tf.test.TestCase

Contents

Class TestCase

Child Classes

Methods

__init__

Class **TestCase**

Defined in `tensorflow/python/framework/test_util.py`.

See the guide: Testing > Unit tests

Base class for tests that need to test TensorFlow.

Child Classes

`class failureException`

Methods

`__init__`

```
__init__(methodName='runTest')
```

`__call__`

```
__call__(
    *args,
    **kwds
)
```

`__eq__`

```
__eq__(other)
```

`__ne__`

```
__ne__(other)
```

`addCleanup`

```
addCleanup(
    function,
    *args,
    **kwargs
)
```

Add a function, with arguments, to be called when the test is completed. Functions added are called on a LIFO basis and are called after tearDown on test failure or success.

Cleanup items are called even if setUp fails (unlike tearDown).

addTypeEqualityFunc

```
addTypeEqualityFunc(
    typeobj,
    function
)
```

Add a type specific assertEqual style function to compare a type.

This method is for use by TestCase subclasses that need to register their own type equality functions to provide nicer error messages.

Args:

- `typeobj`: The data type to call this function on when both values are of the same type in assertEqual().
- `function`: The callable taking two arguments and an optional msg= argument that raises self.failureException with a useful error message when the two arguments are not equal.

assertAllClose

```
assertAllClose(
    a,
    b,
    rtol=1e-06,
    atol=1e-06
)
```

Asserts that two numpy arrays, or dicts of same, have near values.

This does not support nested dicts.

Args:

- `a`: The expected numpy ndarray (or anything can be converted to one), or dict of same. Must be a dict iff `b` is a dict.
- `b`: The actual numpy ndarray (or anything can be converted to one), or dict of same. Must be a dict iff `a` is a dict.
- `rtol`: relative tolerance.
- `atol`: absolute tolerance.

Raises:

- `ValueError`: if only one of `a` and `b` is a dict.

assertAllCloseAccordingToType

```
assertAllCloseAccordingToType(
    a,
    b,
    rtol=1e-06,
    atol=1e-06,
    float_rtol=1e-06,
    float_atol=1e-06,
    half_rtol=0.001,
    half_atol=0.001
)
```

Like assertAllClose, but also suitable for comparing fp16 arrays.

In particular, the tolerance is reduced to 1e-3 if at least one of the arguments is of type float16.

Args:

- `a` : the expected numpy ndarray or anything can be converted to one.
- `b` : the actual numpy ndarray or anything can be converted to one.
- `rtol` : relative tolerance.
- `atol` : absolute tolerance.
- `float_rtol` : relative tolerance for float32.
- `float_atol` : absolute tolerance for float32.
- `half_rtol` : relative tolerance for float16.
- `half_atol` : absolute tolerance for float16.

assertAllEqual

```
assertAllEqual(
    a,
    b
)
```

Asserts that two numpy arrays have the same values.

Args:

- `a` : the expected numpy ndarray or anything can be converted to one.
- `b` : the actual numpy ndarray or anything can be converted to one.

assertAlmostEqual

```
assertAlmostEqual(
    first,
    second,
    places=None,
    msg=None,
    delta=None
)
```

Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places

(default 7) and comparing to zero, or by comparing that the between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

If the two objects compare equal then they will automatically compare almost equal.

assertAlmostEquals

```
assertAlmostEquals(
    first,
    second,
    places=None,
    msg=None,
    delta=None
)
```

Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

If the two objects compare equal then they will automatically compare almost equal.

assertArrayNear

```
assertArrayNear(
    farray1,
    farray2,
    err
)
```

Asserts that two float arrays are near each other.

Checks that for all elements of farray1 and farray2 |f1 - f2| < err. Asserts a test failure if not.

Args:

- `farray1` : a list of float values.
- `farray2` : a list of float values.
- `err` : a float value.

assertDeviceEqual

```
assertDeviceEqual(
    device1,
    device2
)
```

Asserts that the two given devices are the same.

Args:

- `device1` : A string device name or TensorFlow **`DeviceSpec`** object.

