

TensorFlow - Installation and principles

TensorFlow is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) that flow between them. This flexible architecture lets you deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device without rewriting code.

TensorFlow also includes TensorBoard, a utility to display a picture of the computational graph.

TensorFlow was originally developed by researchers and engineers working on the Google Brain team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research

Update brew installation

```
$ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Once in a while, update and upgrade brew with

```
$ brew update && brew upgrade
```

Install python3

Mac OS Sierra comes with python 2.7 out of the box.

- Install the more advanced python3 using
- ```
$ brew install python3
```
- Check python3 version
- ```
$ python3 -V
```

When installing python, you might need to disable System Integrity Protection (SIP) to permit any entity other than Mac App Store to install software. So you will be asked to input your administrator password.

Installing pip3

pip installs and manages software packages written in python.

- To install pip3
- ```
$ sudo easy_install --upgrade pip3
```
- ```
$ sudo easy_install --upgrade six
```
- Check pip3 version
- ```
$ pip3 -V
```
- pip 9.0.1 from /usr/local/lib/python3.6/site-packages (python 3.6)

## Installing TensorFlow using pip3

```
$ pip3 install tensorflow
```

## Uninstalling TensorFlow using pip3

```
$ pip3 uninstall tensorflow
```

## Run a short TensorFlow program

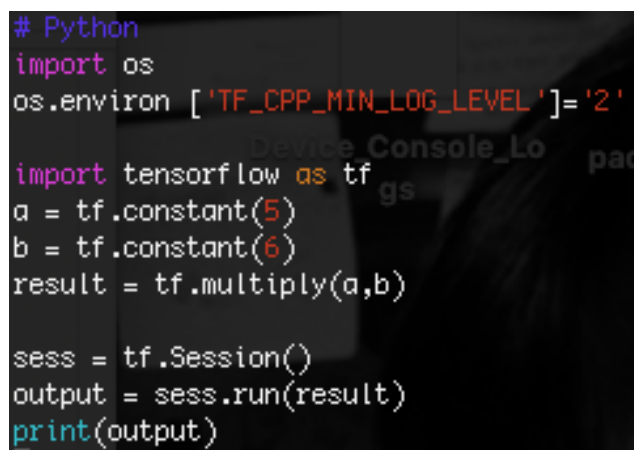
- You can create a TensorFlow program by using the python3 interactive shell

```
$ python3
```

Enter the following short program inside the python interactive shell:

```
>>> import tensorflow as tf
>>> hello = tf.constant('Hello, TensorFlow!')
>>> sess = tf.Session()
>>> sess.run(hello)
'Hello, TensorFlow!'
>>> a = tf.constant(10)
>>> b = tf.constant(32)
>>> sess.run(a+b)
42
```

- You can also create a new file, e.g. tfbasics.py, edit it

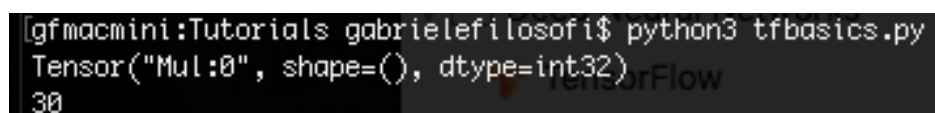


```
Python
import os
os.environ ['TF_CPP_MIN_LOG_LEVEL']='2'

import tensorflow as tf
a = tf.constant(5)
b = tf.constant(6)
result = tf.multiply(a,b)

sess = tf.Session()
output = sess.run(result)
print(output)
```

Then from the shell, execute the script and get the result



```
[gfmacmini:Tutorials gabrielefilosofi$ python3 tfbasics.py
Tensor("Mul:0", shape=(), dtype=int32)
30
```

Let's examine the source a little bit.

The first two lines get rid of some warning logs (see below).  
The following 4 lines define the computational graph, or the model.  
The subsequent 3 lines define and run a session where everything gets really executed in background from the CPU or GPU.

## Undesired Warnings

When you run a session you may encounter warning logs like this

```
2017-07-23 23:25:46.528306: W tensorflow/core/platform/
cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use
SSE4.2 instructions, but these are available on your machine and could speed
up CPU computations.
```

To remove them add the following lines to the python source file

```
Python
import os
os.environ ['TF_CPP_MIN_LOG_LEVEL']='2'
```

## Install Sublime Text 3 editor

- Download and install from <https://www.sublimetext.com/3>
- To install packages  
Click *View > Show Console*  
Once open, paste the following Python code

