Example 4-6. Add a placeholder for dropout probability

```
keep_prob = tf.placeholder(tf.float32)
```

During training, we pass in the desired value, often 0.5, but at test time we set `keep_prob` to 1.0 since we want predictions made with all learned nodes. With this setup, adding dropout to the fully connected network specified in the previous section is simply a single extra line of code (Example 4-7).

Example 4-7. Defining a hidden layer with dropout

```
with tf.name_scope("hidden-layer"):
  W = tf.Variable(tf.random_normal((d, n_hidden)))
  b = tf.Variable(tf.random_normal((n_hidden,)))
  x_hidden = tf.nn.relu(tf.matmul(x, W) + b)
  # Apply dropout
  x_hidden = tf.nn.dropout(x_hidden, keep_prob)
```

Implementing Minibatching

To implement minibatching, we need to pull out a minibatch's worth of data each time we call `sess.run`. Luckily for us, our features and labels are already in NumPy arrays, and we can make use of NumPy's convenient syntax for slicing portions of arrays (Example 4-8).

Example 4-8. Training on minibatches

```
step = 0
for epoch in range(n_epochs):
  pos = 0
  while pos < N:
    batch_X = train_X[pos:pos+batch_size]
    batch_y = train_y[pos:pos+batch_size]
    feed_dict = {x: batch_X, y: batch_y, keep_prob: dropout_prob}
    _, summary, loss = sess.run([train_op, merged, l], feed_dict=feed_dict)
    print("epoch %d, step %d, loss: %f" % (epoch, step, loss))
    train_writer.add_summary(summary, step)

    step += 1
    pos += batch_size
```

Evaluating Model Accuracy

To evaluate model accuracy, standard practice requires measuring the accuracy of the model on data not used for training (namely the validation set). However, the fact that the data is imbalanced makes this tricky. The classification accuracy metric we used in the previous chapter simply measures the fraction of datapoints that were 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).