tf.keras.layers.Dropout

Class `Dropout`

Inherits From: `Dropout`, `Layer`

Defined in `tensorflow/python/keras/_impl/keras/layers/core.py`.

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.

Properties

`activity_regularizer`

Optional regularizer function for the output of this layer.

`dtype`

`graph`

`input`

Retrieves the input tensor(s) of a layer.

Only applicable if the layer has exactly one input, i.e. if it is connected to one incoming layer.

Returns:

Input tensor or list of input tensors.

Raises:

- `AttributeError`: if the layer is connected to more than one incoming layers.

Raises:

- `RuntimeError`: If called in Eager mode.
- `AttributeError`: If no inbound nodes are found.

`input_mask`

Retrieves the input mask tensor(s) of a layer.

Only applicable if the layer has exactly one inbound node, i.e. if it is connected to one incoming layer.

Returns:

Input mask tensor (potentially None) or list of input mask tensors.

Raises:

- `AttributeError`: if the layer is connected to more than one incoming layers.

`input_shape`

Retrieves the input shape(s) of a layer.

Only applicable if the layer has exactly one input, i.e. if it is connected to one incoming layer, or if all inputs have the same shape.

Returns:

Input shape, as an integer shape tuple (or list of shape tuples, one tuple per input tensor).

Raises:

- `AttributeError`: if the layer has no defined input_shape.
- `RuntimeError`: if called in Eager mode.

`losses`

`name`

`non_trainable_variables`

`non_trainable_weights`

`output`

Retrieves the output tensor(s) of a layer.

Only applicable if the layer has exactly one output, i.e. if it is connected to one incoming layer.

Returns:

Output tensor or list of output tensors.

Raises:

- `AttributeError`: if the layer is connected to more than one incoming layers.
- `RuntimeError`: if called in Eager mode.

output_mask

Retrieves the output mask tensor(s) of a layer.

Only applicable if the layer has exactly one inbound node, i.e. if it is connected to one incoming layer.

Returns:

Output mask tensor (potentially None) or list of output mask tensors.

Raises:

- `AttributeError`: if the layer is connected to more than one incoming layers.

output_shape

Retrieves the output shape(s) of a layer.

Only applicable if the layer has one output, or if all outputs have the same shape.

Returns:

Output shape, as an integer shape tuple (or list of shape tuples, one tuple per output tensor).

Raises:

- `AttributeError`: if the layer has no defined output shape.
- `RuntimeError`: if called in Eager mode.

scope_name

trainable_variables

trainable_weights

updates

variables

Returns the list of all layer variables/weights.

Returns:

A list of variables.

`weights`

Returns the list of all layer variables/weights.

Returns:

A list of variables.

Methods

`__init__`

```
__init__(
    rate,
    noise_shape=None,
    seed=None,
    **kwargs
)
```

`__call__`

```
__call__(
    inputs,
    **kwargs
)
```

Wrapper around self.call(), for handling internal references.

If a Keras tensor is passed: - We call self._add_inbound_node(). - If necessary, we `build` the layer to match the shape of the input(s). - We update the _keras_history of the output tensor(s) with the current layer. This is done as part of _add_inbound_node().

Arguments:

- `inputs`: Can be a tensor or list/tuple of tensors.
- `**kwargs`: Additional keyword arguments to be passed to `call()`.

Returns:

Output of the layer's `call` method.

Raises:

- `ValueError`: in case the layer is missing shape information for its `build` call.

__deepcopy__

```
__deepcopy__(memo)
```

add_loss

```
add_loss(
    losses,
    inputs=None
)
```

Add loss tensor(s), potentially dependent on layer inputs.

Some losses (for instance, activity regularization losses) may be dependent on the inputs passed when calling a layer. Hence, when reusing a same layer on different inputs `a` and `b`, some entries in `layer.losses` may be dependent on `a` and some on `b`. This method automatically keeps track of dependencies.

The `get_losses_for` method allows to retrieve the losses relevant to a specific set of inputs.

