TUTORIAL 3

INTRODUCTION TO TENSORFLOW

CSCI3230 (2019-2020 First Term)

By Qi SUN (qsun@cse.cuhk.edu.hk)

Outline

- Environment Setup
- Understanding TensorFlow and its terminologies
- Examples with Codes
- Installation Guide
- Introduction to the course project

TensorFlow

- **TensorFlow^{TM}** is an open-source machine learning library for research and production.
- **TensorFlow^{TM}** was originally developed by the Google Brain team for Google's research and production purposes and later released on November 9, 2015.

Environment Setup

- It’s recommended to use Anaconda. Install Anaconda, https://docs.anaconda.com/anaconda/install/
- Anaconda: virtual environment platform, the default Python version is 3.7
- There is no official TensorFlow binaries built for Python 3.7.
- We can create a new environment in Anaconda for Tensorflow with Python 3.6.

Environment Setup

- Our automatic grading system is using Python 3.6 and TensorFlow 1.5.0 for CPU

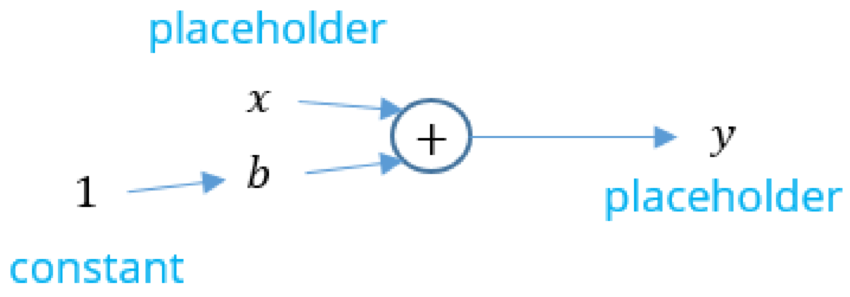
- Don’t use TensorFlow 2.0 RC (new version, unstable).

- The installation guide is attached at the end.

Interpret computation as a graph

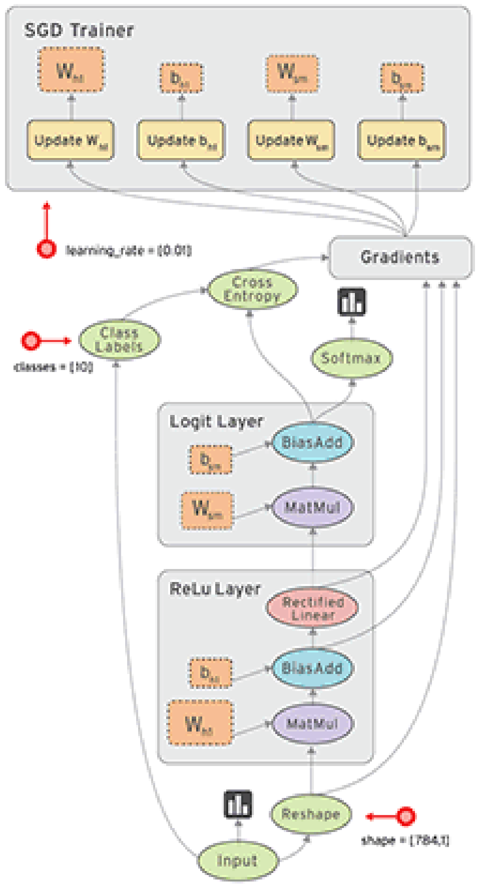
- TensorFlow uses a dataflow graph to represent your computation in terms of the dependencies between individual operations

- Simple example: $y = x + 1$



Interpret computation as a graph

- Complex example: SGD as a dataflow graph
- https://www.tensorflow.org/guide/graphs



Interpret data as tensors

- **Tensors**: multidimensional arrays
- **Rank**
- In TensorFlow, the rank of a tensor is different from the rank of a matrix. It refers to the number of indices required to uniquely select each element of the tensor

- Rank-0 tensors: scalars
- Rank-1 tensors: vectors
- Rank-2 tensors: matrices
- ……

Interpret images as tensors

- Grayscale image: rank-2 tensor
- Color image: rank-3 tensor
- Color image dataset: rank-4 tensor

- NHWC format: num_samples x height x width x channels
- NCHW format: num_samples x channels x height x width

- TensorFlow defaults to NHWC

General Workflow

For both training and testing phase, you are basically doing:

1. **Create TensorFlow objects** that model the calculation you want to carry out
2. **Get the input data** for the model
3. **Run the model** using the input data
4. Do something with the **output**.

