tf.contrib.rnn.DropoutWrapper

Contents

Class **`DropoutWrapper`**

Inherits From: `RNNCell`

Aliases:

- Class `tf.contrib.rnn.DropoutWrapper`
- Class `tf.nn.rnn_cell.DropoutWrapper`

Defined in `tensorflow/python/ops/rnn_cell_impl.py`.

See the guide: RNN and Cells (contrib) > Core RNN Cell wrappers (RNNCells that wrap other RNNCells)

Operator adding dropout to inputs and outputs of the given cell.

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_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_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.

`output_size`

`scope_name`

`state_size`

`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__(
    cell,
    input_keep_prob=1.0,
    output_keep_prob=1.0,
    state_keep_prob=1.0,
    variational_recurrent=False,
    input_size=None,
    dtype=None,
    seed=None,
    dropout_state_filter_visitor=None
)
```

Create a cell with added input, state, and/or output dropout.

If `variational_recurrent` is set to `True` (**NOT** the default behavior), then the same dropout mask is applied at every step, as described in:

Y. Gal, Z Ghahramani. "A Theoretically Grounded Application of Dropout in Recurrent Neural Networks". https://arxiv.org/abs/1512.05287

Otherwise a different dropout mask is applied at every time step.

Note, by default (unless a custom `dropout_state_filter` is provided), the memory state (`c` component of any `LSTMStateTuple`) passing through a `DropoutWrapper` is never modified. This behavior is described in the above article.

Args:

- `cell`: an RNNCell, a projection to output_size is added to it.
- `input_keep_prob`: unit Tensor or float between 0 and 1, input keep probability; if it is constant and 1, no input dropout will be added.
- `output_keep_prob`: unit Tensor or float between 0 and 1, output keep probability; if it is constant and 1, no output dropout will be added.
- `state_keep_prob`: unit Tensor or float between 0 and 1, output keep probability; if it is constant and 1, no output dropout will be added. State dropout is performed on the outgoing states of the cell. **Note** the state components to which dropout is applied when `state_keep_prob` is in `(0, 1)` are also determined by the argument `dropout_state_filter_visitor` (e.g. by default dropout is never applied to the `c` component of an `LSTMStateTuple`).
- `variational_recurrent`: Python bool. If `True`, then the same dropout pattern is applied across all time steps per run call. If this parameter is set, `input_size` **must** be provided.
- `input_size`: (optional) (possibly nested tuple of) `TensorShape` objects containing the depth(s) of the input tensors expected to be passed in to the `DropoutWrapper`. Required and used **iff** `variational_recurrent = True` and `input_keep_prob < 1`.
- `dtype`: (optional) The `dtype` of the input, state, and output tensors. Required and used **iff** `variational_recurrent = True`.
- `seed`: (optional) integer, the randomness seed.
- `dropout_state_filter_visitor`: (optional), default: (see below). Function that takes any hierarchical level of the state and returns a scalar or depth=1 structure of Python booleans describing which terms in the state should be dropped out. In addition, if the function returns `True`, dropout is applied across this sublevel. If the function returns `False`, dropout is not applied across this entire sublevel. Default behavior: perform dropout on all terms except the memory (`c`) state of `LSTMCellState` objects, and don't try to apply dropout to `TensorArray` objects: `def dropout_state_filter_visitor(s): if isinstance(s, LSTMCellState): # Never perform dropout on the c state. return LSTMCellState(c=False, h=True) elif isinstance(s, TensorArray): return False return True`

Raises:

- `TypeError`: if `cell` is not an `RNNCell`, or `keep_state_fn` is provided but not `callable`.
- `ValueError`: if any of the keep_probs are not between 0 and 1.

`__call__`

```
__call__(
    inputs,
    state,
    scope=None
)
```

Run the cell with the declared dropouts.

`__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.

`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(_)
```

`call`

```
call(
    inputs,
    **kwargs
)
```

The logic of the layer lives here.

Arguments:

- `inputs`: input tensor(s).
- `**kwargs`: additional keyword arguments.

Returns:

Output tensor(s).

`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).

`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_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_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.

zero_state

```
zero_state(
    batch_size,
    dtype
)
```