UC Berkeley Extension

Machine Learning With TensorFlow

X433.7-001 (2 semester units in COMPSCI)

Instructor Alexander I. Iliev, Ph.D.

Course Content Outline

- **Machine Learning With TensorFlow®**
- Introduction, Python - pros and cons
- Python modules, DL packages and scientific blocks
- Working with the shell, IPyton and the editor
- Installing the environment with core packages
- Writing “Hello World” HW1 (10pts)

- **Tensorflow and TensorBoard basics (cont.)**
- Ecosystem, Competition, Users
- Linear algebra recap
- Data types in Numpy and Tensorflow
- Basic operations in Tensorflow
- Graph models and structures with Tensorboard

- **TensorFlow operations**
- Sessions, graphs, variables, placeholders
- Overloaded operators
- Using Aliases

- **TF 2.0 vs. 1.X comparison**
- Name scopes HW2 (10pts)

- **TF 1.x vs. 2.0, Machine Learning concepts**
- Upgrading, Migrating 1.x to 2.0
- Machine Learning Models, k-Means

Upgrading Tensorflow 1.x to Tensorflow 2.0

- The " pip install TensorFlow " command is used in the command prompt to install Tensorflow in the current environment *

```
In [2]: !pip show tensorflow
Name: tensorflow
Version: 1.14.0
Summary: TensorFlow is an open source machine learning framework for everyone.
Home-page: https://www.tensorflow.org/
Author: Google Inc.
Author-email: packages@tensorflow.org
License: Apache 2.0
Location: c:\programdata\anaconda3\envs\tensorflow 1.14\lib\site-packages
Requires: six, google-pasta, astor, tensorboard, termcolor, keras-preprocessing, tensorflow-estimator,
protobuf, wheel, wrapt, grpcio, absl-py, gast, keras-applications, numpy
Required-by:

In [3]:
```

- In order to upgrade to Tensorflow 2.0, we need to run the following upgrade script in the command prompt or the Ipython console/terminal:

>>> pip install –upgrade tensorflow==2.0.0-rc1

Upgrading Tensorflow 1.x to Tensorflow 2.0

- After running the script to update Tensorflow, the version will be updated to TF 2.0 for that particular environment. The following message will be displayed after successfully updating.

- Now, we use the "!pip show tensorflow" command to view the current version. As we can see, the version is '2.0.0rc1'. Hence, our upgrade was successful.

```
In [1]: !pip show tensorflow
Name: tensorflow
Version: 2.0.0rc1
Summary: TensorFlow is an open source machine learning framework for everyone.
Home-page: https://www.tensorflow.org/
Author: Google Inc.
Author-email: packages@tensorflow.org
License: Apache 2.0
Location: c:\programdata\anaconda3\envs\tensorflow_env\lib\site-packages
Requires: absl-py, keras-applications, google-pasta, tf-estimator-nightly, gast, six, grpcio, astor, keras-
preprocessing, wrapt, protobuf, opt-einsum, tb-nightly, numpy, termcolor, wheel
```

Migrating a code from TF 1.x to TF 2.0

While migrating from TF 1.x to 2.0, there can be three types of codes:

1. Codes that have functions which have the same syntax in TF 1.x and TF 2.0. Such codes do not require any change, while migrating to 2.0.
   - For ex: tf.concat(), tf.case()

2. Codes that have functions which have been deprecated in TF 2.0. Such codes require a special attribute to be added to make them compatible with TF 2.0, while migrating.
   - For ex: tf.placeholder(), tf.variable_scope(), tf.Session()

3. Codes that have functions whose syntax have changed in TF 2.0. Such codes require an upgrade script to be run in the background which automatically upgrades certain functions in the code to TF 2.0 standards.
   - For ex: tf.arg_max(), tf.batch_to_space()

Different methods for migrating TF 1.x to 2.0

We can make the code in TF 1.x compatible to TF 2.0 in two different ways

- **tf.compat.v1 :**
  - 'tf.compat.v1' is used in the place of 'tf' to make the functions that have been deprecated or incompatible with TF 2.0, compatible:

```
# to make this code compatible in TF 2.0, we make the following changes:
import tensorflow.compat.v1 as tf
```

  - For ex: tf.placeholder() will be written as tf.compat.v1.placeholder()
  - In some cases, TF 2.0 functionalities such as Eager Execution needs to be turned off. This can be done using tf.disable_v2_behavior()

- **Running the upgrade script:**

The script to upgrade should be run in the command prompt. It is as follows:

```
!tf_upgrade_v2 \
--infile filename_ip.py \
--outfile filename_op.py
```

- In order to run this script, we need to preinstall nightly functions. * This can be done with:
  >>> pip install tf-nightly-2.0-preview
- By doing the above, our code will automatically be made compatible to TF 2.0

1. Codes that have functions which have the same syntax in TF 1.x and TF 2.0

Code example:

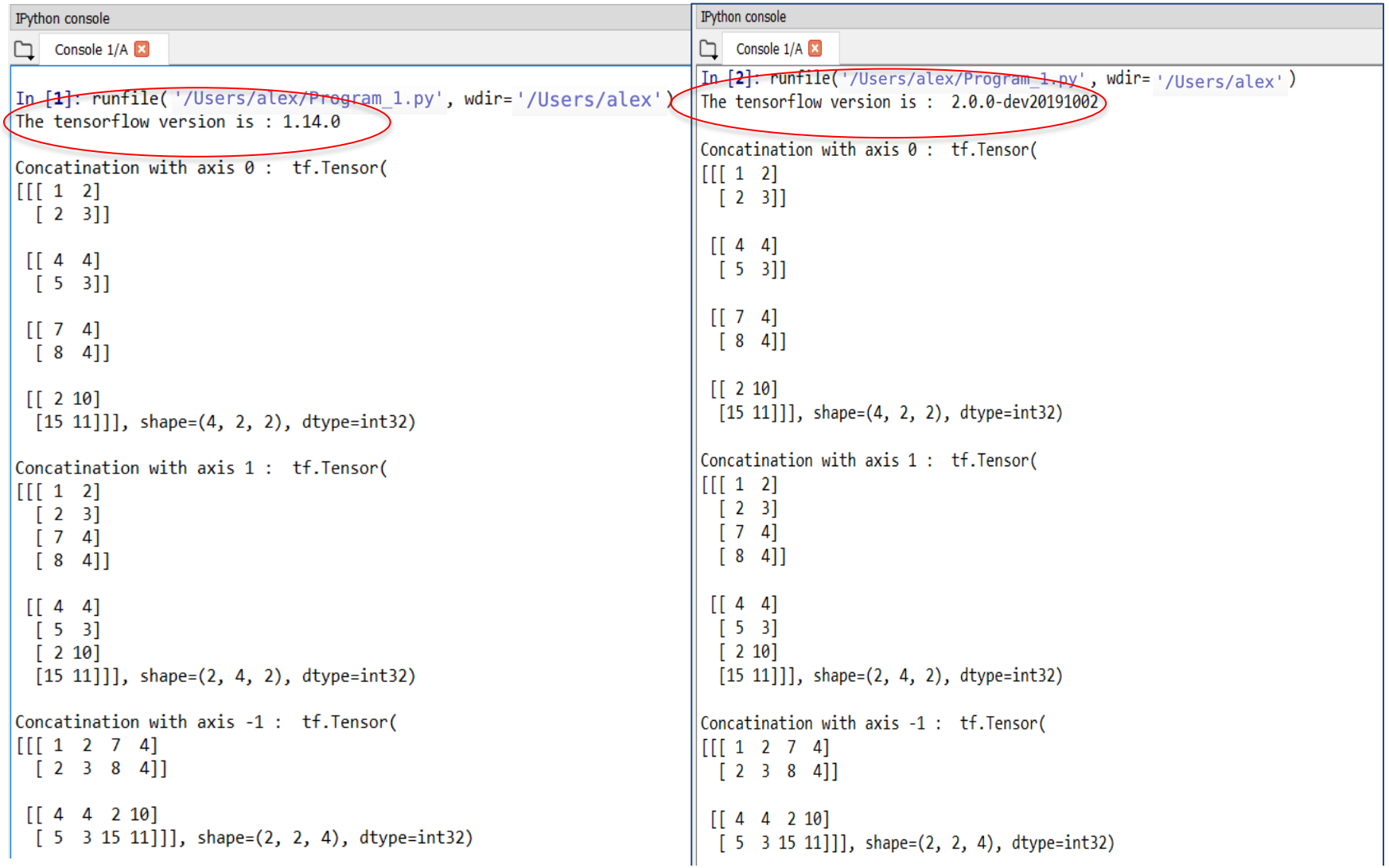
Program_1.py | Program_2.py | Program_3.py | Program_4.py | Program_5.py | Program_6.py

```
# -*- coding: utf-8 -*-
"""
@author: ailiev
"""
"""This program is used to demonstrate the working of Tensorflow functions
that work in both TF 1.x and TF 2.0 and do not require to be changed.
We demonstrate this by running the tf.concat() function to concatinate
two tensors using different axis"""

import tensorflow as tf # first we import Tensorflow


 #here, we define two tensors
tensor1 = [[[1, 2], [2, 3]], [[4, 4], [5, 3]]]
tensor2 = [[[7, 4], [8, 4]], [[2, 10], [15, 11]]]
# This line will display the version of TF
print('The tensorflow version is : 1.14.0','\n')

# This line will concatinate the two tensors with axis as '0' & display it
print('Concatination with axis 0 : ',tf.concat([tensor1, tensor2], 0),'\n')

 # This line will concatinate the two tensors with axis as '0' & display it
print('Concatination with axis 1 : ',tf.concat([tensor1, tensor2], 1),'\n')

# This line will concatinate the two tensors with axis as '0' & display it
print('Concatination with axis -1 : ',tf.concat([tensor1, tensor2], -1),'\n')
z = tf.ones(shape=(1))
print (z)
```

This example illustrates the use of tf.concat() to concatenate two different tensors based on the axis we specify. Functions as such can be used in the same format in both versions.

try in class…

Output:


IPython console
Console 1/A
In [1]: runfile('/Users/alex/Program_1.py', wdir='/Users/alex')
The tensorflow version is : 1.14.0
Concatination with axis 0 : tf.Tensor(
[[[1 2]
[2 3]]
[[4 4]
[5 3]]
[[7 4]
[8 4]]
[[2 10]
[15 11]]], shape=(4, 2, 2), dtype=int32)
Concatination with axis 1 : tf.Tensor(
[[[1 2]
[2 3]
[7 4]
[8 4]]
[[4 4]
[5 3]
[2 10]
[15 11]]], shape=(2, 4, 2), dtype=int32)
Concatination with axis -1 : tf.Tensor(
[[[1 2 7 4]
[2 3 8 4]]
[[4 4 2 10]
[5 3 15 11]]], shape=(2, 2, 4), dtype=int32)
IPython console
Console 1/A
In [2]: runfile('/Users/alex/Program_1.py', wdir='/Users/alex')
The tensorflow version is : 2.0.0-dev20191002
Concatination with axis 0 : tf.Tensor(
[[[1 2]
[2 3]]
[[4 4]
[5 3]]
[[7 4]
[8 4]]
[[2 10]
[15 11]]], shape=(4, 2, 2), dtype=int32)
Concatination with axis 1 : tf.Tensor(
[[[1 2]
[2 3]
[7 4]
[8 4]]
[[4 4]
[5 3]
[2 10]
[15 11]]], shape=(2, 4, 2), dtype=int32)
Concatination with axis -1 : tf.Tensor(
[[[1 2 7 4]
[2 3 8 4]]
[[4 4 2 10]
[5 3 15 11]]], shape=(2, 2, 4), dtype=int32)

2. Codes that have functions which have been deprecated in TF 2.0

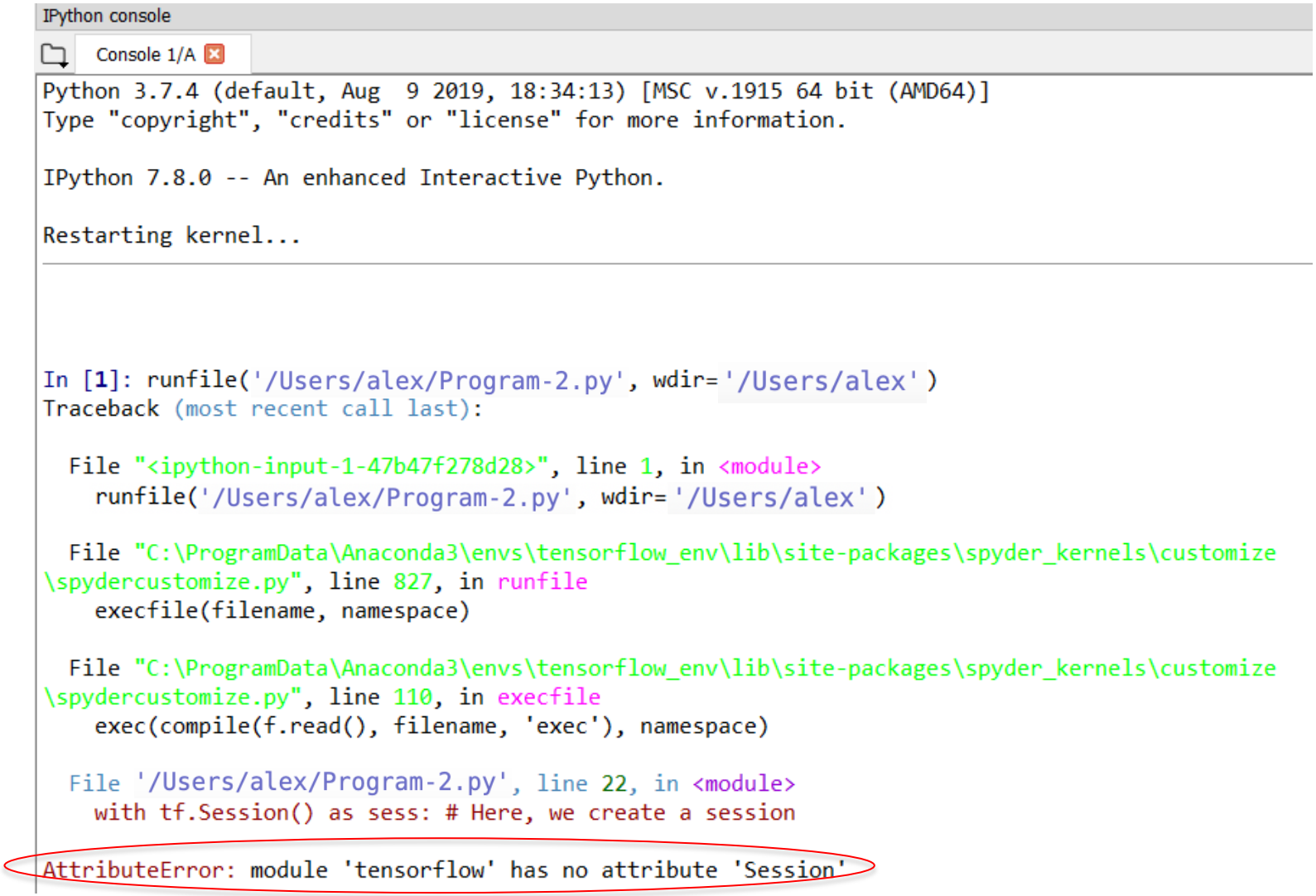
Code example with tf.Session():

