Dataset Iteration

Iterate through a dataset to extract individual data observations.

Chapter Goals:

- Learn how to iterate through a dataset and extract values from data observations
- Implement a function that iterates through a NumPy-based dataset and extracts the feature data

A. Iterator

The previous few chapters focused on creating and configuring datasets. In this chapter, we'll discuss how to iterate through a dataset and extract the data.

To iterate through a dataset, we need to create an Iterator object. There are a few different ways to create an Iterator, but we'll focus on the simplest and most commonly used method, which is the make_one_shot_iterator function.

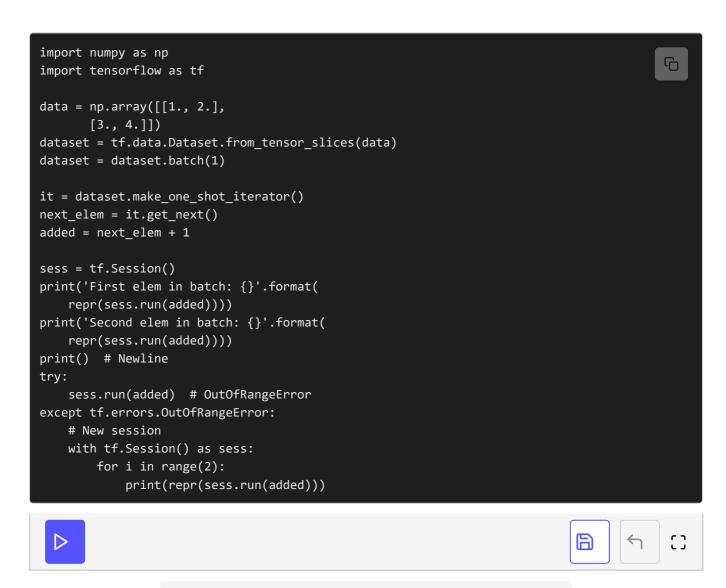
In the example, it represents an Iterator for dataset. The get_next function returns something we'll refer to as the next-element tensor.

The next-element tensor represents the batched data observation(s) at each iteration through the dataset. We can even apply operations or transformations to the next-element tensor. In the example above, we added 1 to each of the values in the data observation represented by next_elem.

B. Running the iteration

You'll notice that the next-element tensor is a tf.Tensor object. We use a tf.Session object to retrieve the values from a tf.Tensor.

tf. Session uses an important function called run, which allows us to extract the tf. Tensor values as NumPy data. For an in-depth look at tf. Session and the basics of TensorFlow execution, check out the Machine Learning for Software Engineers course.



Using tf. Session to extract values from the added variable.

Similar to File I/O in Python, we can create a tf.Session with or without the

with keyword. However, the with keyword lets us define all our computation

within the scope of the tf.Session object, so we don't have to manually close it to free its resources.

In the example, the i^{th} time we call <code>sess.run</code> on <code>added</code> will return the i^{th} observation from the dataset. Since we used a +1 transformation to obtain <code>added</code> from <code>next_elem</code>, each observation's values are incremented by 1.

Notice that if we call <code>sess.run</code> three consecutive times within the same <code>tf.Session</code> object scope, an <code>OutOfRangeError</code> is raised on the third call. This is because the dataset only contains two data observations, and we didn't use the <code>repeat</code> function to increase the number of epochs we can iterate through.

C. Configured dataset

The dataset used in the previous two examples was somewhat simplistic, and only intended to showcase the basics of the iteration process. For a more complex example, we'll iterate through a dataset configured with shuffle, repeat, and batch.

```
import numpy as np
import tensorflow as tf
data = np.array([
  [1., 2.],
  [3., 4.],
  [5., 6.],
  [7., 8.],
  [0., 9.],
  [0., 0.]])
dataset = tf.data.Dataset.from tensor slices(data)
dataset = dataset.shuffle(6)
dataset = dataset.repeat()
dataset = dataset.batch(2)
it = dataset.make_one_shot_iterator()
next elem = it.get next()
with tf.Session() as sess:
  for i in range(4):
    print('Element {}: {}'.format(
      i + 1, repr(sess.run(next_elem))))
```





The first thing to notice is that, despite dataset having only six data observations, we were able to iterate through eight observations because we used the repeat function. In fact, since we used repeat with its default argument setting, we could continuously iterate through the dataset without raising an OutOfRangeError.

Since we set the batch size to 2 using batch, each iteration returned two data observations rather than 1. Furthermore, you'll notice that the observations appear in a random order due to shuffle. However, we still saw all the data observations within the first epoch (i.e. first three iterations), because the shuffling occurs on a per-epoch basis.