Variable, Constant, Placeholder

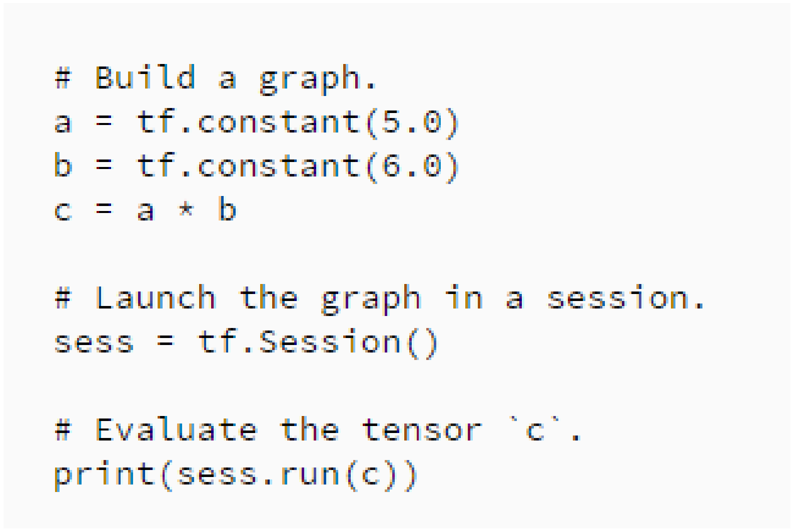
- ***tf.Variable*** is the model weight we want TensorFlow to tune based on some criteria set up in our model
  - Initialization needed

- ***tf.constant*** is the value remains constant during our computation

- ***tf.placeholder*** is an interface between a computational graph element and your data

Session

- A *Session* object encapsulates the environment in which *Operation* objects are executed, and *Tensor* objects are evaluated.

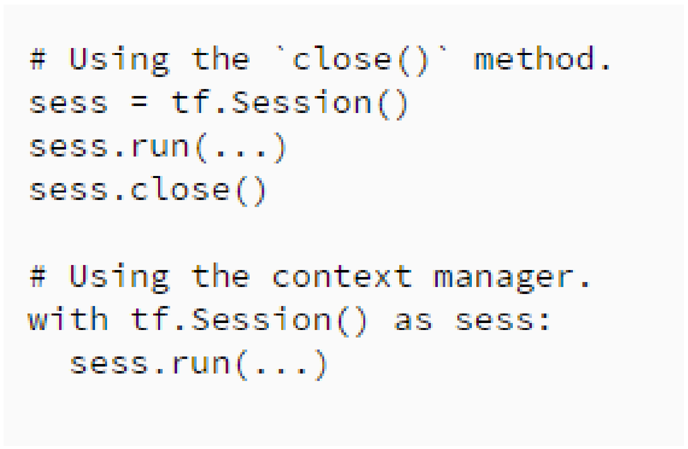


```
# Build a graph.
a = tf.constant(5.0)
b = tf.constant(6.0)
c = a * b

# Launch the graph in a session.
sess = tf.Session()

# Evaluate the tensor `c`.
print(sess.run(c))
```

- Session is like context in programming, explicitly declared
- One *Graph* can have multiple sessions

Session

- A session may own resources, such as variables, queues, and readers. It is important to release these resources when they are no longer required. To do this, either invoke the close() method on the session, or use the session as a context manager.



```
# Using the `close()` method.
sess = tf.Session()
sess.run(...)
sess.close()

# Using the context manager.
with tf.Session() as sess:
  sess.run(...)
```

Input / Output

- Input: feed data to placeholders

```
input1 = tf.placeholder(tf.float32)
input2 = tf.placeholder(tf.float32)
output = tf.mul(input1, input2)

with tf.Session() as sess:
  print(sess.run([output], feed_dict={input1:[7.], input2:[2.]}))

# output:
# [array([ 14.], dtype=float32)]
```