Program_1.py | Program_2.py | Program_3.py | Program_4.py | Program_5.py | Program_6.py

```
# -*- coding: utf-8 -*-
"""
@author: ailiev
"""
"""This program is used to demonstrate the working of Tensorflow functions
that are removed from TF 2.0 and require to be made compatible.
We demonstrate this by running the tf.session() function """

import tensorflow as tf # first we import Tensorflow

#First, we create 2 random tensors with shape [3,3]
x = tf.random.normal(shape=(3,3))
y = tf.random.normal(shape=(3,3))

#Here, we create a tensor of 1
z = tf.ones(shape=(1))

result = x + y * z #we run an arithmetic operation on the three tensors

with tf.Session() as sess: # Here, we create a session
    print(sess.run(result)) # we run the session with parameter 'result',print result
""" This code will run on TF 1.x but will return an Attribute error in
    TF 2.0 because tf.session() is deprecated in 2.0, hence we need to make the code
    compatible"""
```

try in class…

Output for TF 2.0:



Code example of tf.Session() made compatible with TF 2.0:

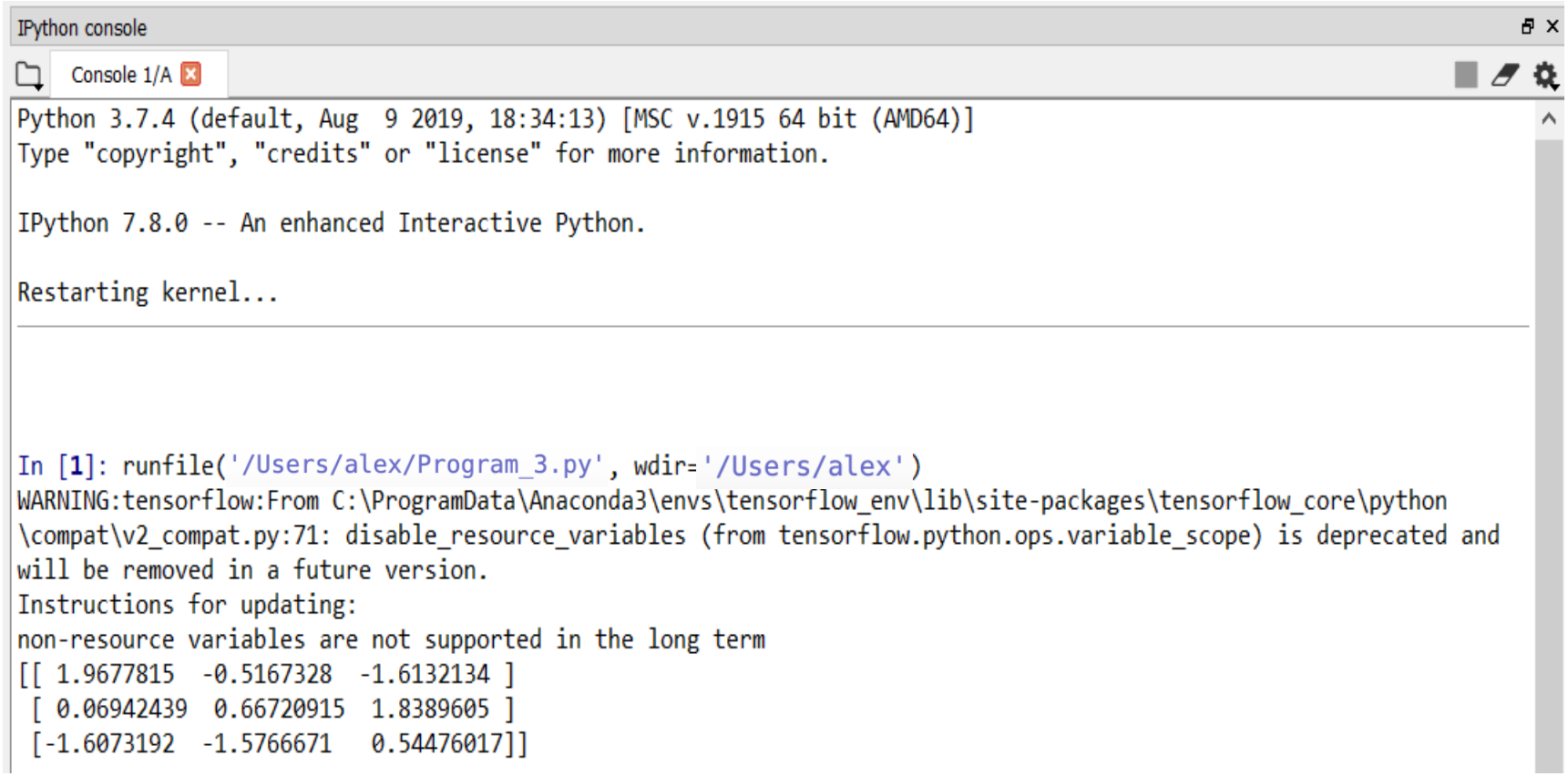
Program_1.py | Program_2.py | Program_3.py | Program_4.py | Program_5.py | Program_6.py

```
# -*- coding: utf-8 -*-
"""
@author: ailiev
"""

# to make this code compatible in TF 2.0, we make the following changes:
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
 # and then we run the same code :
 #First, we create 2 random tensors with shape [3,3]
x = tf.random.normal(shape=(3,3))
y = tf.random.normal(shape=(3,3))

#Here, we create a tensor of 1
z = tf.ones(shape=(1))

result = x + y * z #we run an arithmetic operation on the three tensors

with tf.Session() as sess: # Here, we create a session
    print(sess.run(result)) # we run the session with parameter 'result',print result
""" Now the code will run on TF 2.0"""
```

- The tf.compat.v1 makes the code compatible with TF 2.0 i.e., the code is run in the TF 2.0 environment as TF 1.x code.
- We import tensorflow.compat.v1 as tf so that whenever the tf is used in the code the following function gets compatible for TF 2.0.
- In addition, tf.disable_v2_behavior() is used so that a virtual environment of TF 1.x is created by stopping the behaviour of TF 2.0.

try in class...

Upgraded Output for TF 2.0:



Constant and Placeholder

- Constant
  - The name speaks for itself; constants don't change
  - They create a node that takes a value

  - **SYNTAX**
    - tf.constant(6, tf.int32, name='A')

- Placeholder
  - Placeholders are used to feed in future values at runtime
  - Used when graph depends on external data

  - **SYNTAX**
    - tf.compat.v1.placeholder(tf.int32, shape=[3], name='B')

Example: placeholder

- Syntax: tf.placeholder(dtype , shape=None , name=None)
- Placeholders are not compatible with eager execution

```
1  import tensorflow as tf
2
3  g2= tf.Graph()          # define a default graph which is passed to session
4  with g2.as_default():   # use defined graph as default graph
5      a = tf.compat.v1.placeholder(tf.float32)  # Declaring placeholder of type float
6      b = a * 2
7
8
9  with tf.compat.v1.Session(graph=g2) as sess:
10     dictionary = {a:[[1,2,3],[4,5,6]]}
11     result = sess.run(b,feed_dict = dictionary)# passing dictionary to placeholder
12     print(result)
```

Output:

[12] [[2. 4. 6.]
 8. 10. 12.]]

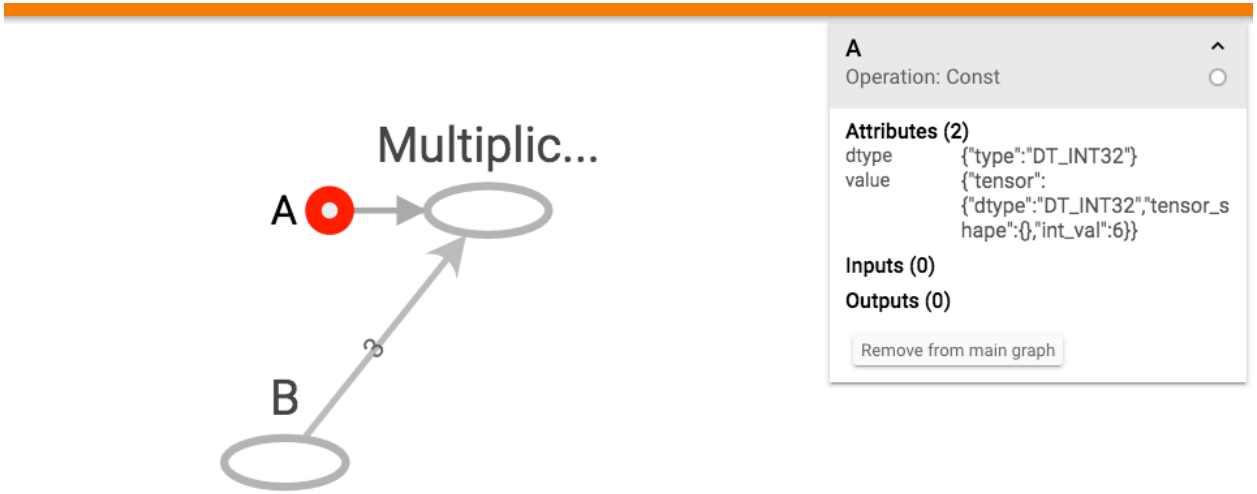
Example: constant and placeholder

```
import tensorflow as tf
tf.compat.v1.disable_eager_execution() # disable eager execution
# TensorFlow's eager execution is an imperative programming environment that
# evaluates operations immediately, without building graphs: operations return
# concrete values instead of constructing a computational graph to run later


g2= tf.Graph()
with g2.as_default():
    a = tf.constant(6, tf.int32, name='A') # the name speaks for itself,
                                           #Constants are used as constants.
                                           # They create a node that takes value and it does not change


    b = tf.compat.v1.placeholder(tf.int32, shape=[3], name='B') #  placeholders are used to
                                                                #  feed in future values at runtime
                                                                #  Used when graph depends on external dat
    c = tf.multiply(a, b, name="Multiplication")

with tf.compat.v1.Session(graph=g2) as sess:
    writer = tf.compat.v1.summary.FileWriter( '/Users/Alex/logs/summaries', sess.graph)
    # create a dictionary:
    d = {b: [1, 2, 3]}
    # feed it to the placeholder
    print(sess.run(c, feed_dict=d))
    writer.close()



#OUTPUT ------   [6, 12, 18]
```

Example: constant and placeholder

Placeholder node in Tensorboard graph

Constant node in Tensorboard

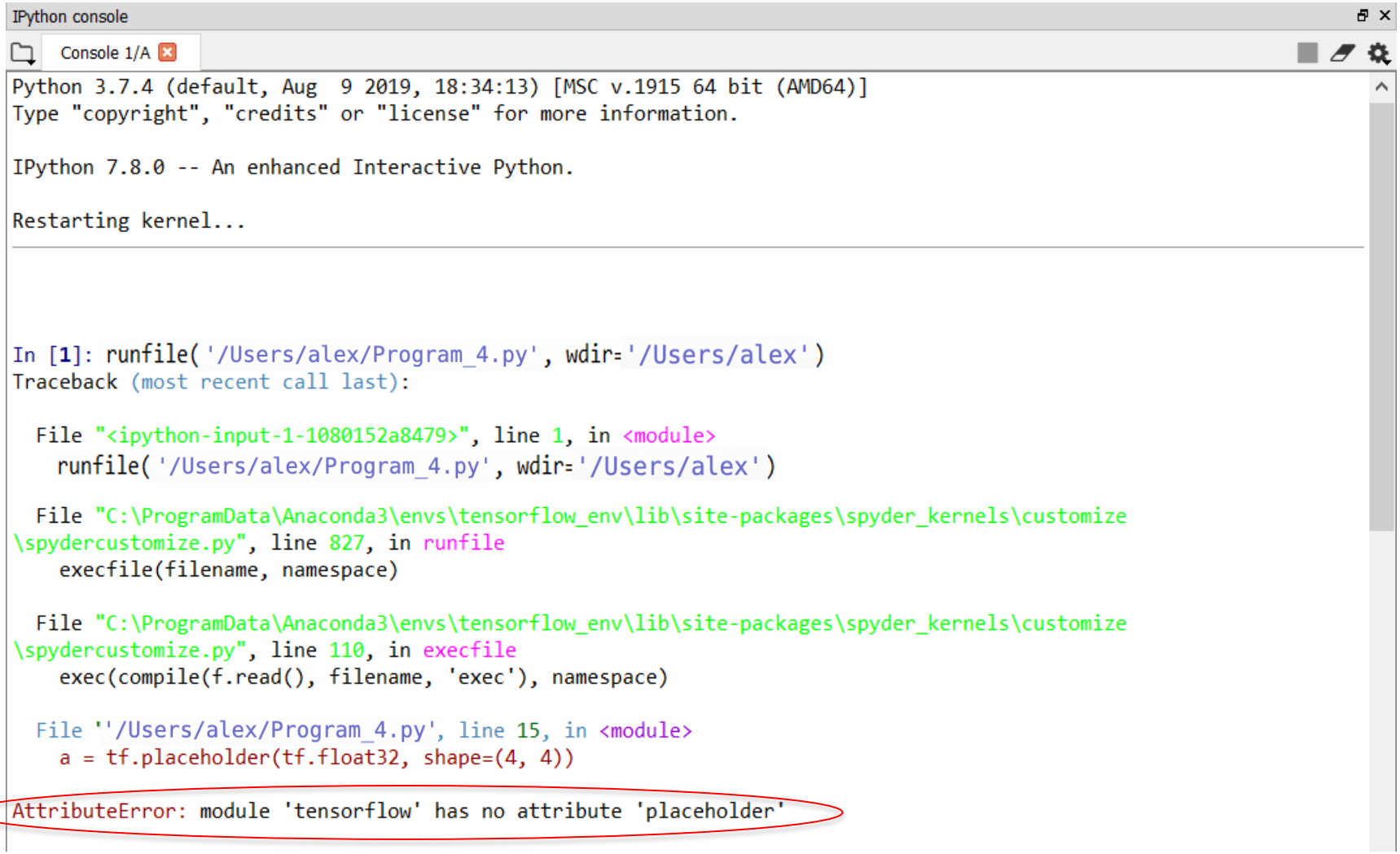


Code example of tf.placeholder():

