SequentialモデルAPI

はじめに，KerasのSequentialモデルのガイド を参照してください．

モデルの有用な属性

- `model.layers` は，モデルに加えたレイヤーのリストです．

Sequentialモデルのメソッド

compile

```
compile(self, optimizer, loss, metrics=None, sample_weight_mode=None, weighted_metrics=None,
```

訓練過程の設定．

引数

- **optimizer**: 文字列（optimizer名）あるいは optimizer のオブジェクト． optimizers を参照してください．
- **loss**: 文字列（目的関数名）あるいは目的関数． losses を参照してください． モデルが複数の出力を持つ場合は，オブジェクトの辞書かリストを渡すことで，各出力に異なる損失を用いることができます． モデルによって最小化される損失値は全ての個々の損失の合計になります．
- **metrics**: 訓練やテストの際にモデルを評価するための評価関数のリスト． 典型的には `metrics=['accuracy']` を使用するでしょう． 多出力モデルの各出力のための各評価関数を指定するために， `metrics={'output_a': 'accuracy'}` のような辞書を渡すこともできます．
- **sample_weight_mode**: もし時間ごとのサンプルの重み付け（2次元の重み）を行う必要があれば `"temporal"` と設定してください． `"None"` の場合，サンプルへの（1次元）重み付けをデフォルトとしています． モデルに複数の出力がある場合，モードとして辞書かリストを渡すことで，各出力に異なる `sample_weight_mode` を使うことができます．
- **weighted_metrics**: 訓練やテストの際にsample_weightまたはclass_weightにより評価と重み付けされるメトリクスのリスト．
- **target_tensors**: Kerasはデフォルトでモデルのターゲットためのプレースホルダを作成します． これは訓練中にターゲットデータが入力されるものです． 代わりの自分のターゲットテンソルを利用したい場合（訓練時にKerasはこれらのターゲットに対して外部のNumpyデータを必要としません）は，それらを `target_tensors` 引数で指定することができます． 単一出力の `Sequential` モデルの場合，これは単一のテンソルでなければなりません．

- **kwargs: Theano/CNTKがバックエンドの場合，これらはK.function に渡されます． Tensorflowバックエンドの場合は `tf.Session.run` に渡されます．

Raises

- **ValueError**: `optimizer` ， `loss` ， `metrics` ，または `sample_weight_mode` に対して無効な引数が与えられた場合．

例

```
model = Sequential()
model.add(Dense(32, input_shape=(500,)))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

---

fit

