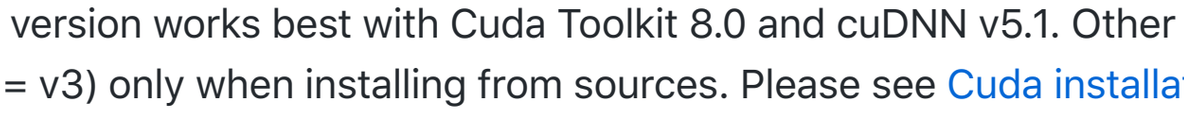
yifeif Clarify doc for tensorflow#7077 (#7103) bc72653 on Jan 27

20 contributors



1277 lines (916 sloc) | 45.1 KB

Download and Setup

You can install TensorFlow either from our provided binary packages or from the github source.

Requirements

The TensorFlow Python API supports Python 2.7 and Python 3.3+.

The GPU version works best with Cuda Toolkit 8.0 and cuDNN v5.1. Other versions are supported (Cuda toolkit >= 7.0 and cuDNN >= v3) only when installing from sources. Please see Cuda installation for details. For Mac OS X, please see Setup GPU for Mac.

Overview

We support different ways to install TensorFlow:

- Pip install: Install TensorFlow on your machine, possibly upgrading previously installed Python packages. May impact existing Python programs on your machine.
- Virtualenv install: Install TensorFlow in its own directory, not impacting any existing Python programs on your machine.
- Anaconda install: Install TensorFlow in its own environment for those running the Anaconda Python distribution. Does not impact existing Python programs on your machine.
- Docker install: Run TensorFlow in a Docker container isolated from all other programs on your machine.
- Installing from sources: Install TensorFlow by building a pip wheel that you then install using pip.

If you are familiar with Pip, Virtualenv, Anaconda, or Docker, please feel free to adapt the instructions to your particular needs. The names of the pip and Docker images are listed in the corresponding installation sections.

If you encounter installation errors, see common problems for some solutions.

Pip installation

Pip is a package management system used to install and manage software packages written in Python. We provide pip packages for TensorFlow on Linux, Mac OS X, and Windows. For Windows instructions, please see Pip installation on Windows.

The packages that will be installed or upgraded during the pip install are listed in the REQUIRED_PACKAGES section of setup.py.

Install pip (or pip3 for python3) if it is not already installed:

```
# Ubuntu/Linux 64-bit
$ sudo apt-get install python-pip python-dev

# Mac OS X
$ sudo easy_install pip
$ sudo easy_install --upgrade six
```

We have also uploaded the binaries to Pypi, so you can simply install tensorflow on Linux, Mac or Windows with pip install. Note you will need pip version 8.1 or later for the following commands to work on Linux :

```
$ pip install tensorflow
```

For installing the version with GPU support, please use:

```
$ pip install tensorflow-gpu
```

If the above commands do not work on your system, you can follow these instructions:

```
# Ubuntu/Linux 64-bit, CPU only, Python 2.7
$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/linux/cpu/tensorflow-0.12.1-cp27-none-l

# Ubuntu/Linux 64-bit, GPU enabled, Python 2.7
# Requires CUDA toolkit 8.0 and CuDNN v5.1. For other versions, see "Installing from sources" below.
$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/linux/gpu/tensorflow_gpu-0.12.1-cp27-no

# Mac OS X, CPU only, Python 2.7:
$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/mac/cpu/tensorflow-0.12.1-py2-none-any.

# Mac OS X, GPU enabled, Python 2.7:
$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/mac/gpu/tensorflow_gpu-0.12.1-py2-none-

# Ubuntu/Linux 64-bit, CPU only, Python 3.4
$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/linux/cpu/tensorflow-0.12.1-cp34-cp34m-

# Ubuntu/Linux 64-bit, GPU enabled, Python 3.4
# Requires CUDA toolkit 8.0 and CuDNN v5.1. For other versions, see "Installing from sources" below.
$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/linux/gpu/tensorflow_gpu-0.12.1-cp34-cp

# Ubuntu/Linux 64-bit, CPU only, Python 3.5
$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/linux/cpu/tensorflow-0.12.1-cp35-cp35m-

# Ubuntu/Linux 64-bit, GPU enabled, Python 3.5
# Requires CUDA toolkit 8.0 and CuDNN v5.1. For other versions, see "Installing from sources" below.
$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/linux/gpu/tensorflow_gpu-0.12.1-cp35-cp

# Mac OS X, CPU only, Python 3.4 or 3.5:
$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/mac/cpu/tensorflow-0.12.1-py3-none-any.

# Mac OS X, GPU enabled, Python 3.4 or 3.5:
$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/mac/gpu/tensorflow_gpu-0.12.1-py3-none-
```

Install TensorFlow:

```
# Python 2
$ sudo pip install --upgrade $TF_BINARY_URL

# Python 3
$ sudo pip3 install --upgrade $TF_BINARY_URL
```

NOTE: If you are upgrading from a previous installation of TensorFlow < 0.7.1, you should uninstall the previous TensorFlow *and protobuf* using `pip uninstall` first to make sure you get a clean installation of the updated protobuf dependency.