Arguments:

- `losses`: Loss tensor, or list/tuple of tensors.
- `inputs`: Optional input tensor(s) that the loss(es) depend on. Must match the `inputs` argument passed to the `__call__` method at the time the losses are created. If `None` is passed, the losses are assumed to be unconditional, and will apply across all dataflows of the layer (e.g. weight regularization losses).

Raises:

- `RuntimeError`: If called in Eager mode.

add_update

```
add_update(
    updates,
    inputs=None
)
```

Add update op(s), potentially dependent on layer inputs.

Weight updates (for instance, the updates of the moving mean and variance in a BatchNormalization layer) may be dependent on the inputs passed when calling a layer. Hence, when reusing a same layer on different inputs `a` and `b`, some entries in `layer.updates` may be dependent on `a` and some on `b`. This method automatically keeps track of dependencies.

The `get_updates_for` method allows to retrieve the updates relevant to a specific set of inputs.

This call is ignored in Eager mode.

Arguments:

- `updates`: Update op, or list/tuple of update ops.
- `inputs`: Optional input tensor(s) that the update(s) depend on. Must match the `inputs` argument passed to the `__call__` method at the time the updates are created. If `None` is passed, the updates are assumed to be unconditional, and will apply across all dataflows of the layer.

`add_variable`

```
add_variable(
    name,
    shape,
    dtype=None,
    initializer=None,
    regularizer=None,
    trainable=True,
    constraint=None
)
```

Adds a new variable to the layer, or gets an existing one; returns it.

Arguments:

- `name`: variable name.
- `shape`: variable shape.
- `dtype`: The type of the variable. Defaults to `self.dtype` or `float32`.
- `initializer`: initializer instance (callable).
- `regularizer`: regularizer instance (callable).
- `trainable`: whether the variable should be part of the layer's "trainable_variables" (e.g. variables, biases) or "non_trainable_variables" (e.g. BatchNorm mean, stddev).
- `constraint`: constraint instance (callable).

Returns:

The created variable.

Raises:

- `RuntimeError`: If called in Eager mode with regularizers.

`add_weight`

```
add_weight(
    name,
    shape,
    dtype=None,
    initializer=None,
    regularizer=None,
    trainable=True,
    constraint=None
)
```

Adds a weight variable to the layer.

Arguments:

- `name`: String, the name for the weight variable.
- `shape`: The shape tuple of the weight.
- `dtype`: The dtype of the weight.

- `initializer`: An Initializer instance (callable).
- `regularizer`: An optional Regularizer instance.
- `trainable`: A boolean, whether the weight should be trained via backprop or not (assuming that the layer itself is also trainable).
- `constraint`: An optional Constraint instance.

Returns:

The created weight variable.

`apply`

```
apply(
    inputs,
    *args,
    **kwargs
)
```

Apply the layer on a input.

This simply wraps **`self.__call__`**.

Arguments:

- `inputs`: Input tensor(s).
- `*args`: additional positional arguments to be passed to **`self.call`**.
- `**kwargs`: additional keyword arguments to be passed to **`self.call`**.

Returns:

Output tensor(s).

`build`

```
build(_)
```

Creates the variables of the layer.

`call`

```
call(
    inputs,
    training=None
)
```

`compute_mask`

```
compute_mask(
    inputs,
    mask=None
)
```

Computes an output mask tensor.

Arguments:

- `inputs`: Tensor or list of tensors.
- `mask`: Tensor or list of tensors.

Returns:

None or a tensor (or list of tensors, one per output tensor of the layer).

`count_params`

```
count_params()
```

Count the total number of scalars composing the weights.

Returns:

An integer count.

Raises:

- `ValueError`: if the layer isn't yet built (in which case its weights aren't yet defined).

`from_config`

```
from_config(
    cls,
    config
)
```

Creates a layer from its config.

This method is the reverse of **`get_config`**, capable of instantiating the same layer from the config dictionary. It does not handle layer connectivity (handled by Container), nor weights (handled by **`set_weights`**).

