# Chapter_14_Going_Deeper_The_Mechanics_of_TensorFlow

March 20, 2024

[1]:
```
import tensorflow as tf
import numpy as np
g = tf.Graph()
## define the computation graph
with g.as_default():
    ## define tensors t1, t2, t3
    t1 = tf.constant(np.pi)
    t2 = tf.constant([1, 2, 3, 4])
    t3 = tf.constant([[1, 2], [3, 4]])

    ## get their ranks
    r1 = tf.rank(t1)
    r2 = tf.rank(t2)
    r3 = tf.rank(t3)

    ## get their shapes
    s1 = t1.get_shape()
    s2 = t2.get_shape()
    s3 = t3.get_shape()
    print('Shapes:', s1, s2, s3)
```

```
C:\Users\ankit19.gupta\OneDrive - Reliance Corporate IT Park Limited\Desktop\Sel
f_Projects\Python_Machine_Learning_Sebastian_Raschka\myenv\lib\site-
packages\tensorflow\python\framework\dtypes.py:458: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\ankit19.gupta\OneDrive - Reliance Corporate IT Park Limited\Desktop\Sel
f_Projects\Python_Machine_Learning_Sebastian_Raschka\myenv\lib\site-
packages\tensorflow\python\framework\dtypes.py:459: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\ankit19.gupta\OneDrive - Reliance Corporate IT Park Limited\Desktop\Sel
f_Projects\Python_Machine_Learning_Sebastian_Raschka\myenv\lib\site-
packages\tensorflow\python\framework\dtypes.py:460: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
```

```
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\ankit19.gupta\OneDrive - Reliance Corporate IT Park Limited\Desktop\Sel
f_Projects\Python_Machine_Learning_Sebastian_Raschka\myenv\lib\site-
packages\tensorflow\python\framework\dtypes.py:461: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\ankit19.gupta\OneDrive - Reliance Corporate IT Park Limited\Desktop\Sel
f_Projects\Python_Machine_Learning_Sebastian_Raschka\myenv\lib\site-
packages\tensorflow\python\framework\dtypes.py:462: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\ankit19.gupta\OneDrive - Reliance Corporate IT Park Limited\Desktop\Sel
f_Projects\Python_Machine_Learning_Sebastian_Raschka\myenv\lib\site-
packages\tensorflow\python\framework\dtypes.py:465: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

```
Shapes: () (4,) (2, 2)
```

[2]:
```
with tf.Session(graph=g) as sess:
    print('Ranks:',r1.eval(),r2.eval(),r3.eval())
```

```
Ranks: 0 1 2
```

[3]:
```
g = tf.Graph()
with g.as_default():
    a = tf.constant(1, name='a')
    b = tf.constant(2, name='b')
    c = tf.constant(3, name='c')
    z = 2*(a-b) + c
```

[4]:
```
with tf.Session(graph=g) as sess:
    print('2*(a-b)+c => ', sess.run(z))
```

```
2*(a-b)+c =>  1
```

[5]:
```
import tensorflow as tf
g = tf.Graph()
with g.as_default():
    tf_a = tf.placeholder(tf.int32, shape=[],name='tf_a')
    tf_b = tf.placeholder(tf.int32, shape=[],name='tf_b')
    tf_c = tf.placeholder(tf.int32, shape=[],name='tf_c')
    r1 = tf_a-tf_b
    r2 = 2*r1
    z = r2 + tf_c
```

[6]:
```
with tf.Session(graph=g) as sess:
    feed = {tf_a: 1,tf_b: 2,tf_c: 3}
    print('z:',sess.run(z, feed_dict=feed))
```

```
z: 1
```

[7]:
```
import tensorflow as tf
g = tf.Graph()
with g.as_default():
    tf_x = tf.placeholder(tf.float32,shape=[None, 2],name='tf_x')
    x_mean = tf.reduce_mean(tf_x,axis=0,name='mean')
```

[8]:
```
import numpy as np
np.random.seed(123)
np.set_printoptions(precision=2)
with tf.Session(graph=g) as sess:
    x1 = np.random.uniform(low=0, high=1,size=(5, 2))
    print('Feeding data with shape ', x1.shape)
    print('Result:', sess.run(x_mean,feed_dict={tf_x: x1}))
    x2 = np.random.uniform(low=0, high=1,size=(10,2))
    print('Feeding data with shape', x2.shape)
    print('Result:', sess.run(x_mean,feed_dict={tf_x: x2}))
```

```
Feeding data with shape  (5, 2)
Result: [0.62 0.47]
Feeding data with shape (10, 2)
Result: [0.46 0.49]
```

[9]:
```
tf_x
```

[9]:
```
<tf.Tensor 'tf_x:0' shape=(?, 2) dtype=float32>
```

[10]:
```
import tensorflow as tf
import numpy as np
g1 = tf.Graph()
with g1.as_default():
    w = tf.Variable(np.array([[1, 2, 3, 4],[5, 6, 7, 8]]), name='w')
    print(w)
```

```
<tf.Variable 'w:0' shape=(2, 4) dtype=int32_ref>
```

[11]:
```
with tf.Session(graph=g1) as sess:
    sess.run(tf.global_variables_initializer())
    print(sess.run(w))
```

```
[[1 2 3 4]
 [5 6 7 8]]
```

[12]:
```
import tensorflow as tf
g2 = tf.Graph()
with g2.as_default():
    w1 = tf.Variable(1, name='w1')
    init_op = tf.global_variables_initializer()
    w2 = tf.Variable(2, name='w2')
```

[13]:
```
with tf.Session(graph=g2) as sess:
    sess.run(init_op)
    print('w1:', sess.run(w1))
```

```
w1: 1
```

[14]:
```
with tf.Session(graph=g2) as sess:
    sess.run(init_op)
    print('w2:', sess.run(w2))
```

```
---------------------------------------------------------------------------
FailedPreconditionError                   Traceback (most recent call last)
~\OneDrive - Reliance Corporate IT Park␣
 ↪Limited\Desktop\Self_Projects\Python_Machine_Learning_Sebastian_Raschka\myenv␣lib\site-pacl
 ↪py in _do_call(self, fn, *args)
   1326     try:
-> 1327       return fn(*args)
   1328     except errors.OpError as e:

~\OneDrive - Reliance Corporate IT Park␣
 ↪Limited\Desktop\Self_Projects\Python_Machine_Learning_Sebastian_Raschka\myenv␣lib\site-pacl
 ↪py in _run_fn(session, feed_dict, fetch_list, target_list, options,␣
 ↪run_metadata)
   1305                                   feed_dict, fetch_list, target_list,
-> 1306                                   status, run_metadata)
   1307

~\OneDrive - Reliance Corporate IT Park␣
 ↪Limited\Desktop\Self_Projects\Python_Machine_Learning_Sebastian_Raschka\myenv␣lib\contextli
 ↪py in __exit__(self, type, value, traceback)
     87             try:
---> 88                 next(self.gen)
     89             except StopIteration:

~\OneDrive - Reliance Corporate IT Park␣
 ↪Limited\Desktop\Self_Projects\Python_Machine_Learning_Sebastian_Raschka\myenv␣lib\site-pacl
 ↪py in raise_exception_on_not_ok_status()
    465           compat.as_text(pywrap_tensorflow.TF_Message(status)),
--> 466           pywrap_tensorflow.TF_GetCode(status))
    467   finally:
```

```
FailedPreconditionError: Attempting to use uninitialized value w2
         [[Node: _retval_w2_0_0 = _Retval[T=DT_INT32, index=0, _device="/job:
 ↪localhost/replica:0/task:0/cpu:0"](w2)]]

During handling of the above exception, another exception occurred:

