Keras backends

"バックエンド"とは?

Kerasはモデルレベルのライブラリで，深層学習モデルを開発するための高水準な構成要素を提供します．テンソル積，畳み込みなどのような低水準の操作をKeras自身で扱うことはありません．その代わりに，Kerasの"バックエンドエンジン"としての役割を果たし，そのような操作を行うために特化し，また最適化されたテンソルを取り扱うライブラリに依存しています．唯一のテンソルのライブラリを選び，そのライブラリに束縛されたKerasの実装を行うのではなく，Kerasはモジュール方式でこの問題を扱い，いくつかの異なるバックエンドエンジンをKerasにシームレスに接続できます．

現在は，Kerasは3つのバックエンドが利用可能で，それは**TensorFlow**バックエンドと**Theano**バックエンド，そして**CNTK**バックエンドです．

- TensorFlow はGoogle, Inc.により開発されたオープンソースで，テンソルをシンボリックに操作ができるフレームワークです．
- Theano はモントリオール大学のLISA/MILA Labにより開発されたオープンソースで，テンソルをシンボリックに操作ができるフレームワークです．
- CNTK はMicrosoftによって開発された深層学習のためのcommercial-grade toolkitというオープンソースです．

将来的に，さらにバックエンドを追加する予定です．

バックエンドの切り替え

少なくとも一度Kerasを実行したら，以下にあるKerasの設定ファイルを見つけるでしょう．

`$HOME/.keras/keras.json`

もしそこにこのファイルがなければ，あなたが作成できます．

Windows ユーザへ注意: `$HOME` を `%USERPROFILE%` に変更してください．

デフォルトの設定ファイルはおそらく以下のように見えるでしょう:

```
{
    "image_data_format": "channels_last",
    "epsilon": 1e-07,
    "floatx": "float32",
    "backend": "tensorflow"
}
```

`backend`フィールドを`"theano"`か`"tensorflow"`，`"cntk"`に変えるだけで，次回の実行時から新しい設定を利用します．

環境変数`KERAS_BACKEND`も定義することができて，かつあなたの設定ファイルで定義されているものを上書きします:

```
KERAS_BACKEND=tensorflow python -c "from keras import backend"
Using TensorFlow backend.
```

---

keras.json の詳細

`keras.json`構成ファイルは次の設定を含みます：

```
{
    "image_data_format": "channels_last",
    "epsilon": 1e-07,
    "floatx": "float32",
    "backend": "tensorflow"
}
```

`$HOME/.keras/keras.json`を編集することでこれらの設定を変更できます．

- `image_data_format`: 文字列，`"channels_last"`か`"channels_first"`のいずれか．Kerasが従うデータのフォーマット規則を指定します．（`keras.backend.image_data_format()`がこれを返します．）
- 2次元データ (例えば画像) に対しては，`"channels_last"`は`(rows, cols, channels)`とみなし，`"channels_first"`は`(channels, rows, cols)`とみなします．
- 3次元データに対しては，`"channels_last"`は`(conv_dim1, conv_dim2, conv_dim3, channels)`とみなし，`"channels_first"`は`(channels, conv_dim1, conv_dim2, conv_dim3)`とみなします．
- `epsilon`: float，いくつかの操作で0除算を避けるために使う微小量定数．
- `floatx`: 文字列，`"float16"`，`"float32"`，か`"float64"`．デフォルトの浮動小数点精度．
- `backend`: 文字列，`"tensorflow"`か`"theano"`か`"cntk"`．

新しいコードを書くための抽象的なKerasバックエンドの利用

もし，あなたがTheano（`th`）とTesorFlow（`tf`）の両方で互換性があるように記述できるKerasモジュールが欲しいときは，抽象的なKerasバックエンドAPIを通じて書く必要があります．以下は導入部になります．

あなたは以下を通じてバックエンドモジュールをインポートできます:

```
from keras import backend as K
```

以下のコードは入力のプレースホルダーのインスタンスを作成します． これは`tf.placeholder()`，`th.tensor.matrix()`，または`th.tensor.tensor3()`，などと同じです．

```
input = K.placeholder(shape=(2, 4, 5))
# 以下も動作します:
input = K.placeholder(shape=(None, 4, 5))
# 以下も動作します:
input = K.placeholder(ndim=3)
```

以下のコードは共有変数のインスタンスを作成します． これは`tf.variable()`，または`th.shared()`と同じことです．

```
import numpy as np
val = np.random.random((3, 4, 5))
var = K.variable(value=val)

# すべて0の変数:
var = K.zeros(shape=(3, 4, 5))
# すべて1の変数:
var = K.ones(shape=(3, 4, 5))
```

あなたが必要とするであろう大抵のテンソルの操作はTensorFlowやTheanoにおいて行うように実行できます:

```
# Initializing Tensors with Random Numbers
b = K.random_uniform_variable(shape=(3, 4), low=0, high=1) # Uniform distribution
c = K.random_normal_variable(shape=(3, 4), mean=0, scale=1) # Gaussian distribution
d = K.random_normal_variable(shape=(3, 4), mean=0, scale=1)

# Tensor Arithmetic
a = b + c * K.abs(d)
c = K.dot(a, K.transpose(b))
a = K.sum(b, axis=1)
a = K.softmax(b)
a = K.concatenate([b, c], axis=-1)
# etc...
```

バックエンド関数

epsilon

```
keras.backend.epsilon()
```

数値演算で使われる微小量を返します．

戻り値

浮動小数点数.

例

```
>>> keras.backend.epsilon()
1e-07
```

---

set_epsilon

```
keras.backend.set_epsilon(e)
```

数値演算で使われる微小量をセットします.

引数

- **e**: 浮動小数点数，新たな微小量（epsilon）.

例

```
>>> from keras import backend as K
>>> K.epsilon()
1e-07
>>> K.set_epsilon(1e-05)
>>> K.epsilon()
1e-05
```

---

floatx

```
keras.backend.floatx()
```

デフォルトのfloat型を文字列で返します（e.g. 'float16', 'float32', 'float64'）.

戻り値

文字列，現在のデフォルトのfloat型.

例

```
>>> keras.backend.floatx()
'float32'
```

---

set_floatx

```
keras.backend.set_floatx(floatx)
```

デフォルトのfloat型をセットします.

引数

- **floatx**: 'float16', 'float32', または'float64'の文字列.

例

```
>>> from keras import backend as K
>>> K.floatx()
'float32'
>>> K.set_floatx('float16')
>>> K.floatx()
'float16'
```

---

cast_to_floatx

```
keras.backend.cast_to_floatx(x)
```

Numpy配列をデフォルトのKerasのfloat型にキャストします.

引数

- **x**: Numpy 配列

戻り値

新しい型にキャストされた同じNumpy 配列.

例

```
>>> from keras import backend as K
>>> K.floatx()
'float32'
>>> arr = numpy.array([1.0, 2.0], dtype='float64')
>>> arr.dtype
dtype('float64')
>>> new_arr = K.cast_to_floatx(arr)
>>> new_arr
array([ 1.,  2.], dtype=float32)
>>> new_arr.dtype
dtype('float32')
```

---

image_data_format

```
keras.backend.image_data_format()
```

画像におけるデフォルトのフォーマット規則（'channels_first' か 'channels_last'）を返します.

戻り値

`'channels_first'`，または`'channels_last'`のどちらかの文字列.

例

```
>>> keras.backend.image_data_format()
'channels_first'
```

---

set_image_data_format

```
keras.backend.set_image_data_format(data_format)
```

デフォルトのフォーマット規則をセットします.

引数

- **data_format**: `'channels_first'`，または`'channels_last'`の文字列.

例

```
>>> from keras import backend as K
>>> K.image_data_format()
'channels_first'
>>> K.set_image_data_format('channels_last')
>>> K.image_data_format()
'channels_last'
```

---

get_uid

```
keras.backend.get_uid(prefix='')
```

デフォルトのグラフにおけるuidを取得します.

引数

- **prefix**: グラフにおける任意の接頭語.

戻り値

グラフにおける唯一の識別子.

---

reset_uids

```
keras.backend.reset_uids()
```

グラフの識別子をリセットします.

clear_session

```
keras.backend.clear_session()
```

現在のTFグラフを壊し，新たなものを作成します.

古いモデル/レイヤが散らかってしまうを避けるのに役立ちます.

---

manual_variable_initialization

```
keras.backend.manual_variable_initialization(value)
```

手動で変数を初期化するかのフラグがセットされます.

この真理値が変数がインスタンス化することで初期化すべきか（デフォルト），利用者側で初期化を制御すべきか（例えば，`tf.initialize_all_variables()` を通じて）を決定します.

引数

- **value**: 真理値.

---

learning_phase

```
keras.backend.learning_phase()
```

学習フェーズのフラグを返します.

学習フェーズのフラグは学習期間とテスト期間で異なる振る舞いをする任意のKeras関数への入力として渡される真理値のテンソル (0 = test, 1 = train) です.

戻り値

学習フェーズ（テンソルのスカラーにおける整数か，Pythonの整数）.

---

set_learning_phase

```
keras.backend.set_learning_phase(value)
```

値を固定化するための学習フェーズをセットします.