You can now test your installation.

Pip installation on Windows

TensorFlow supports only 64-bit Python 3.5 on Windows. We have tested the pip packages with the following distributions of Python:

- Python 3.5 from Anaconda

- Python 3.5 from python.org.

  NOTE: TensorFlow requires `MSVCP140.DLL`, which may not be installed on your system. If, when you `import tensorflow as tf`, you see an error about `No module named "_pywrap_tensorflow"` and/or `DLL load failed`, check whether `MSVCP140.DLL` is in your `%PATH%` and, if not, you should install the Visual C++ 2015 redistributable (x64 version).

Both distributions include pip. To install the CPU-only version of TensorFlow, enter the following command at a command prompt:

```
C:\> pip install --upgrade https://storage.googleapis.com/tensorflow/windows/cpu/tensorflow-0.12.1-cp35-
```

To install the GPU version of TensorFlow, enter the following command at a command prompt:

```
C:\> pip install --upgrade https://storage.googleapis.com/tensorflow/windows/gpu/tensorflow_gpu-0.12.1-c
```

You can now test your installation.

You can also use Virtualenv or Anaconda environments to manage your installation of TensorFlow on Windows.

Virtualenv installation

Virtualenv is a tool to keep the dependencies required by different Python projects in separate places. The Virtualenv installation of TensorFlow will not override pre-existing version of the Python packages needed by TensorFlow.

With Virtualenv the installation is as follows:

- Install pip and Virtualenv.
- Create a Virtualenv environment.
- Activate the Virtualenv environment and install TensorFlow in it.
- After the install you will activate the Virtualenv environment each time you want to use TensorFlow.

Install pip and Virtualenv:

```
# Ubuntu/Linux 64-bit
$ sudo apt-get install python-pip python-dev python-virtualenv

# Mac OS X
$ sudo easy_install pip
$ sudo pip install --upgrade virtualenv
```

Create a Virtualenv environment in the directory `~/tensorflow`:

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

Activate the environment:

```
$ source ~/tensorflow/bin/activate  # If using bash
$ source ~/tensorflow/bin/activate.csh  # If using csh
(tensorflow)$  # Your prompt should change
```

Now, install TensorFlow just as you would for a regular Pip installation. First select the correct binary to install:

```
# Ubuntu/Linux 64-bit, CPU only, Python 2.7
(tensorflow)$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/linux/cpu/tensorflow-0.12.1

# Ubuntu/Linux 64-bit, GPU enabled, Python 2.7
# Requires CUDA toolkit 8.0 and CuDNN v5.1. For other versions, see "Installing from sources" below.
(tensorflow)$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/linux/gpu/tensorflow_gpu-0.
```

```
# Mac OS X, CPU only, Python 2.7:
(tensorflow)$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/mac/cpu/tensorflow-0.12.1-p

# Mac OS X, GPU enabled, Python 2.7:
(tensorflow)$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/mac/gpu/tensorflow_gpu-0.12

# Ubuntu/Linux 64-bit, CPU only, Python 3.4
(tensorflow)$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/linux/cpu/tensorflow-0.12.1

# Ubuntu/Linux 64-bit, GPU enabled, Python 3.4
# Requires CUDA toolkit 8.0 and CuDNN v5.1. For other versions, see "Installing from sources" below.
(tensorflow)$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/linux/gpu/tensorflow_gpu-0.

# Ubuntu/Linux 64-bit, CPU only, Python 3.5
(tensorflow)$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/linux/cpu/tensorflow-0.12.1

# Ubuntu/Linux 64-bit, GPU enabled, Python 3.5
# Requires CUDA toolkit 8.0 and CuDNN v5.1. For other versions, see "Installing from sources" below.
(tensorflow)$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/linux/gpu/tensorflow_gpu-0.

# Mac OS X, CPU only, Python 3.4 or 3.5:
(tensorflow)$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/mac/cpu/tensorflow-0.12.1-p

# Mac OS X, GPU enabled, Python 3.4 or 3.5:
(tensorflow)$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/mac/gpu/tensorflow_gpu-0.12
```

Finally install TensorFlow:

```
# Python 2
(tensorflow)$ pip install --upgrade $TF_BINARY_URL

# Python 3
(tensorflow)$ pip3 install --upgrade $TF_BINARY_URL
```

With the Virtualenv environment activated, you can now test your installation.

When you are done using TensorFlow, deactivate the environment.

```
(tensorflow)$ deactivate

$  # Your prompt should change back
```

To use TensorFlow later you will have to activate the Virtualenv environment again:

```
$ source ~/tensorflow/bin/activate  # If using bash.
$ source ~/tensorflow/bin/activate.csh  # If using csh.
(tensorflow)$  # Your prompt should change.
# Run Python programs that use TensorFlow.
...
# When you are done using TensorFlow, deactivate the environment.
(tensorflow)$ deactivate
```

Anaconda installation

Anaconda is a Python distribution that includes a large number of standard numeric and scientific computing packages. Anaconda uses a package manager called "conda" that has its own environment system similar to Virtualenv.

As with Virtualenv, conda environments keep the dependencies required by different Python projects in separate places. The Anaconda environment installation of TensorFlow will not override pre-existing version of the Python packages needed by TensorFlow.