Program_1.py | Program_2.py | Program_3.py | Program_4.py | Program_5.py | Program_6.py

```
# -*- coding: utf-8 -*-
"""
@author: ailiev
"""
"""This program is used to demonstrate the working of Tensorflow functions
that are removed from TF 2.0 and require to be made compatible.
We demonstrate this by running the tf.placeholder() function """

import numpy as np # since we are using numpy functionalities for matrix operations
import tensorflow as tf# first we import Tensorflow

#a matrix of shape [4,4] with random values will be generated:
a = tf.placeholder(tf.float32, shape=(4, 4))

# assign matrix multiplication of a with itself to b
b = tf.matmul(a, a)

# here, we create a session:
with tf.Session() as sess:
  rand_array = np.random.rand(4, 4)# create a [4,4] array with random values
  # assign this random array to placeholder 'a' & run session
  print(sess.run(b, feed_dict={a: rand_array}))
""" This code will run on TF 1.x but will return an Attribute error in
    TF 2.0 because both tf.placeholder() & tf.session() are deprecated in 2.0,
    hence we need to make the code compatible"""
```

Output for TF 2.0:



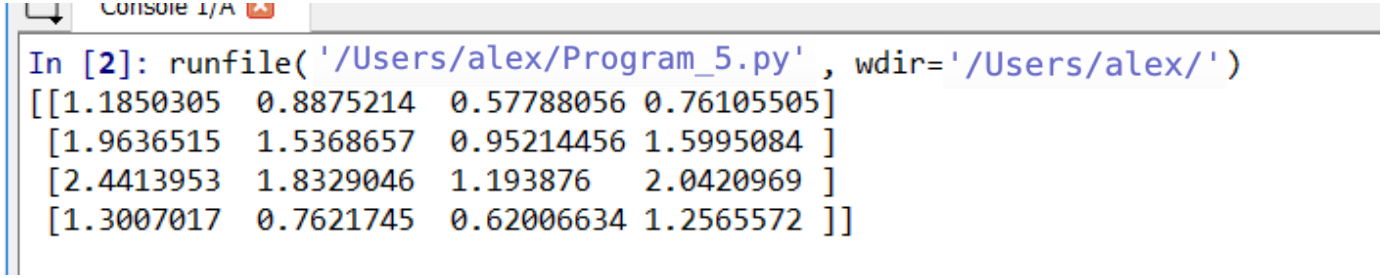
Code example of tf.placeholder() made compatible with TF 2.0:

Program_1.py | Program_2.py | Program_3.py | Program_4.py | Program_5.py | Program_6.py

```
# -*- coding: utf-8 -*-
"""
@author: ailiev
"""

import numpy as np # since we are using numpy functionalities for matrix operations

# to make this code compatible in TF 2.0, we make the following changes:
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

#a matrix of shape [4,4] with random values will be generated:
a = tf.placeholder(tf.float32, shape=(4, 4))

# assign matrix multiplication of a with itself to b
b = tf.matmul(a, a)

# here, we create a session:
with tf.Session() as sess:
  rand_array = np.random.rand(4, 4)# create a [4,4] array with random values
  # assign this random array to placeholder 'a' & run session
  print(sess.run(b, feed_dict={a: rand_array}))
""" Now the code will run on TF 2.0"""
```

Upgraded Output for TF 2.0:

- Here, we have created a placeholder, assigned it with some random values and the multiplied it with itself.



- As we can see from the image, the output does not return an attribute error as tf.placeholder() has been made compatible with TF 2.0 by using tensorflow.compat.v1

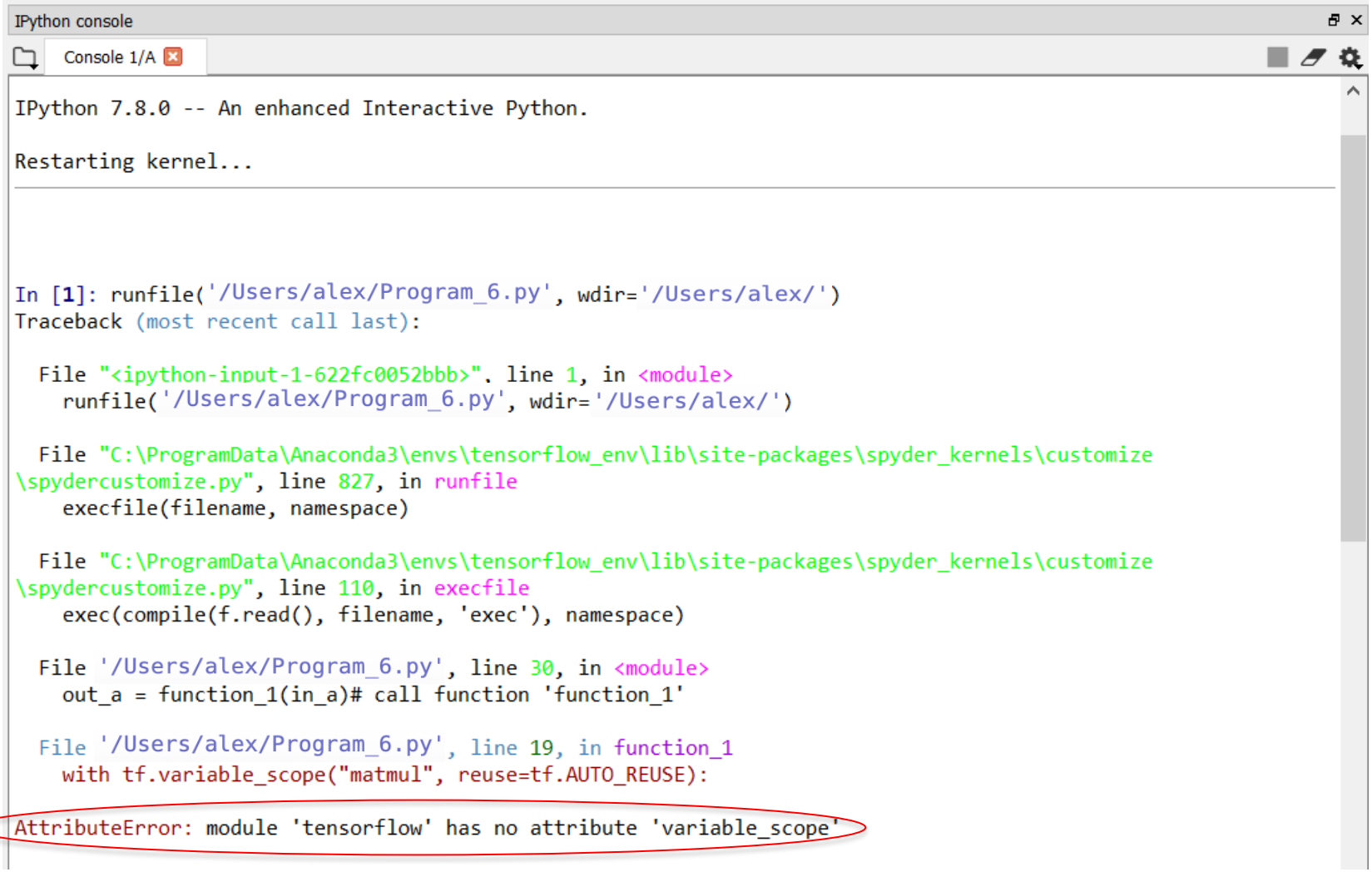
- Now, the code after execution returns the desired output.

Code example of tf.variable_scope():

Program_1.py | Program_2.py | Program_3.py | Program_4.py | Program_5.py | Program_6.py

```
# -*- coding: utf-8 -*-
"""
@author: ailiev
"""

from __future__ import absolute_import, division, print_function, unicode_literals
"""This program is used to demonstrate the working of Tensorflow functions
that are removed from TF 2.0 and require to be made compatible.
We demonstrate this by running the tf.variable_scope() function """

# first, we import the required libraries

import tensorflow as tf
import numpy as np

# we define a function called function_1
def function_1(x):
# tf.variable_scope is a context manager inside which we can define operations
# that create values:
  with tf.variable_scope("matmul", reuse=tf.AUTO_REUSE):
      #first we assign a matrix of 1's of shape[2,2] to variable W
    W = tf.get_variable("W", initializer=tf.ones(shape=(2,2)),
#following line returns a function that can be used to apply L2 regularization to weights:
                        regularizer=tf.contrib.layers.l2_regularizer(0.04))
    # then we assign a matrix of 0's of shape[2] to variable b
    b = tf.get_variable("b", initializer=tf.zeros(shape=(2)))
    return W * x + b #defining an arithmetic operation.
                    #Here,'x' will be the parameter passed to function 'function_1'

in_a = tf.placeholder(dtype=tf.float32, shape=(2))# assign random value to variable
out_a = function_1(in_a)# call function 'function_1'

# function to get  the list of regularization losses
reg_loss = tf.losses.get_regularization_loss(scope="matmul")

with tf.Session() as sess:# create a session
  sess.run(tf.global_variables_initializer())# run the session
  out1 = sess.run([out_a, reg_loss],# run the session
                  feed_dict={in_a: [1, 0]})
  print(out1) # print output
""" This code will run on TF 1.x but will return an Attribute error in
    TF 2.0 because both tf.variable_scope() & tf.session() are deprecated in 2.0,
    hence we need to make the code compatible"""
```

Output for TF 2.0:



Code example of tf.variable_scope() made compatible with TF 2.0 :

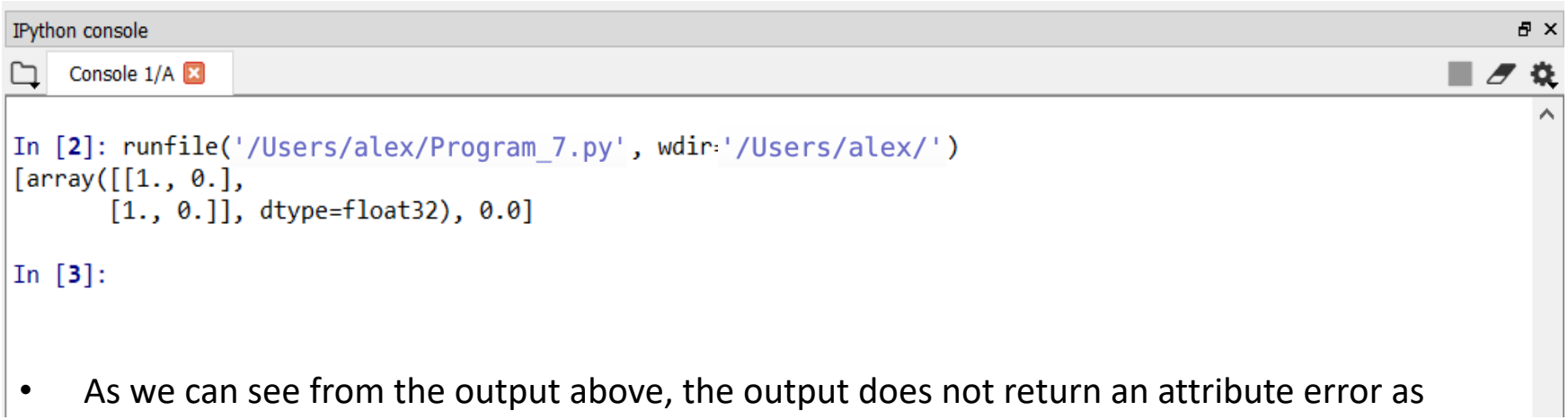
Program_7.py | Program_8.py | Program_8_upgraded.py | Program_9.py | Program_10.py

```
# -*- coding: utf-8 -*-
"""
@author: ailiev
"""
from __future__ import absolute_import, division, print_function, unicode_literals


# to make this code compatible in TF 2.0, we make the following changes:
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

import numpy as np

# we define a function called function_1
def function_1(x):
# tf.variable_scope is a context manager inside which we can define operations
# that create values:
  with tf.variable_scope("matmul", reuse=tf.AUTO_REUSE):
     #first we assign a matrix of 1's of shape[2,2] to variable W
    W = tf.get_variable("W", initializer=tf.ones(shape=(2,2)),
#following line returns a function that can be used to apply L2 regularization to weights:
                        )
    # then we assign a matrix of 0's of shape[2] to variable b
    b = tf.get_variable("b", initializer=tf.zeros(shape=(2)))
    return W * x + b #defining an arithmetic operation.
                     #Here,'x' will be the parameter passed to function 'function_1'

in_a = tf.placeholder(dtype=tf.float32, shape=(2))# assign random value to variable
out_a = function_1(in_a)# call function 'function_1'

# function to get  the list of regularization losses
reg_loss = tf.losses.get_regularization_loss(scope="matmul")

with tf.Session() as sess:# create a session
  sess.run(tf.global_variables_initializer())# run the session
  out1 = sess.run([out_a, reg_loss],# run the session
                  feed_dict={in_a: [1, 0]})
  print(out1) # print output
""" Now the code will run on TF 2.0"""
```