- Output: get the return value of session.run()

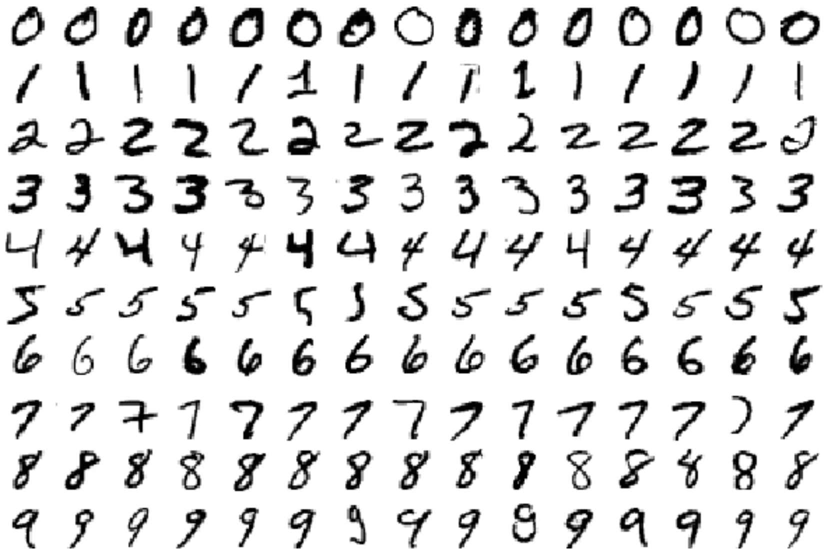
```
input1 = tf.constant([3.0])
input2 = tf.constant([2.0])
input3 = tf.constant([5.0])
intermed = tf.add(input2, input3)
mul = tf.mul(input1, intermed)

with tf.Session() as sess:
  result = sess.run([mul, intermed])
  print(result)

# output:
# [array([ 21.], dtype=float32), array([ 7.], dtype=float32)]
```

Simple Example

```
import numpy as np
import tensorflow as tf
import math

with tf.Session() as session:
    x = tf.placeholder(tf.float32, [1], name='x')
    b = tf.constant(1.0)
    y = x + b # here is our 'model': add one to the input.

    x_in = [2]
    y_final = session.run([y], {x: x_in})
    print(y_final)
```

A more serious example: MNIST

- The MNIST database of handwritten digits
  - a training set of 60,000 examples
  - a test set of 10,000 examples
  - size-normalized
  - centered in a fixed-size image



- train-images-idx3-ubyte.gz
  - training set images
- train-labels-idx1-ubyte.gz
  - training set labels
- t10k-images-idx3-ubyte.gz
  - test set images
- t10k-labels-idx1-ubyte.gz
  - test set labels
- http://yann.lecun.com/exdb/mnist/

Load the MNIST data

- MNIST is an official example for TensorFlow, so there is API for downloading the data using TensorFlow.

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
```

- The code above will create a folder named MNIST_data in your current folder and download the MNIST dataset in it when you run the code for the first time. After that, it will load the dataset and make it available for use.

Start a session

```
import tensorflow as tf
sess = tf.InteractiveSession()
```

Define the input and output

- As we have said, we use placeholder to represent the input and output data, since we don’t know what’s the data in advance.

```
x = tf.placeholder(tf.float32, shape=[None, 784])
y_ = tf.placeholder(tf.float32, shape=[None, 10])
```

- The size of the images in MNIST is 28*28 = 784
- the label for the image is the 10 digits number: 0 to 9
- The number for of the images depends

Self defined function

```
def weight_variable(shape):
  initial = tf.truncated_normal(shape, stddev=0.1)
  return tf.Variable(initial)

def bias_variable(shape):
  initial = tf.constant(0.1, shape=shape)
  return tf.Variable(initial)
```

- The two functions above return tensors (variables) with initial values following different distributions.

Convolution

```
def conv2d(x, W):
  return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')
```

- x is the tensor for the image
- W is the tensor for the filter
- strides is the step length to move
  - why 4 number instead of 2?
- padding is the padding scheme.

Pooling

