The total memory space required for the inputs, weights and bias is around 174 megabytes, which isn't that much memory. You could train this whole dataset on most CPUs and GPUs.

But larger datasets that you'll use in the future measured in gigabytes or more. It's possible to purchase more memory, but it's expensive. A Titan X GPU with 12 GB of memory costs over $1,000.

Instead, in order to run large models on your machine, you'll learn how to use mini-batching.

Let's look at how you implement mini-batching in TensorFlow.

TensorFlow Mini-batching

In order to use mini-batching, you must first divide your data into batches.

Unfortunately, it's sometimes impossible to divide the data into batches of exactly equal size. For example, imagine you'd like to create batches of 128 samples each from a dataset of 1000 samples. Since 128 does not evenly divide into 1000, you'd wind up with 7 batches of 128 samples, and 1 batch of 104 samples. (7*128 + 1*104 = 1000)

In that case, the size of the batches would vary, so you need to take advantage of TensorFlow's `tf.placeholder()` function to receive the varying batch sizes.

Continuing the example, if each sample had `n_input = 784` features and `n_classes = 10` possible labels, the dimensions for `features` would be `[None, n_input]` and `labels` would be `[None, n_classes]`.

```
# Features and Labels
features = tf.placeholder(tf.float32, [None, n_input])
labels = tf.placeholder(tf.float32, [None, n_classes])
```

What does `None` do here?

The `None` dimension is a placeholder for the batch size. At runtime, TensorFlow will