labeled correctly. However, 95% of data in our dataset is labeled 0 and only 5% are labeled 1. As a result the all-0 model (which labels everything negative) would achieve 95% accuracy! This isn't what we want.

A better choice would be to increase the weights of positive examples so that they count for more. For this purpose, we use the recommended per-example weights from MoleculeNet to compute a weighted classification accuracy where positive samples are weighted 19 times the weight of negative samples. Under this weighted accuracy, the all-0 model would have 50% accuracy, which seems much more reasonable.

ForI computing the weighted accuracy, we use the function `accuracy_score(true, pred, sample_weight=given_sample_weight)` from `sklearn.metrics`. This function has a keyword argument `sample_weight`, which lets us specify the desired weight for each datapoint. We use this function to compute the weighted metric on both the training and validation sets (Example 4-9).

Example 4-9. Computing a weighted accuracy

```
train_weighted_score = accuracy_score(train_y, train_y_pred, sample_weight=train_w)
print("Train Weighted Classification Accuracy: %f" % train_weighted_score)
valid_weighted_score = accuracy_score(valid_y, valid_y_pred, sample_weight=valid_w)
print("Valid Weighted Classification Accuracy: %f" % valid_weighted_score)
```

While we could reimplement this function ourselves, sometimes it's easier (and less error prone) to use standard functions from the Python data science infrastructure. Learning about this infrastructure and available functions is part of being a practicing data scientist. Now, we can train the model (for 10 epochs in the default setting) and gauge its accuracy:

```
Train Weighted Classification Accuracy: 0.742045
Valid Weighted Classification Accuracy: 0.648828
```

In Chapter 5, we will show you methods to systematically improve this accuracy and tune our fully connected model more carefully.

Using TensorBoard to Track Model Convergence

Now that we have specified our model, let's use TensorBoard to inspect the model. Let's first check the graph structure in TensorBoard (Figure 4-10).

The graph looks similar to that for logistic regression, with the addition of a new hidden layer. Let's expand the hidden layer to see what's inside (Figure 4-11).

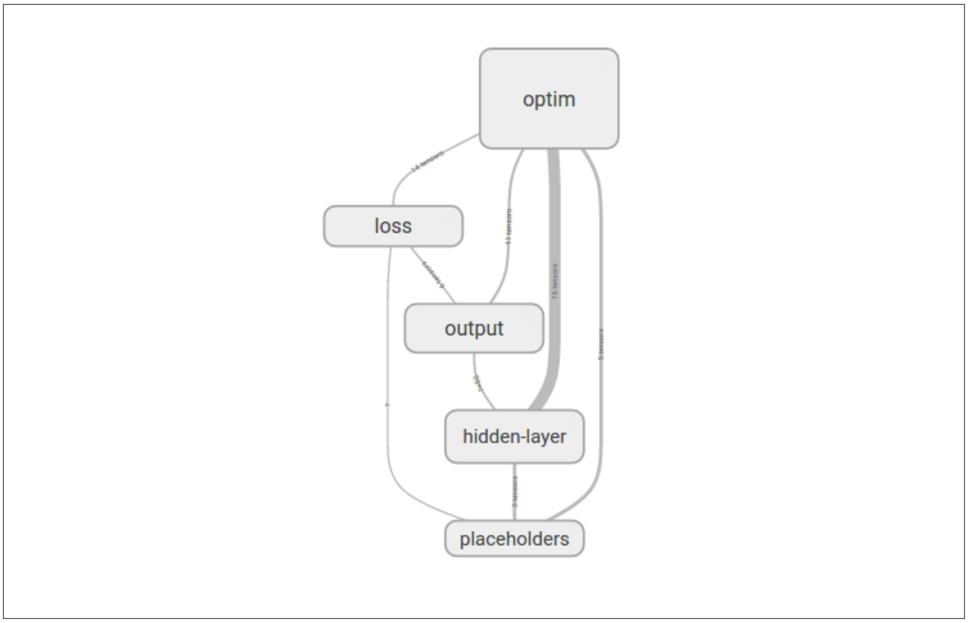

Figure 4-10. Visualizing the computation graph for a fully connected network.

