Tox21 has more datasets than we will analyze here, so we need to remove the labels associated with these extra datasets (Example 4-2).

Example 4-2. Remove extra datasets from Tox21

```
# Remove extra tasks
train_y = train_y[:, 0]
valid_y = valid_y[:, 0]
test_y = test_y[:, 0]
train_w = train_w[:, 0]
valid_w = valid_w[:, 0]
test_w = test_w[:, 0]
```

Accepting Minibatches of Placeholders

In the previous chapters, we created placeholders that accepted arguments of fixed size. When dealing with minibatched data, it is often convenient to be able to feed batches of variable size. Suppose that a dataset has 947 elements. Then with a minibatch size of 50, the last batch will have 47 elements. This would cause the code in Chapter 3 to crash. Luckily, TensorFlow has a simple fix to the situation: using `None` as a dimensional argument to a placeholder allows the placeholder to accept tensors with arbitrary size in that dimension (Example 4-3).

Example 4-3. Defining placeholders that accept minibatches of different sizes

```
d = 1024
with tf.name_scope("placeholders"):
  x = tf.placeholder(tf.float32, (None, d))
  y = tf.placeholder(tf.float32, (None,))
```

Note `d` is 1024, the dimensionality of our feature vectors.

Implementing a Hidden Layer

The code to implement a hidden layer is very similar to code we've seen in the last chapter for implementing logistic regression, as shown in Example 4-4.

Example 4-4. Defining a hidden layer

```
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)
```

We use a `tf.name_scope` to group together introduced variables. Note that we use the matricial form of the fully connected layer. We use the form xW instead of Wx in 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.