TensorFlow Reading Data

Tao Chen

Shanghai LingXian Robotics

Dec 13, 2016

Outline

Outline

Feeding Mechanism

- Feeds
  - Inject data into any Tensor in a computation graph
  - Temporarily replace the output of an operation with a tensor value
- Placeholder Operation
  - Serve as the target of feeds, not initialized and contains no data
- great for small examples and easily interacting with data

Feeding Example I

- Input:

```
import tensorflow as tf

input1 = tf.placeholder(tf.float32)
input2 = tf.placeholder(tf.float32)
output = tf.mul(input1, input2)

with tf.Session() as sess:
     print(sess.run([output],
           feed_dict={input1:[7.], input2:[2.]}))
```

- Output:

```
[array([ 14.], dtype=float32)]
```

Feeding Example II - MNIST

```
def placeholder_inputs(batch_size):
    images_placeholder = tf.placeholder(tf.float32, shape=(batch_size,
                                                           mnist.IMAGE_PIXELS))
    labels_placeholder = tf.placeholder(tf.int32, shape=(batch_size))
    return images_placeholder, labels_placeholder

def fill_feed_dict(data_set, images_pl, labels_pl):
    images_feed, labels_feed = data_set.next_batch(FLAGS.batch_size,
                                                   FLAGS.fake_data)
    feed_dict = {
        images_pl: images_feed,
        labels_pl: labels_feed,
    }
    return feed_dict
```

Feeding Example II -MNIST

```
def run_training():
    data_sets = input_data.read_data_sets(FLAGS.input_data_dir,
                                          FLAGS.fake_data)
    with tf.Graph().as_default():
        images_placeholder, labels_placeholder = placeholder_inputs(
                                                    FLAGS.batch_size)
        ...
        init = tf.global_variables_initializer()
        sess = tf.Session()
        sess.run(init)
        for step in xrange(FLAGS.max_steps):
            feed_dict = fill_feed_dict(data_sets.train,
                                       images_placeholder,
                                       labels_placeholder)
            _, loss_value = sess.run([train_op, loss],
                                     feed_dict=feed_dict)
```

- Full code is available at: fully_connected_feed.py

Outline

Load Image - Using TensorFlow

```
import tensorflow as tf
import numpy as np
from PIL import Image
import glob
data_dir = '/home/chentao/Pictures/'
filename_list = glob.glob('%s*.jpg' % (data_dir))
filename_queue = tf.train.string_input_producer(filename_list)
reader = tf.WholeFileReader()
key, value = reader.read(filename_queue)
img = tf.image.decode_jpeg(value) # use png or jpg decoder based on your files.

init_op = tf.global_variables_initializer()
sess = tf.InteractiveSession()

sess.run(init_op)

coord = tf.train.Coordinator()
threads = tf.train.start_queue_runners(coord=coord, sess=sess)

for i in range(len(filename_list)):  # length of your filename list
    image = img.eval()  # here is your image Tensor
    Image.fromarray(np.asarray(image)).show()

coord.request_stop()
coord.join(threads)
```

Load Image - Using OpenCV

```
import cv2
import glob
data_dir = '/home/chentao/Pictures/'
for filename in glob.glob('%s*.jpg' % (data_dir)):
    image = cv2.imread(filename)
    # Only for display
    cv2.imshow(filename, image)
    cv2.waitKey(1000)
```

Load Image - Using PIL

```
import numpy as np
from PIL import Image
import glob
data_dir = '/home/chentao/Pictures/'
for filename in glob.glob('%s*.jpg' % (data_dir)):
    image = np.asarray(Image.open(filename))
    Image.fromarray(image).show()
```

Load Image - Using scikit-image

```
import skimage.io as ski_io
from PIL import Image
import glob
data_dir = '/home/chentao/Pictures/'
for filename in glob.glob('%s*.jpg' % (data_dir)):
    image = ski_io.imread(filename)
    Image.fromarray(image).show()
```

Load Image - Using scipy

```
from scipy import misc
from scipy import ndimage
from PIL import Image
import glob
data_dir = '/home/chentao/Pictures/'
for filename in glob.glob('%s*.jpg' % (data_dir)):
    # image = misc.imread(filename) # either one
    image = ndimage.imread(filename)
    Image.fromarray(image).show()
```

Load Images into NHWC format

```
import tensorflow as tf
from PIL import Image
import glob
import numpy as np

data_dir = '/home/chentao/Pictures/'
images = []
for filename in glob.glob('%s*.jpg'%(data_dir)):
    image = np.asarray(Image.open(filename))
    images.append(image)
## Method I
images = tf.pack(images, axis=0)
# show the first image
sess = tf.InteractiveSession()
Image.fromarray(images[0].eval()).show()
## Method II
images = np.array(images)
Image.fromarray(images[0]).show()
```

Concatenate multiple images on the channel dimension

```
import tensorflow as tf
from PIL import Image
import glob
import numpy as np

data_dir = '/home/chentao/Pictures/'
images = []
for filename in glob.glob('%s*.jpg'%(data_dir)):
    image = np.asarray(Image.open(filename))
    images.append(image)
## Method I
images = tf.concat(2, images)
sess = tf.InteractiveSession()
Image.fromarray(images[:,:,:3].eval()).show()
## Method II
images = np.concatenate(images, axis=2)
Image.fromarray(images[:,:,:3]).show()
```

Preloaded Data

- only used for small data sets that can be loaded entirely in memory
- Two approaches:
  - Store the data in a constant (simpler, but consumes more memory)
  - Store the data in a variable, that you initialize and then never change.