引数

- **value**: 学習フェーズの値．0，または1の整数．

Raises

- **ValueError**: もし `value` が `0`，または `1` ではなかった場合．

---

is_sparse

```
keras.backend.is_sparse(tensor)
```

テンソルがスパースかどうかを返します．

引数

- **tensor**: テンソルのインスタンス．

戻り値

真理値．

例

```
>>> from keras import backend as K
>>> a = K.placeholder((2, 2), sparse=False)
>>> print(K.is_sparse(a))
False
>>> b = K.placeholder((2, 2), sparse=True)
>>> print(K.is_sparse(b))
True
```

---

to_dense

```
keras.backend.to_dense(tensor)
```

スパースなテンソルを密なテンソルに変換し，それを返します．

引数

- **tensor**: テンソルのインスタンス（潜在的にスパースであること）．

戻り値

密なテンソル．

例

```
>>> from keras import backend as K
>>> b = K.placeholder((2, 2), sparse=True)
>>> print(K.is_sparse(b))
True
>>> c = K.to_dense(b)
>>> print(K.is_sparse(c))
False
```

---

variable

```
keras.backend.variable(value, dtype=None, name=None, constraint=None)
```

テンソルのインスタンス化し，それを返します.

引数

- **value**: テンソルの初期値が含まれたNumpy 配列.
- **dtype**: テンソルの型.
- **name**: テンソルに対する任意の名前を表す文字列.
- **constraint**: オプティマイザの更新後に変数に適用するオプションの射影関数。

戻り値

変数のインスタンス（Kerasのメタ情報が含まれています）.

例

```
>>> from keras import backend as K
>>> val = np.array([[1, 2], [3, 4]])
>>> kvar = K.variable(value=val, dtype='float64', name='example_var')
>>> K.dtype(kvar)
'float64'
>>> print(kvar)
example_var
>>> K.eval(kvar)
array([[ 1.,  2.],
       [ 3.,  4.]])
```

---

constant

```
keras.backend.constant(value, dtype=None, shape=None, name=None)
```

引数

- **value**: 定数，またはリスト.
- **dtype**: 返されたテンソルに対する要素の型.
- **shape**: 返されたテンソルに対する任意の次元.

- **name**: テンソルの任意の名前.

戻り値

不変のテンソル.

---

is_keras_tensor

```
keras.backend.is_keras_tensor(x)
```

`x`がKerasのテンソルかどうかを返します.

「Kerasのテンソル」とはKerasのレイヤー（`Layer`クラス）や`Input`から返されたテンソルです。

引数

- **x**: 潜在的なテンソル.

戻り値

真理値: 引数がKerasのテンソルかどうか.

Raises

- **ValueError**: `x`がシンボリックなテンソルでない場合。

例

```
>>> from keras import backend as K
>>> from keras.layers import Input, Dense
>>> np_var = numpy.array([1, 2])
>>> K.is_keras_tensor(np_var) # A numpy array is not a symbolic tensor.
ValueError
>>> k_var = tf.placeholder('float32', shape=(1,1))
>>> K.is_keras_tensor(k_var) # A variable indirectly created outside of keras is not a Keras
False
>>> keras_var = K.variable(np_var)
>>> K.is_keras_tensor(keras_var)  # A variable created with the keras backend is not a Keras
False
>>> keras_placeholder = K.placeholder(shape=(2, 4, 5))
>>> K.is_keras_tensor(keras_placeholder)  # A placeholder is not a Keras tensor.
False
>>> keras_input = Input([10])
>>> K.is_keras_tensor(keras_input) # An Input is a Keras tensor.
True
>>> keras_layer_output = Dense(10)(keras_input)
>>> K.is_keras_tensor(keras_layer_output) # Any Keras layer output is a Keras tensor.
True
```

placeholder

```
keras.backend.placeholder(shape=None, ndim=None, dtype=None, sparse=False, name=None)
```

プレースホルダーのテンソルをインスタンス化し，それを返します．

引数

- **shape**: プレースホルダーのshape（整数のタプル，`None`を含んでいても構いません）．
- **ndim**: テンソルの軸の数．少なくとも{`shape`, `ndim`}から一つ指定する必要があります．両方が指定されると，`shape`が使われます．
- **dtype**: プレースホルダーの型．
- **sparse**: プレースホルダーがスパースの型を持つべきかどうかの真理値．
- **name**: このプレースホルダーに対する任意の名前を表す文字列．

戻り値

テンソルのインスタンス（Kerasのメタ情報が含まれています）．

例

```
>>> from keras import backend as K
>>> input_ph = K.placeholder(shape=(2, 4, 5))
>>> input_ph._keras_shape
(2, 4, 5)
>>> input_ph
<tf.Tensor 'Placeholder_4:0' shape=(2, 4, 5) dtype=float32>
```

---

is_placeholder

```
keras.backend.is_placeholder(x)
```

`x`がプレースホルダか否かを返します。

引数

- **x**: プレースホルダの候補

戻り値

真理値。

---

shape

```
keras.backend.shape(x)
```

テンソル，または変数のshapeを返します．

引数

- **x**: テンソル，または変数.

戻り値

テンソルで表されたshape.

例

```
# TensorFlow example
>>> from keras import backend as K
>>> tf_session = K.get_session()
>>> val = np.array([[1, 2], [3, 4]])
>>> kvar = K.variable(value=val)
>>> inputs = keras.backend.placeholder(shape=(2, 4, 5))
>>> K.shape(kvar)
<tf.Tensor 'Shape_8:0' shape=(2,) dtype=int32>
>>> K.shape(inputs)
<tf.Tensor 'Shape_9:0' shape=(3,) dtype=int32>
# To get integer shape (Instead, you can use K.int_shape(x))
>>> K.shape(kvar).eval(session=tf_session)
array([2, 2], dtype=int32)
>>> K.shape(inputs).eval(session=tf_session)
array([2, 4, 5], dtype=int32)
```

---

int_shape

```
keras.backend.int_shape(x)
```

整数，またはNoneからなるタプルとしての変数，またはテンソルのshapeを返します．

引数

- **x**: テンソル，または変数.

戻り値

整数のタプル（またはNone）．

例

```
>>> from keras import backend as K
>>> inputs = K.placeholder(shape=(2, 4, 5))
>>> K.int_shape(inputs)
(2, 4, 5)
>>> val = np.array([[1, 2], [3, 4]])
>>> kvar = K.variable(value=val)
>>> K.int_shape(kvar)
(2, 2)
```

ndim

```
keras.backend.ndim(x)
```

テンソルの軸の数を整数で返します.

引数

- **x**: テンソル，または変数.

戻り値

軸の数を表す整数（スカラー）.

例

```
>>> from keras import backend as K
>>> inputs = K.placeholder(shape=(2, 4, 5))
>>> val = np.array([[1, 2], [3, 4]])
>>> kvar = K.variable(value=val)
>>> K.ndim(inputs)
3
>>> K.ndim(kvar)
2
```

---

dtype

```
keras.backend.dtype(x)
```

Kerasのテンソル，または変数のdtypeを文字列で返します.

引数

- **x**: テンソル，または変数.

戻り値

`x` のdtypeを表す文字列.

例

```
>>> from keras import backend as K
>>> K.dtype(K.placeholder(shape=(2,4,5)))
'float32'
>>> K.dtype(K.placeholder(shape=(2,4,5), dtype='float32'))
'float32'
>>> K.dtype(K.placeholder(shape=(2,4,5), dtype='float64'))
'float64'
# Keras variable
>>> kvar = K.variable(np.array([[1, 2], [3, 4]]))
>>> K.dtype(kvar)
'float32_ref'
>>> kvar = K.variable(np.array([[1, 2], [3, 4]]), dtype='float32')
>>> K.dtype(kvar)
'float32_ref'
```

---

eval

```
keras.backend.eval(x)
```

テンソルの変数値を評価します.

引数

- **x**: 変数.

戻り値

Numpy 配列.

例

```
>>> from keras import backend as K
>>> kvar = K.variable(np.array([[1, 2], [3, 4]]), dtype='float32')
>>> K.eval(kvar)
array([[ 1.,  2.],
       [ 3.,  4.]], dtype=float32)
```

---

zeros

```
keras.backend.zeros(shape, dtype=None, name=None)
```

全要素が0の変数をインスタンス化し，それを返します.

引数

- **shape**: 整数のタプル．返されたKerasの変数に対するshape.
- **dtype**: 文字列．返されたKerasの変数に対するデータの型.
- **name**: 文字列．返されたKerasの変数に対する名前.