- Install Anaconda.
- Create a conda environment.

- Activate the conda environment and install TensorFlow in it.
- After the install you will activate the conda environment each time you want to use TensorFlow.
- Optionally install ipython and other packages into the conda environment.

Install Anaconda:

Follow the instructions on the Anaconda download site.

Note: If tensorflow has been installed via pip outside the Anaconda environment previously, then one should uninstall it if one wants to use the tensorflow installed within an Anaconda environment, because Anaconda searches system site-packages from `.local` with higher priority.

```
# Python 2
$ pip uninstall tensorflow

# Python 3
$ pip3 uninstall tensorflow
```

Create a conda environment called `tensorflow`:

```
# Python 2.7
$ conda create -n tensorflow python=2.7

# Python 3.4
$ conda create -n tensorflow python=3.4

# Python 3.5
$ conda create -n tensorflow python=3.5
```

Activate the environment and use conda or pip to install TensorFlow inside it.

Using conda

A community maintained conda package is available from conda-forge.

Only the CPU version of TensorFlow is available at the moment and can be installed in the conda environment for Python 2 or Python 3.

```
$ source activate tensorflow
(tensorflow)$  # Your prompt should change

# Linux/Mac OS X, Python 2.7/3.4/3.5, CPU only:
(tensorflow)$ conda install -c conda-forge tensorflow
```

Using pip

If using pip make sure to use the `--ignore-installed` flag to prevent errors about `easy_install`.

```
$ source activate tensorflow
(tensorflow)$  # Your prompt should change
```

Now, install TensorFlow just as you would for a regular Pip installation. First select the correct binary to install:

```
# Ubuntu/Linux 64-bit, CPU only, Python 2.7
(tensorflow)$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/linux/cpu/tensorflow-0.12.1

# Ubuntu/Linux 64-bit, GPU enabled, Python 2.7
# Requires CUDA toolkit 8.0 and CuDNN v5.1. For other versions, see "Installing from sources" below.
(tensorflow)$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/linux/gpu/tensorflow_gpu-0.

# Mac OS X, CPU only, Python 2.7:
(tensorflow)$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/mac/cpu/tensorflow-0.12.1-p
```

```
# Mac OS X, GPU enabled, Python 2.7:
(tensorflow)$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/mac/gpu/tensorflow_gpu-0.12

# Ubuntu/Linux 64-bit, CPU only, Python 3.4
(tensorflow)$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/linux/cpu/tensorflow-0.12.1

# Ubuntu/Linux 64-bit, GPU enabled, Python 3.4
# Requires CUDA toolkit 8.0 and CuDNN v5.1. For other versions, see "Installing from sources" below.
(tensorflow)$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/linux/gpu/tensorflow_gpu-0.

# Ubuntu/Linux 64-bit, CPU only, Python 3.5
(tensorflow)$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/linux/cpu/tensorflow-0.12.1

# Ubuntu/Linux 64-bit, GPU enabled, Python 3.5
# Requires CUDA toolkit 8.0 and CuDNN v5.1. For other versions, see "Installing from sources" below.
(tensorflow)$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/linux/gpu/tensorflow_gpu-0.
```

```
# Mac OS X, CPU only, Python 3.4 or 3.5:
(tensorflow)$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/mac/cpu/tensorflow-0.12.1-p

# Mac OS X, GPU enabled, Python 3.4 or 3.5:
(tensorflow)$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/mac/gpu/tensorflow_gpu-0.12
```

Finally install TensorFlow:

```
# Python 2
(tensorflow)$ pip install --ignore-installed --upgrade $TF_BINARY_URL

# Python 3
(tensorflow)$ pip3 install --ignore-installed --upgrade $TF_BINARY_URL
```

Usage

With the conda environment activated, you can now test your installation.

When you are done using TensorFlow, deactivate the environment.

```
(tensorflow)$ source deactivate

$  # Your prompt should change back
```

To use TensorFlow later you will have to activate the conda environment again:

```
$ source activate tensorflow
(tensorflow)$  # Your prompt should change.
# Run Python programs that use TensorFlow.
...
# When you are done using TensorFlow, deactivate the environment.
(tensorflow)$ source deactivate
```

Install IPython

To use tensorflow with IPython it may be necessary to install IPython into the tensorflow environment:

```
$ source activate tensorflow
(tensorflow)$ conda install ipython
```

Similarly, other Python packages like pandas may need to get installed into the tensorflow environment before they can be used together with tensorflow.

Docker installation

Docker is a system to build self contained versions of a Linux operating system running on your machine. When you install and run TensorFlow via Docker it completely isolates the installation from pre-existing packages on your machine.

We provide 4 Docker images:

- `gcr.io/tensorflow/tensorflow` : TensorFlow CPU binary image.
- `gcr.io/tensorflow/tensorflow:latest-devel` : CPU Binary image plus source code.
- `gcr.io/tensorflow/tensorflow:latest-gpu` : TensorFlow GPU binary image.
- `gcr.io/tensorflow/tensorflow:latest-devel-gpu` : GPU Binary image plus source code.

