Xin Zhong

Log of Tensorflow

Installation

Linux is often selected as the OS for deep learning. There are various discussions on Google of installing tensorflow(denoted as tf) on different Linux systems. It should be convenient to set up deep learning frameworks on Linux.

Here I record the **installation of tensorflow-gpu on Windows**. For the reasons that although I can run my codes on Linux in the Lab machines, I still have to test some codes on my laptop (with / without external GPU) with a Windows 10. My laptop is for my personal comprehensive usage and I do not want to switch the OS.

I believe setting up tensorflow on Windows would benefit a lot of us. Although most of us may not have access to a powerful GPU, we could test a simple prototype on our desktop or laptop with a normal GPU since CUDA supports most Nvidia Geforce GPUs. At least it is faster than purely running on CPUs.

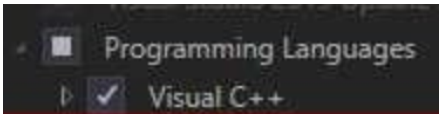
It can be tricky to set up tf on Windows. In a nutshell, on my Windows 10 laptop I set up Python 3.6 (Anaconda 64 bits), Visual Studio 2015 with update, CUDA 8.0 and CuDnn 6.0. My major hint is the https://gist.github.com/mrry/ee5dbcfdd045fa48a27d56664411d41c that checks is a tf is correctly installed on a Windows. The following steps (installing in order) should set up the framework.

1. Python

The official tf for Windows requires 3.5 or 3.6 so far, so choose either version, I use Anaconda https://www.anaconda.com/download/ for an easy collection of the packages.

2. Visual Studio

Up to now (Dec. 30th 2017) tf does not supports CUDA 9, hence we could only use CUDA 8.0. for CUDA 8.0, it only supports Visual Studio up to 2015. So this is the only version we could choose. I downloaded a community version (which is free). When installing Visual Studio 2015, make sure to select “Programming Languages -> Visual C++”, defaultly it is not selected.



3. CUDA

Install CUDA AFTER Visual Studio. As mentioned, we could only use CUDA 8.0 otherwise tensorflow produces errors. In the error message, it directs us to CUDA 9.0, that is not working! Make sure the CUDA is 8.0 by downloading on CUDA developers site. We could unselect the Driver (Display driver) installation if the driver along with CUDA 8.0 is not compatible with our hardware to safely using CUDA 8.0. Check the installation by "nvcc -V" in command prompts.

4. CuDnn

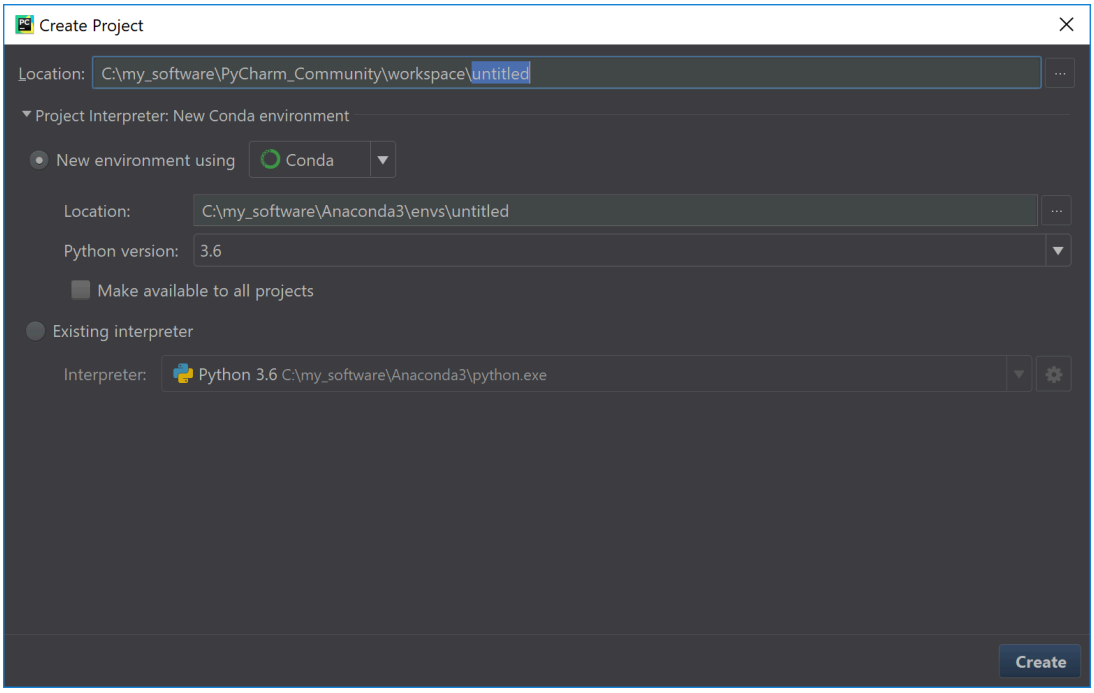
After testing, CuDnn 6.0 for CUDA 8.0 is the only version working for me. CuDNN is a CUDA Deep Neural Network library. Copy CuDnn (when unzipped, three folders: "bin", "include", "lib") into "yourpath\NVIDIA GPU Computing Toolkit\v8.0", by default, my path is "C:\ProgramData\NVIDIA GPU Computing Toolkit\v8.0".

