tf.compat.v1.Session

View source (https://github.com/tensorflow/tensorflow/blob/v2.4.0/tensorflow/python/client/session.py#l on L1685) GitHub

A class for running TensorFlow operations.

```
tf.compat.v1.Session(
    target='', graph=None, config=None
)
```

Used in the notebooks

Used in the guide	Used in the tutorials
• Migrate your TensorFlow 1 code to TensorFlow 2 (https://www.tensorflow.org/guide/migrate)	• Universal Sentence Encoder-Lite demo (https://www.tensorflow.org/hub/tutorials/semantic_sim • Generating Images with Little Data Using S3GAN (https://www.tensorflow.org/hub/tutorials/s3gan_genera • Generating Images with BigBiGAN (https://www.tensorflo • Generating Images with BigGAN (https://www.tensorflow.org/hub/tutorials/biggan_genera • Exploring the TF-Hub CORD-19 Swivel Embeddings (https://www.tensorflow.org/hub/tutorials/cord_19_embe

A `Session` object encapsulates the environment in which `Operation` objects are executed, and `Tensor` objects are evaluated. For example:

```
tf.compat.v1.disable_eager_execution() # need to disable eager in TF2.x
# Build a graph.
a = tf.constant(5.0)
```

```
b = tf.constant(6.0)
c = a * b

# Launch the graph in a session.
sess = tf.compat.v1.Session()

# Evaluate the tensor `c`.
print(sess.run(c)) # prints 30.0
```

A session may own resources, such as `tf.Variable` (https://www.tensorflow.org/api_docs/python/tf/Variable), `tf.queue.QueueBase` (https://www.tensorflow.org/api_docs/python/tf/queue/QueueBase), and `tf.compat.v1.ReaderBase` (https://www.tensorflow.org/api_docs/python/tf/compat/v1/ReaderBase). It is important to release these resources when they are no longer required. To do this, either invoke the `tf.Session.close` method on the session, or use the session as a context manager. The following two examples are equivalent:

```
# Using the `close()` method.
sess = tf.compat.v1.Session()
sess.run(...)
sess.close()

# Using the context manager.
with tf.compat.v1.Session() as sess:
  sess.run(...)
```

The `ConfigProto` (https://www.tensorflow.org/code/tensorflow/core/protobuf/config.proto) protocol buffer exposes various configuration options for a session. For example, to create a session that uses soft constraints for device placement, and log the resulting placement decisions, create a session as follows:

```
# Launch the graph in a session that allows soft device placement and
# logs the placement decisions.
sess = tf.compat.v1.Session(config=tf.compat.v1.ConfigProto(
    allow_soft_placement=True,
    log_device_placement=True))
```

Args	
`target`	(Optional.) The execution engine to connect to. Defaults to using an in-process engine. See Distributed TensorFlow (https://tensorflow.org/deploy/distributed) for more examples.
`graph`	(Optional.) The `Graph` to be launched (described above).
`config`	(Optional.) A `ConfigProto` (https://www.tensorflow.org/code/tensorflow/core/protobuf/config.proto) protocol buffer with configuration options for the session.

Attributes	
`graph`	The graph that was launched in this session.
`graph_def`	A serializable version of the underlying TensorFlow graph.
`sess_str`	The TensorFlow process to which this session will connect.

Methods

as_default

View source (https://github.com/tensorflow/tensorflow/blob/v2.4.0/tensorflow/python/client/session.py#L803-L856)

```
as_default()
```

Returns a context manager that makes this object the default session.

Use with the `with` keyword to specify that calls to `tf.Operation.run` (https://www.tensorflow.org/api_docs/python/tf/Operation#run) or `tf.Tensor.eval` (https://www.tensorflow.org/api_docs/python/tf/Tensor#eval) should be executed in this session.

```
c = tf.constant(..)
sess = tf.compat.v1.Session()
```

```
with sess.as_default():
  assert tf.compat.v1.get_default_session() is sess
  print(c.eval())
```

To get the current default session, use `tf.compat.v1.get_default_session` (https://www.tensorflow.org/api_docs/python/tf/compat/v1/get_default_session).