FailedPreconditionError                   Traceback (most recent call last)
<ipython-input-14-c10f8d6b307c> in <module>
      1 with tf.Session(graph=g2) as sess:
      2     sess.run(init_op)
----> 3     print('w2:', sess.run(w2))

~\OneDrive - Reliance Corporate IT Park␣
 ↪Limited\Desktop\Self_Projects\Python_Machine_Learning_Sebastian_Raschka\myenv\lib\site-pack
 ↪py in run(self, fetches, feed_dict, options, run_metadata)
    893     try:
    894       result = self._run(None, fetches, feed_dict, options_ptr,
--> 895                          run_metadata_ptr)
    896       if run_metadata:
    897         proto_data = tf_session.TF_GetBuffer(run_metadata_ptr)

~\OneDrive - Reliance Corporate IT Park␣
 ↪Limited\Desktop\Self_Projects\Python_Machine_Learning_Sebastian_Raschka\myenv\lib\site-pack
 ↪py in _run(self, handle, fetches, feed_dict, options, run_metadata)
   1122     if final_fetches or final_targets or (handle and feed_dict_tensor):
   1123       results = self._do_run(handle, final_targets, final_fetches,
-> 1124                              feed_dict_tensor, options, run_metadata)
   1125     else:
   1126       results = []

~\OneDrive - Reliance Corporate IT Park␣
 ↪Limited\Desktop\Self_Projects\Python_Machine_Learning_Sebastian_Raschka\myenv\lib\site-pack
 ↪py in _do_run(self, handle, target_list, fetch_list, feed_dict, options,␣
 ↪run_metadata)
   1319     if handle is None:
   1320       return self._do_call(_run_fn, self._session, feeds, fetches,␣
 ↪targets,
-> 1321                            options, run_metadata)
   1322     else:
   1323       return self._do_call(_prun_fn, self._session, handle, feeds,␣
 ↪fetches)

~\OneDrive - Reliance Corporate IT Park␣
 ↪Limited\Desktop\Self_Projects\Python_Machine_Learning_Sebastian_Raschka\myenv\lib\site-pack
 ↪py in _do_call(self, fn, *args)
   1338         except KeyError:
   1339           pass
```

```
-> 1340       raise type(e)(node_def, op, message)
   1341
   1342   def _extend_graph(self):

FailedPreconditionError: Attempting to use uninitialized value w2
         [[Node: _retval_w2_0_0 = _Retval[T=DT_INT32, index=0, _device="/job:
 ↪localhost/replica:0/task:0/cpu:0"](w2)]]
```

[ ]:
```
import tensorflow as tf
g = tf.Graph()
with g.as_default():
    with tf.variable_scope('net_A'):
        with tf.variable_scope('layer-1'):
            w1 = tf.Variable(tf.random_normal(shape=(10,4)), name='weights')
        with tf.variable_scope('layer-2'):
            w2 = tf.Variable(tf.random_normal(shape=(20,10)), name='weights')
    with tf.variable_scope('net_B'):
        with tf.variable_scope('layer-1'):
            w3 = tf.Variable(tf.random_normal(shape=(10,4)), name='weights')
print(w1)
print(w2)
print(w3)
```

[ ]:
```
import tensorflow as tf
    ############################
    ## Helper functions ##
    ############################
def build_classifier(data, labels, n_classes=2):
    data_shape = data.get_shape().as_list()
    weights = tf.
 ↪get_variable(name='weights',shape=(data_shape[1],n_classes),dtype=tf.float32)
    bias = tf.get_variable(name='bias',initializer=tf.zeros(shape=n_classes))
    logits = tf.add(tf.matmul(data, weights),bias,name='logits')
    return logits, tf.nn.softmax(logits)