Check the existence of "CUDA_PATH" and "CUDA_PATH_V8_0" in the Windows Environment Variables, also add the path of CuDnn bin folder into the user path variables (My path by default is "C:\ProgramData\NVIDIA GPU Computing Toolkit\v8.0\bin")

5. Install tf on Pycharm using Anaconda

I use Pycharm as my python IDE (https://www.jetbrains.com/pycharm/), so I want to use tf in Pycharm with conda environment. Here are the steps:

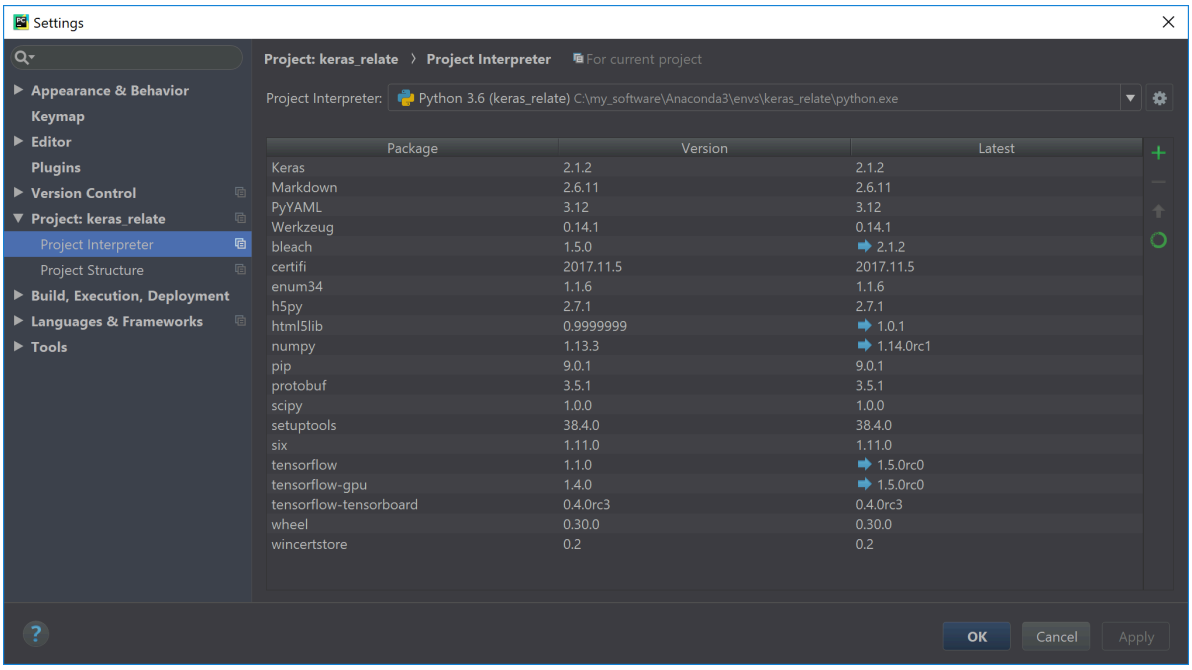
At Pycharm create new conda project.



Open tools -> python console and use pip to install tensorflow-gpu

```
import pip
pip.main(['install','tensorflow-gpu'])
```

Check if tensorflow-gpu is in the project interpreter (File -> settings)



Check is we could import tf and see the local devices.

```
from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())
```

It should output the CPU and the GPU information.

Now we try compute the inner product using tf:

```
import tensorflow as tf
a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3],
name='a')
b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2],
name='b')
c = tf.matmul(a, b)
sess = tf.Session()
print(sess.run(c))
```

Here in the output we should see some information about your GPU, in my case Gefore 940M

"... name: GeForce 940MX major: 5 minor: 0 memoryClockRate(GHz): 1.189

pciBusID: 0000:02:00.0

totalMemory: 2.00GiB freeMemory: 1.66GiB ..."

as well as the result 2 by 2 matrix

“ [[22. 28.]
  [49. 64.]] ”

Ubuntu 16.04 log:

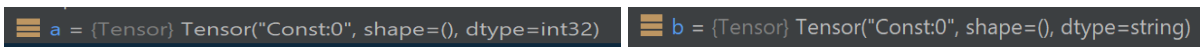
curl -O https://repo.continuum.io/archive/Anaconda3-5.0.1-Linux-x86_64.sh
sha256sum Anaconda3-5.0.1-Linux-x86_64.sh
bash Anaconda3-5.0.1-Linux-x86_64.sh
- Do you approve the license terms? [yes|no]
yes
bash
source ~/.bashrc
conda search "^python$"
conda create --name my_env python=3
source activate my_env
conda install -c anaconda tensorflow-gpu
conda install -c anaconda keras-gpu

