(while we may actually use a different optimizer, we will refer to updates as gradient descent for convenience). How can we iteratively perform gradient descent to learn on this dataset?

The simple answer is that we use a Python `for`-loop. In each iteration, we use `sess.run()` to fetch the `train_op` along with the merged summary op `merged` and the loss `l` from the graph. We feed all datapoints and labels into `sess.run()` using a feed dictionary.

The code snippet in Example 3-10 demonstrates this simple learning method. Note that we don't make use of minibatches for pedagogical simplicity. Code in following chapters will use minibatches when training on larger datasets.

Example 3-10. A simple example of training a model

```
n_steps = 1000
with tf.Session() as sess:
  sess.run(tf.global_variables_initializer())
  # Train model
  for i in range(n_steps):
    feed_dict = {x: x_np, y: y_np}
    _, summary, loss = sess.run([train_op, merged, l], feed_dict=feed_dict)
    print("step %d, loss: %f" % (i, loss))
    train_writer.add_summary(summary, i)
```

Training Linear and Logistic Models in TensorFlow

This section ties together all the TensorFlow concepts introduced in the previous section to train linear and logistic regression models upon the toy datasets we introduced previously in the chapter.

Linear Regression in TensorFlow

In this section, we will provide code to define a linear regression model in TensorFlow and learn its weights. This task is straightforward and you can do it without TensorFlow easily. Nevertheless, it's a good exercise to do in TensorFlow since it will bring together the new concepts that we have introduced throughout the chapter.

Defining and training linear regression in TensorFlow

The model for a linear regression is straightforward:

$$y = wx + b$$

Here w and b are the weights we wish to learn. We transform these weights into `tf.Variable` objects. We then use tensorial operations to construct the L^2 loss: