RNN [source]

```
keras.layers.RNN(cell, return_sequences=False, return_state=False, go_backwards=False, state
```

Recurrentレイヤーに対する基底クラス.

引数

- **cell**: RNN cellインスタンス. RNN cellは以下の属性を持つクラスです.
  - `call(input_at_t, states_at_t)` メソッド, これは `(output_at_t, states_at_t_plus_1)` を返します. cellのメソッド呼び出しはオプションの引数 `constants` も使えます. 下記の「外部定数を渡す場合の注意」を参照してください.
  - `state_size` 属性. これは1つの整数（1つの状態）でもよく, その場合はrecurrent stateのサイズになります（これはcellの出力のサイズと同じである必要があります）. （1つ状態につき1つのサイズが対応するように）整数やリストやタプルもとれます. この場合は最初のエントリ(`state_size[0]`)がcellの出力のサイズと同じである必要があります. `cell` をRNN cellインスタンスのリストとすることも可能です. この場合, cellはRNNの中で他のcellの後にスタックされているいれば, 効率的なstacked RNNが実現されます.
- **return_sequences**: 真理値. 出力系列の最後の出力を返すか, 完全な系列を返すか.
- **return_state**: 真理値. 出力とともに, 最後の状態を返すかどうか.
- **go_backwards**: 真理値（デフォルトはFalse）. Trueなら, 入力系列を逆向きから処理し, 逆順の系列を返します.
- **stateful**: 真理値（デフォルトはFalse）. Trueなら, バッチ内のインデックスiの各サンプル に対する最後の状態が次のバッチ内のインデックスiのサンプルに対する初期状態として使われます.
- **unroll**: 真理値（デフォルトはFalse）. Trueなら, ネットワークは展開され, そうでなければシンボリックループが使われます. 展開はよりメモリ集中傾向になりますが, RNNをスピードアップできます. 展開は短い系列にのみ適しています.
- **input_dim**: 入力の次元（整数）. この引数（または代わりのキーワード引数 `input_shape` ）は, このレイヤーをモデルの最初のレイヤーとして利用するときに必要です.
- **input_length**: 入力系列の長さ. この引数はこのレイヤーの後に `Flatten` から `Dense` レイヤーへ接続する際に必要です (これがないと, denseの出力のshapeを計算できません). Recurrentレイヤーがモデルの最初のレイヤーでなければ, 最初のレイヤーのレベルで入力系列の長さを指定する必要があります （例えば `input_shape` 引数を通じて）.

入力のshape

shapeが `(batch_size, timesteps, input_dim)` の3階テンソル.

出力のshape

- `return_state`の場合：テンソルのリスト．最初のテンソルが出力になります．残りのテンソルは最終状態で，それぞれのshapeは`(batch_size, units)`です．
- `return_sequences`の場合：shapeが`(batch_size, timesteps, input_dim)`の3階テンソル．
- それ以外の場合：shapeが`(batch_size, input_dim)`の2階テンソル．

マスキング

このレイヤーはタイムステップの変数を持つ入力データに対するマスキングをサポートします． あなたのデータにマスクを導入するためには， `mask_zero`パラメータに`True`を渡したEmbeddingレイヤーを利用してください．

RNNで状態管理を利用するときの注意点

RNNレイヤーを'stateful'にすることができます． これはあるバッチでサンプルに対して計算された状態が次のバッチのサンプルの初期状態として再利用されるという意味です． これは別々の連続したバッチ内のサンプルが一対一対応することを仮定します．

状態管理を可能にするためには: - レイヤーコンストラクタにおいて`stateful=True`を指定してください． - モデルに一定のバッチサイズを指定してください． もしsequentialモデルなら: `batch_input_shape=(...)`を最初のレイヤーに 1つ以上の入力層をもったfunctionalモデルなら: `batch_input_shape=(...)`をモデルのすべての最初のレイヤーに 渡すことで固定系列長のバッチサイズを指定してください． これはバッチサイズを含む入力の期待されるshapeです． これは整数のタプルであるべきです，例えば`(32, 10, 100)`． - `fit()`を呼ぶときは， `stateful=False`を指定してください．

モデルの状態を再設定するには，指定したレイヤーもしくはモデル全体で`.reset_states()`を呼び出してください．

RNNの初期状態を指定するときの注意点

`initial_state`のキーワード引数を渡してRNNを呼び出すことで，内部状態の初期値を指定できます． `initial_state`の値は，RNNの初期値を表現したテンソルかテンソルのリストです．

RNNに外部定数を渡すときの注意

（`RNN.call`のように）`RNN.__call__`メソッドの`constants`キーワード引数を使うことによって「外部」定数を渡せます． `cell.call`メソッドが`constants`と同じキーワード変数を受け入れる必要があります． そのような定数は，アテンションメカニズムで知られるような，追加の固定入力（時間変動しない）におけるcellの変化の状態として使われます．

例

```
# First, let's define a RNN Cell, as a layer subclass.

class MinimalRNNCell(keras.layers.Layer):

    def __init__(self, units, **kwargs):
        self.units = units
        self.state_size = units
        super(MinimalRNNCell, self).__init__(**kwargs)

    def build(self, input_shape):
        self.kernel = self.add_weight(shape=(input_shape[-1], self.units),
                                      initializer='uniform',
                                      name='kernel')
        self.recurrent_kernel = self.add_weight(
            shape=(self.units, self.units),
            initializer='uniform',
            name='recurrent_kernel')
        self.built = True

    def call(self, inputs, states):
        prev_output = states[0]
        h = K.dot(inputs, self.kernel)
        output = h + K.dot(prev_output, self.recurrent_kernel)
        return output, [output]

# Let's use this cell in a RNN layer:

cell = MinimalRNNCell(32)
x = keras.Input((None, 5))
layer = RNN(cell)
y = layer(x)

# Here's how to use the cell to build a stacked RNN:

cells = [MinimalRNNCell(32), MinimalRNNCell(64)]
x = keras.Input((None, 5))
layer = RNN(cells)
y = layer(x)
```

SimpleRNN

[source]

```
keras.layers.SimpleRNN(units, activation='tanh', use_bias=True, kernel_initializer='glorot_u
```

出力が入力にフィードバックされる全結合RNN.

引数

- **units**: 正の整数値，出力の次元数.
- **activation**: 活性化関数（activationsを参照）． デフォルト：ハイパボリックタンジェント（`tanh`）． `None`を渡すと活性化関数は適用されません (例．"linear" activation: `a(x) = x`).
- **use_bias**: 真理値，biasベクトルを使うかどうか.
- **kernel_initializer**: 入力の線形変換に使われる`kernel`の重み行列のためのInitializer（initializersを参照）.
- **recurrent_initializer**: 再帰の線形変換に使われる`recurrent_kernel`の重み行列のInitializer（initializersを参照）.
- **bias_initializer**: biasベクトルのInitializer（initializersを参照）.
- **kernel_regularizer**: `kernel`の重み行列に適用するRegularizer関数（regularizerを参照）.

- **recurrent_regularizer**: `recurrent_kernel` の重み行列に適用するRegularizer関数（regularizerを参照）.
- **bias_regularizer**: biasベクトルに適用するRegularizer関数（regularizerを参照）.
- **activity_regularizer**: 出力 (そのactivation) に適用するRegularizer関数（regularizerを参照）.
- **kernel_constraint**: `kernel` の重み行列に適用するConstraint関数（constraintsを参照）.
- **recurrent_constraint**: `recurrent_kernel` の重み行列に適用するConstraint関数（constraintsを参照）.
- **bias_constraint**: biasベクトルに適用するConstraint関数（constraintsを参照）.
- **dropout**: 0から1の間の浮動小数点数．入力の線形変換においてdropするユニットの割合.
- **recurrent_dropout**: 0から1の間の浮動小数点数．再帰の線形変換においてdropするユニットの割合.
- **return_sequences**: 真理値．出力系列の最後の出力を返すか，完全な系列を返すか.
- **return_state**: 真理値．出力とともに，最後の状態を返すかどうか.
- **go_backwards**: 真理値（デフォルトはFalse）．Trueなら，入力系列の後ろから処理し，逆順の系列を返します.
- **stateful**: 真理値（デフォルトはFalse）．Trueなら，バッチ内のインデックスiの各サンプルに対する最後の状態が次のバッチ内のインデックスiのサンプルに対する初期状態として使われます.
- **unroll**: 真理値（デフォルトはFalse）．Trueなら，ネットワークは展開され，そうでなければシンボリックループが使われます．展開はよりメモリ集中傾向になりますが，RNNをスピードアップできます．展開は短い系列にのみ適しています.

