tf.layers.BatchNormalization

Contents

Class `BatchNormalization`

Inherits From: `Layer`

Defined in `tensorflow/python/layers/normalization.py`.

Batch Normalization layer from http://arxiv.org/abs/1502.03167.

"Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift"

Sergey Ioffe, Christian Szegedy

Arguments:

- `axis`: Integer, the axis that should be normalized (typically the features axis). For instance, after a `Conv2D` layer with `data_format="channels_first"`, set `axis=1` in `BatchNormalization`.
- `momentum`: Momentum for the moving average.
- `epsilon`: Small float added to variance to avoid dividing by zero.
- `center`: If True, add offset of `beta` to normalized tensor. If False, `beta` is ignored.
- `scale`: If True, multiply by `gamma`. If False, `gamma` is not used. When the next layer is linear (also e.g. `nn.relu`), this can be disabled since the scaling can be done by the next layer.
- `beta_initializer`: Initializer for the beta weight.
- `gamma_initializer`: Initializer for the gamma weight.
- `moving_mean_initializer`: Initializer for the moving mean.
- `moving_variance_initializer`: Initializer for the moving variance.
- `beta_regularizer`: Optional regularizer for the beta weight.
- `gamma_regularizer`: Optional regularizer for the gamma weight.
- `beta_constraint`: An optional projection function to be applied to the `beta` weight after being updated by an `Optimizer` (e.g. used to implement norm constraints or value constraints for layer weights). The function must take as input the unprojected variable and must return the projected variable (which must have the same shape). Constraints are not safe to use when doing asynchronous distributed training.
- `gamma_constraint`: An optional projection function to be applied to the `gamma` weight after being updated by an `Optimizer`.
- `renorm`: Whether to use Batch Renormalization (https://arxiv.org/abs/1702.03275). This adds extra variables during training. The inference is the same for either value of this parameter.

- `renorm_clipping`: A dictionary that may map keys 'rmax', 'rmin', 'dmax' to scalar `Tensors` used to clip the renorm correction. The correction `(r, d)` is used as `corrected_value = normalized_value * r + d`, with `r` clipped to [rmin, rmax], and `d` to [-dmax, dmax]. Missing rmax, rmin, dmax are set to inf, 0, inf, respectively.
- `renorm_momentum`: Momentum used to update the moving means and standard deviations with renorm. Unlike `momentum`, this affects training and should be neither too small (which would add noise) nor too large (which would give stale estimates). Note that `momentum` is still applied to get the means and variances for inference.
- `fused`: if `True`, use a faster, fused implementation if possible. If `None`, use the system recommended implementation.
- `trainable`: Boolean, if `True` also add variables to the graph collection `GraphKeys.TRAINABLE_VARIABLES` (see tf.Variable).
- `name`: A string, the name of the layer.

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.

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__(
    axis=-1,
    momentum=0.99,
    epsilon=0.001,
    center=True,
    scale=True,
    beta_initializer=tf.zeros_initializer(),
    gamma_initializer=tf.ones_initializer(),
    moving_mean_initializer=tf.zeros_initializer(),
    moving_variance_initializer=tf.ones_initializer(),
    beta_regularizer=None,
    gamma_regularizer=None,
    beta_constraint=None,
    gamma_constraint=None,
    renorm=False,
    renorm_clipping=None,
    renorm_momentum=0.99,
    fused=None,
    trainable=True,
    name=None,
    **kwargs
)
```

__call__

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

Wraps `call`, applying pre- and post-processing steps.

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`. **Note**: kwarg `scope` is reserved for use by the layer.

Returns:

Output tensor(s).

> ★ **Note:** - If the layer's `call` method takes a `scope` keyword argument, this argument will be automatically set to the current variable scope. - If the layer's `call` method takes a `mask` argument (as some Keras layers do), its default value will be set to the mask generated for `inputs` by the previous layer (if `input` did come from a layer that generated a corresponding mask, i.e. if it came from a Keras layer with masking support.

Raises:

- `ValueError`: if the layer's `call` method returns None (an invalid value).

`__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(input_shape)
```

`call`

```
call(
    inputs,
    training=False
)
```

`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.

Last updated November 2, 2017.