LearningTensorFlow.com

Toggle navigation

TensorFlow supports reading larger datasets, specifically so that the data is never all kept in memory at once (it wouldn't be very useful if it had this limitation). There are a few functions and options you can use, from standard Python all the way to specific Ops.

TensorFlow also has support for writing custom data handlers, which is worth looking into if you have a very large project with lots of data already. Writing a custom data loading is a little effort up-front, but can save lots of time later on. Check out the official documentation here for more on this topic.

In this lesson, we will look at the basics of reading a CSV file, using TensorFlow, and using that data in a graph.

Placeholders

The most basic method for reading data is to simply read it with standard python code. Let's take a look at a basic example of this, reading data from this file of the 2016 Olympic Games medal tally.

First, we create our graph, which takes a single line of data, and adds up the total medals.

```
import tensorflow as tf
import os

dir_path = os.path.dirname(os.path.realpath(__file__))
filename = dir_path + "/olympics2016.csv"

features = tf.placeholder(tf.int32, shape=[3], name='features')
country = tf.placeholder(tf.string, name='country')
total = tf.reduce_sum(features, name='total')
```

Next, I'm going to introduce a new operation, called `Print`, which prints out the current value of some nodes on the graph. It's an identity element, which means it takes an operation as input and just returns the same value as an output.

```
printerop = tf.Print(total, [country, features, total], name='printer')
```

What happens when you eval `printerop`? It basically logs the current values in the second parameter (in this case, the list `[country, features, total]`) and returns the first value (`total`). It is considered a Variable, though, so we will need to initialise all variables when we start our session.

Next, we start the session and then open up the file for reading. Note that the file reading is done purely in python - we are just reading it at the same time we execute our graph.

```
with tf.Session() as sess:
    sess.run( tf.global_variables_initializer())
```

```
    with open(filename) as inf:
        # Skip header
        next(inf)
        for line in inf:
            # Read data, using python, into our features
            country_name, code, gold, silver, bronze, total = line.strip().split(",")
            gold = int(gold)
            silver = int(silver)
            bronze = int(bronze)
            # Run the Print ob
            total = sess.run(printerop, feed_dict={features: [gold, silver, bronze], country:country_name})
            print(country_name, total)
```

Within the inner parts of the loop, we read a line of the file, split it by comma, convert the values to integers and then feed the data into a `feed_dict` as placeholder values. If you aren't sure what is going on here, checkout our tutorial on placeholders here.

When you run this, you'll see two outputs for every line. The first output will be the result of `printerop`, which looks a little like this:

```
I tensorflow/core/kernels/logging_ops.cc:79] [\"France\"][10 18 14][42]
```

The next output will be the result from the `print(country_name, total)` line, which prints the current country name (a python variable) and the result from running `printerop`. As `printerop` is an identity function, the result from calling this is just the result from evaluating the `total` operation, which adds up the gold, silver and bronze counts.

It is generally fine to work in a manner similar to this. Create placeholders, load a bit of data into memory, compute on it, and loop with new data. This is, after all, what placeholders are for.

Reading CSV files in TensorFlow

TensorFlow supports directly reading data into tensors, however, the format is a little clunky. I'm going to step through *one* way to do this, but I've chosen a specifically generic method that I hope you can use for your own projects.

The steps are to create a queue (list) of the filenames you want to read, then create a reader operation that will later perform the read. From this reader op, create variables that are replaced with the actual values when they are executed during the graph execution phase.

Let's take a look at what the last couple of steps of that process look like:

```
def create_file_reader_ops(filename_queue):
    reader = tf.TextLineReader(skip_header_lines=1)
    _, csv_row = reader.read(filename_queue)
    record_defaults = [[""], [""], [0], [0], [0], [0]]
    country, code, gold, silver, bronze, total = tf.decode_csv(csv_row, record_defaults=record_defaults)
    features = tf.pack([gold, silver, bronze])
    return features, country
```

The reader here technically takes a queue object, not a normal Python list, so we need to build one before passing it to our function:

```
filename_queue = tf.train.string_input_producer(filenames, num_epochs=1, shuffle=False)
example, country = create_file_reader_ops(filename_queue)
```

Those operations that result from that function call will later represent single entries from our dataset. Running these requires a little more work than normal. The reason is that the queue itself doesn't sit on the graph in the same way a normal operation does, so we need a `Coordinator` to manage running through the queue. This co-ordinator will increment through the dataset everytime `example` and `label` are evaluated, as they effectively *pull* data from the file.

```
with tf.Session() as sess:
     tf.global_variables_initializer().run()
    coord = tf.train.Coordinator()
    threads = tf.train.start_queue_runners(coord=coord)
    while True:
        try:
            example_data, country_name = sess.run([example, country])
            print(example_data, country_name)
        except tf.errors.OutOfRangeError:
            break
```

