AI（機械学習）音楽プログラミング

第８回

Magentaで音楽データを学習
独自のAI作曲モデル作り

教師あり学習
教師なし学習

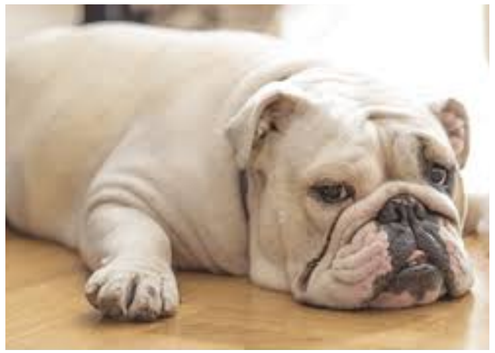
教師あり学習 (Supervised Learning)

学習データに正解となるラベルを付けて学習させる方法。

犬の種類を教えてくれるAIの場合、学習データ（画像）に対して、これは「柴犬」、これは「ダックスフンド」これは「ブルドッグ」というように正解のラベルを付けて学習させます。

柴犬

ダックスフンド



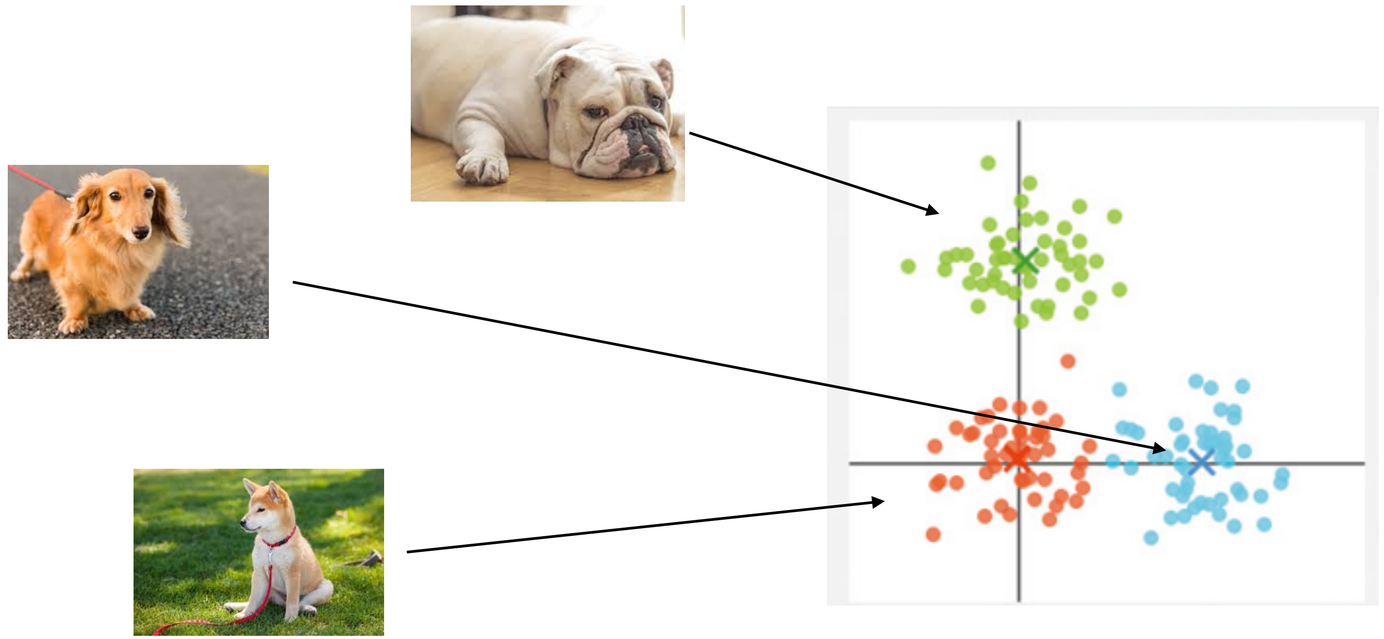
ブルドッグ

教師なし学習

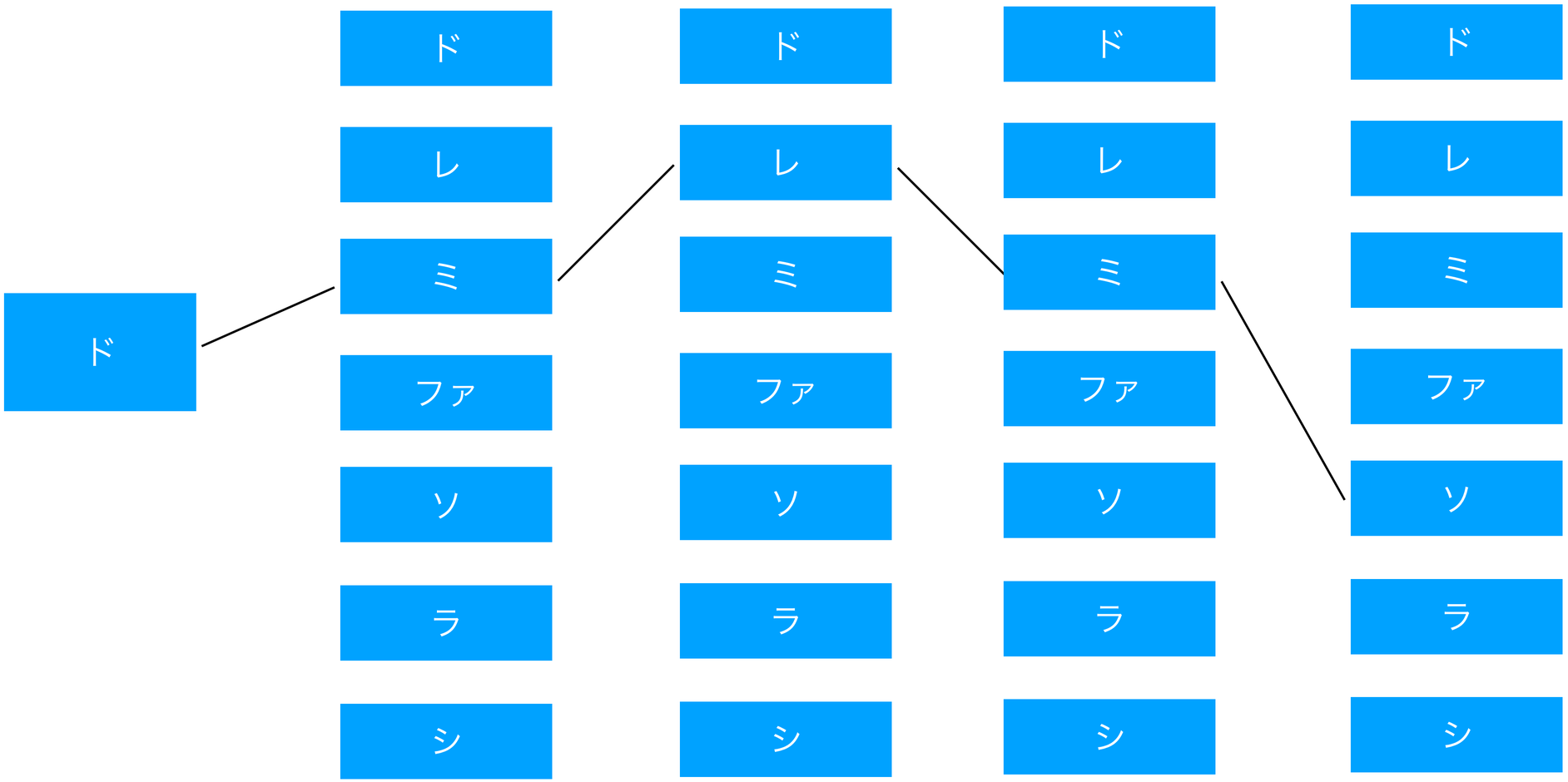
(Unsupervised Learning)
学習データにラベルを付けないで学習する方法。（Magentaは教師なし学習です）

