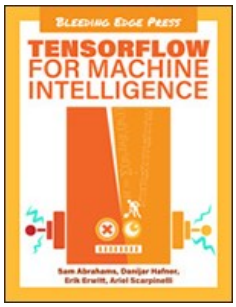


TensorFlow for Machine Intelligence: A Hands-On Introduction to Learning Algorithms

by Sam Abrahams, Danijar Hafner, Erik Erwitt and Ariel Scarpinelli

---

Chapter 2: TensorFlow Installation

Overview

Before you get started using TensorFlow, you'll need to install the software onto your machine. Fortunately, the official TensorFlow website provides a **complete, step-by-step guide to installing TensorFlow** onto Linux and Mac OS X computers. This chapter provides our recommendations for different options available for your installation, as well as information regarding additional third-party software that integrates well with TensorFlow. We also include a reference installation from source to help guide you through installing TensorFlow with GPU support.

If you are already comfortable with using Pip/Conda, virtual environments, or installing from sources, feel free to simply use the official guide here:

https://www.tensorflow.org/versions/master/get_started/ os_setup.html

Selecting an installation environment

Many pieces of software use libraries and packages that are maintained separately. For developers, this is a good thing, as it enables code reuse, and they can focus on creating new functionality instead of recreating code that is already available. However, there is a cost associated with doing this. If a program depends on having another library available in order to work properly, the user or software must ensure that any machine running its code has that library installed. At first glance, this may seem like a trivial problem- simply install the required dependencies along with your software, right? Unfortunately, this approach could have some unintended consequences, and frequently does.

Imagine the following scenario: You find an awesome piece of software, Software A, so you download and install it. As part of its installation script, Software A looks for another piece of software that it depends on and installs it if your computer doesn't have it. We'll call that software Dependency, which is currently on version 1.0. Software A installs Dependency 1.0, finishes its own installation, and all is well. Some time in the future, you stumble upon another program you'd like to have, Software B. Software B uses Dependency 2.0, which is a complete overhaul from Dependency 1.0, and is not backwards compatible. Because of the way Dependency is distributed, there is no way to have both version 1.0 and 2.0 running side-by-side, as this would cause ambiguity when using it (Both are imported as *Dependency*: which version should be used?). Software B overwrites Dependency 1.0 with version 2.0 and completes its install. You find out (the hard way) that Software A is not compatible with Dependency 2.0, and is completely broken. All is not well. How can we run both Software A and Software B on the same machine? It's important for us to know, as TensorFlow depends on several open pieces of software. With Python (the language that packages TensorFlow), there are a couple of ways to get around this dependency clashing, as you'll see next.

1. **Package dependencies inside of codebase:** Instead of relying on a system-level package or library, developers can choose to put the exact version of the library inside of their own code and reference it locally. In this way, all of the software's required code is available and won't be affected by external changes. This is not without its own downsides, however. First, it increases the disk space required to install the software, which means it takes longer to install and becomes more costly to use. Second, the user could have the dependency installed globally anyway, which means that the local version is redundant and eating up space. Finally, it's possible that the dependency puts out a critical, backwards compatible update, which fixes a serious security vulnerability. It now becomes the software developer's responsibility to update the dependency in their codebase, instead of having the user update it from a package manager. Unfortunately, the end-user doesn't have a lot of say with this method, as it's up to the developer to decide when to include dependencies directly. For several of its dependencies, TensorFlow does not include them, so they must be installed separately.

2. **Use dependency environments:** Some package distribution managers have related software that create environments, inside of which specific versions of software can be maintained independently of those contained in other environments. With Python, there are a couple of options. For the standard distributions of Python, Virtualenv is available. If you are using **Anaconda**, it comes with a built-in environment system with its package manager, Conda. We'll cover how to install TensorFlow using both of these below.

3. **Use containers:** Containers, such as **Docker**, are lightweight ways to package software with an entire file system, including its runtime and dependencies. Because of this, any machine (including virtual machines) that can run the container will be able to run the software identically to any other machine running that container. Starting up TensorFlow from a Docker container takes a few more steps than simply activating a Virtualenv or Conda environment, but its consistency across runtime environments can make it invaluable when deploying code across multiple instances (either on virtual machines or physical servers). We'll go over how to install Docker and create your own TensorFlow containers (as well as how to use the official TensorFlow image) below.

In general, we recommend using either Virtualenv or Conda's environments when installing TensorFlow for use on a single computer. They solve the conflicting dependency issue with relatively low overhead, are simple to setup, and require little thought once they are created. If you are preparing TensorFlow code to be deployed on one or more servers, it may be worth creating a Docker container image. While there are a few more steps involved, that cost pays itself back upon deployment across many servers. We do not recommend installing TensorFlow without using either an environment or container.

