tf.variable_scope

Contents

Class `variable_scope`

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

See the guide: Variables > Sharing Variables

A context manager for defining ops that creates variables (layers).

This context manager validates that the (optional) `values` are from the same graph, ensures that graph is the default graph, and pushes a name scope and a variable scope.

If `name_or_scope` is not None, it is used as is. If `scope` is None, then `default_name` is used. In that case, if the same name has been previously used in the same scope, it will be made unique by appending `_N` to it.

Variable scope allows you to create new variables and to share already created ones while providing checks to not create or share by accident. For details, see the Variable Scope How To, here we present only a few basic examples.

Simple example of how to create a new variable:

```
with tf.variable_scope("foo"):
    with tf.variable_scope("bar"):
        v = tf.get_variable("v", [1])
        assert v.name == "foo/bar/v:0"
```

Basic example of sharing a variable AUTO_REUSE:

```
def foo():
  with tf.variable_scope("foo", reuse=tf.AUTO_REUSE):
    v = tf.get_variable("v", [1])
  return v

v1 = foo()  # Creates v.
v2 = foo()  # Gets the same, existing v.
assert v1 == v2

Basic example of sharing a variable with reuse=True:

```python
with tf.variable_scope("foo"):
    v = tf.get_variable("v", [1])
with tf.variable_scope("foo", reuse=True):
    v1 = tf.get_variable("v", [1])
assert v1 == v
```
```

Sharing a variable by capturing a scope and setting reuse:

```
with tf.variable_scope("foo") as scope:
    v = tf.get_variable("v", [1])
    scope.reuse_variables()
    v1 = tf.get_variable("v", [1])
assert v1 == v
```

To prevent accidental sharing of variables, we raise an exception when getting an existing variable in a non-reusing scope.

```
with tf.variable_scope("foo"):
    v = tf.get_variable("v", [1])
    v1 = tf.get_variable("v", [1])
    #  Raises ValueError("... v already exists ...").
```

Similarly, we raise an exception when trying to get a variable that does not exist in reuse mode.

```
with tf.variable_scope("foo", reuse=True):
    v = tf.get_variable("v", [1])
    #  Raises ValueError("... v does not exists ...").
```

Note that the `reuse` flag is inherited: if we open a reusing scope, then all its sub-scopes become reusing as well.

A note about name scoping: Setting `reuse` does not impact the naming of other ops such as mult. See related discussion on github#6189

Note that up to and including version 1.0, it was allowed (though explicitly discouraged) to pass False to the reuse argument, yielding undocumented behaviour slightly different from None. Starting at 1.1.0 passing None and False as reuse has exactly the same effect.

Methods

`__init__`

```
__init__(
    name_or_scope,
    default_name=None,
    values=None,
    initializer=None,
    regularizer=None,
    caching_device=None,
    partitioner=None,
    custom_getter=None,
    reuse=None,
    dtype=None,
    use_resource=None,
    constraint=None
)
```

Initialize the context manager.

Args:

- `name_or_scope`: `string` or `VariableScope`: the scope to open.
- `default_name`: The default name to use if the `name_or_scope` argument is `None`, this name will be uniquified. If name_or_scope is provided it won't be used and therefore it is not required and can be None.
- `values`: The list of `Tensor` arguments that are passed to the op function.

- `initializer`: default initializer for variables within this scope.
- `regularizer`: default regularizer for variables within this scope.
- `caching_device`: default caching device for variables within this scope.
- `partitioner`: default partitioner for variables within this scope.
- `custom_getter`: default custom getter for variables within this scope.
- `reuse`: `True`, None, or tf.AUTO_REUSE; if `True`, we go into reuse mode for this scope as well as all sub-scopes; if tf.AUTO_REUSE, we create variables if they do not exist, and return them otherwise; if None, we inherit the parent scope's reuse flag. In Eager mode, this argument is always forced to be tf.AUTO_REUSE.
- `dtype`: type of variables created in this scope (defaults to the type in the passed scope, or inherited from parent scope).
- `use_resource`: If False, all variables will be regular Variables. If True, experimental ResourceVariables with well-defined semantics will be used instead. Defaults to False (will later change to True). In Eager mode, this argument is always forced to be True.
- `constraint`: An optional projection function to be applied to the variable 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 Tensor representing the value of the variable and return the Tensor for the projected value (which must have the same shape). Constraints are not safe to use when doing asynchronous distributed training.

Returns:

A scope that can be captured and reused.

Raises:

- `ValueError`: when trying to reuse within a create scope, or create within a reuse scope.
- `TypeError`: when the types of some arguments are not appropriate.

`__enter__`

```
__enter__()
```

`__exit__`

```
__exit__(
    type_arg,
    value_arg,
    traceback_arg
)
```