画像を例にすると、ランダムに正解ラベルなしに画像を学習させ、AI自身が犬種を分類できる様になる学習。（クラスタリングによる分類など）

人間の脳の成長・進化に近いため、高度なAI実現のためにはより注目されている。



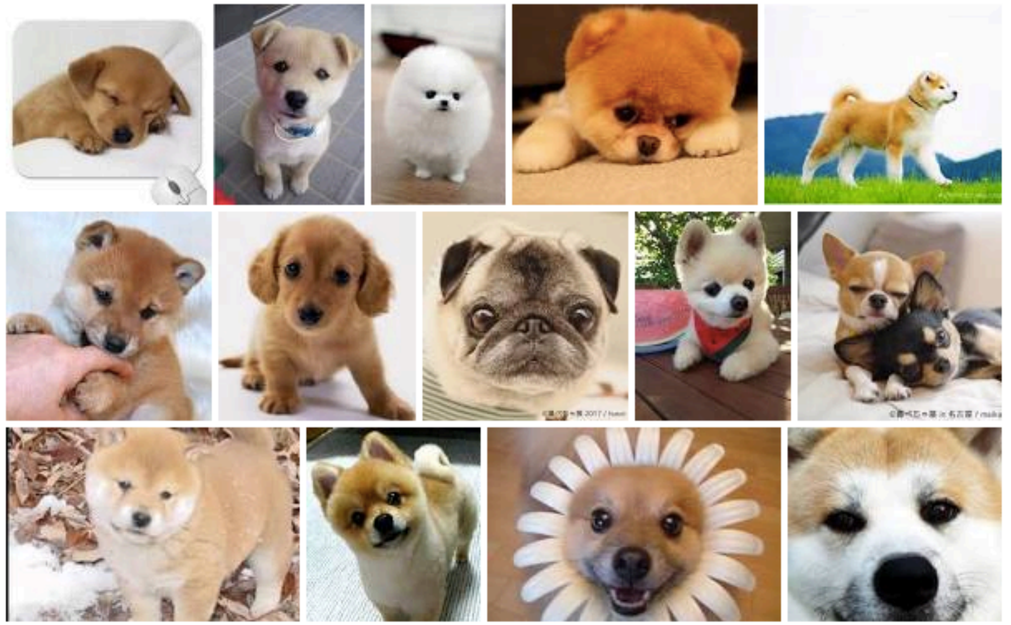
音楽（Magenta）ではメロディーの次の音の出現確率を学習します



過学習

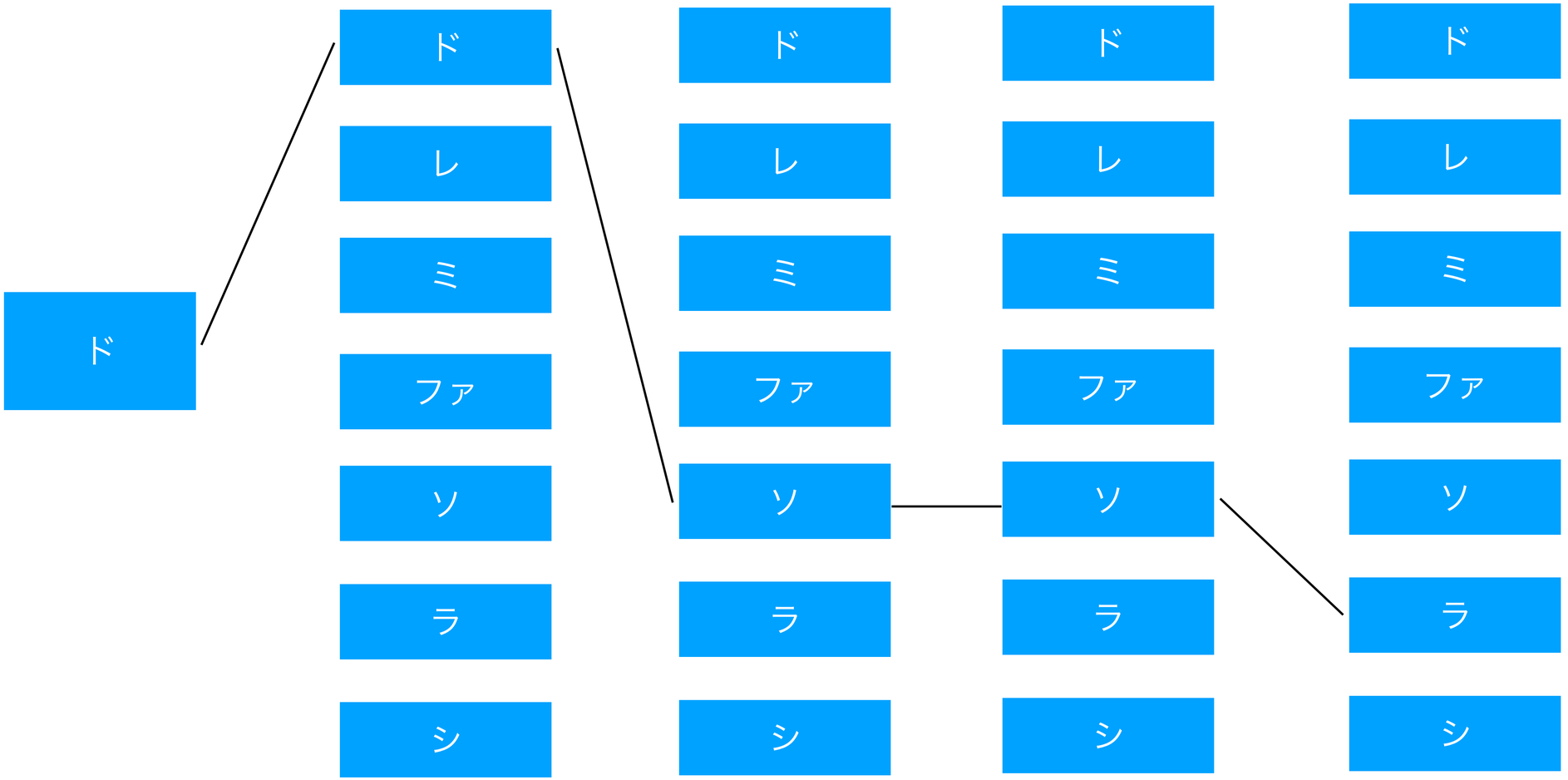
過学習とは

ダックスフンド
だけを学習しすぎると



ダックスフンドのみを正解にしてしまい
他の犬種を犬として認識しない

音楽で過学習になってしまうとその曲しか生成できなくなります

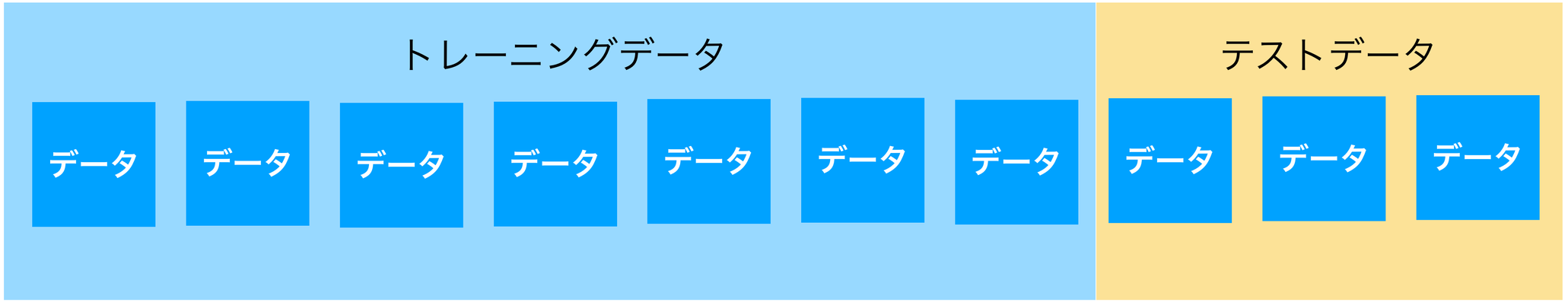


データ数に応じ学習回数を適切に設定しましょう

トレーニングデータ
とテストデータ

トレーニングデータとテストデータ

機械学習の学習データは、トレーニングデータ(訓練データ)とテストデータ（評価データ、検証データ）に分けられる事が多いです。
学習はトレーニングデータのみを使用し、テストデータ によってその精度を検証します。
教師あり学習の場合、データには正解データ(ラベル付けされた教師データ)がペアとして与えられます。
通常、訓練データはテストデータよりも多く設定されます。(7:3や8:2など)
５０対５０など半分ずつの場合もあり、各ケースによりますので明確な基準はありません。

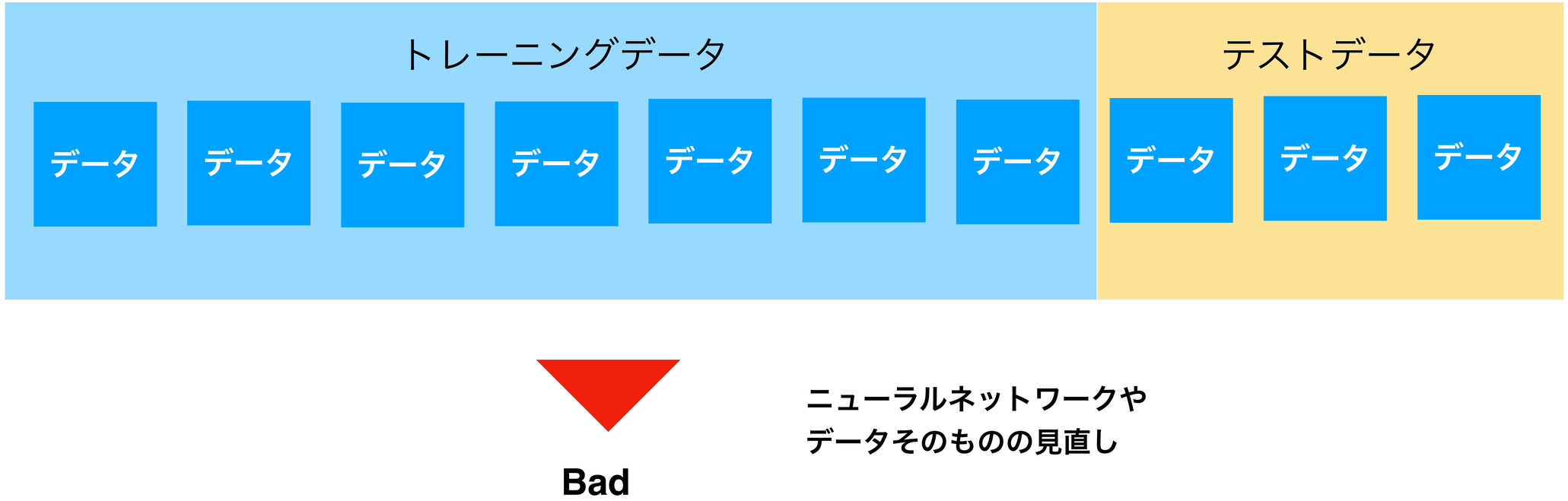


トレーニングデータとテストデータ

トレーニングデータでの学習結果がテストデータ でも良い結果が出せれば未知のデータにも対応できる良い状態です。

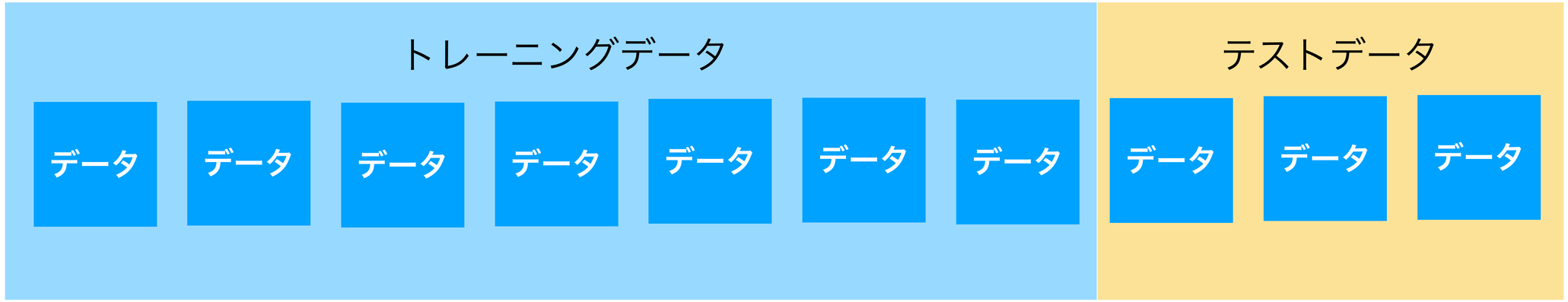
トレーニングデータとテストデータ

トレーニングデータでそもそも良い結果がでなければニューラルネットワークの各設定(**ハイパーパラメーター**など)やデータそのものを見直す必要もあります。



トレーニングデータとテストデータ

トレーニングデータで良い結果が出てもテストデータ で良い結果とならなければ**過学習**な状態と言えます。
（音楽ではトレーニングデータと同じ曲しか生成できなくなります）



Good

未知のデータには対応できない

II

過学習

Bad

ハイパーパラメーター

ハイパーパラメーター

ハイパーパラメータとは

ニューラルネットワークの**ニューロン数**や**ドロップアウト率**、**学習率**といったパラメータなど、人力で調整しなければいけない設定の事を指します。
ニューラルネットワークのチューニングとはこのハイパーパラメーターの調整の事を指します。
ハイパーパラメータの調整(チューニング)の方法は手動調整と自動調整があります。

自動調整には

・**グリッドサーチ**: 総当たりでパラメータを探索する
・**ランダムサーチ**: パラメータの組み合わせをランダムに探索する

などがあります。

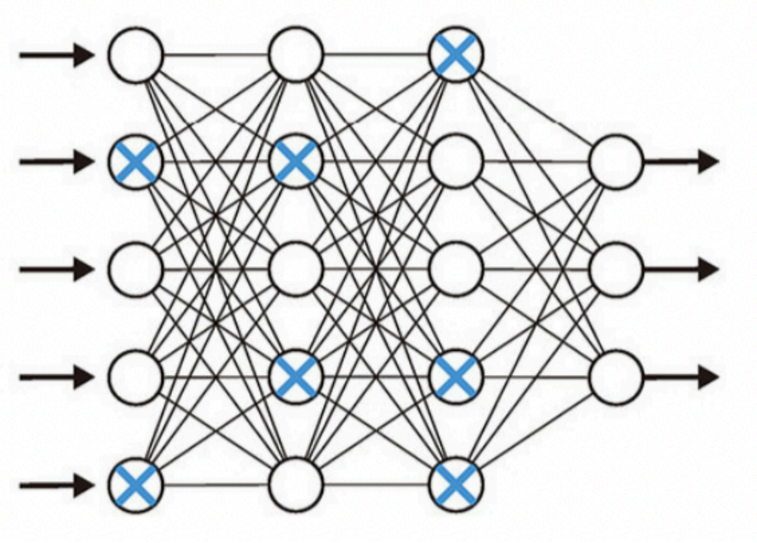
自動調整に使用できる代表的なライブラリーは
Scikit Learnや**Keras**などです。

ドロップアウト

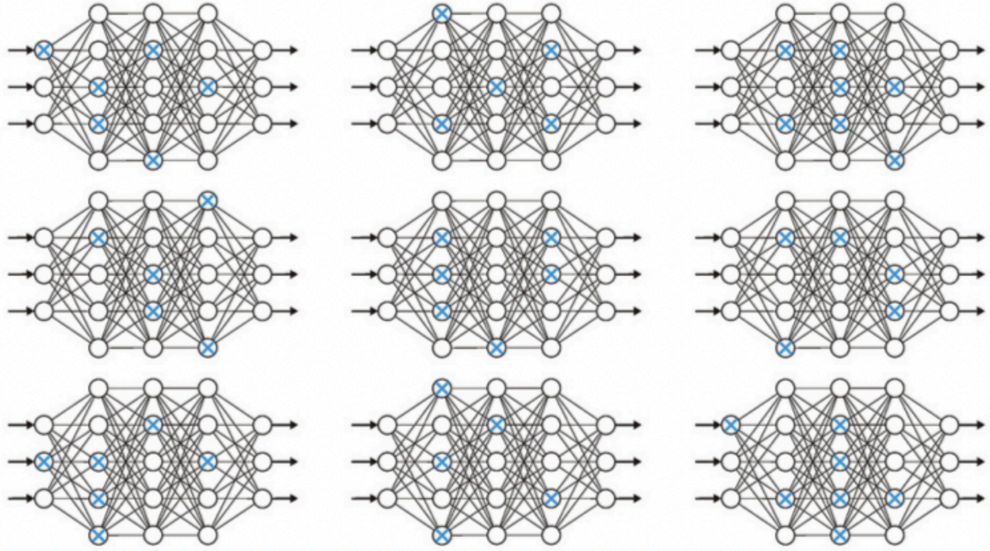
ドロップアウト

ドロップアウトとは

出力層以外のニューロンを一定の割合でランダムに停止し過学習を防ぐ方法。
この割合の事をドロップアウト率と呼ぶ。
規模の大きなニューラルネットワークは過学習を起こす事が多いが、ドロップアウトを利用する事で規模を小さくする事ができます。
またドロップアウトの構成を変更した複数のネットワークを利用し結果の向上につなげる事を機械学習ではアンサンブル効果と呼んでいます。