```
fit(self, x=None, y=None, batch_size=None, epochs=1, verbose=1, callbacks=None, validation_s
```

固定のエポック数でモデルを訓練する．

引数

- **x**: 訓練データのNumpy配列． モデルの入力レイヤーに名前がついていれば，入力の名前とNumpy配列をマップした辞書を渡すことも可能です． フレームワーク固有のテンソル（例えばTensorFlowデータテンソル）からフィードする場合は `x` を（デフォルトの） `None` にすることもできます．
- **y**: ターゲット（ラベル）データのNumpy配列． モデルの出力レイヤーに名前がついていれば，出力の名前とNumpy配列をマップした辞書を渡すことも可能です． フレームワーク固有のテンソル（例えばTensorFlowデータテンソル）からフィードする場合は `y` を（デフォルトの） `None` にすることもできます．
- **batch_size**: 整数または `None` ． 設定したサンプル数ごとに勾配の更新を行います． 指定しなければデフォルトで32になります．
- **epochs**: 整数で，モデルを訓練するエポック数． エポックは，提供される `x` および `y` データ全体の反復です． `initial_epoch` と組み合わせると， `epochs` は「最終エポック」として理解されることに注意してください． このモデルは `epochs` で与えられた反復回数の訓練をするわけではなく，単に `epochs` という指標に試行が達するまで訓練します．
- **verbose**: 0, 1または2． 詳細表示モード．0とすると標準出力にログを出力しません． 1の場合はログをプログレスバーで標準出力，2の場合はエポックごとに1行のログを出力します．
- **callbacks**: `keras.callbacks.Callback` にあるインスタンスのリスト． 訓練中にcallbacksのリストを適用します． callbacksを参照してください．
- **validation_split**: 0から1までの浮動小数点数． 訓練データの中で検証データとして使う割合． 訓練データの中から検証データとして設定されたデータは，訓練時に使用されず，各エポックの最後に計算される損失関数や何らかのモデルの評価関数で使われます．

- **validation_data**: 各エポックの損失関数や評価関数で用いられるタプル `(x_val, y_val)` か `(x_val, y_val, val_sample_weights)`. このデータは訓練には使われません. 設定すると `validation_split` を上書きします.
- **shuffle**: 真理値（各エポックの前に訓練データをシャッフルするか）か文字列('batch'). 'batch' は HDF5 データだけに使える特別なオプションです. バッチサイズのチャンクの中においてシャッフルします. `steps_per_epoch` が `None` に設定されている場合は効果がありません.
- **class_weight**: 辞書で，クラス毎の重みを格納します. （訓練の間だけ）損失関数をスケーリングするために使います. 過小評価されたクラスのサンプルに「より注意を向ける」ようにしたい時に便利です.
- **sample_weight**: 入力サンプルと同じ長さの1次元のNumpy 配列で，訓練のサンプルに対する重みを格納します. これは損失関数をスケーリングするために（訓練の間だけ）使用します. （重みとサンプルが1:1対応するように）入力サンプルと同じ長さのフラットな（1次元の）Numpy配列を渡すことができます. あるいは系列データの場合において，2次元配列の `(samples, sequence_length)` という形式で， すべてのサンプルの各時間において異なる重みを適用できます. この場合， `compile()` の中で `sample_weight_mode="temporal"` と確実に明記すべきです.
- **initial_epoch**: 訓練開始時のepoch（前の学習から再開する際に便利です）. steps (batches of samples) to validate before stopping.
- **steps_per_epoch**: 終了した1エポックを宣言して次のエポックを始めるまでのステップ数の合計（サンプルのバッチ）. TensorFlowのデータテンソルのような入力テンソルを使用して訓練する場合，デフォルトの `None` はデータセットのサンプル数をバッチサイズで割ったものに等しくなります. それが決定できない場合は1になります.
- **validation_steps**: `steps_per_epoch` を指定している場合のみ関係します. 停止する前にバリデーションするステップの総数（サンプルのバッチ）.

戻り値

`History` オブジェクト. `History.history` 属性は 実行に成功したエポックにおける訓練の損失値と評価関数値の記録と，（適用可能ならば）検証における損失値と評価関数値も記録しています.

Raises

- **RuntimeError**: モデルが1度もcompileされていないとき.
- **ValueError**: 与えられた入力データがモデルの期待するものとが異なる場合.

---

evaluate

```
evaluate(self, x=None, y=None, batch_size=None, verbose=1, sample_weight=None, steps=None)
```

バッチごとにある入力データにおける損失値を計算します.

引数

- **x**: 入力データ，Numpy 配列あるいは Numpy 配列のリスト（モデルに複数の入力がある場合）．（TensorFlowのデータテンソルのような）フレームワーク固有のテンソルを与える場合には `x` をデフォルトの `None` にすることもできます．
- **y**: ラベル，Numpy 配列． （TensorFlowのデータテンソルのような）フレームワーク固有のテンソルを与える場合には `y` をデフォルトの `None` にすることもできます．
- **batch_size**: 整数．指定しなければデフォルトで32になります．
- **verbose**: 進行状況メッセージ出力モードで，0か1．
- **sample_weight**: サンプルの重み，Numpy 配列．
- **steps**: 整数または `None` ．評価ラウンド終了を宣言するまでの総ステップ数（サンプルのバッチ）． デフォルト値の `None` ならば無視されます．

戻り値

スカラーで，テストデータの損失値（モデルの評価関数を設定していない場合） あるいはスカラーのリスト（モデルが他の評価関数を計算している場合）． `model.metrics_names` 属性により，スカラーの出力でラベルを表示します．

Raises

- **RuntimeError**: モデルが1度もcompileされていないとき．

---

predict

