where $\| \theta \|_2$ and $\| \theta \|_1$ denote the L^1 and L^2 penalties, respectively. From personal experience, these penalties tend to be less useful for deep models than dropout and early stopping. Some practitioners still make use of weight regularization, so it's worth understanding how to apply these penalties when tuning deep networks.

Training Fully Connected Networks

Training fully connected networks requires a few tricks beyond those you have seen so far in this book. First, unlike in the previous chapters, we will train models on larger datasets. For these datasets, we will show you how to use minibatches to speed up gradient descent. Second, we will return to the topic of tuning learning rates.

Minibatching

For large datasets (which may not even fit in memory), it isn't feasible to compute gradients on the full dataset at each step. Rather, practitioners often select a small chunk of data (typically 50–500 datapoints) and compute the gradient on these datapoints. This small chunk of data is traditionally called a minibatch.

In practice, minibatching seems to help convergence since more gradient descent steps can be taken with the same amount of compute. The correct size for a minibatch is an empirical question often set with hyperparameter tuning.

Learning rates

The learning rate dictates the amount of importance to give to each gradient descent step. Setting a correct learning rate can be tricky. Many beginning deep-learners set learning rates incorrectly and are surprised to find that their models don't learn or start returning NaNs. This situation has improved significantly with the development of methods such as ADAM that simplify choice of learning rate significantly, but it's worth tweaking the learning rate if models aren't learning anything.

Implementation in TensorFlow

In this section, we will show you how to implement a fully connected network in TensorFlow. We won't need to introduce many new TensorFlow primitives in this section since we have already covered most of the required basics.

Installing DeepChem

In this section, you will use the DeepChem machine learning toolchain for your experiments (full disclosure: one of the authors was the creator of DeepChem). Detailed installation directions for DeepChem can be found online, but briefly the Anaconda installation via the `conda` tool will likely be most convenient.

Tox21 Dataset

For our modeling case study, we will use a chemical dataset. Toxicologists are very interested in the task of using machine learning to predict whether a given compound will be toxic or not. This task is extremely complicated, since today's science has only a limited understanding of the metabolic processes that happen in a human body. However, biologists and chemists have worked out a limited set of experiments that provide indications of toxicity. If a compound is a "hit" in one of these experiments, it will likely be toxic for a human to ingest. However, these experiments are often costly to run, so data scientists aim to build machine learning models that can predict the outcomes of these experiments on new molecules.

One of the most important toxicological dataset collections is called Tox21. It was released by the NIH and EPA as part of a data science initiative and was used as the dataset in a model building challenge. The winner of this challenge used multitask fully connected networks (a variant of fully connected networks where each network predicts multiple quantities for each datapoint). We will analyze one of the datasets from the Tox21 collection. This dataset consists of a set of 10,000 molecules tested for interaction with the androgen receptor. The data science challenge is to predict whether new molecules will interact with the androgen receptor.

Processing this dataset can be tricky, so we will make use of the MoleculeNet dataset collection curated as part of DeepChem. Each molecule in Tox21 is processed into a bit-vector of length 1024 by DeepChem. Loading the dataset is then a few simple calls into DeepChem (Example 4-1).

Example 4-1. Load the Tox21 dataset

```python
import deepchem as dc

_, (train, valid, test), _ = dc.molnet.load_tox21()
train_X, train_y, train_w = train.X, train.y, train.w
valid_X, valid_y, valid_w = valid.X, valid.y, valid.w
test_X, test_y, test_w = test.X, test.y, test.w
```

Here the `X` variables hold processed feature vectors, `y` holds labels, and `w` holds example weights. The labels are binary 1/0 for compounds that interact or don't interact with the androgen receptor. Tox21 holds *imbalanced* datasets, where there are far fewer positive examples than negative examples. `w` holds recommended per-example weights that give more emphasis to positive examples (increasing the importance of rare examples is a common technique for handling imbalanced datasets). We won't use these weights during training for simplicity. All of these variables are NumPy arrays.