Chapter 01: Getting Started with TensorFlow

```
ubuntu@ubuntu-PC:~$ sudo apt-get install python-pip python-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
python-dev is already the newest version (2.7.11-1).
python-pip is already the newest version (8.1.1-2ubuntu0.4).
0 upgraded, 0 newly installed, 0 to remove and 222 not upgraded.
ubuntu@ubuntu-PC:~$
```

```
100% | 890kB 924kB/s
Collecting werkzeug>=0.11.10 (from tensorflow-tensorboard<0.2.0,>=0.1.0->tensorflow)
  Downloading Werkzeug-0.12.2-py2.py3-none-any.whl (312kB)
100% | 317kB 965kB/s
Collecting setuptools (from protobuf>=3.3.0->tensorflow)
  Downloading setuptools-36.5.0-py2.py3-none-any.whl (478kB)
100% | 481kB 852kB/s
Installing collected packages: six, funcsigns, pbr, mock, numpy, backports.wkeyword, bleach, markdown, setuptools, protobuf, werkzeug, tensorflow-tensorboard,
Found existing installation: six 1.10.0
  Not uninstalling six at /usr/lib/python2.7/dist-packages, outside environment
Found existing installation: wheel 0.29.0
  Not uninstalling wheel at /usr/lib/python2.7/dist-packages, outside environment
Running setup.py install for html5lib ... done
Running setup.py install for markdown ... done
Found existing installation: setuptools 20.7.0
  Not uninstalling setuptools at /usr/lib/python2.7/dist-packages, outside environment
Successfully installed backports.wkeyword-1.0.post1 bleach-1.5.0 funcsigns-1.0.2
999 markdown-2.6.9 mock-2.0.0 numpy-1.13.1 pbr-3.1.1 protobuf-3.4.0 setuptools-
0 tensorflow-1.3.0 tensorflow-tensorboard-0.1.6 werkzeug-0.12.2 wheel-0.30.0
You are using pip version 8.1.1, however version 9.0.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
ubuntu@ubuntu-PC:~$
```
Last login: Thu Feb 18 12:52:14 on ttys001
~ @ alpha-al-ghaib (saif) :: sudo easy_install pip
Password:
Searching for pip
Best match: pip 8.0.2
Adding pip 8.0.2 to easy-install.pth file
Installing pip script to /Users/saif/anaconda/bin
Installing pip2.7 script to /Users/saif/anaconda/bin
Installing pip2 script to /Users/saif/anaconda/bin

Using /Users/saif/anaconda/lib/python2.7/site-packages
Processing dependencies for pip
Finished processing dependencies for pip

~ @ alpha-al-ghaib (saif) :: sudo easy_install --upgrade six
Password:
Searching for six
Reading https://pypi.python.org/simple/six/
Best match: six 1.10.0
Processing six-1.10.0-py2.7.egg
six 1.10.0 is already the active version in easy-install.pth

Using /Users/saif/anaconda/lib/python2.7/site-packages/six-1.10.0-py2.7.egg
Processing dependencies for six
Finished processing dependencies for six
Install Python 3.6.2 (64-bit)
Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

Install Now
C:\Users\Quan Hua\AppData\Local\Programs\Python\Python36
Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

☑️ Install launcher for all users (recommended)
☐ Add Python 3.6 to PATH

Cancel
Setup was successful

Special thanks to Mark Hammond, without whose years of freely shared Windows expertise, Python for Windows would still be Python for DOS.

New to Python? Start with the online tutorial and documentation.

See what's new in this release.

Disable path length limit
Changes your machine configuration to allow programs, including Python, to bypass the 260 character "MAX_PATH" limitation.
pip3 install tensorflow
Collecting numpy>=1.11.0 (from tensorflow)
  Downloading numpy-1.13.1-cp38-none-win_amd64.whl (7.8MB) 100%
  Downloading wheel-0.30.0-py2.py3-none-any.whl (49kB) 100%
Collecting protobuf>=3.3.0 (from tensorflow)
  Downloading protobuf-3.4.0-py2.py3-none-any.whl (375kB) 100%
Collecting six>=1.10.0 (from tensorflow)
  Downloading six-1.11.0-py2.py3-none-any.whl
Collecting bleach>=1.5.0 (from tensorflow-tensorboard<0.2.0,>=0.1.0->tensorflow)
  Downloading bleach-1.3.0-py2.py3-none-any.whl
Collecting html5lib==0.9999999 (from tensorflow-tensorboard<0.2.0,>=0.1.0->tensorflow)
  Downloading html5lib-0.9999999.tar.gz (889kB) 100%
Collecting werkzeug>=0.11.10 (from tensorflow-tensorboard<0.2.0,>=0.1.0->tensorflow)
  Downloading werkzeug-0.11.2-py2.py3-none-any.whl (312kB) 100%
Collecting markdown==2.6.8 (from tensorflow-tensorboard<0.2.0,>=0.1.0->tensorflow)
  Downloading markdown-2.6.9.tar.gz (271kB) 100%
 Requirement already satisfied: setuptools in c:\users\quang hua\appdata\local\programs\python\... (from protobuf>=3.3.0->tensorflow)
Installing collected packages: six, protobuf, numpy, html5lib, bleach, wheel, werkzeug, markdown, tensorflow
Running setup.py install for html5lib ... done
Running setup.py install for markdown ... done
Successfully installed bleach-1.5.0 html5lib-0.9999999 markdown-2.6.9 numpy-1.13.1 protobuf-3.3.0 tensorflow-tensorboard-0.1.6 werkzeug-0.11.2 wheel-0.30.0
PS C:\Users\Quan Hua>
Welcome to VirtualBox!

The left part of this window is a list of all virtual machines on your computer. The list is empty now because you haven't created any virtual machines yet.

In order to create a new virtual machine, press the New button in the main tool bar located at the top of the window.

You can press the F1 key to get instant help, or visit www.virtualbox.org for the latest information and news.
Create Virtual Machine

Name and operating system

Please choose a descriptive name for the new virtual machine and select the type of operating system you intend to install on it. The name you choose will be used throughout VirtualBox to identify this machine.

Name: TensorFlow
Type: Linux
Version: Ubuntu (64-bit)
Memory size

Select the amount of memory (RAM) in megabytes to be allocated to the virtual machine.

The recommended memory size is **768 MB**.
Create Virtual Machine

Hard disk

If you wish you can add a virtual hard disk to the new machine. You can either create a new hard disk file or select one from the list or from another location using the folder icon.

If you need a more complex storage set-up you can skip this step and make the changes to the machine settings once the machine is created.

The recommended size of the hard disk is **8.00 GB**.

- [ ] Do not add a virtual hard disk
- [x] Create a virtual hard disk now
- [ ] Use an existing virtual hard disk file

[Create]  [Cancel]
Create Virtual Hard Disk

Hard disk file type

Please choose the type of file that you would like to use for the new virtual hard disk. If you do not need to use it with other virtualization software you can leave this setting unchanged.

- VDI (VirtualBox Disk Image)
- VMDK (Virtual Machine Disk)
- VHD (Virtual Hard Disk)
- HDD (Parallels Hard Disk)
- QED (QEMU enhanced disk)
- QCOW (QEMU Copy-On-Write)
Create Virtual Hard Disk

Hard disk file type

Please choose the type of file that you would like to use for the new virtual hard disk. If you do not need to use it with other virtualization software you can leave this setting unchanged.

- VDI (VirtualBox Disk Image)
- VMDK (Virtual Machine Disk)
- VHDI (Virtual Hard Disk)
- HDD (Parallels Hard Disk)
- QED (QEMU enhanced disk)
- QCOW (QEMU Copy-On-Write)
Create Virtual Hard Disk

Storage on physical hard disk

Please choose whether the new virtual hard disk file should grow as it is used (dynamically allocated) or if it should be created at its maximum size (fixed size).

A **dynamically allocated** hard disk file will only use space on your physical hard disk as it fills up (up to a maximum **fixed size**), although it will not shrink again automatically when space on it is freed.

A **fixed size** hard disk file may take longer to create on some systems but is often faster to use.

- Dynamically allocated
- Fixed size

[Next] [Cancel]
Create Virtual Hard Disk

File location and size

Please type the name of the new virtual hard disk file into the box below or click on the folder icon to select a different folder to create the file in.

TensorFlowHD

Select the size of the virtual hard disk in megabytes. This size is the limit on the amount of file data that a virtual machine will be able to store on the hard disk.

4.00 MB | 12.00 GB | 2.00 TB

Create  Cancel
Select start-up disk

Please select a virtual optical disk file or a physical optical drive containing a disk to start your new virtual machine from.

The disk should be suitable for starting a computer from and should contain the operating system you wish to install on the virtual machine if you want to do that now. The disk will be ejected from the virtual drive automatically next time you switch the virtual machine off, but you can also do this yourself if needed using the Devices menu.

Ubuntu-gnome-14.04-desktop-amd64.iso (923M)
Python 3.6.2 (v3.6.2:5fd33b5, Jul 18 2017, 04:57:36) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow as tf
>>> a = tf.constant(1.0)
>>> b = tf.constant(2.0)
>>> c = a + b
>>> sess = tf.Session()
2017-09-27 01:35:50.710620: W C:\tf_jenkins\home\workspace\rel-win\M\windows\PY\36\tensorflow\core\
guard.cc:45: The TensorFlow library wasn't compiled to use AVX instructions, but these are available and could speed up CPU computations.
2017-09-27 01:35:50.710761: W C:\tf_jenkins\home\workspace\rel-win\M\windows\PY\36\tensorflow\core\
guard.cc:45: The TensorFlow library wasn't compiled to use AVX2 instructions, but these are available and could speed up CPU computations.
>>> print(sess.run(c))
3.0
>>>
Chapter 02: Your First Classifier

```
/notMNIST_small/J
Started loading images from: /home/ubuntu/github/mlwithef/datasets/notMNIST/test
/notMNIST_small/J
Finished loading data from: /home/ubuntu/github/mlwithef/datasets/notMNIST/test/
/notMNIST_small/J
    Started pickling: J
    Finished pickling: J
    Finished loading testing data
    Started pickling final dataset
    Merging train, valid data
    Merging test data
        ("Training set", (200000, 28, 28), (200000,))
        ("Validation set", (10000, 28, 28), (10000,))
        ("Test set", (10000, 28, 28), (10000,))
        ("Compressed pickle size:", 69880514)
    Finished pickling final dataset
    Finished preparing notMNIST dataset
    After reformat:
        ("Training set", (200000, 784), (200000, 10))
        ("Validation set", (10000, 784), (10000, 10))
        ("Test set", (10000, 784), (10000, 10))
[work2] ubuntu@ubuntu-PC:/github/mlwithef/chapter_02$  ```
with tf.Session(graph=graph) as session:
    session.run(tf.global_variables_initializer())
    print("Initialized")

    for step in range(num_steps + 1):
        sys.stdout.write("Training on batch %d of %d\r" % (step + 1, num_steps))
        sys.stdout.flush()

        # Pick an offset within the training data, which has been randomized.
        # Note: we could use better randomization across epochs.
        offset = (step * batch_size) % (dataset.train_labels.shape[0] - batch_size)

        # Generate a minibatch.
        batch_data = dataset.train_dataset[offset:(offset + batch_size), :]
        batch_labels = dataset.train_labels[offset:(offset + batch_size), :]

        # Prepare a dictionary telling the session where to feed the minibatch.
        # The key of the dictionary is the placeholder node of the graph to be fed.
        # The value of the dictionary is the numpy array to feed to it.
        feed_dict = {tf_train_dataset: batch_data, tf_train_labels: batch_labels}
        _, predictions = session.run([optimizer, loss, train_prediction], feed_dict=feed_dict)

        if step % data_showing_step == 0:
            acc_minibatch = accuracy(predictions, batch_labels)
            acc_val = accuracy(valid_prediction.eval(), dataset.valid_labels)

            logmanager.logger.info("%d: Acc Train: %2.4f\% Acc Val: %2.4f\% Loss %2.4f\%" %
                                (step, acc_minibatch, acc_val, 1))

            logmanager.logger.info("Test accuracy: %2.4f\%" % accuracy(test_prediction.eval(), dataset.test_labels))
TensorBoard

- Regex filter
- Split on underscores
- Data download links

### Horizontal Axis

- **STEP**
- RELATIVE
- WALL

### Runs

- MNIST_Run1
- MNIST_Run2

### EVENTS

- learning_rate

  ![Graph of learning_rate](image1)

- loss

  ![Graph of loss](image2)
conv3/weights
(weights)
Operation: Variable
Attributes (4)
- container: (type:"DT_FLOAT")
- shape: (shape:["size":3], 
  (size:3),(size:192), 
  (size:384)])
- shared_name: (s:"")
Inputs (0)
Outputs (2)
- conv3/weights/Assign
- conv3/weights/read
Add to main graph
Chapter 04: Cats and Dogs
def nn_model(data, weights, biases):
    layer_fc1 = tf.matmul(data, weights['fc1']) + biases['fc1']
    relu_layer = tf.nn.relu(layer_fc1)
    for relu in range(2, relu_layers + 1):
        relu_layer = tf.nn.relu(relu_layer)
    return tf.matmul(relu_layer, weights['fc2']) + biases['fc2']
```python
weights = {
    'conv1': tf.Variable(tf.truncated_normal(shape=[patch_size, patch_size, num_channels, depth_inc], dtype=tf.float32, stddev=stddev, seed=SEED), name='weights_conv1'),
    'conv2': tf.Variable(tf.truncated_normal(shape=[patch_size, patch_size, depth_inc, depth_inc], dtype=tf.float32, stddev=stddev, seed=SEED), name='weights_conv2'),
    'conv3': tf.Variable(tf.truncated_normal(shape=[patch_size, patch_size, depth_inc, depth_inc], dtype=tf.float32, stddev=stddev, seed=SEED), name='weights_conv3'),
    'fc1': tf.Variable(tf.truncated_normal([fc_first_layer_dim(Image_size, conv_layers) * 2] * depth_inc, num_channels, num_hidden_inc), dtype=tf.float32, stddev=stddev, seed=SEED),
    'fc2': tf.Variable(tf.truncated_normal([num_hidden_inc, num_of_classes], dtype=tf.float32, stddev=stddev, seed=SEED), name='weights_fc2')
}
```

```python
biases = {
    'conv1': tf.Variable(tf.zeros([depth_inc], dtype=tf.float32), name='biases_conv1'),
    'conv2': tf.Variable(tf.zeros([depth_inc], dtype=tf.float32), name='biases_conv2'),
    'conv3': tf.Variable(tf.zeros([depth_inc], dtype=tf.float32), name='biases_conv3'),
    'fc1': tf.Variable(tf.zeros([num_hidden_inc], dtype=tf.float32), name='biases_fc1'),
    'fc2': tf.Variable(tf.zeros([num_of_classes], dtype=tf.float32), name='biases_fc2'),
}
```

with tf.Session(graph=graph) as session:
    writer = tf.summary.FileWriter(log_location, session.graph)
    merged = tf.summary.merge_all()
    tf.global_variables_initializer().run()
    print("Initialized")

    for step in range(num_steps + 1):
        sys.stdout.flush()

        # Pick an offset within the training data, which has been randomized.
        # Note: we could use better randomization across epochs.
        offset = (step * batch_size) % (dataset.train_labels.shape[0] - batch_size)

        train_minibatch = dataset.train_dataset[offset:(offset + batch_size), :]
        train_labels = dataset.train_labels[offset:(offset + batch_size), :]

        # Prepare a dictionary telling the session where to feed the minibatch.
        # The key of the dictionary is the placeholder node of the graph to be fed,
        # and the value is the numpy array to feed to it.
        feed_dict = {tf.train_dataset: batch_data, tf.train_labels: batch_labels}

        summary_result, _, I, predictions = session.run([merged, optimizer, loss, train_prediction], feed_dict=feed_dict)
        writer.add_summary(summary_result, step)

        if step % data_showing_step == 0:
            acc_minibatch = accuracy(predictions, batch_labels)
            acc_val = accuracy(valid_prediction.eval(), dataset.valid_labels)

            logmanager.logger.info("%03d Acc Train: %03.3f Acc Val: %03.3f Loss %f %
            step, acc_minibatch, acc_val, l))

            logmanager.logger.info("Test accuracy: %.3f% "accucuracy(test_prediction.eval(), dataset.test_labels)")

2017-09-09 03:32:05,861 - MLuithTF - INFO - # 19500 Acc Train: 90.62% Acc Val: 87.45% Loss 0.219110
2017-09-09 03:32:06,640 - MLuithTF - INFO - # 26800 Acc Train: 93.75% Acc Val: 87.41% Loss 0.305260
2017-09-09 03:32:07,419 - MLuithTF - INFO - # 26500 Acc Train: 90.62% Acc Val: 86.87% Loss 0.331279
2017-09-09 03:32:08,204 - MLuithTF - INFO - # 21800 Acc Train: 83.38% Acc Val: 84.39% Loss 0.520968
2017-09-09 03:32:08,906 - MLuithTF - INFO - # 21500 Acc Train: 87.59% Acc Val: 86.68% Loss 0.547694
2017-09-09 03:32:09,706 - MLuithTF - INFO - # 22000 Acc Train: 84.38% Acc Val: 87.15% Loss 0.726970
2017-09-09 03:32:10,500 - MLuithTF - INFO - # 22500 Acc Train: 81.10% Acc Val: 86.78% Loss 0.871583
2017-09-09 03:32:11,323 - MLuithTF - INFO - # 23000 Acc Train: 81.25% Acc Val: 87.15% Loss 0.693311
2017-09-09 03:32:11,323 - MLuithTF - INFO - # 23500 Acc Train: 81.25% Acc Val: 87.44% Loss 0.543187
2017-09-09 03:32:12,883 - MLuithTF - INFO - # 24800 Acc Train: 87.59% Acc Val: 87.81% Loss 0.503170
2017-09-09 03:32:13,681 - MLuithTF - INFO - # 24500 Acc Train: 93.75% Acc Val: 87.22% Loss 0.329258
2017-09-09 03:32:14,481 - MLuithTF - INFO - # 25000 Acc Train: 90.62% Acc Val: 87.72% Loss 0.281230
2017-09-09 03:32:15,233 - MLuithTF - INFO - # 25500 Acc Train: 78.40% Acc Val: 87.40% Loss 0.682225
2017-09-09 03:32:16,083 - MLuithTF - INFO - # 26000 Acc Train: 90.62% Acc Val: 87.01% Loss 0.585085
2017-09-09 03:32:16,782 - MLuithTF - INFO - # 26500 Acc Train: 93.75% Acc Val: 87.41% Loss 0.249685
2017-09-09 03:32:17,563 - MLuithTF - INFO - # 27000 Acc Train: 96.88% Acc Val: 87.24% Loss 0.255854
2017-09-09 03:32:18,330 - MLuithTF - INFO - # 27500 Acc Train: 94.38% Acc Val: 87.59% Loss 0.757773
2017-09-09 03:32:19,118 - MLuithTF - INFO - # 28000 Acc Train: 84.38% Acc Val: 87.34% Loss 0.543425
2017-09-09 03:32:19,903 - MLuithTF - INFO - # 28500 Acc Train: 93.75% Acc Val: 87.57% Loss 0.428885
2017-09-09 03:32:20,690 - MLuithTF - INFO - # 29000 Acc Train: 87.50% Acc Val: 87.22% Loss 0.393810
2017-09-09 03:32:21,479 - MLuithTF - INFO - # 29500 Acc Train: 84.38% Acc Val: 87.82% Loss 0.402495
2017-09-09 03:32:22,240 - MLuithTF - INFO - # 30000 Acc Train: 90.62% Acc Val: 87.75% Loss 0.379222
2017-09-09 03:32:22,240 - MLuithTF - INFO - Test accuracy: 93.8%
```python
def load_cifar_10_pickle(pickle_file, image_depth):
    fo = open(pickle_file, 'rb')
    dict = pickle.load(fo)
    fo.close()
    return ((dict['data'].astype(float) / image_depth / 2) / (image_depth)), dict['labels']

