Download from finelybook www.finelybook.com

it outputs a 0, the opposite happens. In other words, whenever a memory must be stored, the location where it will be stored is erased first. This is actually a frequent variant to the LSTM cell in and of itself.

- There is no output gate; the full state vector is output at every time step. However, there is a new gate controller that controls which part of the previous state will be shown to the main layer.

Equation 14-4 summarizes how to compute the cell's state at each time step for a single instance.

Equation 14-4. GRU computations

$$\begin{aligned}
\mathbf{z}_{(t)} &= \sigma\left(\mathbf{W}_{xz}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hz}{}^{T} \cdot \mathbf{h}_{(t-1)}\right)\\
\mathbf{r}_{(t)} &= \sigma\left(\mathbf{W}_{xr}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hr}{}^{T} \cdot \mathbf{h}_{(t-1)}\right)\\
\mathbf{g}_{(t)} &= \tanh\left(\mathbf{W}_{xg}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}{}^{T} \cdot \left(\mathbf{r}_{(t)} \otimes \mathbf{h}_{(t-1)}\right)\right)\\
\mathbf{h}_{(t)} &= \left(1 - \mathbf{z}_{(t)}\right) \otimes \tanh\left(\mathbf{W}_{xg}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{z}_{(t)} \otimes \mathbf{g}_{t}\right)
\end{aligned}$$

Creating a GRU cell in TensorFlow is trivial:

```
gru_cell = tf.contrib.rnn.GRUCell(num_units=n_neurons)
```

LSTM or GRU cells are one of the main reasons behind the success of RNNs in recent years, in particular for applications in *natural language processing* (NLP).

Natural Language Processing

Most of the state-of-the-art NLP applications, such as machine translation, automatic summarization, parsing, sentiment analysis, and more, are now based (at least in part) on RNNs. In this last section, we will take a quick look at what a machine translation model looks like. This topic is very well covered by TensorFlow's awesome Word2Vec and Seq2Seq tutorials, so you should definitely check them out.

Word Embeddings

Before we start, we need to choose a word representation. One option could be to represent each word using a one-hot vector. Suppose your vocabulary contains 50,000 words, then the nth word would be represented as a 50,000-dimensional vector, full of 0s except for a 1 at the nth position. However, with such a large vocabulary, this sparse representation would not be efficient at all. Ideally, you want similar words to have similar representations, making it easy for the model to generalize what it learns about a word to all similar words. For example, if the model is told that "I drink milk" is a valid sentence, and if it knows that "milk" is close to "water" but far from "shoes,"

Download from finelybook www.finelybook.com
then it will know that "I drink water" is probably a valid sentence as well, while "I drink shoes" is probably not. But how can you come up with such a meaningful representation?

The most common solution is to represent each word in the vocabulary using a fairly small and dense vector (e.g., 150 dimensions), called an *embedding*, and just let the neural network learn a good embedding for each word during training. At the beginning of training, embeddings are simply chosen randomly, but during training, backpropagation automatically moves the embeddings around in a way that helps the neural network perform its task. Typically this means that similar words will gradually cluster close to one another, and even end up organized in a rather meaningful way. For example, embeddings may end up placed along various axes that represent gender, singular/plural, adjective/noun, and so on. The result can be truly amazing.[9]

In TensorFlow, you first need to create the variable representing the embeddings for every word in your vocabulary (initialized randomly):

```
vocabulary_size = 50000
embedding_size = 150
embeddings = tf.Variable(
    tf.random_uniform([vocabulary_size, embedding_size], -1.0, 1.0))
```

Now suppose you want to feed the sentence "I drink milk" to your neural network. You should first preprocess the sentence and break it into a list of known words. For example you may remove unnecessary characters, replace unknown words by a predefined token word such as "[UNK]", replace numerical values by "[NUM]", replace URLs by "[URL]", and so on. Once you have a list of known words, you can look up each word's integer identifier (from 0 to 49999) in a dictionary, for example [72, 3335, 288]. At that point, you are ready to feed these word identifiers to TensorFlow using a placeholder, and apply the `embedding_lookup()` function to get the corresponding embeddings:

```
train_inputs = tf.placeholder(tf.int32, shape=[None])  # from ids...
embed = tf.nn.embedding_lookup(embeddings, train_inputs)  # ...to embeddings
```

Once your model has learned good word embeddings, they can actually be reused fairly efficiently in any NLP application: after all, "milk" is still close to "water" and far from "shoes" no matter what your application is. In fact, instead of training your own word embeddings, you may want to download pretrained word embeddings. Just like when reusing pretrained layers (see Chapter 11), you can choose to freeze the pretrained embeddings (e.g., creating the `embeddings` variable using `trainable=False`) or let backpropagation tweak them for your application. The first option will speed up training, but the second may lead to slightly higher performance.

9 For more details, check out Christopher Olah's great post, or Sebastian Ruder's series of posts.

Download from finelybook www.finelybook.com

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).

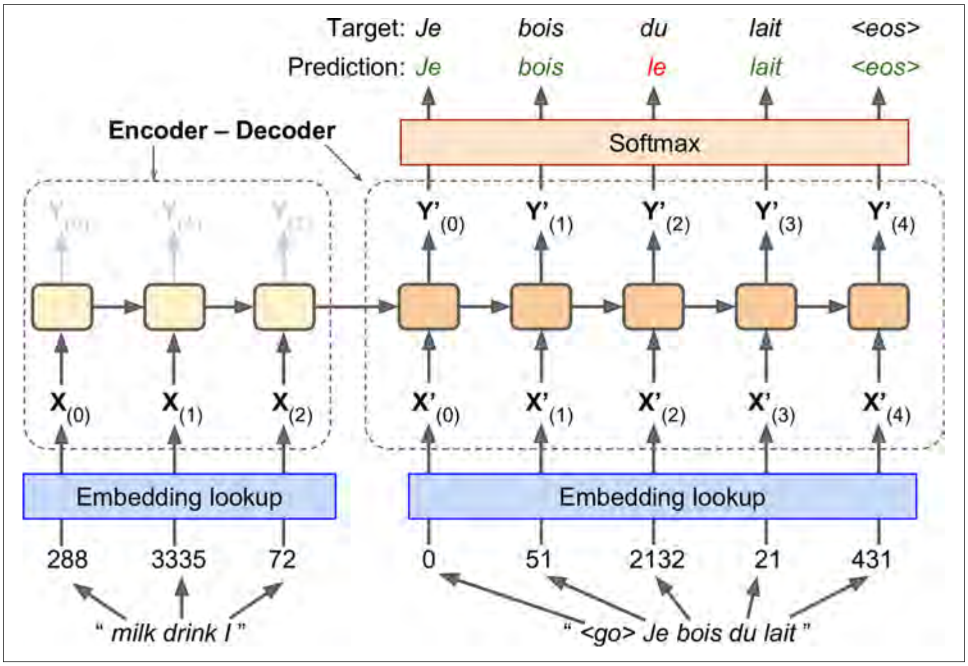

Figure 14-15. A simple machine translation model

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).