TFRecords for Humans

Thursday March 23, 2017

TensorFlow recommends its TFRecords format as the standard TensorFlow format for data on disk.

You don't have to use TFRecords with TensorFlow. But if you need to read data inside your TensorFlow graph, and a reader op doesn't exist for your data, it might be easier to transform your data to TFRecords than to write a custom data reader op.

Before using TFRecords in a distributed setting, you probably want to understand and work with them locally.

The TFRecords Format

TFRecords is a simple binary file format. It lets you put one or more strings of bytes into a file. You could put any bytes you like in a TFRecords file, but it'll be more useful to use the formats provided in TensorFlow.

TensorFlow defines two protocol buffer message types for use with TFRecords: the Example message type and the SequenceExample message type.

The pre-defined protocol buffer message types offer flexibility by letting you arrange your data as a map from string keys to values that are lists of integers, lists of 32-bit floats, or lists of bytes.

Writing and Reading in the style of `Example` Records without TensorFlow

An equivalent representation of an `Example` TFRecord using Python dictionaries might look like this:

```
my_dict = {'features' : {
    'my_ints': [5, 6],
    'my_float': [2.7],
    'my_bytes': ['data']
}}
```

In Python 2, the string literal `'data'` is bytes. The equivalent in Python 3 is `bytes('data', 'utf-8')`. And Python uses 64-bit floats rather than the 32-bit floats that TFRecords uses, so we have more precision in Python.

The values in this `dict` are accessed like this:

- `my_dict['features']['my_ints']`
- `my_dict['features']['my_float']`
- `my_dict['features']['my_bytes']`

Ordinarily, to save this data (serialize and write to disk) and then read it again (read from disk and deserialize) in Python you might use the pickle module. For example:

```
import pickle

my_dict_str = pickle.dumps(my_dict)
with open('my_dict.pkl', 'w') as f:
    f.write(my_dict_str)

with open('my_dict.pkl', 'r') as f:
    that_dict_str = f.read()
that_dict = pickle.loads(that_dict_str)
```

Writing and Reading `Example` Records with TensorFlow

The TFRecords `Example` format defines things in detail: An `Example` contains one `Features`, which is a map from strings to `Feature` elements, which can each be `Int64List`, `FloatList`, or `BytesList`. (See also: example.proto and feature.proto)

```
import tensorflow as tf

my_example = tf.train.Example(features=tf.train.Features(feature={
    'my_ints': tf.train.Feature(int64_list=tf.train.Int64List(value=[5, 6])),
    'my_float': tf.train.Feature(float_list=tf.train.FloatList(value=[2.7])),
    'my_bytes': tf.train.Feature(bytes_list=tf.train.BytesList(value=['data']))
}))
```

The values in the `Example` can be accessed then like this:

- `my_example.features.feature['my_ints'].int64_list.value`
- `my_example.features.feature['my_float'].float_list.value`
- `my_example.features.feature['my_bytes'].bytes_list.value`

Writing to and reading from disk are much like with `pickle`, except that the reader here provides all the records from a TFRecords file. In this example, there's only one record in the file.

```
my_example_str = my_example.SerializeToString()
with tf.python_io.TFRecordWriter('my_example.tfrecords') as writer:
    writer.write(my_example_str)

reader = tf.python_io.tf_record_iterator('my_example.tfrecords')
those_examples = [tf.train.Example().FromString(example_str)
                  for example_str in reader]
```

The file written: `my_example.tfrecords`.

Images in `Example` Records

The `Example` format lets you store pretty much any kind of data, including images. But the mechanism for arranging the data into serialized bytes, and then reconstructing the original format again, is left up to you. For more on this, see my post on Images and TFRecords.

For two more complete *in situ* examples of converting images to TFRecords, check out code for MNIST images and code for ImageNet images. The ImageNet code can be run on the command-line.

The `SequenceExample` Record

The `SequenceExample` message type essentially extends `Example` for sequence data. (You could imagine achieving the same effect with just `Example`, but it would be awkward.)

A `SequenceExample` keeps the same kind of map as `Example`, but calls it `context`, because it's thought of as the static context for the dynamic sequence data. And it adds another map, called `feature_lists`, that maps from string keys to lists of lists.

In Python dictionaries, a `SequenceExample` is like this:

```
my_seq_dict = {
    'context' : {
        'my_bytes':
            ['data']},
    'feature_lists' : {
        'my_ints': [
            [5, 6],
            [7, 8, 9]]}}
```

A corresponding full `SequenceExample` is a bit more verbose:

```
my_seq_ex = tf.train.SequenceExample(
    context=tf.train.Features(feature={
        'my_bytes':
            tf.train.Feature(bytes_list=tf.train.BytesList(value=['data']))}),
    feature_lists=tf.train.FeatureLists(feature_list={
        'my_ints': tf.train.FeatureList(feature=[
            tf.train.Feature(int64_list=tf.train.Int64List(value=[5, 6])),
            tf.train.Feature(int64_list=tf.train.Int64List(value=[7, 8, 9]))])}))
```

In a file: `my_seq_ex.tfrecords`.

You probably don't want to mix `Example` and `SequenceExample` records in the same TFRecords file.

Reading TFRecords in a TensorFlow Graph

You may eventually want to read TFRecords files with ops in a TensorFlow graph, using tf.TFRecordReader. This will involve a filename queue; for an example, check out some MNIST tutorial code.

I'm working on Building TensorFlow systems from components, a workshop at OSCON 2017.

- Plan ← Space
- Edit this page
- Find Aaron on
    - Twitter
    - LinkedIn
    - Google+
    - GitHub
    - email

- Comment below