The **`as_default`** context manager *does not* close the session when you exit the context, and you must close t
n explicitly.

```
c = tf.constant(...)
sess = tf.compat.v1.Session()
with sess.as_default():
  print(c.eval())
# ...
with sess.as_default():
  print(c.eval())

sess.close()
```

Alternatively, you can use `with tf.compat.v1.Session():` to create a session that is automatically closed on exiting the context, including when an uncaught exception is raised.

The default session is a property of the current thread. If you create a new thread, and wish to use the default
n in that thread, you must explicitly add a **`with sess.as_default():`** in that thread's function.

Entering a **`with sess.as_default():`** block does not affect the current default graph. If you are using multip
s, and **`sess.graph`** is different from the value of **`tf.compat.v1.get_default_graph`** (https://www.tensorflow.org/api_docs/python/tf/compat/v1/get_default_graph), you must explicitly enter a **`with graph.as_default():`** block to make **`sess.graph`** the default graph.

Returns

A context manager using this session as the default session.

close

View source (https://github.com/tensorflow/tensorflow/blob/v2.4.0/tensorflow/python/client/session.py#L754-L765)

```
close()
```

Closes this session.

Calling this method frees all resources associated with the session.

Raises	
tf.errors.OpError (https://www.tensorflow.org/api_docs/python/tf/errors/OpError)	Or one of its subclasses if an error occurs while closing the TensorFlow session.

list_devices

View source (https://github.com/tensorflow/tensorflow/blob/v2.4.0/tensorflow/python/client/session.py#L716-L752)

```
list_devices()
```

Lists available devices in this session.

```
devices = sess.list_devices()
for d in devices:
  print(d.name)
```

Where:

Each element in the list has the following properties

- `name`: A string with the full name of the device. ex: `/job:worker/replica:0/task:3/device:CPU:0`
- `device_type`: The type of the device (e.g. `CPU`, `GPU`, `TPU`.)
- `memory_limit`: The maximum amount of memory available on the device. Note: depending on the device, it is possible the usable memory could be substantially less.

Raises	
`tf.errors.OpError` (https://www.tensorflow.org/api_docs/python/tf/errors/OpError)	If it encounters an error (e.g. session is in an invalid state, or network errors occur).

Returns
A list of devices in the session.

make_callable

View source (https://github.com/tensorflow/tensorflow/blob/v2.4.0/tensorflow/python/client/session.py#L1196-L1319)

```
make_callable(
    fetches, feed_list=None, accept_options=False
)
```

Returns a Python callable that runs a particular step.

The returned callable will take `len(feed_list)` arguments whose types must be compatible feed values for the respective elements of `feed_list`. For example, if element `i` of `feed_list` is a `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor), the `i`th argument to the returned callable must be a numpy ndarray (or something convertible to an ndarray) with matching element type and shape. See `tf.Session.run` for details of the allowable feed key and value types.

The returned callable will have the same return type as `tf.Session.run(fetches, ...)`. For example, if `fetches` is a `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor), the

callable will return a numpy ndarray; if `fetches` is a `tf.Operation` (https://www.tensorflow.org/api_docs/python/tf/Operation), it will return `None`.

Args	
`fetches`	A value or list of values to fetch. See `tf.Session.run` for details of the allowable fetch types.
`feed_list`	(Optional.) A list of `feed_dict` keys. See `tf.Session.run` for details of the allowable feed key types.
`accept_options`	(Optional.) If `True`, the returned `Callable` will be able to accept `tf.compat.v1.RunOptions` (https://www.tensorflow.org/api_docs/python/tf/compat/v1/RunOptions) and `tf.compat.v1.RunMetadata` (https://www.tensorflow.org/api_docs/python/tf/compat/v1/RunMetadata) as optional keyword arguments `options` and `run_metadata`, respectively, with the same syntax and semantics as `tf.Session.run`, which is useful for certain use cases (profiling and debugging) but will result in measurable slowdown of the `Callable`'s performance. Default: `False`.

Returns
A function that when called will execute the step defined by `feed_list` and `fetches` in this session.

Raises	
`TypeError`	If `fetches` or `feed_list` cannot be interpreted as arguments to `tf.Session.run`.

partial_run

View source (https://github.com/tensorflow/tensorflow/blob/v2.4.0/tensorflow/python/client/session.py#L979-L1024)

```
partial_run(
    handle, fetches, feed_dict=None
```

```
)
```

Continues the execution with more feeds and fetches.