```
predict(self, x, batch_size=None, verbose=0, steps=None)
```

入力サンプルに対する予測値の出力を生成します．

入力サンプルごとにバッチごとに処理します．

引数

- **x**: 入力データで，Numpy 配列の形式．
- **batch_size**: 整数．指定しなければデフォルトで32になります．
- **verbose**: 進行状況メッセージ出力モード，0または1．
- **steps**: 評価ラウンド終了を宣言するまでの総ステップ数（サンプルのバッチ）． デフォルト値の `None` ならば無視されます．

戻り値

予測値を格納した Numpy 配列．

---

train_on_batch

```
train_on_batch(self, x, y, class_weight=None, sample_weight=None)
```

サンプル中の1つのバッチで勾配を更新します.

引数

- **x**: 入力データ，Numpy 配列または Numpy 配列のリスト（モデルに複数の入力がある場合）.
- **y**: ラベル，Numpy 配列.
- **class_weight**: 辞書で，クラス毎の重みを格納します． （訓練の間だけ）損失関数をスケーリングするために使います.
- **sample_weight**: サンプルの重み，Numpy 配列.

戻り値

スカラーでトレーニングの損失値（モデルに評価関数が設定されていない場合） あるいはスカラーのリスト（モデルが他の評価関数を計算している場合）． `model.metrics_names` 属性により，スカラーの出力でラベルを表示する.

Raises

- **RuntimeError**: モデルが1度もcompileされていないとき.

---

test_on_batch

```
test_on_batch(self, x, y, sample_weight=None)
```

サンプルの単一バッチにおけるモデルの評価を行います.

引数

- **x**: 入力データ，Numpy 配列または Numpy 配列のリスト（モデルに複数の入力がある場合）.
- **y**: ラベル，Numpy 配列の形式.
- **sample_weight**: サンプルの重み，Numpy 配列の形式.

戻り値

スカラーで，テストの損失値（モデルに評価関数が設定されていない場合） あるいはスカラーのリスト（モデルが他の評価関数を計算している場合）． `model.metrics_names` 属性により，スカラーの出力でラベルを表示する.

Raises

- **RuntimeError**: モデルが1度もcompileされていないとき.

predict_on_batch

```
predict_on_batch(self, x)
```

サンプルの単一のバッチに対する予測値を返します.

引数

- **x**: 入力データ，Numpy 配列または Numpy 配列のリスト（モデルに複数の入力がある場合）.

戻り値

予測値を格納した Numpy 配列.

---

fit_generator

```
fit_generator(self, generator, steps_per_epoch=None, epochs=1, verbose=1, callbacks=None, va
```

Python のジェネレータにより，バッチごとに生成されるデータでモデルを学習させます.

ジェネレータは効率化のために，モデルを並列に実行します． たとえば，これを使えば CPU 上でリアルタイムに画像データを拡大しながら，それと並行して GPU 上でモデルを学習できます.

引数

- **generator**: ジェネレータ． ジェネレータの出力は以下のいずれかでなければならず，どの配列も同数のサンプルを含まなければなりません.
  - タプル (inputs, targets)
  - タプル (inputs, targets, sample_weights)． 全ての配列はサンプル数と同じ個数の値が含まれている必要があります． ジェネレータは永遠にそのデータを繰り返すことを期待されています． モデルによって `steps_per_epoch` のバッチが確認されたときにエポックが終了します.
- **steps_per_epoch**: 1エポックを宣言してから次のエポックの開始前までに `generator` から生成されるサンプル (サンプルのバッチ) の総数． 典型的には，データにおけるユニークなサンプル数をバッチサイズで割った値です． `Sequence` でのオプション：指定されていない場合は，`len(generator)` をステップ数として使用します.
- **epochs**: 整数で，イテレーションの総数． `initial_epoch` と組み合わせると， `epochs` は「最終エポック」として理解されることに注意してください． このモデルは `epochs` で与えられたnステップの訓練をするわけではなく，epochが `epochs` に達するまで訓練します.
- **verbose**: 進行状況メッセージ出力モードで，0，1，あるいは 2.
- **callbacks**: callbacks のリストで，訓練の際に呼び出されます.
- **validation_data**: 以下のいずれかです.
  - 検証用データのジェネレータ
  - タプル (inputs, targets)

- タプル (inputs, targets, sample_weights).

- **validation_steps**: `validation_data` がジェネレータである場合だけ関係があります． 各エポックの終わりに検証用ジェネレータから使用するステップ数です．典型的には，検証用データにおけるユニークなサンプル数をバッチサイズで割った値です． `Sequence` におけるオプション：指定しなければ `len(validation_data)` がステップ数として用いられます．
- **class_weight**: 辞書で，クラス毎の重みを格納します．
- **max_queue_size**: ジェネレータのキューの最大サイズ．
- **workers**: スレッドベースのプロセス使用時の最大プロセス数
- **use_multiprocessing**: Trueならスレッドベースのプロセスを使います．実装がmultiprocessingに依存しているため，子プロセスに簡単に渡すことができないものとしてPickableでない引数をgeneratorに渡すべきではないことに注意してください．
- **shuffle**: 各試行の初めにバッチの順番をシャッフルするかどうか． `Sequence` (keras.utils.Sequence) インスタンスの時のみ使用されます．
- **initial_epoch**: 訓練開始時のepoch（前の学習から再開する際に便利です）．

戻り値

`History` オブジェクト．

Raises

- **RuntimeError**: モデルが1度もcompileされていないとき．

例