- All the functions – tf.Session(), tf.placeholder(), tf.get_variable() and tf.variable_scope() which are incompatible with TF 2.0 are made compatible with tf.compat.v1 and tf.disable_v2_behavior()

Upgraded Output for TF 2.0:

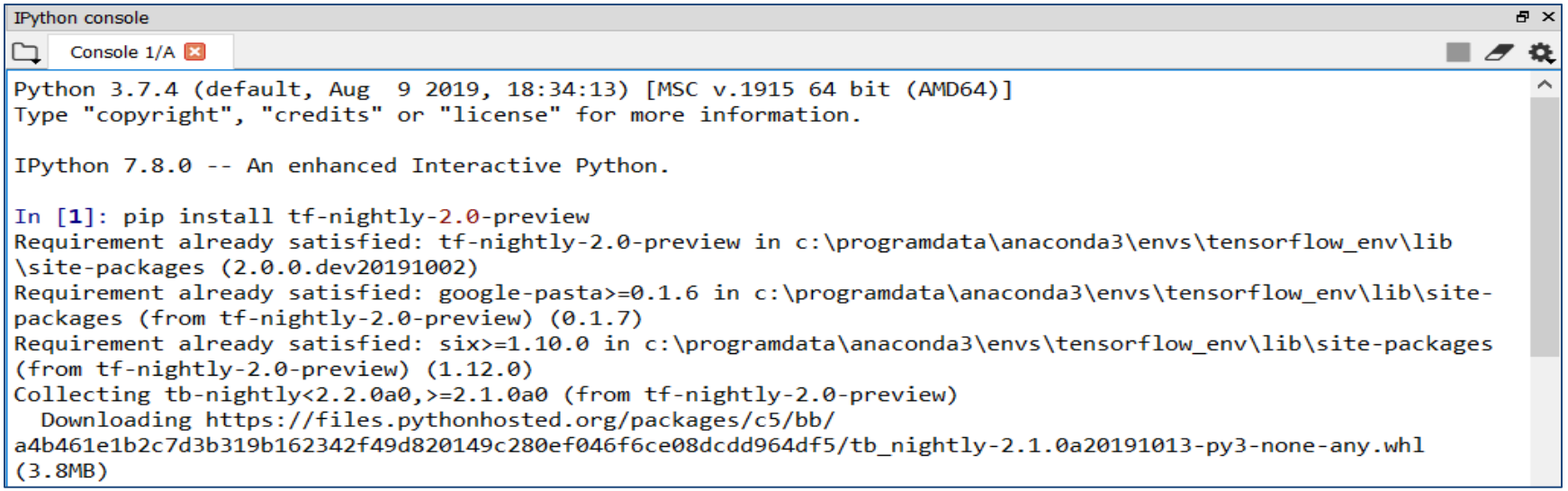
- We used tf. variable_scope() to define the variables 'W' and 'b'. We assign the values to these variables using the tf.get_variable() functionality. Then we performed operations on these variables and print the resulting array, as seen in the output here:



- As we can see from the output above, the output does not return an attribute error as tf.variable_scope() has been made compatible with TF 2.0 by using tensorflow.compat.v1. Now, the code after execution returns the desired output

3. Codes that have functions whose syntax is changed in TF 2.0

- The entire TF 1.x code file can be upgraded to be made compatible for TF 2.0, using a simple upgrade script.
- To run the upgrade script, we first need to install TF Nightly preview as shown below:



- Once the installation is successful, we get the following message:

- Once this is done, we can preface our commands with an exclamation point. Now we are ready to use the upgrade script.

Making the code 2.0 compatible using the upgrade script

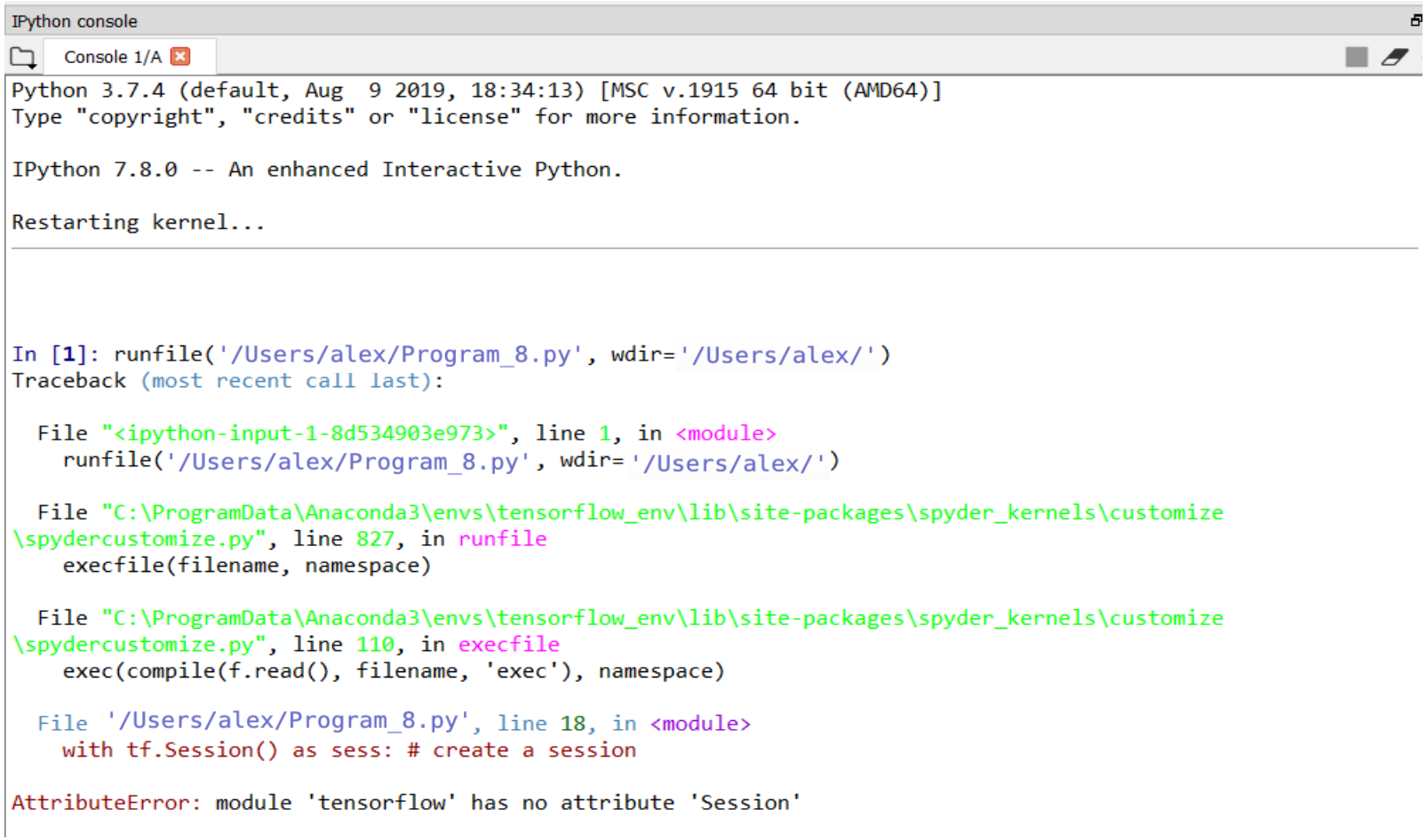
- Code example for tf.Session() using upgrade script :

Program_7.py | Program_8.py | Program_8_upgraded.py | Program_9.py | Program_10.py

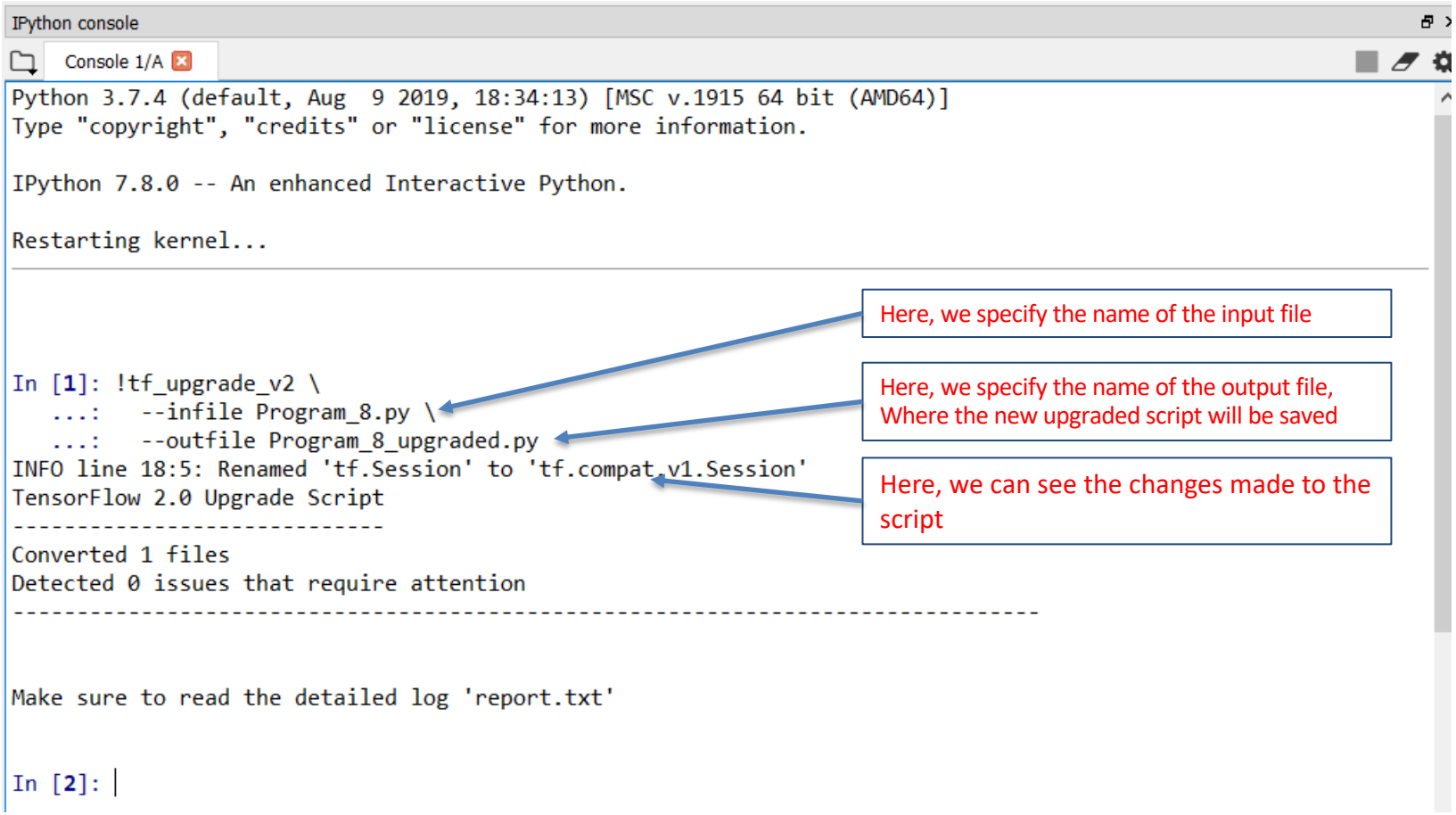
```
# -*- coding: utf-8 -*-
"""
@author: ailiev
"""
"""This program is used to demonstrate the working of the upgrade script for tensorflow,
which will upgrade the entire code to be compatible with TF 2.0, We demonstrate this by
running a simple program with tf.session()"""

import tensorflow as tf # import tensorflow

# assign random values to variables
x = tf.random.normal(shape=(3,3))
y = tf.random.normal(shape=(3,3))
z = tf.ones(shape=(1))
# define an arithmetic operation
result = x + y * z

with tf.Session() as sess: # create a session
    print(sess.run(result)) # run session, print result.
""" This code will run on TF 1.x but will return an Attribute error in
    TF 2.0 because tf.session() is deprecated in 2.0, hence we need to make the code
    compatible by running the upgrade script in the command prompt"""
```

- This example illustrates the use of tf.Session() function, but here we are upgrading instead of making it compatible manually. This is done by the upgrade script.
- The syntax of this function is changed in TF 2.0 automatically.