def load_cifar_10_from_pickles(train_pickle_files, test_pickle_files, pickle_batch_size, image_size, image_depth, num_of_channels):
    all_train_data = np.asarray([shape(pickle_batch_size * len(train_pickle_files), image_size, image_size, num_of_channels),
                                 dtype=np.float32])
    all_train_labels = np.asarray([shape(pickle_batch_size * len(train_pickle_files), test_pickle_files),
                                    dtype=np.float32])
    all_test_data = np.asarray([shape(pickle_batch_size * len(test_pickle_files), image_size, image_size, num_of_channels),
                                dtype=np.float32])
    all_test_labels = np.asarray([shape(pickle_batch_size * len(test_pickle_files), test_pickle_files),
                                   dtype=np.float32])
    print('Started loading training data:
    all_train_data[0] = pickle_batch_size * [index + 1] * pickle_batch_size,
    all_train_labels[0] = pickle_batch_size * [index + 1] * pickle_batch_size,
    load_cifar_10_pickle(train_pickle_file, image_depth)
    print('Finished loading training data.
    print('Start loading testing data:
    for index, test_pickle_file in enumerate(test_pickle_files):
      all_test_data[0] = pickle_batch_size = [index + 1] * pickle_batch_size,
      all_test_labels[0] = pickle_batch_size * [index + 1] * pickle_batch_size,
      load_cifar_10_pickle(test_pickle_file, image_depth)
    print('Finished loading testing data:
    return all_train_data, all_train_labels, all_test_data, all_test_labels

load_cifar_10_pickle
```
def pickle_cifar_10(all_train_data, all_train_labels, all_test_data, all_test_labels, 
train_size, valid_size, test_size, output_file_path, FORCE=False):

    if os.path.isfile(output_file_path) and not FORCE:
        print("Pickle file: %s already exist" % output_file_path)

        with open(output_file_path, "rb") as f:
            save = pickle.load(f)
            train_dataset = save["train_dataset"]
            train_labels = save["train_labels"]
            valid_dataset = save["valid_dataset"]
            valid_labels = save["valid_labels"]
            test_dataset = save["test_dataset"]
            test_labels = save["test_labels"]

            del save  # hint to help gc free up memory

        print("Validation set", valid_dataset.shape, valid_labels.shape)
        print("Test set", test_dataset.shape, test_labels.shape)

    return train_dataset, train_labels, valid_dataset, valid_labels, test_dataset, test_labels

else:
    train_dataset = all_train_data[0:train_size]
    train_labels = all_train_labels[0:train_size]
    valid_dataset = all_train_data[train_size:train_size+valid_size]
    valid_labels = all_train_labels[train_size:train_size+valid_size]
    test_dataset = all_test_data[0:test_size]
    test_labels = all_test_labels[0:test_size]

    try:
        f = open(output_file_path, "wb")
        save = {
            "train_dataset": train_dataset,
            "train_labels": train_labels,
            "valid_dataset": valid_dataset,
            "valid_labels": valid_labels,
            "test_dataset": test_dataset,
            "test_labels": test_labels,
        }

        pickle.dump(save, f, pickle.HIGHEST_PROTOCOL)
        f.close()

    except Exception as e:
        print("Unable to save data to", output_file_path, ":", e)
        raise

    statinfo = os.stat(output_file_path)
    print("Compressed pickle size": statinfo.st_size)

    return train_dataset, train_labels, valid_dataset, valid_labels, test_dataset, test_labels
if (evaluateFile is not None):
    image = (ndimage.imread(evaluateFile).astype(float) - 255 / 2) / 255
    image = image.reshape((image_size, image_size, num_channels)).astype(np.float32)
    random_data = np.ndarray((1, image_size, image_size, num_channels), dtype=np.float32)
    random_data[0, :, :, :] = image

    feed_dict = {tf_random_dataset: random_data}
    output = session.run(
        [random_prediction], feed_dict=feed_dict)

    for i, smx in enumerate(output):
        prediction = smx[0].argmax(axis=0)
        print 'The prediction is: %d' % (prediction)
Chapter 05: Sequence to Sequence Models—Parlez-vous?

def prepare_wmt_dataset(tokenizer=None):
    vocab_size = 40000

    # URLs for WMT data.
    _WMT_ENFR_DEV_URL = "http://www.statmt.org/wmt15/dev-v2.tgz"

    # Expected number of bytes for the above two file downloads
    _WMT_ENFR_TRAIN_SIZE = 2595002728
    _WMT_ENFR_DEV_SIZE = 23993583

    train_file_path = download_file(_WMT_ENFR_TRAIN_URL,
                                    os.path.realpath('..datasets/WMT'), _WMT_ENFR_TRAIN_SIZE)

    dev_file_path = download_file(_WMT_ENFR_DEV_URL,
                                   os.path.realpath('..datasets/WMT'), _WMT_ENFR_DEV_SIZE)

    train_extracted_folder = extract_file(train_file_path, os.path.realpath('..datasets/WMT/train'), IS_SUB=False)
    dev_extracted_folder = extract_file(dev_file_path, os.path.realpath('..datasets/WMT'), IS_SUB=False, FORCE=True)

    train_sub_gzip_files = train_extracted_folder + '/x for x in train_sub_gzip_files]

    if not os.path.exists(train_extracted_folder + '/data/tar.gz'): os.makedirs(train_extracted_folder + '/data/tar.gz')

    train_sub_extracted_files = [None] * 2
    for index, train_sub_gzip_file in enumerate(train_sub_gzip_files):
        train_sub_extracted_files[index] = extract_file(train_sub_gzip_file, train_extracted_folder + '/data/\' + train_sub_gzip_file.split('/')[-1].split('.')[0], TYPE="gz", IS_SUB=False)
```python
vocab_paths = [None] * 2
token_paths = [None] * 2

# Create vocabularies of the appropriate sizes and tokenizing the vocabulary
for index, train_sub_extracted_file in enumerate(train_sub_extracted_files):
    type = train_sub_extracted_file.split('.')[1][-1]
    vocab_paths[index] = "%s%s.%s" % (train_extracted_folder + '/data/', 'vocab%d' % vocab_size, type)
    token_paths[index] = "%s%s.%s" % (train_extracted_folder + '/data/', 'token%d' % vocab_size, type)
    create_vocabulary(vocab_paths[index], train_sub_extracted_files[index], vocab_size, tokenizer)
    data_to_token_ids(train_sub_extracted_files[index], token_paths[index], vocab_paths[index], tokenizer)

dev_sub_required_files = ['dev/newstest2013.fr', 'dev/newstest2013.en']
dev_sub_required_files = [dev_extracted_folder + '/' + x for x in dev_sub_required_files]

if not os.path.exists(dev_extracted_folder + '/dev/data'):
    os.makedirs(dev_extracted_folder + '/dev/data')

dev_token_paths = [None] * 2
for index, dev_sub_required_file in enumerate(dev_sub_required_files):
    type = dev_sub_required_file.split('.')[1][-1]
    dev_token_paths[index] = "%s%s.%s" % (dev_extracted_folder + '/dev/data/', 'token%d' % vocab_size, type)
    data_to_token_ids(dev_sub_required_files[index], dev_token_paths[index], vocab_paths[index], tokenizer)

def wmt(): pass
```
```python
def create_vocabulary(vocabulary_path, data_path, max_vocabulary_size, 
    tokenizer=None, normalize_digits=True):
    if not gfile.Exists(vocabulary_path):
        print("Creating vocabulary %s from data %s" % (vocabulary_path, data_path))
        vocab = {}
        with gfile.GFile(data_path, mode="rb") as f:
            counter = 0
            for line in f:
                counter += 1
                if counter % 100000 == 0:
                    print("  processing line %d (" % counter)
                tokens = tokenizer(line) if tokenizer else basic_tokenizer(line)
                for w in tokens:
                    word = re.sub(_DIGIT_RE, b"\0", w) if normalize_digits else w
                    if word in vocab:
                        vocab[word] += 1
                    else:
                        vocab[word] = 1
                vocab_list = _START_VOCAB + sorted(vocab, key=vocab.get, reverse=True)
                if len(vocab_list) > max_vocabulary_size:
                    vocab_list = vocab_list[:max_vocabulary_size]
        with gfile.GFile(vocabulary_path, mode="wb") as vocab_file:
            for w in vocab_list:
                vocab_file.write(w + b"\n")

