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