Output for TF 2.0:

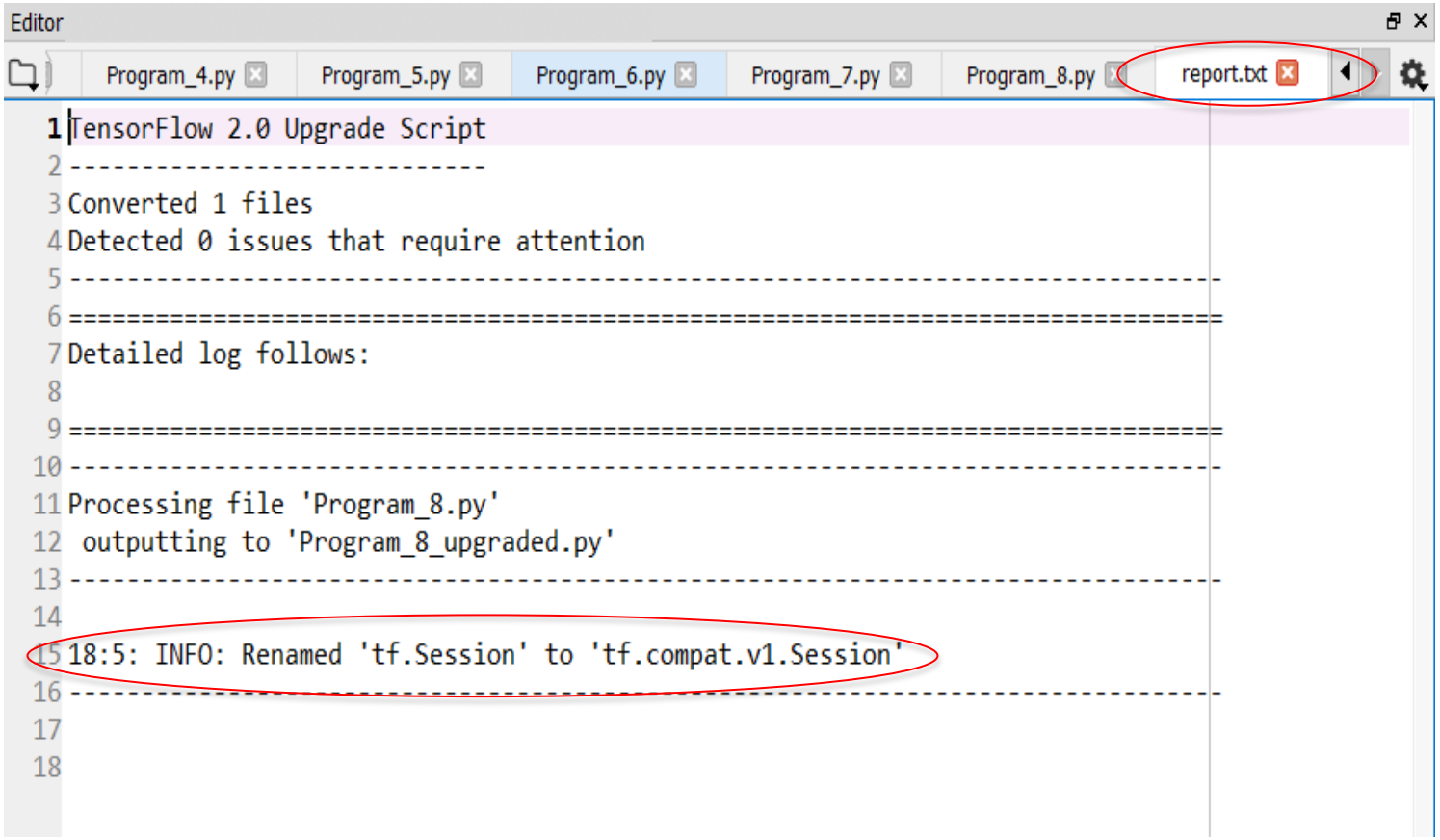


THE UPGRADE SCRIPT:



REPORT of UPGRADE SCRIPT:

We take a look at the report file that generates automatically while running the script. This file provides the details of the changes made to the code



Upgraded Input & Output for TF 2.0:

As we can see from the images, the input script has been modified from ‘tf.session()’ to ‘tf.compat.v1.Session()’ at line no: 17 and the desired output is obtained.

Program_7.py | Program_8.py | Program_8_upgraded.py | Program_9.py | Program_10.py

```
1  # -*- coding: utf-8 -*-
2  """
3  @author: ailiev
4  """
5  """This is the upgraded program resulting after running the
6  upgrade script in command prompt"""
7
8  import tensorflow as tf # import tensorflow
9
10 # assign random values to variables
11 x = tf.random.normal(shape=(3,3))
12 y = tf.random.normal(shape=(3,3))
13 z = tf.ones(shape=(1))
14 # define an arithmetic operation
15 result = x + y * z
16
17 with tf.compat.v1.Session() as sess:# this line is upgraded!!!!!!
18     print(sess.run(result)) # run session, print result.
19 """ Now the code will run on TF 2.0"""
```

IPython console

Console 1/A

```
In [5]: runfile('/Users/alex/Program_8_upgraded.py', wdir='/Users/alex/')
[[-0.8175378   3.928189    1.6188669 ]
 [-0.07639858  0.42909622  1.0391068 ]
 [ 1.1485648   1.4232877   1.411381  ]]
```

Codes that have functions whose syntax is changed in TF 2.0.

Code example:

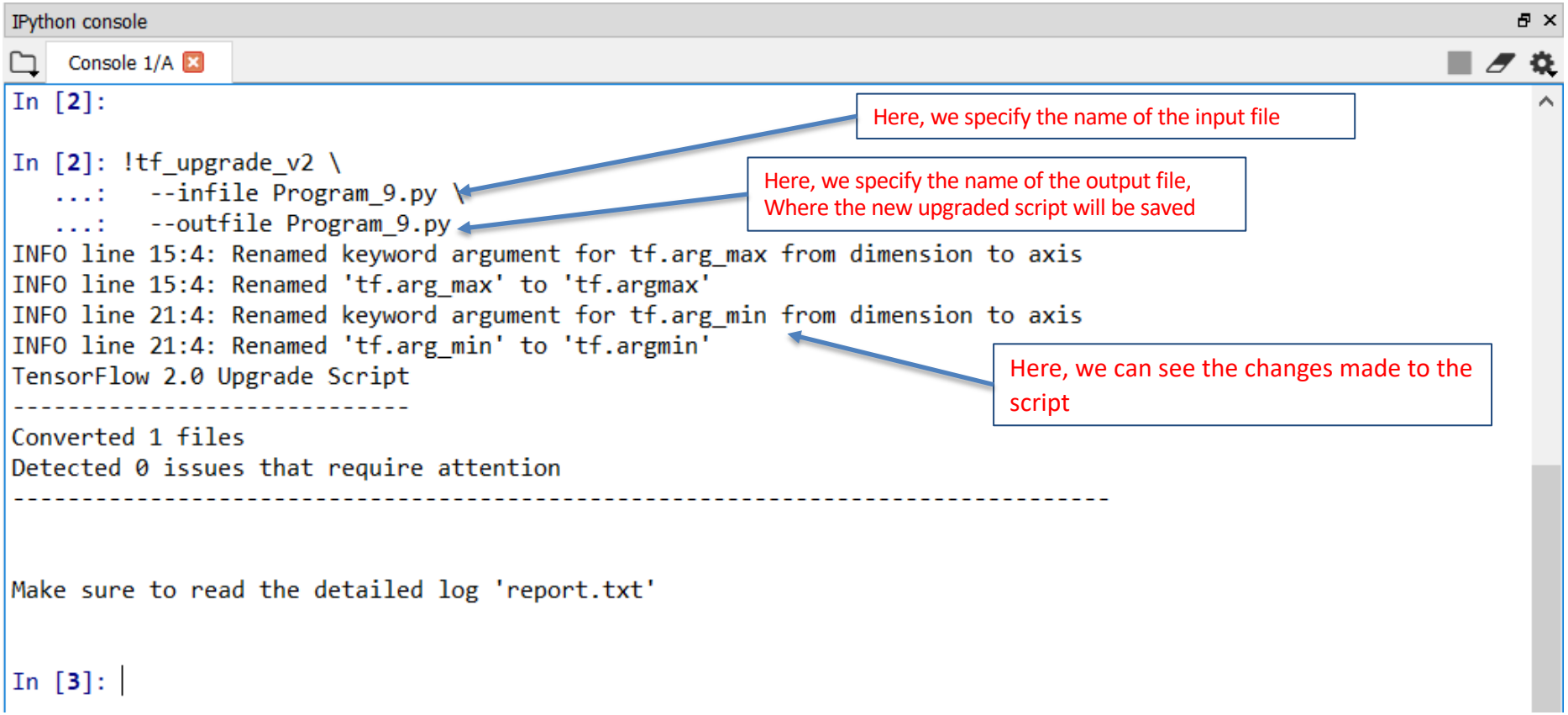
Here, we look at functions that are available in TF 2.0 but their syntax is changed.

Program_7.py | Program_8.py | Program_8_upgraded.py | **Program_9.py** | Program_10.py

```
# -*- coding: utf-8 -*-
"""
@author: ailiev
"""
"""This program is used to demonstrate the working of Tensorflow functions
that are CHANGED in TF 2.0 and are different in TF1.x and TF2.0.
We demonstrate this by running the tf.arg_max() and tf.arg_min() functions """

import tensorflow as tf# import tensorflow


a = [1, 10, 21.1, 2.8, 190.9, 111] # create a tensor with float values
b = tf.argmax(input = a, axis = 0) # this will give the index of the highest no in a
c = tf.keras.backend.eval(b) #Evaluates the value of the variable
# This line will display the version of TF
print('The tensorflow version is : ',tf.__version__,'\n')
print('The index of the highest no is : ',c,'\n')

d = tf.argmin(input = a, axis = 0) # this will give the index of the lowest no in a
e = tf.keras.backend.eval(d) #Evaluates the value of the variable
# This line will display the version of TF
print('The tensorflow version is : ',tf.__version__,'\n')
print('The index of the lowest no is : ',e,'\n')

""" This code will run on TF 1.x but will return an Attribute error in
    TF 2.0 because tf.arg_max() is changed in 2.0,
    hence we need to make the code compatible"""
```

This example illustrates the use of tf.arg_max() & arg_min() functions which return the index of the maximum and minimum value in a data set respectively. The syntax of these functions are changed in TF 2.0.

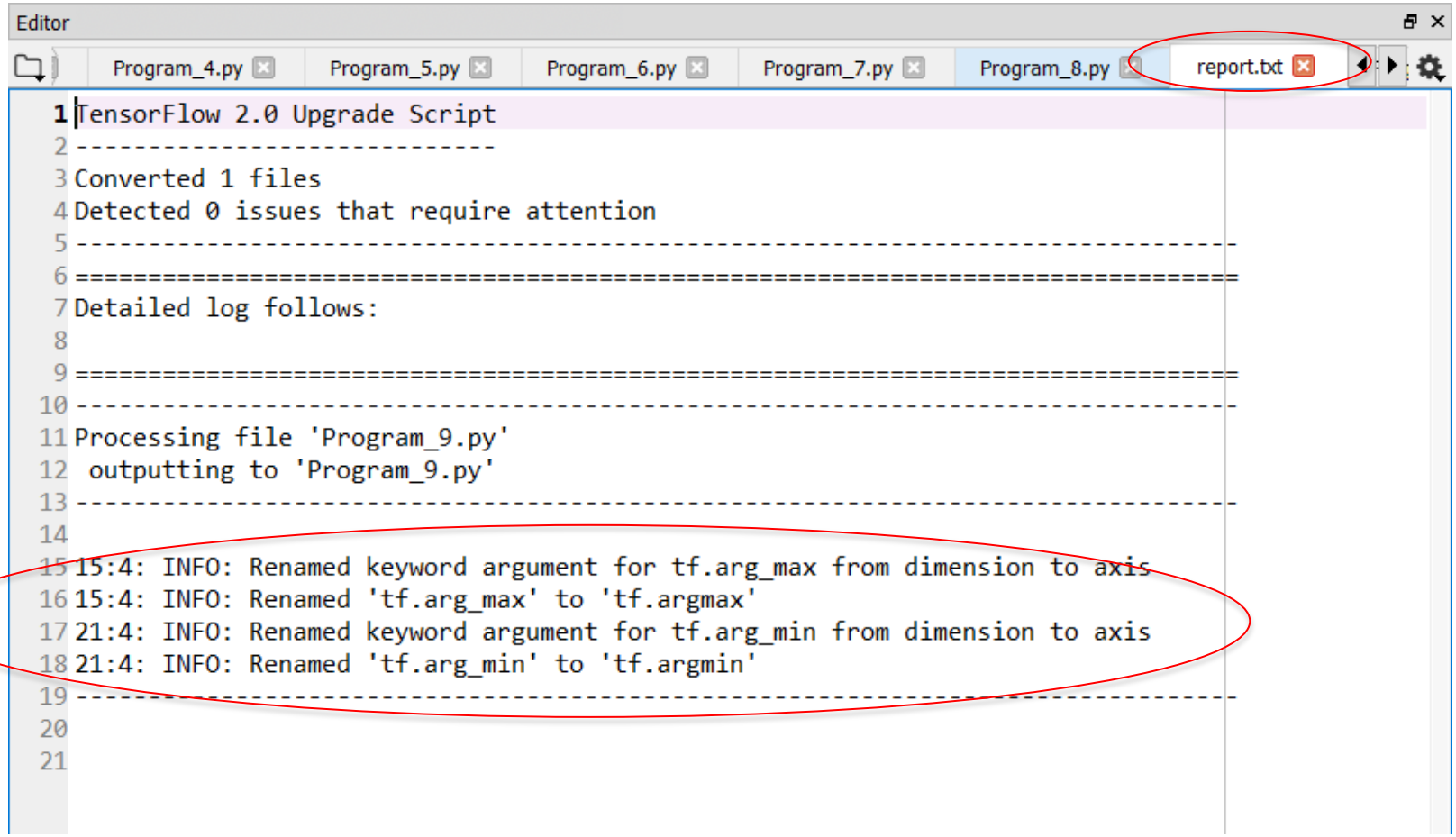
THE UPGRADE SCRIPT:



Here, we provide the same file name for input and output , so that the changes are made to the same current file.