We also have tags with `latest` replaced by a released version (e.g., `0.12.1-gpu`).

With Docker the installation is as follows:

- Install Docker on your machine.
- Create a Docker group to allow launching containers without `sudo` .
- Launch a Docker container with the TensorFlow image. The image gets downloaded automatically on first launch.

See installing Docker for instructions on installing Docker on your machine.

After Docker is installed, launch a Docker container with the TensorFlow binary image as follows.

```
$ docker run -it -p 8888:8888 gcr.io/tensorflow/tensorflow
```

The option `-p 8888:8888` is used to publish the Docker container's internal port to the host machine, in this case to ensure Jupyter notebook connection.

The format of the port mapping is `hostPort:containerPort` . You can specify any valid port number for the host port but have to use `8888` for the container port portion.

If you're using a container with GPU support, some additional flags must be passed to expose the GPU device to the container.

For NVidia GPU support install latest NVidia drivers and nvidia-docker. Run with

```
$ nvidia-docker run -it -p 8888:8888 gcr.io/tensorflow/tensorflow:latest-gpu
```

If you run into a problem running `nvidia-docker` , Please report an issue here.

For more details see TensorFlow docker readme.

You can now test your installation within the Docker container.

Test the TensorFlow installation

(Optional, Linux) Enable GPU Support

If you installed the GPU version of TensorFlow, you must also install the Cuda Toolkit 8.0 and cuDNN v5.1. Please see Cuda installation.

You also need to set the `LD_LIBRARY_PATH` and `CUDA_HOME` environment variables. Consider adding the commands below to your `~/.bash_profile` . These assume your CUDA installation is in `/usr/local/cuda` :

```
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:/usr/local/cuda/lib64:/usr/local/cuda/extras/CUPTI/lib64"
export CUDA_HOME=/usr/local/cuda
```

Run TensorFlow from the Command Line

See common problems if an error happens.

Open a terminal and type the following:

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

Installing from sources

When installing from source you will build a pip wheel that you then install using pip. You'll need pip for that, so install it as described above.

To build TensorFlow from source on Windows, you can use experimental support for Bazel on Windows or the TensorFlow CMake build.

Clone the TensorFlow repository

```
$ git clone https://github.com/tensorflow/tensorflow
```

Note that these instructions will install the latest master branch of tensorflow. If you want to install a specific branch (such as a release branch), pass `-b <branchname>` to the `git clone` command and `--recurse-submodules` for r0.8 and earlier to fetch the protobuf library that TensorFlow depends on.

Prepare environment for Linux

Install Bazel

Follow instructions here to install the dependencies for bazel. Then download the latest stable bazel version using the installer for your system and run the installer as mentioned there:

```
$ chmod +x PATH_TO_INSTALL.SH
$ ./PATH_TO_INSTALL.SH --user
```

Remember to replace `PATH_TO_INSTALL.SH` with the location where you downloaded the installer.

Finally, follow the instructions in that script to place `bazel` into your binary path.

Install other dependencies

```
# For Python 2.7:
$ sudo apt-get install python-numpy python-dev python-wheel python-mock
# For Python 3.x:
$ sudo apt-get install python3-numpy python3-dev python3-wheel python3-mock
```

Optional: Install CUDA (GPUs on Linux)

In order to build or run TensorFlow with GPU support, both NVIDIA's Cuda Toolkit (>= 7.0) and cuDNN (>= v3) need to be installed.

TensorFlow GPU support requires having a GPU card with NVidia Compute Capability (>= 3.0). Supported cards include but are not limited to:

- NVidia Titan
- NVidia Titan X

- NVidia K20
- NVidia K40

Check NVIDIA Compute Capability of your GPU card

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

Download and install Cuda Toolkit

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

Install version 8.0 if using our binary releases.

Install the toolkit into e.g. `/usr/local/cuda`.

Download and install cuDNN

https://developer.nvidia.com/cudnn

Download cuDNN v5.1.

Uncompress and copy the cuDNN files into the toolkit directory. Assuming the toolkit is installed in `/usr/local/cuda`, run the following commands (edited to reflect the cuDNN version you downloaded):

```
tar xvzf cudnn-8.0-linux-x64-v5.1.tgz
sudo cp -P cuda/include/cudnn.h /usr/local/cuda/include/
sudo cp -P cuda/lib64/libcudnn* /usr/local/cuda/lib64/
sudo chmod a+r /usr/local/cuda/include/cudnn.h /usr/local/cuda/lib64/libcudnn*
```

Install other dependencies

```
$ sudo apt-get install libcupti-dev
```

Optional: Install OpenCL (Experimental, Linux only)

In order to build or run TensorFlow with OpenCL support, both OpenCL (>= 1.2) and ComputeCpp (>= 0.1.1) need to be installed.

TensorFlow can only take advantage of accelerators that support OpenCL 1.2. Supported accelerators include but are not limited to:

- AMD Fiji
- AMD Hawaii

Note that this support is currently experimental and should not be relied upon for production (though it will mature over time).