def sentence_to_token_ids(sentence, vocabulary, 
    tokenizer=None, normalize_digits=True):
    if tokenizer:
        words = tokenizer(sentence)
    else:
        words = basic_tokenizer(sentence)
    if not normalize_digits:
        return [vocabulary.get(w, UNK_ID) for w in words]
    # Normalize digits by 0 before looking words up in the vocabulary.
    return [vocabulary.get(re.sub(_DIGIT_RE, b"\0", w), UNK_ID) for w in words]
```
def data_to_token_ids(data_path, target_path, vocabulary_path, 
                     tokenizer=None, normalize_digits=True):
    if not gfile.Exists(target_path):
        print("Tokenizing data in %s" % data_path)
        vocab, _ = initialize_vocabulary(vocabulary_path)
        with gfile.GFile(data_path, mode="rb") as data_file:
            with gfile.GFile(target_path, mode="w") as tokens_file:
                counter = 0
                for line in data_file:
                    counter += 1
                    if counter % 100000 == 0:
                        print(" tokenizing line %d" % counter)
                    token_ids = sentence_to_token_ids(line, vocab, tokenizer,
                                                        normalize_digits)
                    tokens_file.write(" ".join([str(tok) for tok in token_ids]) + ",

# def train():
# wmt = data_utils.prepare_wmt_dataset()
# en_train, fr_train, en_dev, fr_dev, _, _ = data_utils.prepare_wmt_dataset()

with tf.Session() as sess:
    # Create model.
    print("Creating %d layers of %d units." % (FLAGS.num_layers, FLAGS.size))
    model = create_model(sess, False)

    # Read data into buckets and compute their sizes.
    print("Reading development and training data (limit: %d)." %
          FLAGS.max_train_data_size)
    dev_set = read_data(wmt.en_dev_ids_path, wmt.fr_dev_ids_path)
    train_set = read_data(wmt.en_train_ids_path, wmt.fr_train_ids_path, FLAGS.max_train_data_size)
    train_bucket_sizes = [len(train_set[b]) for b in xrange(len(train_buckets_size))]
    train_total_size = sum(train_bucket_sizes)

    # A bucket scale is a list of increasing numbers from 0 to 1 that we'll use
    # to select a bucket. Length of [scale[1], scale[1+1]] is proportional to
    # the size if i th training bucket, as used later.
    train_buckets_scale = [sum(train_bucket_sizes[:i + 1]) / train_total_size for i in xrange(len(train_bucket_sizes))]
step_time, loss = 0.0, 0.0
current_step = 0
previous_losses = []

while True:
    # Choose a bucket according to data distribution. We pick a random number
    # in [0, 1] and use the corresponding interval in train_buckets_scale.
    random_number_01 = np.random.random_sample()
    bucket_id = min([i for i in range(len(train_buckets_scale))
                     if train_buckets_scale[i] > random_number_01])

    # Get a batch and make a step.
    start_time = time.time()
    encoder_inputs, decoder_inputs, target_weights = model.get_batch()
    _, step_loss, _ = model.step(sess, encoder_inputs, decoder_inputs,
                                target_weights, bucket_id, False)
    step_time += (time.time() - start_time) / FLAGS.steps_per_checkpoint
    loss += step_loss / FLAGS.steps_per_checkpoint
    current_step += 1

    # Once in a while, we save checkpoint, print statistics, and run evals.
    if current_step % FLAGS.steps_per_checkpoint == 0:
        # Print statistics for the previous epoch.
        perplexity = math.exp(loss) if loss < 300 else float('inf')
        print('global step %d learning rate %.4f step-time %.2f perplexity %.2f' %
              (model.global_step.eval(), model.learning_rate.eval(),
               step_time, perplexity))
        # Decrease learning rate if no improvement was seen over last 3 times.
        if len(previous_losses) > 2 and loss > max(previous_losses[-3:]):
            sess.run(model.learning_rate_decay_op)
        previous_losses.append(loss)
        # Save checkpoint and zero timer and loss.
        checkpoint_path = os.path.join(FLAGS.train_dir, "translate.ckpt")
        model.saver.save(sess, checkpoint_path, global_step=model.global_step)
        step_time, loss = 0.0, 0.0

    # Run evals on development set and print their perplexity.
    for bucket_id in range(len(buckets)):
        if len(dev_set[bucket_id]) == 0:
            print("eval: empty bucket %d" % (bucket_id))
            continue
        encoder_inputs, decoder_inputs, target_weights = model.get_batch()
        _, eval_loss, _ = model.step(sess, encoder_inputs, decoder_inputs,
                                      target_weights, bucket_id, True)
        eval_ppx = math.exp(eval_loss) if eval_loss < 300 else float('inf')
        print("eval: bucket %d perplexity %.2f" % (bucket_id, eval_ppx))
sys.stdout.flush()
class Seq2SeqModel(object):
    def __init__(self, source_vocab_size, target_vocab_size, buckets, size,
                 num_layers, max_gradient_norm, batch_size, learning_rate,
                 learning_rate_decay_factor, use_lstm=False, num_samples=512, forward_only=False):

        self.source_vocab_size = source_vocab_size
        self.target_vocab_size = target_vocab_size
        self.buckets = buckets
        self.batch_size = batch_size
        self.learning_rate = tf.Variable(math.sqrt(learning_rate), trainable=False)
        self.learning_rate_decay_op = self.learning_rate.assign(
            self.learning_rate * learning_rate_decay_factor)
        self.global_step = tf.Variable(0, trainable=False)

        # If we use sampled softmax, we need an output projection.
        output_projection = None
        softmax_loss_function = None
        # Sampled softmax only makes sense if we sample less than vocabulary size.
        if num_samples > 0 and num_samples < self.target_vocab_size:
            with tf.device("/cpu:0"):
                w = tf.get_variable("proj_w", [size, self.target_vocab_size])
                w_t = tf.transpose(w)
                b = tf.get_variable("proj_b", [self.target_vocab_size])
                output_projection = (w, b)

        def sampled_loss(inputs, labels):
            with tf.device("/cpu:0"):
                labels = tf.reshape(labels, [-1, 1])
                return tf.nn.sampled_softmax_loss(w_t, b, inputs, labels, num_samples,
                                                  self.target_vocab_size)

        softmax_loss_function = sampled_loss

        # Create the internal multi-layer cell for our RNN.
        single_cell = tf.nn.rnn_cell.GRUCell(size)
        if use_lstm:
            single_cell = tf.nn.rnn_cell.BasicLSTMCell(size)
        cell = single_cell
        if num_layers > 1:
            cell = tf.nn.rnn_cell.MultiRNNCell([single_cell] * num_layers)

        # The seq2seq function: we use embedding for the input and attention.
        def seq2seq(encoder_inputs, decoder_inputs, do_decode):
            return tf.nn.seq2seq.embedding_attention_seq2seq(
                encoder_inputs, decoder_inputs, cell,
                num_encoder_symbols=source_vocab_size,
                num_decoder_symbols=target_vocab_size,
                embedding_size=size,
                output_projection=output_projection,
                feed_previous=do_decode)
# Feeds for inputs.
self.encoder_inputs = []
self.decoder_inputs = []
self.target_weights = []
for i in xrange(buckets[-1][1]):  # Last bucket is the biggest one.
    self.encoder_inputs.append(tf.placeholder(tf.int32, shape=[None],
name="encoder(%d)"%i))
for i in xrange(buckets[-1][1]+1):
    self.decoder_inputs.append(tf.placeholder(tf.int32, shape=[None],
name="decoder(%d)"%i))
self.target_weights.append(tf.placeholder(tf.float32, shape=[None],
name="weight(%d)"%i))

# Our targets are decoder inputs shifted by one.
targets = [self.decoder_inputs[i-1]
    for i in xrange(len(self.decoder_inputs)-1)]

# Training outputs and losses.
if forward_only:
    self.outputs, self.losses = tf.nn.seq2seq.model_with_buckets(
        self.encoder_inputs, self.decoder_inputs, targets,
        self.target_weights, buckets, lambda x, y: seq2seq_f(x, y, True),
        softmax_loss_function=softmax_loss_function)
else:
    self.outputs, self.losses = tf.nn.seq2seq.model_with_buckets(
        self.encoder_inputs, self.decoder_inputs, targets,
        self.target_weights, buckets,
        lambda x, y: seq2seq_f(x, y, False),
        softmax_loss_function=softmax_loss_function)

# Training graph that applies the update operation for training the model.
params = tf.trainable_variables()
if not forward_only:
    self.gradient_norms = []
    self.updates = []
    opt = tf.train.GradientDescentOptimizer(self.learning_rate)
    for b in xrange(len(buckets)):
        gradients = tf.gradients(self.losses[b], params)
        clipped_gradients, norm = tf.clip_by_global_norm(gradients,
            max_gradient_norm)
        self.gradient_norms.append(norm)
        self.updates.append(opt.apply_gradients(
            zip(clipped_gradients, params), global_step=self.global_step))

self.saver = tf.train.Saver(tf.all_variables())
def step(self, session, encoder_inputs, decoder_inputs, target_weights, bucket_id, forward_only):

    encoder_size, decoder_size = self.buckets[bucket_id]
    if len(encoder_inputs) != encoder_size:
        raise ValueError("Encoder length must be equal to the one in bucket,"
                         " \%d \!\= \%d." % (len(encoder_inputs), encoder_size))
    if len(decoder_inputs) != decoder_size:
        raise ValueError("Decoder length must be equal to the one in bucket,"
                         " \%d \!\= \%d." % (len(decoder_inputs), decoder_size))
    if len(target_weights) != decoder_size:
        raise ValueError("Weights length must be equal to the one in bucket,"
                         " \%d \!\= \%d." % (len(target_weights), decoder_size))

    # Input feed: encoder inputs, decoder inputs, target_weights, as provided.
    input_feed = {}
    for i in xrange(encoder_size):
        input_feed[self.encoder_inputs[i].name] = encoder_inputs[i]
    for i in xrange(decoder_size):
        input_feed[self.decoder_inputs[i].name] = decoder_inputs[i]
        input_feed[self.target_weights[i].name] = target_weights[i]

    # Since our targets are decoder inputs shifted by one, we need one more.
    last_target = self.decoder_inputs[decoder_size].name
    input_feed[last_target] = np.zeros((self.batch_size, self.size), dtype=np.int32)

    # Output feed: depends on whether we do a backward step or not.
    if not forward_only:
        output_feed = [self.updates[bucket_id],  # Update Op that does SGD.
                       self.gradient_norms[bucket_id],  # Gradient norm.
                       self.losses[bucket_id]]  # Loss for this batch.
    else:
        output_feed = [self.losses[bucket_id]]  # Loss for this batch.

    outputs = session.run(output_feed, input_feed)

    if not forward_only:
        return outputs[1], outputs[2], None  # Gradient norm, loss, no outputs.
    else:
        return None, outputs[0], outputs[1:]  # No gradient norm, loss, outputs.
```python
def get_batch(self, data, bucket_id):
    encoder_size, decoder_size = self.buckets[bucket_id]
    encoder_inputs, decoder_inputs = [], []