REPORT of UPGRADE SCRIPT:

We take a look at the report file that generates automatically while running the script. This file provides the details of the changes made to the code



Upgraded Input & Output for TF 2.0:

Here, we can see that arg_max() & arg_min() have been changed automatically to argmax() and argmin() respectively as per the new 2.0 syntax:

Program_7.py | Program_8.py | Program_8_upgraded.py | Program_9.py | Program_10.py

```
# -*- coding: utf-8 -*-
"""
@author: ailiev
"""
"""This program is used to demonstrate the working of Tensorflow functions
that are CHANGED in TF 2.0 and are different in TF1.x and TF2.0.
We demonstrate this by running the tf.arg_max() and tf.arg_min() functions """

import tensorflow as tf# import tensorflow


a = [1, 10, 21.1, 2.8, 190.9, 111] # create a tensor with float values
b = tf.argmax(input = a, axis = 0) # this will give the index of the highest no in a
c = tf.keras.backend.eval(b) #Evaluates the value of the variable
# This line will display the version of TF
print('The tensorflow version is : ',tf.__version__,'\n')
print('The index of the highest no is : ',c,'\n')

d = tf.argmin(input = a, axis = 0) # this will give the index of the lowest no in a
e = tf.keras.backend.eval(d) #Evaluates the value of the variable
# This line will display the version of TF
print('The tensorflow version is : ',tf.__version__,'\n')
print('The index of the lowest no is : ',e,'\n')

""" This code will run on TF 1.x but will return an Attribute error in
    TF 2.0 because tf.arg_max() is changed in 2.0,
    hence we need to make the code compatible"""
```

IPython console

Console 1/A

```
In [2]:

In [2]: !tf_upgrade_v2 \
   ...:   --infile Program_9.py \
   ...:   --outfile Program_9.py
INFO line 15:4: Renamed keyword argument for tf.arg_max from dimension to axis
INFO line 15:4: Renamed 'tf.arg_max' to 'tf.argmax'
INFO line 21:4: Renamed keyword argument for tf.arg_min from dimension to axis
INFO line 21:4: Renamed 'tf.arg_min' to 'tf.argmin'
TensorFlow 2.0 Upgrade Script
-----------------------------
Converted 1 files
Detected 0 issues that require attention
--------------------------------------------------------------------------------


Make sure to read the detailed log 'report.txt'


In [3]: runfile('/Users/alex/Program_9.py', wdir='/Users/alex/')
The tensorflow version is :  2.0.0-dev20191002

The index of the highest no is :  4

The tensorflow version is :  2.0.0-dev20191002

The index of the lowest no is :  0
```

Automatically Upgrading the code during run time execution using Exception Handling

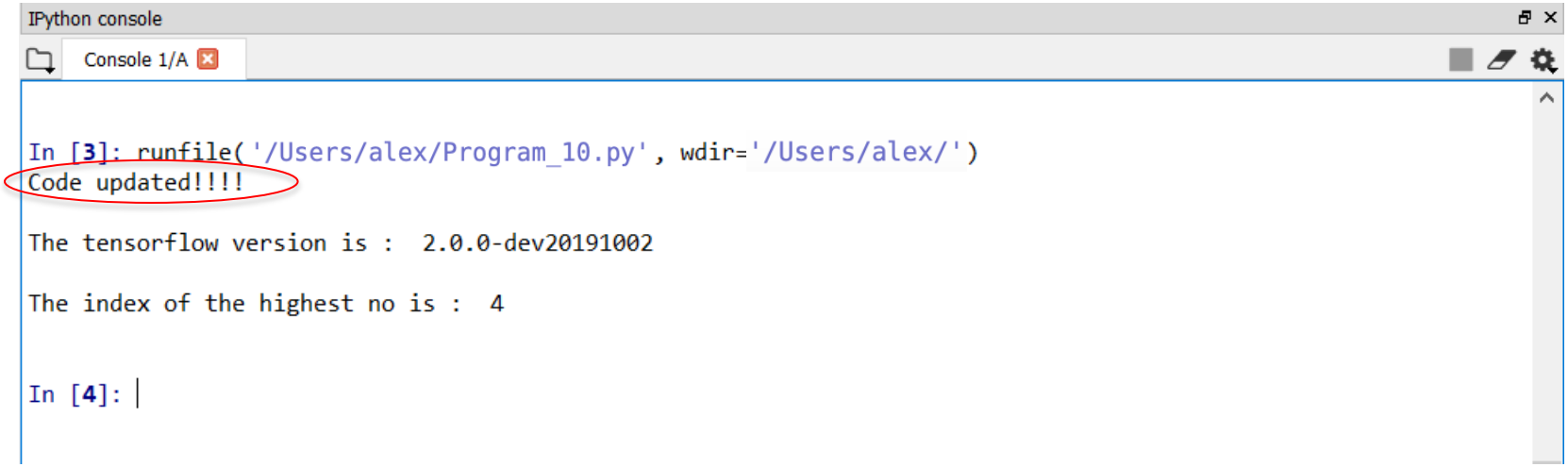
Code example:

Program_7.py | Program_8.py | Program_8_upgraded.py | Program_9.py | Program_10.py

```
# -*- coding: utf-8 -*-
"""
@author: ailiev
"""
""" In this program, we will try to use exception handling to automatically
make the code TF2.0 compatible if an 'AttributeError' is detected."""

import tensorflow as tf# import tensorflow

# we first define a function to return index of highest no
def return_index() :
    a = [1, 90, 89.9, 2.8, 444.3, 64.4]
    b = tf.argmax(input = a, dimension = 0)
    c = tf.keras.backend.eval(b)
    # This line will display the version of TF
    print('The tensorflow version is : ',tf.__version__,'\n')
    print('The index of the highest no is : ',c,'\n')

try :
    return_index()
    # this function will not work in 2.0, since tf.arg_max() is changed in it.
    #hence, AttributeError will be returned
except AttributeError: # once this error is detected, we make the code 2.0 compatible
    import tensorflow.compat.v1 as tf
    tf.disable_v2_behavior()
    print('Code updated!!!! \n')
    return_index()
    # now the function will work!!!!!!
""" hence , the code was updated during runtime"""
```

Output in TF 2.0:

As we can see from the output, the code failed in ‘try’ and was executed in the ‘except’ section.



EAGER EXECUTION IN TENSORFLOW 2.0

- Tensorflow's Eager Execution helps in executing the operations immediately without building the graphs.
- In Tensorflow 2.0 Eager Execution is enabled by default , this makes easy to debug models.

Code Example
for Eager Execution:

eagerexecution.py

```
1  """
2  @author : ailiev
3  """
4
5  import tensorflow as tf
6
7  tf.executing_eagerly()
8
9  a = tf.constant([[1, 2],
10                   [3, 4]])
11 b = tf.constant([[3, 1],
12                   [5, 6]])
13 print('the tensorflow version is :', tf.__version__)
14 print(a*b)
```

Output in TF 2.0:

```
the tensorflow version is : 2.0.0-rc1
tf.Tensor(
[[ 3  2]
 [15 24]], shape=(2, 2), dtype=int32)

In [2]:
```

Output in 1.x:

```
the tensorflow version is : 1.14.0
Tensor("mul_1:0", shape=(2, 2), dtype=int32)

In [4]:
```

CONTROL STATEMENTS IN TENSORFLOW

- **Syntax of if:**

  tf.cond(condition, execute if it is true , execute if condition is false)

- **Syntax of while:**

  tf.while_loop(cond , body , loop_vars , shape_invariants=None , parallel_iterations=10,back_prop=True,swap_memory=False,maximum_iterations=None, name=None)

- while_loop implements non-strict semantics, enabling multiple iterations to run in parallel

While loop in TensorFlow

```
1 import tensorflow as tf
2
3 g3= tf.Graph()
4 with g3.as_default():
5     @tf.function
6     def cond(Calculated_op, Desired_op):  #condition on which while will iterate
7         return tf.math.not_equal(Calculated_op, Desired_op)  #return equality between Calculated_op, Desired_op
8
9     @tf.function
10    def body(Calculated_op, Desired_op):    # body which will execute
11        #checks if calculated value is greater than Desired_op and take decision
12        re = tf.cond(tf.less(Calculated_op, Desired_op), lambda: Calculated_op + 10 , lambda: Calculated_op - 10)
13        return [re  , Desired_op]
14
15    Calculated_op = tf.constant(100)
16    Desired_op = tf.constant(50000)
17
18 with tf.compat.v1.Session(graph=g3) as sess:
19     # while loop (condition , body,(parameters on which while loop will run)
20     res = sess.run(tf.while_loop(cond, body, [Calculated_op, Desired_op]))
21     print(res)
22
```

Output:

[12] [50000, 50000]

REDUCTION

```python
import tensorflow as tf
import numpy as np

g3= tf.Graph()
with g3.as_default():
    def convert(v, t=tf.float32):
        return tf.convert_to_tensor(v, dtype=t)

    x = convert(
        np.array(
            [
                (1, 2, 3),
                (4, 5, 6),
                (7, 8, 9)
            ]), tf.int32)

    bool_tensor = convert([(True, False, True), (False, False, True), (True, False, False)], tf.bool)

    red_sum_0 = tf.reduce_sum(x) # Reduce sum without passed axis parameter
    red_sum = tf.reduce_sum(x, axis=1) # Reduce sum with passed axis=1

    red_prod_0 = tf.reduce_prod(x) # Reduce product without passed axis parameter
    red_prod = tf.reduce_prod(x, axis=1) #Reduce product with passed axis=1

    red_min_0 = tf.reduce_min(x) # Reduce min without passed axis parameter
    red_min = tf.reduce_min(x, axis=1) # Reduce min with passed axis=1

    red_max_0 = tf.reduce_max(x) # Reduce max without passed axis parameter
    red_max = tf.reduce_max(x, axis=1) # Reduce max with passed axis=1

    red_mean_0 = tf.reduce_mean(x) # Reduce mean without passed axis parameter
    red_mean = tf.reduce_mean(x, axis=1) # Reduce mean with passed axis=1

    red_bool_all_0 = tf.reduce_all(bool_tensor) # Reduce bool all without passed axis parameter
    red_bool_all = tf.reduce_all(bool_tensor, axis=1) # Reduce bool all with passed axis=1

    red_bool_any_0 = tf.reduce_any(bool_tensor) # Reduce bool any without passed axis parameter
    red_bool_any = tf.reduce_any(bool_tensor, axis=1) #Reduce bool any with passed axis=1


with tf.compat.v1.Session(graph=g3) as session:
    print (session.run(red_sum_0))
    print (session.run(red_sum))

    print (session.run(red_prod_0))
    print (session.run(red_prod))

    print (session.run(red_min_0))
    print (session.run(red_min))

    print (session.run(red_max_0))
    print (session.run(red_max))

    print (session.run(red_mean_0))
    print (session.run(red_mean))
```

Output:

```
45
[ 6 15 24]
362880
[  6 120 504]
1
[1 4 7]
9
[3 6 9]
5
[2 5 8]
False
[False False False]
True
[ True  True  True]
```

Example: Simple Calculator using class and functions

```
import tensorflow as tf
class calculator():

    def __init__(self,num1,num2): #initialize variables
        self.num1 = num1
        self.num2 = num2

    def multiply(self):
        g= tf.Graph() # define a default graph which is passed to session
        with g.as_default():

                    x1 = self.num1
                    x2 = self.num2
                    result = tf.multiply(x1, x2)

            # Multiply


            # Initialize Session and run `result`
                    with tf.compat.v1.Session(graph=g) as sess:
                      output = sess.run(result)
                      return output


    def add(self):
        g2= tf.Graph() # define a default graph which is passed to session
        with g2.as_default():

                    x1 = self.num1
                    x2 = self.num2
                    result = tf.add(x1, x2)

            # Addition

            # Initialize Session and run `result`
                    with tf.compat.v1.Session(graph=g2) as sess:
                      output = sess.run(result)
                      return output
```

Example: Simple Calculator using class and functions

```
def subtract(self):
    g3= tf.Graph() # define a default graph which is passed to session
    with g3.as_default():

                x1 = self.num1
                x2 = self.num2
                result = tf.subtract(x1, x2)

        # Subtract


        # Initialize Session and run `result`
                with tf.compat.v1.Session(graph=g3) as sess:
                  output = sess.run(result)
                  return output

def divisi(self):
    g6= tf.Graph() # define a default graph which is passed to session

    with g6.as_default():

                x1 = self.num1
                x2 = self.num2
                result = tf.truediv(x1, x2)

        # divide
                if x2 == 0:
                    return "Cannot divide by 0"

        # Initialize Session and run `result`
                with tf.compat.v1.Session(graph=g6) as sess:
                  output = sess.run(result)
                  return output
```

Example: Simple Calculator using class and functions

```
oper = input("what operation ")
num1 = int(input("Enter num1 "))
num2 = int(input("Enter num2 "))

c = calculator(num1, num2) #instantiation
if(oper == "multiply"):
    print("Answer: ",c.multiply())
elif oper == "addition":
    print("Answer: ",c.add())
elif oper == "subtraction":
    print("Answer: ",c.subtract())
elif oper == "divide":
    print("Answer: ",c.divisi())
```

Output:

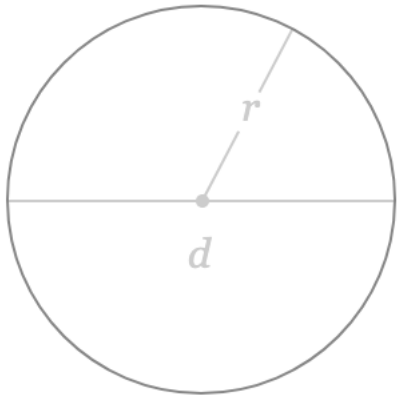
```
what operation divide

Enter num1 8

Enter num2 2
Answer:  4.0
```

Class exercise:

Using TF, find the area of a circle, where:

- You use a constant for *π (3.14 or 22/7)*
- You ask the user for an input of a radius r = using *input*
- Use the formula for an area that is: *A = π * r ^{2}*
- Print your result like this: "Area of circle 78.5"

