Dense

[source]

```
keras.layers.Dense(units, activation=None, use_bias=True, kernel_initializer='glorot_uniform',
```

Just your regular densely-connected NN layer.

`Dense` implements the operation: `output = activation(dot(input, kernel) + bias)` where `activation` is the element-wise activation function passed as the `activation` argument, `kernel` is a weights matrix created by the layer, and `bias` is a bias vector created by the layer (only applicable if `use_bias` is `True`).

Note: if the input to the layer has a rank greater than 2, then it is flattened prior to the initial dot product with `kernel`.

Example

```
# as first layer in a sequential model:
model = Sequential()
model.add(Dense(32, input_shape=(16,)))
# now the model will take as input arrays of shape (*, 16)
# and output arrays of shape (*, 32)

# after the first layer, you don't need to specify
# the size of the input anymore:
model.add(Dense(32))
```

Arguments

- **units**: Positive integer, dimensionality of the output space.
- **activation**: Activation function to use (see activations). If you don't specify anything, no activation is applied (ie. "linear" activation: `a(x) = x`).
- **use_bias**: Boolean, whether the layer uses a bias vector.
- **kernel_initializer**: Initializer for the `kernel` weights matrix (see initializers).
- **bias_initializer**: Initializer for the bias vector (see initializers).
- **kernel_regularizer**: Regularizer function applied to the `kernel` weights matrix (see regularizer).
- **bias_regularizer**: Regularizer function applied to the bias vector (see regularizer).
- **activity_regularizer**: Regularizer function applied to the output of the layer (its "activation"). (see regularizer).
- **kernel_constraint**: Constraint function applied to the `kernel` weights matrix (see constraints).
- **bias_constraint**: Constraint function applied to the bias vector (see constraints).

Input shape

nD tensor with shape: `(batch_size, ..., input_dim)`. The most common situation would be a 2D input with shape `(batch_size, input_dim)`.

Output shape

nD tensor with shape: `(batch_size, ..., units)`. For instance, for a 2D input with shape `(batch_size, input_dim)`, the output would have shape `(batch_size, units)`.

Activation

[source]

```
keras.layers.Activation(activation)
```

Applies an activation function to an output.

Arguments

- **activation**: name of activation function to use (see: activations), or alternatively, a Theano or TensorFlow operation.

Input shape

Arbitrary. Use the keyword argument `input_shape` (tuple of integers, does not include the samples axis) when using this layer as the first layer in a model.

Output shape

Same shape as input.

Dropout

[source]

```
keras.layers.Dropout(rate, noise_shape=None, seed=None)
```

Applies Dropout to the input.

Dropout consists in randomly setting a fraction `rate` of input units to 0 at each update during training time, which helps prevent overfitting.

Arguments

- **rate**: float between 0 and 1. Fraction of the input units to drop.
- **noise_shape**: 1D integer tensor representing the shape of the binary dropout mask that will be multiplied with the input. For instance, if your inputs have shape `(batch_size, timesteps, features)` and you want the dropout mask to be the same for all timesteps, you can use `noise_shape=(batch_size, 1, features)`.
- **seed**: A Python integer to use as random seed.

References

- Dropout: A Simple Way to Prevent Neural Networks from Overfitting

---

Flatten

[source]

```
keras.layers.Flatten(data_format=None)
```

Flattens the input. Does not affect the batch size.

Arguments

- **data_format**: A string, one of `channels_last` (default) or `channels_first`. The ordering of the dimensions in the inputs. The purpose of this argument is to preserve weight ordering when switching a model from one data format to another. `channels_last` corresponds to inputs with shape `(batch, ..., channels)` while `channels_first` corresponds to inputs with shape `(batch, channels, ...)`. It defaults to the `image_data_format` value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be "channels_last".

Example

```
model = Sequential()
model.add(Conv2D(64, 3, 3,
                 border_mode='same',
                 input_shape=(3, 32, 32)))
# now: model.output_shape == (None, 64, 32, 32)

model.add(Flatten())
# now: model.output_shape == (None, 65536)
```

Input

[source]

```
keras.engine.input_layer.Input()
```

`Input()` is used to instantiate a Keras tensor.

A Keras tensor is a tensor object from the underlying backend (Theano, TensorFlow or CNTK), which we augment with certain attributes that allow us to build a Keras model just by knowing the inputs and outputs of the model.

For instance, if a, b and c are Keras tensors, it becomes possible to do: `model = Model(input=[a, b], output=c)`

The added Keras attributes are: `_keras_shape`: Integer shape tuple propagated via Keras-side shape inference. `_keras_history`: Last layer applied to the tensor. the entire layer graph is retrievable from that layer, recursively.