    # Get a random batch of encoder and decoder inputs from data, 
    # pad them if needed, reverse encoder inputs and add GO to decoder.
    for _ in xrange(self.batch_size):
        encoder_input, decoder_input = random.choice(data[bucket_id])

        # Encoder inputs are padded and then reversed.
        encoder_pad = [data_utils.PAD_ID] * (encoder_size - len(encoder_input))
        encoder_inputs.append(list(reversed(encoder_input + encoder_pad)))

        # Decoder inputs get an extra "GO" symbol, and are padded then.
        decoder_pad_size = decoder_size - len(decoder_input) - 1
        decoder_inputs.append([data_utils.GO_ID] + decoder_input +
                               [data_utils.PAD_ID] * decoder_pad_size)

    # Now we create batch-major vectors from the data selected above.
    batch_encoder_inputs, batch_decoder_inputs, batch_weights = [], [], []

    # Batch encoder inputs are just re-indexed encoder_inputs.
    for length_idx in xrange(encoder_size):
        batch_encoder_inputs.append(
            np.array([encoder_inputs[batch_idx][length_idx]
                      for batch_idx in xrange(self.batch_size)], dtype=np.int32))

    # Batch decoder inputs are re-indexed decoder_inputs, we create weights.
    for length_idx in xrange(decoder_size):
        batch_decoder_inputs.append(
            np.array([decoder_inputs[batch_idx][length_idx]
                      for batch_idx in xrange(self.batch_size)], dtype=np.int32))

    # Create target_weights to be 0 for targets that are padding.
    batch_weight = np.ones((self.batch_size, decoder_size), dtype=np.float32)
    for batch_idx in xrange(self.batch_size):
        if length_idx < decoder_size - 1:
            target = decoder_inputs[batch_idx][length_idx + 1]
            if length_idx == decoder_size - 1 or target == data_utils.PAD_ID:
                batch_weight[batch_idx] = 0.0
        batch_weights.append(batch_weight)

    return batch_encoder_inputs, batch_decoder_inputs, batch_weights
```

Chapter 06: Finding Meaning

```python
import tensorflow as tf
import numpy as np
import os
import time
import datetime
import data_helpers
from text_cnn import TextCNN

# Parameters
#
# Model Hyperparameters
tf.flags.DEFINE_integer("embedding_dim", 128, "Dimensionality of character embedding (default: 128)"
) tf.flags.DEFINE_string("filter_sizes", "3,4,5", "Comma-separated filter sizes (default: '3,4,5')"
) tf.flags.DEFINE_integer("num_filters", 128, "Number of filters per filter size (default: 128)"
) tf.flags.DEFINE_float("dropout_keep_prob", 0.5, "Dropout keep probability (default: 0.5)"
) tf.flags.DEFINE_float("l2_reg_lambda", 0.0, "L2 regularization lambda (default: 0.0)"
)

# Training parameters
tf.flags.DEFINE_integer("batch_size", 64, "Batch Size (default: 64)"
) tf.flags.DEFINE_integer("num_epochs", 200, "Number of training epochs (default: 200)"
) tf.flags.DEFINE_integer("evaluate_every", 100, "Evaluate model on dev set after this many steps (default: 100)"
) tf.flags.DEFINE_integer("checkpoint_every", 100, "Save model after this many steps (default: 100)"
)
# Misc Parameters
tf.flags.DEFINE_boolean("allow_soft_placement", True, "Allow device soft device placement")
tf.flags.DEFINE_boolean("log_device_placement", False, "Log placement of ops on devices")

FLAGS = tf.flags.FLAGS
FLAGS._parse_flags()
print("Parameters:")
for attr, value in sorted(FLAGS.__flags.items()):
    print("{}":{}).format(attr.upper(), value)
print("")
# Data Preparation
# -----------------------------------------------

# Load data
print("Loading data...")
x_text, y = data_helpers.load_data_and_labels(FLAGS.positive_data_file, FLAGS.negative_data_file)

# Build vocabulary
max_document_length = max([len(x.split(" ")) for x in x_text])
vocab_processor = learn.preprocessing.VocabularyProcessor(max_document_length)
x = np.array(list(vocab_processor.fit_transform(x_text)))

# Randomly shuffle data
np.random.seed(10)
shuffle_indices = np.random.permutation(np.arange(len(y)))
x_shuffled = x[shuffle_indices]
y_shuffled = y[shuffle_indices]

# Split train/test set
# TODO: This is very crude, should use cross-validation
dev_sample_index = -1 * int(FLAGS.dev_sample_percentage * float(len(y)))
x_train, x_dev = x_shuffled[:dev_sample_index], x_shuffled[dev_sample_index:]
y_train, y_dev = y_shuffled[:dev_sample_index], y_shuffled[dev_sample_index:]
print("Vocabulary Size: {}".format(len(vocab_processor.vocabulary_)))
print("Train/Dev split: {} / {}".format(len(y_train), len(y_dev)))
# Training
#
# =================================================

with tf.Graph().as_default():
    session_conf = tf.ConfigProto(
        allow_soft_placement=FLAGS.allow_soft_placement,
        log_device_placement=FLAGS.log_device_placement)
    sess = tf.Session(config=session_conf)
    with sess.as_default():
        cnn = TextCNN(
            sequence_length=x_train.shape[1],
            num_classes=2,
            vocab_size=len(vocabulary),
            embedding_size=FLAGS.embedding_dim,
            filter_sizes=list(map(int, FLAGS.filter_sizes.split(','))),
            num_filters=FLAGS.num_filters,
            l2_reg_lambda=FLAGS.l2_reg_lambda)

    # Define Training procedure
    global_step = tf.Variable(0, name="global_step", trainable=False)
    optimizer = tf.train.AdamOptimizer(1e-3)
    grads_and_vars = optimizer.compute_gradients(cnn.loss)
    train_op = optimizer.apply_gradients(grads_and_vars, global_step=global_step)

    # Keep track of gradient values and sparsity (optional)
    grad_summaries = []
    for g, v in grads_and_vars:
        if g is not None:
            grad_hist_summary = tf.histogram_summary("{}/grad/hist".format(v.name), g)
            sparsity_summary = tf.scalar_summary("{}/grad/sparsity".format(v.name), tf.nn.zero_fraction(g))
            grad_summaries.append(grad_hist_summary)
            grad_summaries.append(sparsity_summary)
    grad_summaries_merged = tf.merge_summary(grad_summaries)

    # Output directory for models and summaries
    timestamp = str(int(time.time()))
    out_dir = os.path.abspath(os.path.join(os.path.curdir, "runs", timestamp))
    print("writing to {}\n".format(out_dir))
# Summaries for loss and accuracy

```python
tf.scalar_summary("loss", cnn.loss)
```

```python
tf.scalar_summary("accuracy", cnn.accuracy)
```

# Train Summaries

```python
tf.merge_summary([loss_summary, acc_summary, grad_summaries_merged])
tf.train.SummaryWriter(train_summary_dir, sess.graph_def)
```

# Dev summaries

```python
tf.merge_summary([loss_summary, acc_summary])
tf.train.SummaryWriter(dev_summary_dir, sess.graph_def)
```

# Checkpoint directory. Tensorflow assumes this directory already exists so we need to create it

```python
tf.train.Saver(tf.all_variables())
```python
def train_step(x_batch, y_batch):
    # A single training step
    feed_dict = {
        cnn.input_x: x_batch,
        cnn.input_y: y_batch,
        cnn.dropout_keep_prob: FLAGS.dropout_keep_prob
    }
    _, step, summary_str, loss, accuracy = sess.run(
        [train_op, global_step, train_summary_op, cnn.loss, cnn.accuracy],
        feed_dict=feed_dict
    )
    time_str = str(datetime.datetime.now()).isoformat()
    print('[%s]: step (%d), loss (%g), acc (%g)' % (time_str, step, loss, accuracy))
    train_summary_writer.add_summary(summary_str, step)

def dev_step(x_batch, y_batch, writer=None):
    # Evaluates model on a dev set
    feed_dict = {
        cnn.input_x: x_batch,
        cnn.input_y: y_batch,
        cnn.dropout_keep_prob: 1.0
    }
    step, summary_str, loss, accuracy = sess.run(
        [global_step, dev_summary_op, cnn.loss, cnn.accuracy],
        feed_dict=feed_dict
    )
    time_str = str(datetime.datetime.now()).isoformat()
    print('[%s]: step (%d), loss (%g), acc (%g)' % (time_str, step, loss, accuracy))
    if writer:
        writer.add_summary(summary_str, step)

# Generate batches
batches = data_helpers.batch_iter(
    list(zip(x_train, y_train)), FLAGS.batch_size, FLAGS.num_epochs)
# Training loop. For each batch...
for batch in batches:
    x_batch, y_batch = zip(*batch)
    train_step(x_batch, y_batch)
    current_step = tf.train.global_step(sess, global_step)
    if current_step % FLAGS.evaluate_every == 0:
        print('Evaluation:
')
        dev_step(x_dev, y_dev, writer=dev_summary_writer)
        print('')
    if current_step % FLAGS.checkpoint_every == 0:
        path = saver.save(sess, checkpoint_prefix, global_step=current_step)
        print('Saved model checkpoint to %s
' % path)
```