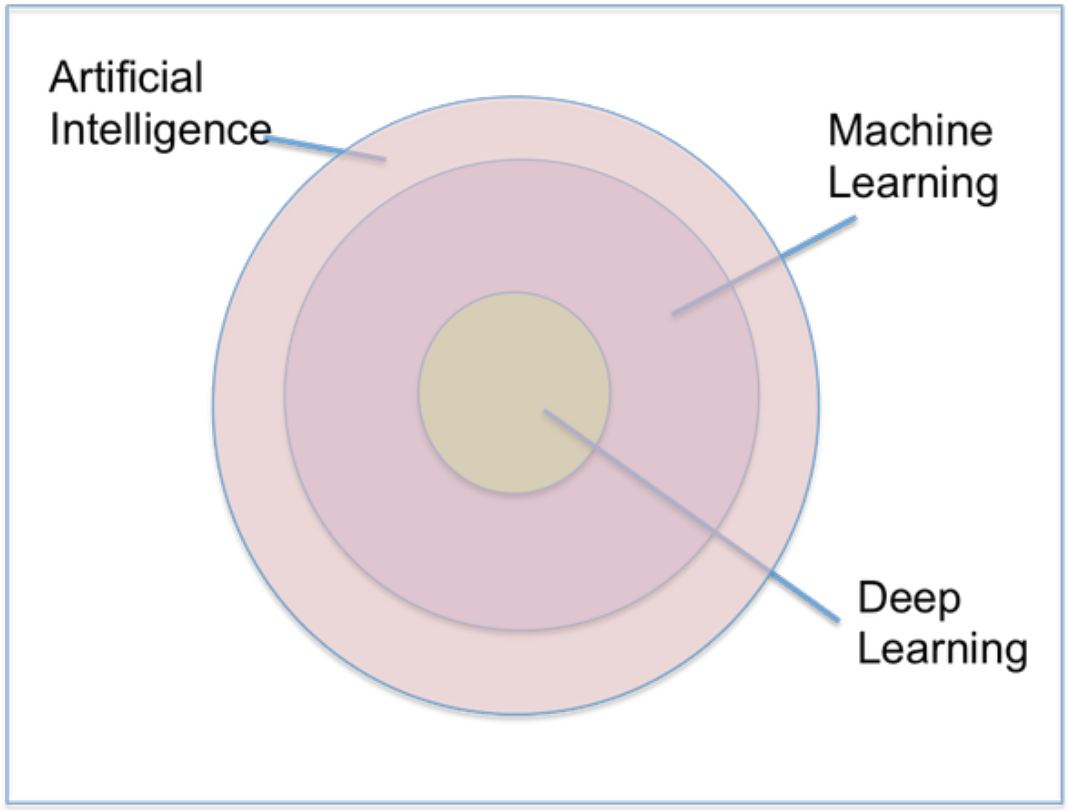
$$A = \pi r^2$$



Solution: find the area of a circle

```
import tensorflow as tf

g = tf.Graph()

with g.as_default():
    pi = tf.constant(3.14) #define constant
    radius = tf.constant(5, tf.float32)
    area = tf.multiply((radius ** 2 ), pi)

# launch the graph in a session
with tf.compat.v1.Session(graph = g) as sess:

    print("Area of circle ",sess.run(area))

#OUTPUT ------   Area of circle 78.5
```

The Big Picture



Data Mining

- What is *data mining*?
  - Data mining is defined as the process of discovering patterns in data
  - Data mining is about solving problems by analyzing data already present in datasets / databases
  - In data mining, the data is stored electronically, and the search is automated
  - It has been estimated that the amount of data stored in the world's databases doubles every 20 months

Data Mining

- What is *data mining*?
  - Data mining is a topic that involves learning in a practical, nontheoretical sense
  - It is about finding and describing previously unknown patterns in data
  - The output may include a description of a structure that can be used to classify unknown examples
  - Finally it is about the acquisition of knowledge and the ability to use it

Data Mining

- Finding patterns in data that provide insight or enable fast and accurate decision making
- Strong, accurate patterns are needed to make decisions
  - Problem 1: most patterns are not interesting
  - Problem 2: patterns may be inexact (or spurious)
  - Problem 3: data may be garbled or missing
- Machine learning techniques identify patterns in data and provide many tools for data mining
- Of primary interest are machine learning techniques that provide structural descriptions

Machine Learning

- What is *machine learning*?

- The dictionary defines “to learn” as:
  - To get knowledge of something by study, experience, or being taught
  - To become aware by information or from observation
  - To commit to memory
  - To be informed of or to ascertain
  - To receive instruction

Machine Learning

- Machine Learning basics:
  - What is an *attribute*?

    Each instance that provides the input to machine learning is characterized by its values on a fixed, predefined set of features or attributes

    Example:

    in the weather, the attribute *temperature* has the values: hot > mild > cold

  - Basic attribute data types can be: nominal and numeric

    Example:

    string attributes and date attributes are effectively nominal and numeric

Machine Learning

- What is *machine learning*?
- An operational definition can be formulated in the same way for learning:
  - Things learn when they change their behavior in a way that makes them perform better in the future
- This ties learning to *performance* rather than *knowledge*
- You can test learning by observing present behavior and comparing it with past behavior

Machine Learning

- What is *machine learning?*

- … We therefore *choose* the word training to denote a mindless kind of learning
- We train animals and even plants, although it would be stretching the word a bit to talk of training objects such as slippers, which are not in any sense alive
- … But learning is different. Learning implies thinking and purpose.
- Something that learns has to do so intentionally
- That is why we wouldn't say that a vine has learned to grow around a trellis in a vineyard—we'd say it has been trained
- Learning without purpose is merely training

Machine Learning

- Supervised Machine Learning:
  - The majority of practical machine learning uses supervised learning
  - Supervised learning is to have input variables (x) and an output variable (Y) and we use an algorithm to learn the mapping function from the input to the output, hence finding:

$$Y = f(X)$$

  - The goal is to approximate the mapping function so well that when you have new input data (x) that you can predict the output variables (Y) for that data
  - It is supervised learning because the process of learning from the training dataset can be thought of as a teacher supervising the learning process

Machine Learning

- Supervised Machine Learning:
  - The two groups of supervised learning are:
    - Classification: A classification problem is when the output variable is a category, such as “red” or “blue” or “disease” and “no disease”
    - Regression: A regression problem is when the output variable is a real value, such as “dollars” or “weight”
  - Some popular examples of supervised machine learning algorithms are:
    - Linear regression for regression problems.
    - Random forest for classification and regression problems.
    - Support vector machines (SVM) for classification problems.

Machine Learning

- Unsupervised Machine Learning:
  - Unsupervised learning is where you only have input data (X) and no corresponding output variables.
  - The goal for unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data.
  - These are called unsupervised learning because unlike supervised learning above there is no correct answers and there is no teacher. Algorithms are left to their own devises to discover and present the interesting structure in the data.

Machine Learning

- Unsupervised Machine Learning:
  - Unsupervised learning problems can be further grouped into clustering and association problems:
    - Clustering: A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.
    - Association: An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.
  - Some popular examples of unsupervised learning algorithms are:
    - k-means for clustering problems.
    - A priori algorithm for association rule learning problems.

Machine Learning

- Semi-Supervised Machine Learning:
  - Problems where we have a large amount of input data (X) and only some of it is labeled (Y) are called semi-supervised learning problems
  - These problems sit in between both supervised and unsupervised learning.
  - A good example is a photo archive where only some of the images are labeled, (e.g. dog, cat, person) and the majority are unlabeled.

Machine Learning

- Semi-Supervised Machine Learning:
  - Many real world machine learning problems fall into this area
  - It can be expensive or time-consuming to label data as it may require access to domain experts
  - Hence unlabeled data is cheap and easy to collect and store

Machine Learning

- Semi-Supervised Machine Learning:
  - You can use unsupervised learning techniques to discover and learn the structure in the input variables
  - You can also use supervised learning techniques to make best guess predictions for the unlabeled data, feed that data back into the supervised algorithm as training data and use the model to make predictions on new unseen data *(back propagation)*

What's in an example?

- Instance: specific type of example
  - Thing to be classified, associated, or clustered
  - Individual, independent example of target concept
  - Characterized by a predetermined set of attributes
- Input to learning scheme: set of instances/dataset
  - Represented as a single relation/flat file
- Rather restricted form of input
  - No relationships between objects
- Most common form in practical data mining

What's in an attribute?

- Each instance is described by a fixed predefined set of features, its attributes
- But: number of attributes may vary in practice
  - Possible solution: "irrelevant value" flag
- Related problem: existence of an attribute may depend of value of another one
- Possible attribute types ("levels of measurement"):
  - *Nominal*, *ordinal*, *interval* and *ratio*

Machine Learning

- Summary:

attributes = features = predictors

examples = instances = variables

sunny,85,85,FALSE	no
sunny,80,90,TRUE	no
overcast,83,86,FALSE	yes
rainy,70,96,FALSE	yes
rainy,68,80,FALSE	yes
rainy,65,70,TRUE	no
overcast,64,65,TRUE	yes
sunny,72,95,FALSE	no
sunny,69,70,FALSE	yes
rainy,75,80,FALSE	yes
sunny,75,70,TRUE	yes
overcast,72,90,TRUE	yes
overcast,81,75,FALSE	yes
rainy,71,91,TRUE	no

class = outcome

Clustering

- Clustering techniques apply when there is no class to be predicted: they perform unsupervised learning
- Aim: divide instances into "natural" groups
- Clusters can be:
  - disjoint vs. overlapping
  - deterministic vs. probabilistic
  - flat vs. hierarchical
- We will look at a classic clustering algorithm called *k-means*
- *k-means* clusters are disjoint, deterministic, and flat

The *k*-means algorithm

- Step 1: Choose *k* random cluster centers
- Step 2: Assign each instance to its closest cluster center based on Euclidean distance
- Step 3: Re-compute cluster centers by computing the average (aka *centroid*) of the instances pertaining to each cluster
- Step 4: If cluster centers have moved, go back to Step 2

- This algorithm minimizes the squared Euclidean distance of the instances from their corresponding cluster centers
  - Determines a solution that achieves a *local* minimum of the squared Euclidean distance
- Equivalent termination criterion: stop when assignment of instances to cluster centers has not changed

The *k*-means algorithm

- Manhattan distance

Taxicab geometry

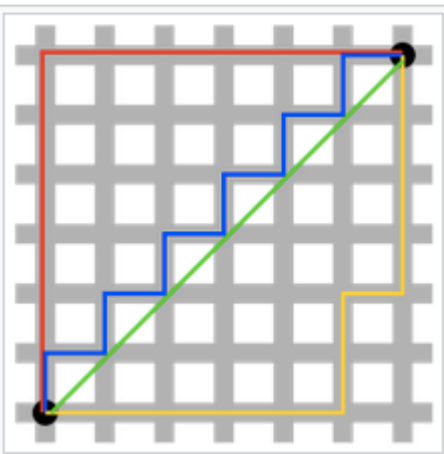
From Wikipedia, the free encyclopedia
(Redirected from Manhattan distance)

A **taxicab geometry**, considered by Hermann Minkowski in 19th-century Germany, is a form of geometry in which the usual distance function or metric of Euclidean geometry is replaced by a new metric in which the distance between two points is the sum of the absolute differences of their Cartesian coordinates.

The **taxicab metric** is also known as **rectilinear distance**, **L_1 distance** or **ℓ_1 norm** (see L^p space), **snake distance**, **city block distance**, **Manhattan distance** or **Manhattan length**, with corresponding variations in the name of the geometry.[1] The latter names allude to the grid layout of most streets on the island of Manhattan, which causes the shortest path a car could take between two intersections in the borough to have length equal to the intersections' distance in taxicab geometry.

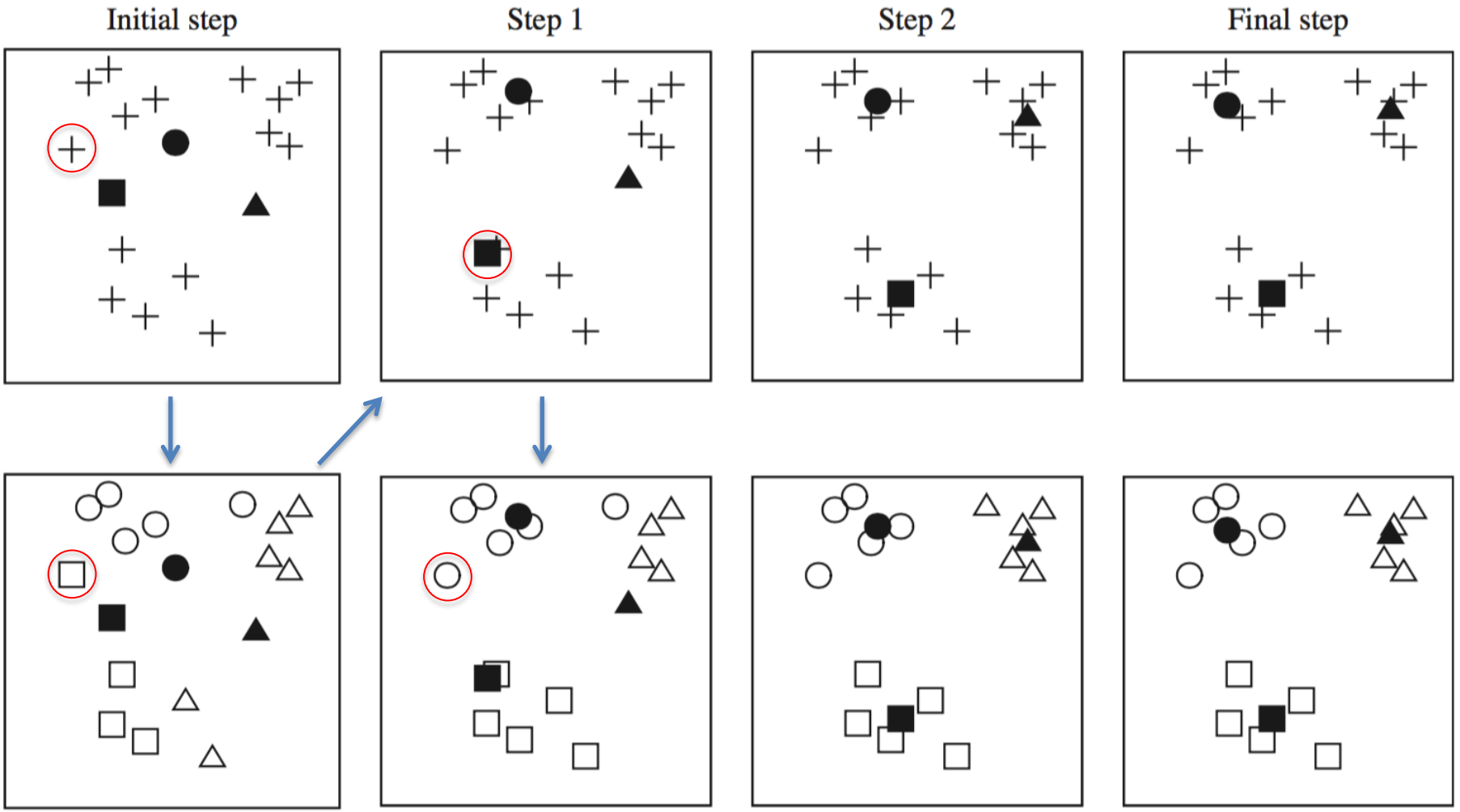
Contents [hide]

1 Formal definition
2 Properties



Taxicab geometry versus Euclidean distance: In taxicab geometry, the red, yellow, and blue paths all have the shortest length of 12. In Euclidean geometry, the green line has length $6\sqrt{2} \approx 8.49$, and is the unique shortest path.