Jupyter Notebook and Matplotlib

Two excellent pieces of software that are frequently incorporated in data science workflows are the Jupyter Notebook and matplotlib. These have been used in conjunction with NumPy for years, and TensorFlow's tight integration with NumPy allows users to take advantage of their familiar work patterns. Both are open source and use permissive BSD licenses.

The **Jupyter Notebook** (formerly the iPython Notebook) allows you to interactively write documents that include code, text, outputs, LaTeX,

and other visualizations. This makes it incredibly useful for creating reports out of exploratory analysis, since you can show the code used to create your visualizations right next to your charts. You can also include Markdown cells provide richly formatted text to share your insight on your particular approach. Additionally, the Jupyter Notebook is fantastic for prototyping ideas, as you can go back and edit portions of your code and run it directly from the notebook. Unlike many other interactive Python environments that require you to execute code line-by-line, the Jupyter Notebook has you write your code into logical chunks, which can make it easier to debug specific portions of your script. In TensorFlow, this is particularly useful, since a typical TensorFlow program is already split into "graph definition" and "graph running" portions.

Matplotlib is a charting library that allows you to create dynamic, custom visualizations in Python. It integrates seamlessly with NumPy, and its graphs can be displayed directly from the Jupyter Notebook. Matplotlib can also be used to display numeric data as images, which can be used for verifying outputs for image recognition tasks as well as visualizing internal components of neural networks. Additional layers on top of matplotlib, such as **Seaborn**, can be used to augment its capabilities.

Creating a Virtualenv environment

To keep our dependencies nice and clean, we're going to be using **virtualenv** to create a virtual Python environment. First, we need to make sure that Virtualenv is installed along with pip, Python's package manager. Run the following commands (depending on which operating system you are running):

Linux 64-bit

```
# Python 2.7
$ sudo apt-get install python-pip python-dev python-virtualenv
# Python 3
$ sudo apt-get install python3-pip python3-dev python3-virtualenv
```

Mac OS X

```
$ sudo easy_install pip
$ sudo pip install --upgrade virtualenv
```

Now that we're ready to roll, let's create a directory to contain this environment, as well as any future environments you might create in the future:

```
$ sudo mkdir ~/env
```

Next, we'll create the environment using the `virtualenv` command. In this example, it will be located in `~/env/tensorflow`.

```
$ virtualenv --system-site-packages ~/env/tensorflow
```

Once it has been created, we can activate the environment using the `source` command.

```
$ source ~/env/tensorflow/bin/activate
# Notice that your prompt now has a '(tensorflow)' indicator
(tensorflow)$
```

We'll want to make sure that the environment is active when we install anything with `pip`, as that is how Virtualenv keeps track of various dependencies.

When you're done with the environment, you can shut it off just by using the `deactivate` command:

```
(tensorflow)$ deactivate
```

Since you'll be using the virtual environment frequently, it will be useful to create a shortcut for activating it instead of having to write out the entire `source ...` command each time. This next command adds a bash alias to your ~/.bashrc file, which will let you simply type `tensorflow` whenever you want to start up the environment:

```
$ sudo printf '\nalias tensorflow="source ~/env/tensorflow/bin/activate"' >>
~/.bashrc
```

To test it out, restart your bash terminal and type `tensorflow`:

```
$ tensorflow
# The prompt should change, as before
(tensorflow)$
```

Simple installation of TensorFlow

If you just want to get on to the tutorials as quickly as possible and don't care about GPU support, you can install one of TensorFlow's official pre-built binaries. Simply make sure that your Virtualenv environment from the previous section is active and run the following command corresponding to your operating system and version of Python:

Linux 64-bit installation

```
# Linux, Python 2.7
(tensorflow)$ pip install --upgrade https://storage.googleapis.com/tensor-
flow/linux/cpu/tensorflow-0.9.0-cp27-none-linux_x86_64.whl
```

```
# Linux 64-bit, Python 3.4
(tensorflow)$ pip3 install --upgrade https://storage.googleapis.com/tensor-
flow/linux/cpu/tensorflow-0.9.0-cp34-cp34m-linux_x86_64.whl

# Linux 64-bit, Python 3.5
(tensorflow)$ pip3 install --upgrade https://storage.googleapis.com/tensor-
flow/linux/cpu/tensorflow-0.9.0-cp35-cp35m-linux_x86_64.whl
```

Mac OS X installation

```
# Mac OS X, Python 2.7:
(tensorflow)$ pip install --upgrade https://storage.googleapis.com/tensor-
flow/mac/tensorflow-0.9.0-py2-none-any.whl

# Mac OS X, Python 3.4+
(tensorflow)$ pip3 install --upgrade https://storage.googleapis.com/tensor-
flow/mac/tensorflow-0.9.0-py3-none-any.whl
```

> Technically, there are pre-built binaries for TensorFlow with GPU support, but they require specific versions of NVIDIA software and are incompatible with future versions.