```
import urllib.request,os,hashlib; h = 'df21e130d211cfc94d9b0905775a7c0f' +
'1e3d39e33b79698005270310898eea76'; pf = 'Package Control.sublime-
package'; ipp = sublime.installed_packages_path();
urllib.request.install_opener(urllib.request.build_opener(urllib.request.ProxyHa
ndler())); by = urllib.request.urlopen('http://packagecontrol.io/' + pf.replace(' ',
'%20')).read(); dh = hashlib.sha256(by).hexdigest(); print('Error validating
download (got %s instead of %s), please try manual install' % (dh, h)) if dh != h
else open(os.path.join(ipp, pf), 'wb').write(by)
```

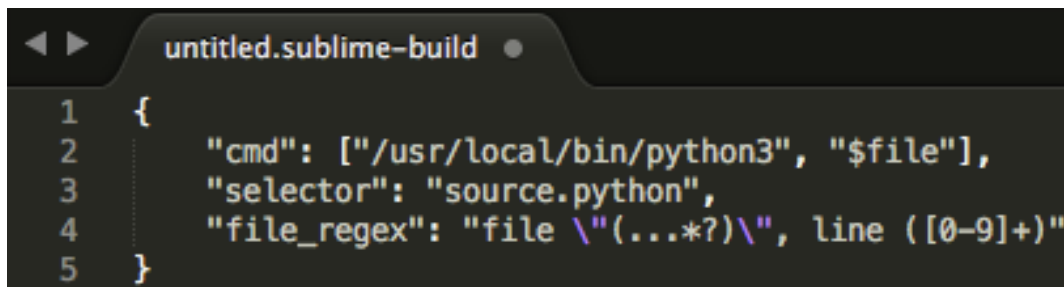
This code creates the Installed Packages folder for you (if necessary), and then downloads the *Package Control.sublime-package* into it

## Get Python3 to Sublime Text 3

- Click *Tools > Build > New Build System* and enter the following:  
{  
"cmd": ["/usr/local/bin/python3", "\$file"]  
, "selector": "source.python"

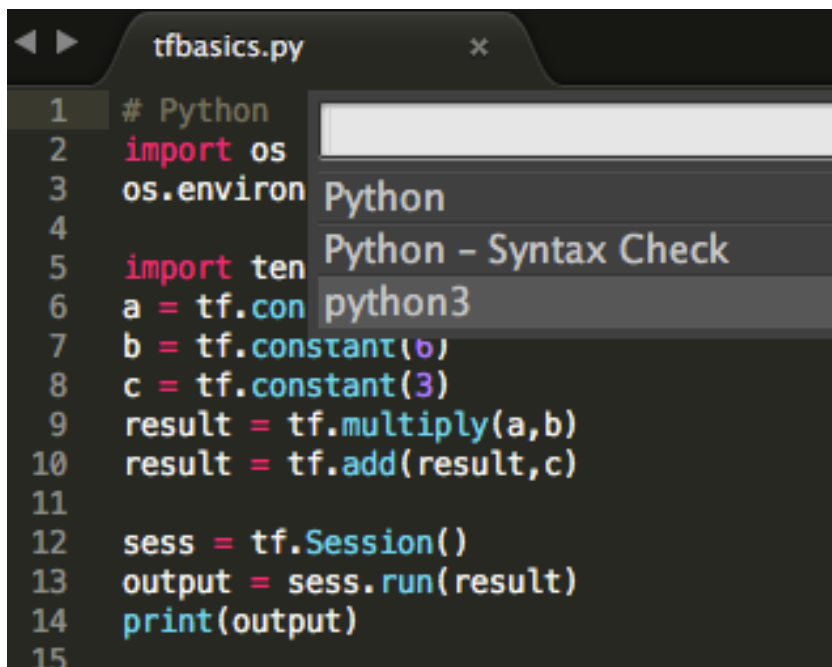
```
, "file_regex": "file \"(...*?)\", line ([0-9]+)"
}
```

Note: to know the python exec path you ran *which python3* from terminal



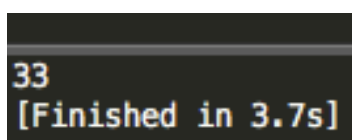
```
1 {
2 "cmd": ["/usr/local/bin/python3", "$file"],
3 "selector": "source.python",
4 "file_regex": "file \"(...*?)\", line ([0-9]+)"
5 }
```

- save it to the following directory:  
~/Library/Application Support/Sublime Text 3/Packages/User
- Close and run Sublime Text 3
- Open a python script, for example tfbasics.py, then click Tools > Build With > Python3



```
1 # Python
2 import os
3 os.environ
4
5 import tensorflow as tf
6 a = tf.constant(6)
7 b = tf.constant(6)
8 c = tf.constant(3)
9 result = tf.multiply(a,b)
10 result = tf.add(result,c)
11
12 sess = tf.Session()
13 output = sess.run(result)
14 print(output)
15
```

Click Enter and you get the script execution done



```
33
[Finished in 3.7s]
```

ToDo: <http://filipminev.com/post/14262857223/9-reasons-you-must-install-sublime-text-2-code>

## Jupyter Notebook (IPython)

- Jupyter Notebook, formerly known as the IPython Notebook, is a web-based interactive computing system that enables users to author

documents that include live code, images, comments, formulae, plots, HTML, images and video all together. These documents contain a full record of a computation and its results and can be shared on email, [Dropbox](#), version control systems (like git/[GitHub](#)) or [nbviewer.ipython.org](#).

- Jupyter is quite extensible, supports many programming languages and is easily hosted on your computer or on almost any server — you only need to have ssh or http access. Best of all, it's completely free
- The name Jupyter is an indirect acronym of the three core languages it was designed for: JULia, PYThon, and R
- Notebook documents are just files on your local filesystem with a .ipynb extension

## Installing Jupyter Notebook using pip3

- Install Jupyter Notebook
- ```
$ pip3 install jupyter
```
- Install a useful library to plot data
- ```
$ pip3 install matplotlib
```

## Running Jupyter Notebook

- Move to the top level directory containing your notebooks. For example
- ```
$ cd /Users/gabrielefilosofi/TensorflowProjects/myNotebooks
```

- Run the notebook server
- ```
$ jupyter notebook
```

This will print some information about the notebook server in your terminal, including the URL of the web application (by default, <http://localhost:8888>) and then will open the web browser to this URL.

The Notebook Dashboard get appeared, showing the subdirectories where the notebook server was started.

If you wish, change the server port

```
$ jupyter notebook --port <port number>
```

To start notebook server without the web browser

```
$ jupyter notebook --no-browser
```

The Notebook is organised in Cells.

For example, this is a cell

```
In [2]: import tensorflow as tf
import matplotlib

hello = tf.constant('Hello, Gabriele!')
sess = tf.Session()
print(sess.run(hello))

b'Hello, Gabriele!'
```

which you can evaluate by clicking  
Shift+Enter

## TensorFlow principles

- TensorFlow is a software for multidimensional array manipulation
- Computation is defined as a directed acyclic graph (DAG) to optimize an objective function
- Graph is defined in high-level language (Python, C++, go)
- Graph is compiled and optimized
- Graph is executed on available low level devices (CPU, GPU) and platforms (laptop, datacenters, iOS, Android, RaspberryPi)
- Data (tensors) flow through the graph
- TensorFlow can compute gradients automatically
- The basic API is TensorFlow Core. Higher level API are built on top of it

## Tensors

- The tensor's *rank* is N, the number of dimensions. It is the number of indexes you need to represent the generic element of your tensor. For example, a matrix is a tensor of rank 2
- The tensor's *shape* is [dim 0, dim 1, ..., dim N]. It is the set of top boundaries for the ranges of the indexes
- 2 is a tensor with rank 0 and shape []
- [1., 2.] is a tensor with rank 1 and shape [2]
- [[[1.,2.]], [[3.,4.]]] is a tensor with rank 3 and shape [2, 1, 2]
- TensorFlow executes intensive calculation outside Python. Because data transfer outside Python is a high cost operation, the program is structured in two main sections. Define the *computational graph* and running the computational graph in a *session*
- The computational graph is a graph of nodes. Each nodes takes zero or more tensors as inputs and produces a tensor as an output
- To use TensorFlow Core classes use this import statement

```
5 import tensorflow as tf
```

## Constants

The simplest node is a constant.

Let's define a graph with three nodes, two constant tensors (a scalar and a simple vector) and a multiply operation between the two constants

```
7 scalar = tf.constant(5.0, tf.float32)
8 vector = tf.constant([.1,.2,0.3], dtype=tf.float32)
9 result = tf.multiply(scalar, vector)
```

If we print those three objects

```
10 print(scalar,vector,result)
```

we don't get numbers, but only the nodes structure