Download and install OpenCL drivers

The exact steps required for a functional OpenCL installation will depend on your environment. For Unbuntu 14.04, the following steps are known to work:

```
sudo apt-get install ocl-icd-opencl-dev opencl-headers
```

You will also need to install the drivers for the accelerator itself. We've tested that the following drivers for AMD Fiji and Hawaii GPUs on Ubuntu 14.04:

```
sudo apt-get install fglrx-core fglrx-dev
```

Download and install the ComputeCpp compiler

Download the compiler from Codeplay's website, uncompress and copy the files into e.g. `/usr/local/computecpp`:

```
tar -xvzf ComputeCpp-CE-0.1.1-Ubuntu.14.04-64bit.tar.gz
sudo mkdir /usr/local/computecpp
sudo cp -R ComputeCpp-CE-0.1.1-Linux /usr/local/computecpp
sudo chmod -R a+r /usr/local/computecpp/
sudo chmod -R a+x /usr/local/computecpp/bin
```

Prepare environment for Mac OS X

We recommend using homebrew to install the bazel dependency, and installing python dependencies using easy_install or pip.

Dependencies

Follow instructions here to install the dependencies for bazel. You can then use homebrew to install bazel:

```
$ brew install bazel
```

You can install the python dependencies using easy_install or pip. Using easy_install, run

```
$ sudo easy_install -U six
$ sudo easy_install -U numpy
$ sudo easy_install wheel
```

We also recommend the ipython enhanced python shell, which you can install as follows:

```
$ sudo easy_install ipython
```

Optional: Setup GPU for Mac

If you plan to build with GPU support you will need to make sure you have GNU coreutils installed via homebrew:

```
$ brew install coreutils
```

Next you will need to make sure you have a recent CUDA Toolkit installed by either downloading the package for your version of OSX directly from NVIDIA or by using the Homebrew Cask extension:

```
$ brew tap caskroom/cask
$ brew cask install cuda
```

Once you have the CUDA Toolkit installed you will need to setup the required environment variables by adding the following to your `~/.bash_profile`:

```
export CUDA_HOME=/usr/local/cuda
export DYLD_LIBRARY_PATH="$DYLD_LIBRARY_PATH:$CUDA_HOME/lib"
export PATH="$CUDA_HOME/bin:$PATH"
```

Finally, you will also want to install the CUDA Deep Neural Network (cuDNN v5.1) library which currently requires an Accelerated Computing Developer Program account. Once you have it downloaded locally, you can unzip and move the header and libraries to your local CUDA Toolkit folder:

```
$ sudo mv include/cudnn.h /Developer/NVIDIA/CUDA-8.0/include/
$ sudo mv lib/libcudnn* /Developer/NVIDIA/CUDA-8.0/lib
$ sudo ln -s /Developer/NVIDIA/CUDA-8.0/lib/libcudnn* /usr/local/cuda/lib/
```

To verify the CUDA installation, you can build and run deviceQuery to make sure it passes.

```
$ cp -r /usr/local/cuda/samples ~/cuda-samples
```

```
$ pushd ~/cuda-samples
$ make
$ popd
$ ~/cuda-samples/bin/x86_64/darwin/release/deviceQuery
```

If you want to compile tensorflow and have XCode 7.3 and CUDA 7.5 installed, note that Xcode 7.3 is not yet compatible with CUDA 7.5. You can either upgrade to CUDA 8.0, or you will need to download Xcode 7.2 and select it as your default:

```
$ sudo xcode-select -s /Application/Xcode-7.2/Xcode.app
```

Configure the installation

Run the `configure` script at the root of the tree. The configure script asks you for the path to your python interpreter and allows (optional) configuration of the CUDA libraries.

This step is used to locate the python and numpy header files as well as enabling GPU support if you have a CUDA enabled GPU and Toolkit installed. Select the option `Y` when asked to build TensorFlow with GPU support.

If you have several versions of Cuda or cuDNN installed, you should definitely select one explicitly instead of relying on the system default.

For example:

```
$ ./configure
Please specify the location of python. [Default is /usr/bin/python]:
Do you wish to build TensorFlow with Google Cloud Platform support? [y/N] N
No Google Cloud Platform support will be enabled for TensorFlow
Do you wish to build TensorFlow with GPU support? [y/N] y
Do you wish to build TensorFlow with OpenCL support? [y/N] N
GPU support will be enabled for TensorFlow
Please specify which gcc nvcc should use as the host compiler. [Default is /usr/bin/gcc]:
Please specify the Cuda SDK version you want to use, e.g. 7.0. [Leave empty to use system default]: 8.0
Please specify the location where CUDA 8.0 toolkit is installed. Refer to README.md for more details. [D
Please specify the cuDNN version you want to use. [Leave empty to use system default]: 5
Please specify the location where cuDNN 5 library is installed. Refer to README.md for more details. [De
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 s

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

This creates a canonical set of symbolic links to the Cuda libraries on your system. Every time you change the Cuda library paths you need to run this step again before you invoke the bazel build command. For the cuDNN libraries, use '7.0' for R3, and '4.0.7' for R4.

If you want to built support for OpenCL, select the option `Y` when asked to build TensorFlow with OpenCL support, and provide the location of the ComputeCpp compiler (e.g. /usr/local/computecpp).

Known issues

- Although it is possible to build both Cuda and non-Cuda configs under the same source tree, we recommend to run `bazel clean` when switching between these two configs in the same source tree.
- You have to run configure before running bazel build. Otherwise, the build will fail with a clear error message. In the future, we might consider making this more convenient by including the configure step in our build process.

Create the pip package and install

When building from source, you will still build a pip package and install that.

Please note that building from sources takes a lot of memory resources by default and if you want to limit RAM usage you can add `--local_resources 2048,.5,1.0` while invoking bazel.