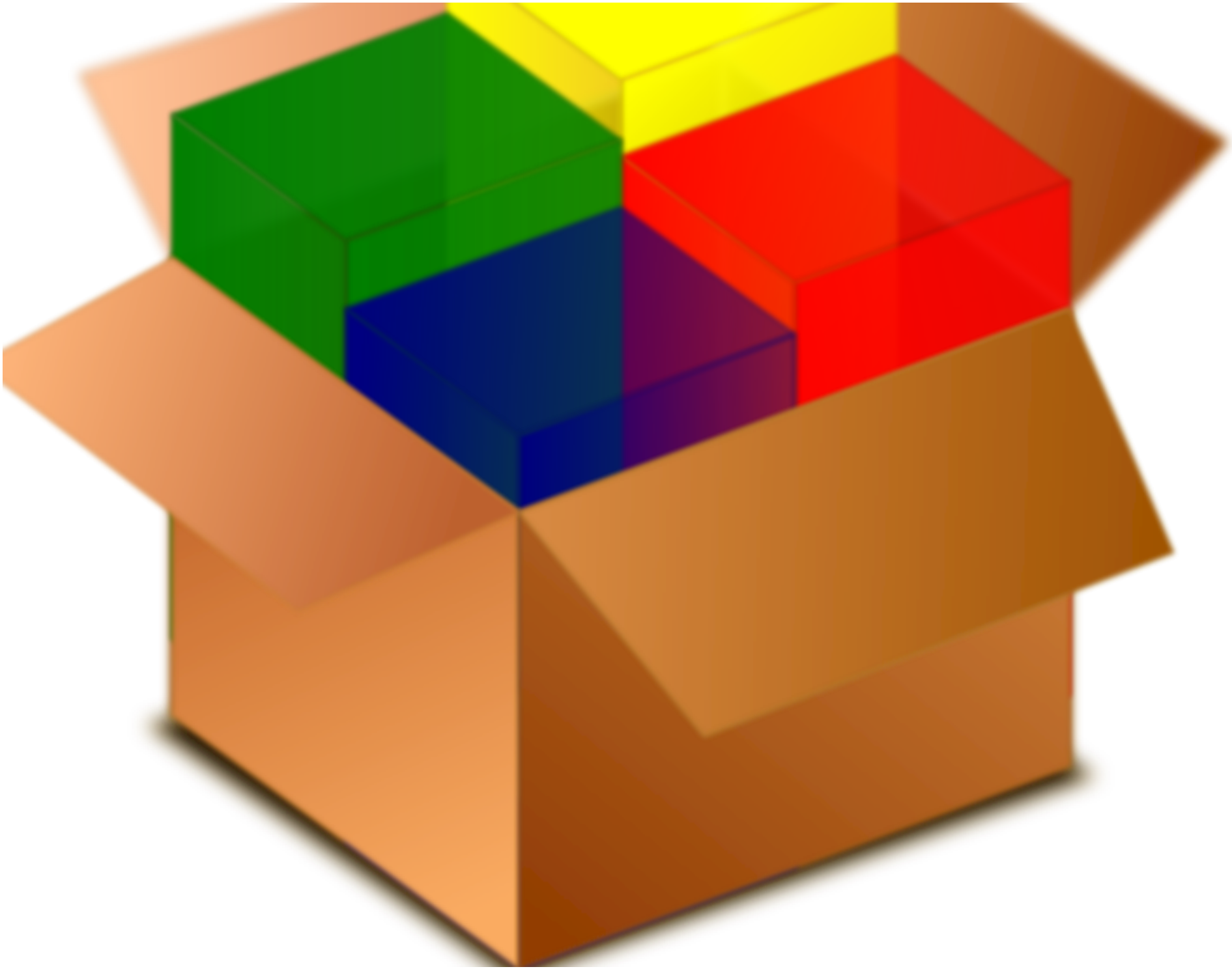
The inner `while` loop keeps looping until we hit an `OutOfRangeError`, indicating there is no more data to recover.

With this code, we now get one at a time, the rows from our dataset, loaded straight into our graph. There are other functions for creating batches and shuffling - check out some of the parameters from `tf.train.string_input_producer` and `tf.train.shuffle_batch` if you'd like to learn more about these.

In this lesson we looked at:

1. Reading data using Python while executing a TensorFlow graph
2. The `tf.Print` operation
3. Reading data directly into TensorFlow graphs/variables
4. Queue objects

Exercises



Stuck? Looking for more content?

If you are looking for solutions on the exercises, or just want to see how I solved them, then our solutions bundle is what you are after. Buying the bundle gives you **free updates for life** - meaning when we add a new lesson, you get an updated bundle with the solutions. It's just $7, and it also helps us to keep running the site with free lessons.

1. Update the code for the second example (reading the file directly into TensorFlow) to output the sum in the same manner as the python-version (i.e. both print it out and use `tf.Print`)
2. Unpack the features op in create_file_reader_ops, i.e. do not do the `tf.pack` line. Change the rest of the code to follow the case where features is returned as three separate features instead of a single packed feature. What needs to change?
3. Split the data file into several different files (this can be done with a text editor) and update the queue to read them all.
4. Use `tf.train.shuffle_batch` to batch together multiple lines into a single variable. This is more useful for larger datasets than reading on a row-by-row basis.

For question 4, a good target is to load as much data as you can in a single batch, but not too much that it overloads your computer's RAM. That won't matter for this dataset, but keep it in mind for later down the track.

As another aside, not all of the data is returned when using a batch – if the batch doesn't fill, it isn't returned.

LearningTensorFlow.com, providing free lessons on TensorFlow, including Machine Learning, Linear Algebra, Distributed Computing, and more!

A dataPipeline.com.au initiative. Terms and Conditions
Current Tensorflow version supported on site is 1.2.0

We are a participant in the Amazon Services LLC Associates Program, an affiliate advertising program designed to provide a means for us to earn fees by linking to Amazon.com and affiliated sites.