Initialization

The core data structure in tf is a tensor. First order tensors (e.g. [1,2,3,4,5]) are vectors (similar as arrays) and the second order tensors are matrices. (e.g. [[1,2], [3,4], [5,6]]).

```
''' constant '''
a = tf.constant(10)
b = tf.constant('Hello, tensorflow')
```

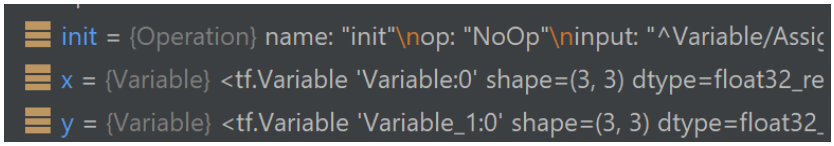
Get




```
''' variable '''
x = tf.Variable(tf.ones([3,3]))
y = tf.Variable(tf.zeros([3,3]))
init = tf.global_variables_initializer()
```

Why “constant” and “**V**ariable” with capitalized V... anyway those are the function names. tf has the ones and zeros function as numpy and Matlab. The initialization is done by function global_variables_initializer()

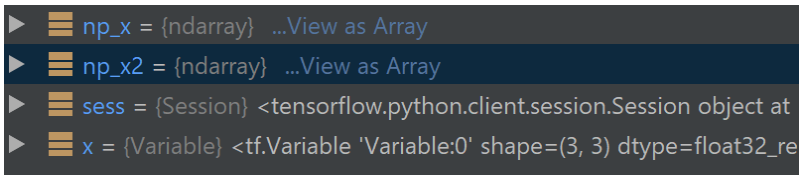
Get



Any tensor returned by Session.run or eval is a NumPy array, so we could simply run .eval() on the transformed tensor to convert it back to numpy

```
''' convert from tensor to np.array  '''
x = tf.Variable(tf.ones([3,3]))
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    np_x = sess.run(x)
    np_x2 = x.eval(tf.Session())
```

Get



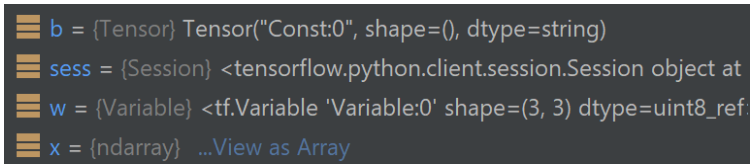

```
''' Initialize with other arrays '''
import numpy as np
x = np.ones([3,3], dtype='uint8')
w = tf.Variable(initial_value=x)
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    print(sess.run(w))
```

We need to run the initializer before running variable. The execution is function sess.run(), we can also write

```
sess = tf.Session()
sess.run(tf.global_variables_initializer())
print(sess.run(w))
```

However, using “with” let the system release a session automatically.

Output:




```
[ [1 1 1]
  [1 1 1]
  [1 1 1] ]
```

```
''' place holder '''
x = tf.placeholder(tf.float32, [None, None])
```

Initialize a place without giving exact values

Get `x = {Tensor} Tensor("Placeholder:0", shape=(?, ?), dtype=float32)`

When using place holders, feed actual values when executing. E.g. use feed_dict function to assign actual values to multiple variables

```
a = tf.placeholder(tf.int16)
b = tf.placeholder(tf.int16)
add = a + b # tf.add(a, b)
mul = a * b # tf.multiply(a, b)
with tf.Session() as sess:
    print('a+b=',sess.run(add, feed_dict={a: 2, b: 3}))
    print('a*b=',sess.run(mul, feed_dict={a: 7, b: 3}))
```

a+b=5

a*b=21

Different from others, for acceleration, the operation, in this case addition, process is:

(1) Create x, y, z three tensors.

(2) Instead of a top-down process, tf put all the operations into a graph where each node is an operation.

(3) Pass the graph to a session.

```
''' graph (tf operations) '''
x, y = tf.Variable(6), tf.Variable(7)
z = x + y
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    print(sess.run(z))
```

output

```
sess = {Session} <tensorflow.python.client.session.Session object at
x = {Variable} <tf.Variable 'Variable:0' shape=() dtype=int32_ref>
y = {Variable} <tf.Variable 'Variable_1:0' shape=() dtype=int32_ref>
z = {Tensor} Tensor("add:0", shape=(), dtype=int32)
```

13

Some operations: (more on https://www.tensorflow.org/api_guides/python/math_ops)

Element-wise: tf.add, tf.subtract, tf.multiply, tf.div (python 2) / tf.truediv (python 3)

Linear Algebra: tf.matmul, tf.matrix_inverse