Deep Learning

Setup Guide for Helix

Last Updated: February 8, 2019

Contributors to this document:

Asa Thibodeau (asa.thibodeau@jax.org)

I. Setting up a python virtual environment 3

II. Installing TensorFlow 4

III. Installing keras & testing python environment 8

# I. Setting up a python virtual environment

When programming or running scripts in python, there will inevitably be the need to install specific libraries to get the latest software to run. This procedure is simple enough when you have full administrative access to install these libraries. For example, if running python on a laptop assigned to you, it can be as easy as opening a terminal and running “pip install <librarynamehere>”. Unfortunately, it’s not always the case that you will have full administrative access to install libraries that you need directly into current python build directory. This is especially true when working with shared resources where allowing users to install or uninstall libraries can cause issues for other users that may be currently working with those libraries. So in order to work around these administrative access issues and to be mindful of other users that may also be using these resources, we can set up a virtual python environment that will install libraries locally to the virtual environment without the need for updating the python build library.

**The following instructions will show you:**

* *How to set up a python virtual environment.*
* *Activating the virtual environment.*
* *Install python libraries to this python virtual environment.*

1. Load the python module

module load python

1. Create the directory you will be installing the new virtual environment

mkdir <yourdirectoryname>

1. Create a new virtual environment for python, specifying the directory you just created

virtualenv <yourdirectorypath>

**Example output of steps 1-3:**

[username]$ mkdir envr

[username]$ virtualenv envr

New python executable in /example/path/envr/bin/python

Installing setuptools, pip, wheel...done.

1. To activate the virtual environment run the activate script in the bin directory using the source command.

source <yourdirectorypath>/bin/activate

**Note:** To leave the virtual environment, simply use the deactivate command

1. With the virtual environment activated, you are now free to install python libraries with pip just as you normally would if you had administrative privileges. Install the following python libraries for deep learning:
   1. numpy
   2. scipy
   3. matplotlib
   4. pyyaml
   5. h5py
   6. graphviz
   7. pydot-ng
   8. opencv-python

pip install numpy scipy matplotlib pyyaml h5py graphviz pydot-ng opencv-python

[username]$ source ./envr/bin/activate

(envr) [username]$ pip install numpy scipy matplotlib pyyaml h5py graphviz pydot-ng opencv-python

**Example of steps 4-5:**

# II. Installing TensorFlow

Installing TensorFlow to the python virtual environment requires a build compatible with the cuda and cudnn library configuration available. In this case, the current version of cuda is 8.0, which was compiled with cudnn 5. In order to run TensorFlow, an installation must be built with this specific configuration. A python installation of TensorFlow 1.6, (built specifically for the cuda/cudnn configuration) is available, with details for building this custom version of TensorFlow described in **a**. Part a is entirely optional and is mostly provided as a rough guide for building TensorFlow from source on helix. The path to the TensorFlow installation .whl file previous built is provided in **b**.

1. Before installing TensorFlow you will need to log into an interactive session with a GPU node and activate the python session. To do this, first log into a GPU node:

qsub –I –q gpu

1. Load the necessary modules. Note: It is important that the python module is loaded BEFORE activating the virtual environment.

module load python

module load cuda/8.0

module load cudnn/8.0/6.0

module load java/1.8.0\_73

module load bazel/0.11.0

module load gcc

1. Activate the virtual environment.

Activate the virtual environment just like in part 1:

source <yourdirectorypath>/bin/activate

1. Build TensorFlow from source (OPTIONAL)

Building TensorFlow from source has proven to be difficult, mainly because of the bazel. In my brief experience working with bazel, many errors arise and there is very little information available on how to correct some of the compiling issues that may arise. The instructions for this step are for compiling TensorFlow 1.6 using version 0.11.0 of bazel. These instructions can be skipped in favor of using the resulting .whl file described in Step 5, but serve as a reference in case a difference version of TensorFlow needs to be configured or the cuda/cudnn libraries are updated.