戻り値

`0.0` で埋まった変数（Kerasのメタ情報が含まれています）． `shape` シンボリックだった場合、変数を返せず、その代わりに動的な形のテンソルを返すことに注意してください。

例

```
>>> from keras import backend as K
>>> kvar = K.zeros((3,4))
>>> K.eval(kvar)
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]], dtype=float32)
```

---

ones

```
keras.backend.ones(shape, dtype=None, name=None)
```

全要素が1の変数をインスタンス化し，それを返します．

引数

- **shape**: 整数のタプル．返されたKerasの変数に対するshape．
- **dtype**: 文字列．返されたKerasの変数に対するデータの型．
- **name**: 文字列．返されたKerasの変数に対する名前．

戻り値

`1.0` で埋まった変数． `shape` シンボリックだった場合、変数を返せず、その代わりに動的な形のテンソルを返すことに注意してください。

例

```
>>> from keras import backend as K
>>> kvar = K.ones((3,4))
>>> K.eval(kvar)
array([[ 1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.]], dtype=float32)
```

---

eye

```
keras.backend.eye(size, dtype=None, name=None)
```

単位行列をインスタンス化し，それを返します．

引数

- **shape**: 整数のタプル．返されたKerasの変数に対するshape．

- **dtype**: 文字列．返されたKerasの変数に対するデータの型．
- **name**: 文字列．返されたKerasの変数に対する名前．

戻り値

単位行列を表すKerasの変数．

例

```
>>> from keras import backend as K
>>> kvar = K.eye(3)
>>> K.eval(kvar)
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]], dtype=float32)
```

---

zeros_like

```
keras.backend.zeros_like(x, dtype=None, name=None)
```

別のテンソルと同じshapeを持つ全要素が0の変数のインスタンスを作成します．

引数

- **x**: Kerasのテンソル，または変数．
- **dtype**: 文字列．返されたKerasの変数に対するデータの型．
- **name**: 文字列．返されたKerasの変数に対する名前．

戻り値

xのshapeを持つ全要素が0のKerasの変数．

例

```
>>> from keras import backend as K
>>> kvar = K.variable(np.random.random((2,3)))
>>> kvar_zeros = K.zeros_like(kvar)
>>> K.eval(kvar_zeros)
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]], dtype=float32)
```

---

ones_like

```
keras.backend.ones_like(x, dtype=None, name=None)
```

別のテンソルと同じshapeを持つ全要素が1の変数のインスタンスを作成します．

引数

- **x**: Kerasのテンソル，または変数.
- **dtype**: 文字列．返されたKerasの変数に対するデータの型.
- **name**: 文字列．返されたKerasの変数に対する名前.

戻り値

xのshapeを持つ全要素が1のKerasの変数.

例

```
>>> from keras import backend as K
>>> kvar = K.variable(np.random.random((2,3)))
>>> kvar_ones = K.ones_like(kvar)
>>> K.eval(kvar_ones)
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.]], dtype=float32)
```

---

identity

```
keras.backend.identity(x, name=None)
```

入力されたテンソルと同じ内容を持つテンソルを返します.

引数

- **x**: 入力テンソル.
- **name**: 文字列、作る変数の名前。

戻り値

同じshape，型，及び内容を持つテンソル.

---

random_uniform_variable

```
keras.backend.random_uniform_variable(shape, low, high, dtype=None, name=None, seed=None)
```

一様分布からサンプリングされた値を持つ変数のインスタンスを作成します.

引数

- **shape**: 整数のタプル．返されたKerasの変数に対するshape.
- **low**: 浮動小数点数．出力の区間における下限.
- **high**: 浮動小数点数．出力の区間における上限.

- **dtype**: 文字列．返されたKerasの変数に対するデータの型．
- **name**: 文字列．返されたKerasの変数に対する名前．
- **seed**: 整数．ランダムシード値．

戻り値

サンプリング値で埋まったKerasの変数．

例

```
# TensorFlow example
>>> kvar = K.random_uniform_variable((2,3), 0, 1)
>>> kvar
<tensorflow.python.ops.variables.Variable object at 0x10ab40b10>
>>> K.eval(kvar)
array([[ 0.10940075,  0.10047495,  0.476143  ],
       [ 0.66137183,  0.00869417,  0.89220798]], dtype=float32)
```

---

random_normal_variable

```
keras.backend.random_normal_variable(shape, mean, scale, dtype=None, name=None, seed=None)
```

ガウス分布からサンプリングされた値を持つ変数のインスタンスを作成します．

引数

- **shape**: 整数のタプル．返されたKerasの変数に対するshape．
- **mean**: 浮動小数点数．ガウス分布の平均．
- **scale**: 浮動小数点数．ガウス分布の標準偏差．
- **dtype**: 文字列．返されたKerasの変数に対するデータの型．
- **name**: 文字列．返されたKerasの変数に対する名前．
- **seed**: 整数．ランダムシード値．

戻り値

サンプリング値で埋まったKerasの変数．

例

```
# TensorFlow example
>>> kvar = K.random_normal_variable((2,3), 0, 1)
>>> kvar
<tensorflow.python.ops.variables.Variable object at 0x10ab12dd0>
>>> K.eval(kvar)
array([[ 1.19591331,  0.68685907, -0.63814116],
       [ 0.92629528,  0.28055015,  1.70484698]], dtype=float32)
```

---

count_params

```
keras.backend.count_params(x)
```

Kerasの変数におけるスカラーの数を返します.

引数

- **x**: Kerasの変数.

戻り値

`x` におけるスカラーの数を表す整数.

例

```
>>> kvar = K.zeros((2,3))
>>> K.count_params(kvar)
6
>>> K.eval(kvar)
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]], dtype=float32)
```

---

cast

```
keras.backend.cast(x, dtype)
```

テンソルを異なる型にキャストします.

Kerasの変数をキャストできますが，Kerasのテンソルが返されます.

引数

- **x**: Kerasのテンソル（または変数）.
- **dtype**: 文字列. `'float16'`, `'float32'`, または `'float64'` のいずれか.

戻り値

`dtype` を持つKerasのテンソル.

例

```
>>> from keras import backend as K
>>> input = K.placeholder((2, 3), dtype='float32')
>>> input
<tf.Tensor 'Placeholder_2:0' shape=(2, 3) dtype=float32>
# It doesn't work in-place as below.
>>> K.cast(input, dtype='float16')
<tf.Tensor 'Cast_1:0' shape=(2, 3) dtype=float16>
>>> input
<tf.Tensor 'Placeholder_2:0' shape=(2, 3) dtype=float32>
# you need to assign it.
>>> input = K.cast(input, dtype='float16')
>>> input
<tf.Tensor 'Cast_2:0' shape=(2, 3) dtype=float16>
```

---

update

```
keras.backend.update(x, new_x)
```

`x`の値を`new_x`のものに更新する.

引数

- **x**: 変数.
- **new_x**: `x`と同じshapeを持つテンソル.

戻り値

更新された`x`.

---

update_add

```
keras.backend.update_add(x, increment)
```

`x`の値を`increment`で加算することで更新する.

引数

- **x**: 変数.
- **increment**: `x`と同じshapeを持つテンソル.

戻り値

更新された`x`.

---

update_sub

```
keras.backend.update_sub(x, decrement)
```

`x`の値を`decrement`で減算することで更新する.

引数

- **x**: 変数.
- **decrement**: `x`と同じshapeを持つテンソル.

戻り値

更新された`x`.

---

moving_average_update

```
keras.backend.moving_average_update(x, value, momentum)
```

変数における移動平均を計算します.

引数

- **x**: `Variable`
- **value**: `x`と同じshapeを持つテンソル.
- **momentum**: 移動平均のモーメンタム.

戻り値

変数を更新するための命令.

---

dot

```
keras.backend.dot(x, y)
```

2つのテンソル（かつ/または変数）を掛け合わせ，テンソルを返します.

n階テンソルにn次元のを掛ける場合，Theanoの振る舞いを再現します
（例`(2, 3) * (4, 3, 5) -> (2, 4, 5)`）.

引数

- **x**: テンソル，または変数.
- **y**: テンソル，または変数.

戻り値

`x`と`y`でドット積を行なったテンソル.

例

```
# dot product between tensors
>>> x = K.placeholder(shape=(2, 3))
>>> y = K.placeholder(shape=(3, 4))
>>> xy = K.dot(x, y)
>>> xy
<tf.Tensor 'MatMul_9:0' shape=(2, 4) dtype=float32>
```

```
# dot product between tensors
>>> x = K.placeholder(shape=(32, 28, 3))
>>> y = K.placeholder(shape=(3, 4))
>>> xy = K.dot(x, y)
>>> xy
<tf.Tensor 'MatMul_9:0' shape=(32, 28, 4) dtype=float32>
```

```
# Theano-like behavior example
>>> x = K.random_uniform_variable(shape=(2, 3), low=0, high=1)
>>> y = K.ones((4, 3, 5))
>>> xy = K.dot(x, y)
>>> K.int_shape(xy)
(2, 4, 5)
```

---

batch_dot

```
keras.backend.batch_dot(x, y, axes=None)
```

バッチ式のドット積.

`batch_dot`は`x`と`y`がバッチに含まれる，すなわち`(batch_size, :)`のshapeの中で，`x`と`y`のドット積を計算するために使われます．`batch_dot`の結果は入力より小さい次元を持つテンソルになります．次元数が1になれば，ndimが少なくとも2であることを保証するために`expand_dims`を利用します.

引数

- **x**: `ndim >= 2`のKerasのテンソル.
- **y**: `ndim >= 2`のKerasのテンソル.
- **axes**: 目標となる次元を持つ整数のリスト（もしくは整数単体）．`axes[0]`と`axes[1]`の長さは同じにすべきです.

