tf.contrib.graph_editor.SubGraphView

Contents

Class **SubGraphView**

Defined in `tensorflow/contrib/graph_editor/subgraph.py`.

See the guide: Graph Editor (contrib) > Module: subgraph

A subgraph view on an existing `tf.Graph`.

An instance of this class is a subgraph view on an existing `tf.Graph`. "subgraph" means that it can represent part of the whole `tf.Graph`. "view" means that it only provides a passive observation and do not to act on the `tf.Graph`. Note that in this documentation, the term "subgraph" is often used as substitute to "subgraph view".

A subgraph contains:

- a list of input tensors, accessible via the `inputs` property.
- a list of output tensors, accessible via the `outputs` property.
- and the operations in between, accessible via the "ops" property.

An subgraph can be seen as a function F(i0, i1, ...) -> o0, o1, ... It is a function which takes as input some input tensors and returns as output some output tensors. The computation that the function performs is encoded in the operations of the subgraph.

The tensors (input or output) can be of two kinds:

- connected: a connected tensor connects to at least one operation contained in the subgraph. One example is a subgraph representing a single operation and its inputs and outputs: all the input and output tensors of the op are "connected".
- passthrough: a passthrough tensor does not connect to any operation contained in the subgraph. One example is a subgraph representing a single tensor: this tensor is passthrough. By default a passthrough tensor is present both in the input and output tensors of the subgraph. It can however be remapped to only appear as an input (or output) only.

The input and output tensors can be remapped. For instance, some input tensor can be omitted. For instance, a subgraph representing an operation with two inputs can be remapped to only take one input. Note that this does not change at all the underlying `tf.Graph` (remember, it is a view). It means that the other input is being ignored, or is being treated as "given". The analogy with functions can be extended like this: F(x,y) is the original function. Remapping the inputs from [x, y] to just [x] means that the subgraph now represent the function F_y(x) (y is "given").

The output tensors can also be remapped. For instance, some output tensor can be omitted. Other output tensor can be duplicated as well. As mentioned before, this does not change at all the underlying `tf.Graph`. The analogy with functions

can be extended like this: F(...)->x,y is the original function. Remapping the outputs from [x, y] to just [y,y] means that the subgraph now represent the function M(F(...)) where M is the function M(a,b)->b,b.

It is useful to describe three other kind of tensors:

- internal: an internal tensor is a tensor connecting operations contained in the subgraph. One example in the subgraph representing the two operations A and B connected sequentially: -> A -> B ->. The middle arrow is an internal tensor.
- actual input: an input tensor of the subgraph, regardless of whether it is listed in "inputs" or not (masked-out).
- actual output: an output tensor of the subgraph, regardless of whether it is listed in "outputs" or not (masked-out).
- hidden input: an actual input which has been masked-out using an input remapping. In other word, a hidden input is a non-internal tensor not listed as a input tensor and one of whose consumers belongs to the subgraph.
- hidden output: a actual output which has been masked-out using an output remapping. In other word, a hidden output is a non-internal tensor not listed as an output and one of whose generating operations belongs to the subgraph.

Here are some useful guarantees about an instance of a SubGraphView:

- the input (or output) tensors are not internal.
- the input (or output) tensors are either "connected" or "passthrough".
- the passthrough tensors are not connected to any of the operation of the subgraph.

Note that there is no guarantee that an operation in a subgraph contributes at all to its inputs or outputs. For instance, remapping both the inputs and outputs to empty lists will produce a subgraph which still contains all the original operations. However, the remove_unused_ops function can be used to make a new subgraph view whose operations are connected to at least one of the input or output tensors.

An instance of this class is meant to be a lightweight object which is not modified in-place by the user. Rather, the user can create new modified instances of a given subgraph. In that sense, the class SubGraphView is meant to be used like an immutable python object.

A common problem when using views is that they can get out-of-sync with the data they observe (in this case, a `tf.Graph`). This is up to the user to ensure that this doesn't happen. To keep on the safe side, it is recommended that the life time of subgraph views are kept very short. One way to achieve this is to use subgraphs within a "with make_sgv(...) as sgv:" Python context.

To alleviate the out-of-sync problem, some functions are granted the right to modified subgraph in place. This is typically the case of graph manipulation functions which, given some subgraphs as arguments, can modify the underlying `tf.Graph`. Since this modification is likely to render the subgraph view invalid, those functions can modify the argument in place to reflect the change. For instance, calling the function swap_inputs(svg0, svg1) will modify svg0 and svg1 in place to reflect the fact that their inputs have now being swapped.

Properties

`connected_inputs`

The connected input tensors of this subgraph view.

`connected_outputs`

The connected output tensors of this subgraph view.

`graph`

The underlying `tf.Graph`.

`inputs`

The input tensors of this subgraph view.

`ops`

The operations in this subgraph view.

`outputs`

The output tensors of this subgraph view.

`passthroughs`

The passthrough tensors, going straight from input to output.

Methods

`__init__`

```
__init__(
    inside_ops=(),
    passthrough_ts=()
)
```

Create a subgraph containing the given ops and the "passthrough" tensors.

Args:

- `inside_ops`: an object convertible to a list of `tf.Operation`. This list defines all the operations in the subgraph.
- `passthrough_ts`: an object convertible to a list of `tf.Tensor`. This list define all the "passthrough" tensors. A passthrough tensor is a tensor which goes directly from the input of the subgraph to it output, without any intermediate operations. All the non passthrough tensors are silently ignored.