```
def generate_arrays_from_file(path):
    while 1:
        with open(path) as f:
            for line in f:
                # create Numpy arrays of input data
                # and labels, from each line in the file
                x, y = process_line(line)
                yield (x, y)

model.fit_generator(generate_arrays_from_file('/my_file.txt'),
                    steps_per_epoch=1000, epochs=10)
```

---

evaluate_generator

```
evaluate_generator(self, generator, steps=None, max_queue_size=10, workers=1, use_multiproce
```

ジェネレータのデータによってモデルを評価します．

ジェネレータは `test_on_batch` が受け取るデータと同じ種類のデータを返すべきです．

引数

- **generator**: (inputs, targets)あるいは(inputs, targets, sample_weights)のタプルを生成するジェネレータ.
- **steps**: `generator`が停止するまでに生成するサンプル（サンプルのバッチ）の総数. `Sequence`におけるオプション：指定しなければ`len(generator)`がステップ数として用いられます.
- **max_queue_size**: ジェネレータのキューの最大サイズ
- **workers**: スレッドベースのプロセス使用時の最大プロセス数
- **use_multiprocessing**: Trueならスレッドベースのプロセスを使います. 実装がmultiprocessingに依存しているため，子プロセスに簡単に渡すことができないものとしてPickableでない引数をgeneratorに渡すべきではないことに注意してください.

戻り値

スカラーで，テストの損失値（モデルに評価関数が設定されていない場合）あるいはスカラーのリスト（モデルが他の評価関数を計算している場合）. `model.metrics_names`属性により，スカラーの出力でラベルを表示する.

Raises

- **RuntimeError**: モデルが1度もcompileされていないとき.

---

predict_generator

```
predict_generator(self, generator, steps=None, max_queue_size=10, workers=1, use_multiproces
```

ジェネレータから生成されたデータに対して予測します.

ジェネレータは`predict_on_batch`が受け取るデータと同じ種類のデータを返すべきです.

引数

- **generator**: 入力サンプルのバッチを生成するジェネレータ.
- **steps**: `generator`が停止するまでに生成するサンプル（サンプルのバッチ）の総数. `Sequence`におけるオプション：指定しなければ`len(generator)`がステップ数として用いられます.
- **max_queue_size**: ジェネレータのキューの最大サイズ
- **workers**: スレッドベースのプロセス使用時の最大プロセス数
- **use_multiprocessing**: Trueならスレッドベースのプロセスを使います. 実装がmultiprocessingに依存しているため，子プロセスに簡単に渡すことができないものとしてPickableでない引数をgeneratorに渡すべきではないことに注意してください.
- **verbose**: 進行状況メッセージ出力モード，0または1.

戻り値

予測値の Numpy 配列.

get_layer

```
get_layer(self, name=None, index=None)
```

モデルの一部分であるレイヤーを探します．

（ユニークな）名前かグラフのインデックスに基づいてレイヤーを返します．インデックスはボトムアップの幅優先探索の順番に基づきます．

引数

- **name**: 文字列，レイヤーの名前．
- **index**: 整数，レイヤーのインデックス．

戻り値

レイヤーインスタンス．