ModelクラスAPI

functional APIでは，テンソルの入出力が与えられると，`Model`を以下のようにインスタンス化できます．

```
from keras.models import Model
from keras.layers import Input, Dense

a = Input(shape=(32,))
b = Dense(32)(a)
model = Model(inputs=a, outputs=b)
```

このモデルは，`a`を入力として`b`を計算する際に必要となるあらゆる層を含むことになります．

また，マルチ入力またはマルチ出力のモデルの場合は，リストを使うこともできます．

```
model = Model(inputs=[a1, a2], outputs=[b1, b3, b3])
```

`Model`の詳しい解説は，Keras functional APIをご覧ください．

モデルの便利な属性

- `model.layers`はモデルのグラフで構成される層を平坦化したリストです．
- `model.inputs`はテンソル入力のリストです．
- `model.outputs`はテンソル出力のリストです．

メソッド

compile

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

学習のためのモデルを設定します．

引数

- **optimizer**: 文字列（optimizer名）またはoptimizerのオブジェクト．詳細はoptimizersを参照してください．
- **loss**: 文字列（目的関数名）または目的関数．詳細は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**: 訓練時とテスト時にモデルにより評価される評価関数のリスト．一般的には `metrics=['accuracy']` を使うことになります．マルチ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．
- **loss_weights**: 異なるモデルの出力における損失寄与度に重み付けをするためのスカラ係数（Pythonの浮動小数点数）を表すオプションのリスト，または辞書．モデルによって最小化される損失値は，`loss_weights` 係数で重み付けされた個々の損失の加重合計です．リストの場合，モデルの出力と1:1対応している必要があります．テンソルの場合，出力の名前（文字列）がスカラー係数に対応している必要があります．
- **sample_weight_mode**: タイムステップ毎にサンプルを重み付け（2次元の重み）する場合は，この値を `"temporal"` に設定してください．`None` はデフォルト値で，サンプル毎の重み（1次元の重み）です．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．これは単一のテンソル（単一出力モデルの場合），テンソルのリスト，または出力名をターゲットのテンソルにマッピングした辞書になります．
- ****kwargs**: バックエンドにTheano/CNTKを用いる時は，これら引数は `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．

---

fit

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

固定回数（データセットの反復）の試行でモデルを学習させます．

引数

- **x**: モデルが単一の入力を持つ場合は訓練データのNumpy配列，もしくはモデルが複数の入力を持つ場合はNumpy配列のリスト．モデル内のあらゆる入力に名前を当てられている場合，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 `None` にすることもできます．
- **y**: モデルが単一の入力を持つ場合は教師（targets）データのNumpy配列，もしくはモデルが複数の出力を持つ場合はNumpy配列のリスト．モデル内のあらゆる出力が名前を当てられている場合，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 `None` にすることもできます．

- **batch_size**: 整数または `None` ．勾配更新毎のサンプル数を示す整数．指定しなければ `batch_size` はデフォルトで32になります．
- **epochs**: 整数．訓練データ配列の反復回数を示す整数．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だ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**: 整数．0，1，2のいずれか．進行状況の表示モード．0 = 表示なし，1 = プログレスバー，2 = 各試行毎に一行の出力．
- **callbacks**: `keras.callbacks.Callback` インスタンスのリスト．訓練時に呼ばれるコールバックのリスト．詳細はcallbacksを参照．
- **validation_split**: 0から1の間の浮動小数点数．バリデーションデータとして使われる訓練データの割合．モデルはこの割合の訓練データを区別し，それらでは学習を行わず，各試行の終わりにこのデータにおける損失とモデル評価関数を評価します．このバリデーションデータは，シャッフルを行う前に，与えられた `x` と `y` のデータの後ろからサンプリングされます．
- **validation_data**: 各試行の最後に損失とモデル評価関数を評価するために用いられる `(x_val, y_val)` のタプル，または `(val_x, val_y, val_sample_weights)` のタプル．モデルはこのデータで学習を行いません． `validation_data` は `validation_split` を上書きします．
- **shuffle**: 真理値（訓練データを各試行の前にシャッフルするかどうか）または文字列（'batch'の場合）．'batch'はHDF5データの限界を扱うための特別なオプションです．バッチサイズのチャンクでシャッフルします． `steps_per_epoch` が `None` でない場合には効果がありません．
- **class_weight**: クラスのインデックスと重み（浮動小数点数）をマップするオプションの辞書で，訓練時に各クラスのサンプルに関するモデルの損失に適用します．これは過小評価されたクラスのサンプルに「より注意を向ける」ようモデルに指示するために有用です．
- **sample_weight**: オプションのNumpy配列で訓練サンプルの重みです．（訓練時のみ）損失関数への重み付けに用いられます．（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1次元Numpy配列を渡すこともできますし，時系列データの場合には， `(samples, sequence_length)` の形式の2次元配列を渡すことができ，各サンプルの各タイムステップに異なる重みを割り当てられます．この場合， `compile()` 内で， `sample_weight_mode="temporal"` と指定するようにします．
- **initial_epoch**: 整数．訓練を開始するエポック（前回の学習を再開するのに便利です）．
- **steps_per_epoch**: 整数または `None` ．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` インスタンス．本インスタンスの `History.history` 属性は訓練時に得られた全ての情報を含みます．