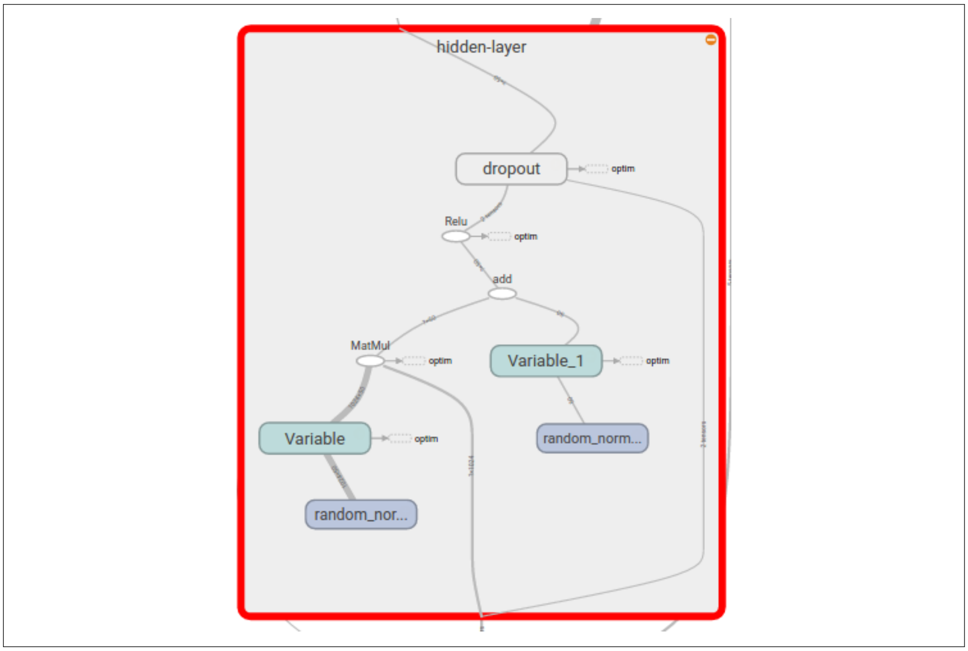

Figure 4-11. Visualizing the expanded computation graph for a fully connected network.

You can see how the new trainable variables and the dropout operation are represented here. Everything looks to be in the right place. Let's end now by looking at the loss curve over time (Figure 4-12).

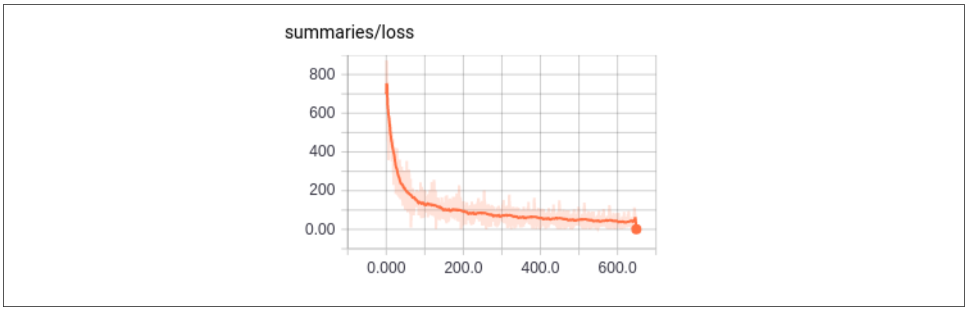

Figure 4-12. Visualizing the loss curve for a fully connected network.

The loss curve trends down as we saw in the previous section. But, let's zoom in to see what this loss looks like up close (Figure 4-13).

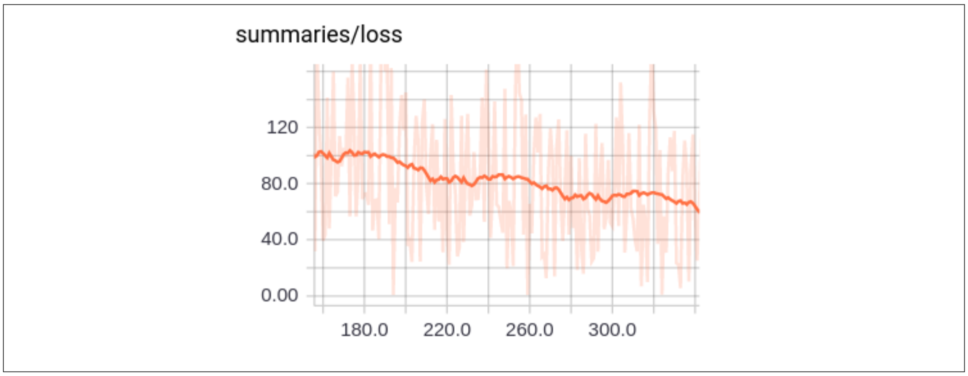

Figure 4-13. Zooming in on a section of the loss curve.

Note that loss looks much bumpier! This is one of the prices of using minibatch training. We no longer have the beautiful, smooth loss curves that we saw in the previous sections.

Review

In this chapter, we've introduced you to fully connected deep networks. We delved into the mathematical theory of these networks, and explored the concept of "universal approximation," which partially explains the learning power of fully connected networks. We ended with a case study, where you trained a deep fully connected architecture on the Tox21 dataset.