TensorFlow Basics

In [3]:

```
import tensorflow as tf
```

Hello TensorFlow World!

In [5]:

```
# First, declare two string constant tensors

hello = tf.constant("Hello ")

world = tf.constant("Wolrd")
```

In [6]:

```
type(hello)
```

Out[6]:

```
tensorflow.python.framework.ops.Tensor
```

In [7]:

```
print(hello)
```

```
Tensor("Const:0", shape=(), dtype=string)
```

In [8]:

```
# If we want to concatenate these two tensors of strings
# --> MUST run the operation in a TF session to get the result

with tf.Session() as sess:
    result = sess.run(hello+world)
```

In [11]:

```
# Then print the result

print(result)
```

```
b'Hello Wolrd'
```

In [ ]:

```

```

In [ ]:

```

```

Add Two Constant Integer Tensors

In [112]:

```
1 import tensorflow as tf
```

In [113]:

```
1 a = tf.constant(10)
2
3 b = tf.constant(20)
```

In [114]:

```
1 type(a)
```

Out[114]:

```
tensorflow.python.framework.ops.Tensor
```

In [115]:

```
1 # The result is a tensor
2 a+b
```

Out[115]:

```
<tf.Tensor 'add_2:0' shape=() dtype=int32>
```

In [118]:

```
1 # addOp_1 is a tensor
2
3 addOp_1 = a+b
```

In [119]:

```
1 addOp_1
```

Out[119]:

```
<tf.Tensor 'add_4:0' shape=() dtype=int32>
```

In [122]:

```
1 # If we want to actually perform the operation
2 # --> MUST run the operation in a TF session to get the real result
3
4 with tf.Session() as sess:
5     result = sess.run(addOp_1)
```

In [123]:

```
1 # Then print the result
2
3 print(result)
```

```
30
```

In [ ]:

```
1
```

Matrix Tensors

In [12]:

```
1 import tensorflow as tf
```

In [13]:

```
1 const = tf.constant(10)
```

In [14]:

```
1 int_mat = tf.fill((3, 3), 10)
```

In [15]:

```
1 myzeros = tf.zeros((3, 3))
```

In [16]:

```
1 myones = tf.ones((3,3))
```

In [17]:

```
1 # This is a matrix of (3, 3) of random values following the normal distrubution
2
3 myrandn = tf.random_normal((3,3), mean=0, stddev=1.0)
```

In [18]:

```
1 # This is a matrix of (3, 3) of random values between 0 and 1, following the uniform d
2
3 myrandu = tf.random_uniform((3,3), minval=0, maxval=1)
```

In [19]:

```
1 myzeros
```

Out[19]:

```
<tf.Tensor 'zeros:0' shape=(3, 3) dtype=float32>
```

In [20]:

```
1 # Declare a Python list consists of all the above contants and matrices
2 list_ops =[const, int_mat, myzeros, myones, myrandn, myrandu]
```

In [21]:

```
# Print out the values of these constants
# We have to run them in a TensorFlow session

with tf.Session() as sess:
    for op in list_ops:
        print(sess.run(op))
        print("\n")
```

```
10


[[10 10 10]
 [10 10 10]
 [10 10 10]]


[[ 0.  0.  0.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]]


[[ 1.  1.  1.]
 [ 1.  1.  1.]
 [ 1.  1.  1.]]


[[ 0.07902075  0.75892991  0.88125086]
 [ 1.48290956 -0.25409916  1.02617967]
 [ 0.06318574 -1.34071517 -2.22565365]]


[[ 0.18044448  0.22187924  0.08576691]
 [ 0.94796872  0.29433632  0.39441597]
 [ 0.84827399  0.49283469  0.53558969]]
```

Use op.eval()

In [22]:

```
import tensorflow as tf
```

In [25]:

```
# Running an operation: Can use op.eval() instead of tf.Session().run()

with tf.Session() as sess:
    for op in list_ops:
        print(op.eval())
        print("\n")
```

```
10


[[10 10 10]
 [10 10 10]
 [10 10 10]]


[[ 0.  0.  0.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]]


[[ 1.  1.  1.]
 [ 1.  1.  1.]
 [ 1.  1.  1.]]


[[-1.87558389  0.31252941 -0.1943495 ]
 [-0.116242   -1.17692435 -1.2005899 ]
 [ 1.53036153 -1.88336897  1.19986761]]


[[ 0.945014    0.01303291  0.17202246]
 [ 0.13927543  0.23197246  0.05567217]
 [ 0.16254103  0.77079165  0.02621484]]
```