GRU

[source]

```
keras.layers.GRU(units, activation='tanh', recurrent_activation='hard_sigmoid', use_bias=Tru
```

ゲートのあるリカレントユニット - Cho et al. 2014.

2つの異なる変種があります．デフォルトは1406.1078v3を基にしたもので，行列の乗算の前に隠れ状態にリセットゲートを適用します．もう1つはオリジナルである1406.1078v1をベースにしているもので，処理の順番が逆です.

2つ目の変種は（GPU限定の）CuDNNGRUに互換があり，CPUでの推論も可能です．結果として `kernel` と `recurrent_kernel` に対して異なるバイアスがあります． `'reset_after'=True` と `recurrent_activation='sigmoid'` を使用してください.

引数

- **units**: 正の整数値，出力の次元数.
- **activation**: 活性化関数（activationsを参照）． デフォルト：ハイパボリックタンジェント（ `tanh` ）． `None` を渡すと活性化関数は適用されません (例．"linear" activation: `a(x) = x`).
- **recurrent_activation**: 再帰計算時に使う活性化関数（activationsを参照）.
- **use_bias**: 真理値，biasベクトルを使うかどうか.

- **kernel_initializer**: 入力の線形変換に使われる `kernel` の重み行列のためのInitializer（initializersを参照）.
- **recurrent_initializer**: 再帰の線形変換に使われる `recurrent_kernel` の重み行列のInitializer（initializersを参照）.
- **bias_initializer**: biasベクトルのInitializer（initializersを参照）.
- **kernel_regularizer**: `kernel` の重み行列に適用するRegularizer関数（regularizerを参照）.
- **recurrent_regularizer**: `recurrent_kernel` の重み行列に適用するRegularizer関数（regularizerを参照）.
- **bias_regularizer**: biasベクトルに適用するRegularizer関数（regularizerを参照）.
- **activity_regularizer**: 出力 (そのactivation) に適用するRegularizer関数（regularizerを参照）.
- **kernel_constraint**: `kernel` の重み行列に適用するConstraint関数（constraintsを参照）.
- **recurrent_constraint**: `recurrent_kernel` の重み行列に適用するConstraint関数（constraintsを参照）.
- **bias_constraint**: biasベクトルに適用するConstraint関数（constraintsを参照）.
- **dropout**: 0から1の間の浮動小数点数．入力の線形変換においてdropするユニットの割合.
- **recurrent_dropout**: 0から1の間の浮動小数点数．再帰の線形変換においてdropするユニットの割合.
- **implementation**: 実装モードで，1か2．モード1は小さなドット積や加算処理を多数行う構造である一方，モード2は少数の大きな操作をバッチ処理します． これらのモードはハードウェアやアプリケーションによって異なるパフォーマンスプロファイルとなるでしょう.
- **return_sequences**: 真理値．出力系列の最後の出力を返すか，完全な系列を返すか.
- **return_state**: 真理値．出力とともに，最後の状態を返すかどうか.
- **go_backwards**: 真理値（デフォルトはFalse）． Trueなら，入力系列の後ろから処理し，逆順の系列を返します.
- **stateful**: 真理値（デフォルトはFalse）． Trueなら，バッチ内のインデックスiの各サンプルに対する最後の状態が次のバッチ内のインデックスiのサンプルに対する初期状態として使われます.
- **unroll**: 真理値（デフォルトはFalse）． Trueなら，ネットワークは展開され， そうでなければシンボリックループが使われます． 展開はよりメモリ集中傾向になりますが，RNNをスピードアップできます． 展開は短い系列にのみ適しています.
- **reset_after**: GRUの慣習（行列の乗算の前後のどちらでリセットゲートの適用を行うか）． False = "before" (デフォルト), True = "after" (CuDNN互換).

