How to *actually* read CSV data in TensorFlow?

I'm relatively new to the world of TensorFlow, and pretty perplexed by how you'd ***actually*** read CSV data into a usable example/label tensors in TensorFlow. The example from the TensorFlow tutorial on reading CSV data is pretty fragmented and only gets you part of the way to being able to train on CSV data.

Here's my code that I've pieced together, based off that CSV tutorial:

```
from __future__ import print_function
import tensorflow as tf

def file_len(fname):
    with open(fname) as f:
        for i, l in enumerate(f):
            pass
    return i + 1

filename = "csv_test_data.csv"

# setup text reader
file_length = file_len(filename)
filename_queue = tf.train.string_input_producer([filename])
reader = tf.TextLineReader(skip_header_lines=1)
_, csv_row = reader.read(filename_queue)

# setup CSV decoding
record_defaults = [[0],[0],[0],[0],[0]]
col1,col2,col3,col4,col5 = tf.decode_csv(csv_row, record_defaults=record_defaults)
```

```
  # start populating filename queue
  coord = tf.train.Coordinator()
  threads = tf.train.start_queue_runners(coord=coord)

  for i in range(file_length):
    # retrieve a single instance
    example, label = sess.run([features, col5])
    print(example, label)

  coord.request_stop()
  coord.join(threads)
  print("\ndone loading")
```

And here is an brief example from the CSV file I'm loading - pretty basic data - 4 feature columns, and 1 label column:

```
0,0,0,0,0
0,15,0,0,0
0,30,0,0,0
0,45,0,0,0
```

All the code above does is **print each example from the CSV file, one by one**, which, while nice, is pretty darn useless for training.

What I'm struggling with here is how you'd actually turn those individual examples, loaded one-by-one, into a training dataset. For example, here's a notebook I was working on in the Udacity Deep Learning course. I basically want to take the CSV data I'm loading, and plop it into something like **train_dataset** and **train_labels**:

```
def reformat(dataset, labels):
  dataset = dataset.reshape((-1, image_size * image_size)).astype(np.float32)
  # Map 2 to [0.0, 1.0, 0.0 ...], 3 to [0.0, 0.0, 1.0 ...]
  labels = (np.arange(num_labels) == labels[:,None]).astype(np.float32)
  return dataset, labels
train_dataset, train_labels = reformat(train_dataset, train_labels)
valid_dataset, valid_labels = reformat(valid_dataset, valid_labels)
test_dataset, test_labels = reformat(test_dataset, test_labels)
print('Training set', train_dataset.shape, train_labels.shape)
print('Validation set', valid_dataset.shape, valid_labels.shape)
print('Test set', test_dataset.shape, test_labels.shape)
```

I've tried using `tf.train.shuffle_batch`, like this, but it just inexplicably hangs:

```
    print(example, label)
```

So to sum up, here are my questions:

- **What am I missing about this process?**
  - It feels like there is some key intuition that I'm missing about how to properly build an input pipeline.
- **Is there a way to avoid having to know the length of the CSV file?**
  - It feels pretty inelegant to have to know the number of lines you want to process (the `for i in range(file_length)` line of code above)

Edit: As soon as Yaroslav pointed out that I was likely mixing up imperative and graph-construction parts here, it started to become clearer. I was able to pull together the following code, which I think is closer to what would typically done when training a model from CSV (excluding any model training code):

```
from __future__ import print_function
import numpy as np
import tensorflow as tf
import math as math
import argparse

parser = argparse.ArgumentParser()
parser.add_argument('dataset')
args = parser.parse_args()

def file_len(fname):
    with open(fname) as f:
        for i, l in enumerate(f):
            pass
    return i + 1

def read_from_csv(filename_queue):
  reader = tf.TextLineReader(skip_header_lines=1)
  _, csv_row = reader.read(filename_queue)
  record_defaults = [[0],[0],[0],[0],[0]]
  colHour,colQuarter,colAction,colUser,colLabel = tf.decode_csv(csv_row,
record_defaults=record_defaults)
  features = tf.pack([colHour,colQuarter,colAction,colUser])
  label = tf.pack([colLabel])
  example, label = read_from_csv(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

file_length = file_len(args.dataset) - 1
examples, labels = input_pipeline(file_length, 1)

with tf.Session() as sess:
  tf.initialize_all_variables().run()

  # start populating filename queue
  coord = tf.train.Coordinator()
  threads = tf.train.start_queue_runners(coord=coord)

  try:
    while not coord.should_stop():
      example_batch, label_batch = sess.run([examples, labels])
      print(example_batch)
  except tf.errors.OutOfRangeError:
    print('Done training, epoch reached')
  finally:
    coord.request_stop()

  coord.join(threads)
```

python csv tensorflow

edited May 8 '16 at 1:51

asked May 7 '16 at 17:57

Rob Ringham
14.1k 26 95 140

I've been trying out your code, but can't get it to work. Is there something I'm missing that you determined? Thanks. I've posted a thread here so you can get more details: stackoverflow.com/questions/40143019/… – Link Oct 19 '16 at 23:19

`tf.train.shuffle_batch` creates a new queue node, and a single node can be used to process the entire dataset. So I think you are hanging because you created a bunch of `shuffle_batch` queues in your for loop and didn't start queue runners for them.