- `device2` : A string device name or TensorFlow `DeviceSpec` object.

assertDictContainsSubset

```
assertDictContainsSubset(
    expected,
    actual,
    msg=None
)
```

Checks whether actual is a superset of expected.

assertDictEqual

```
assertDictEqual(
    d1,
    d2,
    msg=None
)
```

assertEqual

```
assertEqual(
    first,
    second,
    msg=None
)
```

Fail if the two objects are unequal as determined by the '==' operator.

assertEquals

```
assertEquals(
    first,
    second,
    msg=None
)
```

Fail if the two objects are unequal as determined by the '==' operator.

assertFalse

```
assertFalse(
    expr,
    msg=None
)
```

Check that the expression is false.

assertGreater

```
assertGreater(
    a,
    b,
    msg=None
)
```

Just like self.assertTrue(a > b), but with a nicer default message.

assertGreaterEqual

```
assertGreaterEqual(
    a,
    b,
    msg=None
)
```

Just like self.assertTrue(a >= b), but with a nicer default message.

assertIn

```
assertIn(
    member,
    container,
    msg=None
)
```

Just like self.assertTrue(a in b), but with a nicer default message.

assertIs

```
assertIs(
    expr1,
    expr2,
    msg=None
)
```

Just like self.assertTrue(a is b), but with a nicer default message.

assertIsInstance

```
assertIsInstance(
    obj,
    cls,
    msg=None
)
```

Same as self.assertTrue(isinstance(obj, cls)), with a nicer default message.

assertIsNone

```
assertIsNone(
    obj,
    msg=None
)
```

Same as self.assertTrue(obj is None), with a nicer default message.

assertIsNot

```
assertIsNot(
    expr1,
    expr2,
    msg=None
)
```

Just like self.assertTrue(a is not b), but with a nicer default message.

assertIsNotNone

```
assertIsNotNone(
    obj,
    msg=None
)
```

Included for symmetry with assertIsNone.

assertItemsEqual

```
assertItemsEqual(
    expected_seq,
    actual_seq,
    msg=None
)
```

An unordered sequence specific comparison. It asserts that actual_seq and expected_seq have the same element counts. Equivalent to::

```
self.assertEqual(Counter(iter(actual_seq)),
                 Counter(iter(expected_seq)))
```

Asserts that each element has the same count in both sequences. Example: - [0, 1, 1] and [1, 0, 1] compare equal. - [0, 0, 1] and [0, 1] compare unequal.

assertLess

```
assertLess(
    a,
    b,
    msg=None
)
```

Just like self.assertTrue(a < b), but with a nicer default message.

assertLessEqual

```
assertLessEqual(
    a,
    b,
    msg=None
)
```

Just like self.assertTrue(a <= b), but with a nicer default message.

assertListEqual

```
assertListEqual(
    list1,
    list2,
    msg=None
)
```

A list-specific equality assertion.

Args:

- `list1` : The first list to compare.
- `list2` : The second list to compare.
- `msg` : Optional message to use on failure instead of a list of differences.

assertMultiLineEqual

```
assertMultiLineEqual(
    first,
    second,
    msg=None
)
```

Assert that two multi-line strings are equal.

assertNDArrayNear

```
assertNDArrayNear(
    ndarray1,
    ndarray2,
    err
)
```

Asserts that two numpy arrays have near values.

Args:

- `ndarray1` : a numpy ndarray.
- `ndarray2` : a numpy ndarray.
- `err` : a float. The maximum absolute difference allowed.

assertNear

```
assertNear(
    f1,
    f2,
    err,
    msg=None
)
```

Asserts that two floats are near each other.

Checks that |f1 - f2| < err and asserts a test failure if not.

Args:

- `f1` : A float value.
- `f2` : A float value.
- `err` : A float value.
- `msg` : An optional string message to append to the failure message.

assertNotAlmostEqual

```
assertNotAlmostEqual(
    first,
    second,
    places=None,
    msg=None,
    delta=None
)
```

Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

Objects that are equal automatically fail.

assertNotAlmostEquals

```
assertNotAlmostEquals(
    first,
    second,
    places=None,
    msg=None,
    delta=None
)
```

Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

Objects that are equal automatically fail.

assertNotEqual

```
assertNotEqual(
    first,
    second,
    msg=None
)
```

Fail if the two objects are equal as determined by the '!=' operator.

assertNotEquals

```
assertNotEquals(
    first,
    second,
    msg=None
)
```

Fail if the two objects are equal as determined by the '!=' operator.

assertNotIn

```
assertNotIn(
    member,
    container,
    msg=None
)
```

Just like self.assertTrue(a not in b), but with a nicer default message.

assertNotIsInstance

```
assertNotIsInstance(
    obj,
    cls,
    msg=None
)
```

Included for symmetry with assertIsInstance.

assertNotRegexpMatches

```
assertNotRegexpMatches(
    text,
    unexpected_regexp,
    msg=None
)
```

Fail the test if the text matches the regular expression.

assertProtoEquals

```
assertProtoEquals(
    expected_message_maybe_ascii,
    message
)
```

Asserts that message is same as parsed expected_message_ascii.