Example installation from source: 64-bit Ubuntu Linux with GPU support

If you want to use TensorFlow with GPU(s) support, you will most likely have to build from source. We've included a reference installation example that goes step-by-step through all of the things you'll need to do to get TensorFlow up and running. Note that this example is for an Ubuntu Linux 64-bit distribution, so you may have to change certain commands (such as `apt-get`). If you'd like to build from source on Mac OS X, we recommend the official guide on the TensorFlow website:

https://www.tensorflow.org/versions/master/get_started/ os_setup.html#installation-for-mac-os-x

Installing dependencies

> This assumes you've already installed `python-pip`, `python-dev`, and `python-virtualenv` from the previous section on installing Virtualenv.

Building TensorFlow requires a few more dependencies, though! Run the following commands, depending on your version of Python:

Python 2.7

```
$ sudo apt-get install python-numpy python-wheel python-imaging swig
```

Python 3

```
$ sudo apt-get install python3-numpy python3-wheel python3-imaging swig
```

Installing Bazel

Bazel is an open source build tool based on Google's internal software, Blaze. As of writing, TensorFlow requires Bazel in order to build from source, so we must install it ourselves. The Bazel website has **complete installation instructions**, but we include the basic steps here.

The first thing to do is ensure that Java Development Kit 8 is installed on your system. The following commands will add the Oracle JDK 8 repository as a download location for `apt` and then install it:

```
$ sudo apt-get install software-properties-common
$ sudo add-apt-repository ppa:webupd8team/java
$ sudo apt-get update
$ sudo apt-get install oracle-java8-installer
```

> Ubuntu versions 15.10 and later can install OpenJDK 8 instead of the Oracle JDK. This is easier and recommended- use the following commands instead of the above to install OpenJDK on your system:
>
> ```
> # Ubuntu 15.10
> $ sudo apt-get install openjdk-8-jdk
> # Ubuntu 16.04
> $ sudo apt-get install default-jdk
> ```

Before moving on, verify that Java is installed correctly:

```
$ java -version
# Should see similar output as below
java version "1.8.0_91"
Java(TM) SE Runtime Environment (build 1.8.0_91-b14)
```

```
Java HotSpot(TM) 64-Bit Server VM (build 25.91-b14, mixed mode)
```

Once Java is setup, there are a few more dependencies to install:

```
$ sudo apt-get install pkg-config zip g++ zlib1g-dev unzip
```

Next, you'll need to download the Bazel installation script. To do so, you can either go to the **Bazel releases page** on GitHub, or you can use the following `wget` command. Note that for Ubuntu, you'll want to download "bazel—installer-linux-x86_64.sh":

```
# Downloads Bazel 0.3.0
$ wget https://github.com/bazelbuild/bazel/releases/download/0.3.0/
bazel-0.3.0-installer-linux-x86_64.sh
```

Finally, we'll make the script executable and run it:

```
$ chmod +x bazel-<version>-installer-linux-x86_64.sh
$ ./bazel-<version>-installer-linux-x86_64.sh --user
```

By using the `--user flag`, Bazel is installed to the /bin directory for the user. To ensure that this is added to your PATH, run the following to update your /.bashrc:

```
$ sudo printf '\nexport PATH="$PATH:$HOME/bin"' >> ~/.bashrc
```

Restart your bash terminal and run `bazel` to make sure everything is working properly:

```
$ bazel version
# You should see some output like the following
Build label: 0.3.0
Build target: ...
...
```

Great! Next up, we need to get the proper dependencies for GPU support.

Installing CUDA Software (NVIDIA CUDA GPUs only)

If you have an NVIDIA GPU that supports **CUDA**, you can install TensorFlow with GPU support. There is a list of CUDA-enabled video cards available here:

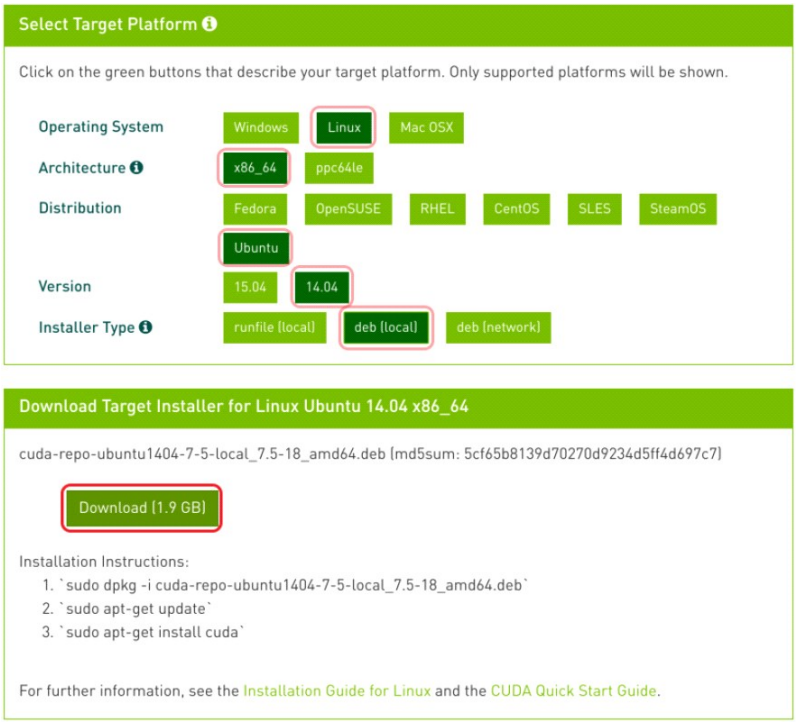
https://developer.nvidia.com/cuda-gpus

In addition to making sure that your GPU is on the list, make a note of the "Compute Capability" number associated with your card. For example, the GeForce GTX 1080 has a compute capability of 6.1, and the GeForce GTX TITAN X has a compute capability of 5.2. You'll need this number for when you compile TensorFlow. Once you've determined that you're able to take advantage of CUDA, the first thing you'll want to do is sign up for NVIDIA's "Accelerated Computer Developer Program". This is required to download all of the files necessary to install CUDA and cuDNN. The link to do so is here:

https://developer.nvidia.com/accelerated-computing-developer

Once you're signed up, you'll want to download CUDA. Go to the following link and use the following instructions:

https://developer.nvidia.com/cuda-downloads



1. Under "Select Target Platform", choose the following options:

   Linux

   x86_64

   Ubuntu

   14.04/15.04 (whichever version you are using)

   deb (local)

2. Click the "Download" button and save it somewhere on your computer. This file is large, so it will take a while.
3. Navigate to the directory containing the downloaded file and run the following commands:

```
$ sudo dpkg -i cuda-repo-ubuntu1404-7-5-local_7.5-18_amd64.deb
$ sudo apt-get update
$ sudo apt-get install cuda
```

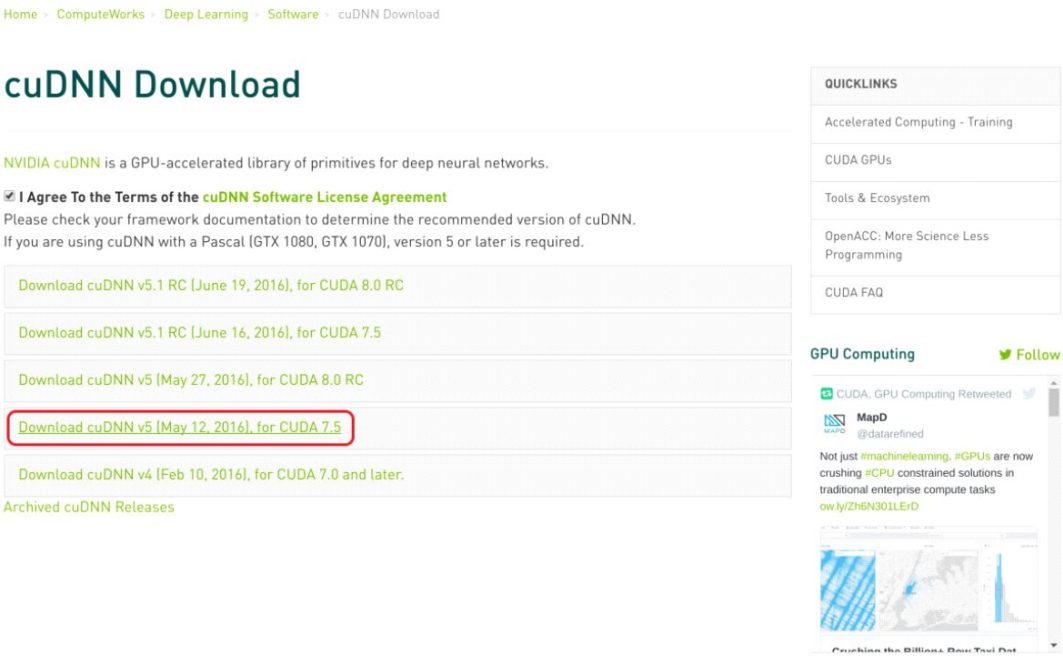
This will install CUDA into the `/usr/local/cuda` directory.