戻り値

（次元数の総和より少ない）`x`のshapeと（バッチの次元の総和より少ない）`y`のshapeを連結したshapeに等しいテンソル．もし最後のランクが1なら，`(batch_size, 1)`に整形します.

例

`x = [[1, 2], [3,4]]`, `y = [[5, 6], [7, 8]]` と仮定すると，非対角成分を計算しなくても，`x.dot(y.T)` の主対角成分である `batch_dot(x, y, axes=1) = [[17, 53]]` が得られます．

shapeの推定: `x` と `y` のshapeがそれぞれ `(100, 20)`，`(100, 30, 20)` としましょう．`axes` が(1, 2)の場合，出力されたテンソルのshapeを見つけるために，`x` と `y` のshapeにおけるそれぞれの次元でループさせることになります．

- `x.shape[0]` : 100: 出力されるshapeに付加されます．
- `x.shape[1]` : 20: 出力されるshapeには付加されず，`x` の次元1は総和が取られています（`dot_axes[0]` = 1）．
- `y.shape[0]` : 100: 出力されるshapeには付加されず，`y` の最初の次元はいつも無視されます．
- `y.shape[1]` : 30: 出力されるshapeに付加されます．
- `y.shape[2]` : 20: 出力されるshapeには付加されず，`y` の次元1は総和が取られています（`dot_axes[1]` = 2）`output_shape` = `(100, 30)`．

```
>>> x_batch = K.ones(shape=(32, 20, 1))
>>> y_batch = K.ones(shape=(32, 30, 20))
>>> xy_batch_dot = K.batch_dot(x_batch, y_batch, axes=[1, 2])
>>> K.int_shape(xy_batch_dot)
(32, 1, 30)
```

---

transpose

```
keras.backend.transpose(x)
```

行列を転置します．

引数

- **x**: テンソル，または変数．

戻り値

テンソル．

例

```
>>> var = K.variable([[1, 2, 3], [4, 5, 6]])
>>> K.eval(var)
array([[ 1.,  2.,  3.],
       [ 4.,  5.,  6.]], dtype=float32)
>>> var_transposed = K.transpose(var)
>>> K.eval(var_transposed)
array([[ 1.,  4.],
       [ 2.,  5.],
       [ 3.,  6.]], dtype=float32)
```

```
>>> inputs = K.placeholder((2, 3))
>>> inputs
<tf.Tensor 'Placeholder_11:0' shape=(2, 3) dtype=float32>
>>> input_transposed = K.transpose(inputs)
>>> input_transposed
<tf.Tensor 'transpose_4:0' shape=(3, 2) dtype=float32>
```

---

gather

```
keras.backend.gather(reference, indices)
```

テンソルの`reference`における添字の要素`indices`を探索します.

引数

- **reference**: テンソル.
- **indices**: 添字の整数テンソル.

戻り値

`reference`と同じ型を持つテンソル.

---

max

```
keras.backend.max(x, axis=None, keepdims=False)
```

テンソル内の最大値.

引数

- **x**: テンソル，または変数.
- **axis**: 整数，最大値を探すため軸.
- **keepdims**: 次元を保つかどうかの真理値. `keepdims`が`False`の場合，テンソルのランクは1に削減します. `keepdims`が`True`の場合，縮小された次元は1の長さにとどめます.

戻り値

`x`の中の最大値を持ったテンソル.

min

```
keras.backend.min(x, axis=None, keepdims=False)
```

テンソル内の最大値.

引数

- **x**: テンソル，または変数.
- **axis**: 整数，最小値を探すため軸.
- **keepdims**: 次元を保つかどうかの真理値. `keepdims` が `False` の場合，テンソルのランクは1に削減します. `keepdims` が `True` の場合，縮小された次元は1の長さにとどめます.

戻り値

`x` の中の最小値を持ったテンソル.

---

sum

```
keras.backend.sum(x, axis=None, keepdims=False)
```

テンソルに対して，指定した軸に沿って和を計算します.

引数

- **x**: テンソル，または変数.
- **axis**: 整数. 和を計算する軸方向.
- **keepdims**: 次元を保つかどうかの真理値. `keepdims` が `False` の場合，テンソルのランクは1に削減します. `keepdims` が `True` の場合，縮小された次元は1の長さにとどめます.

戻り値

`x` の和をとったテンソル.

---

prod

```
keras.backend.prod(x, axis=None, keepdims=False)
```

テンソルに対して，指定した軸に沿って積を計算します.

引数

- **x**: テンソル，または変数.

- **axis**: 整数．積を計算する軸方向．
- **keepdims**: 次元を保つかどうかの真理値．`keepdims`が`False`の場合，テンソルのランクは1に削減します．`keepdims`が`True`の場合，縮小された次元は1の長さにとどめます．

戻り値

`x`の積をとったテンソル．

---

cumsum

```
keras.backend.cumsum(x, axis=0)
```

テンソルに対して，指定した軸に沿って累積和を計算します．

引数

- **x**: テンソル，または変数．
- **axis**: 整数．和を計算する軸方向．

戻り値

`x`を`axis`に沿って累積和をとったテンソル．

---

cumprod

```
keras.backend.cumprod(x, axis=0)
```

テンソルに対して，指定した軸に沿って累積積を計算します．

引数

- **x**: テンソル，または変数．
- **axis**: 整数．積を計算する軸方向．

戻り値

`x`を`axis`に沿って累積積をとったテンソル．

---

var

```
keras.backend.var(x, axis=None, keepdims=False)
```

指定した軸に沿ったテンソルの分散を計算します．

引数

- **x**: テンソル，または変数．
- **axis**: 整数．分散を計算する軸方向．
- **keepdims**: 次元を保つかどうかの真理値． `keepdims` が `False` の場合，テンソルのランクは1に削減します． `keepdims` が `True` の場合，縮小された次元は1の長さにとどめます．

戻り値

`x` の要素の分散を持つテンソル．

---

std

```
std(x, axis=None, keepdims=False)
```

指定した軸に沿ったテンソルの標準偏差を計算します．

引数

- **x**: テンソル，または変数．
- **axis**: 整数．標準偏差を計算する軸方向．
- **keepdims**: 次元を保つかどうかの真理値． `keepdims` が `False` の場合，テンソルのランクは1に削減します． `keepdims` が `True` の場合，縮小された次元は1の長さにとどめます．

戻り値

`x` の要素の標準偏差を持つテンソル．

---

mean

```
keras.backend.mean(x, axis=None, keepdims=False)
```

指定した軸に沿ったテンソルの平均を計算します．

引数

- **x**: テンソル，または変数．
- **axis**: 整数．平均を計算する軸方向．
- _keepdims: 次元を保つかどうかの真理値． `keepdims` が `False` の場合，テンソルのランクは1に削減します． `keepdims` が `True` の場合，縮小された次元は1の長さにとどめます．

戻り値

`x`の要素の平均を持つテンソル.

---

any

```
keras.backend.any(x, axis=None, keepdims=False)
```

ビット単位の縮約（論理OR）.

引数

- **x**: テンソル，または変数.
- **axis**: 整数．縮約する軸方向.
- **keepdims**: 次元を保つかどうかの真理値． `keepdims` が `False` の場合，テンソルのランクは1に削減します． `keepdims` が `True` の場合，縮小された次元は1の長さにとどめます.

戻り値

uint8のテンソル.

---

all

```
keras.backend.all(x, axis=None, keepdims=False)
```

ビット単位の縮約（論理AND）.

引数

- **x**: テンソル，または変数.
- **axis**: 整数．縮約する軸方向.
- **keepdims**: 次元を保つかどうかの真理値． `keepdims` が `False` の場合，テンソルのランクは1に削減します． `keepdims` が `True` の場合，縮小された次元は1の長さにとどめます.

戻り値

uint8のテンソル.

---

argmax

```
keras.backend.argmax(x, axis=-1)
```

テンソルの軸に沿った最大値の添字を返します.

引数

- **x**: テンソル，または変数.
- **axis**: 整数．縮約する軸方向.

戻り値

テンソル.

---

argmin

```
keras.backend.argmin(x, axis=-1)
```

テンソルの軸に沿った最小値の添字を返します.

引数

- **x**: テンソル，または変数.
- **axis**: 整数．縮約する軸方向.

戻り値

テンソル.

---

square

```
keras.backend.square(x)
```

要素ごとの二乗.

引数

- **x**: テンソル，または変数.

戻り値

テンソル.

---

abs

```
keras.backend.abs(x)
```

要素ごとの絶対値.

引数

- **x**: テンソル，または変数.

戻り値

テンソル.

---

sqrt

```
keras.backend.sqrt(x)
```

要素ごとの平方根.

引数

- **x**: テンソル，または変数.

戻り値

テンソル.

---

exp

```
keras.backend.exp(x)
```

要素ごとの指数関数値.

引数

- **x**: テンソル，または変数.

戻り値

テンソル.

---

log

```
keras.backend.log(x)
```

要素ごとの対数.

引数

- **x**: テンソル，または変数.

戻り値

テンソル.

---

logsumexp

```
keras.backend.logsumexp(x, axis=None, keepdims=False)
```

log(sum(exp(テンソルの次元を横断した要素)))を計算します.

この関数はlog(sum(exp(x)))よりも計算上安定します．小さい入力に対して対数をとることで発生するアンダーフローと，大きな入力に対して指数関数にかけることで発生するオーバーフローを回避します.

引数

- **x**: テンソル，または変数.
- **axis**: 整数．縮約する軸方向.
- **keepdims**: 次元を保つかどうかの真理値． `keepdims` が `False` の場合，テンソルのランクは1に削減します． `keepdims` が `True` の場合，縮小された次元は1の長さにとどめます.

戻り値

縮約されたテンソル.

---

round

```
keras.backend.round(x)
```

要素ごとの最も近い整数への丸め.

同点であれば偶数よりに丸め込まれます。

引数

- **x**: テンソル，または変数.

戻り値

テンソル.

---

sign

```
keras.backend.sign(x)
```

要素ごとの符号.

引数

- **x**: テンソル，または変数.

戻り値

テンソル.

---

pow

```
keras.backend.pow(x, a)
```

要素ごとの指数乗.

引数

- **x**: テンソル，または変数.
- **a**: Pythonの整数.

戻り値

テンソル.

---

clip

```
keras.backend.clip(x, min_value, max_value)
```

要素ごとのクリッピング.

引数

- **x**: テンソル，または変数.
- **min_value**: Pythonの浮動小数点数，または整数.
- **max_value**: Pythonの浮動小数点数，または整数.

戻り値

テンソル.

---

equal

```
keras.backend.equal(x, y)
```

2つのテンソル間の要素ごとの等値性.

引数

- **x**: テンソル，または変数.
- **y**: テンソル，または変数.

戻り値

真理値からなるテンソル.

---

not_equal

```
keras.backend.not_equal(x, y)
```

2つのテンソル間の要素ごとの不等性.

引数

- **x**: テンソル，または変数.
- **y**: テンソル，または変数.

戻り値

真理値からなるテンソル.

---

greater

```
keras.backend.greater(x, y)
```

要素ごとの(x > y)の真理値.

引数

- **x**: テンソル，または変数.
- **y**: テンソル，または変数.

戻り値

真理値からなるテンソル.

greater_equal

```
keras.backend.greater_equal(x, y)
```

要素ごとの(x >= y)の真理値.

引数

- **x**: テンソル，または変数.
- **y**: テンソル，または変数.

戻り値

真理値からなるテンソル.

---

less

```
keras.backend.less(x, y)
```

要素ごとの(x < y)の真理値.

引数

- **x**: テンソル，または変数.
- **y**: テンソル，または変数.

戻り値

真理値からなるテンソル.

---

less_equal

```
keras.backend.less_equal(x, y)
```

要素ごとの(x <= y)の真理値.

引数

- **x**: テンソル，または変数.
- **y**: テンソル，または変数.

戻り値

真理値からなるテンソル.

---

maximum

```
keras.backend.maximum(x, y)
```

2つのテンソルの要素ごとの最大値.

引数

- **x**: テンソル，または変数.
- **y**: テンソル，または変数.

戻り値

テンソル.

---

minimum

```
keras.backend.minimum(x, y)
```

2つのテンソルの要素ごとの最小値.

引数

- **x**: テンソル，または変数.
- **y**: テンソル，または変数.

戻り値

テンソル.

---

sin

```
keras.backend.sin(x)
```

要素ごとにxのsinを計算します.

引数

- **x**: テンソル，または変数.

戻り値

テンソル.

---

cos

```
keras.backend.cos(x)
```

要素ごとにxのcosを計算します.

引数

- **x**: テンソル，または変数.

戻り値

テンソル.

---

normalize_batch_in_training

```
keras.backend.normalize_batch_in_training(x, gamma, beta, reduction_axes, epsilon=0.001)
```

平均と標準偏差を計算したのちに，バッチとしてbatch_normalizationを適用します.

引数

- **x**: テンソル，または変数.
- **gamma**: 入力をスケールするためのテンソル.
- **beta**: 入力を補正するためのテンソル.
- **reduction_axes**: 繰り返し可能な整数，軸上の値すべてにわたって正規化を行う.
- **epsilon**: 微小量.

戻り値

3つの要素 `(normalize_tensor, mean, variance)` からなるタプル.

---

batch_normalization

```
keras.backend.batch_normalization(x, mean, var, beta, gamma, epsilon=0.001)
```

与えられたmean，var，beta，gammaを使ってxにbatch normalizationを適用します.

すなわち，`output = (x - mean) / (sqrt(var) + epsilon) * gamma + beta` が返されます.

引数

- **x**: テンソル，または変数.
- **mean**: バッチにおける平均.
- **var**: バッチにおける分散.
- **gamma**: 入力をスケールするためのテンソル.
- **beta**: 入力を補正するためのテンソル.
- **reduction_axes**: 繰り返し可能な整数，軸上の値すべてにわたって正規化を行う.
- **epsilon**: 微小量.

戻り値

テンソル.

---

concatenate

```
keras.backend.concatenate(tensors, axis=-1)
```

指定した軸に沿ってテンソルのリストを連結します.

引数

- **tensor**: 連結するためのテンソルのリスト.
- **axis**: 連結する軸方向.

戻り値

テンソル.

---

reshape

```
keras.backend.reshape(x, shape)
```

指定したshapeにテンソルを整形します.

引数

- **x**: テンソル，または変数.
- **shape**: shapeのタプル.

戻り値

テンソル.

permute_dimensions

```
keras.backend.permute_dimensions(x, pattern)
```

テンソルにおける軸の順序を変更します.

引数

- **x**: テンソル，または変数.
- **pattern**: 次元の添字かなるタプル，e.g. `(0, 2, 1)`.

戻り値

テンソル.

---

resize_images

```
keras.backend.resize_images(x, height_factor, width_factor, data_format)
```

4階テンソルに含まれる画像をリサイズします.

引数

- **x**: リサイズのためのテンソル，または変数.
- **height_factor**: 自然数.
- **width_factor**: 自然数.
- **data_format**: `channels_first`，または`channels_last"`のどちらか.

戻り値

テンソル.

Raises

- **ValueError**: `data_format`が`channels_last`，または`channels_first`ではない場合.

---

resize_volumes

```
keras.backend.resize_volumes(x, depth_factor, height_factor, width_factor, data_format)
```

5階テンソルに含まれるvolumeをリサイズします.

引数

- **x**: リサイズのためのテンソル，または変数.
- **depth_factor**: 自然数.
- **height_factor**: 自然数.
- **width_factor**: 自然数.
- **data_format**: `channels_first`，または`channels_last`のどちらか.

戻り値

テンソル.

Raises

- **ValueError**: `data_format`が`channels_last`，または`channels_first`ではない場合.

---

repeat_elements

```
keras.backend.repeat_elements(x, rep, axis)
```

`np.repeat`のように，軸に沿ってテンソルの要素を繰り返します.

`x`がshape`(s1, s2, s3)`を持ち，`axis`が`1`の場合，この出力は
shape`(s1, s2 * rep, s3)`を持ちます.

引数

- **x**: テンソル，または変数.
- **rep**: Pythonの整数，繰り返す回数.
- **axis**: 繰り返す軸方向.

Raises

- **ValueError**: `x.shape[axis]`が定義されていない場合.

戻り値

テンソル.

---

repeat

```
keras.backend.repeat(x, n)
```

2階テンソルを繰り返します.

`x`がshape (samples, dim)を持ち`n`が`2`であれば，この出力はshape`(samples, 2, dim)`を持ちます．

引数

- **x**: テンソル，または変数．
- **n**: Pythonの整数，繰り返す回数．

戻り値

テンソル．

---

arange

```
keras.backend.arange(start, stop=None, step=1, dtype='int32')
```

整数の並びからなる1階テンソルを作成します．

関数の引数はTheanoのarangeの慣例と同じです: 唯一の引数が与えられた場合，実際には"stop"の引数です．

返されたテンソルのデフォルトの型は`'int32'`でTensorFlowのデフォルトと一致します．

引数

- **start**: 始めの値．
- **stop**: 終わりの値．
- **step**: 2つの連続値の差分．
- **dtype**: 整数のデータ型．

戻り値

整数のテンソル．

---

tile

```
tile(x, n)
```

`x`を`n`でタイル状に配置したテンソルを作成します．

引数

- **x**: テンソル，または変数．

- **n**: 整数のリスト. `x` の次元数と同じでなければなりません.

戻り値

タイル状に配置されたテンソル.

---

flatten

```
keras.backend.flatten(x)
```

平滑化されたテンソル.

引数

- **x**: テンソル，または変数.

戻り値

1次元に整形されたテンソル.

---

batch_flatten

```
keras.backend.batch_flatten(x)
```

n階テンソルを0番目の次元が保たれるように2階テンソルに変換します.

言い換えると，バッチのそれぞれのサンプルに対して平滑化を行います.

引数

- **x**: テンソル，または変数.

戻り値

テンソル.

---

expand_dims

```
keras.backend.expand_dims(x, axis=-1)
```

添字"axis"でのサイズ1の次元を加えます.