This is EXPERIMENTAL and subject to change.

To use partial execution, a user first calls `partial_run_setup()` and then a sequence of `partial_run()`. `partial_run_setup` specifies the list of feeds and fetches that will be used in the subsequent `partial_run` calls.

The optional `feed_dict` argument allows the caller to override the value of tensors in the graph. See run() for more information.

Below is a simple example:

```
a = array_ops.placeholder(dtypes.float32, shape=[])
b = array_ops.placeholder(dtypes.float32, shape=[])
c = array_ops.placeholder(dtypes.float32, shape=[])
r1 = math_ops.add(a, b)
r2 = math_ops.multiply(r1, c)

h = sess.partial_run_setup([r1, r2], [a, b, c])
res = sess.partial_run(h, r1, feed_dict={a: 1, b: 2})
res = sess.partial_run(h, r2, feed_dict={c: res})
```

Args	
`handle`	A handle for a sequence of partial runs.
`fetches`	A single graph element, a list of graph elements, or a dictionary whose values are graph elements or lists of graph elements (see documentation for `run`).
`feed_dict`	A dictionary that maps graph elements to values (described above).

Returns
Either a single value if `fetches` is a single graph element, or a list of values if `fetches` is a list, or a dictionary with the same keys as `fetches` if that is a dictionary (see documentation for `run`).

Raises	
`tf.errors.OpError` (https://www.tensorflow.org/api_docs/python/tf/errors/OpError)	Or one of its subclasses on error.

`partial_run_setup`

View source (https://github.com/tensorflow/tensorflow/blob/v2.4.0/tensorflow/python/client/session.py#L1026-L1100)

```
partial_run_setup(
    fetches, feeds=None
)
```

Sets up a graph with feeds and fetches for partial run.

This is EXPERIMENTAL and subject to change.

Note that contrary to `run`, `feeds` only specifies the graph elements. The tensors will be supplied by the subsequent `partial_run` calls.

Args	
`fetches`	A single graph element, or a list of graph elements.
`feeds`	A single graph element, or a list of graph elements.

Returns
A handle for partial run.

Raises	
`RuntimeError`	If this `Session` is in an invalid state (e.g. has been closed).
`TypeError`	If `fetches` or `feed_dict` keys are of an inappropriate type.

`tf.errors.OpError` (https://www.tensorflow.org/api_docs/python/tf/errors/OpError)	Or one of its subclasses if a TensorFlow error happens.

reset

View source (https://github.com/tensorflow/tensorflow/blob/v2.4.0/tensorflow/python/client/session.py#L1653-L1685)

```
@staticmethod
reset(
    target, containers=None, config=None
)
```

Resets resource containers on `target`, and close all connected sessions.

A resource container is distributed across all workers in the same cluster as `target`. When a resource container on `target` is reset, resources associated with that container will be cleared. In particular, all Variables in the container will become undefined: they lose their values and shapes.

NOTE:

(i) reset() is currently only implemented for distributed sessions. (ii) Any sessions on the master named by `target` will be closed.

If no resource containers are provided, all containers are reset.

Args	
`target`	The execution engine to connect to.
`containers`	A list of resource container name strings, or **None** if all of all the containers are to be reset.
`config`	(Optional.) Protocol buffer with configuration options.

Raises

`tf.errors.OpError` (https://www.tensorflow.org/api_docs/python/tf/errors/OpError)	Or one of its subclasses if an error occurs while resetting containers.

run

View source (https://github.com/tensorflow/tensorflow/blob/v2.4.0/tensorflow/python/client/session.py#L858-L977)

```
run(
    fetches, feed_dict=None, options=None, run_metadata=None
)
```

Runs operations and evaluates tensors in `fetches`.

This method runs one "step" of TensorFlow computation, by running the necessary graph fragment to execute every `Operation` and evaluate every `Tensor` in `fetches`, substituting the values in `feed_dict` for the corresponding input values.