```
$ bazel build -c opt //tensorflow/tools/pip_package:build_pip_package

# To build with support for CUDA:
$ bazel build -c opt --config=cuda //tensorflow/tools/pip_package:build_pip_package

# Alternatively, to build with support for OpenCL:
$ bazel build -c opt --config=sycl //tensorflow/tools/pip_package:build_pip_package

$ bazel-bin/tensorflow/tools/pip_package/build_pip_package /tmp/tensorflow_pkg

# The name of the .whl file will depend on your platform.
$ sudo pip install /tmp/tensorflow_pkg/tensorflow-0.12.1-py2-none-any.whl
```

Optimizing CPU performance

To be compatible with as wide a range of machines as possible, TensorFlow defaults to only using SSE4.1 SIMD instructions on x86 machines. Most modern PCs and Macs support more advanced instructions, so if you're building a binary that you'll only be running on your own machine, you can enable these by using `--copt=-march=native` in your bazel build command. For example:

```
$ bazel build --copt=-march=native -c opt //tensorflow/tools/pip_package:build_pip_package
```

If you are distributing a binary but know the capabilities of the machines you'll be running on, you can manually choose the right instructions with something like `--copt=-march=avx`. You may also want to enable multiple features using several arguments, for example `--copt=-mavx2 --copt=-mfma`.

If you run a binary built using SIMD instructions on a machine that doesn't support them, you'll see an illegal instruction error when that code is executed.

Setting up TensorFlow for Development

If you're working on TensorFlow itself, it is useful to be able to test your changes in an interactive python shell without having to reinstall TensorFlow.

To set up TensorFlow such that all files are linked (instead of copied) from the system directories, run the following commands inside the TensorFlow root directory:

```
bazel build -c opt //tensorflow/tools/pip_package:build_pip_package

# To build with GPU support:
bazel build -c opt --config=cuda //tensorflow/tools/pip_package:build_pip_package

mkdir _python_build
cd _python_build
ln -s ../bazel-bin/tensorflow/tools/pip_package/build_pip_package.runfiles/org_tensorflow/* .
ln -s ../tensorflow/tools/pip_package/* .
python setup.py develop
```

Note that this setup still requires you to rebuild the `//tensorflow/tools/pip_package:build_pip_package` target every time you change a C++ file; add, delete, or move any python file; or if you change bazel build rules.

Note also that `bazel test` will not always properly resolve dependencies through these symlinks, so test results may be unreliable. A workaround is to remove the `_python_build` directory before running `bazel test`.

One more thing is that `python setup.py development` does not install the packages listed in `REQUIRED_PACKAGES` part of `setup.py`. You might need to install them separately.

Train your first TensorFlow neural net model

Start by cloning the TensorFlow models repo from GitHub. Run the following commands:

```
$ cd models/tutorials/image/mnist
$ python convolutional.py
Successfully downloaded train-images-idx3-ubyte.gz 9912422 bytes.
Successfully downloaded train-labels-idx1-ubyte.gz 28881 bytes.
Successfully downloaded t10k-images-idx3-ubyte.gz 1648877 bytes.
Successfully downloaded t10k-labels-idx1-ubyte.gz 4542 bytes.
Extracting data/train-images-idx3-ubyte.gz
Extracting data/train-labels-idx1-ubyte.gz
Extracting data/t10k-images-idx3-ubyte.gz
Extracting data/t10k-labels-idx1-ubyte.gz
Initialized!
Epoch 0.00
Minibatch loss: 12.054, learning rate: 0.010000
Minibatch error: 90.6%
Validation error: 84.6%
Epoch 0.12
Minibatch loss: 3.285, learning rate: 0.010000
Minibatch error: 6.2%
Validation error: 7.0%
...
...
```

Common Problems

GPU-related issues

If you encounter the following when trying to run a TensorFlow program:

```
ImportError: libcudart.so.7.0: cannot open shared object file: No such file or directory
```

Make sure you followed the GPU installation instructions. If you built from source, and you left the Cuda or cuDNN version empty, try specifying them explicitly.

Protobuf library related issues

TensorFlow pip package depends on protobuf pip package version 3.1.0. Protobuf's pip package downloaded from PyPI (when running `pip install protobuf`) is a Python only library, that has Python implementations of proto serialization/deserialization which can be 10x-50x slower than the C++ implementation. Protobuf also supports a binary extension for the Python package that contains fast C++ based proto parsing. This extension is not available in the standard Python only PIP package. We have created a custom binary pip package for protobuf that contains the binary extension. Follow these instructions to install the custom binary protobuf pip package:

```
# Ubuntu/Linux 64-bit:
$ pip install --upgrade https://storage.googleapis.com/tensorflow/linux/cpu/protobuf-3.1.0-cp27-none-lin