Creates another prototype of message, reads the ascii message into it and then compares them using self._AssertProtoEqual().

Args:

- `expected_message_maybe_ascii` : proto message in original or ascii form.

- `message` : the message to validate.

assertProtoEqualsVersion

```
assertProtoEqualsVersion(
    expected,
    actual,
    producer=versions.GRAPH_DEF_VERSION,
    min_consumer=versions.GRAPH_DEF_VERSION_MIN_CONSUMER
)
```

assertRaises

```
assertRaises(
    excClass,
    callableObj=None,
    *args,
    **kwargs
)
```

Fail unless an exception of class excClass is raised by callableObj when invoked with arguments args and keyword arguments kwargs. If a different type of exception is raised, it will not be caught, and the test case will be deemed to have suffered an error, exactly as for an unexpected exception.

If called with callableObj omitted or None, will return a context object used like this::

```
 with self.assertRaises(SomeException):
     do_something()
```

The context manager keeps a reference to the exception as the 'exception' attribute. This allows you to inspect the exception after the assertion::

```
with self.assertRaises(SomeException) as cm:
    do_something()
the_exception = cm.exception
self.assertEqual(the_exception.error_code, 3)
```

assertRaisesOpError

```
assertRaisesOpError(expected_err_re_or_predicate)
```

assertRaisesRegexp

```
assertRaisesRegexp(
    expected_exception,
    expected_regexp,
    callable_obj=None,
    *args,
    **kwargs
)
```

Asserts that the message in a raised exception matches a regexp.

Args:

- `expected_exception`: Exception class expected to be raised.
- `expected_regexp`: Regexp (re pattern object or string) expected to be found in error message.
- `callable_obj`: Function to be called.
- `args`: Extra args.
- `kwargs`: Extra kwargs.

assertRaisesWithPredicateMatch

```
assertRaisesWithPredicateMatch(
    *args,
    **kwds
)
```

Returns a context manager to enclose code expected to raise an exception.

If the exception is an OpError, the op stack is also included in the message predicate search.

Args:

- `exception_type`: The expected type of exception that should be raised.
- `expected_err_re_or_predicate`: If this is callable, it should be a function of one argument that inspects the passed-in exception and returns True (success) or False (please fail the test). Otherwise, the error message is expected to match this regular expression partially.

Returns:

A context manager to surround code that is expected to raise an exception.

assertRegexpMatches

```
assertRegexpMatches(
    text,
    expected_regexp,
    msg=None
)
```

Fail the test unless the text matches the regular expression.

assertSequenceEqual

```
assertSequenceEqual(
    seq1,
    seq2,
    msg=None,
    seq_type=None
)
```

An equality assertion for ordered sequences (like lists and tuples).

For the purposes of this function, a valid ordered sequence type is one which can be indexed, has a length, and has an equality operator.

Args:

- `seq1` : The first sequence to compare.
- `seq2` : The second sequence to compare.
- `seq_type` : The expected datatype of the sequences, or None if no datatype should be enforced.
- `msg` : Optional message to use on failure instead of a list of differences.

`assertSetEqual`

```
assertSetEqual(
    set1,
    set2,
    msg=None
)
```

A set-specific equality assertion.

Args:

- `set1` : The first set to compare.
- `set2` : The second set to compare.
- `msg` : Optional message to use on failure instead of a list of differences.

assertSetEqual uses ducktyping to support different types of sets, and is optimized for sets specifically (parameters must support a difference method).

`assertShapeEqual`

```
assertShapeEqual(
    np_array,
    tf_tensor
)
```

Asserts that a Numpy ndarray and a TensorFlow tensor have the same shape.

Args:

- `np_array` : A Numpy ndarray or Numpy scalar.
- `tf_tensor` : A Tensor.

Raises:

- `TypeError` : If the arguments have the wrong type.

`assertStartsWith`

```
assertStartsWith(
    actual,
    expected_start,
    msg=None
)
```

Assert that actual.startswith(expected_start) is True.

Args:

- `actual` : str
- `expected_start` : str
- `msg` : Optional message to report on failure.

assertTrue

```
assertTrue(
    expr,
    msg=None
)
```

Check that the expression is true.

assertTupleEqual

```
assertTupleEqual(
    tuple1,
    tuple2,
    msg=None
)
```

A tuple-specific equality assertion.

Args:

- `tuple1` : The first tuple to compare.
- `tuple2` : The second tuple to compare.
- `msg` : Optional message to use on failure instead of a list of differences.

assert_

```
assert_(
    expr,
    msg=None
)
```

Check that the expression is true.

checkedThread

```
checkedThread(
    target,
    args=None,
    kwargs=None
)
```

Returns a Thread wrapper that asserts 'target' completes successfully.