ドロップアウト



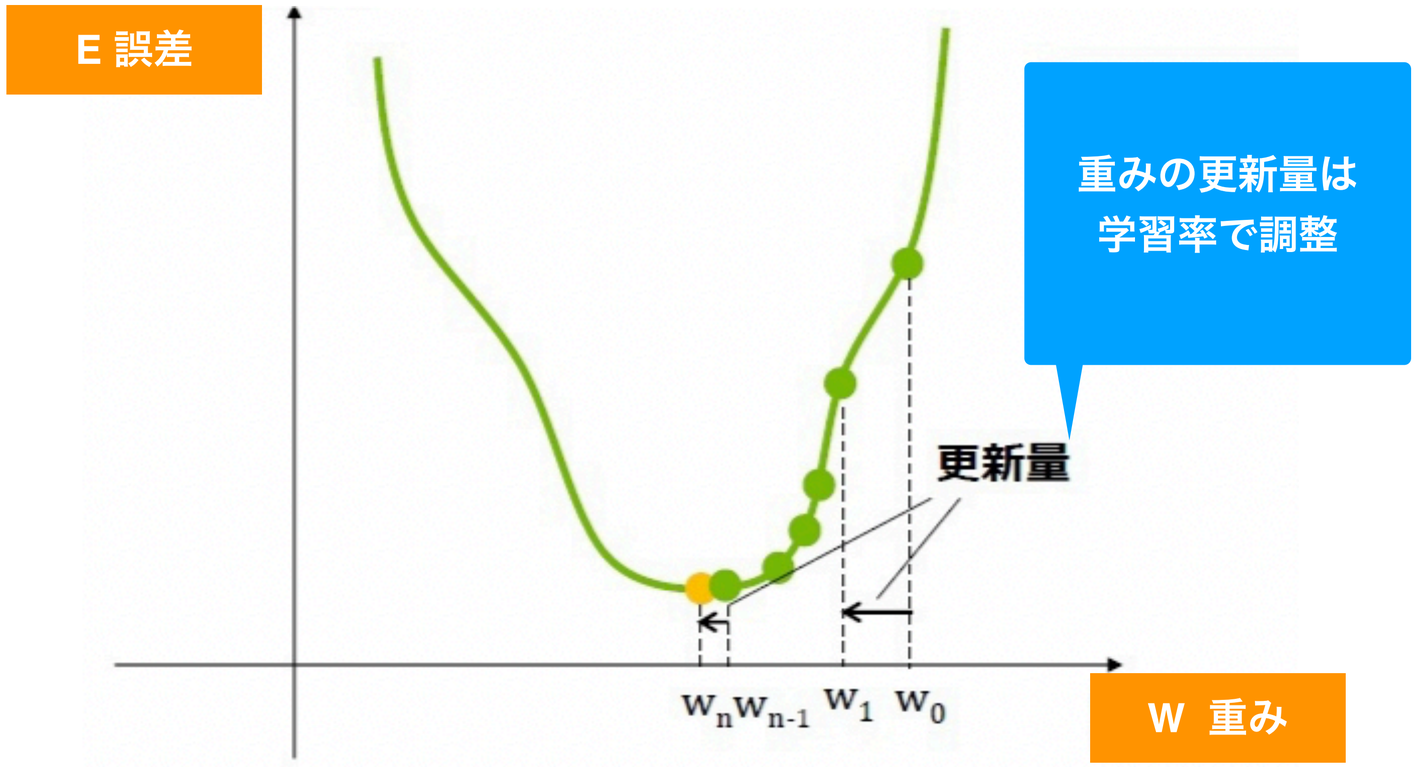
アンサンブル効果

学習率

学習率

学習率（学習係数とも呼びます）とは
機械学習の最適化(重みやバイアスの更新)においてどのくらい値を動かすかというパラメーター。

学習率を大きくしすぎると発散し(結果最適化できず)、小さくしすぎると学習完了までの時間がかかります。



※学習率は数学では

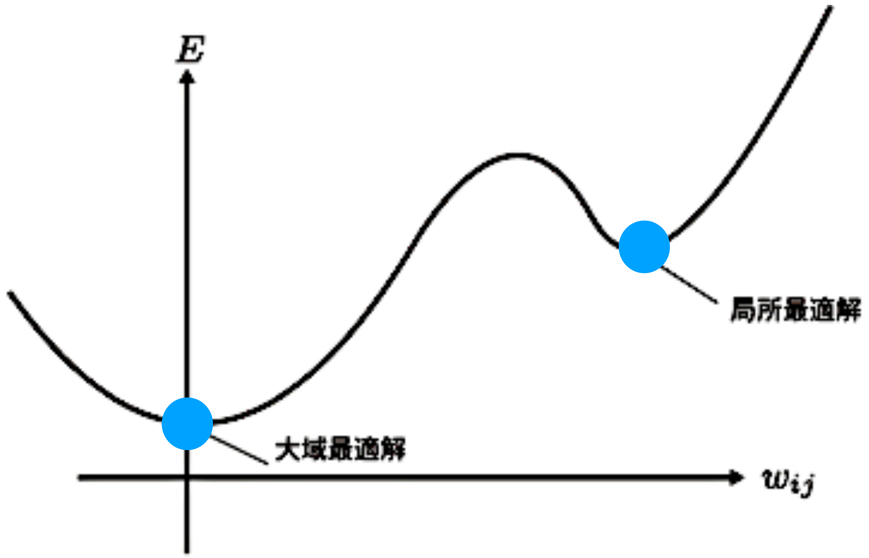
ηで表す

最適化とは

プログラムなどがあらかじめ定められた判定基準に照して，最も望ましい値をとるようにすること。
機械学習で言えば勾配が最小である状態に至るもっとも効率の良い確実なパラメーターを選択する事。

最適化アルゴリズムとは

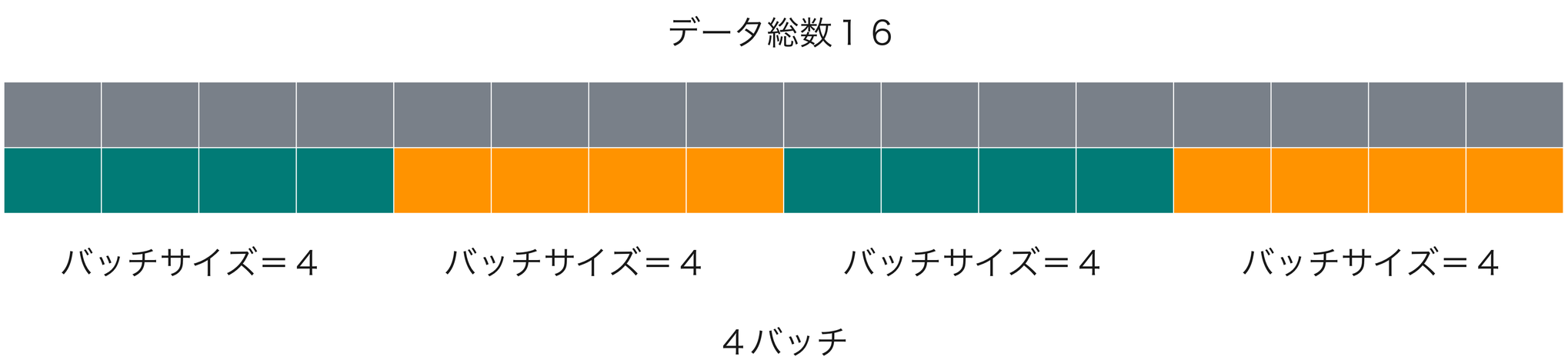
機械学習で勾配が最小、つまり大域最適解に早く確実にたどり着くためのアルゴリズム



バッチサイズ

バッチサイズ

機械学習においては、通常手持ちのデータをまとめて全部学習に使うようなことはしません。
全データをトレーニングデータとテストデータに分け、さらにトレーニングデータもいくつかの
バッチ（かたまり）に分割して、バッチごとに重みの最適化を実行します。
言い換えれば、バッチとは重みの更新間隔です。
1 バッチに含まれるデータ数を バッチサイズ とよびます。
一般的にはどのバッチもサイズが等しくなるように訓練データを均等分割します。



Epoch

Epoch

すべてのトレーニングデータについて学習し終えた段階を 1 エポック (epoch) といいます。
たとえば、5000 個のトレーニングデータをサイズ 100 のバッチに分割した場合、バッチ数は 50 個となるので、50 バッチの学習を終えたときに 1 エポックとなります。
下の表で説明すると、
１６個のトレーニングデータをサイズ ４ のバッチに分割した場合、バッチ数は ４個となるので、 ４ バッチの学習を終えたときに 1 エポックとなります。

データ総数１６

バッチサイズ＝４　　バッチサイズ＝４　　バッチサイズ＝４　　バッチサイズ＝４

（１エポック＝４バッチ）

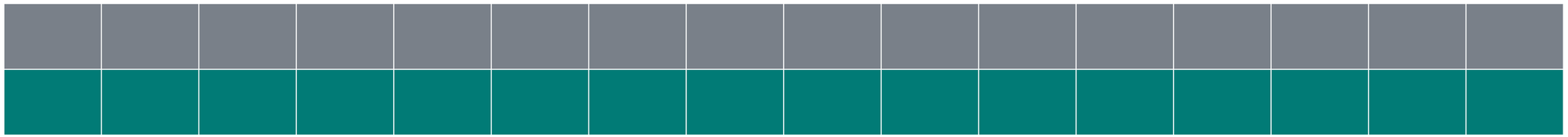
バッチ学習

バッチ学習

バッチサイズがデータの総数と等しい場合、の学習をバッチ学習とよびます。
全トレーニングデータの数とバッチサイズが等しくなるので、重みの更新間隔は 1 エポックとなります。
手持ちのデータをすべて使うので、個々のデータが学習に与える影響は相対的に小さくなり、極端に間違った方向に学習を進めることがありません。
ただし局所最適解に陥る可能性があります。

新しいデータが追加されると、すべてのデータを使って学習をやり直さなくてはならないので、古いデータも保管しておく必要があり、十分なメモリを確保できる環境が要求されます。

データ総数１６



バッチサイズ＝（１バッチのデータ数）１６　　（１エポック＝１バッチ）

オンライン学習

オンライン学習

バッチサイズを１にした場合はオンライン学習とよばれます。
オンライン学習では１回の学習で１個のデータのみを使用します。
学習に使ったデータを捨てることができるので、
「ビッグデータを使って学習させたいけれど、すべてのデータをメモリに保管できない」
というような状況で有効な学習法です。
また、学習中にも逐次追加データが入ってくるような状況（天気予報やスパム分類など）にも、
オンライン学習が適しています。

データ総数１６

バッチサイズ＝（１バッチのデータ数１）　　（１エポック＝１６バッチ）

ミニバッチ学習

ミニバッチ学習

バッチ学習とオンライン学習の中間です。
全データをトレーニングデータとテストデータに分け、さらにトレーニングデータもいくつかのバッチに分割して、バッチごとに重みとバイアスの最適化を実行します。
バッチとは重みとバイアスの更新間隔です。
例えばデータ総数１６バッチサイズ４とすれば１Epoch辺り４回の更新が行われます。

ミニバッチ学習は
バッチ学習に比べ局所最適解に囚われにくく
オンライン学習の様におかしな方向に学習が進む事がありません。

データ総数１６

バッチサイズ＝４　　（１エポック＝４バッチ）

Magenta
独自モデルの作成

Magenta 独自モデルの作成

Magentaで音楽データを学習させ独自のAI作曲モデルを作成します。
まずは音楽データを学習させ**独自の学習済みデータ**を作成します。
今回は配布するMIDIファイルを使用して学習しますが、もちろんご自身の音楽ファイルを使用する事も可能です。
学習とモデル作成の方法を覚えていただく事が目的のため、音楽データ数は５曲です。
本来全くデータ数としては足りない（最低でも１００曲、汎用性を求めるのであれば数万曲は必要かもしれません）ので実際の音楽生成実践の際は多数のデータを（時間をかけ少しづつでも）用意しましょう。
データ数が足りない、または適切なデータでない場合はMagentaで配布している学習済みデータの方が圧倒的に整合性のある音楽が生成されます。
この講義の中では時間的にもそこまでのレベルのデータとモデルを作成する事は難しいですが、いずれは是非チャレンジしてみてください。