**Step 4.1)** Download/Unzip TensorFlow 1.6

TensorFlow 1.6 can be obtained from

<https://github.com/tensorflow/tensorflow/releases/tag/v1.6.0>

wget https://github.com/tensorflow/tensorflow/archive/v1.6.0.tar.gz

tar -xvzf v1.6.0.tar.gz

**Step 4.2)** Configure the build

Change to the TensorFlow 1.6 directory and run the configure script

cd tensorflow-1.6.0

./configure

During the configure script you will be asked a series of questions regarding the path of several files and how TensorFlow will be configured. The responses to these are outlined below:

**Q1: Please specify the location of python. [Default is /yourpath/envr/bin/python]:**

*/*path\_to\_your\_python\_virtual\_environment*/bin/python*

**Q2: Please input the desired Python library path to use. Default is [/projects/athib/TestDeepLearningInstall/envr/lib/python2.7/site-packages]:**

*/yourpath/envr/lib/python2.7/site-packages*

**Q3: Do you wish to build TensorFlow with jemalloc as malloc support? [Y/n]:**

*n*

**Q4: Do you wish to build TensorFlow with Google Cloud Platform support? [Y/n]:**

*n*

**Q5: Do you wish to build TensorFlow with Hadoop File System support? [Y/n]:**

*n*

**Q6: Do you wish to build TensorFlow with Amazon S3 File System support? [Y/n]:**

*n*

**Q7: Do you wish to build TensorFlow with Apache Kafka Platform support? [y/N]:**

*n*

**Q8: Do you wish to build TensorFlow with XLA JIT support? [y/N]:**

*n*

**Q9: Do you wish to build TensorFlow with GDR support? [y/N]:**

*n*

**Q10: Do you wish to build TensorFlow with VERBS support? [y/N]:**

*n*

**Q11: Do you wish to build TensorFlow with OpenCL SYCL support? [y/N]:**

*n*

**Q12: Do you wish to build TensorFlow with CUDA support? [y/N]:**

*Y*

**Q13: Please specify the CUDA SDK version you want to use, e.g. 7.0. [Leave empty to default to CUDA 9.0]:**

*8.0*

**Q14: Please specify the location where CUDA 8.0 toolkit is installed. Refer to README.md for more details. [Default is /usr/local/cuda]:**

*/opt/compsci/cuda/8.0*

**Q15: Please specify the cuDNN version you want to use. [Leave empty to default to cuDNN 7.0]:**

*5*

**Q16: Please specify the location where cuDNN 5 library is installed. Refer to README.md for more details. [Default is /opt/compsci/cuda/8.0]:**

*/opt/software/helix/cudnn/8.0*

**Q17: Do you wish to build TensorFlow with TensorRT support? [y/N]:**

*n*

**Q18:**

**Please specify a list of comma-separated Cuda compute capabilities you want to build with.**

**You can find the compute capability of your device at:**

**https://developer.nvidia.com/cuda-gpus.**

**Please note that each additional compute capability significantly increases your build time and binary size. [Default is: 6.0,6.0,6.0,6.0]**

*6.0,6.0,6.0,6.0*

**Q19: Do you want to use clang as CUDA compiler? [y/N]:**

*n*

**Q20: Please specify which gcc should be used by nvcc as the host compiler. [Default is /opt/software/helix/gcc/4.9.2/bin/gcc]:**

*/opt/compsci/gcc/4.9.2/bin/gcc*

**Q21: Do you wish to build TensorFlow with MPI support? [y/N]:**

*n*

**Q22: Please specify optimization flags to use during compilation when bazel option "--config=opt" is specified [Default is -march=native]:**

*-march=native*

**Q23: Would you like to interactively configure ./WORKSPACE for Android builds? [y/N]:**

*n*

**Step 4.3)** Compile TensorFlow with bazel

We are now ready to compile. Before doing so, check over the next step as this will need to be done before the compiler reaches the problem files.

bazel build --config=opt --config=cuda //tensorflow/tools/pip\_package:build\_pip\_package –config=monolithic

**Step 4.4)** Fix protobuf before bazel reaches that step in compilation

You will first need to find the file protobuf.bzl in the .cache folder in your home directory. This file will only be present after running the instruction to build so you will need to use a second terminal to fix this issue.

