Mapping

Transform each individual observation in a dataset through mapping.

Chapter Goals:

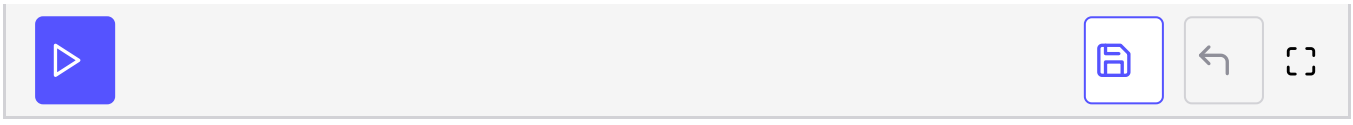
- Learn how to map a function onto each observation of a dataset
- Implement a function that creates a dataset of serialized protocol buffers and parses each observation

A. Mapping function

After initially creating a dataset from NumPy arrays or files, we oftentimes want to apply changes to make the dataset observations more useful. For example, we might create a dataset from heights measured in inches, but we want to train a model on the heights in centimeters. We can convert each observation to the desired format by using the `map` function.

```
import numpy as np
import tensorflow as tf

data = np.array([65.2, 70. ])
d1 = tf.data.Dataset.from_tensor_slices(data)
d2 = d1.map(lambda x:x * 2.54)
print(d2)
```



Using the map function to convert a dataset of heights from inches to cm.

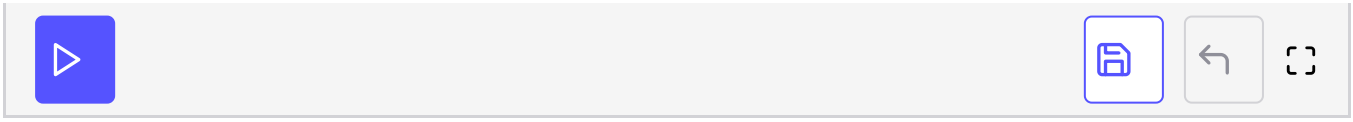
In the example above, `d1` is a dataset containing the height values from `data`, measured in inches. We use `map` to apply a function onto each observation of `d1`. The mapping function (represented by the lambda input to `map`) multiplies each observation value by 2.54 (the inch-centimeter conversion).

The output of `map`, which is `d2` in the example, is the resulting dataset containing the mapped observation values. In this case, the values of `d2` will be 165.608 and 182.88.

When a dataset is created from a tuple, the input function for `map` must take in a tuple as its argument.

```
import numpy as np
import tensorflow as tf

data1 = np.array([[1.2, 2.2],
       [7.3, 0. ]])
data2 = np.array([0.1, 1.1])
d1 = tf.data.Dataset.from_tensor_slices((data1, data2))
d2 = d1.map(lambda x,y:x + y)
print(d2)
```



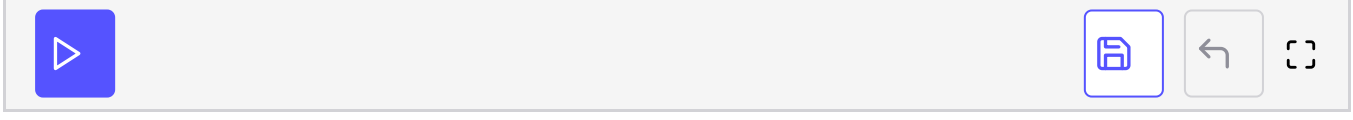
Using the map function to convert a dataset of tuple observations.

B. Wrapper functions

One thing to note about `map` is that its input function must only take in a single argument, representing an individual dataset observation. However, we may want to use a multi-argument function as the input to `map`. In this case, we can use a *wrapper* to ensure that the input function is in the correct format.

```
import numpy as np
import tensorflow as tf

def f(a, b):
       return a - b
data1 = np.array([[4.3, 2.7],
       [1.3, 1. ]])
data2 = np.array([0.2, 0.5])
d1 = tf.data.Dataset.from_tensor_slices(data1)
d2 = d1.map(lambda x:f(x, data2))
print(d2)
```



Using the map function with a wrapper function around function f.

In the example above, `f` is an external function that subtracts its second argument from its first argument. To use `f` as the mapping function for `d1` (with `data2` as the second argument), we create a wrapper function for `f`.

represented by the lambda input to `map`.

The wrapper function takes in a single argument, `x`, so it meets the criteria as an input to `map`. It then uses `x` as the first argument to `f`, while using `data2` as the second argument.

Time to Code!

In this chapter you'll be completing the `dataset_from_examples` function, which maps the `parse_example` function from chapter 5 onto a `TFRecordDataset`.

The first thing we'll do is create the Example spec that's used for parsing, by using the `create_example_spec` function from chapter 4.

Set `example_spec` equal to `create_example_spec` applied with `config` as the only argument.

Next, we create a dataset from the TFRecords files given by `filenames`.

Set `dataset` equal to `tf.data.TFRecordsDataset` initialized with `filenames`.

The `dataset` we created contains serialized protocol buffers for each observation. To parse each serialized protocol buffer, we need to map the `parse_example` function from chapter 5 onto each observation of the dataset.

Since the input function for `map` can only take in a single argument, we'll create a lambda wrapper around the `parse_example` function.

Set `wrapper` equal to a lambda function whose input argument is named `example`. The lambda function should return `parse_example` applied with `example`, `example_spec`, and `output_features` as the first, second, and third arguments.

Finally, we can apply the `map` function onto the dataset and return the output.

Set `dataset` equal to `dataset.map` applied with `wrapper` as the input function. Then return `dataset`.

```
import tensorflow as tf

def create_example_spec(config):
    example_spec = {}
```

```
    for feature_name, feature_config in config.items():
        if feature_config['type'] == 'int':
            tf_type = tf.int64
        elif feature_config['type'] == 'float':
            tf_type = tf.float32
        else:
            tf_type = tf.string
        shape = feature_config['shape']
        if shape is None:
            feature = tf.VarLenFeature(tf_type)
        else:
            default_value = feature_config.get('default_value', None)
            feature = tf.FixedLenFeature(shape, tf_type, default_value)
        example_spec[feature_name] = feature
    return example_spec

def parse_example(example_bytes, example_spec, output_features=None):
    parsed_features = tf.parse_single_example(example_bytes, example_spec)
    if output_features is not None:
        parsed_features = {k: parsed_features[k] for k in output_features}
    return parsed_features

# Map the parse_example function onto a TFRecord Dataset
def dataset_from_examples(filenames, config, output_features=None):
    # CODE HERE
    pass
```