order to deal more conveniently with a minibatch of input at a time. (As an exercise, try working out the dimensions involved to see why this is so.) Finally, we apply the ReLU nonlinearity with the built-in `tf.nn.relu` activation function.

The remainder of the code for the fully connected layer is quite similar to that used for the logistic regression in the previous chapter. For completeness, we display the full code used to specify the network in Example 4-5. As a quick reminder, the full code for all models covered is available in the GitHub repo associated with this book. We strongly encourage you to try running the code for yourself.

Example 4-5. Defining the fully connected architecture

```
with tf.name_scope("placeholders"):
  x = tf.placeholder(tf.float32, (None, d))
  y = tf.placeholder(tf.float32, (None,))
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)
with tf.name_scope("output"):
  W = tf.Variable(tf.random_normal((n_hidden, 1)))
  b = tf.Variable(tf.random_normal((1,)))
  y_logit = tf.matmul(x_hidden, W) + b
  # the sigmoid gives the class probability of 1
  y_one_prob = tf.sigmoid(y_logit)
  # Rounding P(y=1) will give the correct prediction.
  y_pred = tf.round(y_one_prob)
with tf.name_scope("loss"):
  # Compute the cross-entropy term for each datapoint
  y_expand = tf.expand_dims(y, 1)
  entropy = tf.nn.sigmoid_cross_entropy_with_logits(logits=y_logit, labels=y_expand)
  # Sum all contributions
  l = tf.reduce_sum(entropy)

with tf.name_scope("optim"):
  train_op = tf.train.AdamOptimizer(learning_rate).minimize(l)

with tf.name_scope("summaries"):
  tf.summary.scalar("loss", l)
  merged = tf.summary.merge_all()
```

Adding Dropout to a Hidden Layer

TensorFlow takes care of implementing dropout for us in the built-in primitive `tf.nn.dropout(x, keep_prob)`, where `keep_prob` is the probability that any given node is kept. Recall from our earlier discussion that we want to turn on dropout when training and turn off dropout when making predictions. To handle this correctly, we will introduce a new placeholder for `keep_prob`, as shown in Example 4-6.

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