参考文献

- On the Properties of Neural Machine Translation: Encoder-Decoder Approaches
- Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling
- A Theoretically Grounded Application of Dropout in Recurrent Neural Networks

---

LSTM

[source]

```
keras.layers.LSTM(units, activation='tanh', recurrent_activation='hard_sigmoid', use_bias=Tr
```

長短期記憶ユニット - Hochreiter 1997.

引数

- **units**: 正の整数値，出力の次元数.
- **activation**: 活性化関数（activationsを参照）． デフォルト：ハイパボリックタンジェント（`tanh`）． `None`を渡すと活性化関数は適用されません (例．"linear" activation: `a(x) = x`).
- **recurrent_activation**: 再帰計算時に使う活性化関数（activationsを参照）． デフォルト：ハードシグモイド（`hard_sigmoid`）． `None`を渡すと活性化関数は適用されません (例．"linear" activation: `a(x) = x`).
- **use_bias**: 真理値，biasベクトルを使うかどうか.
- **kernel_initializer**: 入力の線形変換に使われる`kernel`の重み行列のためのInitializer（initializersを参照）.
- **recurrent_initializer**: 再帰の線形変換に使われる`recurrent_kernel`の重み行列のInitializer（initializersを参照）.
- **bias_initializer**: biasベクトルのInitializer（initializersを参照）.
- **unit_forget_bias**: 真理値．Trueなら，初期化時に忘却ゲートのbiasに1加算．また，trueの場合は強制的に`bias_initializer="zeros"`になります． これはJozefowicz et al.で推奨されています.
- **kernel_regularizer**: `kernel`の重み行列に適用するRegularizer関数（regularizerを参照）.
- **recurrent_regularizer**: `recurrent_kernel`の重み行列に適用するRegularizer関数（regularizerを参照）.
- **bias_regularizer**: biasベクトルに適用するRegularizer関数（regularizerを参照）.
- **activity_regularizer**: 出力（そのactivation）に適用するRegularizer関数（regularizerを参照）.
- **kernel_constraint**: `kernel`の重み行列に適用するConstraint関数（constraintsを参照）.
- **recurrent_constraint**: `recurrent_kernel`の重み行列に適用するConstraint関数（constraintsを参照）.
- **bias_constraint**: biasベクトルに適用するConstraint関数（constraintsを参照）.
- **dropout**: 0から1の間の浮動小数点数．入力の線形変換においてdropするユニットの割合.
- **recurrent_dropout**: 0から1の間の浮動小数点数．再帰の線形変換においてdropするユニットの割合.
- **implementation**: 実装モードで，1か2．モード1は小さなドット積や加算処理を多数行う構造である一方，モード2は少数の大きな操作をバッチ処理します． これらのモードはハードウェアやアプリケーションによって異なるパフォーマンスプロファイルとなるでしょう.
- **return_sequences**: 真理値．出力系列の最後の出力を返すか，完全な系列を返すか.
- **return_state**: 真理値．出力とともに，最後の状態を返すかどうか.
- **go_backwards**: 真理値（デフォルトはFalse）． Trueなら，入力系列の後ろから処理し，逆順の系列を返します.
- **stateful**: 真理値（デフォルトはFalse）． Trueなら，バッチ内のインデックスiの各サンプル に対する最後の状態が次のバッチ内のインデックスiのサンプルに対する初期状態として使われます.
- **unroll**: 真理値（デフォルトはFalse）． Trueなら，ネットワークは展開され， そうでなければシンボリックループが使われます． 展開はよりメモリ集中傾向になりますが，RNNをスピードアップできます． 展開は短い系列にのみ適しています.

参考文献

- Long short-term memory (original 1997 paper)

- Learning to forget: Continual prediction with LSTM
- Supervised sequence labeling with recurrent neural networks
- A Theoretically Grounded Application of Dropout in Recurrent Neural Networks

---

ConvLSTM2D [source]

畳み込みLSTM.

LSTMレイヤーに似ていますが，入力の変換とリカレントな変換が畳み込みです.

引数

- **filters**: 整数，出力空間の次元（つまり畳み込みにおける出力フィルタの数）.
- **kernel_size**: 整数かn個の整数からなるタプル/リストで，n次元の畳み込みウィンドウを指定します.
- **strides**: 整数かn個の整数からなるタプル/リストで，畳み込みのストライドをそれぞれ指定できます．strides value != 1とすると `dilation_rate` value != 1と指定できません.
- **padding**: `"valid"` か `"same"` のどちらかを指定します.
- **data_format**: 文字列，`channels_last`（デフォルト）か `channels_first` のどちらかを指定します． これは入力における次元の順序です． `"channels_last"` の場合，入力のshapeは `(batch, time, ..., channels)` となり，`"channels_first"` の場合は `(batch, time, channels, ...)` となります． デフォルトはKerasの設定ファイル `~/.keras/keras.json` の `image_data_format` の値です． 一度も値を変更していなければ，"channels_last"になります.
- **dilation_rate**: 整数かn個の整数からなるタプル/リストで，dilated convolutionで使われる膨張率を指定します． 現在，`dilation_rate` value != 1 とすると，`strides` value != 1を指定することはできません.
- **activation**: 使用する活性化関数の名前（activationsを参照）， 何も指定しなければ，活性化は一切適用されません（つまり"線形"活性 `a(x) = x`）.
- **recurrent_activation**: recurrentステップで適用される活性化関数（activationsを参照）.
- **use_bias**: 真理値，レイヤーがバイアスベクトルを使うかどうか.
- **kernel_initializer**: `kernel` の重み行列の初期値を指定します．入力の線形変換に使われます．（initializersを参照）.
- **recurrent_initializer**: `recurrent_kernel` の重み行列の初期値を指定します． recurrent stateの線形変換に使われます．（initializersを参照）.
- **bias_initializer**: バイアスベクトルの初期値を指定します．（initializersを参照）
- **unit_forget_bias**: 真理値．Trueなら，初期化時に忘却ゲートのバイアスに1を加えます．`bias_initializer="zeros"` とともに用いられます． これはJozefowicz et al.により推奨されています.
- **kernel_regularizer**: `kernel` の重み行列に適用させるRegularizerを指定します．（regularizerを参照）
- **recurrent_regularizer**: `recurrent_kernel` の重み行列に適用させるRegularizerを指定します．（regularizerを参照）.
- **bias_regularizer**: バイアスベクトルに適用させるRegularizerを指定します．（regularizerを参照）.

- **activity_regularizer**: 出力テンソルに適用させるRegularizerを指定します．（regularizerを参照）．
- **kernel_constraint**: `kernel` の重み行列に適用させるConstraintを指定します．（constraintを参照）．
- **recurrent_constraint**: recurrent_kernelの重み行列に適用させるConstraintを指定します．（constraintを参照）．
- **bias_constraint**: バイアスベクトルに適用させるConstraintを指定します．（constraintを参照）．
- **return_sequences**: 真理値．出力系列の最後の出力を返すか，完全な系列を返すか．
- **go_backwards**: 真理値（デフォルトはFalse）．Trueなら，入力系列の後ろから処理し，逆順の系列を返します．
- **stateful**: 真理値（デフォルトはFalse）．Trueなら，バッチ内のインデックスiの各サンプルに対する最後の状態が次のバッチ内のインデックスiのサンプルに対する初期状態として使われます．
- **dropout**: 0から1の間の浮動小数点数．入力の線形変換においてdropするユニットの割合．
- **recurrent_dropout**: 0から1の間の浮動小数点数．再帰の線形変換においてdropするユニットの割合．

入力のshape

- data_format='channels_first'の場合は次のshapeの5階テンソル：`(samples,time, channels, rows, cols)`
- data_format='channels_last'の場合は次のshapeの5階テンソル：`(samples,time, rows, cols, channels)`

出力のshape

- `return_sequences` の場合
  - data_format='channels_first'なら次のshapeの5階テンソル：
    `(samples, time, filters, output_row, output_col)`
  - data_format='channels_last'なら次のshapeの5階テンソル：
    `(samples, time, output_row, output_col, filters)`
- それ以外の場合
  - data_format='channels_first'なら次のshapeの4階テンソル：
    `(samples, filters, output_row, output_col)`
  - data_format='channels_last'なら次のshapeの4階テンソル：
    `(samples, output_row, output_col, filters)` o_rowsとo_colsはフィルタのshapeやパディングに依存します．

Raise

- **ValueError**: 無効なコンストラクタ引数の場合

参考文献

- Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting 現状の実装ではcell出力におけるフィードバックループを含んでいません

SimpleRNNCell [source]

```
keras.layers.SimpleRNNCell(units, activation='tanh', use_bias=True, kernel_initializer='glor
```

SimpleRNNのCellクラス.

引数

- **units**: 正の整数値，出力の次元数.
- **activation**: 活性化関数（activationsを参照）． デフォルト：ハイパボリックタンジェント（`tanh`）． `None`を渡すと活性化関数は適用されません（つまり"線形"活性: `a(x) = x`）.
- **use_bias**: 真理値，レイヤーがバイアスベクトルを使うかどうか.
- **kernel_initializer**: 入力の線形変換に使われる`kernel`の重み行列のためのInitializer（initializersを参照）.
- **recurrent_initializer**: 再帰の線形変換に使われる`recurrent_kernel`の重み行列のInitializer（initializersを参照）.
- **bias_initializer**: バイアスベクトルのInitializer（initializersを参照）.
- **kernel_regularizer**: `kernel`の重み行列に適用するRegularizer関数（regularizerを参照）.
- **recurrent_regularizer**: `recurrent_kernel`の重み行列に適用するRegularizer関数（regularizerを参照）.
- **bias_regularizer**: biasベクトルに適用するRegularizer関数（regularizerを参照）.
- **kernel_constraint**: `kernel`の重み行列に適用するConstraint関数（constraintsを参照）.
- **recurrent_constraint**: `recurrent_kernel`の重み行列に適用するConstraint関数（constraintsを参照）.
- **bias_constraint**: biasベクトルに適用するConstraint関数（constraintsを参照）.
- **dropout**: 0から1の間の浮動小数点数．入力の線形変換においてdropするユニットの割合.
- **recurrent_dropout**: 0から1の間の浮動小数点数．再帰の線形変換においてdropするユニットの割合.

GRUCell [source]

```
keras.layers.GRUCell(units, activation='tanh', recurrent_activation='hard_sigmoid', use_bias
```

GRUレイヤーのためのCellクラス.

- **units**: 正の整数値，出力の次元数.
- **activation**: 活性化関数（activationsを参照）． デフォルト：ハイパボリックタンジェント（`tanh`）． `None`を渡すと活性化関数は適用されません（つまり"線形"活性: `a(x) = x`）.
- **recurrent_activation**: 再帰計算時に使う活性化関数（activationsを参照）.
- **use_bias**: 真理値，レイヤーがバイアスベクトルを使うかどうか.
- **kernel_initializer**: 入力の線形変換に使われる`kernel`の重み行列のためのInitializer（initializersを参照）.

- **recurrent_initializer**: 再帰の線形変換に使われる `recurrent_kernel` の重み行列のInitializer（initializersを参照）.
- **bias_initializer**: biasベクトルのInitializer（initializersを参照）.
- **kernel_regularizer**: `kernel` の重み行列に適用するRegularizer関数（regularizerを参照）.
- **recurrent_regularizer**: `recurrent_kernel` の重み行列に適用するRegularizer関数（regularizerを参照）.
- **bias_regularizer**: biasベクトルに適用するRegularizer関数（regularizerを参照）.
- **kernel_constraint**: `kernel` の重み行列に適用するConstraint関数（constraintsを参照）.
- **recurrent_constraint**: `recurrent_kernel` の重み行列に適用するConstraint関数（constraintsを参照）.
- **bias_constraint**: biasベクトルに適用するConstraint関数（constraintsを参照）.
- **dropout**: 0から1の間の浮動小数点数. 入力の線形変換においてdropするユニットの割合.
- **recurrent_dropout**: 0から1の間の浮動小数点数. 再帰の線形変換においてdropするユニットの割合.
- **implementation**: 実装モードで，1か2. モード1は小さなドット積や加算処理を多数行う構造である一方，モード2は少数の大きな操作をバッチ処理します. これらのモードはハードウェアやアプリケーションによって異なるパフォーマンスプロファイルとなるでしょう.
- **reset_after**: GRUの慣習（行列の乗算の前後のどちらでリセットゲートの適用を行うか）. False = "before" (デフォルト), True = "after" (CuDNN互換).

LSTMCell [source]

```
keras.layers.LSTMCell(units, activation='tanh', recurrent_activation='hard_sigmoid', use_bia
```

LSTMレイヤーのためのcellクラス.

引数

- **units**: 正の整数値，出力の次元数.
- **activation**: 活性化関数（activationsを参照）. デフォルト：ハイパボリックタンジェント（`tanh`）. `None` を渡すと活性化関数は適用されません（つまり"線形"活性: `a(x) = x`）.
- **activation**: 活性化関数（activationsを参照）. デフォルト：ハイパボリックタンジェント（`tanh`）. `None` を渡すと活性化関数は適用されません（つまり"線形"活性: `a(x) = x`）.
- **use_bias**: 真理値，biasベクトルを使うかどうか.
- **kernel_initializer**: 入力の線形変換に使われる `kernel` の重み行列のためのInitializer（initializersを参照）.
- **recurrent_initializer**: 再帰の線形変換に使われる `recurrent_kernel` の重み行列のInitializer（initializersを参照）.
- **bias_initializer**: biasベクトルのInitializer（initializersを参照）.
- **unit_forget_bias**: 真理値. Trueなら，初期化時に忘却ゲートのバイアスに1を加えます. `bias_initializer="zeros"` とともに用いられます. これはJozefowicz et al.により推奨されています.
- **kernel_regularizer**: `kernel` の重み行列に適用するRegularizer関数（regularizerを参照）.

- **recurrent_regularizer**: `recurrent_kernel`の重み行列に適用するRegularizer関数（regularizerを参照）.
- **bias_regularizer**: biasベクトルに適用するRegularizer関数（regularizerを参照）.
- **kernel_constraint**: `kernel`の重み行列に適用するConstraint関数（constraintsを参照）.
- **recurrent_constraint**: `recurrent_kernel`の重み行列に適用するConstraint関数（constraintsを参照）.
- **bias_constraint**: biasベクトルに適用するConstraint関数（constraintsを参照）.
- **dropout**: 0から1の間の浮動小数点数．入力の線形変換においてdropするユニットの割合.
- **recurrent_dropout**: 0から1の間の浮動小数点数．再帰の線形変換においてdropするユニットの割合.
- **implementation**: 実装モードで，1か2．モード1は小さなドット積や加算処理を多数行う構造である一方，モード2は少数の大きな操作をバッチ処理します． これらのモードはハードウェアやアプリケーションによって異なるパフォーマンスプロファイルとなるでしょう.

StackedRNNCells

[source]

```
keras.layers.StackedRNNCells(cells)
```

RNN cellのスタックの振る舞いを単一のcellのようにするためのラッパー.

効率的なstacked RNNを実装するために使われます.

引数

- **cells**: RNN cellインスタンスのリスト.

例

```
cells = [
    keras.layers.LSTMCell(output_dim),
    keras.layers.LSTMCell(output_dim),
    keras.layers.LSTMCell(output_dim),
]

inputs = keras.Input((timesteps, input_dim))
x = keras.layers.RNN(cells)(inputs)
```

CuDNNGRU

[source]

```
keras.layers.CuDNNGRU(units, kernel_initializer='glorot_uniform', recurrent_initializer='ort
```

CuDNNを利用した高速なGRU実装.

TensorFlowバックエンドでGPU上でのみ動作します.

引数

- **units**: 正の整数値，出力の次元数.
- **kernel_initializer**: 入力の線形変換に使われる `kernel` の重み行列のためのInitializer（initializersを参照）.
- **recurrent_initializer**: 再帰の線形変換に使われる `recurrent_kernel` の重み行列のInitializer（initializersを参照）.
- **bias_initializer**: biasベクトルのInitializer（initializersを参照）.
- **kernel_regularizer**: `kernel` の重み行列に適用するRegularizer関数（regularizerを参照）.
- **recurrent_regularizer**: `recurrent_kernel` の重み行列に適用するRegularizer関数（regularizerを参照）.
- **bias_regularizer**: biasベクトルに適用するRegularizer関数（regularizerを参照）.
- **activity_regularizer**: 出力 (そのactivation) に適用するRegularizer関数（regularizerを参照）.
- **kernel_constraint**: `kernel` の重み行列に適用するConstraint関数（constraintsを参照）.
- **recurrent_constraint**: `recurrent_kernel` の重み行列に適用するConstraint関数（constraintsを参照）.
- **bias_constraint**: biasベクトルに適用するConstraint関数（constraintsを参照）.
- **return_sequences**: 真理値．出力系列の最後の出力を返すか，完全な系列を返すか.
- **return_state**: 真理値．出力とともに，最後の状態を返すかどうか.
- **stateful**: 真理値（デフォルトはFalse）．Trueなら，バッチ内のインデックスiの各サンプルに対する最後の状態が次のバッチ内のインデックスiのサンプルに対する初期状態として使われます.

CuDNNLSTM [source]

```
keras.layers.CuDNNLSTM(units, kernel_initializer='glorot_uniform', recurrent_initializer='or
```

CuDNNを利用した高速なLSTM実装.

TensorFlowバックエンドでGPU上でのみ動作します.

引数

- **units**: 正の整数値，出力の次元数.
- **kernel_initializer**: 入力の線形変換に使われる `kernel` の重み行列のためのInitializer（initializersを参照）.
- **unit_forget_bias**: 真理値．Trueなら，初期化時に忘却ゲートのバイアスに1を加えます．`bias_initializer="zeros"` とともに用いられます． これはJozefowicz et al.により推奨されています.
- **recurrent_initializer**: 再帰の線形変換に使われる `recurrent_kernel` の重み行列のInitializer（initializersを参照）.
- **bias_initializer**: biasベクトルのInitializer（initializersを参照）.
- **kernel_regularizer**: `kernel` の重み行列に適用するRegularizer関数（regularizerを参照）.

- **recurrent_regularizer**: `recurrent_kernel`の重み行列に適用するRegularizer関数（regularizerを参照）.
- **bias_regularizer**: biasベクトルに適用するRegularizer関数（regularizerを参照）.
- **activity_regularizer**: 出力 (そのactivation) に適用するRegularizer関数（regularizerを参照）.
- **kernel_constraint**: `kernel`の重み行列に適用するConstraint関数（constraintsを参照）.
- **recurrent_constraint**: `recurrent_kernel`の重み行列に適用するConstraint関数（constraintsを参照）.
- **bias_constraint**: biasベクトルに適用するConstraint関数（constraintsを参照）.
- **return_sequences**: 真理値．出力系列の最後の出力を返すか，完全な系列を返すか.
- **return_state**: 真理値．出力とともに，最後の状態を返すかどうか.
- **stateful**: 真理値（デフォルトはFalse）．Trueなら，バッチ内のインデックスiの各サンプルに対する最後の状態が次のバッチ内のインデックスiのサンプルに対する初期状態として使われます.