```
def max_pool_2x2(x):
  return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                        strides=[1, 2, 2, 1], padding='SAME')
```

- x is the input image
- ksize is the size of the area for pooling
  - why [1, 2, 2, 1] instead of [2, 2]?
- strides is as the strides in previous function

First convolutional layer

- Define the filter

```
W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])
```

- Reshape the input data

```
x_image = tf.reshape(x, [-1,28,28,1])
```

- Computational node in the graph

```
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)
```

Second convolutional layer

- Similar to the first layer

```
W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])

h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2)
```

Fully connected (Dense) layer

- A linear operation computing $Ax + b$

```
W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])

h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
```

Dropout

- Randomly set the outputs of some neurons to be zero (for this run), which will makes the model more robust

```
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
```

- keep_prob is the probability that each element is kept

Readout Layer

- This is the last layer, which is fully connected
- To transform the vectors’ dimensions to what we want

```
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])

y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2
```

Cross Entropy

- Cross entropy is a way to measure the difference between the real label and your predicted label

```
cross_entropy = tf.reduce_mean(
    tf.nn.softmax_cross_entropy_with_logits_v2(labels=y_, logits=y_conv))
```

- tf.reduce_mean() calculates the mean of the input tensor along some given dimension
  - and thus reduce the number of dimensions
- tf.nn.softmax_cross_entropy_with_logits_v2() is basically Softmax + Cross Entropy

Optimizer

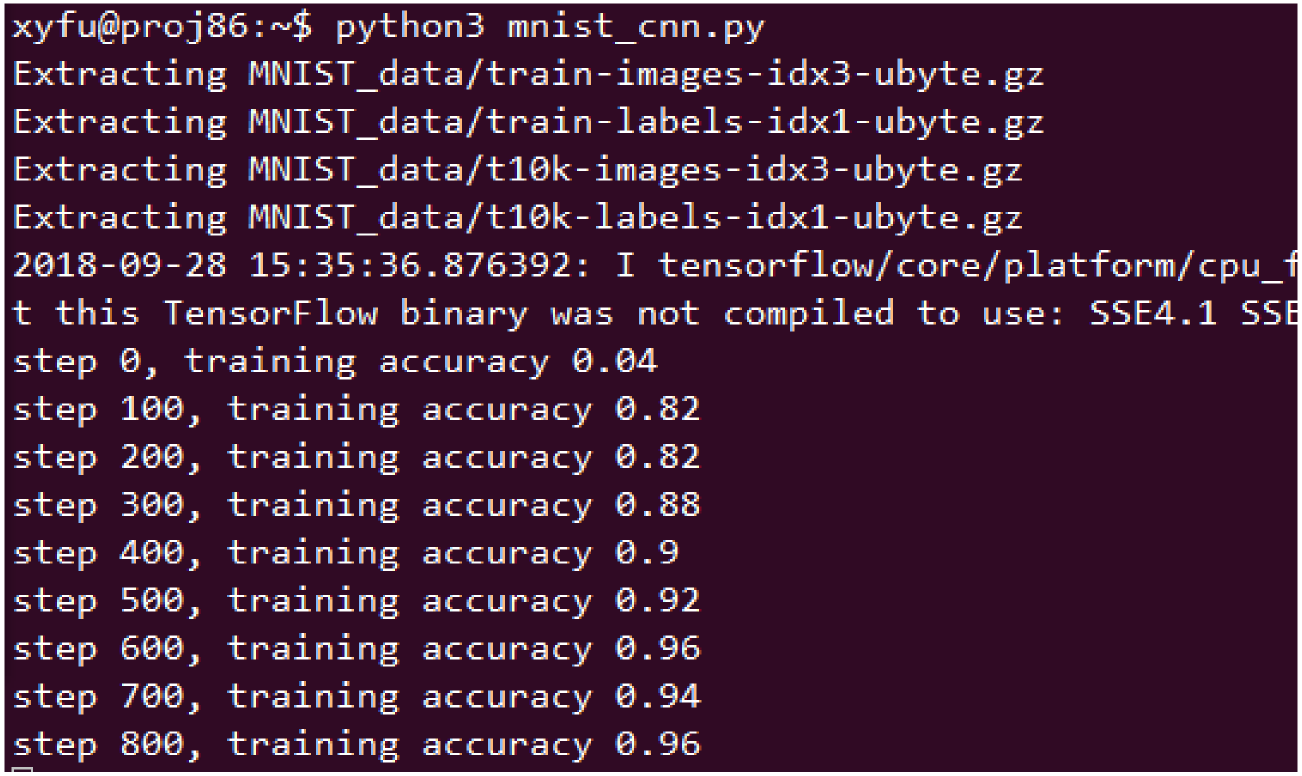
- In TensorFlow, an optimizer calling minimize() creates a node taking care of both computing the gradients and applying them to the variables
- Adam (Adaptive Moment Estimation) is an algorithm for gradient descent optimization, with modified updating policy to attain much better performance

