tf.keras.layers.ConvLSTM2D

Contents

Class **ConvLSTM2D**

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

Convolutional LSTM.

It is similar to an LSTM layer, but the input transformations and recurrent transformations are both convolutional.

Arguments:

- `filters`: Integer, the dimensionality of the output space (i.e. the number output of filters in the convolution).
- `kernel_size`: An integer or tuple/list of n integers, specifying the dimensions of the convolution window.
- `strides`: An integer or tuple/list of n integers, specifying the strides of the convolution. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1.
- `padding`: One of `"valid"` or `"same"` (case-insensitive).
- `data_format`: A string, one of `channels_last` (default) or `channels_first`. The ordering of the dimensions in the inputs. `channels_last` corresponds to inputs with shape `(batch, time, ..., channels)` while `channels_first` corresponds to inputs with shape `(batch, time, 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".
- `dilation_rate`: An integer or tuple/list of n integers, specifying the dilation rate to use for dilated convolution. Currently, specifying any `dilation_rate` value != 1 is incompatible with specifying any `strides` value != 1.
- `activation`: Activation function to use. If you don't specify anything, no activation is applied (ie. "linear" activation: `a(x) = x`).
- `recurrent_activation`: Activation function to use for the recurrent step.
- `use_bias`: Boolean, whether the layer uses a bias vector.
- `kernel_initializer`: Initializer for the `kernel` weights matrix, used for the linear transformation of the inputs..
- `recurrent_initializer`: Initializer for the `recurrent_kernel` weights matrix, used for the linear transformation of the recurrent state..
- `bias_initializer`: Initializer for the bias vector.
- `unit_forget_bias`: Boolean. If True, add 1 to the bias of the forget gate at initialization. Use in combination with `bias_initializer="zeros"`. This is recommended in Jozefowicz et al.
- `kernel_regularizer`: Regularizer function applied to the `kernel` weights matrix.
- `recurrent_regularizer`: Regularizer function applied to the `recurrent_kernel` weights matrix.

- `bias_regularizer`: Regularizer function applied to the bias vector.
- `activity_regularizer`: Regularizer function applied to the output of the layer (its "activation")..
- `kernel_constraint`: Constraint function applied to the `kernel` weights matrix.
- `recurrent_constraint`: Constraint function applied to the `recurrent_kernel` weights matrix.
- `bias_constraint`: Constraint function applied to the bias vector.
- `return_sequences`: Boolean. Whether to return the last output in the output sequence, or the full sequence.
- `go_backwards`: Boolean (default False). If True, rocess the input sequence backwards.
- `stateful`: Boolean (default False). If True, the last state for each sample at index i in a batch will be used as initial state for the sample of index i in the following batch.
- `dropout`: Float between 0 and 1. Fraction of the units to drop for the linear transformation of the inputs.
- `recurrent_dropout`: Float between 0 and 1. Fraction of the units to drop for the linear transformation of the recurrent state.

Input shape: - if data_format='channels_first' 5D tensor with shape: `(samples,time, channels, rows, cols)` - if data_format='channels_last' 5D tensor with shape: `(samples,time, rows, cols, channels)`

Output shape: - if `return_sequences` - if data_format='channels_first' 5D tensor with shape: `(samples, time, filters, output_row, output_col)` - if data_format='channels_last' 5D tensor with shape: `(samples, time, output_row, output_col, filters)` - else - if data_format ='channels_first' 4D tensor with shape: `(samples, filters, output_row, output_col)` - if data_format='channels_last' 4D tensor with shape: `(samples, output_row, output_col, filters)` where o_row and o_col depend on the shape of the filter and the padding

Raises:

- `ValueError`: in case of invalid constructor arguments.

References: - Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting The current implementation does not include the feedback loop on the cells output

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__(
    filters,
    kernel_size,
    strides=(1, 1),
    padding='valid',
    data_format=None,
    dilation_rate=(1, 1),
    activation='tanh',
    recurrent_activation='hard_sigmoid',
    use_bias=True,
    kernel_initializer='glorot_uniform',
    recurrent_initializer='orthogonal',
    bias_initializer='zeros',
    unit_forget_bias=True,
    kernel_regularizer=None,
    recurrent_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    recurrent_constraint=None,
    bias_constraint=None,
    return_sequences=False,
    go_backwards=False,
    stateful=False,
    dropout=0.0,
    recurrent_dropout=0.0,
    **kwargs
)
```

`__call__`

```
__call__(
    inputs,
    initial_state=None,
    **kwargs
)
```

`__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(input_shape)
```

`call`

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

`compute_mask`

```
compute_mask(
    inputs,
    mask
)
```

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_constants

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

get_initial_state

```
get_initial_state(inputs)
```

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.

input_conv

```
input_conv(
    x,
    w,
    b=None,
    padding='valid'
)
```

preprocess_input

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

reccurent_conv

```
reccurent_conv(
    x,
    w
)
```

reset_states

```
reset_states()
```

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.

step

```
step(
    inputs,
    states
)
```

Last updated November 2, 2017.