Arguments:

- `config`: A Python dictionary, typically the output of get_config.

Returns:

A layer instance.

`get_config`

```
get_config()
```

`get_input_at`

```
get_input_at(node_index)
```

Retrieves the input tensor(s) of a layer at a given node.

Arguments:

- `node_index`: Integer, index of the node from which to retrieve the attribute. E.g. `node_index=0` will correspond to the first time the layer was called.

Returns:

A tensor (or list of tensors if the layer has multiple inputs).

Raises:

- `RuntimeError`: If called in Eager mode.

`get_input_mask_at`

```
get_input_mask_at(node_index)
```

Retrieves the input mask tensor(s) of a layer at a given node.

Arguments:

- `node_index`: Integer, index of the node from which to retrieve the attribute. E.g. `node_index=0` will correspond to the first time the layer was called.

Returns:

A mask tensor (or list of tensors if the layer has multiple inputs).

`get_input_shape_at`

```
get_input_shape_at(node_index)
```

Retrieves the input shape(s) of a layer at a given node.

Arguments:

- `node_index`: Integer, index of the node from which to retrieve the attribute. E.g. `node_index=0` will correspond to the first time the layer was called.

Returns:

A shape tuple (or list of shape tuples if the layer has multiple inputs).

Raises:

- `RuntimeError`: If called in Eager mode.

`get_losses_for`

```
get_losses_for(inputs)
```

Retrieves losses relevant to a specific set of inputs.

Arguments:

- `inputs`: Input tensor or list/tuple of input tensors. Must match the `inputs` argument passed to the `__call__` method at the time the losses were created. If you pass `inputs=None`, unconditional losses are returned, such as weight regularization losses.

Returns:

List of loss tensors of the layer that depend on `inputs`.

Raises:

- `RuntimeError`: If called in Eager mode.

`get_output_at`

```
get_output_at(node_index)
```

Retrieves the output tensor(s) of a layer at a given node.

Arguments:

- `node_index`: Integer, index of the node from which to retrieve the attribute. E.g. `node_index=0` will correspond to the first time the layer was called.

Returns:

A tensor (or list of tensors if the layer has multiple outputs).

Raises:

- `RuntimeError`: If called in Eager mode.

`get_output_mask_at`

```
get_output_mask_at(node_index)
```

Retrieves the output mask tensor(s) of a layer at a given node.

Arguments:

- `node_index`: Integer, index of the node from which to retrieve the attribute. E.g. `node_index=0` will correspond to the first time the layer was called.

Returns:

A mask tensor (or list of tensors if the layer has multiple outputs).

`get_output_shape_at`

```
get_output_shape_at(node_index)
```

Retrieves the output shape(s) of a layer at a given node.

Arguments:

- `node_index`: Integer, index of the node from which to retrieve the attribute. E.g. `node_index=0` will correspond to the first time the layer was called.

Returns:

A shape tuple (or list of shape tuples if the layer has multiple outputs).

Raises:

- `RuntimeError`: If called in Eager mode.

`get_updates_for`

```
get_updates_for(inputs)
```

Retrieves updates relevant to a specific set of inputs.

Arguments:

- `inputs`: Input tensor or list/tuple of input tensors. Must match the `inputs` argument passed to the `__call__` method at the time the updates were created. If you pass `inputs=None`, unconditional updates are returned.

Returns:

List of update ops of the layer that depend on `inputs`.

Raises:

- `RuntimeError`: If called in Eager mode.

`get_weights`

```
get_weights()
```

Returns the current weights of the layer.

Returns:

Weights values as a list of numpy arrays.

`set_weights`

```
set_weights(weights)
```

Sets the weights of the layer, from Numpy arrays.

Arguments:

- `weights`: a list of Numpy arrays. The number of arrays and their shape must match number of the dimensions of the weights of the layer (i.e. it should match the output of `get_weights`).

Raises:

- `ValueError`: If the provided weights list does not match the layer's specifications.

Last updated November 2, 2017.