Dense

[source]

```
keras.layers.core.Dense(output_dim, init='glorot_uniform', activation='linear', weights=None, W
```

Just your regular fully connected NN layer.

Example

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

# this is equivalent to the above:
model = Sequential()
model.add(Dense(32, input_shape=(16,)))

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

Arguments

- **output_dim**: int > 0.
- **init**: name of initialization function for the weights of the layer (see initializations), or alternatively, Theano function to use for weights initialization. This parameter is only relevant if you don't pass a `weights` argument.
- **activation**: name of activation function to use (see activations), or alternatively, elementwise Theano function. If you don't specify anything, no activation is applied (ie. "linear" activation: a(x) = x).
- **weights**: list of Numpy arrays to set as initial weights. The list should have 2 elements, of shape `(input_dim, output_dim)` and (output_dim,) for weights and biases respectively.
- **W_regularizer**: instance of WeightRegularizer (eg. L1 or L2 regularization), applied to the main weights matrix.
- **b_regularizer**: instance of WeightRegularizer, applied to the bias.
- **activity_regularizer**: instance of ActivityRegularizer, applied to the network output.
- **W_constraint**: instance of the constraints module (eg. maxnorm, nonneg), applied to the main weights matrix.
- **b_constraint**: instance of the constraints module, applied to the bias.
- **bias**: whether to include a bias (i.e. make the layer affine rather than linear).
- **input_dim**: dimensionality of the input (integer). This argument (or alternatively, the keyword argument `input_shape`) is required when using this layer as the first layer in a model.

Input shape

2D tensor with shape: `(nb_samples, input_dim)`.

Output shape

2D tensor with shape: `(nb_samples, output_dim)`.

Activation

[source]

```
keras.layers.core.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.core.Dropout(p)
```

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

Arguments

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

References

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

---

Flatten

[source]

```
keras.layers.core.Flatten()
```

Flattens the input. Does not affect the batch size.

Example

```
model = Sequential()
model.add(Convolution2D(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)
```

Reshape

[source]

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

Reshapes an output to a certain shape.

Arguments

- **target_shape**: target shape. Tuple of integers, does not include the samples dimension (batch size).

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 samples 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)
```

Permute

[source]

```
keras.layers.core.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.core.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 `(nb_samples, features)`.

Output shape

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

Merge

[source]

```
keras.engine.topology.Merge(layers=None, mode='sum', concat_axis=-1, dot_axes=-1, output_shape=
```

A `Merge` layer can be used to merge a list of tensors into a single tensor, following some merge `mode`.

Example usage

```
model1 = Sequential()
model1.add(Dense(32))

model2 = Sequential()
model2.add(Dense(32))

merged_model = Sequential()
merged_model.add(Merge([model1, model2], mode='concat', concat_axis=1)
- ____TODO__: would this actually work? it needs to.__

# achieve this with get_source_inputs in Sequential.
```

Arguments

- **layers**: can be a list of Keras tensors or a list of layer instances. Must be more than one layer/tensor.
- **mode**: string or lambda/function. If string, must be one

    - **of**: 'sum', 'mul', 'concat', 'ave', 'cos', 'dot', 'max'. If lambda/function, it should take as input a list of tensors and return a single tensor.
- **concat_axis**: integer, axis to use in mode `concat`.
- **dot_axes**: integer or tuple of integers, axes to use in mode `dot`.
- **output_shape**: either a shape tuple (tuple of integers), or a lambda/function to compute `output_shape` (only if merge mode is a lambda/function). If the argument is a tuple, it should be expected output shape, *not* including the batch size (same convention as the `input_shape` argument in layers). If the argument is callable, it should take as input a list of shape tuples
    - **(1**:1 mapping to input tensors) and return a single shape tuple, including the batch size (same convention as the `get_output_shape_for` method of layers).
- **node_indices**: optional list of integers containing the output node index for each input layer (in case some input layers have multiple output nodes). will default to an array of 0s if not provided.
- **tensor_indices**: optional list of indices of output tensors to consider for merging (in case some input layer node returns multiple tensors).
- **output_mask**: mask or lambda/function to compute the output mask (only if merge mode is a lambda/function). If the latter case, it should take as input a list of masks and return a single mask.

Lambda

[source]

```
keras.layers.core.Lambda(function, output_shape=None, arguments={})
```

Used for evaluating an arbitrary Theano / TensorFlow expression on the output of the previous layer.

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 one argument: the output of previous layer
- **output_shape**: Expected output shape from function. 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.

ActivityRegularization

[source]

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

Layer that passes through its input unchanged, but applies an update to the cost function based on the 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.core.Masking(mask_value=0.0)
```

Masks an input sequence by using a mask value to identify timesteps to be skipped.

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 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 a 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))
```

Highway

[source]

```
keras.layers.core.Highway(init='glorot_uniform', transform_bias=-2, activation='linear', weight
```

Densely connected highway network, a natural extension of LSTMs to feedforward networks.