Next, we need to download cuDNN, which is a separate add-on to CUDA designed for deep neural networks. Click the "Download" button on the following page:

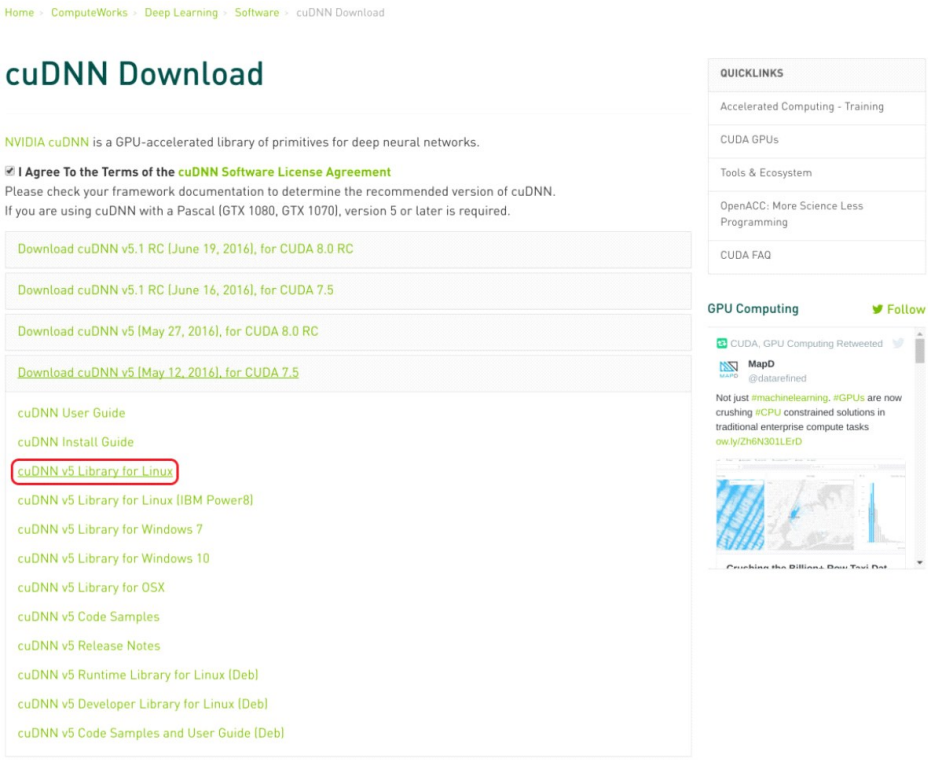
https://developer.nvidia.com/cudnn

After signing in with the account you created above, you'll be taken to a brief survey. Fill that out and you'll be taken to the download page. Click "I Agree to the Terms…" to be presented with the different download options. Because we installed CUDA 7.5 above, we'll want to download cuDNN for CUDA 7.5 (as of writing, we are using cuDNN version 5.0).

Click "Download cuDNN v5 for CUDA 7.5" to expand a bunch of download options:



Click "cuDNN v5 Library for Linux" to download the zipped cuDNN files:



Navigate to where the .tgz file was downloaded and run the following commands to place the correct files inside the `/usr/local/cuda` directory:

```
$ tar xvzf cudnn-7.5-linux-x64-v5.0-ga.tgz
$ sudo cp cuda/include/cudnn.h /usr/local/cuda/include
$ sudo cp cuda/lib64/libcudnn* /usr/local/cuda/lib64
$ sudo chmod a+r /usr/local/cuda/include/cudnn.h /usr/local/cuda/lib64/
libcudnn*
```

And that's it for installing CUDA! With all of the dependencies taken care of, we can now move on to the installation of TensorFlow itself.

Building and Installing TensorFlow from Source

First things first, clone the Tensorflow repository from GitHub and enter the directory:

```
$ git clone --recurse-submodules https://github.com/tensorflow/tensorflow
$ cd tensorflow
```

Once inside, we need to run the `./configure` script, which will tell Bazel which compiler to use, which version of CUDA to use, etc. Make sure that you have the "compute capability" number (as mentioned previously) for your GPU card available:

```
$ ./configure
Please specify the location of python. [Default is /usr/bin/
python]: /usr/bin/python
# NOTE: For Python 3, specify /usr/bin/python3 instead
Do you wish to build TensorFlow with Google Cloud Platform support? [y/N] N
Do you wish to build TensorFlow with GPU support? [y/N] y

Please specify which gcc nvcc should use as the host compiler. [Default
is /usr/bin/gcc]: /usr/bin/gcc

Please specify the Cuda SDK version you want to use, e.g. 7.0. [Leave empty
to use system default]: 7.5

Please specify the Cudnn version you want to use. [Leave empty to use system
default]: 5.0.5

Please specify the location where cuDNN 5.0.5 library is installed. Refer to
README.md for more details. [Default is /usr/local/cuda]: /usr/local/cuda

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://develop-
er.nvidia.com/cuda-gpus.
Please note that each additional compute capability significantly increases
your build time and binary size.
[Default is: "3.5,5.2"]: <YOUR-COMPUTE-CAPABILITY-NUMBER-HERE>

Setting up Cuda include
Setting up Cuda lib64
Setting up Cuda bin
Setting up Cuda nvvm
Setting up CUPTI include
Setting up CUPTI lib64
Configuration finished
```

> Google Cloud Platform support is currently in a closed alpha. If you have access to the program, feel free to answer yes to the Google Cloud Platform support question.