```
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
```

Train and Evaluate the Model

```
correct_prediction = tf.equal(tf.argmax(y_conv,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
sess.run(tf.global_variables_initializer())
for i in range(20000):
  batch = mnist.train.next_batch(50)
  if i % 100 == 0:
    train_accuracy = accuracy.eval(feed_dict={
        x:batch[0], y_: batch[1], keep_prob: 1.0})
    print("step %d, training accuracy %g"%(i, train_accuracy))
  train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob: 0.5})

print("test accuracy %g"%accuracy.eval(feed_dict={
    x: mnist.test.images, y_: mnist.test.labels, keep_prob: 1.0}))
```

Experiments



```
xyfu@proj86:~$ python3 mnist_cnn.py
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
2018-09-28 15:35:36.876392: I tensorflow/core/platform/cpu_f
t this TensorFlow binary was not compiled to use: SSE4.1 SSE
step 0, training accuracy 0.04
step 100, training accuracy 0.82
step 200, training accuracy 0.82
step 300, training accuracy 0.88
step 400, training accuracy 0.9
step 500, training accuracy 0.92
step 600, training accuracy 0.96
step 700, training accuracy 0.94
step 800, training accuracy 0.96
```

Save model

- Copy and paste these two lines to the end of your code

```
saver = tf.train.Saver()
saver.save(sess, './my_model', global_step = 1)
```

Restore model

1. Copy your original code to a new file
2. Delete everything about the training process
3. Add the four lines below before the evaluation code

```
saver = tf.train.Saver()
ckpt = tf.train.get_checkpoint_state('.')
if ckpt and ckpt.model_checkpoint_path:
  saver.restore(sess, ckpt.model_checkpoint_path)
```

- Alternatively, you may just use these lines