Preloaded Data - Using Constants

```
training_images = ...
training_labels = ...
with tf.Session():
    input_images = tf.constant(training_images)
    input_labels = tf.constant(training_labels)
    ...
```

Preloaded Data - Using Variables

```
training_images = ...
training_labels = ...
with tf.Session() as sess:
    images_initializer = tf.placeholder(dtype=training_images.dtype,
                                        shape=training_images.shape)
    label_initializer = tf.placeholder(dtype=training_labels.dtype,
                                       shape=training_labels.shape)
    input_images = tf.Variable(images_initializer, trainable=False,
                               collections=[])
    input_labels = tf.Variable(label_initializer, trainable=False,
                               collections=[])
    ...
    sess.run(input_images.initializer,
             feed_dict={images_initializer: training_images})
    sess.run(input_labels.initializer,
             feed_dict={label_initializer: training_labels})
```

Preloaded Data - Generate Batch

```
image, label = tf.train.slice_input_producer(
                                        [input_images,
                                         input_labels],
                                         num_epochs=FLAGS.num_epochs,
                                         shuffle=True)
label = tf.cast(label, tf.int32)
images, labels = tf.train.batch([image, label],
                                batch_size=FLAGS.batch_size)
```

- Full code is available at:
  fully_connected_preloaded.py
  fully_connected_preloaded_var.py

Outline

Reading Records from Files Pipeline

- The list of filenames
- Optional filename shuffling
- Optional epoch limit
- Filename queue
- A Reader for the file format
- A decoder for a record read by the reader
- Optional Preprocessing
- Example queue

CSV Files

```
import tensorflow as tf
filename_queue = tf.train.string_input_producer(['csv_data/file0.csv',
                                                 'csv_data/file1.csv'])
reader = tf.TextLineReader()
key, value = reader.read(filename_queue)

record_defaults = [[1], [1], [1], [1], [1]]
col1, col2, col3, col4, col5 = tf.decode_csv(
                                    value, record_defaults=record_defaults)
features = tf.pack([col1, col2, col3, col4])

with tf.Session() as sess:
    # Start populating the filename queue.
    coord = tf.train.Coordinator()
    threads = tf.train.start_queue_runners(coord=coord)

    for i in range(2000):
        key_in,value_in,example,label = sess.run([key, value, features, col5])
        print('key:',key_in, ' value:',value_in, '  example:',list(example),
              ' label:',label)

    coord.request_stop()
    coord.join(threads)
```

Fixed length records

```
filenames = [os.path.join(data_dir, 'data_batch_%d.bin' % i)
             for i in xrange(1, 6)]
filename_queue = tf.train.string_input_producer(filenames)
reader = tf.FixedLengthRecordReader(record_bytes=record_bytes)
key, value = reader.read(filename_queue)
record_bytes = tf.decode_raw(value, tf.uint8)

# Suppose the first bytes represent the label
label = tf.cast(tf.slice(record_bytes, [0], [label_bytes]), tf.int32)

# Suppose the remaining bytes after the label represent the image,
# which we reshape from [depth * height * width] to [depth, height, width].
depth_major = tf.reshape(tf.slice(record_bytes, [label_bytes], [image_bytes]),
                         [depth, height, width])
# Convert from [depth, height, width] to [height, width, depth].
image = tf.transpose(depth_major, [1, 2, 0])
```

- Full code is available at: cifar10_input.py

Standard TensorFlow Format

- Convert whatever data into a supported format(TFRecords File)

TFRecord File

- tf.train.Example protocol buffers
  - tf.train.Features (a map of strings to Feature)
    - tf.train.Feature (a FloatList, a ByteList or a Int64List)

Standard TensorFlow Format - tf.train.Example

```
# construct the Example proto boject
example = tf.train.Example(
    # Example contains a Features proto object
    features=tf.train.Features(
        # Features contains a map of string to Feature proto objects
        feature={
        # A Feature contains one of either a int64_list,
        # float_list, or bytes_list
        'label': tf.train.Feature(
                    int64_list=tf.train.Int64List(value=[label])),
        'image': tf.train.Feature(
                    int64_list=tf.train.Int64List(value=
                                features.astype('int64'))),
}))
```

Standard TensorFlow Format - Convert to TFRecord

```
def _int64_feature(value):
    return tf.train.Feature(int64_list=
                            tf.train.Int64List(value=[value]))


def _bytes_feature(value):
    return tf.train.Feature(bytes_list=
                            tf.train.BytesList(value=[value]))

filename = 'data.tfrecords'
writer = tf.python_io.TFRecordWriter(filename)
for index in range(dataset_size):
    image_raw = images[index].tostring()
    example = tf.train.Example(features=tf.train.Features(feature={
        'height': _int64_feature(rows),
        'width': _int64_feature(cols),
        'depth': _int64_feature(depth),
        'label': _int64_feature(int(labels[index])),
        'image_raw': _bytes_feature(image_raw)}))
    writer.write(example.SerializeToString())
writer.close()
```