Shape of Tensors

In [6]:

```
import tensorflow as tf
```

In [7]:

```
# Declare a constant tensor that is a matrix of two rows and two columns
# First row: 1, 2; 2nd row: 3, 4

# VIP NOTES:

# A matrix: [ ]
# ROWS: --> Each row inside this matrix is an embedded array:
# ----> 1 row: [ [] ]
# ----> 2 rows: [ [], []]

# COLUMNS: --> Each column is one value inside the row array
# ----> 2 rows, 1 column: [ [1], [2] ]
# ----> 2 rows, 2 columns: [ [1, 3], [2, 4] ]
# ----> 2 rows, 3 columns: [ [1, 3, 5], [2, 4, 6] ]

a = tf.constant([ [1, 2],
                  [3, 4] ])
```

In [8]:

```
a
```

Out[8]:

```
<tf.Tensor 'Const_1:0' shape=(2, 2) dtype=int32>
```

In [9]:

```
a.get_shape()
```

Out[9]:

```
TensorShape([Dimension(2), Dimension(2)])
```

In [10]:

```
# Display the info of the tensor a

print(a)
```

```
Tensor("Const_1:0", shape=(2, 2), dtype=int32)
```

Another Example of Matrices Operations in TensorFlow

In [17]:

```
import tensorflow as tf
```

In [18]:

```
# Declare a constant tensor that is a matrix of two rows and two columns
# First row: 1, 2; 2nd row: 3, 4

a = tf.constant([ [1, 2],
                  [3, 4] ])
```

In [19]:

```
# Declare a constant tensor
# --> A matrix of 2 rows and one column == a vector

b = tf.constant([ [10], [100] ])
```

In [20]:

```
b
```

Out[20]:

```
<tf.Tensor 'Const_4:0' shape=(2, 1) dtype=int32>
```

In [21]:

```
b.get_shape()
```

Out[21]:

```
TensorShape([Dimension(2), Dimension(1)])
```

In [22]:

```
# Perform the matrix operation on a and b

mat_product = tf.matmul(a, b)
```

In [23]:

```
# Display the info about THE MATRIX PRODUCT mat_product
# It should be also a matrix tensor with the shape [2, 1] of data type int32

mat_product
```

Out[23]:

```
<tf.Tensor 'MatMul_1:0' shape=(2, 1) dtype=int32>
```

In [24]:

```
# Display the values of the result
# MUST run it in a TF session

# FIRST: RUN THE OPERATION
with tf.Session() as sess:
        results = sess.run(mat_product)

#THEN: Print out the results
print (results)
```

```
[[210]
 [430]]
```

In [ ]:

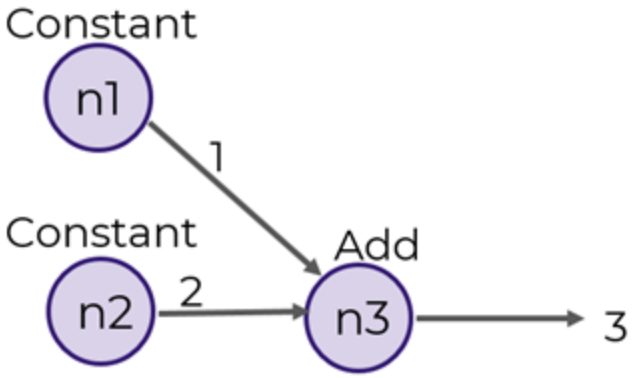
```

```

In [ ]:

```
1
```

tf.Graph() and tf.Session()

TensorFlow uses a dataflow graph
--) to represent some computation in terms of the dependencies between individual operations.
This leads to a low-level programming model:
--) in which you first define the dataflow graph,
--) then create a TensorFlow session to run parts of the graph across a set of local and remote devices.



In [162]:

```
1 import tensorflow as tf
```

Construct Dataflow Graph of the Computation

In [163]:

```
1 # Declare two tf.constant: n1 = 1, n2 = 2
2
3 n1 = tf.constant(1)
4 n2 = tf.constant(2)
```

In [164]:

```
1 # Construct a data graph of the computation
2 # This expression represents a graph:
3 # --) Two input nodes: n1 and n2
4 # --) One operation: addition (+)
5 # --) Output: n3
6
7 n3 = n1 + n2
```

In [165]:

```
# n3 is a tensor output(type: int32) of the opeation +
n3
```

Out[165]:

```
<tf.Tensor 'add_5:0' shape=() dtype=int32>
```

In [166]:

```
print(n3)
```

```
Tensor("add_5:0", shape=(), dtype=int32)
```

Run a tf.Session() to execute the Computation

In [168]:

```
with tf.Session() as sess:
    result= sess.run(n3)

print(result)
```

```
3
```

Default Graphs

In [116]:

```
print (tf.get_default_graph())
```

```
<tensorflow.python.framework.ops.Graph object at 0x00000213156AA7B8>
```