Arguments

- **shape**: A shape tuple (integer), not including the batch size. For instance, `shape=(32,)` indicates that the expected input will be batches of 32-dimensional vectors.
- **batch_shape**: A shape tuple (integer), including the batch size. For instance, `batch_shape=(10, 32)` indicates that the expected input will be batches of 10 32-dimensional vectors. `batch_shape=(None, 32)` indicates batches of an arbitrary number of 32-dimensional vectors.
- **name**: An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided.
- **dtype**: The data type expected by the input, as a string (`float32`, `float64`, `int32`...)
- **sparse**: A boolean specifying whether the placeholder to be created is sparse.
- **tensor**: Optional existing tensor to wrap into the `Input` layer. If set, the layer will not create a placeholder tensor.

Returns

A tensor.

Example

```
# this is a logistic regression in Keras
x = Input(shape=(32,))
y = Dense(16, activation='softmax')(x)
model = Model(x, y)
```

Reshape

[source]

```
keras.layers.Reshape(target_shape)
```

Reshapes an output to a certain shape.

Arguments

- **target_shape**: target shape. Tuple of integers. Does not include the batch axis.

Input shape

Arbitrary, although all dimensions in the input shaped must be fixed. Use the keyword argument `input_shape` (tuple of integers, does not include the batch axis) when using this layer as the first layer in a model.

Output shape

`(batch_size,) + target_shape`

Example

```
# as first layer in a Sequential model
model = Sequential()
model.add(Reshape((3, 4), input_shape=(12,)))
# now: model.output_shape == (None, 3, 4)
# note: `None` is the batch dimension

# as intermediate layer in a Sequential model
model.add(Reshape((6, 2)))
# now: model.output_shape == (None, 6, 2)

# also supports shape inference using `-1` as dimension
model.add(Reshape((-1, 2, 2)))
# now: model.output_shape == (None, 3, 2, 2)
```

Permute

[source]

```
keras.layers.Permute(dims)
```

Permutes the dimensions of the input according to a given pattern.

Useful for e.g. connecting RNNs and convnets together.

Example

```
model = Sequential()
model.add(Permute((2, 1), input_shape=(10, 64)))
# now: model.output_shape == (None, 64, 10)
# note: `None` is the batch dimension
```

Arguments

- **dims**: Tuple of integers. Permutation pattern, does not include the samples dimension. Indexing starts at 1. For instance, `(2, 1)` permutes the first and second dimension of the input.

Input shape

Arbitrary. Use the keyword argument `input_shape` (tuple of integers, does not include the samples axis) when using this layer as the first layer in a model.

Output shape

Same as the input shape, but with the dimensions re-ordered according to the specified pattern.

RepeatVector

[source]

```
keras.layers.RepeatVector(n)
```

Repeats the input n times.

Example

```
model = Sequential()
model.add(Dense(32, input_dim=32))
# now: model.output_shape == (None, 32)
# note: `None` is the batch dimension

model.add(RepeatVector(3))
# now: model.output_shape == (None, 3, 32)
```

Arguments

- **n**: integer, repetition factor.

Input shape

2D tensor of shape `(num_samples, features)`.

Output shape

3D tensor of shape `(num_samples, n, features)`.

Lambda

[source]

```
keras.layers.Lambda(function, output_shape=None, mask=None, arguments=None)
```

Wraps arbitrary expression as a `Layer` object.

Examples

```
# add a x -> x^2 layer
model.add(Lambda(lambda x: x ** 2))
```

```
# add a layer that returns the concatenation
# of the positive part of the input and
# the opposite of the negative part

def antirectifier(x):
    x -= K.mean(x, axis=1, keepdims=True)
    x = K.l2_normalize(x, axis=1)
    pos = K.relu(x)
    neg = K.relu(-x)
    return K.concatenate([pos, neg], axis=1)

def antirectifier_output_shape(input_shape):
    shape = list(input_shape)
    assert len(shape) == 2  # only valid for 2D tensors
    shape[-1] *= 2
    return tuple(shape)

model.add(Lambda(antirectifier,
                 output_shape=antirectifier_output_shape))
```

Arguments

- **function**: The function to be evaluated. Takes input tensor as first argument.
- **output_shape**: Expected output shape from function. Only relevant when using Theano. Can be a tuple or function. If a tuple, it only specifies the first dimension onward; sample dimension is assumed either the same as the input: `output_shape = (input_shape[0], ) + output_shape` or, the input is `None` and the sample dimension is also `None`: `output_shape = (None, ) + output_shape` If a function, it specifies the entire shape as a function of the input shape: `output_shape = f(input_shape)`
- **arguments**: optional dictionary of keyword arguments to be passed to the function.

