All that remains now is to define the loss associated with the graph in order to train it. Conveniently, TensorFlow offers a loss for training language models in `tf.contrib`. We need only make a call to `tf.contrib.seq2seq.sequence_loss` (Example 7-9). Underneath the hood, this loss turns out to be a form of perplexity.

Example 7-9. Add the sequence loss

```
# use the contrib sequence loss and average over the batches
loss = tf.contrib.seq2seq.sequence_loss(
   logits,
   input_.targets,
   tf.ones([batch_size, num_steps], dtype=tf.float32),
   average_across_timesteps=False,
   average_across_batch=True
)
# update the cost variables
self._cost = cost = tf.reduce_sum(loss)
```

Perplexity

Perplexity is often used for language modeling challenges. It is a variant of the binary cross-entropy that is useful for measuring how close the learned distribution is to the true distribution of data. Empirically, perplexity has proven useful for many language modeling challenges and we make use of it here in that capacity (since the `sequence_loss` just implements perplexity specialized to sequences inside).

We can then train this graph using a standard gradient descent method. We leave out some of the messy details of the underlying code, but suggest you check GitHub if curious. Evaluating the quality of the trained model turns out to be straightforward as well, since the perplexity is used both as the training loss and the evaluation metric. As a result, we can simply display `self._cost` to gauge how the model is training. We encourage you to train the model for yourself!

Challenge for the Reader

Try lowering perplexity on the Penn Treebank by experimenting with different model architectures. Note that these experiments might be time-consuming without a GPU.

Review

This chapter introduced you to recurrent neural networks (RNNs), a powerful architecture for learning on sequential data. RNNs are capable of learning the underlying evolution rule that governs a sequence of data. While RNNs can be used for modeling simple time-series, they are most powerful when modeling complex sequential data such as speech and natural language.

We introduced you to a number of RNN variants such as LSTMs and GRUs, which perform better on data with complex long-range interactions, and also took a brief detour to discuss the exciting prospect of Neural Turing machines. We ended the chapter with an in-depth case study that applied LSTMs to model the Penn Treebank.

In Chapter 8, we will introduce you to reinforcement learning, a powerful technique for learning to play games.

CHAPTER 8
Reinforcement Learning

The learning techniques we've covered so far in this book fall into the categories of supervised or unsupervised learning. In both cases, solving a given problem requires a data scientist to design a deep architecture that handles and processes input data and to connect the output of the architecture to a loss function suitable for the problem at hand. This framework is widely applicable, but not all applications fall neatly into this style of thinking. Let's consider the challenge of training a machine learning model to win a game of chess. It seems reasonable to process the board as spatial input using a convolutional network, but what would the loss entail? None of our standard loss functions such as cross-entropy or L^2 loss quite seem to apply.

Reinforcement learning provides a mathematical framework well suited to solving games. The central mathematical concept is that of the *Markov decision process*, a tool for modeling AI agents that interact with *environments* that offer *rewards* upon completion of certain *actions*. This framework proves to be flexible and general, and has found a number of applications in recent years. It's worth noting that reinforcement learning as a field is quite mature and has existed in recognizable form since the 1970s. However, until recently, most reinforcement learning systems were only capable of solving toy problems. Recent work has revealed that these limitations were likely due to the lack of sophisticated data intake mechanisms; hand-engineered features for many games or robotic environments often did not suffice. Deep representation extractions trained end-to-end on modern hardware seem to break through the barriers of earlier reinforcement learning systems and have achieved notable results in recent years.

Arguably, the first breakthrough in deep reinforcement learning was on ATARI arcade games. ATARI arcade games were traditionally played in video game arcades and offered users simple games that don't typically require sophisticated strategizing but might require good reflexes. Figure 8-1 shows a screenshot from the popular