Example 5-2. A function mapping hyperparameters to different Tox21 fully connected networks

```
def eval_tox21_hyperparams(n_hidden=50, n_layers=1, learning_rate=.001,
                           dropout_prob=0.5, n_epochs=45, batch_size=100,
                           weight_positives=True):
```

Let's walk through each of these hyperparameters. `n_hidden` controls the number of neurons in each hidden layer of the network. `n_layers` controls the number of hidden layers. `learning_rate` controls the learning rate used in gradient descent, and `dropout_prob` is the probability neurons are not dropped during training steps. `n_epochs` controls the number of passes through the total data and `batch_size` controls the number of datapoints in each batch.

`weight_positives` is the only new hyperparameter here. For unbalanced datasets, it can often be helpful to weight examples of both classes to have equal weight. For the Tox21 dataset, DeepChem provides weights for us to use. We simply multiply the per-example cross-entropy terms by the weights to perform this weighting (Example 5-3).

Example 5-3. Weighting positive samples for Tox21

```
entropy = tf.nn.sigmoid_cross_entropy_with_logits(logits=y_logit, labels=y_expand)
# Multiply by weights
if weight_positives:
  w_expand = tf.expand_dims(w, 1)
  entropy = w_expand * entropy
```

Why is the method of picking hyperparameter values called graduate student descent? Machine learning, until recently, has been a primarily academic field. The tried-and-true method for designing a new machine learning algorithm has been describing the method desired to a new graduate student, and asking them to work out the details. This process is a bit of a rite of passage, and often requires the student to painfully try many design alternatives. On the whole, this is a very educational experience, since the only way to gain design aesthetic is to build up a memory of settings that work and don't work.

Grid Search

After having tried a few manual settings for hyperparameters, the process will begin to feel very tedious. Experienced programmers will be tempted to simply write a `for` loop that iterates over the choices of hyperparameters desired. This process is more or less the grid-search method. For each hyperparameter, pick a list of values that might be good hyperparameters. Write a nested `for` loop that tries all combinations of these values to find their validation accuracies, and keep track of the best performers.

There is one subtlety in the process, however. Deep networks can be fairly sensitive to the choice of random seed used to initialize the network. For this reason, it's worth repeating each choice of hyperparameter settings multiple times and averaging the results to damp the variance.

The code to do this is straightforward, as Example 5-4 shows.

Example 5-4. Performing grid search on Tox21 fully connected network hyperparameters

```
scores = {}
n_reps = 3
hidden_sizes = [50]
epochs = [10]
dropouts = [.5, 1.0]
num_layers = [1, 2]

for rep in range(n_reps):
  for n_epochs in epochs:
    for hidden_size in hidden_sizes:
      for dropout in dropouts:
        for n_layers in num_layers:
          score = eval_tox21_hyperparams(n_hidden=hidden_size, n_epochs=n_epochs,
                                         dropout_prob=dropout, n_layers=n_layers)
          if (hidden_size, n_epochs, dropout, n_layers) not in scores:
            scores[(hidden_size, n_epochs, dropout, n_layers)] = []
          scores[(hidden_size, n_epochs, dropout, n_layers)].append(score)
print("All Scores")
print(scores)

avg_scores = {}
for params, param_scores in scores.iteritems():
  avg_scores[params] = np.mean(np.array(param_scores))
print("Scores Averaged over %d repetitions" % n_reps)
```

Random Hyperparameter Search

For experienced practitioners, it will often be very tempting to reuse magical hyperparameter settings or search grids that worked in previous applications. These settings can be valuable, but they can also lead us astray. Each machine learning problem is slightly different, and the optimal settings might lie in a region of parameter space we haven't previously considered. For that reason, it's often worthwhile to try random settings for hyperparameters (where the random values are chosen from a reasonable range).

There's also a deeper reason to try random searches. In higher-dimensional spaces, regular grids can miss a lot of information, especially if the spacing between grid