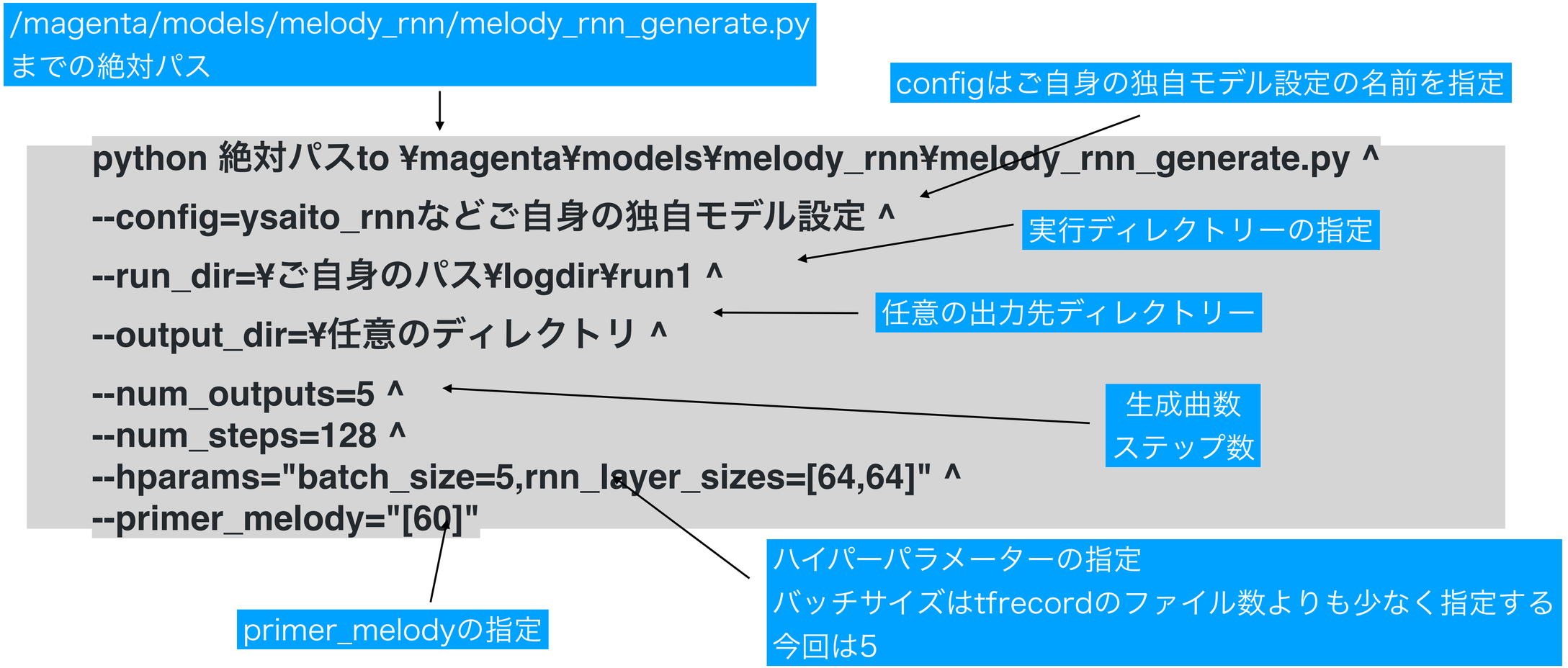
学習を実行したディレクトリー

上記コマンド実行後

http://localhost:6006
にてTensorBoard dashboardにアクセス（コピペで開いてください）

5-７・音楽生成 Windows

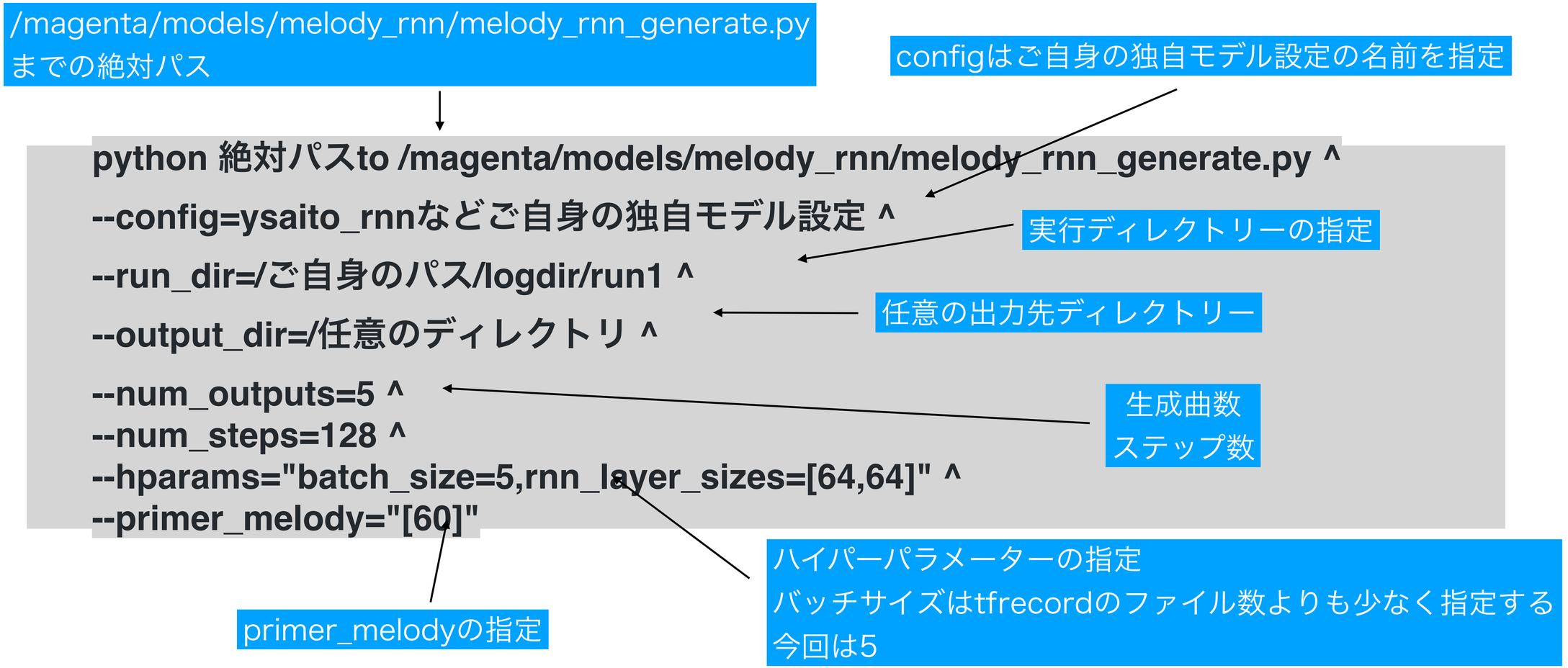
学習させたデータを使用して音楽生成を行なってみましょう。



magenta開発環境では実行は全てpythonファイルになります

5-７・音楽生成 Mac

学習させたデータを使用して音楽生成を行なってみましょう。



magenta開発環境では実行は全てpythonファイルになります

６・独自モデルのBundleファイル（.magファイル）作成

ここで生成したBundlefile(.mag)ファイルとmelody_rnn_model.pyを最終課題で提出してください

5-８・Bundleファイル（.magファイル）作成 Windows

学習済みデータとしてBundleファイルを作成します。

/magenta/models/melody_rnn/melody_rnn_generate.py
までの絶対パス

configはご自身の独自モデル設定の名前を指定

```
python 絶対パスto ¥magenta¥models¥melody_rnn¥melody_rnn_generate.py ^
--config=ysaito_rnnなどご自身の独自モデル設定 ^
--run_dir=¥ご自身のパス¥logdir/run1 ^
--hparams="batch_size=5,rnn_layer_sizes=[64,64]" ^
--bundle_file=¥任意のディレクトリ¥ysaito_rnn.mag ^
--save_generator_bundle
```