Magenta
学習データ作成

Magenta 学習データ作成の流れ

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。

配布したmdata5をご使用ください。

８小節の単音メロディー５曲のmidiファイル

が入っているフォルダーです。

仮想環境はMagenta用（magentaenvなど)にしてください

```
conda activate magentaenv
```

Melody RNN
1・midiファイルの用意

１・midiファイルの用意

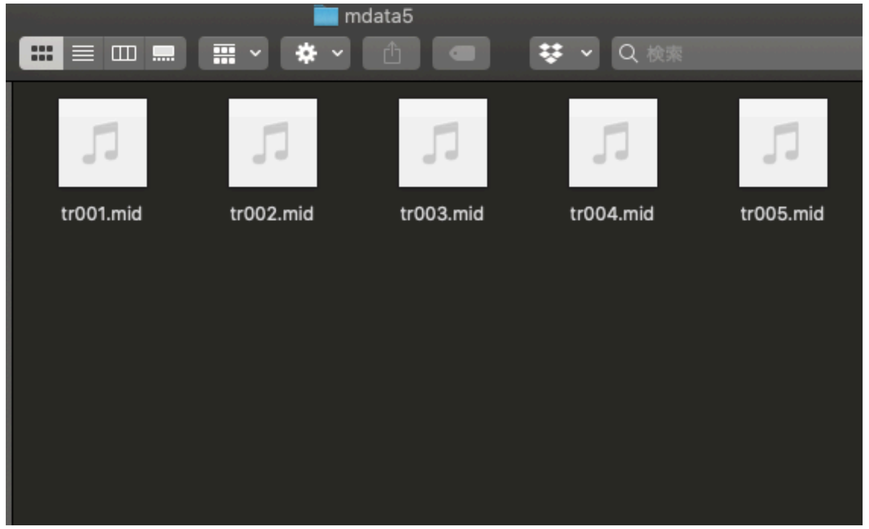
midiファイルを作成し任意のフォルダーへ。配布したmidiデータが入ったフォルダーを使用してください。

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

Melody RNN
２・NoteSequence（tfrecord）の作成

２・NoteSequence（tfrecord）の作成 Windows

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。

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）

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

```
convert_dir_to_note_sequences ^
--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
```

Notesequence作成の実行

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。

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）

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

```
convert_dir_to_note_sequences \
--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
```

Notesequence作成の実行

Melody RNN
３・トレーニングデータとテストデータ の作成

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。

configを一つ指定
basic, mono, lookback, attention
のいずれか。今回はbasic_rnnを使用

notesequenceの指定

```
melody_rnn_create_dataset ^
--config=basic_rnn ^
--input=/ご自身のパス/notesequences.tfrecord ^
--output_dir=/任意のパス/sequence_examples ^
--eval_ratio=0.0
```

保存先ディレクトリーの指定
sequence_exampleフォルダが作成されます

トレーニングデータとテストデータの割合

２つのtfrecordファイルが作成されます。（今回はテストデータは使用しません）
トレーニングデータとテスト（eval) データです。
eval_ratioはテストデータの割合です。
例：0.10＝トレーニングデータ９０%、評価データ１０%

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。

configを一つ指定
basic, mono, lookback, attention
のいずれか。今回はbasic_rnnを使用

notesequenceの指定

```
melody_rnn_create_dataset \
--config=basic_rnn \
--input=/ご自身のパス/notesequences.tfrecord \
--output_dir=/任意のパス/sequence_examples \
--eval_ratio=0.0
```

保存先ディレクトリーの指定
sequence_exampleフォルダが作成されます

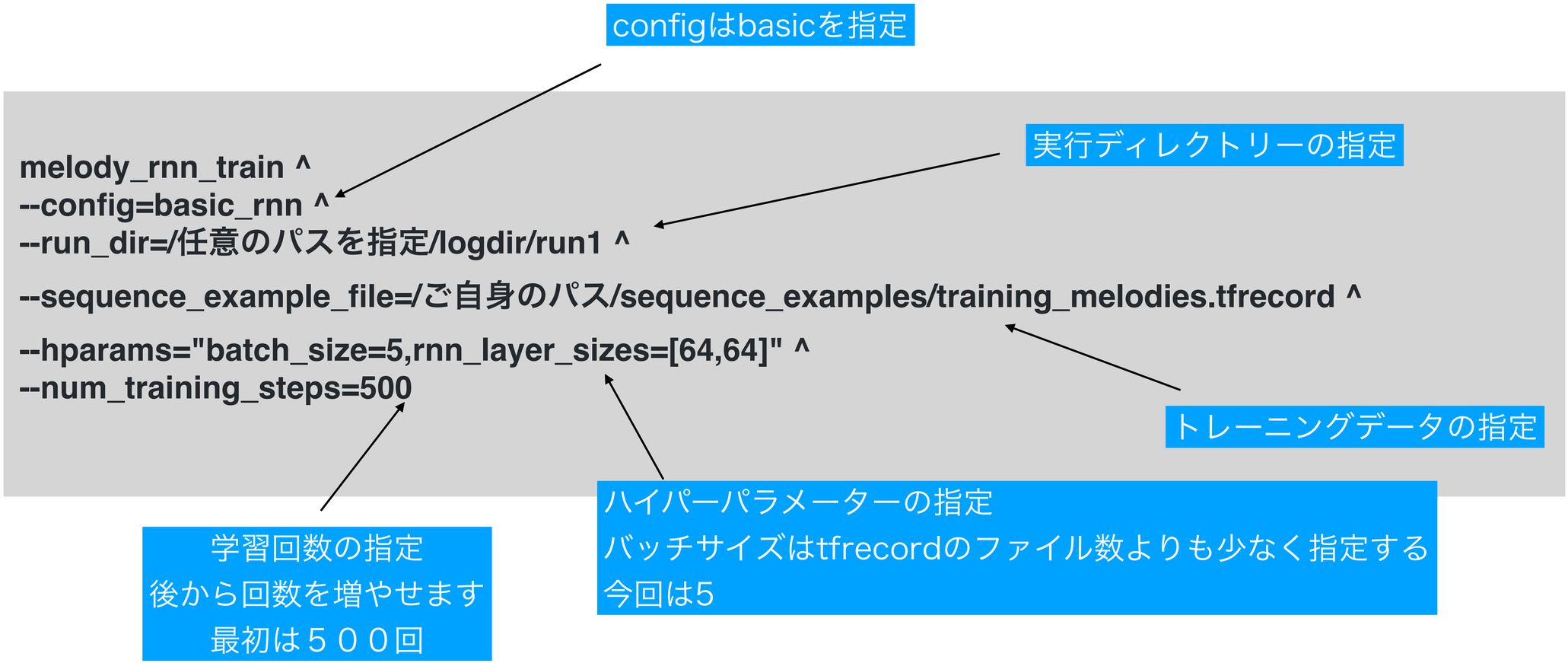
トレーニングデータとテストデータの割合

２つのtfrecordファイルが作成されます。（今回はテストデータは使用しません）
トレーニングデータとテスト（eval) データです。
eval_ratioはテストデータの割合です。
例：0.10＝トレーニングデータ９０%、評価データ１０%

Melody RNN
４・トレーニング

４・トレーニング　Windows

音楽データを学習させます。



４・トレーニング　Mac

音楽データを学習させます。

configはbasicを指定

実行ディレクトリーの指定

```
melody_rnn_train \
--config=basic_rnn \
--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
```

トレーニングデータの指定

学習回数の指定
後から回数を増やせます
最初は５００回

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

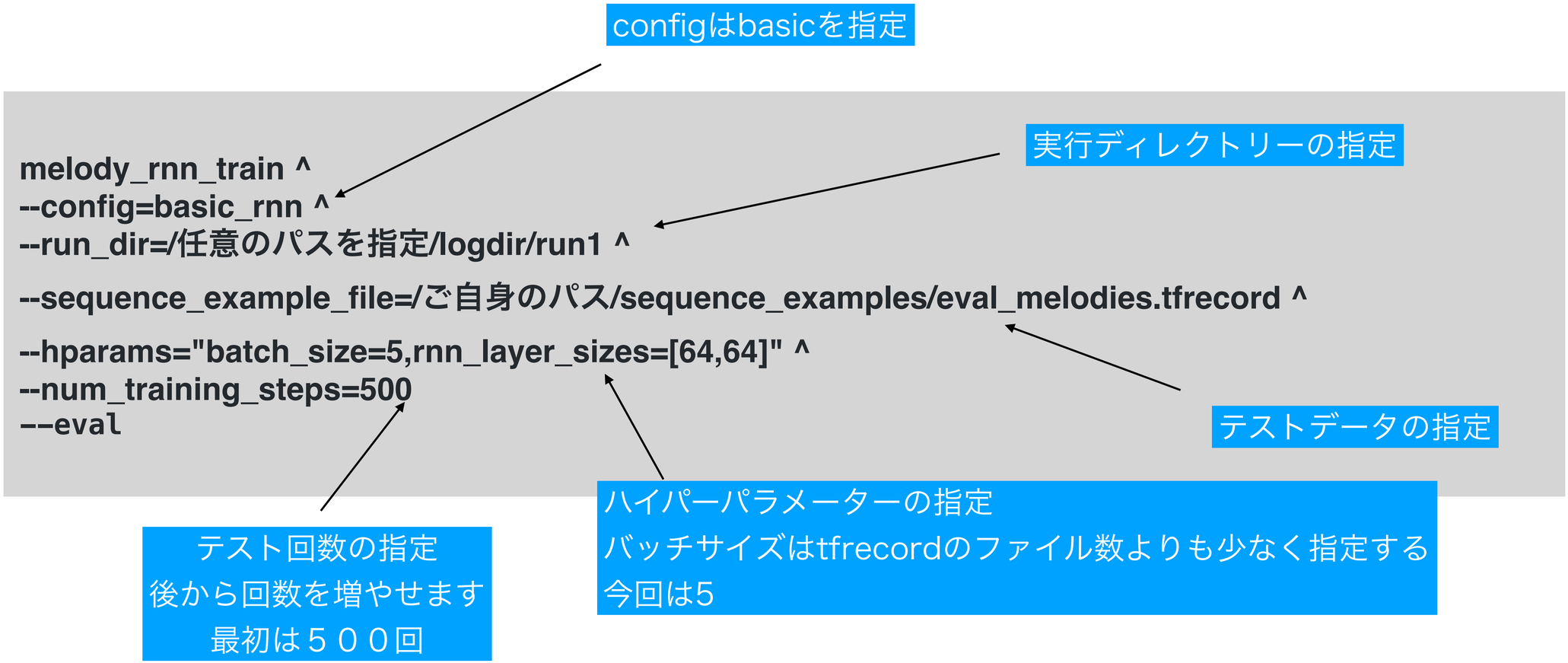
```
(magentaenv) YoshihiroSaito@MacBook-Air-2 ~ % melody_rnn_train \
--config=basic_rnn \
--run_dir=/Users/YoshihiroSaito/Downloads/logdir/run1 \
--sequence_example_file=/Users/YoshihiroSaito/Downloads/sequence_examples/training_melodies.tfrecord \
--hparams="batch_size=5,rnn_layer_sizes=[64,64]" \
--num_training_steps=500
```

５００回学習させています

Melody RNN
５・テスト

５・テスト　Windows

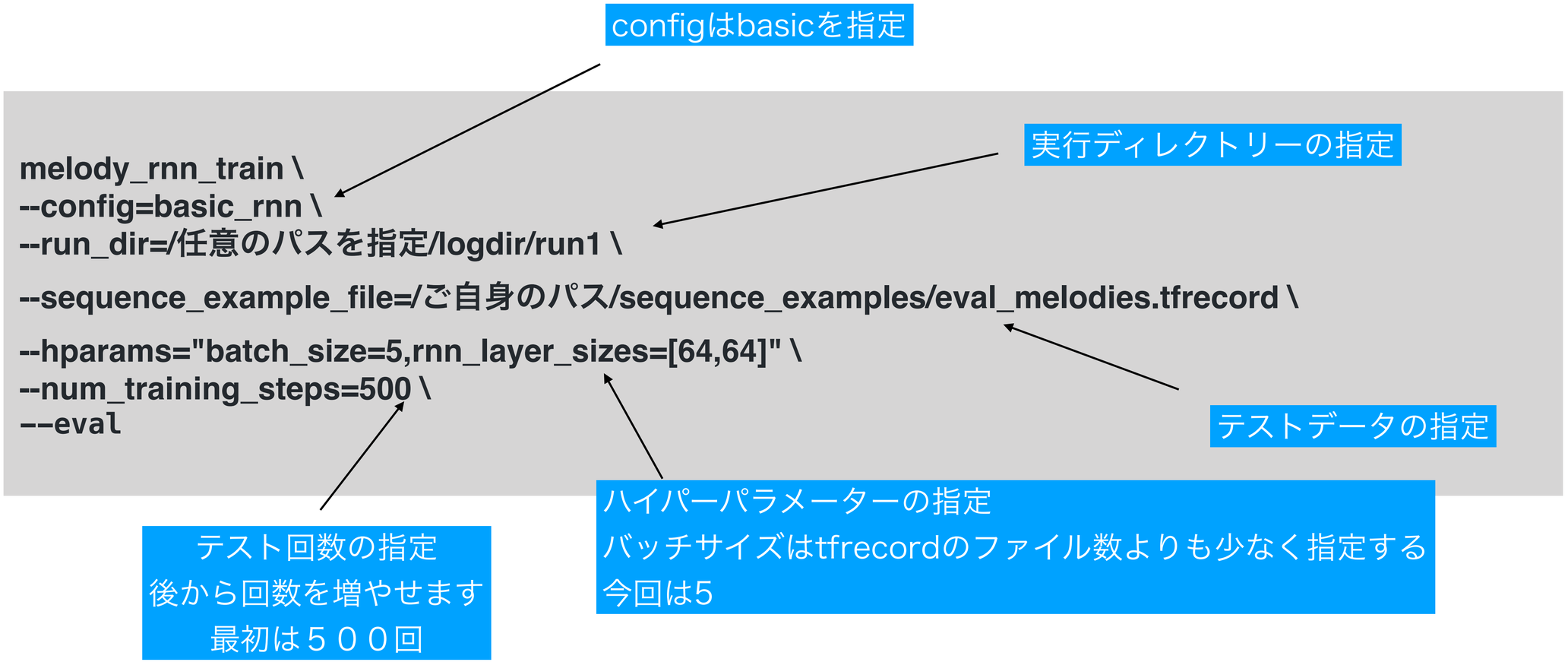
学習検証のためテストデータでテストをします（解説しますが、現在テストはできない様です）



