Embeddings are also useful for representing categorical attributes that can take on a large number of different values, especially when there are complex similarities between values. For example, consider professions, hobbies, dishes, species, brands, and so on.

You now have almost all the tools you need to implement a machine translation system. Let's look at this now.

An Encoder–Decoder Network for Machine Translation

Let's take a look at a simple machine translation model[10] that will translate English sentences to French (see Figure 14-15).

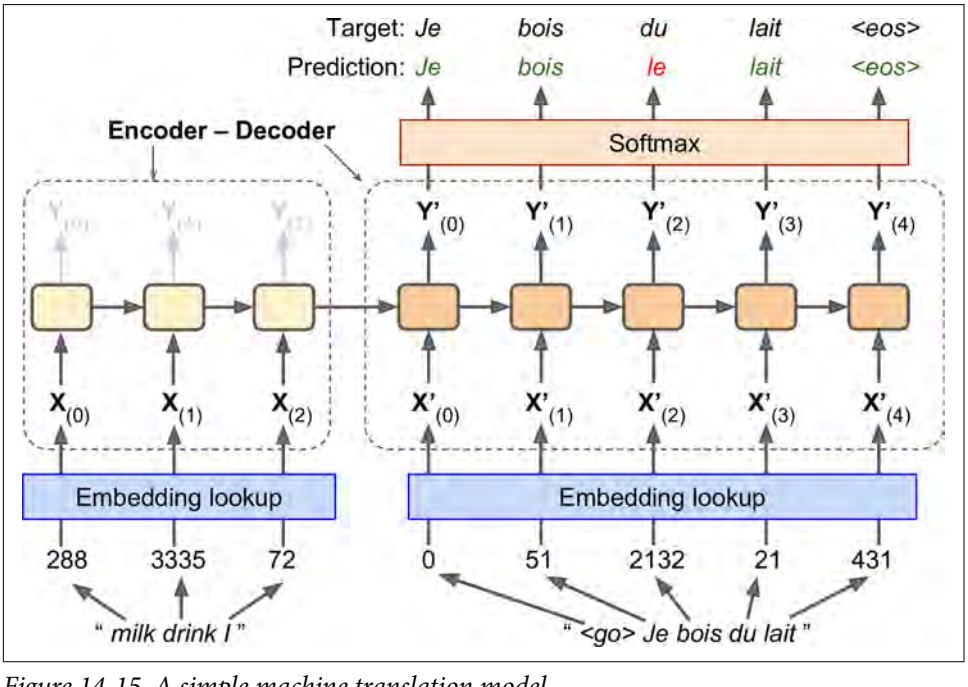

Figure 14-15. A simple machine translation model

The English sentences are fed to the encoder, and the decoder outputs the French translations. Note that the French translations are also used as inputs to the decoder, but pushed back by one step. In other words, the decoder is given as input the word that it *should* have output at the previous step (regardless of what it actually output). For the very first word, it is given a token that represents the beginning of the sen-

10 "Sequence to Sequence learning with Neural Networks," I. Sutskever et al. (2014).

tence (e.g., "<go>"). The decoder is expected to end the sentence with an end-of-sequence (EOS) token (e.g., "<eos>").

Note that the English sentences are reversed before they are fed to the encoder. For example "I drink milk" is reversed to "milk drink I." This ensures that the beginning of the English sentence will be fed last to the encoder, which is useful because that's generally the first thing that the decoder needs to translate.

Each word is initially represented by a simple integer identifier (e.g., 288 for the word "milk"). Next, an embedding lookup returns the word embedding (as explained earlier, this is a dense, fairly low-dimensional vector). These word embeddings are what is actually fed to the encoder and the decoder.

At each step, the decoder outputs a score for each word in the output vocabulary (i.e., French), and then the Softmax layer turns these scores into probabilities. For example, at the first step the word "Je" may have a probability of 20%, "Tu" may have a probability of 1%, and so on. The word with the highest probability is output. This is very much like a regular classification task, so you can train the model using the `softmax_cross_entropy_with_logits()` function.

Note that at inference time (after training), you will not have the target sentence to feed to the decoder. Instead, simply feed the decoder the word that it output at the previous step, as shown in Figure 14-16 (this will require an embedding lookup that is not shown on the diagram).