# Mac OS X:
$ pip install --upgrade https://storage.googleapis.com/tensorflow/mac/cpu/protobuf-3.1.0-cp27-none-macos
```

And for Python 3.5:

```
# Ubuntu/Linux 64-bit:
$ pip3 install --upgrade https://storage.googleapis.com/tensorflow/linux/cpu/protobuf-3.1.0-cp35-none-li

# Mac OS X:
$ pip3 install --upgrade https://storage.googleapis.com/tensorflow/mac/cpu/protobuf-3.1.0-cp35-none-mac
```

If your system/configuration is not listed above, you can use the following instructions to build your own protobuf wheel file. To install its prerequisites, see here:

Then:

```
$ git clone https://github.com/google/protobuf.git
$ cd protobuf
$ ./autogen.sh
$ CXXFLAGS="-fPIC -g -O2" ./configure
$ make -j12
$ export PROTOC=$PWD/src/protoc
$ cd python
$ python setup.py bdist_wheel --cpp_implementation --compile_static_extension
$ pip uninstall protobuf
$ pip install dist/<wheel file name>
```

Install the above package *after* you have installed TensorFlow via pip, as the standard `pip install tensorflow` would install the python only pip package. The above pip package will over-write the existing protobuf package. Note that the binary pip package already has support for protobuf larger than 64MB, that should fix errors such as these :

```
[libprotobuf ERROR google/protobuf/src/google/protobuf/io/coded_stream.cc:207] A
protocol message was rejected because it was too big (more than 67108864 bytes).
To increase the limit (or to disable these warnings), see
CodedInputStream::SetTotalBytesLimit() in google/protobuf/io/coded_stream.h.
```

Pip installation issues

Cannot import name 'descriptor'

```
ImportError: Traceback (most recent call last):
  File "/usr/local/lib/python3.4/dist-packages/tensorflow/core/framework/graph_pb2.py", line 6, in <modu
    from google.protobuf import descriptor as _descriptor
ImportError: cannot import name 'descriptor'
```

If you the above error when upgrading to a newer version of TensorFlow, try uninstalling both TensorFlow and protobuf (if installed) and re-installing TensorFlow (which will also install the correct protobuf dependency).

Can't find setup.py

If, during `pip install`, you encounter an error like:

```
...
IOError: [Errno 2] No such file or directory: '/tmp/pip-o6Tpui-build/setup.py'
```

Solution: upgrade your version of pip:

```
pip install --upgrade pip
```

This may require `sudo`, depending on how pip is installed.

SSLError: SSL_VERIFY_FAILED

If, during pip install from a URL, you encounter an error like:

```
...
SSLError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed
```

Solution: Download the wheel manually via curl or wget, and pip install locally.

Operation not permitted

If, despite using `sudo`, you encounter an error like:

```
...
Installing collected packages: setuptools, protobuf, wheel, numpy, tensorflow
Found existing installation: setuptools 1.1.6
Uninstalling setuptools-1.1.6:
Exception:
...
[Errno 1] Operation not permitted: '/tmp/pip-a1DXRT-uninstall/System/Library/Frameworks/Python.framework
```

Solution: Add an `--ignore-installed` flag to the pip command.

Cannot remove entries from nonexistent file: easy-install.pth

If during a `pip` installation using an Anaconda Python distribution you encounter the error:

```
Cannot remove entries from nonexistent file <path-to-anaconda-
instalation>/anaconda[version]/lib/site-packages/easy-install.pth
```

1. Upgrade setuptools:
   `pip install --upgrade -I setuptools`
2. Install TensorFlow again adding `--ignore-installed` flag:
   `pip install --ignore-installed --upgrade <tensorflow_url>`

Step #1 might already solve the problem, however if it still persists, execute step #2.

This issue occurs with new Anaconda installations when `pip` tries to remove `easy-install.pth`. This file is not included in Anaconda packages, which causes the `pip` installation to fail.

Cupti_wrapper.cc: Could not find cuptiActivityRegisterCallbacksin libcupti DSO

If, when running a TensorFlow Python script, you encounter the following error:

```
c:\tf_jenkins\home\workspace\nightly-
win\device\gpu\os\windows\tensorflow\core\platform\default\gpu\cupti_wrapper.cc:59] Check failed:
::tensorflow::Status::OK() == (::tensorflow::Env::Default()->GetSymbolFromLibrary( GetDsoHandle(),
kName, &f  )) (OK vs. Not found: cuptiActivityRegisterCallbacks not found)could not find
cuptiActivityRegisterCallbacksin libcupti DSO
```

Add `<path-to-cuda-folder>\NVIDIA GPU Computing Toolkit\CUDA\v8.0\extras\CUPTI\libx64` to your `PATH`.

This issue occurs because on CUDA 8.0 the location of the file `cupti64_80.dll` is not on `PATH` by default.

Linux issues

If you encounter:

```
...
 "__add__", "__radd__",
             ^
SyntaxError: invalid syntax
```

Solution: make sure you are using Python 2.7.

Ubuntu build issue on Linux 16.04 when building with --config=cuda: build fail with cuda: identifier "__builtin_ia32_mwaitx" is undefined.

GitHub issue: https://github.com/tensorflow/tensorflow/issues/1066

Solution: Add the following compiler flags to third_party/gpus/crosstool/CROSSTOOL

cxx_flag: "-D_MWAITXINTRIN_H_INCLUDED" cxx_flag: "-D_FORCE_INLINES"

Mac OS X: ImportError: No module named copyreg

On Mac OS X, you may encounter the following when importing tensorflow.