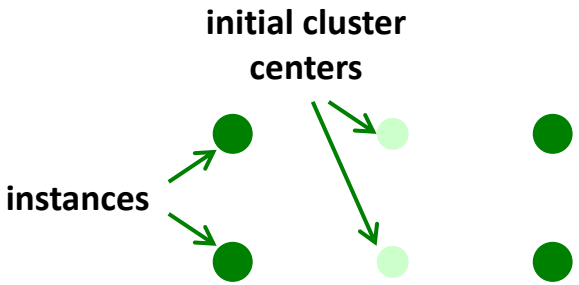
The *k*-means algorithm: example



Discussion

- Algorithm minimizes squared distance to cluster centers

- Result can vary significantly based on initial choice of seeds
- Can get trapped in local minimum
  - Example:



- To increase chance of finding global optimum:
  restart with different random seeds
- Can be applied recursively with $k = 2$

Faster distance calculations

- Can we use *k*D-trees or ball trees to speed up the process?
- Yes, we can:
  - First, build the tree data structure
  - At each node, store the number of instances and the sum of all instances (summary statistics)
  - In each iteration of *k*-means, descend the tree and find out which cluster each node belongs to
  - Can stop descending as soon as we find out that a node belongs entirely to a particular cluster

    Stop = iterations * clusters * instances * dimensions
  - Use summary statistics stored previously at the nodes to compute new cluster centers

K-means with TensorFlow

- Example:

```
## k-means
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf

# Create aliases:
tf.sub = tf.subtract

points_n = 200
clusters_n = 3
iteration_n = 100

# 1. Let's generate random data points with a uniform distribution and assign them
#    to a 2D tensor constant. Then, randomly choose initial centroids from the set
#    of data points:
points = tf.constant(np.random.uniform(0, 10, (points_n, 2)))
centroids = tf.Variable(tf.slice(tf.random_shuffle(points), [0, 0], [clusters_n, -1]))

# 2. Next we want to be able to do element-wise subtraction of points and centroids
#    that are 2D tensors. Because the tensors have different shape, let's expend points
#    and centroids into 3 dimensions, which allows us to use the broadcasting feature of
#    subtraction operation:
points_expanded = tf.expand_dims(points, 0)
centroids_expanded = tf.expand_dims(centroids, 1)
```

```
(<module>)>>> sess.run(points).shape
(200, 2)

(<module>)>>> sess.run(points_expanded).shape
(1, 200, 2)

(<module>)>>> sess.run(centroids).shape
(3, 2)

(<module>)>>> sess.run(centroids_expanded).shape
(3, 1, 2)
```

K-means with TensorFlow

- Example:

```
34  # 3. Then, calculate the distances between points and centroids and determine the
35  #    cluster assignments:
36  distances = tf.reduce_sum(tf.square(tf.sub(points_expanded, centroids_expanded)), 2)
37  assignments = tf.argmin(distances, 0)
38
39  # 4. Ne                                                                    nts
40  #    as                                                                    efined
41  #    ce
42  means =
43  for c i
44      mea
45                                                                           indices=[1]))
46
47  new_cen                                                                  ts, c)).shape
48
49  update_
50  init = tf.global_variables_initializer()
```

```
(<module>)>>> sess.run(assignments)
array([1, 2, 2, 0, 2, 1, 2, 1, 0, 2, 0, 2, 2, 0, 2, 1, 1, 1, 1, 1, 2, 2,
       0, 1, 2, 2, 0, 1, 1, 0, 1, 0, 0, 1, 2, 2, 0, 2, 1, 0, 1, 0, 0, 2,
       2, 1, 2, 1, 1, 2, 0, 0, 1, 2, 1, 0, 0, 2, 1, 2, 1, 1, 0, 2, 1, 1,
       1, 1, 1, 2, 2, 1, 2, 2, 1, 1, 2, 1, 0, 1, 2, 1, 0, 1, 0, 1, 2, 2,
       0, 2, 1, 1, 2, 2, 0, 1, 1, 2, 2, 0, 2, 1, 2, 1, 1, 1, 2, 2, 0, 1,
       1, 0, 1, 1, 2, 1, 1, 0, 2, 1, 2, 1, 2, 1, 0, 1, 1, 2, 1, 2, 1, 1,
       1, 0, 2, 0, 0, 1, 2, 1, 2, 0, 1, 2, 1, 1, 1, 1, 2, 0, 1, 1, 0, 1,
       2, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 2, 2, 1, 0, 1, 1, 1, 2, 0,
       1, 2, 1, 1, 2, 1, 2, 2, 2, 1, 1, 0, 1, 1, 0, 2, 2, 1, 1, 0, 1, 0,
       1, 1])
```

```
(<module>)>>> sess.run(tf.equal(assignments, c))
array([False, False, False,  True, False, False, False, False,  True,
       False,  True, False, False,  True, False, False, False, False,
       False, False, False, False,  True, False, False, False,  True,
       False, False,  True, False,  True,  True, False, False, False,
        True, False, False,  True, False,  True,  True, False, False,
       False, False, False, False, False,  True,  True, False, False,
       False,  True,  True, False, False, False, False, False,  True,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False,  True, False, False,
       False,  True, False,  True, False, False, False,  True, False,
       False, False, False, False,  True, False, False, False, False,
        True, False, False, False, False, False, False, False, False,
        True, False, False,  True, False, False, False, False, False,
```

for c = 0

K-means with TensorFlow

- Example:

```
34  # 3. Then, calculate the distances between points and centroids and determine the
35  #    cluster assignments:
36  distances = tf.reduce_sum(tf.square(tf.sub(points_expanded, centroids_expanded)), 2)
37  assignments = tf.argmin(distances, 0)
38
39  # 4. Next, we can compare each cluster with a cluster assignments vector, get points
40  #    assigned to each cluster, and calculate mean values. These mean values are refined
41  #    centroids, so let's update the centroids variable with the new values:
42  means = []
43  for c in range(clusters_n):
44      means.append(tf.reduce_mean( tf.gather(points,
45                  tf.reshape(tf.where(tf.equal(assignments, c)),[1,-1])),reduction_indices=[1]))
46
47  new_centroids = tf.concat(means, 0)
48
49  update_centroids = tf.assign(centroids, new_centroids)
50  init = tf.global_variables_initializer()
```

for c = 0

```
(<module>)>>> sess.run(tf.reshape(tf.where(tf.equal(assignments, c)),[1,-1])).shape
(1, 45)

(<module>)>>> sess.run(tf.reshape(tf.where(tf.equal(assignments, c)),[1,-1]))
array([[  3,   8,  10,  13,  22,  26,  29,  31,  32,  36,  39,  41,  42,
         50,  51,  55,  56,  62,  78,  82,  84,  88,  94,  99, 108, 111,
        117, 124, 133, 135, 136, 141, 149, 152, 156, 157, 159, 161, 162,
        170, 175, 187, 190, 195, 197]])
```

K-means with TensorFlow

- Example:

```
# 3. Then, calculate the distances between points and centroids and determine the
#    cluster assignments:
distances
assignmen
# 4. Next
#    assi
#    cent
means = [
for c in
    means
new_centr
update_ce
init = tf
```

```
(<module>)>>> sess.run(tf.equal(assignments, c))
array([False, False, False,  True, False, False, False, False,  True,
       False,  True, False, False,  True, False, False, False, False,
       False, False, False, False,  True, False, False, False,  True,
       False, False,  True, False,  True,  True, False, False, False,
        True, False, False,  True, False,  True,  True, False, False,
       False, False, False, False, False,  True,  True, False, False,
       False,  True,  True, False, False, False, False, False,  True,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False,  True, False, False,
       False,  True, False,  True, False, False, False,  True, False,
       False, False, False, False,  True, False, False, False, False,
        True, False, False, False, False, False, False, False, False,
        True, False, False,  True, False, False, False, False, False,

(<module>)>>> sess.run(tf.reshape(tf.where(tf.equal(assignments, c)),[1,-1])).shape
(1, 45)

(<module>)>>> sess.run(tf.reshape(tf.where(tf.equal(assignments, c)),[1,-1]))
array([[  3,   8,  10,  13,  22,  26,  29,  31,  32,  36,  39,  41,  42,
         50,  51,  55,  56,  62,  78,  82,  84,  88,  94,  99, 108, 111,
        117, 124, 133, 135, 136, 141, 149, 152, 156, 157, 159, 161, 162,
        170, 175, 187, 190, 195, 197]])
```

for c = 0

K-means with TensorFlow

- Example:

```
# 3. Then, calculate the distances between points and centroids and determine the
#    cluster assignments:
distances = tf.reduce_sum(tf.square(tf.sub(points_expanded, centroids_expanded)), 2)
assignments = tf.argmin(distances, 0)

# 4. Next, we can compare each cluster with a cluster assignments vector, get points
#    assigned to each cluster, and calculate mean values. These mean values are refined
#    centroids, so let's update the centroids variable with the new values:
means = []
for c in range(clusters_n):
    means.append(tf.reduce_mean( tf.gather(points,
                 tf.reshape(tf.where(tf.equal(assignments, c)),[1,-1])),reduction_indices=[1]))

new
upd
ini
```

```
(<module>)>>> sess.run(tf.gather(points, tf.reshape(tf.where(tf.equal(assignments, c)),[1,-1]))).shape
(1, 45, 2)

(<module>)>>> sess.run(tf.gather(points, tf.reshape(tf.where(tf.equal(assignments, c)),[1,-1])))
```

```
(<module>)>>> sess.run((tf.reduce_mean( tf.gather(points,
                tf.reshape(tf.where(tf.equal(assignments, c)),[1,-1])),reduction_indices=[1]))).shape
(1, 2)

(<module>)>>> sess.run((tf.reduce_mean( tf.gather(points,
                tf.reshape(tf.where(tf.equal(assignments, c)),[1,-1])),reduction_indices=[1])))
array([[1.97671635, 7.45052149]])
```

```
                [3.04537011, 9.07137666],
                [1.11503944, 8.77419734],
```

K-means with TensorFlow

- Example:

```
34  # 3. Then, calculate the distances between points and centroids and determine the
35  #    cluster assignments:
36  distances = tf.reduce_sum(tf.square(tf.sub(points_expanded, centroids_expanded)), 2)
37  assignments = tf.argmin(distances, 0)
38
39  # 4. Next, we can compare each cluster with a cluster assignments vector, get points
40  #    assigned to each cluster, and calculate mean values. These mean values are refined
41  #    centroids, so let's update the centroids variable with the new values:
42  means = []
43  for c in range(clusters_n):
44      means.append(tf.reduce_mean( tf.gather(points,
45                  tf.reshape(tf.where(tf.equal(assignments, c)),[1,-1])),reduction_indices=[1]))
46
47  new_centroids = tf.concat(means, 0)
48
49  update_centroids = tf.assign(centroids, new_centroids)
50  init = tf.global_variables_initializer()
```

```
(<module>)>>> sess.run(distances).shape
(3, 200)

(<module>)>>> sess.run(assignments).shape
(200,)

(<module>)>>> sess.run(new_centroids).shape
(3, 2)

(<module>)>>> sess.run(update_centroids).shape
(3, 2)
```

K-means with TensorFlow

- Example:

```
49  # 5. Next we run the graph. For each iteration, we update the centroids and return
50  #    their values along with the cluster assignments values:
51  with tf.Session() as sess:
52    sess.run(init)
53    for step in range(iteration_n):
54      [_, centroid_values, points_values, assignment_values] = sess.run([update_centroids,
    centroids, points, assignments])
55
56  # 6. Finally, we display the coordinates of the final centroids and a multi-colored
57  #    scatter plot showing how the data points have been clustered:
58  print("centroids" + "\n", centroid_values)
59
60  plt.scatter(points_values[:, 0], points_values[:, 1], c=assignment_values, s=50, alpha=0.5)
61  plt.plot(centroid_values[:, 0], centroid_values[:, 1], 'kx', markersize=15)
62  plt.pause(1)
63
64  # The data in a training set is grouped into clusters as the result of implementing
65  # the k-means algorithm in TensorFlow.
```

K-means with TensorFlow

- Example: