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