This method should be used to create all threads in test cases, as otherwise there is a risk that a thread will silently fail,

and/or assertions made in the thread will not be respected.

Args:

- `target`: A callable object to be executed in the thread.
- `args`: The argument tuple for the target invocation. Defaults to ().
- `kwargs`: A dictionary of keyword arguments for the target invocation. Defaults to {}.

Returns:

A wrapper for threading.Thread that supports start() and join() methods.

`countTestCases`

```
countTestCases()
```

`debug`

```
debug()
```

Run the test without collecting errors in a TestResult

`defaultTestResult`

```
defaultTestResult()
```

`doCleanups`

```
doCleanups()
```

Execute all cleanup functions. Normally called for you after tearDown.

`evaluate`

```
evaluate(tensors)
```

Evaluates tensors and returns numpy values.

Args:

- `tensors`: A Tensor or a nested list/tuple of Tensors.

Returns:

tensors numpy values.

`fail`

```
fail(msg=None)
```

Fail immediately, with the given message.

failIf

```
failIf(
    *args,
    **kwargs
)
```

failIfAlmostEqual

```
failIfAlmostEqual(
    *args,
    **kwargs
)
```

failIfEqual

```
failIfEqual(
    *args,
    **kwargs
)
```

failUnless

```
failUnless(
    *args,
    **kwargs
)
```

failUnlessAlmostEqual

```
failUnlessAlmostEqual(
    *args,
    **kwargs
)
```

failUnlessEqual

```
failUnlessEqual(
    *args,
    **kwargs
)
```

failUnlessRaises

```
failUnlessRaises(
    *args,
    **kwargs
)
```

get_temp_dir

```
get_temp_dir()
```

Returns a unique temporary directory for the test to use.

If you call this method multiple times during in a test, it will return the same folder. However, across different runs the directories will be different. This will ensure that across different runs tests will not be able to pollute each others environment. If you need multiple unique directories within a single test, you should use tempfile.mkdtemp as follows: tempfile.mkdtemp(dir=self.get_temp_dir()):

Returns:

string, the path to the unique temporary directory created for this test.

id

```
id()
```

run

```
run(result=None)
```

setUp

```
setUp()
```

setUpClass

```
setUpClass(cls)
```

Hook method for setting up class fixture before running tests in the class.

shortDescription

```
shortDescription()
```

Returns a one-line description of the test, or None if no description has been provided.

The default implementation of this method returns the first line of the specified test method's docstring.

skipTest

```
skipTest(reason)
```

Skip this test.

tearDown

```
tearDown()
```

tearDownClass

```
tearDownClass(cls)
```

Hook method for deconstructing the class fixture after running all tests in the class.

test_session

```
test_session(
    *args,
    **kwds
)
```

Returns a TensorFlow Session for use in executing tests.

This method should be used for all functional tests.

This method behaves different than session.Session: for performance reasons `test_session` will by default (if `graph` is None) reuse the same session across tests. This means you may want to either call the function `reset_default_graph()` before tests, or if creating an explicit new graph, pass it here (simply setting it with `as_default()` won't do it), which will trigger the creation of a new session.

Use the `use_gpu` and `force_gpu` options to control where ops are run. If `force_gpu` is True, all ops are pinned to `/device:GPU:0`. Otherwise, if `use_gpu` is True, TensorFlow tries to run as many ops on the GPU as possible. If both `force_gpu and` use_gpu` are False, all ops are pinned to the CPU.

Example:

```
class MyOperatorTest(test_util.TensorFlowTestCase):
  def testMyOperator(self):
    with self.test_session(use_gpu=True):
      valid_input = [1.0, 2.0, 3.0, 4.0, 5.0]
      result = MyOperator(valid_input).eval()
      self.assertEqual(result, [1.0, 2.0, 3.0, 5.0, 8.0]
      invalid_input = [-1.0, 2.0, 7.0]
      with self.assertRaisesOpError("negative input not supported"):
        MyOperator(invalid_input).eval()
```

Args:

- `graph`: Optional graph to use during the returned session.
- `config`: An optional config_pb2.ConfigProto to use to configure the session.
- `use_gpu`: If True, attempt to run as many ops as possible on GPU.
- `force_gpu`: If True, pin all ops to `/device:GPU:0`.

Returns:

A Session object that should be used as a context manager to surround the graph building and execution code in a test case.

Class Members

`longMessage`

`maxDiff`

Last updated November 2, 2017.