With the configuration finished, we can use Bazel to create an executable that will create our Python binaries:

```
$ bazel build -c opt --config=cuda //tensorflow/tools/pip_pack-
age:build_pip_package
```

This will take a fair amount of time, depending on how powerful your computer is. Once Bazel is done, run the output executable and pass in a location to save the Python wheel:

```
$ bazel-bin/tensorflow/tools/pip_package/build_pip_package ~/tensorflow/bin
```

This creates a Python .whl file inside of `~/tensorflow/bin/`. Make sure that your "tensorflow" Virtualentv is active, and install the wheel with pip! (Note that the exact name of the binary will differ depending on which version of TensorFlow is installed, which operating system you're using, and which version of Python you installed with):

```
$ tensorflow
(tensorflow)$ sudo pip install ~/tensorflow/bin/tensorflow-0.9.0-py2-none-
any.whl
```

If you have multiple machines with similar hardware, you can use this wheel to quickly install TensorFlow on all of them.

You should be good to go! We'll finish up by installing the Jupyter Notebook and matplotlib.

Installing Jupyter Notebook:

First, run the following commands to install **iPython**, an incredibly useful interactive Python kernel, and the backbone of the Jupyter Notebook. We highly recommend installing both the Python 2 and Python 3 kernels below, as it will give you more options moving forward (i.e., run all of the following!):

```
# Python 2.7
$ sudo python2 -m pip install ipykernel
$ sudo python2 -m ipykernel install
# Python 3
$ sudo python3 -m pip install jupyterhub notebook ipykernel
$ sudo python3 -m ipykernel install
```

After that, two simple commands should get you going. First install the `build-essential` dependency:

```
$ sudo apt-get install build-essential
```

Then use `pip` to install the Jupyter Notebook (`pip3` if you are using Python 3):

```
# For Python 2.7
$ sudo pip install jupyter
# For Python 3
$ sudo pip3 install jupyter
```

Official installation instructions are available at the Jupyter website:

http://jupyter.readthedocs.io/en/latest/install.html

Installing matplotlib

Installing matplotlib on Linux/Ubuntu is easy. Just run the following:

```
# Python 2.7
$ sudo apt-get build-dep python-matplotlib python-tk
# Python 3
$ sudo apt-get build-dep python3-matplotlib python3-tk
```

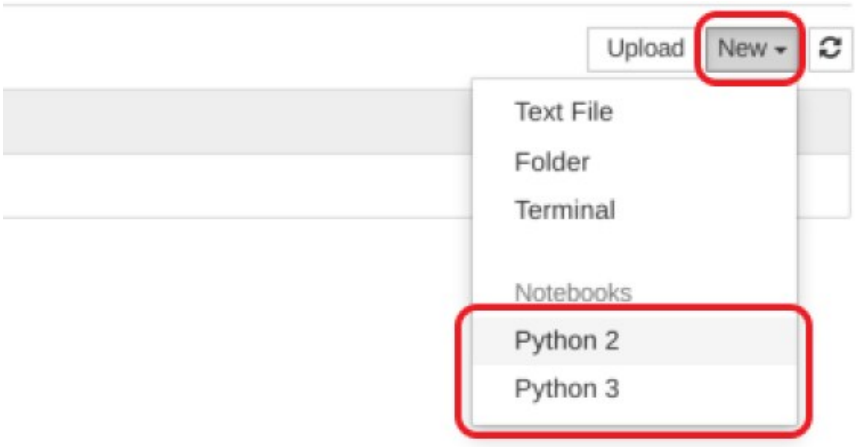
And that's it!

Testing Out TensorFlow, Jupyter Notebook, and matplotlib

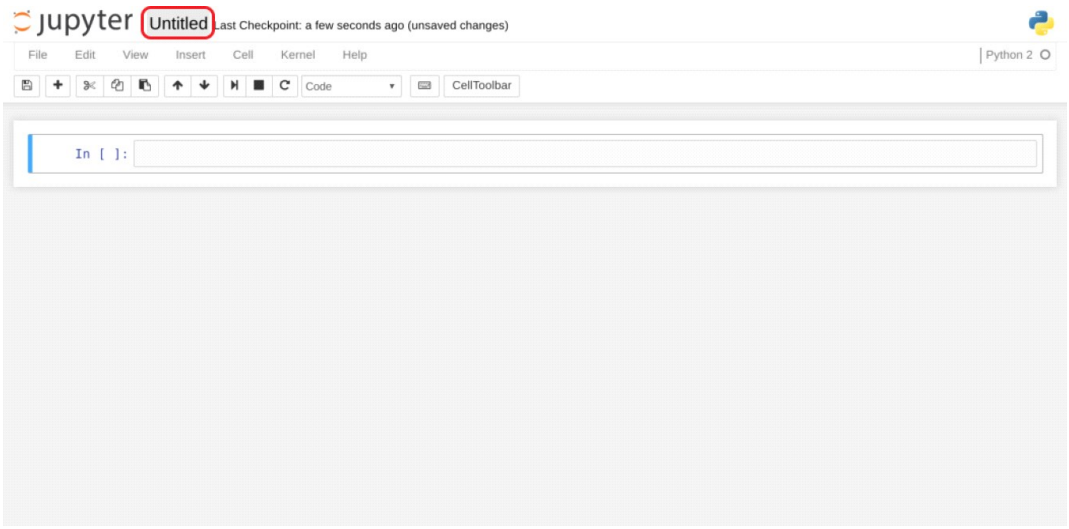
Let's run some dummy code to double check that things are working properly. Create a new directory called "tf-notebooks" to play around in. Enter that directory and run `jupyter notebook`. Again, make sure that the "tensorflow" environment is active:

```
(tensorflow)$ mkdir tf-notebooks
(tensorflow)$ cd tf-notebooks
(tensorflow)$ jupyter notebook
```

