Parsing

Parse data from serialized protocol buffers.

Chapter Goals:

- Learn how to parse a serialized protocol buffer
- Understand how feature data is represented in TensorFlow
- Implement a function that parses a serialized protocol buffer

A. Feature parsing

After creating an Example spec, it can be used to parse serialized protocol buffers that are read from a TFRecords file. Specifically, we use the Example spec as an argument to the tf.parse_single_example function, which converts a serialized protocol buffer into a usable feature dictionary.

```
import tensorflow as tf

print(example_spec)
print(repr(ex))

parsed = tf.parse_single_example(
    ex.SerializeToString(), example_spec)
print(repr(parsed))
```

You'll notice that the output of tf.parse_single_example is a dictionary mapping feature names to either a tf.Tensor or a tf.SparseTensor. Each FixedLenFeature is converted to a tf.Tensor and each VarLenFeature is converted to a tf.SparseTensor.

A tf.Tensor is basically TensorFlow's version of NumPy arrays, meaning it is a container for feature data with a fixed shape. tf.SparseTensor is used to represent data that may have many missing or empty values, making it useful for variable-length features.

In upcoming chapters, we'll discuss how we can combine the tf.Tensor and tf.SparseTensor values of the parsed dictionary to create an input layer for a TensorFlow model.

B. Shapes: () vs. 1

In the previous chapter, we brought up how using () for the shape of a FixedLenFeature is different from using 1.

Using () (or []) corresponds to a single data value, while using 1 (represented as (1,) in tf.Tensor) corresponds to a list containing a single data value.

Time to Code!

In this chapter you'll complete the parse_example function, which parses data from a serialized Example object.

Using the input Example spec, example_spec, we'll parse the serialized
protocol buffer, example_bytes.

Set parsed_features equal to tf.parse_single_example with example_bytes as the first argument and example_spec as the second argument.

After parsing the serialized protocol buffer into a dictionary, we may only want to return certain features. The features we return depend on the value of output_features.

If output_features is None (the default), we'll return the entire parsed dictionary. Otherwise, we only return the key-value pairs if the key is in the output_features list.

If output_features is not None, set parsed_features to a dictionary containing only the key-value pairs for keys that are listed in output_features.

Return parsed_features.