- Full code is available at: convert_to_records.py and build_image_data.py

Standard TensorFlow Format - Read TFRecord Method I

```
import tensorflow as tf

filename = "data.tfrecords"
for serialized_example in tf.python_io.tf_record_iterator(filename):
    example = tf.train.Example()
    example.ParseFromString(serialized_example)

    # traverse the Example format to get data
    image = example.features.feature['image_raw'].int64_list.value
    label = example.features.feature['label'].int64_list.value[0]
```

Standard TensorFlow Format - Read TFRecord Method II

```
filename = "data.tfrecords"
filename_queue = tf.train.string_input_producer(
                                    [filename], num_epochs=num_epochs)
reader = tf.TFRecordReader()
_, serialized_example = reader.read(filename_queue)
features = tf.parse_single_example(
                    serialized_example,
                    features={
                         'image_raw': tf.FixedLenFeature([], tf.string),
                         'label': tf.FixedLenFeature([], tf.int64),
                    })
image = tf.decode_raw(features['image_raw'], tf.uint8)
label = tf.cast(features['label'], tf.int32)
```

- Full code is available at: fully_connected_reader.py

Preprocessing

```
image = tf.cast(image, tf.float32)
# Image processing for training the network. Note the many random
# distortions applied to the image.

# Randomly crop a [height, width] section of the image.
distorted_image = tf.random_crop(image, [height, width, 3])

# Randomly flip the image horizontally.
distorted_image = tf.image.random_flip_left_right(distorted_image)

distorted_image = tf.image.random_brightness(distorted_image,max_delta=63)
distorted_image = tf.image.random_contrast(distorted_image,
                                           lower=0.2, upper=1.8)

# Subtract off the mean and divide by the variance of the pixels.
float_image = tf.image.per_image_standardization(distorted_image)
```

- Full code is available at: cifar10_input.py

Batching

```
def read_my_file_format(filename_queue):
    reader = tf.SomeReader()
    key, record_string = reader.read(filename_queue)
    example, label = tf.some_decoder(record_string)
    processed_example = some_processing(example)
    return processed_example, label
# Method 1
# filenames is a list of image files like ['data/1.jpg', 'data/2.jpg',...]
# or a list of tfrecord files, csv files, binary files, etc.

def input_pipeline(filenames, batch_size, num_epochs=None):
    filename_queue = tf.train.string_input_producer(
            filenames, num_epochs=num_epochs, shuffle=True)
    example, label = read_my_file_format(filename_queue)
    min_after_dequeue = 10000
    capacity = min_after_dequeue + 3 * batch_size
    example_batch, label_batch = tf.train.shuffle_batch(
           [example, label], batch_size=batch_size, capacity=capacity,
           min_after_dequeue=min_after_dequeue)
    return example_batch, label_batch
```

- tf.train.shuffle_batch(tensors,enqueue_many=False): tensors is a single example
- tf.train.shuffle_batch(tensors,enqueue_many=True): tensors is a batch of examples

Batching

```
# Method 2
# filenames is a list of filenames like ['data/1.jpg', 'data/2.jpg',...]
def input_pipeline(filenames, batch_size, num_epochs=None):
    image_files = tf.convert_to_tensor(all_filenames, dtype=dtypes.string)
    labels = tf.convert_to_tensor(all_labels, dtype=dtypes.int32)
    train_input_queue = tf.train.slice_input_producer(
                                [image_files, labels],
                                shuffle=True)
    file_content = tf.read_file(train_input_queue[0])
    train_image = tf.image.decode_jpeg(file_content, channels=NUM_CHANNELS)
    train_label = train_input_queue[1]
    train_image.set_shape([IMAGE_HEIGHT, IMAGE_WIDTH, NUM_CHANNELS])
    train_image_batch, train_label_batch = tf.train.batch(
                                                [train_image, train_label],
                                                batch_size=BATCH_SIZE)
    return train_image_batch, train_label_batch
```

Prefetch by QueueRunner

- Many of the **tf.train** functions add **QueueRunner** to graph
- Call **tf.train.start_queue_runners** before training or inferencing
- Use **tf.train.Coordinator** to coordinate the termination of a set of threads

Prefetch by QueueRunner

```
init_op = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init_op)

# Start input enqueue threads.
coord = tf.train.Coordinator()
threads = tf.train.start_queue_runners(sess=sess, coord=coord)

try:
    while not coord.should_stop():
        # Run training steps or whatever
        sess.run(train_op)

except tf.errors.OutOfRangeError:
    print('Done training -- epoch limit reached')
finally:
    # When done, ask the threads to stop.
    coord.request_stop()

# Wait for threads to finish.
coord.join(threads)
sess.close()
```