テストは検証のみですので重みやバイアスの更新が行われません

５・テスト　Mac

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）



テストは検証のみですので重みやバイアスの更新が行われません

Melody RNN
６・tensorboardで学習の確認

６・tensorboardで学習の確認

学習の結果を視覚的に確認します。

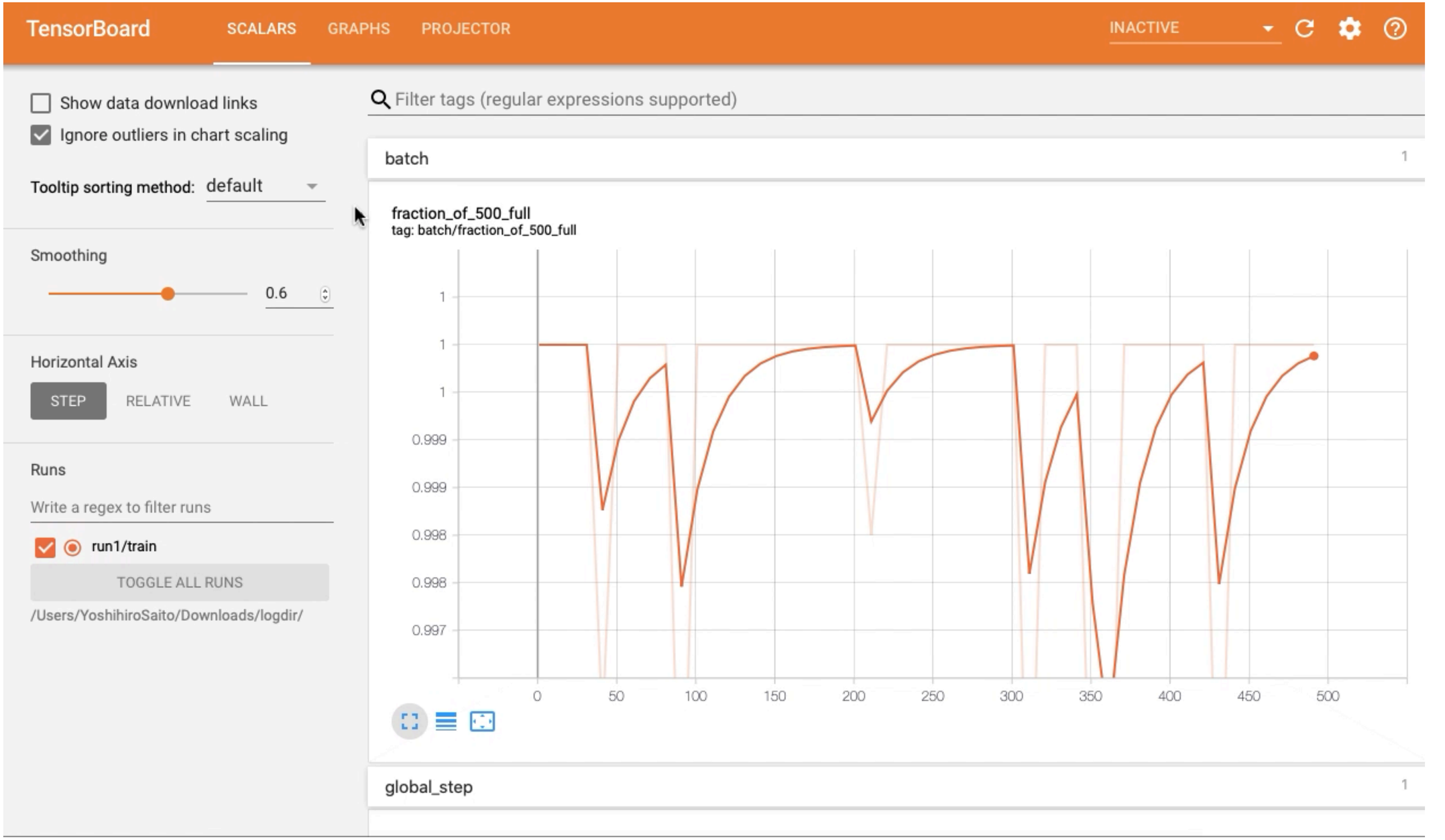
TensorBoardで学習結果を確認できます

```
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

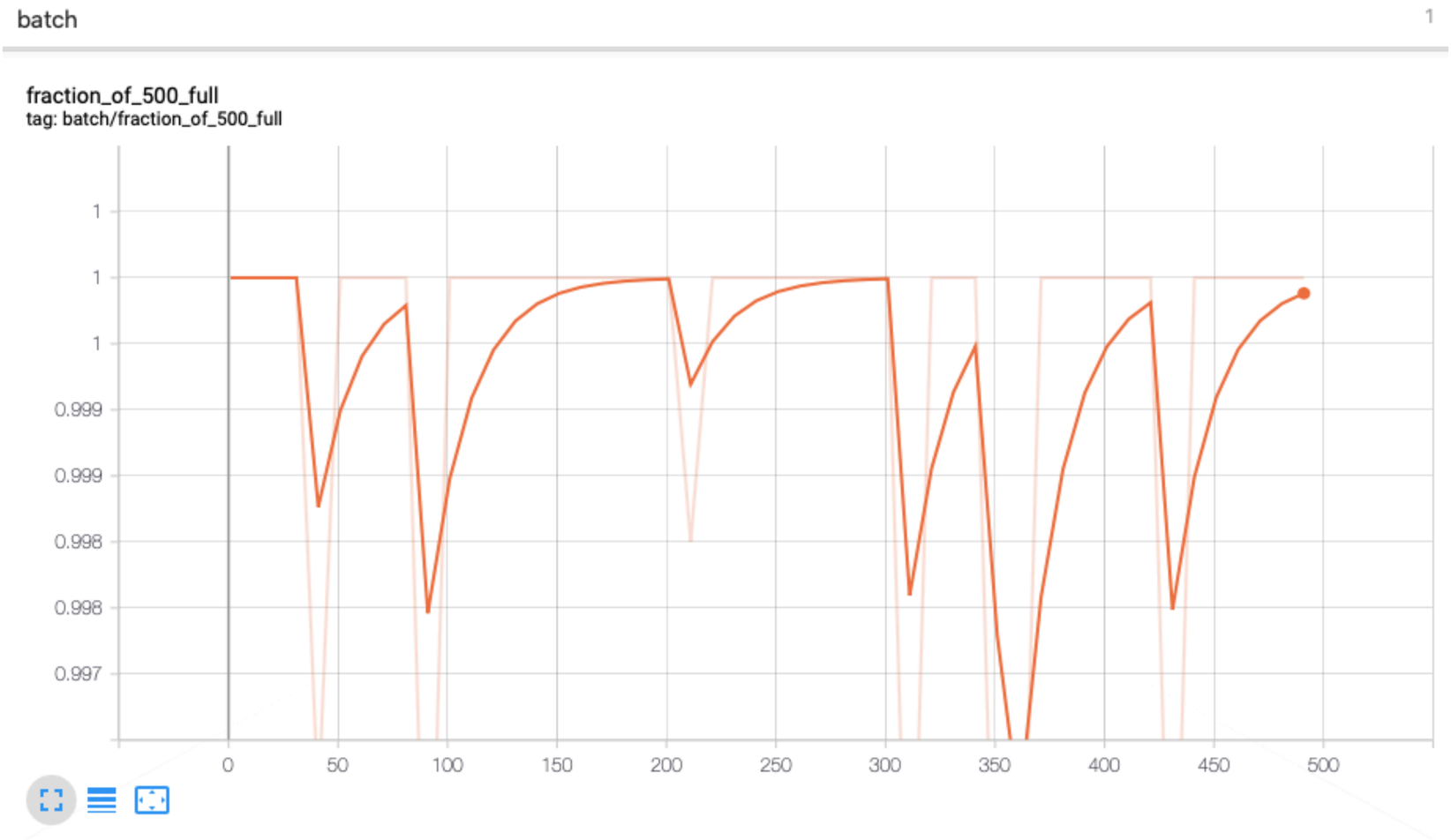
http://localhost:6006
にてTensorBoard dashboardにアクセス（コピペで開いてください）



tensorboard

６・tensorboardで学習の確認

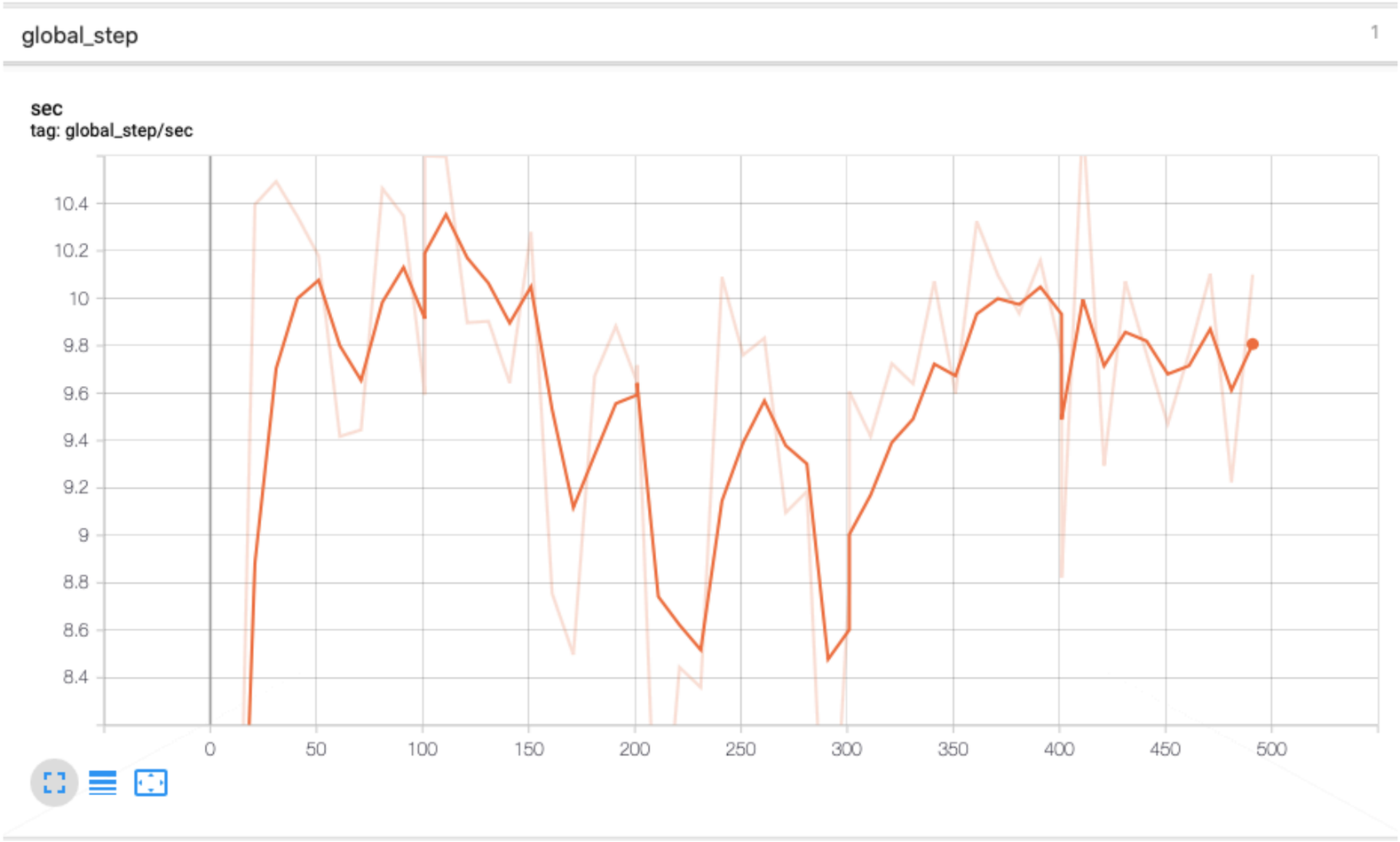
学習の結果を視覚的に確認します。



各バッチごとにどれだけのデータ数が学習されているのか確認

６・tensorboardで学習の確認

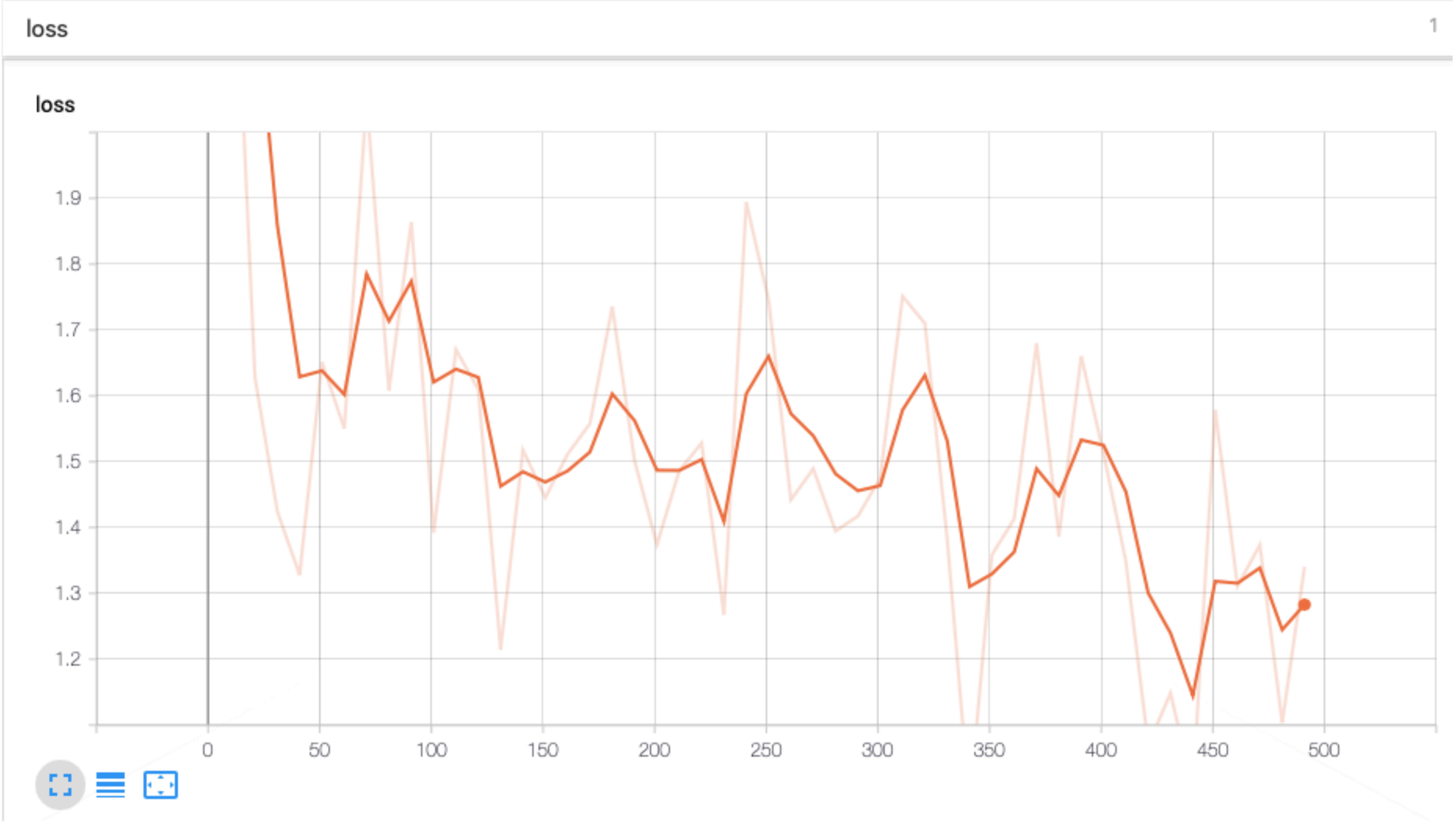
学習の結果を視覚的に確認します。



各ステップごとおよび全体の学習にかかった時間

６・tensorboardで学習の確認

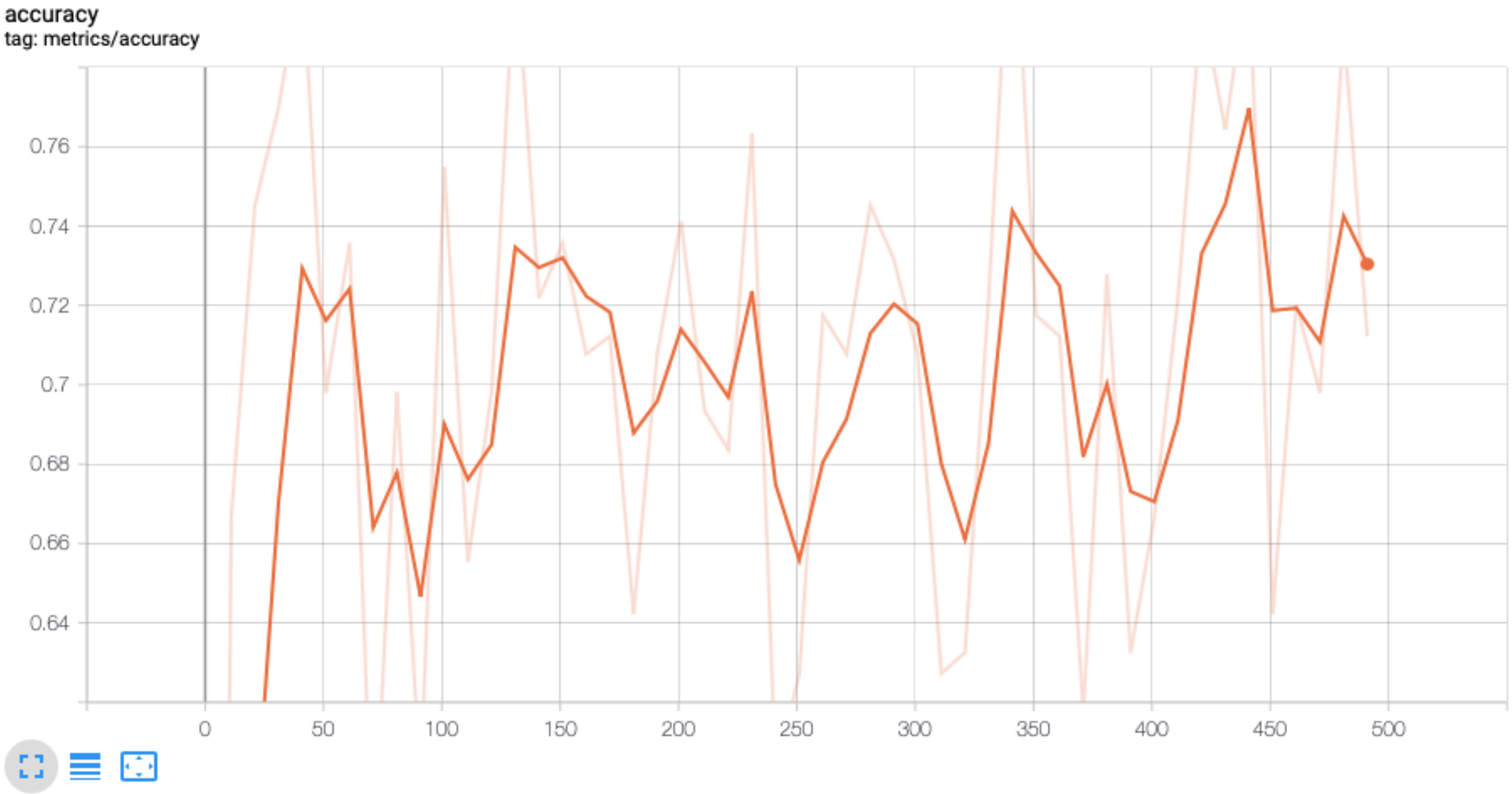