Arguments

- **init**: name of initialization function for the weights of the layer (see initializations), or alternatively, Theano function to use for weights initialization. This parameter is only relevant if you don't pass a `weights` argument.
- **transform_bias**: value for the bias to take on initially (default -2)
- **activation**: name of activation function to use (see activations), or alternatively, elementwise Theano function. If you don't specify anything, no activation is applied (ie. "linear" activation: a(x) = x).
- **weights**: list of Numpy arrays to set as initial weights. The list should have 2 elements, of shape `(input_dim, output_dim)` and (output_dim,) for weights and biases respectively.

- **W_regularizer**: instance of WeightRegularizer (eg. L1 or L2 regularization), applied to the main weights matrix.
- **b_regularizer**: instance of WeightRegularizer, applied to the bias.
- **activity_regularizer**: instance of ActivityRegularizer, applied to the network output.
- **W_constraint**: instance of the constraints module (eg. maxnorm, nonneg), applied to the main weights matrix.
- **b_constraint**: instance of the constraints module, applied to the bias.
- **bias**: whether to include a bias (i.e. make the layer affine rather than linear).
- **input_dim**: dimensionality of the input (integer). This argument (or alternatively, the keyword argument `input_shape`) is required when using this layer as the first layer in a model.

Input shape

2D tensor with shape: `(nb_samples, input_dim)`.

Output shape

2D tensor with shape: `(nb_samples, input_dim)`.

References

- Highway Networks

MaxoutDense

[source]

```
keras.layers.core.MaxoutDense(output_dim, nb_feature=4, init='glorot_uniform', weights=None, W_
```

A dense maxout layer.

A `MaxoutDense` layer takes the element-wise maximum of `nb_feature` `Dense(input_dim, output_dim)` linear layers. This allows the layer to learn a convex, piecewise linear activation function over the inputs.

Note that this is a *linear* layer; if you wish to apply activation function (you shouldn't need to --they are universal function approximators), an `Activation` layer must be added after.

Arguments

- **output_dim**: int > 0.
- **nb_feature**: number of Dense layers to use internally.

- **init**: name of initialization function for the weights of the layer (see initializations), or alternatively, Theano function to use for weights initialization. This parameter is only relevant if you don't pass a `weights` argument.
- **weights**: list of Numpy arrays to set as initial weights. The list should have 2 elements, of shape `(input_dim, output_dim)` and (output_dim,) for weights and biases respectively.
- **W_regularizer**: instance of WeightRegularizer (eg. L1 or L2 regularization), applied to the main weights matrix.
- **b_regularizer**: instance of WeightRegularizer, applied to the bias.
- **activity_regularizer**: instance of ActivityRegularizer, applied to the network output.
- **W_constraint**: instance of the constraints module (eg. maxnorm, nonneg), applied to the main weights matrix.
- **b_constraint**: instance of the constraints module, applied to the bias.
- **bias**: whether to include a bias (i.e. make the layer affine rather than linear).
- **input_dim**: dimensionality of the input (integer). This argument (or alternatively, the keyword argument `input_shape`) is required when using this layer as the first layer in a model.

Input shape

2D tensor with shape: `(nb_samples, input_dim)`.

Output shape

2D tensor with shape: `(nb_samples, output_dim)`.

References

- Maxout Networks

TimeDistributedDense

[source]

```
keras.layers.core.TimeDistributedDense(output_dim, init='glorot_uniform', activation='linear',
```

Apply a same Dense layer for each dimension[1] (time_dimension) input. Especially useful after a recurrent network with 'return_sequence=True'.

- **Note**: this layer is deprecated, prefer using the `TimeDistributed` wrapper:

```
model.add(TimeDistributed(Dense(32)))
```

Input shape

3D tensor with shape `(nb_sample, time_dimension, input_dim)`.

Output shape

3D tensor with shape `(nb_sample, time_dimension, output_dim)`.

Arguments

- **output_dim**: int > 0.
- **init**: name of initialization function for the weights of the layer (see initializations), or alternatively, Theano function to use for weights initialization. This parameter is only relevant if you don't pass a `weights` argument.
- **activation**: name of activation function to use (see activations), or alternatively, elementwise Theano function. If you don't specify anything, no activation is applied (ie. "linear" activation: a(x) = x).
- **weights**: list of Numpy arrays to set as initial weights. The list should have 2 elements, of shape `(input_dim, output_dim)` and (output_dim,) for weights and biases respectively.
- **W_regularizer**: instance of WeightRegularizer (eg. L1 or L2 regularization), applied to the main weights matrix.
- **b_regularizer**: instance of WeightRegularizer, applied to the bias.
- **activity_regularizer**: instance of ActivityRegularizer, applied to the network output.
- **W_constraint**: instance of the constraints module (eg. maxnorm, nonneg), applied to the main weights matrix.
- **b_constraint**: instance of the constraints module, applied to the bias.
- **bias**: whether to include a bias (i.e. make the layer affine rather than linear).
- **input_dim**: dimensionality of the input (integer). This argument (or alternatively, the keyword argument `input_shape`) is required when using this layer as the first layer in a model.
- **input_length**: length of inputs sequences (integer, or None for variable-length sequences).