```python
#!/usr/bin/env python

import tensorflow as tf
import numpy as np
import os
import time
import datetime
import data_helpers
from text_cnn import TextCNN

# Parameters
# ---------------------------------------------

# Eval Parameters
tf.flags.DEFINE_integer("batch_size", 64, "Batch Size (default: 64)")
tf.flags.DEFINE_string("checkpoint_dir", ", "Checkpoint directory from training run"

# Misc Parameters
tf.flags.DEFINE_boolean("allow_soft_placement", True, "Allow device soft device placement")
tf.flags.DEFINE_boolean("log_device_placement", False, "Log placement of ops on devices")

FLAGS = tf.flags.FLAGS
FLAGS._parse_flags()
print("\nParameters:")
for attr, value in sorted(FLAGS._flags.items()):
    print("{}={}").format(attr.upper(), value)
print("

# Load data. Load your own data here
print("Loading data...")
x_test, y_test, vocabulary, vocabulary_inv = data_helpers.load_data()
y_test = np.argmax(y_test, axis=1)
print("Vocabulary size: {:d}".format(len(vocabulary)))
print("Test set size {:d}".format(len(y_test)))

print("\nEvaluating...\n")
```

print("\n\nEvaluating...\n")

# Evaluation
# ---------------------------------
checkpoint_file = tf.train.latest_checkpoint(FLAGS.checkpoint_dir)
graph = tf.Graph()
with graph.as_default():
    session_conf = tf.ConfigProto(
        allow_soft_placement=FLAGS.allow_soft_placement,
        log_device_placement=FLAGS.log_device_placement)
sess = tf.Session(config=session_conf)
with sess.as_default():
    # Load the saved meta graph and restore variables
    saver = tf.train.import_meta_graph("{}.meta").format(checkpoint_file))
saver.restore(sess, checkpoint_file)

    # Get the placeholders from the graph by name
    input_x = graph.get_operation_by_name("input_x").outputs[0]
    # input_y = graph.get_operation_by_name("input_y"). outputs[0]
    dropout_keep_prob = graph.get_operation_by_name("dropout_keep_prob").outputs[0]

    # Tensors we want to evaluate
    predictions = graph.get_operation_by_name("output/predictions").outputs[0]

    # Generate batches for one epoch
    batches = data_helpers.batch_iter(x_test, FLAGS.batch_size, 1, shuffle=False)

    # Collect the predictions here
    all_predictions = []
    for x_test_batch in batches:
        batch_predictions = sess.run(predictions, {input_x: x_test_batch, dropout_keep_prob: 1.0})
        all_predictions = np.concatenate([all_predictions, batch_predictions])

    # Print accuracy
    correct_predictions = float(sum(all_predictions == y_test))
    print("Total number of test examples: {}".format(len(y_test)))
    print("Accuracy: {:.g}".format(correct_predictions/float(len(y_test)))))
Chapter 07: Making Money with Machine Learning

<table>
<thead>
<tr>
<th>DATE</th>
<th>OPEN</th>
<th>HIGH</th>
<th>LOW</th>
<th>CLOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1999-11-18</td>
<td>45.5</td>
<td>50</td>
<td>40</td>
<td>44</td>
</tr>
<tr>
<td>1999-11-19</td>
<td>42.94</td>
<td>43</td>
<td>39.81</td>
<td>40.38</td>
</tr>
<tr>
<td>1999-11-22</td>
<td>41.31</td>
<td>44</td>
<td>40.06</td>
<td>44</td>
</tr>
<tr>
<td>1999-11-23</td>
<td>42.5</td>
<td>43.63</td>
<td>40.25</td>
<td>40.25</td>
</tr>
<tr>
<td>1999-11-24</td>
<td>40.13</td>
<td>41.04</td>
<td>40</td>
<td>41.06</td>
</tr>
<tr>
<td>1999-11-26</td>
<td>40.88</td>
<td>41.5</td>
<td>40.75</td>
<td>41.19</td>
</tr>
<tr>
<td>1999-11-29</td>
<td>41</td>
<td>42.44</td>
<td>40.56</td>
<td>42.13</td>
</tr>
<tr>
<td>1999-11-30</td>
<td>42</td>
<td>42.94</td>
<td>40.04</td>
<td>42.19</td>
</tr>
<tr>
<td>1999-12-01</td>
<td>42.19</td>
<td>43.44</td>
<td>41.88</td>
<td>42.94</td>
</tr>
<tr>
<td>1999-12-02</td>
<td>43.75</td>
<td>45</td>
<td>43.19</td>
<td>44.13</td>
</tr>
<tr>
<td>1999-12-03</td>
<td>44.04</td>
<td>45.60</td>
<td>44.21</td>
<td>44.5</td>
</tr>
<tr>
<td>1999-12-06</td>
<td>45.25</td>
<td>46.44</td>
<td>45.19</td>
<td>45.75</td>
</tr>
<tr>
<td>1999-12-07</td>
<td>45.75</td>
<td>46</td>
<td>44.31</td>
<td>45.25</td>
</tr>
<tr>
<td>1999-12-08</td>
<td>45.25</td>
<td>45.63</td>
<td>44.81</td>
<td>45.19</td>
</tr>
<tr>
<td>1999-12-09</td>
<td>45.25</td>
<td>45.94</td>
<td>45.25</td>
<td>45.81</td>
</tr>
<tr>
<td>1999-12-10</td>
<td>45.09</td>
<td>45.94</td>
<td>44.75</td>
<td>44.75</td>
</tr>
<tr>
<td>1999-12-13</td>
<td>45.5</td>
<td>46.25</td>
<td>44.38</td>
<td>45.5</td>
</tr>
<tr>
<td>1999-12-14</td>
<td>45.38</td>
<td>45.38</td>
<td>42.06</td>
<td>43</td>
</tr>
<tr>
<td>1999-12-15</td>
<td>42</td>
<td>42.31</td>
<td>41</td>
<td>41.09</td>
</tr>
<tr>
<td>1999-12-16</td>
<td>42</td>
<td>48</td>
<td>42</td>
<td>47.25</td>
</tr>
<tr>
<td>1999-12-17</td>
<td>46.38</td>
<td>47.12</td>
<td>45.64</td>
<td>45.84</td>
</tr>
</tbody>
</table>
Chapter 08: The Doctor Will See You Now
fileInputDICOM = r.repop(redisqueue)
newFile = fileInputDICOM.replace(ON_DIR, OUT_DIR) + "_.png"
print(num, ",", fileInputDICOM)
plan = dicomio.read_file(fileInputDICOM)
shape = plan.pixel_array.shape
wBuffer = MAX_SIZE - shape[0]
buffer = MAX_SIZE - shape[1]
image_2d = []
for row in plan.pixel_array:
    pixels = []
    for col in row:
        pixels.append(col)
    for h in range(wBuffer):
        pixels.append(255)
    image_2d.append(pixels)
for w in range(wBuffer):
    image_2d.append([255]*MAX_SIZE)

# Rescaling greyscale between 0-255
image_2d_scaled = []
for row in image_2d:
    row_scaled = []
    for col in row:
        col_scaled = int((float(col)/float(max_val))*255.0)
        col_scaled = 255.0 - col_scaled
        row_scaled.append(col_scaled)
    image_2d_scaled.append(row_scaled)

if not os.path.exists(os.path.dirname(newFile)):
    try:
        os.makedirs(os.path.dirname(newFile))
    except OSError as exc:
        # Guard against race condition
        if exc.errno !=errno.EXIST:
            raise

f = open(newFile, 'wb')
w = png.Writer(MAX_SIZE, MAX_SIZE, greyscale=True)
w.write(f, image_2d_scaled)
f.close()}
fileInputDICOM = r.rpop(redisQueue)
newFile = fileInputDICOM.replace(IN_DIR, OUT_DIR) + ".png"
print(num, ",", fileInputDICOM)
shape = plan.pixel_array.shape
wBuffer=MAX_SIZE-shape[0]
hBuffer=MAX_SIZE-shape[1]
image_2d = []
for row in plan.pixel_array:
    pixels = []
    for col in row:
        pixels.append(col)
    for h in range(hBuffer):
        pixels.append(32767)
        image_2d.append(pixels)
    for w in range(wBuffer):
        image_2d.append([32767]*MAX_SIZE)

# Rescaling greyscale between 0-255
image_2d_scaled = []
for row in image_2d:
    row_scaled = []
    for col in row:
        col_scaled = int((float(col)/float(max_val))*255.0)
        col_scaled = 255.0 - col_scaled
        row_scaled.append(col_scaled)
    image_2d_scaled.append(row_scaled)

if not os.path.exists(os.path.dirname(newFile)):
    try:
        os.makedirs(os.path.dirname(newFile))
    except OSError as exc: # Guard against race condition
        if exc.errno != errno.EEXIST:
            raise

f = open(newFile, 'wb')
w = png.Writer(MAX_SIZE, MAX_SIZE, greyscale=True)
w.write(f, image_2d_scaled)
f.close()
Chapter 09: Cruise Control – Automation
Chapter 10: Go Live and Go Big
## Per Instance Prices & Projected Costs (all in USD)

<table>
<thead>
<tr>
<th>Select</th>
<th>Name</th>
<th>Upfront Price</th>
<th>Effective Hourly Cost</th>
<th>Effective Monthly Cost</th>
<th>1 Year Cost</th>
<th>3 Year Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>On-Demand (No Contract)</td>
<td>-</td>
<td>-</td>
<td>0.00</td>
<td>-</td>
<td>658.89</td>
<td>7905.60</td>
</tr>
<tr>
<td>1 Yr No Upfront Reserved</td>
<td>6.00</td>
<td>0.614</td>
<td>448.22</td>
<td>5378.64</td>
<td>16135.92</td>
<td></td>
</tr>
<tr>
<td>1 Yr Partial Upfront Reserved</td>
<td>2562.00</td>
<td>0.585</td>
<td>427.92</td>
<td>5124.24</td>
<td>15372.72</td>
<td></td>
</tr>
<tr>
<td>1 Yr All Upfront Reserved</td>
<td>5822.00</td>
<td>0.573</td>
<td>418.59</td>
<td>5022.00</td>
<td>15866.09</td>
<td></td>
</tr>
<tr>
<td>3 Yr Partial Upfront Reserved</td>
<td>5584.00</td>
<td>0.425</td>
<td>310.24</td>
<td>-</td>
<td>-</td>
<td>11168.32</td>
</tr>
<tr>
<td>3 Yr All Upfront Reserved</td>
<td>18499.00</td>
<td>0.399</td>
<td>291.64</td>
<td>-</td>
<td>-</td>
<td>18469.09</td>
</tr>
<tr>
<td>3 Yr No Upfront Convertible</td>
<td>6.00</td>
<td>0.528</td>
<td>385.44</td>
<td>-</td>
<td>-</td>
<td>13875.84</td>
</tr>
<tr>
<td>3 Yr Partial Upfront Convertible</td>
<td>6422.00</td>
<td>0.488</td>
<td>356.51</td>
<td>-</td>
<td>-</td>
<td>12824.32</td>
</tr>
<tr>
<td>3 Yr All Upfront Convertible</td>
<td>12588.00</td>
<td>0.479</td>
<td>349.67</td>
<td>-</td>
<td>-</td>
<td>12588.00</td>
</tr>
</tbody>
</table>

### train/accuracy

![Graph of train/accuracy](image)

### train/cross_entropy_loss

![Graph of train/cross_entropy_loss](image)
Mechanical Turk is a marketplace for work.
We give businesses and developers access to an on-demand, scalable workforce.
Workers select from thousands of tasks and work whenever it's convenient.

210,950 HITs available. View them now.

Make Money
by working on HITs

HITs - Human Intelligence Tasks - are individual tasks that you work on.
Find HITs now.

As a Mechanical Turk Worker you:
- Can work from home
- Choose your own work hours
- Get paid for doing good work

Work

Find an interesting task

Earn money

Get Results
from Mechanical Turk Workers

Ask workers to complete HITs - Human Intelligence Tasks - and get results using Mechanical Turk.
Get Started.

As a Mechanical Turk Requester you:
- Have access to a global, on-demand, 24 x 7 workforce
- Get thousands of HITs completed in minutes
- Pay only when you're satisfied with the results

Fund your account
Lead your tasks
Get results

or learn more about being a Worker
Chapter 11: Going Further - 21 Problems