```
[gfmacmini:Tutorials gabrielefilosofi$ python3 tfbasics.py
Tensor("Const:0", shape=(), dtype=float32) Tensor("Const_1:0", shape=(3,), dtype=float32) Tensor("Mul:0", shape=(3,), dtype=float32)
```

To evaluate the nodes we need to run the graph into a session

```
12 sess = tf.Session()
13 output = sess.run(result)
14 print(output)
```

then the node numeric output is given

```
[gfmacmini:Tutorials gabrielefilosofi$ python3 tfbasics.py
[0.5 1. 1.5]
```

## Placeholders

A *placeholder* is a node which can accept external input values

```
7 a = tf.placeholder(tf.float32)
8 b = tf.placeholder(tf.float32)
9 result = tf.multiply(a,b)
```

The preceding three lines are a bit like a function with parameters a and b.  
The print statement

```
11 print(a,b,result)
```

gives only the structure

```
Tensor("Placeholder:0", dtype=float32) Tensor("Placeholder_1:0", dtype=float32) Tensor("Mul:0", dtype=float32)
```

To evaluate the function with concrete values, we can pass them directly when running the session (*feed\_dict*)

```
12 sess = tf.Session()
13 output = sess.run(result, {a: .1, b: [5.0, 2.0]})
14 print(output)
```

```
[gfmacmini:Tutorials gabrielefilosofi$ python3 tfbasics.py
[0.5 0.2]
```

## Variables

A *variable* is a trainable parameter of a graph.

Let's define two variables W and b, with initial values 0.3 and -0.3, a input placeholder x and a output node y (the linear regression of x)

```

7 W = tf.Variable([.3], tf.float32)
8 b = tf.Variable([-.3])
9 x = tf.placeholder(tf.float32)
10 y = W * x + b

```

- The variable's initialisation takes place only when a special operation is called in the session

```

13 sess.run(tf.global_variables_initializer())

```

- x is a placeholder, then you can evaluate the y for several values of x in a single statement, as follows

```

14 output = sess.run(y, {x: [1, 2, 3, 4]})

```

The final session looks like this

```

12 with tf.Session() as sess:
13 sess.run(tf.global_variables_initializer())
14 output = sess.run(y, {x: [1, 2, 3, 4]})
15 print(output)

```

And the result is

```

[gfmacmini:Tutorials gabrielefilosofi$ python3 tfbasics.py
[0. 0.30000001 0.60000002 0.90000004]

```

## Cost

A cost (or loss) function measures how far apart the current model is from the provided data

The standard cost model for a linear regression is the sum of squares of the deltas between the current model (y) and the provided data (*real\_y*)

```

11 real_y = tf.placeholder(tf.float32)
12 delta_y = tf.square(y - real_y)
13 cost = tf.reduce_sum(delta_y)

```

The *reduce\_sum* function is a summation over all *delta\_y*.

Now let's run the session and print the total cost over a set of values for x and *real\_y* placeholders

```

17 output = sess.run(cost, {x: [1, 2, 3, 4], real_y: [0, -3, -6, -9]})

```

```

[gfmacmini:Tutorials gabrielefilosofi$ python3 tfbasics.py
152.46

```

## Assign

For our particular example, the optimal parameters that pull the cost down to zero are  $W = -3.0$  and  $b = 3.0$ .



We can check out this by using the assign function

```
15 fixW = tf.assign(W, [-3.])
16 fixb = tf.assign(b, [3.])
```

```
20 sess.run([fixW, fixb])
```

```
[gfmacmini:Tutorials gabrielefilosofi$ python3 tfbasics.py
0.0
```

## Optimizers

We guessed the perfect values for W and b. But a learning machine has find those values automatically.

An *optimizer* is a object that slowly changes each variable in order to minimise the cost function.

The simplest optimiser is the *gradient descent*.

```
18 optimizer = tf.train.GradientDescentOptimizer(0.01)
19 train = optimizer.minimize(cost)
```

Then we run the train over the data set

```
24 for i in range(1000):
25 sess.run(train, {x: [1, 2, 3, 4], real_y: [0, -3, -6, -9]})
26 print(sess.run([W,b]))
27 print('cost:', sess.run(cost, {x: [1, 2, 3, 4], real_y: [0, -3, -6, -9]}))
```

The final execution gives a good approximation to the solution

```
[array([-2.99999213], dtype=float32), array([2.99997663], dtype=float32)]
cost: 3.56806e-10
```