引数

- **x**: テンソル，または変数.
- **axis**: 新しい軸を追加する場所.

戻り値

次元が拡張されたテンソル.

---

squeeze

```
keras.backend.squeeze(x, axis)
```

テンソルから添字"axis"での1次元を除きます.

引数

- **x**: テンソル，または変数.
- **axis**: 削除する軸.

戻り値

`x`と同じデータで，次元が削除されたテンソル.

---

temporal_padding

```
keras.backend.temporal_padding(x, padding=(1, 1))
```

3階テンソルの真ん中の次元に対してパディングを行います.

引数

- **x**: テンソル，または変数.
- **padding**: 2つの整数からなるタプル．次元1の始めと終わりにいくつ0をパディングするか.

戻り値

パディングされた3階テンソル.

---

spatial_2d_padding

```
keras.backend.spatial_2d_padding(x, padding=((1, 1), (1, 1)), data_format=None)
```

4階テンソルの2番目と3番目の次元に対してパディングを行います.

引数

- **x**: テンソル，または変数.
- **padding**: 2つのタプルのタプル．パディングのパターン.
- **data_format**: `channels_last` か `channels_first` のどちらか.

戻り値

パディングされた4階テンソル.

Raises

- **ValueError**: `data_format` が `channels_last` ，または `channels_first` ではない場合.

---

spatial_3d_padding

```
keras.backend.spatial_3d_padding(x, padding=((1, 1), (1, 1), (1, 1)), data_format=None)
```

5階テンソルに対して深さ，高さ，幅を表す次元に沿って0パディングを行います.

"padding[0]"，"padding[1]"，かつ"padding[2]"それぞれの次元に対して左右を0パディングします.

'channels_last'のdata_formatに対して，2，3，4番目の次元がパディングされます．'channels_first'のdata_formatに対して，3，4，5番目の次元がパディングされます.

引数

- **x**: テンソル，または変数.
- **padding**: 3つのタプルのタプル．パディングのパターン.
- **data_format**: `channels_last` か `channels_first` のどちらか.

戻り値

パディングされた5階テンソル.

Raises

- **ValueError**: `data_format` が `channels_last` ，または `channels_first` ではない場合.

---

stack

```
keras.backend.stack(x, axis=0)
```

ランク`R`のテンソルのリストをランク`R+1`のテンソルに積み上げます.

引数

- **x**: テンソルのリスト.
- **axis**: 積み上げる軸方向.

戻り値

テンソル.

---

one_hot

```
keras.backend.one_hot(indices, num_classes)
```

整数のテンソルone-hot表現を導出します.

引数

- **indices**: `(batch_size, dim1, dim2, ... dim(n-1))`のshapeを持つn階テンソル.
- **num_classes**: 整数. いくつのクラスを考慮するか.

戻り値

`(batch_size, dim1, dim2, ... dim(n-1), num_classes)`のshapeを持つ(n + 1)次元のone-hot表現が含まれたテンソル.

---

reverse

```
keras.backend.reverse(x, axes)
```

指定した軸に沿ってテンソルを逆順にする.

引数

- **x**: 逆順にするテンソル.
- **axes**: 整数，または繰り返し可能な整数. 逆順にする軸.

戻り値

テンソル.

---

get_value

```
keras.backend.get_value(x)
```

変数の値を返します.

引数

- **x**: 入力変数.

戻り値

Numpy 配列.

---

batch_get_value

```
keras.backend.batch_get_value(ops)
```

1つ以上のテンソルの変数の値を返します.

引数

- **ops**: 実行する命令のリスト.

戻り値

Numpy 配列のリスト.

---

set_value

```
keras.backend.set_value(x, value)
```

Numpy 配列から，変数の値を設定します.

引数

- **x**: 新しい値をセットするテンソル.
- **value**: Numpy 配列（同じshapeを持ちます）テンソルにセットする値.

---

batch_set_value

```
keras.backend.batch_set_value(tuples)
```

複数のテンソルの変数の値を一度にセットします.

引数

- **tuples**: `(tensor, value)` のタプルのリスト. `value` はNumpy 配列であるべきです.

---

print_tensor

```
keras.backend.print_tensor(x, message='')
```

`message` と評価されたテンソルの値を表示します. `print_tensor` は次のコードのように使われると `x` と等価な新しいテンソルを返すことに留意してください。 そうしないと表示処理は評価中に考慮されません。

例

```
>>> x = K.print_tensor(x, message="x is: ")
```

引数

- **x**: 表示するテンソル.
- **message**: テンソルと一緒に表示するメッセージ.

戻り値

`x` と同じテンソル.

---

function

```
function(inputs, outputs, updates=None)
```

Kerasの関数のインスタンスを作成します.

引数

- **inputs**: プレースホルダーテンソルのリスト.
- **outputs**: 出力のテンソルのリスト.
- **updates**: 更新する命令のリスト.
- ****kwargs**: TensorFlowでは利用されません.

戻り値

Numpy 配列.

---

gradients

```
gradients(loss, variables)
```

`variables` の `loss` に関しての勾配を返します．

引数

- **loss**: 最小化するためのスカラーからなるテンソル．
- **variables**: 変数のリスト．

戻り値

勾配からなるテンソル．

---

stop_gradient

```
stop_gradient(variables)
```

全ての変数に関して，0の勾配を持つ `variable` を返します．

引数

- **variables**: 変数のリスト．

戻り値

同様の変数のリスト．

---

rnn

```
rnn(step_function, inputs, initial_states, go_backwards=False, mask=None, constants=None, un
```

テンソルの時間次元にわたって反復します．

引数

- **step_function**: RNN のステップ関数
- **Parameters**:
  - **input**: shape `(samples, ...)` （時間次元はありません）を持つテンソルで，ある時間ステップでのサンプルのバッチに対する入力を表します．
  - **states**: テンソルのリスト．
- **戻り値**:
  - **output**: shape `(samples, output_dim)` を持つテンソル（時間次元はありません）．

- **new_states**: 'states'と同じ長さとshapeを持つテンソルのリスト．リストの中の最初のステートは前回の時間ステップでの出力されたテンソルでなければなりません．

- **inputs**: shape `(samples, time, ...)` を持つ一時的なテンソル（少なくとも3次元です）．
- **initial_states**: ステップ関数で利用される状態に対する初期値を含む，shape (samples, output_dim) を持つテンソル（時間軸を持たない）．ステップ関数で扱うstatesの初期値が含まれます．
- **go_backwards**: 真理値．真ならば，逆順で時間次元にわたって反復します．
- **mask**: マスクされたすべての要素に対して0となるような，shape `(samples, time, 1)` を持つバイナリ型のテンソル．
- **constants**: 各ステップで渡される定数値のリスト．
- **unroll**: RNNをアンロールするか，またはシンボリックループ（バックエンドに応じた `while_loop`，または `scan`）どうか．
- **input_length**: TensorFlowの実装では関係ありません．Theanoでアンロールを利用するときは指定する必要があります．

戻り値

`(last_output, outputs, new_states)` のタプル．

- **last_output**: shape `(samples, ...)` を持つRNNの最新の出力．
- **outputs**: 各 `output[s, t]` がサンプル `s` に対する時刻 `t` でのステップ関数の出力であるような，shape `(samples, time, ...)` を持つテンソル
- **new_states**: shape `(samples, ...)` を持つ，ステップ関数で返される最新の状態を表すテンソルのリスト．

Raises

- **ValueError**: 3以下の次元の入力が与えられた場合．
- **ValueError**: `unoll` が `True` だが，入力の時間ステップが固定値ではない場合．
- **ValueError**: `None` ではない `mask` が与えられたが，statesが与えられていない（`len(states)` == 0）場合．

---

switch

```
switch(condition, then_expression, else_expression)
```

スカラー値に応じて2つの命令を入れ替えます．

`then_expression` と `else_expression` はともに同じ*shape*を持つシンボリックなテンソルであるべきであることに注意してください．

引数

- **condition**: スカラーからなるテンソル（`整数`，または `真理値`）．
- **then_expression**: テンソル，またはテンソルを返すcallable．

- **else_expression**: テンソル，またはテンソルを返すcallable.

戻り値

選択されたテンソル.

---

in_train_phase

```
in_train_phase(x, alt, training=None)
```

学習フェーズでは`x`を選択し，それ以外では`alt`を選択します.

`alt`は`x`と同じ*shape*を持つべきであることに注意してください.

引数

- **x**: 学習フェーズにおいて何を返すか（テンソル，またはテンソルを返すcallable）.
- **alt**: 学習フェーズ以外において何を返すか（テンソル，またはテンソルを返すcallable）.
- **training**: 学習フェーズを指定した任意のスカラーからなるテンソル（またはPythonの真理値，整数）.

戻り値

`training`のフラグに基づいた`x`，または`alt`のどちらか. `training`のフラグは`K.learning_phase()`をデフォルトにします.

---

in_test_phase

```
in_test_phase(x, alt, training=None)
```

テストフェーズでは`x`を選択し，それ以外では`alt`を選択します.

`alt`は`x`と同じ*shape*を持つべきであることに注意してください.