def build_generator(data, n_hidden):
    data_shape = data.get_shape().as_list()
    w1 = tf.Variable(tf.random_normal(shape=(data_shape[1],n_hidden)),name='w1')
    b1 = tf.Variable(tf.zeros(shape=n_hidden),name='b1')
    hidden = tf.add(tf.matmul(data, w1), b1,name='hidden_pre-activation')
    hidden = tf.nn.relu(hidden, 'hidden_activation')
    w2 = tf.Variable(tf.random_normal(shape=(n_hidden,data_shape[1])),name='w2')
    b2 = tf.Variable(tf.zeros(shape=data_shape[1]),name='b2')
    output = tf.add(tf.matmul(hidden, w2), b2,name = 'output')
    return output, tf.nn.sigmoid(output)
```

```
    ############################
    ## Build the graph ##
    ############################

batch_size=64
g = tf.Graph()

with g.as_default():
    tf_X = tf.placeholder(shape=(batch_size, 100),dtype=tf.float32,name='tf_X')
    ## build the generator
    with tf.variable_scope('generator'):
        gen_out1 = build_generator(data=tf_X,n_hidden=50)
    ## build the classifier
    with tf.variable_scope('classifier') as scope:
        ## classifier for the original data:
        cls_out1 = build_classifier(data=tf_X,labels=tf.ones(shape=batch_size))
        ## reuse the classifier for generated data
        scope.reuse_variables()
        cls_out2 = build_classifier(data=gen_out1[1],labels=tf.
 ↪zeros(shape=batch_size))
```

[ ]:
```
g = tf.Graph()
with g.as_default():
    tf_X = tf.placeholder(shape=(batch_size, 100),dtype=tf.float32,name='tf_X')
    ## build the generator
    with tf.variable_scope('generator'):
        gen_out1 = build_generator(data=tf_X,n_hidden=50)
    ## build the classifier
    with tf.variable_scope('classifier'):
        ## classifier for the original data:
        cls_out1 = build_classifier(data=tf_X,labels=tf.ones(shape=batch_size))
    with tf.variable_scope('classifier', reuse=True):
        ## reuse the classifier for generated data
        cls_out2 = build_classifier(data=gen_out1[1],labels=tf.
 ↪zeros(shape=batch_size))
```

[ ]:
```
import tensorflow as tf
import numpy as np
g = tf.Graph()
with g.as_default():
    tf.set_random_seed(123)
    ## placeholders
    tf_x = tf.placeholder(shape=(None),dtype=tf.float32,name='tf_x')
    tf_y = tf.placeholder(shape=(None),dtype=tf.float32,name='tf_y')

    ## define the variable (model parameters)
```

```
    weight = tf.Variable(tf.random_normal(shape=(1, 1),stddev=0.
↪25),name='weight')
    bias = tf.Variable(0.0, name='bias')
    ## build the model
    y_hat = tf.add(weight * tf_x, bias,name='y_hat')
    ## compute the cost
    cost = tf.reduce_mean(tf.square(tf_y - y_hat),name='cost')

    ## train the model
    optim = tf.train.GradientDescentOptimizer(learning_rate=0.001)
    train_op = optim.minimize(cost, name='train_op')
```

[ ]:
```
## create a random toy dataset for regression
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(0)
def make_random_data():
    x = np.random.uniform(low=-2, high=4, size=200)
    y = []
    for t in x:
        r = np.random.normal(loc=0.0,scale=(0.5 + t*t/3),size=None)
        y.append(r)
    return x, 1.726*x -0.84 + np.array(y)
x, y = make_random_data()
plt.plot(x, y, 'o')
plt.show()
```

[ ]:
```
## train/test splits
x_train, y_train = x[:100], y[:100]
x_test, y_test = x[100:], y[100:]
n_epochs = 500
training_costs = []
with tf.Session(graph=g) as sess:
    sess.run(tf.global_variables_initializer())
    ## train the model for n_epochs
    for e in range(n_epochs):
        c, _ = sess.run([cost, train_op],feed_dict={tf_x: x_train,tf_y:␣
↪y_train})
        training_costs.append(c)
        if not e % 50:
            print('Epoch %4d: %.4f' % (e, c))
```

[ ]:
```
plt.plot(training_costs)
plt.show()
```

[ ]:
```
n_epochs = 500
training_costs = []
```