```python
# Concatenate the groups
output = tf.concat(output_groups, 3)

# Add the biases
output = tf.concat(3, output_groups)

# Add the bias

if biased:

@layer
def concat(self, inputs, axis, name):
    return tf.concat(values=inputs, axis=axis, name=name)
```

```python
@layer
def concat(self, inputs, axis, name):
    return tf.concat(axis=axis, values=inputs, name=name)
```
Appendix: Advanced Installation

```
libcuda1-375 - NVIDIA CUDA runtime library
nvidia-304 - NVIDIA legacy binary driver - version 304.135
nvidia-304-updates - Transitional package for nvidia-304
nvidia-304-updates-dev - Transitional package for nvidia-304-dev
nvidia-340 - NVIDIA binary driver - version 340.102
nvidia-361 - Transitional package for nvidia-367
nvidia-361-dev - Transitional package for nvidia-367-dev
nvidia-367 - Transitional package for nvidia-375
nvidia-367-dev - Transitional package for nvidia-367-dev
nvidia-375 - NVIDIA binary driver - version 375.66
nvidia-375-dev - NVIDIA binary Xorg driver development files
nvidia-libopencl1-304 - Transitional package for nvidia-libopencl1-375
nvidia-libopencl1-375 - NVIDIA OpenCL Driver and ICD Loader library
nvidia-opengl-icd-304-updates - Transitional package for nvidia-opengl-icd-304
nvidia-opengl-icd-361 - Transitional package for nvidia-opengl-icd-367
nvidia-opengl-icd-367 - Transitional package for nvidia-opengl-icd-375
nvidia-opengl-icd-375 - NVIDIA OpenCL ICD
nvidia-libopencl1-304-updates - Transitional package for nvidia-libopencl1-304
nvidia-libopencl1-361 - Transitional package for nvidia-libopencl1-367
nvidia-352 - Transitional package for nvidia-375
cuda-visual-tools-8-0 - CUDA visual tools
nvidia-361-updates - Transitional package for nvidia-375
cuda-drivers - CUDA Driver meta-package
nvidia-opengl-icd-352-updates - Transitional package for nvidia-opengl-icd-375
nvidia-libopencl1-361-updates - Transitional package for nvidia-libopencl1-375
nvidia-libopencl1-352 - Transitional package for nvidia-libopencl1-375
nvidia-opengl-icd-361-updates - Transitional package for nvidia-opengl-icd-375
nvidia-modprobe - Load the NVIDIA kernel driver and create device files
nvidia-libopencl1-352-updates - Transitional package for nvidia-libopencl1-375
nvidia-352-updates-dev - Transitional package for nvidia-375-dev
nvidia-352-dev - Transitional package for nvidia-375-dev
nvidia-361-updates-dev - Transitional package for nvidia-375-dev
nvidia-352-updates - Transitional package for nvidia-375
nvidia-opengl-icd-352 - Transitional package for nvidia-opengl-icd-375
```

sudo apt-get install nvidia-375

Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
cuda-command-line-tools-8-0 cuda-core-8-0 cuda-cublas-8-0
cuda-cublas-dev-8-0 cuda-cudart-8-0 cuda-cudart-dev-8-0 cuda-cufft-8-0
cuda-cufft-dev-8-0 cuda-cublas-dev-8-0 cuda-curand-8-0 cuda-curand-dev-8-0 cuda-cusolver-8-0
cuda-cusolver-dev-8-0 cuda-cusparse-8-0 cuda-cusparse-dev-8-0
cuda-documentation-8-0 cuda-driver-dev-8-0 cuda-license-8-0
cuda-misc-headers-8-0 cuda-npp-8-0 cuda-npp-dev-8-0 cuda-nvgraph-8-0
cuda-nvgraph-dev-8-0 cuda-nvml-dev-8-0 cuda-nvrtc-8-0 cuda-nvrtc-dev-8-0
cuda-samples-8-0 cuda-toolkit-8-0 cuda-visual-tools-8-0 libgles1-mesa
libxmu-dev libxmu-headers linux-headers-4.8.0-52
linux-headers-4.8.0-52-generic linux-headers-4.8.0-54
linux-headers-4.8.0-54-generic linux-image-4.8.0-52-generic
linux-image-4.8.0-54-generic linux-image-extra-4.8.0-52-generic
linux-image-extra-4.8.0-54-generic linux-signed-image-4.8.0-52-generic
linux-signed-image-4.8.0-54-generic nvidia-modprobe screen snap-connman

Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  libcuda-375 nvidia-opengl-icd-375
The following NEW packages will be installed:
  libcuda-375 nvidia-375 nvidia-opengl-icd-375
0 upgraded, 3 newly installed, 0 to remove and 65 not upgraded.
Need to get 75,2 MB of archives.
After this operation, 333 MB of additional disk space will be used.
Do you want to continue? [Y/n]
- Installation
  - Installing to /lib/modules/4.8.0-58-generic/updates/dkms/

nvidia_375_drm.ko:
Running module version sanity check.
  - Original module
    - No original module exists within this kernel
  - Installation
    - Installing to /lib/modules/4.8.0-58-generic/updates/dkms/

nvidia_375_uvm.ko:
Running module version sanity check.
  - Original module
    - No original module exists within this kernel
  - Installation
    - Installing to /lib/modules/4.8.0-58-generic/updates/dkms/

depmod....

DKMS: install completed.
Setting up libcudai-375 (375.66-0ubuntu0.16.04.1) ...
Setting up nvidia-opengl-icd-375 (375.66-0ubuntu0.16.04.1) ...
Processing triggers for libc-bin (2.23-0ubuntu9) ...
/sbin/ldconfig.real: /usr/local/lib/libpocketsphinx.so.3 is not a symbolic link
/sbin/ldconfig.real: /usr/lib/nvidia-375/libEGL.so.1 is not a symbolic link
/sbin/ldconfig.real: /usr/lib32/nvidia-375/libEGL.so.1 is not a symbolic link

Processing triggers for libnvidia-frontend-375 (375.66-0ubuntu0.16.04.1) ...
update-initramfs: Generating /boot/initrd.img-4.8.0-58-generic
Processing triggers for shrim-signed (1.28-16.04.1+0.9+1474479173.6c180c6-1ubuntu1) ...
Secure Boot not enabled on this system.

ubuntu@ubuntu-TITAN:~/Downloads
Select Target Platform

Click on the green buttons that describe your target platform. Only supported platforms will be shown.

Operating System
- Windows
- Linux
- Mac OSX

Refer to the Release Notes for enhancements and bug fixes in the latest version.

Get Started
End User License Agreement

Preface

The following contains specific license terms and conditions for four separate NVIDIA products. By accepting this agreement, you agree to comply with all the terms and conditions applicable to the specific product(s) included herein.

NVIDIA CUDA Toolkit

Description

The NVIDIA CUDA Toolkit provides command-line and graphical tools for building, debugging and optimizing the performance of applications accelerated by NVIDIA GPUs, runtime and math libraries, and documentation including programming guides.
Preface
-------

The following contains specific license terms and conditions for four separate NVIDIA products. By accepting this agreement, you agree to comply with all the terms and conditions applicable to the specific product(s) included herein.

NVIDIA CUDA Toolkit

Description

The NVIDIA CUDA Toolkit provides command-line and graphical tools for building, debugging and optimizing the performance of applications accelerated by NVIDIA GPUs, runtime and math libraries, and documentation including programming guides.

Do you accept the previously read EULA?

accept/decline/quit:
Default Install Location of CUDA Toolkit

Windows platform:

Do you accept the previously read EULA?
accept/decline/quit: accept

Install NVIDIA Accelerated Graphics Driver for Linux-x86_64 375.26?
(y)es/(n)o/(q)uit: n

Install the CUDA 8.0 Toolkit?
(y)es/(n)o/(q)uit: y

Enter Toolkit Location
[ default is /usr/local/cuda-8.0 ]:

Do you want to install a symbolic link at /usr/local/cuda?
(y)es/(n)o/(q)uit: y

Install the CUDA 8.0 Samples?
(y)es/(n)o/(q)uit: y

Enter CUDA Samples Location
[ default is /home/ubuntu ]:

Installing the CUDA Toolkit in /usr/local/cuda-8.0 ...
= Summary =

Driver: Not Selected
Toolkit: Installed in /usr/local/cuda-8.0
Samples: Installed in /home/ubuntu

Please make sure that
- PATH includes /usr/local/cuda-8.0/bin