引数

- **x**: テストフェーズにおいて何を返すか（テンソル，またはテンソルを返すcallable）.
- **alt**: テストフェーズ以外において何を返すか（テンソル，またはテンソルを返すcallable）.
- **training**: 学習フェーズを指定した任意のスカラーからなるテンソル（またはPythonの真理値，整数）.

戻り値

`K.learning_phase` のフラグに基づいた `x`，または `alt` のどちらか．

---

relu

```
relu(x, alpha=0.0, max_value=None)
```

Rectified linear unit．

デフォルトは，要素ごとに `max(x, 0)` を返します．

引数

- **x**: テンソル，または変数．
- **alpha**: スカラー値．負の領域における関数の傾き（デフォルトは `0.`）．
- **max_value**: 飽和度の閾値．

戻り値

テンソル．

---

elu

```
elu(x, alpha=1.0)
```

Exponential linear unit．

引数

- **x**: テンソル，または変数．
- **alpha**: スカラー値．正の領域における関数の傾き．

戻り値

テンソル．

---

softmax

```
softmax(x)
```

Softmax．

引数

- **x**: テンソル，または変数.

戻り値

テンソル.

---

softplus

```
softplus(x)
```

Softplus.

引数

- **x**: テンソル，または変数.

戻り値

テンソル.

---

softsign

```
softsign(x)
```

引数

- **x**: テンソル，または変数.

戻り値

テンソル.

---

categorical_crossentropy

```
categorical_crossentropy(output, target, from_logits=False)
```

出力テンソルと目標テンソルの間のカテゴリカルクロスエントロピー.

引数

- **output**: softmaxに適用したテンソル（`from_logits`がTrueでない限り，`output`はロジット値で表されるでしょう）.
- **target**: `output`と同じshapeからなるテンソル.

- **from_logits**: 真理値． `output` がsoftmaxの結果，またはロジット値からなるテンソルかどうか．

戻り値

出力のテンソル．

---

sparse_categorical_crossentropy

```
sparse_categorical_crossentropy(output, target, from_logits=False)
```

整数の目標におけるカテゴリカルクロスエントロピー．

引数

- **output**: softmaxに適用したテンソル（ `from_logits` がTrueでない限り， `output` はロジット値で表されるでしょう）．
- **target**: 整数のテンソル．
- **from_logits**: 真理値． `output` がsoftmaxの結果，またはロジット値からなるテンソルかどうか．

戻り値

出力のテンソル．

---

binary_crossentropy

```
binary_crossentropy(output, target, from_logits=False)
```

出力テンソルと目標テンソルの間のバイナリクロスエントロピー．

引数

- **output**: softmaxに当てはめたテンソル（ `from_logits` がTrueでない限り， `output` はロジット値で表されるでしょう）．
- **target**: `output` と同じshapeからなるテンソル．
- **from_logits**: 真理値． `output` がsoftmaxの結果，またはロジット値からなるテンソルかどうか．

戻り値

テンソル．

---

sigmoid

```
sigmoid(x)
```

要素ごとのシグモイド.

引数

- **x**: テンソル，または変数.

戻り値

テンソル.

---

hard_sigmoid

```
hard_sigmoid(x)
```

セグメントごとのシグモイドの線形近似.

シグモイドよりも高速. `x < -2.5`の場合，`0.`，`x > 2.5`の場合，`1.`，`-2.5 <= x <= 2.5`の場合，`0.2 * x + 0.5`が返される.

引数

- **x**: テンソル，または変数.

戻り値

テンソル.

---

tanh

```
tanh(x)
```

要素ごとのtanh.

引数

- **x**: テンソル，または変数.

戻り値

テンソル.

---

dropout

```
dropout(x, level, seed=None)
```

`x`の要素をランダムに0にセットし，その上，テンソル全体をスケールさせます．

引数

- **x**: テンソル
- **level**: 0に設定されるテンソルにおける要素の割合
- **noise_shape**: ランダムに生成された保持/棄却のフラグのshapeで，`x`のshapeにブロードキャスト可能でなければなりません．
- **seed**: 決定論を保証するランダムシード．

戻り値

テンソル．

---

l2_normalize

```
l2_normalize(x, axis)
```

指定した軸に沿って，L2ノルムでテンソルを正則化します．

引数

- **x**: テンソル，または変数．
- **axis**: 正則化する軸方向．

戻り値

テンソル．

---

in_top_k

```
in_top_k(predictions, targets, k)
```

`targets`が`predictions`の上位`k`に含まれているかどうか，を返します．

引数

- **predictions**: shape`(batch_size, classes)`で`float32`型のテンソル．
- **target**: 長さ`batch_size`で`int32`，または`int64`の1階テンソル．
- **k**: 整数．上位何件を考慮するかの数．

戻り値

`batch_size`の長さで真理値からなる1階テンソル: `predictions[i]`が上位`k`に含まれていたら`output[i]`は`True`.

---

conv1d

```
conv1d(x, kernel, strides=1, padding='valid', data_format=None, dilation_rate=1)
```

1次元の畳み込み.

引数

- **x**: テンソル，または変数.
- **kernel**: カーネルを表すテンソル.
- **strides**: ストライドの整数.
- **padding**: 文字列． `same`， `causal`，または`valid`.
- **data_format**: 文字列`channels_last`，または`channels_first`のどちらか.
- **dilation_rate**: 整数．ディレーションを行う割合.

戻り値

1次元の畳み込みの結果からなるテンソル.

---

conv2d

```
conv2d(x, kernel, strides=(1, 1), padding='valid', data_format=None, dilation_rate=(1, 1))
```

2次元の畳み込み.

引数

- **x**: テンソル，または変数.
- **kernel**: カーネルを表すテンソル.
- **strides**: ストライドの整数.
- **padding**: 文字列． `same`，または`valid`.
- **data_format**: 文字列． `channels_last`，または`channels_first`のどちらか．入力/カーネル/出力でTheanoもしくはTensorFlowのデータ形式を利用するかどうか.
- **dilation_rate**: 整数のタプル.

戻り値

2次元の畳み込みの結果からなるテンソル.

Raises

- **ValueError**: `data_format` が `channels_last` ，または `channels_first` ではない場合.

---

conv2d_transpose

```
conv2d_transpose(x, kernel, output_shape, strides=(1, 1), padding='valid', data_format=None)
```

2次元の逆畳み込み（すなわち，転置畳み込み）.

引数

- **x**: テンソル，または変数.
- **kernel**: カーネルを表すテンソル.
- **output_shape**: 出力するshapeに対する整数の1階テンソル.
- **strides**: ストライドの整数.
- **padding**: 文字列． `same` ，または `valid` .
- **data_format**: 文字列． `channels_last` ，または `channels_first` のどちらか．入力/カーネル/出力でTheanoもしくはTensorFlowのデータ形式を利用するかどうか.

戻り値

2次元の転置畳み込みの結果からなるテンソル.

Raises

- **ValueError**: `data_format` が `channels_last` ，または `channels_first` ではない場合.

---

separable_conv2d

```
separable_conv2d(x, depthwise_kernel, pointwise_kernel, strides=(1, 1), padding='valid', dat
```

separableフィルタ込みで2次元の畳み込み.

引数

- **x**: テンソル，または変数.
- **depthwise_kernel**: 深さごとの畳み込みに対するカーネル.
- **pointwise_kernel**: 1x1の畳み込みに対するカーネル.
- **strides**: ストライドのタプル（長さ2）.
- **padding**: パディングのモード． `same` ，または `valid` .
- **data_format**: 文字列． `channels_last` ，または `channels_first` のどちらか.
- **dilation_rate**: 整数のタプル．ディレーションを行う割合.

戻り値

出力テンソル.

Raises

- **ValueError**: `data_format`が`channels_last`，または`channels_first`ではない場合.

---

conv3d

```
conv3d(x, kernel, strides=(1, 1, 1), padding='valid', data_format=None, dilation_rate=(1, 1,
```

3次元の畳み込み.

引数

- **x**: テンソル，または変数.
- **kernel**: カーネルのテンソル.
- **strides**: ストライドのタプル.
- **padding**: 文字列． `same`，または`valid`.
- **data_format**: 文字列． `channels_last`，または`channels_first`のどちらか．入力/カーネル/出力でTheanoもしくはTensorFlowのデータ形式を利用するかどうか.
- **dilation_rate**: 3つの整数からなるタプル

戻り値

3次元の畳み込みの結果からなるテンソル.

Raises

- **ValueError**: `data_format`が`channels_last`，または`channels_first`ではない場合.

---

pool2d

```
pool2d(x, pool_size, strides=(1, 1), padding='valid', data_format=None, pool_mode='max')
```

2次元のプーリング.

引数

- **x**: テンソル，または変数.
- **pool_size**: 2つの整数からなるタプル.
- **strides**: 2つの整数からなるタプル.

- **padding**: 文字列. `same` , または `valid` .
- **data_format**: 文字列. `channels_last` , または `channels_first` のどちらか.
- **pool_mode**: `max` , `avg` のどちらか.

戻り値

2次元のプーリングの結果からなるテンソル.

Raises

- **ValueError**: `data_format` が `channels_last` , または `channels_first` ではない場合.
- **ValueError**: `pool_mode` が `max` , または `avg` ではない場合.