学習の結果を視覚的に確認します。



損失（エラー）の推移

６・tensorboardで学習の確認

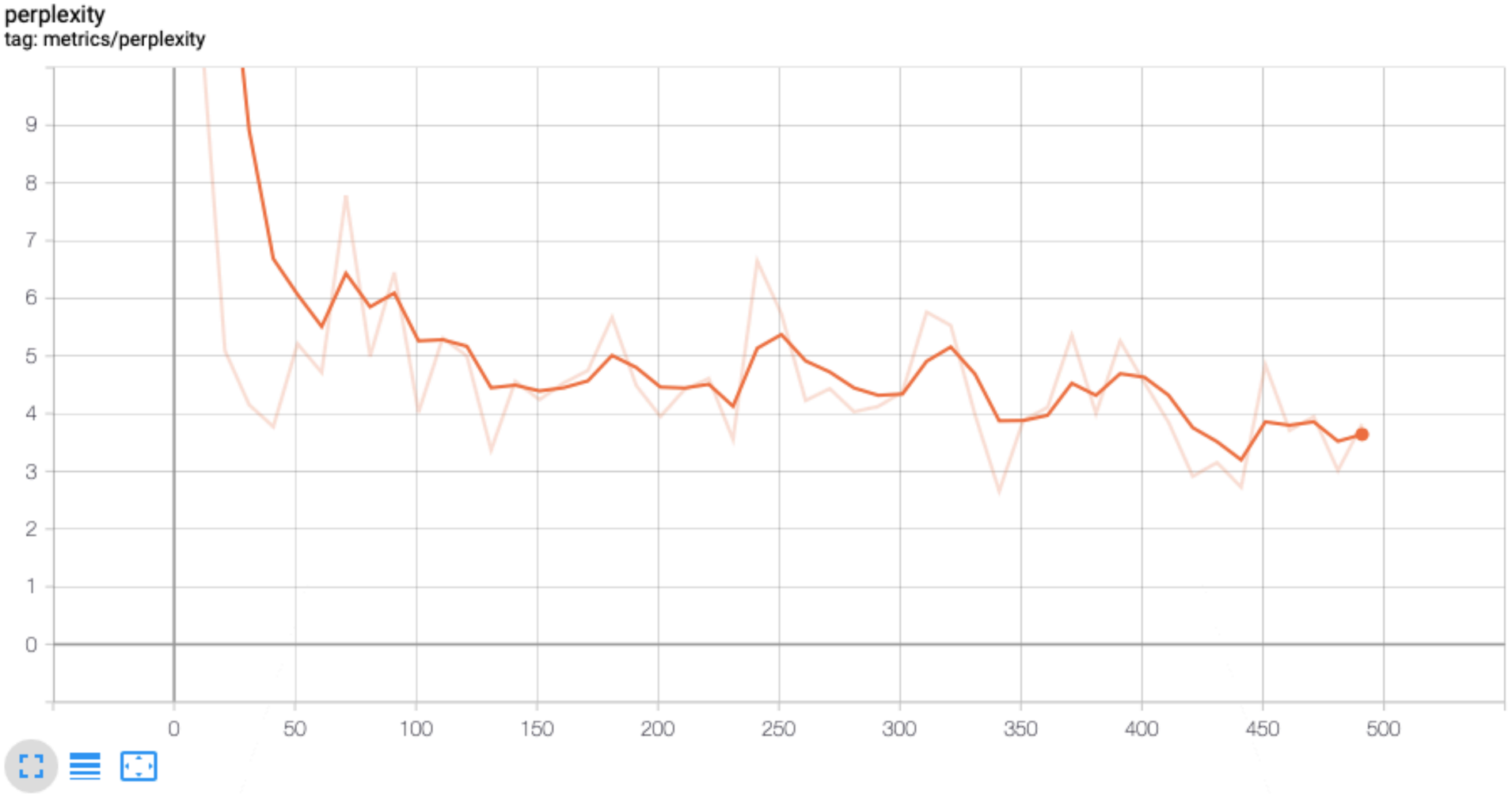
学習の結果を視覚的に確認します。



Accuracy（精度）の推移

６・tensorboardで学習の確認

学習の結果を視覚的に確認します。



Perplexityの推移

Melody RNN
７・音楽生成

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。

学習データで音楽生成

```
melody_rnn_generate ^
--config=basic_rnn ^
--run_dir=/ご自身のパス/logdir/run1 ^
--output_dir=/任意のディレクトリ ^
--num_outputs=5 ^
--num_steps=128 ^
--hparams="batch_size=5,rnn_layer_sizes=[64,64]" ^
--primer_melody="[60]"
```

configはbasicを指定

実行ディレクトリーの指定

任意の出力先ディレクトリー

生成曲数
ステップ数

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

primer_melodyの指定

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。

学習データで音楽生成

```
melody_rnn_generate \
--config=basic_rnn \
--run_dir=/ご自身のパス/logdir/run1 \
--output_dir=/任意のディレクトリ \
--num_outputs=5 \
--num_steps=128 \
--hparams="batch_size=5,rnn_layer_sizes=[64,64]" \
--primer_melody="[60]"
```

configはbasicを指定

実行ディレクトリーの指定

任意の出力先ディレクトリー

生成曲数
ステップ数

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

primer_melodyの指定

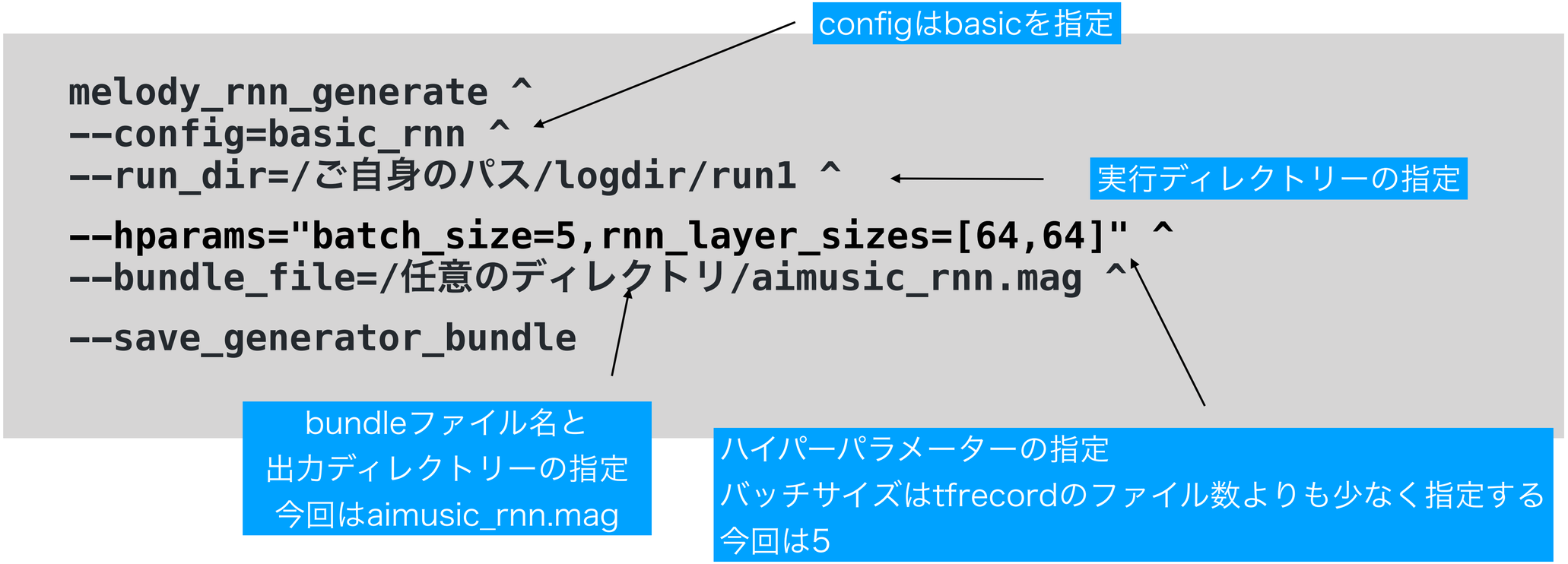
Melody RNN
８・Bundleファイル
（.magファイル）作成

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。

bundle file（.magファイル）の作成

```
melody_rnn_generate ^
--config=basic_rnn ^
--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/aimusic_rnn.mag ^
--save_generator_bundle
```



８・Bundleファイル（.magファイル）作成 Mac

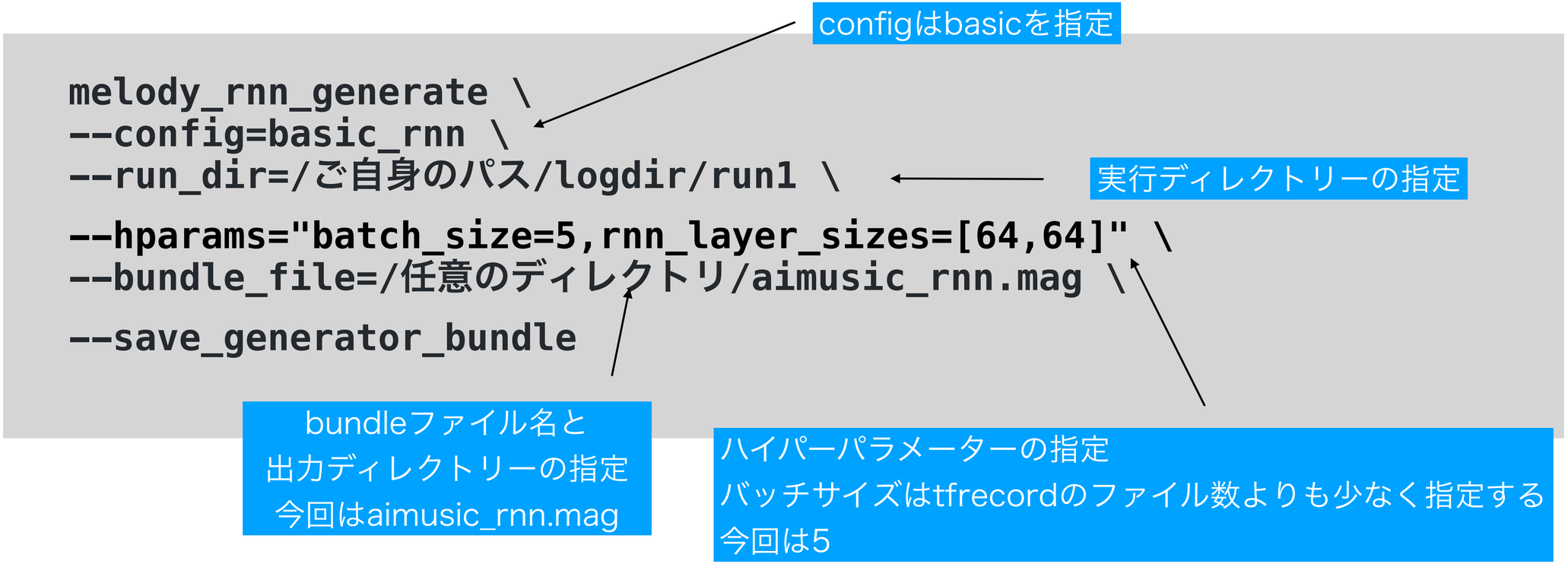
学習済みデータとしてBundleファイルを作成します。

bundle file（.magファイル）の作成

```
melody_rnn_generate \
--config=basic_rnn \
--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/aimusic_rnn.mag \

--save_generator_bundle
```



Melody RNN
作成したご自身の学習データで音楽生成

作成したご自身の学習データで音楽生成 Windows

```
melody_rnn_generate ^
--config=basic_rnn ^
--bundle_file=作成したaimusic_rnn.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]”
```

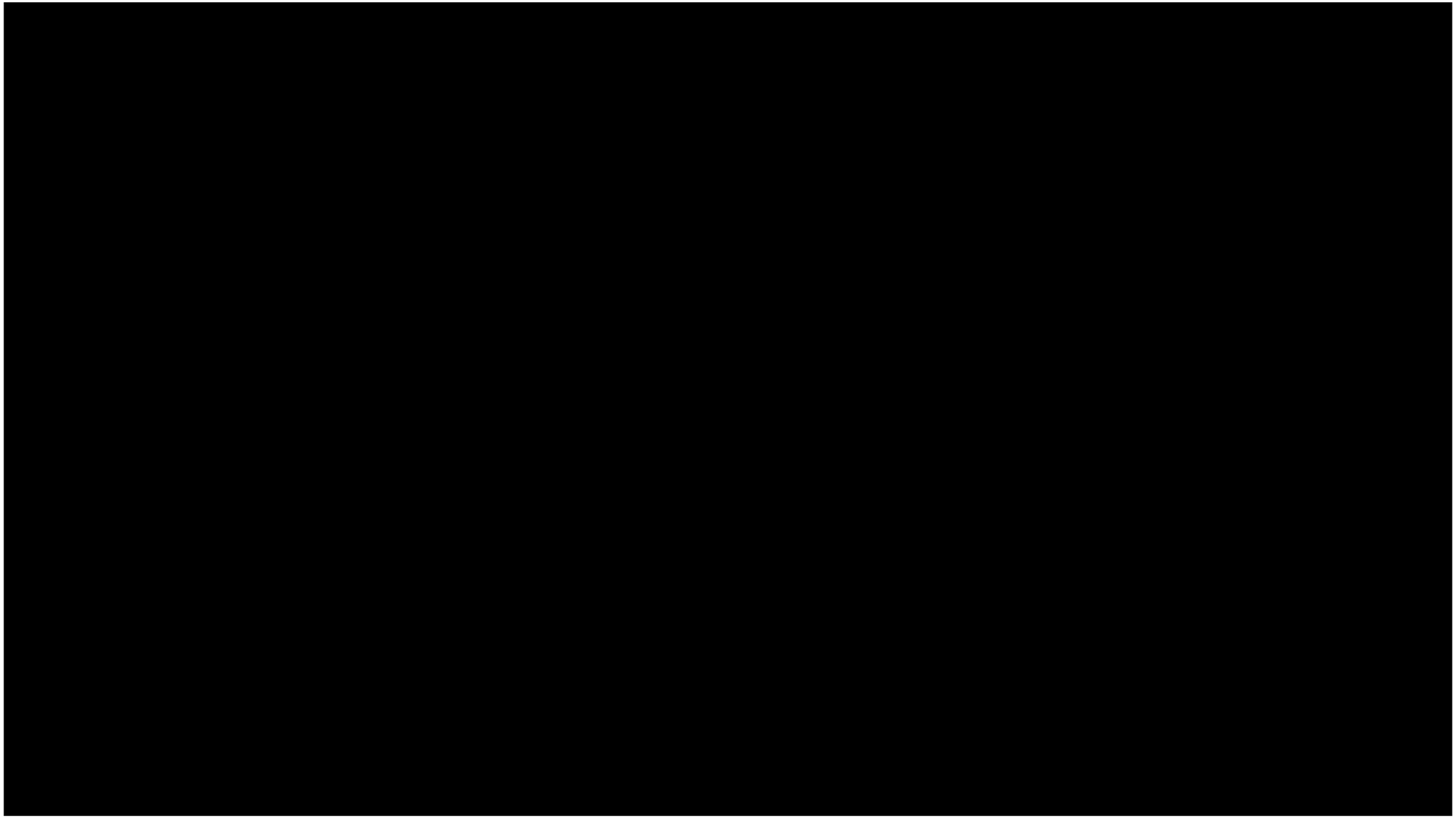
ご自身で作成したaimusic_rnn/magファイルへのパスを指定

作成したご自身の学習データで音楽生成 Mac

```
melody_rnn_generate \
--config=basic_rnn \
--bundle_file=作成したaimusic_rnn.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aimusic_rnn/magファイルへのパスを指定

Melody RNN
作成したご自身の学習データで音楽生成
５００回～２００００回まで学習し音楽生成



５００回～２００００回まで学習し音楽生成

Magenta学習データ作成

他のモデルでの学習

Performance RNN
学習データ作成

Magenta 学習データ作成の流れ

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。

Performance RNN
1・midiファイルの用意

１・midiファイルの用意

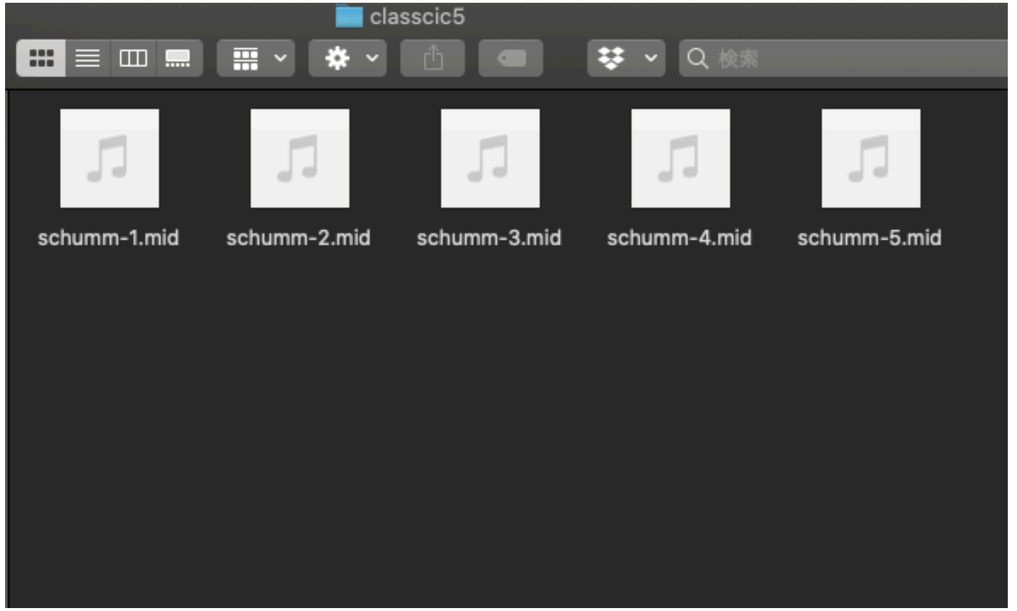
midiファイルを作成し任意のフォルダーへ。配布したmidiデータが入ったフォルダーを使用してください。