The following code can be used to identify the file location

cd ~/.cache/bazel/

find . -name '\*protobuf.bzl'

Identify which ‘protobuf.bzl’ is the newest is multiple exist and note the path to this file.

protobuf.bzl will need to be edited to add the following line:

env=ctx.configuration.default\_shell\_env,

to:

if args:

ctx.action(

inputs=inputs,

outputs=ctx.outputs.outs,

arguments=args + import\_flags + [s.path for s in srcs],

executable=ctx.executable.protoc,

mnemonic="ProtoCompile",

env=ctx.configuration.default\_shell\_env,

)

[Credit goes to <https://gist.github.com/taylorpaul/3e4e405ffad1be8355bf742fa24b41f0> ]

Optionally, you can use the edited file I created and copy over the file. The edited file can be found in:

/projects/athib/DeepLearning/protobuf.bzl

**Step 4.5)** Build .whl file

If the compile was successful, run the build pip package tool which should be located in:

**./bazel-bin/tensorflow/tools/pip\_package/**

in the tensorflow directory.

Run **build\_pip\_package** specifying the output directory

Cd <path to tensorflow 1.6 directory>

./bazel-bin/tensorflow/tools/pip\_package/build\_pip\_package <selected destination>

This will create a *tensorflow-version-tags.whl* file in the selected directory.

1. Install TensorFLow

If you’ve gone through the process of building TensorFlow, use the .whl generated in the previous step. Otherwise, you can use the prebuilt .whl used in the example below:

pip install /projects/athib/DeepLearning/tensorflow-1.6.0-cp27-cp27m-linux\_x86\_64.whl

TensorFlow should now be installed for your python virtual environment!

# III. Installing keras & testing python environment

1. **Install keras**
2. Before installing keras you will need to log into an interactive session with a GPU node and activate the python session. To do this, first log into a GPU node:

qsub –I –q gpu

1. Load the necessary modules. Note: It is important that the python module is loaded BEFORE activating the virtual environment.

module load python

module load cuda/8.0

module load cudnn/8.0/6.0

1. Activate the virtual environment.

Activate the virtual environment just like in part 1:

source <yourdirectorypath>/bin/activate

1. Install keras using pip

pip install keras

1. **Run mnist example**
2. Clone the git repository for keras and install:

git clone https://github.com/fchollet/keras

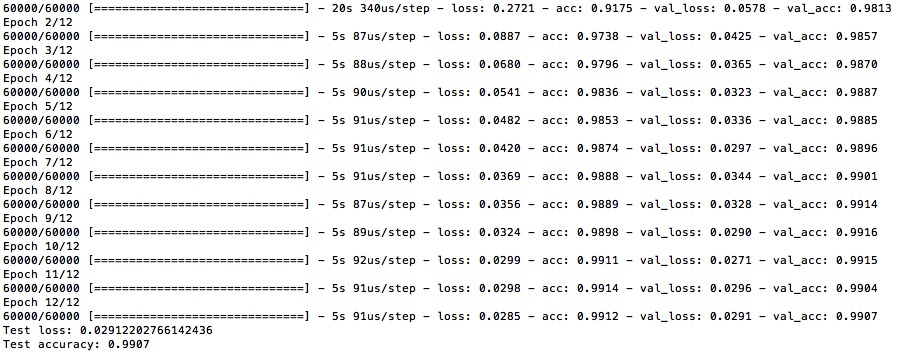
cd keras

python setup.py install

1. Run the mnist example

python examples/mnist\_cnn.py

The output should look something similar to below if everything was configured correctly:



You should now have a python virtual environment to run deep learning programs!