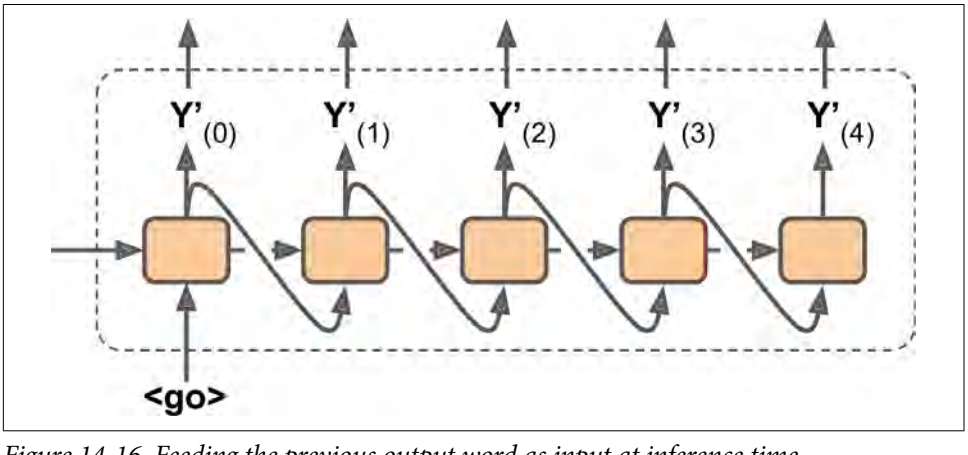

Figure 14-16. Feeding the previous output word as input at inference time

Okay, now you have the big picture. However, if you go through TensorFlow's sequence-to-sequence tutorial and you look at the code in *rnn/translate/seq2seq_model.py* (in the TensorFlow models), you will notice a few important differences:

- First, so far we have assumed that all input sequences (to the encoder and to the decoder) have a constant length. But obviously sentence lengths may vary. There are several ways that this can be handled—for example, using the `sequence_length` argument to the `static_rnn()` or `dynamic_rnn()` functions to specify each sentence's length (as discussed earlier). However, another approach is used in the tutorial (presumably for performance reasons): sentences are grouped into buckets of similar lengths (e.g., a bucket for the 1- to 6-word sentences, another for the 7- to 12-word sentences, and so on[11]), and the shorter sentences are padded using a special padding token (e.g., "<pad>"). For example "I drink milk" becomes "<pad> <pad> <pad> milk drink I", and its translation becomes "Je bois du lait <eos> <pad>". Of course, we want to ignore any output past the EOS token. For this, the tutorial's implementation uses a `target_weights` vector. For example, for the target sentence "Je bois du lait <eos> <pad>", the weights would be set to `[1.0, 1.0, 1.0, 1.0, 1.0, 0.0]` (notice the weight 0.0 that corresponds to the padding token in the target sentence). Simply multiplying the losses by the target weights will zero out the losses that correspond to words past EOS tokens.
- Second, when the output vocabulary is large (which is the case here), outputting a probability for each and every possible word would be terribly slow. If the target vocabulary contains, say, 50,000 French words, then the decoder would output 50,000-dimensional vectors, and then computing the softmax function over such a large vector would be very computationally intensive. To avoid this, one solution is to let the decoder output much smaller vectors, such as 1,000-dimensional vectors, then use a sampling technique to estimate the loss without having to compute it over every single word in the target vocabulary. This *Sampled Softmax* technique was introduced in 2015 by Sébastien Jean et al.[12] In TensorFlow you can use the `sampled_softmax_loss()` function.
- Third, the tutorial's implementation uses an *attention mechanism* that lets the decoder peek into the input sequence. Attention augmented RNNs are beyond the scope of this book, but if you are interested there are helpful papers about machine translation,[13] machine reading,[14] and image captions[15] using attention.
- Finally, the tutorial's implementation makes use of the `tf.nn.legacy_seq2seq` module, which provides tools to build various Encoder–Decoder models easily.

11 The bucket sizes used in the tutorial are different.

12 "On Using Very Large Target Vocabulary for Neural Machine Translation," S. Jean et al. (2015).

13 "Neural Machine Translation by Jointly Learning to Align and Translate," D. Bahdanau et al. (2014).

14 "Long Short-Term Memory-Networks for Machine Reading," J. Cheng (2016).

15 "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention," K. Xu et al. (2015).

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.