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
classic5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Magenta
各モデル共通です。



classic5
８小節のクラシックピアノ曲データ
が5曲入っています

Performance RNN
２・NoteSequence（tfrecord）の作成

２・NoteSequence（tfrecord）の作成 Windows

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。

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）

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

```
convert_dir_to_note_sequences ^
--input_dir=配布したclassic5への絶対パス ^
--output_file=任意の保存ディレクトリ/notesequences.tfrecord ^
--recursive
```

Notesequence作成の実行

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。

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）

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

```
convert_dir_to_note_sequences \
--input_dir=配布したclassic5への絶対パス \
--output_file=任意の保存ディレクトリ/notesequences.tfrecord \
--recursive
```

Notesequence作成の実行

Performance RNN
３・トレーニングデータとテストデータ の作成

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。

configを一つ指定
それぞれのモデルに合わせる

```
performance_rnn_create_dataset ^
--config=performance ^
--input=/ご自身のパス/notesequences.tfrecord ^
--output_dir=/任意のパス/sequence_examples ^
--eval_ratio=0.0
```

notesequenceの指定

保存先ディレクトリーの指定
sequence_example（以外の名称でも可能）
フォルダが作成されます

トレーニングデータとテストデータの割合

２つのtfrecordファイルが作成されます。（今回はテストデータは使用しません）
トレーニングデータとテスト（eval) データです。
eval_ratioはテストデータの割合です。
例：0.10＝トレーニングデータ９０%、評価データ１０%

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。

configを一つ指定
それぞれのモデルに合わせる

```
performance_rnn_create_dataset \
--config=performance \
--input=/ご自身のパス/notesequences.tfrecord \
--output_dir=/任意のパス/sequence_examples \
--eval_ratio=0.0
```

notesequenceの指定

保存先ディレクトリーの指定
sequence_example（以外の名称でも可能）
フォルダが作成されます

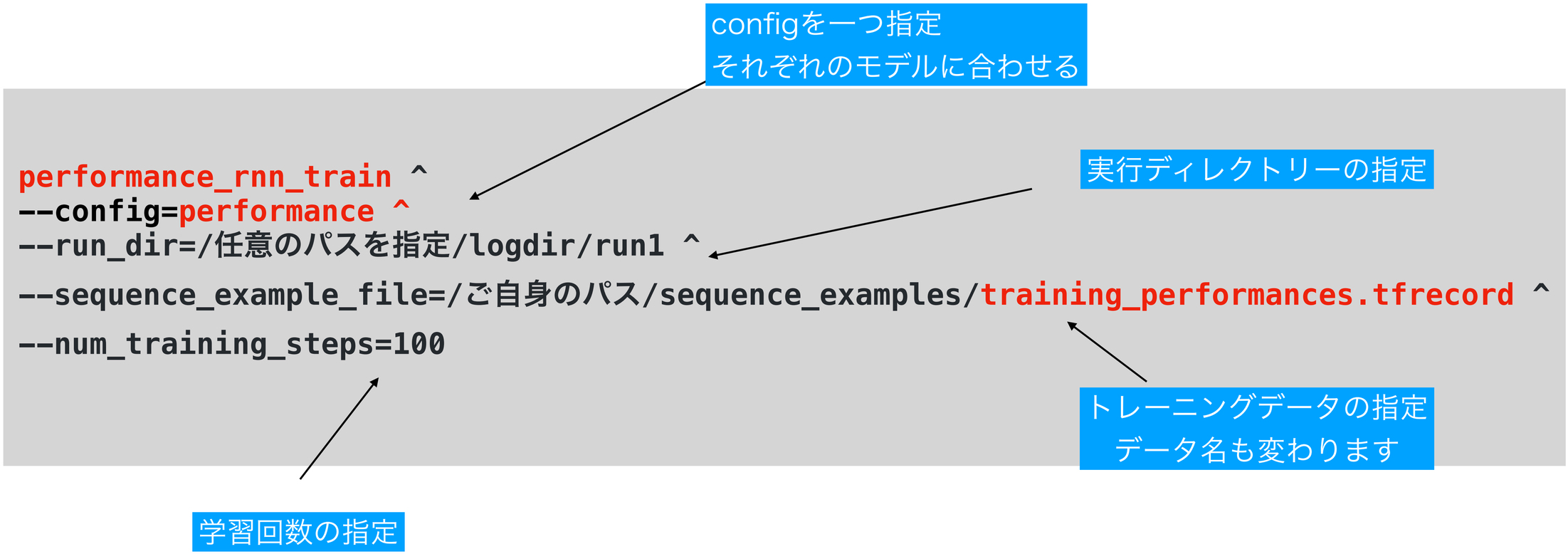
トレーニングデータとテストデータの割合

２つのtfrecordファイルが作成されます。（今回はテストデータは使用しません）
トレーニングデータとテスト（eval) データです。
eval_ratioはテストデータの割合です。
例：0.10＝トレーニングデータ９０%、評価データ１０%

Performance RNN
４・トレーニング

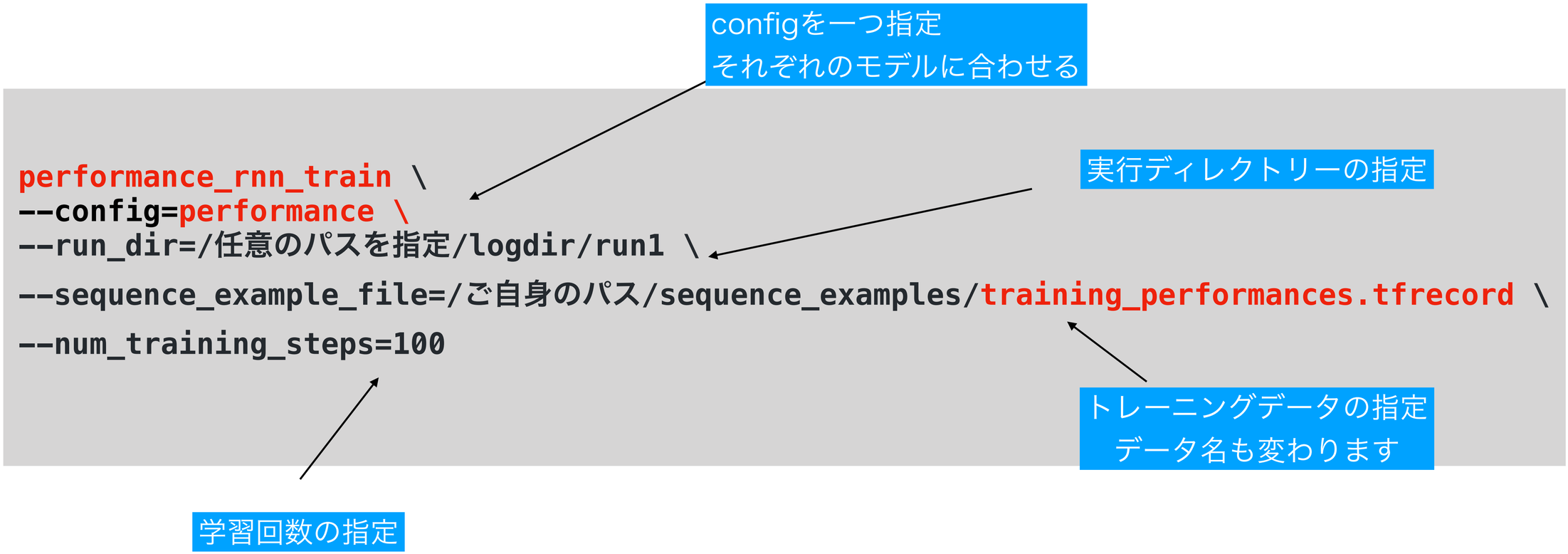
４・トレーニング　Windows

音楽データを学習させます。



４・トレーニング　Mac

音楽データを学習させます。



Performance RNN
６・tensorboardで学習の確認

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

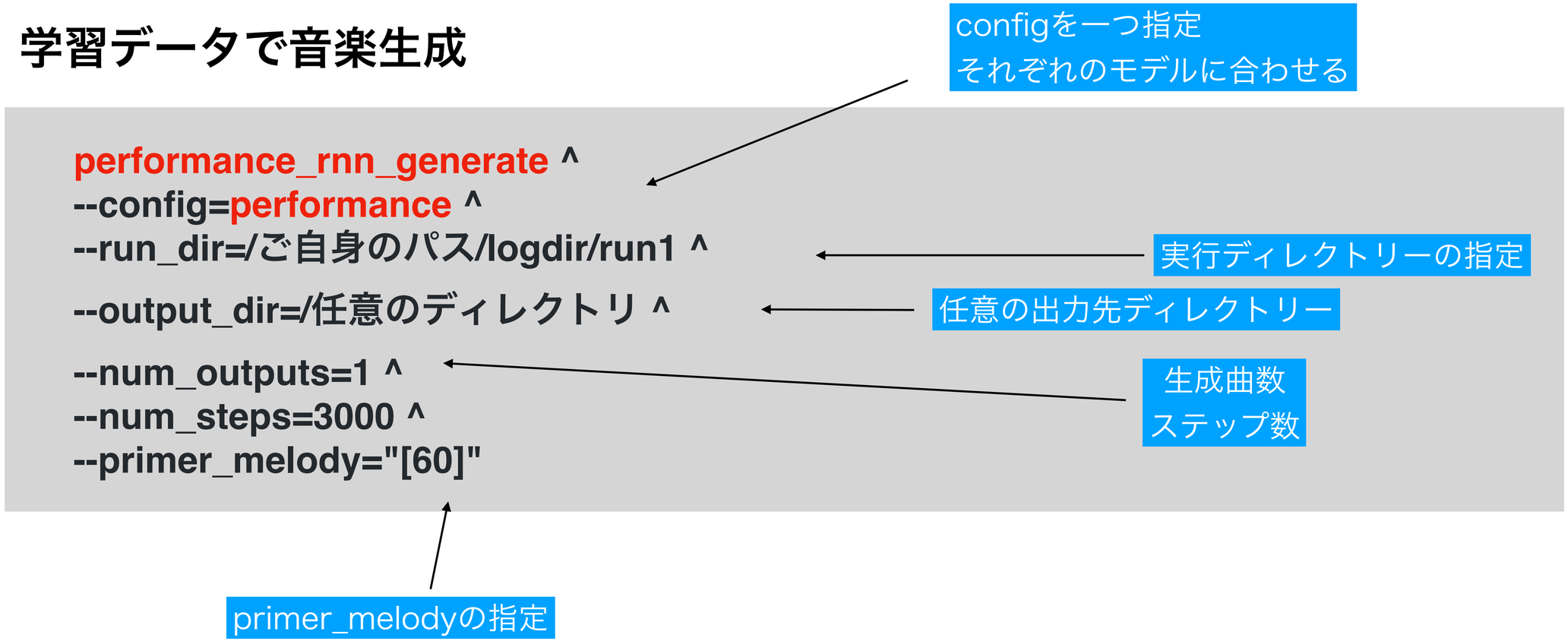
（アドレスはコピペ、またはご自身で入力して開いてください）

Performance RNN
７・音楽生成

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。

学習データで音楽生成



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。

学習データで音楽生成

```
performance_rnn_generate \
--config=performance \
--run_dir=/ご自身のパス/logdir/run1 \
--output_dir=/任意のディレクトリ \
--num_outputs=1 \
--num_steps=3000 \
--primer_melody="[60]"
```

configを一つ指定
それぞれのモデルに合わせる

実行ディレクトリーの指定

任意の出力先ディレクトリー

生成曲数
ステップ数

primer_melodyの指定

Performance RNN
８・Bundleファイル（.magファイル）作成

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。

bundle file（.magファイル）の作成

```
performance_rnn_generate ^
--config=performance ^
--run_dir=/ご自身のパス/logdir/run1 ^
--bundle_file=/任意のディレクトリ/aiclassic_rnn.mag ^
--save_generator_bundle
```

configを一つ指定
それぞれのモデルに合わせる

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
任意のファイル名を指定できます
今回はaiclassic_rnn.mag

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。

bundle file（.magファイル）の作成

```
performance_rnn_generate \
--config=performance \
--run_dir=/ご自身のパス/logdir/run1 \
--bundle_file=/任意のディレクトリ/aiclassic_rnn.mag \
--save_generator_bundle
```

configを一つ指定
それぞれのモデルに合わせる

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
任意のファイル名を指定できます
今回はaiclassic_rnn.mag