```
ckpt = tf.train.get_checkpoint_state('.')
saver = tf.train.import_meta_graph(
    ckpt.model_checkpoint_path + '.meta')
saver.restore(sess, ckpt.model_checkpoint_path)
```

Introduction to the course project

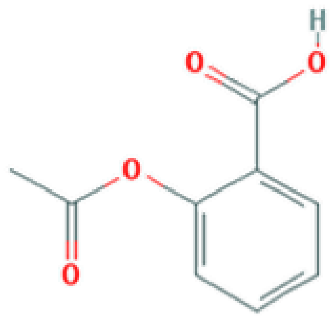
- Description
- Requirements

Drug Molecular Toxicity Prediction

- Design and train a Deep Neural Network using TensorFlow to predict the toxicity of drug molecules from the their chemical expressions.
- Given data: a list of drug molecules and their matrix expressions, and the binary label of whether toxic or not.

SMILES

- Simplified Molecular-Input Line-Entry System (SMILES) is a linear representation for molecular structure using 1D ASCII strings.
- E.g. aspirin



- Its SMILES is
- CC(=O)OC1=CC=CC=C1C(=O)O

Dataset

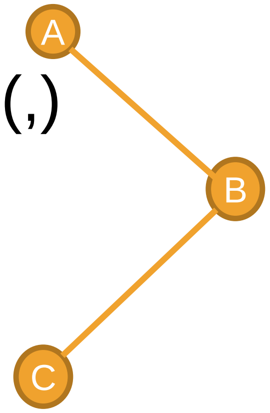
- Tox21_AR-train and Tox21_AR-test are provided for you
- Tox21_AR-score reserved for grading

- Each folder provided contains three files
  - *names_graphs.csv*
  - *names_labels.csv*
  - *names_nodes.csv*

- Details explained in the project specification in server (will upload soon)

names_graphs.csv

- A csv file, contains all the molecules' structures in the Matrix format.
- All the graphs are represented as matrices.
- Separated by comma (,)



	A	B	C
A	0	1	0
B	1	0	1
C	0	1	0

names_labels.csv

- A csv file, each line contains a drug molecule’s index in the set and its toxicity label, where 0 means non-toxic and 1 means toxic, separated by comma (,)

names_nodes.csv

- A csv file, each line contains a node in a drug molecule in the set and its feature.
- This feature is the Atomic Expression.
- Separated by comma (,).
- You can try to use Graph Neural Network to solve this problem.

Project Details

- One person per project
- Auto-submission will be provided
- Repeated submission allowed
- Use CNN or any other kind of DNN
- Use data provided as you like
- Python 3
- TensorFlow 1.5.0 and Numpy only
- Stick to the output format and the file names specified

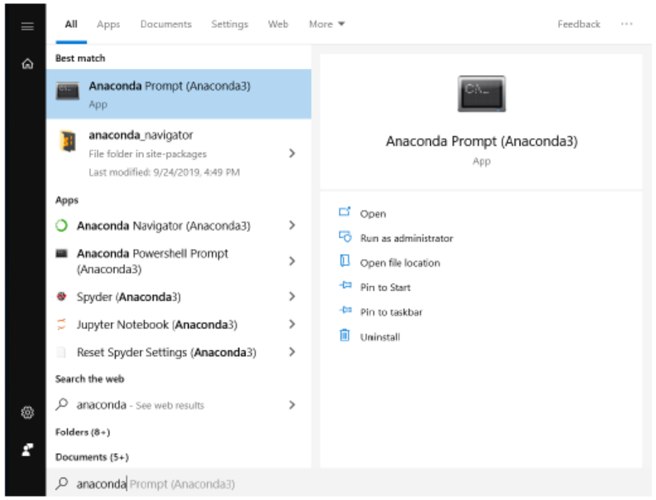
Reference

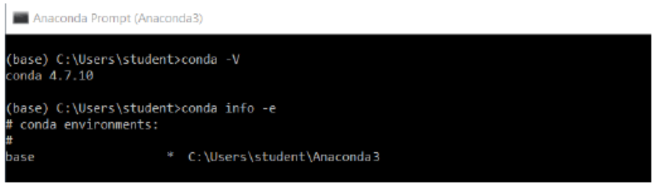
- TensorFlow
  - https://www.tensorflow.org/

Install Anaconda

Windows

1. Download and install Anaconda.
2. Check Installation (Search `Anaconda Prompt` in `Windows Start` and open the `Anaconda Prompt`)

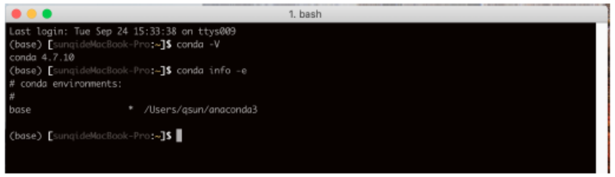

All
Apps
Documents
Settings
Web
More
Feedback
Best match
Anaconda Prompt (Anaconda3)
App
anaconda_navigator
File folder in site-packages
Last modified: 9/24/2019, 4:49 PM
Apps
Anaconda Navigator (Anaconda3)
Anaconda Powershell Prompt (Anaconda3)
Spyder (Anaconda3)
Jupyter Notebook (Anaconda3)
Reset Spyder Settings (Anaconda3)
Search the web
anaconda - See web results
Folders (8+)
Documents (5+)
anaconda Prompt (Anaconda3)
Anaconda Prompt (Anaconda3)
App
Open
Run as administrator
Open file location
Pin to Start
Pin to taskbar
Uninstall


Anaconda Prompt (Anaconda3)

```
(base) C:\Users\student>conda -V
conda 4.7.10

(base) C:\Users\student>conda info -e
# conda environments:
#
base                  *  C:\Users\student\Anaconda3
```

Mac

1. Download and install Anaconda.
2. Test Anaconda



