Tensor Flow

**Install**:

pip install tensorflow

**To check Tensor Flow version:**

**Code**:

import tensorflow as tf

print(tf.\_\_version\_\_)

**Output**:

2.7.0

## **TensorFlow Architecture**:

→Tensor flow architecture works in three parts:

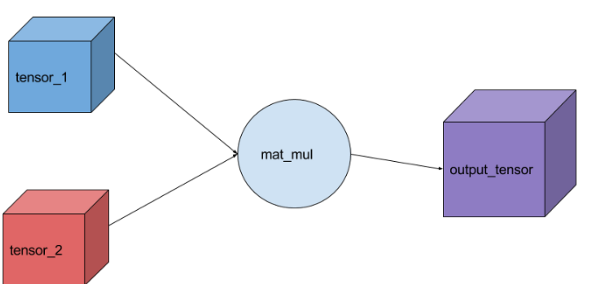
* Preprocessing the data
* Build the model
* Train and estimate the model

→It is called Tensorflow because it takes input as a multi-dimensional array, also known as **tensors**.

→You can construct a sort of **flowchart** of operations (called a Graph) that you want to perform on that input.

The input goes in at one end, and then it flows through this system of multiple operations and comes out the other end as output.

This is why it is called TensorFlow because the tensor goes in it flows through a list of operations, and then it comes out the other side.



The code above is creating two constant tensors and multiplying them together and outputting our result.

This is a trivial example that demonstrates how you can create a graph and run the session.

All inputs needed by the op are run automatically.

They’re typically ran in parallel. This session run actually causes the execution of three operations in the graph, creating the two constants then the matrix multiplication.

**Graph:**

→The constants and operation that we created above was automagically added to the graph in TensorFlow.

→The graph default is instantiated when the library is imported.

→Creating a Graph object instead of using the default graph is useful when creating multiple models in one file that do not depend on each other.

new\_graph = tf.Graph()

with new\_graph.as\_default():  
 new\_g\_const = tf.constant([1., 2.])

→any variables or operations used outside of the with new\_graph.as\_default() will be added to the default graph that is created when the library is loaded. You can even get a handle to the default graph with

default\_g = tf.get\_default\_graph()

**Session:**

→There are two kinds of Session objects in TensorFlow:

**tf.Session()**

→This encapsulates the environment that operations and tensors are executed and evaluated.

→Sessions can have their own variables, queues and readers that are allocated. So it’s important to use the close() method when the session is over.

→There are 3 arguments for a Session, all of which are optional.

1. target — The execution engine to connect to.
2. graph — The Graph to be launched.
3. config — A ConfigProto protocol buffer with configuration options for the session

To run one “step” of the TensorFlow computation this function is called and all of the necessary dependencies for the graph to execute are ran.

**tf.InteractiveSession()**

→This is the exact same as tf.Session() but is targeted for using IPython and Jupyter Notebooks that allows you to add things and use Tensor.eval() and Operation.run() instead of having to do Session.run() every time you want something to be computed.

sess = tf.InteractiveSession()  
a = tf.constant(1)  
b = tf.constant(2)  
c = a + b  
# instead of sess.run(c)  
c.eval()

InteractiveSession allows so that you don't have to explicitly pass Session object.

**Variables:**

→Variables in TensorFlow are managed by the Session.

→They persist between sessions which are useful because Tensor and Operation objects are immutable.

→Variables can be created by tf.Variable().

tensorflow\_var = tf.Variable(1, name="my\_variable")

Most of the time you will want to create these variables as tensors of zeros, ones or random values:

* tf.zeros() — creates a matrix full of zeros
* tf.ones() — creates a matrix full of ones
* tf.random\_normal() — a matrix with random uniform values between an interval
* tf.random\_uniform() — random normally distributed numbers
* tf.truncated\_normal() — same as random normal but doesn’t include any numbers more than 2 standard deviations.

These functions take an initial shape parameter where the dimension of the matrix is defined. For example:

# 4x4x4 matrix normally distributed mean 0 std 1  
normal = tf.truncated\_normal([4, 4, 4], mean=0.0, stddev=1.0)

To have your variable set to one of these matrix helper functions:

normal\_var = tf.Variable(tf.truncated\_normal([4,4,4] , mean=0.0, stddev=1.0)

To have these variables initialized you must use TensorFlow’s variable initialization function then pass it to the session. This way when multiple sessions are ran the variables are the same.

init = tf.initialize\_all\_variables()  
sess = tf.Session()  
sess.run(init)

If you’d like to completely change the value of a variable you can use Variable.assign() operation, this must be run in a session update the value.

initial\_var = tf.Variable(1)

changed\_var = initial\_var.assign(initial\_var + initial\_var)

init = tf.initialize\_all\_variables()  
sess = tf.Session()  
sess.run(init)

sess.run(changed\_var)  
# 2

sess.run(changed\_var)  
# 4

sess.run(changed\_var)  
# 8

# .... and so on

→Sometimes you would like to add a counter inside your model this is where you can do a Variable.assign\_add() method which takes a numeric parameter and increments it by the parameter. Similarly there is Variable.assign\_sub().

counter = tf.Variable(0)

sess.run(counter.assign\_add(1))  
# 1

sess.run(counter.assign\_sub(1))  
# back to 0

#### Scope:

→To control the complexity of models and make them easier to break down into individual pieces TensorFlow has scopes.

→Scopes are very simple and even help break down your model when using TensorBoard (which will be covered in Part 2). Scopes can even be nested inside of other scopes.

with tf.name\_scope("Scope1"):  
 with tf.name\_scope("Scope\_nested"):  
 nested\_var = tf.mul(5, 5)

Scopes may not seem that powerful right now but used in collaboration with TensorBoard and they’re very useful.

P: **Hello World with out session**

**Logic**:

**Code**:

import tensorflow as tf

wish=tf.constant("Hello World")

print(wish)

**Output**:

Tensor("Const\_4:0", shape=(), dtype=string)

P:Multiplication

**Logic**:

**Code**:

# Build a graph.

a = tf.constant(5.0)

b = tf.constant(6.0)

c = a \* b

# Launch the graph in a session.

sess = tf.Session()

# Evaluate the tensor `c`.

print(sess.run(c))

**Output**:

30.0