For example, the `embedding_rnn_seq2seq()` function creates a simple Encoder–Decoder model that automatically takes care of word embeddings for you, just like the one represented in Figure 14-15. This code will likely be updated quickly to use the new `tf.nn.seq2seq` module.

You now have all the tools you need to understand the sequence-to-sequence tutorial's implementation. Check it out and train your own English-to-French translator!

Exercises

1. Can you think of a few applications for a sequence-to-sequence RNN? What about a sequence-to-vector RNN? And a vector-to-sequence RNN?
2. Why do people use encoder–decoder RNNs rather than plain sequence-to-sequence RNNs for automatic translation?
3. How could you combine a convolutional neural network with an RNN to classify videos?
4. What are the advantages of building an RNN using `dynamic_rnn()` rather than `static_rnn()`?
5. How can you deal with variable-length input sequences? What about variable-length output sequences?
6. What is a common way to distribute training and execution of a deep RNN across multiple GPUs?
7. *Embedded Reber grammars* were used by Hochreiter and Schmidhuber in their paper about LSTMs. They are artificial grammars that produce strings such as "BPBTSXXVPSEPE." Check out Jenny Orr's nice introduction to this topic. Choose a particular embedded Reber grammar (such as the one represented on Jenny Orr's page), then train an RNN to identify whether a string respects that grammar or not. You will first need to write a function capable of generating a training batch containing about 50% strings that respect the grammar, and 50% that don't.
8. Tackle the "How much did it rain? II" Kaggle competition. This is a time series prediction task: you are given snapshots of polarimetric radar values and asked to predict the hourly rain gauge total. Luis Andre Dutra e Silva's interview gives some interesting insights into the techniques he used to reach second place in the competition. In particular, he used an RNN composed of two LSTM layers.
9. Go through TensorFlow's Word2Vec tutorial to create word embeddings, and then go through the Seq2Seq tutorial to train an English-to-French translation system.

Solutions to these exercises are available in Appendix A.

CHAPTER 15
Autoencoders

Autoencoders are artificial neural networks capable of learning efficient representations of the input data, called *codings*, without any supervision (i.e., the training set is unlabeled). These codings typically have a much lower dimensionality than the input data, making autoencoders useful for dimensionality reduction (see Chapter 8). More importantly, autoencoders act as powerful feature detectors, and they can be used for unsupervised pretraining of deep neural networks (as we discussed in Chapter 11). Lastly, they are capable of randomly generating new data that looks very similar to the training data; this is called a *generative model*. For example, you could train an autoencoder on pictures of faces, and it would then be able to generate new faces.

Surprisingly, autoencoders work by simply learning to copy their inputs to their outputs. This may sound like a trivial task, but we will see that constraining the network in various ways can make it rather difficult. For example, you can limit the size of the internal representation, or you can add noise to the inputs and train the network to recover the original inputs. These constraints prevent the autoencoder from trivially copying the inputs directly to the outputs, which forces it to learn efficient ways of representing the data. In short, the codings are byproducts of the autoencoder's attempt to learn the identity function under some constraints.

In this chapter we will explain in more depth how autoencoders work, what types of constraints can be imposed, and how to implement them using TensorFlow, whether it is for dimensionality reduction, feature extraction, unsupervised pretraining, or as generative models.