Input shape

Arbitrary. Use the keyword argument input_shape (tuple of integers, does not include the samples axis) when using this layer as the first layer in a model.

Output shape

Specified by `output_shape` argument (or auto-inferred when using TensorFlow).

ActivityRegularization

[source]

```
keras.layers.ActivityRegularization(l1=0.0, l2=0.0)
```

Layer that applies an update to the cost function based input activity.

Arguments

- **l1**: L1 regularization factor (positive float).
- **l2**: L2 regularization factor (positive float).

Input shape

Arbitrary. Use the keyword argument `input_shape` (tuple of integers, does not include the samples axis) when using this layer as the first layer in a model.

Output shape

Same shape as input.

Masking

[source]

```
keras.layers.Masking(mask_value=0.0)
```

Masks a sequence by using a mask value to skip timesteps.

For each timestep in the input tensor (dimension #1 in the tensor), if all values in the input tensor at that timestep are equal to `mask_value`, then the timestep will be masked (skipped) in all downstream layers (as long as they support masking).

If any downstream layer does not support masking yet receives such an input mask, an exception will be raised.

Example

Consider a Numpy data array `x` of shape `(samples, timesteps, features)`, to be fed to an LSTM layer. You want to mask timestep #3 and #5 because you lack data for these timesteps. You can:

- set `x[:, 3, :] = 0.` and `x[:, 5, :] = 0.`
- insert a `Masking` layer with `mask_value=0.` before the LSTM layer:

```
model = Sequential()
model.add(Masking(mask_value=0., input_shape=(timesteps, features)))
model.add(LSTM(32))
```

SpatialDropout1D

[source]

```
keras.layers.SpatialDropout1D(rate)
```

Spatial 1D version of Dropout.

This version performs the same function as Dropout, however it drops entire 1D feature maps instead of individual elements. If adjacent frames within feature maps are strongly correlated (as is normally the case in early convolution layers) then regular dropout will not regularize the activations and will otherwise just result in an effective learning rate decrease. In this case, SpatialDropout1D will help promote independence between feature maps and should be used instead.

Arguments

- **rate**: float between 0 and 1. Fraction of the input units to drop.

Input shape

3D tensor with shape: `(samples, timesteps, channels)`

Output shape

Same as input

References

- Efficient Object Localization Using Convolutional Networks

SpatialDropout2D

[source]

```
keras.layers.SpatialDropout2D(rate, data_format=None)
```

Spatial 2D version of Dropout.

This version performs the same function as Dropout, however it drops entire 2D feature maps instead of individual elements. If adjacent pixels within feature maps are strongly correlated (as is normally the case in early convolution layers) then regular dropout will not regularize the activations and will otherwise just result in an effective learning rate decrease. In this case, SpatialDropout2D will help promote independence between feature maps and should be used instead.

Arguments

- **rate**: float between 0 and 1. Fraction of the input units to drop.
- **data_format**: 'channels_first' or 'channels_last'. In 'channels_first' mode, the channels dimension (the depth) is at index 1, in 'channels_last' mode is it at index 3. It defaults to the `image_data_format` value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be "channels_last".

Input shape

4D tensor with shape: `(samples, channels, rows, cols)` if data_format='channels_first' or 4D tensor with shape: `(samples, rows, cols, channels)` if data_format='channels_last'.

Output shape

Same as input

References

- Efficient Object Localization Using Convolutional Networks

SpatialDropout3D

[source]

```
keras.layers.SpatialDropout3D(rate, data_format=None)
```

Spatial 3D version of Dropout.

This version performs the same function as Dropout, however it drops entire 3D feature maps instead of individual elements. If adjacent voxels within feature maps are strongly correlated (as is normally the case in early convolution layers) then regular dropout will not regularize the activations and will otherwise just result in an effective learning rate decrease. In this case, SpatialDropout3D will help promote independence between feature maps and should be used instead.

Arguments

- **rate**: float between 0 and 1. Fraction of the input units to drop.
- **data_format**: 'channels_first' or 'channels_last'. In 'channels_first' mode, the channels dimension (the depth) is at index 1, in 'channels_last' mode is it at index 4. It defaults to the `image_data_format` value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be "channels_last".

Input shape

5D tensor with shape: `(samples, channels, dim1, dim2, dim3)` if data_format='channels_first' or 5D tensor with shape: `(samples, dim1, dim2, dim3, channels)` if data_format='channels_last'.

Output shape

Same as input

References

- Efficient Object Localization Using Convolutional Networks