The `fetches` argument may be a single graph element, or an arbitrarily nested list, tuple, namedtuple, dict, or OrderedDict containing graph elements at its leaves. A graph element can be one of the following types:

- A `tf.Operation` (https://www.tensorflow.org/api_docs/python/tf/Operation). The corresponding fetched value will be `None`.
- A `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor). The corresponding fetched value will be a numpy ndarray containing the value of that tensor.
- A `tf.sparse.SparseTensor` (https://www.tensorflow.org/api_docs/python/tf/sparse/SparseTensor). The corresponding fetched value will be a `tf.compat.v1.SparseTensorValue` (https://www.tensorflow.org/api_docs/python/tf/compat/v1/SparseTensorValue) containing the value of that sparse tensor.
- A `get_tensor_handle` op. The corresponding fetched value will be a numpy ndarray containing the handle of that tensor.
- A `string` which is the name of a tensor or operation in the graph.

The value returned by `run()` has the same shape as the `fetches` argument, where the leaves are replaced by the corresponding values returned by TensorFlow.

Example:

```
a = tf.constant([10, 20])
b = tf.constant([1.0, 2.0])
# 'fetches' can be a singleton
v = session.run(a)
# v is the numpy array [10, 20]
# 'fetches' can be a list.
v = session.run([a, b])
# v is a Python list with 2 numpy arrays: the 1-D array [10, 20] and the
# 1-D array [1.0, 2.0]
# 'fetches' can be arbitrary lists, tuples, namedtuple, dicts:
MyData = collections.namedtuple('MyData', ['a', 'b'])
v = session.run({'k1': MyData(a, b), 'k2': [b, a]})
# v is a dict with
# v['k1'] is a MyData namedtuple with 'a' (the numpy array [10, 20]) and
# 'b' (the numpy array [1.0, 2.0])
# v['k2'] is a list with the numpy array [1.0, 2.0] and the numpy array
# [10, 20].
```

The optional `feed_dict` argument allows the caller to override the value of tensors in the graph. Each key in `feed_dict` can be one of the following types:

- If the key is a `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor), the value may be a Python scalar, string, list, or numpy ndarray that can be converted to the same `dtype` as that tensor. Additionally, if the key is a `tf.compat.v1.placeholder` (https://www.tensorflow.org/api_docs/python/tf/compat/v1/placeholder), the shape of the value will be checked for compatibility with the placeholder.
- If the key is a `tf.sparse.SparseTensor` (https://www.tensorflow.org/api_docs/python/tf/sparse/SparseTensor), the value should be a `tf.compat.v1.SparseTensorValue` (https://www.tensorflow.org/api_docs/python/tf/compat/v1/SparseTensorValue).
- If the key is a nested tuple of `Tensor`s or `SparseTensor`s, the value should be a nested tuple with the same structure that maps to their corresponding values as above.

Each value in `feed_dict` must be convertible to a numpy array of the dtype of the corresponding key.

The optional `options` argument expects a [`RunOptions`] proto. The options allow controlling the behavior of this particular step (e.g. turning tracing on).

The optional `run_metadata` argument expects a [`RunMetadata`] proto. When appropriate, the non-Tensor output of this step will be collected there. For example, when users turn on tracing in `options`, the profiled info will be collected into this argument and passed back.

Args	
`fetches`	A single graph element, a list of graph elements, or a dictionary whose values are graph elements or lists of graph elements (described above).
`feed_dict`	A dictionary that maps graph elements to values (described above).
`options`	A [`RunOptions`] protocol buffer
`run_metadata`	A [`RunMetadata`] protocol buffer

Returns
Either a single value if `fetches` is a single graph element, or a list of values if `fetches` is a list, or a dictionary with the same keys as `fetches` if that is a dictionary (described above). Order in which `fetches` operations are evaluated inside the call is undefined.

Raises	
`RuntimeError`	If this `Session` is in an invalid state (e.g. has been closed).
`TypeError`	If `fetches` or `feed_dict` keys are of an inappropriate type.
`ValueError`	If `fetches` or `feed_dict` keys are invalid or refer to a `Tensor` that doesn't exist.

`__enter__`

View source
(https://github.com/tensorflow/tensorflow/blob/v2.4.0/tensorflow/python/client/session.py#L1601-L1611)

```
__enter__()
```

__exit__

View source (https://github.com/tensorflow/tensorflow/blob/v2.4.0/tensorflow/python/client/session.py#L1613-L1651)

```
__exit__(
    exec_type, exec_value, exec_tb
)
```


Last updated 2020-12-14 UTC.