Protocol Buffer

Discover the benefits of using protocol buffers in TensorFlow.

Chapter Goals:

- Learn how protocol buffers are used in TensorFlow
- Implement a function to convert a Python dictionary to a `tf.train.Example` object

A. TensorFlow protocol buffer

Since protocol buffers use a structured format when storing data, they can be represented with Python classes. In TensorFlow, the `tf.train.Example` class represents the protocol buffer used to store data for the input pipeline.

Each individual `tf.train.Example` object describes data for a single dataset observation (e.g. a single row in a data table). We convert raw data to a protocol buffer by initializing a `tf.train.Example` object with the data's values.

When we initialize a `tf.train.Example` object, we need to set that object's `features` argument to a `tf.train.Features` object. The `tf.train.Features` class is initialized by setting the `feature` field to a dictionary that maps feature names to feature values.

```
import tensorflow as tf

features = tf.train.Features(feature=f_dict)  # f_dict is a dict
ex = tf.train.Example(features=features)
print(repr(ex))
```

The code above initializes a `tf.train.Example` object. Note that `f_dict` is a dictionary mapping feature names to feature values. We discuss how to create the feature dictionary later in this chapter.

the feature dictionary later in this chapter.

B. Features

Each feature value is represented by a `tf.train.Feature` object, which is initialized with exactly one of the following fields:

- `int64_list`, for integer data values. Set with a `tf.train.Int64List` object.
- `float_list`, for floating point data values. Set with a `tf.train.FloatList` object.
- `bytes_list`, for byte string data values. Set with a `tf.train.BytesList` object.

The code below creates `tf.train.Feature` objects from data values. The `encode` function used in the last example converts the string to bytes, so the type is compatible with `tf.train.BytesList`.

```
import tensorflow as tf

int_f = tf.train.Feature(
    int64_list=tf.train.Int64List(value=[1, 2]))
print(repr(int_f) + '\n')

float_f = tf.train.Feature(
    float_list=tf.train.FloatList(value=[-8.2, 5]))
print(repr(float_f) + '\n')

bytes_f = tf.train.Feature(
    bytes_list=tf.train.BytesList(value=[b'\xff\xcc', b'\xac']))
print(repr(bytes_f) + '\n')

str_f = tf.train.Feature(
    bytes_list=tf.train.BytesList(value=['joe'.encode()]))
print(repr(str_f) + '\n')
```

Note that the `value` field for the `tf.train.Int64List`, `tf.train.FloatList`, and `tf.train.BytesList` classes must be an iterable (e.g. list, NumPy array, or pandas Series). If a feature only has one data value, we would pass in an iterable containing the single value.

With the `tf.train.Feature` objects, we can create the dictionary that's used to initialize a `tf.train.Features` object.

```
import tensorflow as tf
```

```
f_dict = {
    'int_vals': int_f,

    'float_vals': float_f,
    'bytes_vals': bytes_f,
    'str_vals': str_f
}

features = tf.train.Features(feature=f_dict)

print(repr(features))
```

C. Bytes and text

When dealing with datasets containing bytes (e.g. images or videos) or text (e.g. articles or sentences), it is beneficial to first read all the data files and then store the read data in the `bytes_list` field of a `tf.train.Feature`. This saves us from having to open each individual file within our input pipeline, which can drastically improve efficiency.

task.py

story.txt

```
import tensorflow as tf

with open('story.txt') as f:
    words = f.read().split()

encw = [w.encode() for w in words]
words_feature = tf.train.Feature(
    bytes_list=tf.train.BytesList(value=encw))
print(repr(words_feature))

with open('img.jpg', 'rb') as f:
    img_bytes = f.read()

img_feature = tf.train.Feature(
    bytes_list=tf.train.BytesList(value=[img_bytes]))
print(repr(img_feature))
```

Time to Code!

In this chapter, you'll be completing the `dict_to_example` function, which converts a regular dictionary into a `tf.train.Example` object.

```
import tensorflow as tf
```

```
def dict_to_example(data_dict, config):
    feature_dict = {}
    for feature_name, value in data_dict.items():

        feature_config = config[feature_name]
        shape = feature_config['shape']
        if shape == () or shape == []:
            value = [value]
        value_type = feature_config['type']
        if value_type == 'int':
            feature_dict[feature_name] = make_int_feature(value)
        elif value_type == 'float':
            feature_dict[feature_name] = make_float_feature(value)
        elif value_type == 'string' or value_type == 'bytes':
            feature_dict[feature_name] = make_bytes_feature(
              value, value_type)
    features = tf.train.Features(feature=feature_dict)
    return tf.train.Example(features=features)
```

The `dict_to_example` function uses 3 helper functions: `make_int_feature`, `make_float_feature`, and `make_bytes_feature`. You will be creating each of these helper functions.

The `make_int_feature` function returns a `tf.train.Feature` initialized with the `int64_list` field.

Return a `tf.train.Feature` object with the `int64_list` attribute set equal to a `tf.train.Int64List` object (initialized with `value`).

```
import tensorflow as tf

def make_int_feature(value):
    # CODE HERE
    pass
```

The `make_float_feature` function returns a `tf.train.Feature` initialized with the `float_list` field.

Return a `tf.train.Feature` object with the `float_list` attribute set equal to a `tf.train.FloatList` object (initialized with `value`).

```
import tensorflow as tf

def make_float_feature(value):
    # CODE HERE
    pass
```

The `make_float_feature` function returns a `tf.train.Feature` initialized with the `bytes_list` field. Note that if the `value_type` is `'string'` we need to encode each of the elements in `value`.

If `value_type` is equal to `'string'`, encode each element in `value` (and keep the resultant list as `value`).

Return a `tf.train.Feature` object with the `bytes_list` attribute set equal to a `tf.train.BytesList` object (initialized with `value`).

```
import tensorflow as tf

def make_bytes_feature(value, value_type):
    # CODE HERE
    pass
```