Normal input pipeline usage looks like this:

1. Add nodes like `shuffle_batch` to input pipeline
2. (optional, to prevent unintentional graph modification) finalize graph

--- end of graph construction, beginning of imperative programming --

3. `tf.start_queue_runners`
4. `while(True): session.run()`

To be more scalable (to avoid Python GIL), you could generate all of your data using TensorFlow pipeline. However, if performance is not critical, you can hook up a numpy array to an input pipeline by using `slice_input_producer`. Here's an example with some `Print` nodes to see what's going on (messages in `Print` go to stdout when node is run)

```
tf.reset_default_graph()

num_examples = 5
num_features = 2
data = np.reshape(np.arange(num_examples*num_features), (num_examples,
num_features))
print data

(data_node,) = tf.slice_input_producer([tf.constant(data)], num_epochs=1,
shuffle=False)
data_node_debug = tf.Print(data_node, [data_node], "Dequeueing from data_node ")
data_batch = tf.batch([data_node_debug], batch_size=2)
data_batch_debug = tf.Print(data_batch, [data_batch], "Dequeueing from data_batch
")

sess = tf.InteractiveSession()
sess.run(tf.initialize_all_variables())
tf.get_default_graph().finalize()
tf.start_queue_runners()
```

You should see something like this

```
[[0 1]
 [2 3]
 [4 5]
 [6 7]
 [8 9]]
[[0 1]
 [2 3]]
[[4 5]
 [6 7]]
No more inputs.
```

The "8, 9" numbers didn't fill up the full batch, so they didn't get produced. Also `tf.Print` are printed to sys.stdout, so they show up in separately in Terminal for me.

PS: a minimal of connecting `batch` to a manually initialized queue is in github issue 2193

Also, for debugging purposes you might want to set `timeout` on your session so that your IPython notebook doesn't hang on empty queue dequeues. I use this helper function for my sessions

```
def create_session():
  config = tf.ConfigProto(log_device_placement=True)
  config.gpu_options.per_process_gpu_memory_fraction=0.3 # don't hog all vRAM
  config.operation_timeout_in_ms=60000   # terminate on long hangs
  # create interactive session to register a default session
  sess = tf.InteractiveSession("", config=config)
  return sess
```

Scalability Notes:

1. `tf.constant` inlines copy of your data into the Graph. There's a fundamental limit of 2GB on size of Graph definition so that's an upper limit on size of data
2. You could get around that limit by using `v=tf.Variable` and saving the data into there by running `v.assign_op` with a `tf.placeholder` on right-hand side and feeding numpy array to the placeholder (`feed_dict`)

edited May 7 '16 at 20:34

answered May 7 '16 at 20:03

Yaroslav Bulatov
27.9k 11 72 124

2 Ahh, yes! You are totally right - as soon as you said: "I think you are mixing up imperative and graph-construction parts here", I started to see where I was going wrong. I've posted an edit to my question that includes the latest code I put together, which actually gets me closer to what I want - I'm able to successfully read in CSV data and batch it in such a way that I could train a model. – Rob Ringham May 8 '16 at 1:54

2 I suggest updating this answer so it works with recent versions of TensorFlow: replace `tf.slice_input_producer()` with `tf.train.slice_input_producer()` (and similarly for several other functions). And also add `sess.run(tf.initialize_local_variables())` after `sess.run(tf.initialize_all_variables())` . – MiniQuark Aug 29 '16 at 21:54

Some more changes to make: `pack()` is now `stack()` , and `initialize_all_variables()` should be replaced with `global_variables_initializer()` and `local_variables_initializer()` . – MiniQuark Apr 1 at 20:04

With tensorflow 1.0.1 you need to initialize local and global variables as `tf.group(tf.global_variables_initializer(), tf.local_variables_initializer()).run()` . You will need to initialize local variables since you are using num_epochs and as per documentation *"Note: if `num_epochs` is not `None` , this function creates local counter `epochs` ."* – Bruno R. Cardoso Apr 12 at 10:09

Or you could try this, the code loads the Iris dataset into tensorflow using pandas and numpy and a simple one neuron output is printed in the session. Hope it helps for a basic understanding.... [I havent added the way of one hot decoding labels].

```
import tensorflow as tf
import numpy
import pandas as pd
df=pd.read_csv('/home/nagarjun/Desktop/Iris.csv',usecols = [0,1,2,3,4],skiprows =
[0],header=None)
d = df.values
l = pd.read_csv('/home/nagarjun/Desktop/Iris.csv',usecols = [5] ,header=None)
labels = l.values
```

```
x = tf.placeholder(tf.float32,shape=(150,5))
x = data
w = tf.random_normal([100,150],mean=0.0, stddev=1.0, dtype=tf.float32)
y = tf.nn.softmax(tf.matmul(w,x))

with tf.Session() as sess:
    print sess.run(y)
```

answered Jan 6 at 20:54

Nagarjun Gururaj
112 1 4

This was very instructive, but if I understand correctly it does not show how to use the data for training... – dividebyzero Jan 21 at 17:38

yes, i'll add them soon... It should be trivial isn't it.... calculate the loss, run the optimizer anaway i'll add them soon – Nagarjun Gururaj Jan 23 at 21:32

2 Hi dividebyzero, sorry i'm late ! I found another link which is interesting and really eases the problem tensorflow.org/tutorials/tflearn.... Here you can load the csv files, train them, perform classification... – Nagarjun Gururaj Feb 14 at 16:15

@NagarjunGururaj Can I use the dataset constructed by the contrib_learn in the normal tensorflow routine? – Jay Wong Mar 31 at 21:40

which dataset ? You mean Iris or any other ? – Nagarjun Gururaj Apr 2 at 10:07