```
>>> import tensorflow as tf
...
ImportError: No module named copyreg
```

Solution: TensorFlow depends on protobuf, which requires the Python package `six-1.10.0`. Apple's default Python installation only provides `six-1.4.1`.

You can resolve the issue in one of the following ways:

- Upgrade the Python installation with the current version of `six`:

```
$ sudo easy_install -U six
```

- Install TensorFlow with a separate Python library:
  - Using Virtualenv.
  - Using Docker.
- Install a separate copy of Python via Homebrew or MacPorts and re-install TensorFlow in that copy of Python.

Mac OS X: OSError: [Errno 1] Operation not permitted:

On El Capitan, "six" is a special package that can't be modified, and this error is reported when "pip install" tried to modify this package. To fix use "ignore-installed" flag, ie

sudo pip install --ignore-installed six https://storage.googleapis.com/....

Mac OS X: TypeError: `__init__()` got an unexpected keyword argument 'syntax'

On Mac OS X, you may encounter the following when importing tensorflow.

```
>>> import tensorflow as tf
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/local/lib/python2.7/site-packages/tensorflow/__init__.py", line 4, in <module>
    from tensorflow.python import *
  File "/usr/local/lib/python2.7/site-packages/tensorflow/python/__init__.py", line 13, in <module>
    from tensorflow.core.framework.graph_pb2 import *
...
  File "/usr/local/lib/python2.7/site-packages/tensorflow/core/framework/tensor_shape_pb2.py", line
22, in <module>

serialized_pb=_b('\n,tensorflow/core/framework/tensor_shape.proto\x12\ntensorflow\"d\n\x10TensorShapePro
\n\x03\x64im\x18\x02 \x03(\x0b\x32
.tensorflow.TensorShapeProto.Dim\x1a!\n\x03\x44im\x12\x0c\n\x04size\x18\x01
\x01(\x03\x12\x0c\n\x04name\x18\x02 \x01(\tb\x06proto3')
TypeError: __init__() got an unexpected keyword argument 'syntax'
```

This is due to a conflict between protobuf versions (we require protobuf 3.1.0). The best current solution is to make sure older versions of protobuf are not installed, such as:

```
$ pip install --upgrade protobuf
```

Mac OS X: Segmentation Fault when import tensorflow

On Mac OS X, you might get the following error when importing tensorflow in python:

```
>>> import tensorflow
```

```
I tensorflow/stream_executor/dso_loader.cc:108] successfully opened CUDA library libcublas.dylib
locally
I tensorflow/stream_executor/dso_loader.cc:108] successfully opened CUDA library libcudnn.dylib
locally
I tensorflow/stream_executor/dso_loader.cc:108] successfully opened CUDA library libcufft.dylib
locally
"import tensorflow" terminated by signal SIGSEGV (Address boundary error)
```

This is due to the fact that by default, cuda creates libcuda.dylib, but tensorflow tries to load libcuda.1.dylib. This can be resolved by create a symbolic link:

```
ln -sf /usr/local/cuda/lib/libcuda.dylib /usr/local/cuda/lib/libcuda.1.dylib
```

Mac OS X: RuntimeError: Broken toolchain: cannot link a simple C program

On Mac OS X, when installing tensorflow you might see lots of warnings and errors, ending with a `Broken toolchain: cannot link a simple C program` message:

```
>>> sudo pip install --upgrade $TF_BINARY_URL

...<lots more warnings and errors>

You have not agreed to the Xcode license agreements, please run 'xcodebuild -license' (for user-level
acceptance) or 'sudo xcodebuild -license' (for system-wide acceptance) from within a Terminal window
to review and agree to the Xcode license agreements.

...<more stack trace output>

  File "numpy/core/setup.py", line 653, in get_mathlib_info

    raise RuntimeError("Broken toolchain: cannot link a simple C program")

RuntimeError: Broken toolchain: cannot link a simple C program
```

This is typically because you have the Xcode build tools installed, but you still need to accept the license agreements. To resolve it, accept the license agreement by opening Xcode, or by running `xcodebuild -license` from the command line.

Import Error

When importing tensorflow, you may see an "ImportError" raised. Below are some possible examples and solutions:

```
ImportError: /lib64/libc.so.6: version `GLIBC_2.16' not found (required by ..._pywrap_tensorflow.so)
```

This can occur if your operating system libraries are too old for our provided pip package binaries. Solution: Try building from sources.

```
ImportError: cannot import name pywrap_tensorflow
```

This can occur if you happen to be in the tensorflow source code directory and try to import tensorflow: the order of search path prefers the current directory, and so tries to import directly from the source code instead of your installed tensorflow package. Solution: don't import tensorflow from the tensorflow source code root directory, if you are.