In [117]:

```
g = tf.Graph()
```

In [118]:

```
g
```

Out[118]:

```
<tensorflow.python.framework.ops.Graph at 0x21316822128>
```

In [120]:

```
# Display the info of the graph

print(g)
```

```
<tensorflow.python.framework.ops.Graph object at 0x0000021316822128>
```

In [121]:

```
graph_one = tf.get_default_graph()
```

In [122]:

```
print(graph_one)
```

```
<tensorflow.python.framework.ops.Graph object at 0x00000213156AA7B8>
```

In [123]:

```
graph_two = tf.Graph()
```

In [124]:

```
print(graph_two)
```

```
<tensorflow.python.framework.ops.Graph object at 0x0000021316817BA8>
```

In [126]:

```
# Set graph_two as the default graph temporarily
# Print out the address of the default graph

with graph_two.as_default():
    print(graph_two)
```

```
<tensorflow.python.framework.ops.Graph object at 0x0000021316817BA8>
```

In [127]:

```
with graph_two.as_default():
    print(graph_two is tf.get_default_graph())
```

```
True
```

In [128]:

```
# graph_two is still a normal graph

print(graph_two is tf.get_default_graph())
```

```
False
```

TensorFlow: Variables

Declare tf.Variable()

In [124]:

```
import tensorflow as tf
```

In [125]:

```
aTensor = tf.random_uniform ((3, 3), 0, 1)
```

In [126]:

```
aTensor
```

Out[126]:

```
<tf.Tensor 'random_uniform_6:0' shape=(3, 3) dtype=float32>
```

In [127]:

```
a_tf_var = tf.Variable(initial_value=aTensor)
```

In [128]:

```
a_tf_var
```

Out[128]:

```
<tf.Variable 'Variable_7:0' shape=(3, 3) dtype=float32_ref>
```

In [129]:

```
print(a_tf_var)
```

```
<tf.Variable 'Variable_7:0' shape=(3, 3) dtype=float32_ref>
```

In [130]:

```
# Cause an error: Variable must be initialized first

"""
with tf.Session() as sess:
    result=sess.run(a_tf_var)
"""

```

Out[130]:

```
'\nwith tf.Session() as sess:\n    result=sess.run(a_tf_var)\n'
```

Initialize a tf.Variable

In [131]:

```
# Get initializer, the operation to initialize the variable

initVar = tf.global_variables_initializer()
```

In [132]:

```
# Display its info

initVar
```

Out[132]:

```
<tf.Operation 'init_6' type=NoOp>
```

In [133]:

```
# This statement performs the operation to initialize the variable

with tf.Session() as sess:
    sess.run(initVar)
    results = sess.run(a_tf_var)
```

In [134]:

```
# Display the values of the variable
print (results)
```

```
[[ 0.76514602  0.49285698  0.89413798]
 [ 0.56111598  0.30426228  0.35631084]
 [ 0.66837347  0.60118461  0.0989747 ]]
```

Another way

In [135]:

```
with tf.Session() as sess:
    sess.run(initVar)
    print(sess.run(a_tf_var))
```

```
[[ 0.1923238   0.56539917  0.55756402]
 [ 0.58319819  0.37634909  0.84810627]
 [ 0.63401532  0.00871527  0.70810938]]
```

TensorFlow: Placeholders & feed_dict

In [147]:

```
import tensorflow as tf
```

In [150]:

```
x1 = tf.placeholder(tf.float32)
```

In [153]:

```
# Feed the scalar 111 into the placeholder x1
# Then print it out

with tf.Session() as session:
    result = session.run(x1, feed_dict={x1: [111]})

print(result)
```

```
[ 111.]
```

In [154]:

```
x2 = tf.placeholder(tf.float32, None)
y = x2 * 2
```

In [155]:

```
# Run a tf.Session(), feed data into tf.placehpolder
# And print out the results

with tf.Session() as session:
    result = session.run(y, feed_dict={x2: [1, 2, 3]})

print(result)
```

```
[ 2.  4.  6.]
```

Multi-Dimensional Placeholders:

Placeholders can also have multiple dimensions, allowing for storing arrays.
In the following example, we create a 2x3 matrix, and store some numbers in it.
We then use the same operation as before to do element-wise doubling of the numbers.

In [173]:

```
# NOTES:
# Parameters: shape = [None, 3]
# --) The placeholder x3 has a shape of Nx3 matrix
# --) Where N can be any number >= 1

x3 = tf.placeholder(tf.float32, shape = [None, 3])
y = x * 2
```

In [174]:

```
with tf.Session() as session:
    x_data = [[1, 2, 3],
              [4, 5, 6],]
    result = session.run(y, feed_dict={x: x_data})

print(result)
```

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

In [ ]:

```

```