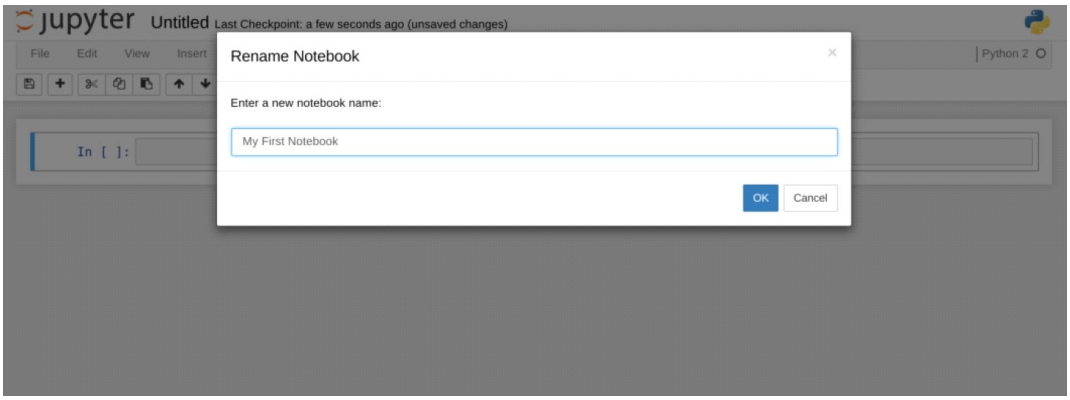
This will start up a Jupyter Notebook server and open the software up in your default web browser. Assuming you don't have any files in your `tf-notebooks` directory, you'll see an empty workspace with the message "Notebook list is empty". To create a new notebook, click the "New" button in the upper right corner of the page, and then select either "Python 2" or "Python 3", depending on which version of Python you've used to install TensorFlow.



Your new notebook will open up automatically, and you'll be presented with a blank slate to work with. Let's quickly give the notebook a new name. At the top of the screen, click the word "Untitled":

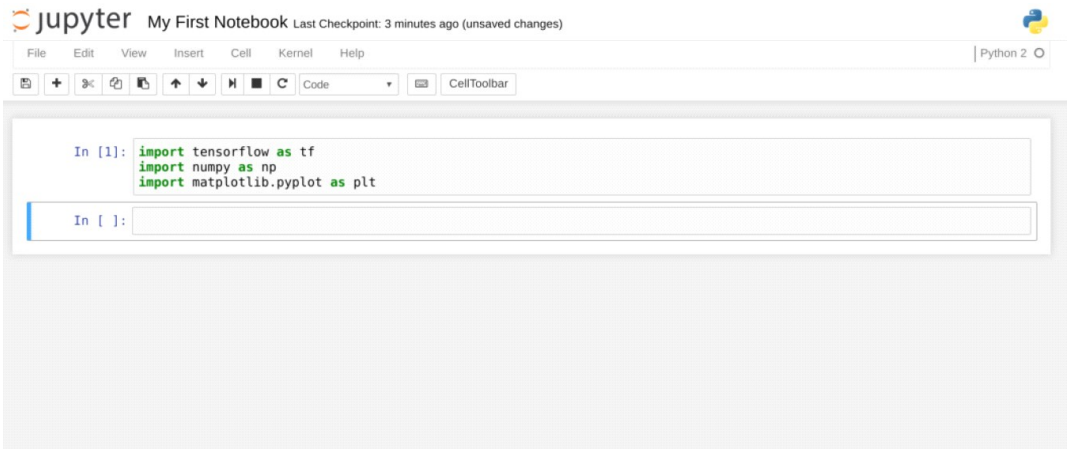


This will pop up a window that allows you to rename the notebook. This also changes the name of the notebook file (with the extension .ipynb). You can call this whatever you'd like- in this example we're calling it "My First Notebook"



Now, let's look at the actual interface. You'll notice an empty cell with the block `In [ ]:` next to it. You can type Python code directly into this cell, and it can include multiple lines. Let's import TensorFlow, NumPy, and the pyplot module of matplotlib into notebook:

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
```

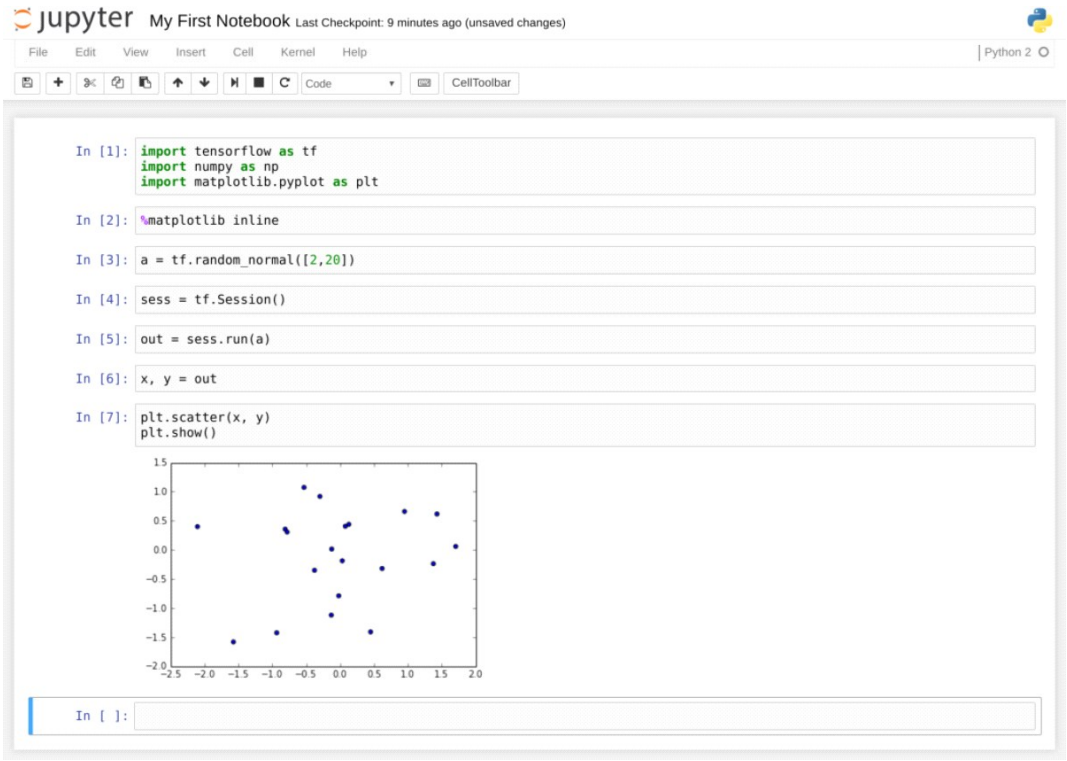


In order to run the cell, simply type *shift-enter*, which will run your code and create a new cell below. You'll notice the indicator to the left now reads `In [1]:`, which means that this cell was the first block of code to run in the kernel. Fill in the notebook with the following code, using as many or as few cells as you find appropriate. You can use the breaks in the cells to naturally group related code together.

```
%matplotlib inline
a = tf.random_normal([2,20])
```

```
sess = tf.Session()
out = sess.run(a)
x, y = out

plt.scatter(x, y)
plt.show()
```



This line is special, and worth mentioning in particular:

```
%matplotlib inline
```

It is a special command that tells the notebook to display matplotlib charts directly inside the browser.

Let's go over what the rest of the code does, line-by-line. Don't worry if you don't understand some of the terminology, as we'll be covering it in the book:

1. Use TensorFlow to define a 2x20 matrix of random numbers and assign it to the variable a
2. Start a TensorFlow Session and assign it to `sess`
3. Execute a with the `sess.run()` method, and assign the output (which is a NumPy array) to `out`
4. Split up the 2x20 matrix into two 1x10 vectors, x and y
5. Use pyplot to create a scatter plot with x and y

Assuming everything is installed correctly, you should get an output similar to the above! It's a small first step, but hopefully it feels good to get the ball rolling.

For a more thorough tutorial on the ins-and-outs of the Jupyter Notebook, check out the examples page here:

http://jupyter-notebook.readthedocs.io/en/latest/examples/ Notebook/examples_index.html

Conclusion

Voila! You should have a working version of TensorFlow ready to go. In the next chapter, you'll learn fundamental TensorFlow concepts and build your first models in the library. If you had any issues installing TensorFlow on your system, the official installation guide should be the first place to look:

https://www.tensorflow.org/versions/master/get_started/ os_setup.html