

tf.contrib.rnn.IntersectionRNNCell

Contents

Class IntersectionRNNCell

Properties

activity_regularizer

dtype

Class IntersectionRNNCell

Inherits From: [RNNCell](#)

Defined in [tensorflow/contrib/rnn/python/ops/rnn_cell.py](#).

Intersection Recurrent Neural Network (+RNN) cell.

Architecture with coupled recurrent gate as well as coupled depth gate, designed to improve information flow through stacked RNNs. As the architecture uses depth gating, the dimensionality of the depth output (y) also should not change through depth (input size == output size). To achieve this, the first layer of a stacked Intersection RNN projects the inputs to N (num units) dimensions. Therefore when initializing an IntersectionRNNCell, one should set `num_in_proj = N` for the first layer and use default settings for subsequent layers.

This implements the recurrent cell from the paper:

<https://arxiv.org/abs/1611.09913>

Jasmine Collins, Jascha Sohl-Dickstein, and David Sussillo. "Capacity and Trainability in Recurrent Neural Networks" Proc. ICLR 2017.

The Intersection RNN is built for use in deeply stacked RNNs so it may not achieve best performance with depth 1.

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__(
    num_units,
    num_in_proj=None,
    initializer=None,
    forget_bias=1.0,
    y_activation=tf.nn.relu,
    reuse=None
)

```

Initialize the parameters for an +RNN cell.

Args:

- `num_units` : int, The number of units in the +RNN cell
- `num_in_proj` : (optional) int, The input dimensionality for the RNN. If creating the first layer of an +RNN, this should be set to `num_units`. Otherwise, this should be set to `None` (default). If `None`, dimensionality of `inputs` should be equal to `num_units`, otherwise `ValueError` is thrown.
- `initializer` : (optional) The initializer to use for the weight matrices.
- `forget_bias` : (optional) float, default 1.0, The initial bias of the forget gates, used to reduce the scale of forgetting at the beginning of the training.
- `y_activation` : (optional) Activation function of the states passed through depth. Default is 'tf.nn.relu'.
- `reuse` : (optional) Python boolean describing whether to reuse variables in an existing scope. If not `True`, and the existing scope already has the given variables, an error is raised.

`__call__`

```

__call__(
    inputs,
    state,
    scope=None
)

```

Run this RNN cell on inputs, starting from the given state.

Args:

- `inputs` : 2-D tensor with shape `[batch_size x input_size]`.
- `state` : if `self.state_size` is an integer, this should be a 2-D Tensor with shape `[batch_size x self.state_size]`. Otherwise, if `self.state_size` is a tuple of integers, this should be a tuple with shapes `[batch_size x s] for s in self.state_size`.
- `scope` : VariableScope for the created subgraph; defaults to class name.

Returns:

A pair containing:

- Output: A 2-D tensor with shape `[batch_size x self.output_size]`.
- New state: Either a single 2-D tensor, or a tuple of tensors matching the arity and shapes of `state`.

`__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,  
    state  
)
```

Run one step of the Intersection RNN.

Args:

- `inputs` : input Tensor, 2D, batch x input size.
- `state` : state Tensor, 2D, batch x num units.

Returns:

- `new_y` : batch x num units, Tensor representing the output of the +RNN after reading `inputs` when previous state was `state` .
- `new_state` : batch x num units, Tensor representing the state of the +RNN after reading `inputs` when previous state was `state` .

Raises:

- `ValueError` : If input size cannot be inferred from `inputs` via static shape inference.
- `ValueError` : If input size != output size (these must be equal when using the Intersection RNN).

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  
)
```

Return zero-filled state tensor(s).

Args:

- `batch_size`: int, float, or unit Tensor representing the batch size.
- `dtype`: the data type to use for the state.

Returns:

If `state_size` is an int or TensorShape, then the return value is a **N-D** tensor of shape `[batch_size x state_size]` filled with zeros.

If `state_size` is a nested list or tuple, then the return value is a nested list or tuple (of the same structure) of **2-D** tensors with the shapes `[batch_size x s]` for each s in `state_size`.

Except as otherwise noted, the content of this page is licensed under the [Creative Commons Attribution 3.0 License](#), and code samples are licensed under the [Apache 2.0 License](#). For details, see our [Site Policies](#). Java is a registered trademark of Oracle and/or its affiliates.

Last updated November 2, 2017.

Stay Connected

[Blog](#)

[GitHub](#)

[Twitter](#)

Support

[Issue Tracker](#)

[Release Notes](#)

English

[Terms](#) | [Privacy](#)