実行ディレクトリーの指定

bundleファイル名と
出力ディレクトリーの指定
今回はaimusic_rnn.mag

ハイパーパラメーターの指定
バッチサイズはtfrecordのファイル数よりも少なく指定する
今回は5

magenta開発環境では実行は全てpythonファイルになります

ここで生成したBundlefile(.mag)ファイルとmelody_rnn_model.pyを最終課題で提出してください

5-８・Bundleファイル（.magファイル）作成 Mac

学習済みデータとしてBundleファイルを作成します。

/magenta/models/melody_rnn/melody_rnn_generate.py
までの絶対パス

configはご自身の独自モデル設定の名前を指定

```
python 絶対パスto /magenta/models/melody_rnn/melody_rnn_generate.py \
--config=ysaito_rnnなどご自身の独自モデル設定 ^
--run_dir=/ご自身のパス/logdir/run1 \
--hparams="batch_size=5,rnn_layer_sizes=[64,64]" \
--bundle_file=/任意のディレクトリ/ysaito_rnn.mag \
--save_generator_bundle
```

実行ディレクトリーの指定

bundleファイル名と
出力ディレクトリーの指定
今回はaimusic_rnn.mag

ハイパーパラメーターの指定
バッチサイズはtfrecordのファイル数よりも少なく指定する
今回は5

magenta開発環境では実行は全てpythonファイルになります

ここで生成したBundlefile(.mag)ファイルとmelody_rnn_model.pyを最終課題で提出してください

Melody RNN
ご自身の独自AI作曲モデルで音楽生成

Melody RNN ご自身の独自AI作曲モデルで音楽生成 Windows

```
python 絶対パスto ¥magenta¥models¥melody_rnn¥melody_rnn_generate.py ^
--config=ysaito_rnnなどご自身の独自モデル設定 ^
--bundle_file=ysaito_rnn.magなど作成した独自モデル.magファイルへのパス ^
--output_dir=任意で生成先ディレクトリーを指定 ^
--num_outputs=5 ^
--num_steps=128 ^
--primer_melody="[60]”
```

ご自身で作成した独自モデル.magファイルへのパスを指定

学習データ5曲、学習回数５００回では圧倒的に不足です。（magentaのモデルほどの精度は出せません）
特にデータ数はたくさん必要で、それに合わせて学習回数は増えます。
今後実践していく場合は是非たくさんの音楽データを用意して実践してください。

Melody RNN ご自身の独自AI作曲モデルで音楽生成 Mac

```
python 絶対パスto /magenta/models/melody_rnn/melody_rnn_generate.py \
--config=ysaito_rnnなどご自身の独自モデル設定 \
--bundle_file=ysaito_rnn.magなど作成した独自モデル.magファイルへのパス \
--output_dir=任意で生成先ディレクトリーを指定 \
--num_outputs=5 \
--num_steps=128 \
--primer_melody="[60]”
```

ご自身で作成した独自モデル.magファイルへのパスを指定

学習データ5曲、学習回数５００回では圧倒的に不足です。（magentaのモデルほどの精度は出せません）
特にデータ数はたくさん必要で、それに合わせて学習回数は増えます。
今後実践していく場合は是非たくさんの音楽データを用意して実践してください。

最終課題

提出ファイル

・melody_rnn_model.py

ご自身のモデルを記述した（コピーして提出。オリジナルを削除しない事）

ご自身の名前.mag

ご自身の名前をファイル名に使用した学習済みデータ(.magファイル）

の２ファイルです。

配布したサンプルコード

melody_rnn_model.py
169~185行目

が講師が自身の名前で作成した

モデル設定サンプルコードです。

難しければこれをコピペで使用してください。