Raises:

- `TypeError`: if inside_ops cannot be converted to a list of `tf.Operation` or if `passthrough_ts` cannot be converted to a list of `tf.Tensor`.

`__bool__`

```
__bool__()
```

Allows for implicit boolean conversion.

`__copy__`

```
__copy__()
```

Create a copy of this subgraph.

Note that this class is a "view", copying it only create another view and does not copy the underlying part of the `tf.Graph`.

Returns:

A new identical instance of the original subgraph view.

`__enter__`

```
__enter__()
```

Allow Python context to minimize the life time of a subgraph view.

A subgraph view is meant to be a lightweight and transient object. A short lifetime will alleviate the "out-of-sync" issue mentioned earlier. For that reason, a SubGraphView instance can be used within a Python context. For example:

from tensorflow.contrib import graph_editor as ge with ge.make_sgv(...) as sgv: print(sgv)

Returns:

Itself.

`__exit__`

```
__exit__(
    exc_type,
    exc_value,
    traceback
)
```

`__nonzero__`

```
__nonzero__()
```

Allows for implicit boolean conversion.

`consumers`

```
consumers()
```

Return a Python set of all the consumers of this subgraph view.

A consumer of a subgraph view is a tf.Operation which is a consumer of one of the output tensors and is not in the subgraph.

Returns:

A list of `tf.Operation` which are the consumers of this subgraph view.

`copy`

```
copy()
```

Return a copy of itself.

Note that this class is a "view", copying it only create another view and does not copy the underlying part of the tf.Graph.

Returns:

A new instance identical to the original one.

`find_op_by_name`

```
find_op_by_name(op_name)
```

Return the op named op_name.

Args:

- `op_name` : the name to search for

Returns:

The op named op_name.

Raises:

- `ValueError` : if the op_name could not be found.
- `AssertionError` : if the name was found multiple time.

`input_index`

```
input_index(t)
```

Find the input index corresponding to the given input tensor t.

Args:

- `t` : the input tensor of this subgraph view.

Returns:

The index in the self.inputs list.

Raises:

- `Error` : if t in not an input tensor.

`is_passthrough`

```
is_passthrough(t)
```

Check whether a tensor is passthrough.

op

```
op(op_id)
```

Get an op by its index.

output_index

```
output_index(t)
```

Find the output index corresponding to given output tensor t.

Args:

- `t`: the output tensor of this subgraph view.

Returns:

The index in the self.outputs list.

Raises:

- `Error`: if t in not an output tensor.

remap

```
remap(
    new_input_indices=None,
    new_output_indices=None
)
```

Remap the inputs and outputs of the subgraph.

Note that this is only modifying the view: the underlying tf.Graph is not affected.

Args:

- `new_input_indices`: an iterable of integers or tf.Tensors representing a mapping between the old inputs and the new ones. Integers must be positive and smaller than the number of old inputs. tf.Tensors must belong to the old list of inputs. This mapping can be under-complete and must be without repetitions.
- `new_output_indices`: an iterable of integers or tf.Tensors representing a mapping between the old outputs and the new ones. Integers must be positive and smaller than the number of old outputs. tf.Tensors must belong to the old list of outputs. This mapping can be under-complete and can have repetitions.

Returns:

A new modified instance of the original subgraph view with remapped inputs and outputs.

remap_default

```
remap_default(
    remove_input_map=True,
    remove_output_map=True
)
```

Remap the inputs and/or outputs to the default mapping.

Args:

- `remove_input_map`: if True the input map is reset to the default one.
- `remove_output_map`: if True the output map is reset to the default one.

Returns:

A new modified instance of the original subgraph view with its input and/or output mapping reset to the default one.

`remap_inputs`

```
remap_inputs(new_input_indices)
```

Remap the inputs of the subgraph.

If the inputs of the original subgraph are [t0, t1, t2], remapping to [2,0] will create a new instance whose inputs is [t2, t0].

Note that this is only modifying the view: the underlying `tf.Graph` is not affected.

Args:

- `new_input_indices`: an iterable of integers or tf.Tensors representing a mapping between the old inputs and the new ones. Integers must be positive and smaller than the number of old inputs. tf.Tensors must belong to the old list of inputs. This mapping can be under-complete and must be without repetitions.

Returns:

A new modified instance of the original subgraph view with remapped inputs.

`remap_outputs`

```
remap_outputs(new_output_indices)
```

Remap the output of the subgraph.

If the output of the original subgraph are [t0, t1, t2], remapping to [1,1,0] will create a new instance whose outputs is [t1, t1, t0].

Note that this is only modifying the view: the underlying tf.Graph is not affected.

Args:

- `new_output_indices`: an iterable of integers or tf.Tensors representing a mapping between the old outputs and the new ones. Integers must be positive and smaller than the number of old outputs. tf.Tensors must belong to the old list of outputs. This mapping can be under-complete and can have repetitions.

Returns:

A new modified instance of the original subgraph view with remapped outputs.

`remap_outputs_make_unique`

```
remap_outputs_make_unique()
```

Remap the outputs so that all the tensors appears only once.

`remap_outputs_to_consumers`

```
remap_outputs_to_consumers()
```

Remap the outputs to match the number of consumers.

`remove_unused_ops`

```
remove_unused_ops(control_inputs=True)
```

Remove unused ops.

Args:

- `control_inputs`: if True, control inputs are used to detect used ops.

Returns:

A new subgraph view which only contains used operations.