- LD_LIBRARY_PATH includes /usr/local/cuda-8.0/lib64, or, add /usr/local/cuda-8.0/lib64 to /etc/ld.so.conf and run ldconfig as root

To uninstall the CUDA Toolkit, run the uninstall script in /usr/local/cuda-8.0/bin

Please see CUDA_Installation_Guide_Linux.pdf in /usr/local/cuda-8.0/doc/pdf for detailed information on setting up CUDA.

***WARNING: Incomplete installation! This installation did not install the CUDA Driver. A driver of version at least 361.00 is required for CUDA 8.0 functionality to work.
To install the driver using this installer, run the following command, replacing "<CudaInstaller>" with the name of this run file:
  sudo <CudaInstaller>.run -silent -driver

Logfile is /tmp/cuda_install_10612.log
cuDNN Download

NVIDIA cuDNN is a GPU-accelerated library of primitives for deep neural networks.

I Agree To the Terms of the cuDNN Software License Agreement
Please check your framework documentation to determine the recommended version of cuDNN.
If you are using cuDNN with a Pascal (GTx 1060, GTx 1070), version 5 or later is required.

Download cuDNN v5.0 (April 27, 2017), for CUDA 8.0
Download cuDNN v5.0 (April 27, 2017), for CUDA 7.5
Download cuDNN v5.1 (Jan 20, 2017), for CUDA 8.0
Download cuDNN v5.1 (Jan 20, 2017), for CUDA 7.5
Download cuDNN v5 (May 27, 2016), for CUDA 8.0
Download cuDNN v5 (May 12, 2016), for CUDA 7.5
Download cuDNN v4 (Feb 19, 2016), for CUDA 7.0 and later.

Archived cuDNN Releases
Download cuDNN v5.1 (Jan 20, 2017), for CUDA 8.0

cuDNN User Guide

cuDNN Install Guide

cuDNN v5.1 Library for Linux

Ubuntu@ubuntu-TITAN:~/Downloads$ tar -xf cudnn-8.0-linux-x64-v5.1.tgz
Ubuntu@ubuntu-TITAN:~/Downloads$ cd cuda
Ubuntu@ubuntu-TITAN:~/Downloads/cuda$ sudo cp -P include/cudnn.h /usr/include/
Ubuntu@ubuntu-TITAN:~/Downloads/cuda$ sudo cp -P lib64/libcudnn*.a /usr/lib/x86_64-linux-gnu/
Ubuntu@ubuntu-TITAN:~/Downloads/cuda$ sudo chmod a+r /usr/lib/x86_64-linux-gnu/libcudnn*

Installing collected packages: html5lib, bleach, markdown, backports.weakref, tensorflow-gpu

Running setup.py install for html5lib ... done
Running setup.py install for markdown ... done
Successfully installed backports.weakref-1.0rc1 bleach-1.5.0 html5lib-0.9999999 markdown-2.6.8 tensorflow-gpu-1.2.1

Ubuntu@ubuntu-TITAN:~/Downloads$
Do you approve the license terms? [yes|no]

>>> yes

Miniconda2 will now be installed into this location:
/home/ubuntu/miniconda2

- Press ENTER to confirm the location
- Press CTRL-C to abort the installation
- Or specify a different location below

Installing: yamll-0.1.6-0 ...
Installing: zlib-1.2.8-3 ...
Installing: conda-4.3.21-py27_0 ...
Installing: pip-9.0.1-py27_1 ...
Installing: wheel-0.29.0-py27_0 ...
Python 2.7.13 :: Continuum Analytics, Inc.
creating default environment...
Installation finished.
Do you wish the installer to prepend the Miniconda2 install location
to PATH in your /home/ubuntu/.bashrc? [yes|no]

[no] >>> yes

Prepending PATH=/home/ubuntu/miniconda2/bin to PATH in /home/ubuntu/.bashrc
A backup will be made to: /home/ubuntu/.bashrc-miniconda2.bak

For this change to become active, you have to open a new terminal.

Thank you for installing Miniconda2!

Share your notebooks and packages on Anaconda Cloud!
Sign up for free: https://anaconda.org

ubuntu@ubuntu-TITAN:~/github/chapter11S
name: env2
cannels:
  - https://conda.anaconda.org/nenpo
  - conda-forge
dependencies:
  - python=2.7
  - opencv3
  - numpy
  - matplotlib
  - jupyter
  - pillow
  - scikitlearn
  - scikit-image
  - scipy
  - h5py
  - eventlet
  - flask-socketio
  - seaborn
  - pandas
  - ffmpeg
  - pyqt
  - ptp:
    - mumpy
    - tensorflow-gpu
    - keras
    - prettytable

Successfully installed backportsWeakref-1.0rc1 funcslug-1.0.2 imagem-2.1.2 keras-2.0.6 markdown-2.6.8 mock-2.0.0 mطيع-0.2.3.2 pbr-3.1.1 prettytable-0.7.2 protobuf-3.3.0 tensorflow-gpu-1.2.1 theano-0.9.0 tifdir-4.11.2
#
# To activate this environment, use:
# > source activate env2
#
# To deactivate this environment, use:
# > source deactivate env2
#

ubuntu@ubuntu-TITAN:~/github/chapter11$
(env2) ubuntu@ubuntu-TITAN:~/github/chapter11$ python
Python 2.7.13 | packaged by conda-forge | (default, May 2 2017, 12:48:11)
[GCC 4.8.2 20140120 (Red Hat 4.8.2-15)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow as tf
>>> tf.Session()
2017-07-10 00:35:14.578030: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use SSE4.1 instructions, but these are available on your machine and could speed up CPU computations.
2017-07-10 00:35:14.578051: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use SSE4.2 instructions, but these are available on your machine and could speed up CPU computations.
2017-07-10 00:35:14.578055: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use AVX instructions, but these are available on your machine and could speed up CPU computations.
2017-07-10 00:35:14.578058: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use AVX2 instructions, but these are available on your machine and could speed up CPU computations.
2017-07-10 00:35:14.578061: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use FMA instructions, but these are available on your machine and could speed up CPU computations.
2017-07-10 00:35:14.710550: I tensorflow/core/common_runtime/gpu/gpu_device.cc:940] Found device 0 with properties:
with properties:
  name: GeForce GTX TITAN X
  major: 5minor: 2memoryClockRate (GHz) 1.076
  pciBusID 0000:01:00.0
  Total memory: 11.92GB
  Free memory: 11.27GB
2017-07-10 00:35:14.710563: I tensorflow/core/common_runtime/gpu/gpu_device.cc:961] DMA: 0
2017-07-10 00:35:14.710567: I tensorflow/core/common_runtime/gpu/gpu_device.cc:971] 0: Y
2017-07-10 00:35:14.710580: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1030] Creating TensorFlow device (/gpu:0) -> (device: 0, name: GeForce GTX TITAN X, pci bus id: 0000:01:00.0)
<tensorflow.python.client.session.Session object at 0x7fd35b3e4110>
name: env3
channels:
- https://conda.anaconda.org/menpo
- conda-forge
dependencies:
- python=3
- opencv
- numpy
- matplotlib
- jupyter
- pillow
- scikit-learn
- scikit-image
- scipy
- h5py
- eventlet
- flask-socketio
- seaborn
- pandas
- ffmpeg
- pyqt
- ptp:
  - moviepy
  - tensorflow-gpu
  - keras
  - prettytable
CUDA Toolkit Download

Select Target Platform

Click on the green buttons that describe your target platform. Only supported platforms will be shown.

Operating System
- Windows
- Linux
- Mac OSX

Architecture
- x86_64
- spc24le

Distribution
- Fedora
- OpenSUSE
- RHEL
- CentOS
- SLES
- Ubuntu

Version
- 16.04
- 14.04

Installer Type
- runFile (local)
- deb (local)
- deb (network)
- cluster (local)

Download Installers for Linux Ubuntu 16.04 x86_64

The base installer is available for download below.
There is 1 patch available. This patch requires the base installer to be installed first.

- **Base Installer**
  - Installation Instructions:
    1. Run `sudo sh cuda_8.0.61_975.36_linux.run`
    2. Follow the command-line prompts

- **Patch 2 [Released Jun 26, 2017]**
  - Download (95.3 MB)
  - cuBLAS Patch Update to CUDA 8: Includes performance enhancements and bug fixes

The CUDA Toolkit contains Open-Source Software. The source code can be found here. The check sums for the installer and patches can be found in Installer Checksums. For further information, see the Installation Guide for Linux and the CUDA Quick Start Guide.