Raises

- **RuntimeError**: モデルがコンパイルされていない場合.
- **ValueError**: 与えられた入力データとモデルが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)
```

テストモードにおいて，モデルの損失値と評価値を返します.

その計算はバッチ処理で行われます.

引数

- **x**: モデルが単一の入力を持つ場合は訓練データのNumpy配列，もしくはモデルが複数の入力を持つ場合はNumpy配列のリスト．モデル内のあらゆる入力に名前を当てられている場合，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 `None` にすることもできます.
- **y**: モデルが単一の入力を持つ場合は教師（targets）データのNumpy配列，もしくはモデルが複数の出力を持つ場合はNumpy配列のリスト．モデル内のあらゆる出力が名前を当てられている場合，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 `None` にすることもできます.
- **batch_size**: 整数または `None` ．勾配更新毎のサンプル数を示す整数．指定しなければ `batch_size` はデフォルトで32になります.
- **verbose**: 0または1．進行状況の表示モード．0 = 表示なし，1 = プログレスバー.
- **sample_weight**: オプションのNumpy配列で訓練サンプルの重みです．（訓練時のみ）損失関数への重み付けに用いられます．（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1次元Numpy配列を渡すこともできますし，時系列データの場合には，`(samples, sequence_length)` の形式の2次元配列を渡すことができ，各サンプルの各タイムステップに異なる重みを割り当てられます．この場合，`compile()` 内で，`sample_weight_mode="temporal"` と指定するようにします.
- **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

テストの損失を表すスカラ値（モデルが単一の出力を持ち，かつ評価関数がない場合），またはスカラ値のリスト（モデルが複数の出力や評価関数を持つ場合）． `model.metrics_names` 属性はスカラ出力の表示ラベルを提示します.

---

predict

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

入力サンプルに対する予測の出力を生成します．

その計算はバッチ処理で行われます．

引数

- **x**: Numpy配列の入力データ（もしくはモデルが複数の出力を持つ場合はNumpy配列のリスト）．
- **batch_size**: 整数値．指定しなければデフォルトで32になります．
- **verbose**: 進行状況の表示モードで，0または1．
- **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

予測結果のNumpy配列．

Raises

- **ValueError**: 与えられた入力データとモデルが期待するものが異なる場合，またはステートフルなモデルがバッチサイズの倍数でないサンプル数を受け取った場合．

---

train_on_batch

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

単一バッチデータにつき一度の勾配更新を行います．

引数

- **x**: モデルが単一の入力を持つ場合は訓練データのNumpy配列，もしくはモデルが複数の入力を持つ場合はNumpy配列のリスト．モデル内のあらゆる入力に名前を当てられている場合，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．
- **y**: モデルが単一の入力を持つ場合は教師（targets）データのNumpy配列，もしくはモデルが複数の出力を持つ場合はNumpy配列のリスト．モデル内のあらゆる出力が名前を当てられている場合，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．
- **sample_weight**: オプションのNumpy配列で訓練サンプルの重みです．（訓練時のみ）損失関数への重み付けに用いられます．（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1次元Numpy配列を渡すこともできますし，時系列データの場合には，`(samples, sequence_length)` の形式の2次元配列を渡すことができ，各サンプルの各タイムステップに異なる重みを割り当てられます．この場合，`compile()` 内で，`sample_weight_mode="temporal"` と指定するようにします．

- **class_weight**: クラスのインデックスと重み（浮動小数点数）をマップするオプションの辞書で，訓練時に各クラスのサンプルに関するモデルの損失に適用します．これは過小評価されたクラスのサンプルに「より注意を向ける」ようモデルに指示するために有用です．

戻り値

学習の損失を表すスカラ値（モデルが単一の出力を持ち，かつ評価関数がない場合），またはスカラ値のリスト（モデルが複数の出力や評価関数を持つ場合）． `model.metrics_names` 属性はスカラ出力の表示ラベルを提示します．

---

test_on_batch

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

サンプルの単一バッチでモデルをテストします．

引数

- **x**: テストデータのNumpy配列，もしくはモデルが複数の入力を持つ場合はNumpy配列のリスト．モデル内のあらゆる入力が名前を当てられている場合，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．
- **y**: 教師データのNumpy配列，もしくはモデルが複数の出力を持つ場合はNumpy配列のリスト． モデル内のあらゆる出力が名前を当てられている場合，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．
- **sample_weight**: オプションのNumpy配列で訓練サンプルの重みです． （訓練時のみ）損失関数への重み付けに用いられます． （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1次元Numpy配列を渡すこともできますし，時系列データの場合には， `(samples, sequence_length)` の形式の2次元配列を渡すことができ，各サンプルの各タイムステップに異なる重みを割り当てられます．この場合， `compile()` 内で， `sample_weight_mode="temporal"` と指定するようにします．

戻り値

テストの損失を表すスカラ値（モデルが単一の出力を持ち，かつ評価関数がない場合），またはスカラ値のリスト（モデルが複数の出力や評価関数を持つ場合）． `model.metrics_names` 属性はスカラ出力の表示ラベルを提示します．

---

predict_on_batch

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

サンプルの単一バッチに関する予測を返します．

引数

- **x**: 入力データ，Numpy配列．

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ジェネレータによりバッチ毎に生成されたデータでモデルを訓練します．

本ジェネレータは効率性のためモデルに並列して実行されます．例えば，モデルをGPUで学習させながらCPU上で画像のリアルタイムデータ拡張を行うことができるようになります．

`use_multiprocessing=True` のときに， `keras.utils.Sequence` を使うことで順序とエポックごとに全入力を1度だけ使用することを保証します．

引数

- **generator**: ジェネレータかマルチプロセッシング時にデータの重複を防ぐための `Sequence` （ `keras.utils.Sequence` ）オブジェクトのインスタンス．本ジェネレータの出力は，以下のいずれかです．
  - `(inputs, targets)` のタプル．
  - `(inputs, targets, sample_weights)` のタプル．このタプル（単一出力のジェネレータ）は単一のバッチを作ります．つまり，このタプルにある全ての配列は全て同じ長さ（バッチサイズと等しい）でなければなりません．バッチによってサイズが異なる場合もあります．例えば，データセットのサイズがバッチサイズで割り切れない場合，一般的にエポックの最後のバッチはそれ以外よりも小さくなります．このジェネレータはデータが無限にループすることを期待します． `steps_per_epoch` 数のサンプルがモデルに与えられると1度の試行が終了します．
- **steps_per_epoch**: ある一つのエポックが終了し，次のエポックが始まる前に `generator` から使用する総ステップ数（サンプルのバッチ数）．もし，データサイズをバッチサイズで割った時，通常ユニークなサンプル数に等しくなります． `Sequence` 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．モデルを訓練させるエポック数． エポックは与えられたデータ全体の反復で， `steps_per_epoch` で定義されます． `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だ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**: 整数．0，1，2のいずれか．進行状況の表示モード．0 = 表示なし，1 = プログレスバー，2 = 各試行毎に一行の出力．

- **callbacks**: `keras.callbacks.Callback` インスタンスのリスト．訓練時に呼ばれるコールバックのリスト．詳細はcallbacksを参照．
- **validation_data**: これは以下のいずれかです．
  - バリデーションデータ用のジェネレータ．
  - (inputs, targets)のタプル．
  - (inputs, targets, sample_weights)のタプル．各エポックの最後に損失関数やモデルの評価関数の評価に用いられます．このデータは学習には使われません．
- **validation_steps**: `validation_data` がジェネレータの場合にのみ関係します．終了する前に `generator` から使用する総ステップ数（サンプルのバッチ数）． `Sequence` のオプション：指定されていない場合は， `len(validation_data)` をステップ数として使用します．
- **class_weight**: クラスインデックスと各クラスの重みをマップする辞書です． （訓練のときだけ）損失関数の重み付けに使われます． 過小評価されたクラスのサンプルに「より注意を向ける」場合に有用です．
- **max_queue_size**: 整数．ジェネレータのキューのための最大サイズ． 指定しなければ `max_queue_size` はデフォルトで10になります．
- **workers**: 整数．スレッドベースのプロセス使用時の最大プロセス数．指定しなければ `workers` はデフォルトで1になります．もし0ならジェネレータはメインスレッドで実行されます．
- **use_multiprocessing**: 真理値． `True` ならスレッドベースのプロセスを使います．指定しなければ `workers` はデフォルトでFalseになります．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でない引数をジェネレータに渡すべきではないことに注意してください．
- **shuffle**: 真理値．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`)の時のみ使用されます．
- **initial_epoch**: 整数．学習を開始するエポック（前回の学習を再開するのに便利です）．

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．

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
                x1, x2, y = process_line(line)
                yield ({'input_1': x1, 'input_2': x2}, {'output': y})

model.fit_generator(generate_arrays_from_file('/my_file.txt'),
                    steps_per_epoch=10000, epochs=10)
```

Raises

- **ValueError**: ジェネレータが無効なフォーマットのデータを使用した場合．

evaluate_generator

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

データジェネレータでモデルを評価します．

ジェネレータは`test_on_batch`で受け取られたのと同じ種類のデータを返します．

引数:

- **generator**: ジェネレータは(inputs, targets)タプルもしくは(inputs, targets, sample_weights)タプルかマルチプロセッシング時にデータの重複を防ぐためのSequence (keras.utils.Sequence) オブジェクトのインスタンスを使用します．
- **steps**: 終了する前に`generator`から使用する総ステップ数（サンプルのバッチ数）．`Sequence`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．
- **max_queue_size**: ジェネレータのキューのための最大サイズ．
- **workers**: 整数．スレッドベースのプロセス使用時の最大プロセス数．指定しなければ`workers`はデフォルトで1になります．もし0ならジェネレータはメインスレッドで実行されます．
- **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でない引数をジェネレータに渡すべきではないことに注意してください．

戻り値

テストの損失を表すスカラ値（モデルが単一の出力を持ち，かつ評価関数がない場合），またはスカラ値のリスト（モデルが複数の出力や評価関数を持つ場合）．`model.metrics_names`属性はスカラ出力の表示ラベルを提示します．

Raises

- **ValueError**: ジェネレータが無効なフォーマットのデータを使用した場合．

---

predict_generator

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

データジェネレータから得た入力サンプルに対する予測を生成します．

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します．

引数

- **generator**: 入力サンプルのバッチかマルチプロセッシング時にデータの重複を防ぐためのSequence (keras.utils.Sequence) オブジェクトのインスタンスを生成するジェネレータ.
- **steps**: 終了する前に `generator` から使用する総ステップ数（サンプルのバッチ数）. `Sequence` 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.
- **max_queue_size**: ジェネレータのキューの最大サイズ.
- **workers**: 整数. スレッドベースのプロセス使用時の最大プロセス数. 指定しなければ `workers` はデフォルトで1になります. もし0ならジェネレータはメインスレッドで実行されます.
- **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でない引数をジェネレータに渡すべきではないことに注意してください.
- **verbose**: 進行状況の表示モードで，0または1.

戻り値

予測値のNumpy配列.

Raises

- **ValueError**: ジェネレータが無効なフォーマットのデータを使用した場合.

---

get_layer

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

（ユニークな）名前，またはインデックスに基づきレイヤーを探します.

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**: レイヤーの名前を表す文字列.
- **index**: レイヤーのインデックスを表す整数.

戻り値

レイヤーのインスタンス.

Raises

- **ValueError**: 無効なレイヤーの名前，またはインデックスの場合.