```
with tf.Session(graph=g) as sess:
    ## first, run the variables initializer
    sess.run(tf.global_variables_initializer())
    ## train the model for n_eopchs
    for e in range(n_epochs):
        c, _ = sess.run(['cost:0', 'train_op'],feed_dict={'tf_x:0':
 ↪x_train,'tf_y:0':y_train})
        training_costs.append(c)
        if e%50 == 0:
            print('Epoch {:4d} : {:.4f}'.format(e, c))
```

[ ]:
```
with g.as_default():
    saver = tf.train.Saver()
```

[ ]:
```
n_epochs = 500
training_costs = []
with tf.Session(graph=g) as sess:
    sess.run(tf.global_variables_initializer())
    ## train the model for n_epochs
    for e in range(n_epochs):
        c, _ = sess.run([cost, train_op],feed_dict={tf_x: x_train,tf_y:␣
 ↪y_train})
        training_costs.append(c)
        if not e % 50:
            print('Epoch %4d: %.4f' % (e, c))
    saver.save(sess, './trained-model')
```

[ ]:
```
with tf.Session() as sess:
    new_saver = tf.train.import_meta_graph('./trained-model.meta')
```

[ ]:
```
import tensorflow as tf
import numpy as np
g2 = tf.Graph()
with tf.Session(graph=g2) as sess:
    new_saver = tf.train.import_meta_graph('./trained-model.meta')
    new_saver.restore(sess, './trained-model')
    y_pred = sess.run('y_hat:0',feed_dict={'tf_x:0': x_test})
```

[ ]:
```
import matplotlib.pyplot as plt
x_arr = np.arange(-2, 4, 0.1)
g2 = tf.Graph()
with tf.Session(graph=g2) as sess:
    new_saver = tf.train.import_meta_graph('./trained-model.meta')
    new_saver.restore(sess, './trained-model')
    y_arr = sess.run('y_hat:0',feed_dict={'tf_x:0' : x_arr})
plt.figure()
plt.plot(x_train, y_train, 'bo')
```

```
plt.plot(x_test, y_test, 'bo', alpha=0.3)
plt.plot(x_arr, y_arr.T[:, 0], '-r', lw=3)
plt.show()
```

[ ]:
```
import tensorflow as tf
import numpy as np
g = tf.Graph()
with g.as_default():
    arr = np.array([[1., 2., 3., 3.5],[4., 5., 6., 6.5],[7., 8., 9., 9.5]])
    T1 = tf.constant(arr, name='T1')
    print(T1)
    s = T1.get_shape()
    print('Shape of T1 is', s)
    T2 = tf.Variable(tf.random_normal(shape=s))
    print(T2)
    T3 = tf.Variable(tf.random_normal(shape=(s.as_list()[0],)))
    print(T3)
```

[ ]:
```
with g.as_default():
    T4 = tf.reshape(T1, shape=[1, 1, -1],name='T4')
    print(T4)
    T5 = tf.reshape(T1, shape=[1, 3, -1],name='T5')
    print(T5)
```

[ ]:
```
with tf.Session(graph = g) as sess:
    print(sess.run(T4))
    print()
    print(sess.run(T5))
```

[ ]:
```
with g.as_default():
    T6 = tf.transpose(T5, perm=[2, 1, 0],name='T6')
    print(T6)
    T7 = tf.transpose(T5, perm=[0, 2, 1],name='T7')
    print(T7)
```

[ ]:
```
with g.as_default():
    t5_splt = tf.split(T5,num_or_size_splits=2,axis=2, name='T8')
    print(t5_splt)
```

[ ]:
```
g = tf.Graph()
with g.as_default():
    t1 = tf.ones(shape=(5, 1),dtype=tf.float32, name='t1')
    t2 = tf.zeros(shape=(5, 1),dtype=tf.float32, name='t2')
    print(t1)
    print(t2)
with g.as_default():
    t3 = tf.concat([t1, t2], axis=0, name='t3')
```