---

pool3d

```
pool3d(x, pool_size, strides=(1, 1, 1), padding='valid', data_format=None, pool_mode='max')
```

2次元のプーリング.

引数

- **x**: テンソル, または変数.
- **pool_size**: 3つの整数からなるタプル.
- **strides**: 3つの整数からなるタプル.
- **padding**: 文字列. `same` , または `valid` .
- **data_format**: 文字列. `channels_last` , または `channels_first` のどちらか.
- **pool_mode**: `max` , `avg` のどちらか.

戻り値

3次元のプーリングの結果からなるテンソル.

Raises

- **ValueError**: `data_format` が `channels_last` , または `channels_first` ではない場合.
- **ValueError**: `pool_mode` が `max` , または `avg` ではない場合.

---

bias_add

```
bias_add(x, bias, data_format=None)
```

テンソルにバイアスベクトルを付加します.

引数

- **x**: テンソル，または変数．
- **bias**: 付加するバイアスを表すテンソル．
- **data_format**: 3，4，5階テンソルに対するデータの形式: "channels_last"，または"channels_first"のどちらか．

戻り値

出力テンソル．

Raises

- **ValueError**: 以下の2つの場合の一方：
  1. 不正な `data_format` が与えられた場合．
  2. 不正なbiasのshape．biasはベクトルかndim(x) - 1のテンソルにすべきです．

---

random_normal

```
random_normal(shape, mean=0.0, stddev=1.0, dtype=None, seed=None)
```

ガウス分布の値を持つテンソルを返します．

引数

- **shape**: 整数のタプル．作成するテンソルのshape．
- **mean**: 浮動小数点数．サンプリングするためのガウス分布の平均．
- **stddev**: 浮動小数点数．サンプリングするためのガウス分布の標準偏差．
- **dtype**: 文字列．返されるテンソルのデータ型．
- **seed**: 整数．ランダムシード．

戻り値

テンソル．

---

random_uniform

```
random_uniform(shape, minval=0.0, maxval=1.0, dtype=None, seed=None)
```

一様分布の値を持つテンソルを返します．

引数

- **shape**: 整数のタプル．作成するテンソルのshape．

- **minval**: 浮動小数点数．サンプリングするための一様分布の下限．
- **maxval**: 浮動小数点数．サンプリングするための一様分布の上限．
- **dtype**: 文字列．返されるテンソルのデータ型．
- **seed**: 整数．ランダムシード．

戻り値

テンソル．

---

random_binomial

```
random_binomial(shape, p=0.0, dtype=None, seed=None)
```

二項分布の値を持つテンソルを返します．

引数

- **shape**: 整数のタプル．作成するテンソルのshape．
- **p**: 浮動小数点数． `0. <= p <= 1` ，二項分布の確率．
- **dtype**: 文字列．返されるテンソルのデータ型．
- **seed**: 整数．ランダムシード．

戻り値

テンソル．

---

truncated_normal

```
truncated_normal(shape, mean=0.0, stddev=1.0, dtype=None, seed=None)
```

切断ガウス分布の値を持つテンソルを返します．

生成された値は，指定された平均値と標準偏差を持つガウス分布に従いますが，平均値から2の標準偏差を超える値が削除され，再選択されます。

引数

- **shape**: 整数のタプル．作成するテンソルのshape．
- **mean**: 浮動小数点数．値の平均．
- **stddev**: 浮動小数点数．値の標準偏差．
- **dtype**: 文字列．返されるテンソルのデータ型．
- **seed**: 整数．ランダムシード．

戻り値

テンソル.

---

ctc_label_dense_to_sparse

```
ctc_label_dense_to_sparse(labels, label_lengths)
```

CTCのラベルを密からスパースなものに変換します.

引数

- **labels**: 密なCTCのラベル.
- **label_length**: ラベルの長さ.

戻り値

ラベルにおけるスパース表現からなるテンソル.

---

ctc_batch_cost

```
ctc_batch_cost(y_true, y_pred, input_length, label_length)
```

各バッチ要素に対してCTCのlossアルゴリズムを実行.

引数

- **y_true**: 真のラベルを含むテンソル `(samples, max_string_length)` .
- **y_pred**: 予測値かsoftmaxの出力を含むテンソル `(samples, time_steps, num_categories)` .
- **input_length**: `y_pred` の各バッチの系列長を含むテンソル `(samples,1)` .
- **label_length**: `y_true` の各バッチの系列長を含むテンソル `(samples,1)` .

戻り値

各要素のCTCの損失値を含んだshape(samples, 1)のテンソル.

---

ctc_decode

```
ctc_decode(y_pred, input_length, greedy=True, beam_width=100, top_paths=1)
```

softmaxの出力をデコードします.

（最適な探索として知られる）貪欲法かconstrained dictionary searchを使います .

引数

- **y_pred**: 予測値かsoftmaxの出力を含むテンソル `(samples, time_steps, num_categories)`.
- **input_length**: y_predの各バッチの系列長を含むテンソル `(samples,1)`.
- **greedy**: `true` なら高速な最適パス探索を行います．このとき，辞書を使わない
- **beam_width**: `greedy` が `False` の場合，この幅を使ったビームサーチを行います．
- **top_paths**: `greedy` が `False` の場合，最も辿る可能性の高いパスがどれだけあるか返されます．

戻り値

- **Tuple**:
- **List**: `greedy` が `true` の場合，デコードされたシーケンスを含む1つの要素のリストが返されます． `greedy` が `false` の場合，最も辿る可能性の高いデコードされたシーケンスを返します．
- **Important**: 空白のラベルは `-1` を返されます．デコードされたシーケンスの対数確率を含むテンソル `(top_paths, )` です．

---

map_fn

```
map_fn(fn, elems, name=None, dtype=None)
```

関数fnをelemsの要素全てに対して当てはめ，その出力を返します．

引数

- **fn**: elemsの各要素に対して呼び出されるCallable．
- **elems**: テンソル．
- **name**: グラフの中のmapのノードに対する文字列の名前．
- **dtype**: 出力のデータ型．

戻り値

データ型 `dtype` を持つテンソル．

---

foldl

```
foldl(fn, elems, initializer=None, name=None)
```

fnを使って左から右にelemsの要素を結合させることでelemsを縮約します．

引数

- **fn**: elemsの各要素に対して呼び出されるCallable．例えば， `lambda acc, x: acc + x`
- **elems**: テンソル．
- **initializer**: 使用される最初の値．(Noneの場合は `elems[0]` を指す)
- **name**: グラフの中のfoldlのノードに対する文字列の名前．

戻り値

`initializer`の同じ型とshapeを持つテンソル.

---

foldr

```
foldr(fn, elems, initializer=None, name=None)
```

fnを使って右から左にelemsの要素を結合させることでelemsを縮約します.

引数

- **fn**: elemsの各要素に対して呼び出されるCallable. 例えば, `lambda acc, x: acc + x`
- **elems**: テンソル.
- **initializer**: 使用される最初の値. (Noneの場合は`elems[-1]`を指す)
- **name**: グラフの中のfoldrのノードに対する文字列の名前.

戻り値

`initializer`の同じ型とshapeを持つテンソル.

---

local_conv1d

```
local_conv1d(inputs, kernel, kernel_size, strides, data_format=None)
```

重みを共有しない1次元畳み込みの適用.

引数

- **inputs**: (batch_size, steps, input_dim)のshapeをもつ3階テンソル
- **kernel**: (output_length, feature_dim, filters)のshapeをもつ畳み込みのため共有なしの重み
- **kernel_size**: 1次元の畳み込みにおけるwondowの長さを指定する整数1つをもつタプル
- **strides**: 畳み込みのstrideの長さを指定する整数1つをもつタプル
- **data_format**: channels_first か channels_last のデータフォーマット

戻り値

重みを共有しない1次元の畳み込みを適用した (batch_size, output_lenght, filters) のshapeをもつテンソル

Raises

- **ValueError**: `data_format`が`channels_last`か`channels_first`でないとき.

local_conv2d

```
local_conv2d(inputs, kernel, kernel_size, strides, output_shape, data_format=None)
```

重みを共有しない2次元畳み込みの適用.

引数

- **inputs**: 4階テンソル: data_format='channels_first'なら (batch_size, filters, new_rows, new_cols), data_format='channels_last'なら (batch_size, new_rows, new_cols, filters)
- **kernel**: (output_items, feature_dim, filters)のshapeをもつ畳み込みのため共有なしの重み
- **kernel_size**: 2次元の畳み込みにおけるwondowの幅と高さを指定する整数2つをもつタプル
- **strides**: 幅と高さにそった畳み込みのstrideを指定する整数2つをもつタプル
- **output_shape**: (output_row, output_col) のタプル
- **data_format**: channels_first か channels_last のデータフォーマット

戻り値

4階テンソル: data_format='channels_first'なら(batch_size, filters, new_rows, new_cols)のshapeの4階テンソル，data_format='channels_last'なら(batch_size, new_rows, new_cols, filters)のshapeの4階テンソル.

Raises

- **ValueError**: `data_format` が `channels_last` か `channels_first` でないとき.