```
Last login: Tue Sep 24 15:33:38 on ttys009
(base) [sunqideMacBook-Pro:~]$ conda -V
conda 4.7.10
(base) [sunqideMacBook-Pro:~]$ conda info -e
# conda environments:
#
base                  *  /Users/qsun/anaconda3

(base) [sunqideMacBook-Pro:~]$
```

Linux

Guide.

Install TensorFlow

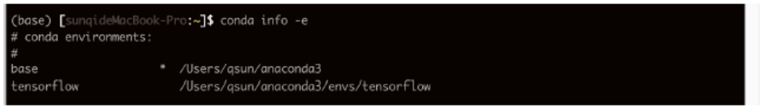
> For Windows, open `Anaconda Prompt`.
>
> For Mac or Linux, open `bash`.

1. Create a new anaconda environment with Python 3.6

   ```
   conda create -n <env_name> pip python=<python_version>
   ```

   ```
   # example:
   conda create -n tensorflow pip python=3.6
   # python version : 3.6
   # environment name : tensorflow (Any other name is ok.)
   ```

   You can then check the new environment:

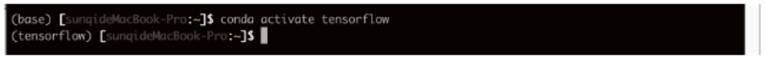
   

   ```
   (base) [sunqideMacBook-Pro:~]$ conda info -e
   # conda environments:
   #
   base                  *  /Users/qsun/anaconda3
   tensorflow               /Users/qsun/anaconda3/envs/tensorflow
   ```

2. Activate the environment

   ```
   conda activate <env_name>
   ```

   ```
   # example:
   conda actiavte tensorflow
   ```

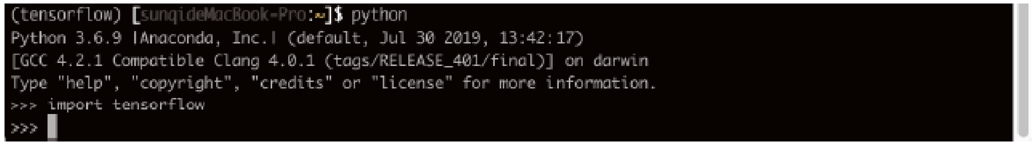
   ```
   (base) [sunqideMacBook-Pro:~]$ conda activate tensorflow
   (tensorflow) [sunqideMacBook-Pro:~]$
   ```

3. Install TensorFlow

   ```
   pip install tensorflow==1.15.0rc1
   ## Tensorflow version : 1.15
   ```

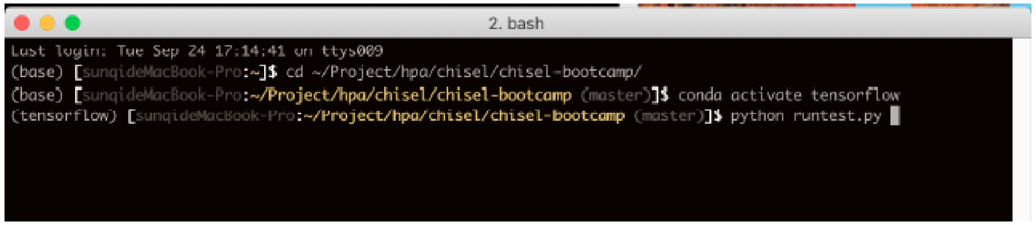
4. Check Installation

   No error occurred !



```
(tensorflow) [sunqideMacBook-Pro:~]$ python
Python 3.6.9 |Anaconda, Inc.| (default, Jul 30 2019, 13:42:17)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow
>>>
```

5. To run your own python code with TensorFlow:



```
Last login: Tue Sep 24 17:14:41 on ttys009
(base) [sunqideMacBook-Pro:~]$ cd ~/Project/hpa/chisel/chisel-bootcamp/
(base) [sunqideMacBook-Pro:~/Project/hpa/chisel/chisel-bootcamp (master)]$ conda activate tensorflow
(tensorflow) [sunqideMacBook-Pro:~/Project/hpa/chisel/chisel-bootcamp (master)]$ python runtest.py
```

- Step into the file location
- activate the tensorflow environment
- run your code