```
    print(t3)
    t4 = tf.concat([t1, t2], axis=1, name='t4')
    print(t4)
```

[ ]:
```
with tf.Session(graph=g) as sess:
    print(t3.eval())
    print()
    print(t4.eval())
```

[ ]:
```
import tensorflow as tf
x, y = 1.0, 2.0
g = tf.Graph()
with g.as_default():
    tf_x = tf.placeholder(dtype=tf.float32,shape=None, name='tf_x')
    tf_y = tf.placeholder(dtype=tf.float32,shape=None, name='tf_y')
    if x < y:
        res = tf.add(tf_x, tf_y, name='result_add')
    else:
        res = tf.subtract(tf_x, tf_y, name='result_sub')
    print('Object:', res)
with tf.Session(graph=g) as sess:
    print('x < y: %s -> Result:' % (x < y),res.eval(feed_dict={'tf_x:0':␣
 ↪x,'tf_y:0': y}))
    x, y = 2.0, 1.0
    print('x < y: %s -> Result:' % (x < y),res.eval(feed_dict={'tf_x:0':␣
 ↪x,'tf_y:0': y}))
```

[ ]:
```
import tensorflow as tf
x, y = 1.0, 2.0
g = tf.Graph()
with g.as_default():
    tf_x = tf.placeholder(dtype=tf.float32,shape=None, name='tf_x')
    tf_y = tf.placeholder(dtype=tf.float32,shape=None, name='tf_y')
    res = tf.cond(tf_x < tf_y,lambda: tf.add(tf_x,␣
 ↪tf_y,name='result_add'),lambda: tf.subtract(tf_x, tf_y,name='result_sub'))
    print('Object:', res)
with tf.Session(graph=g) as sess:
    print('x < y: %s -> Result:' % (x < y),res.eval(feed_dict={'tf_x:0':␣
 ↪x,'tf_y:0': y}))
    x, y = 2.0, 1.0
    print('x < y: %s -> Result:' % (x < y),res.eval(feed_dict={'tf_x:0':␣
 ↪x,'tf_y:0': y}))
```

[ ]:
```
f1 = lambda: tf.constant(1)
f2 = lambda: tf.constant(0)
result = tf.case([(tf.less(x, y), f1)], default=f2)
```

[ ]:
```
result
```

[ ]:
```
i = tf.constant(0)
threshold = 100
c = lambda i: tf.less(i, 100)
b = lambda i: tf.add(i, 1)
r = tf.while_loop(cond=c, body=b, loop_vars=[i])
```

[ ]:
```
batch_size=64
g = tf.Graph()
with g.as_default():
    tf_X = tf.placeholder(shape=(batch_size, 100),dtype=tf.float32,name='tf_X')
    ## build the generator
    with tf.variable_scope('generator'):
        gen_out1 = build_generator(data=tf_X,n_hidden=50)
    ## build the classifier
    with tf.variable_scope('classifier') as scope:
        ## classifier for the original data:
        cls_out1 = build_classifier(data=tf_X,labels=tf.ones(shape=batch_size))
    ## reuse the classifier for generated data
    scope.reuse_variables()
    cls_out2 = build_classifier(data=gen_out1[1],labels=tf.
 ↪zeros(shape=batch_size))
```

[ ]:
```
with tf.Session(graph=g) as sess:
    sess.run(tf.global_variables_initializer())
    file_writer = tf.summary.FileWriter(logdir='./logs/', graph=g)
```

[ ]:
```
# ! tensorboard --logdir logs/
```

[ ]:
```

```