There are many other variants of the LSTM cell. One particularly popular variant is the GRU cell, which we will look at now.

GRU Cell

The *Gated Recurrent Unit* (GRU) cell (see Figure 14-14) was proposed by Kyunghyun Cho et al. in a 2014 paper[7] that also introduced the Encoder–Decoder network we mentioned earlier.

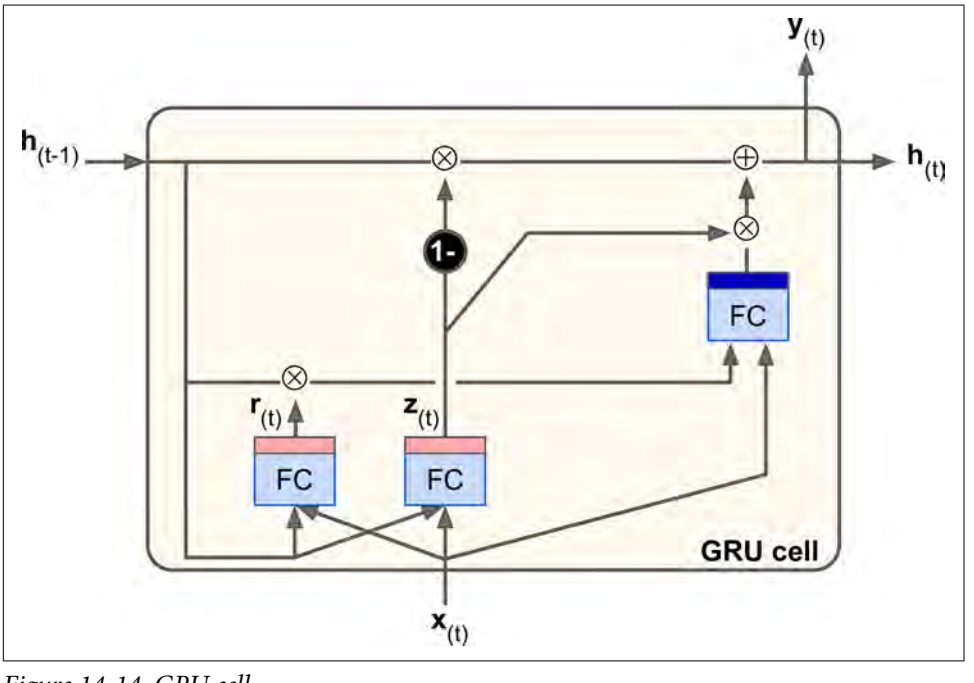

Figure 14-14. GRU cell

The GRU cell is a simplified version of the LSTM cell, and it seems to perform just as well[8] (which explains its growing popularity). The main simplifications are:

- Both state vectors are merged into a single vector $\mathbf{h}_{(t)}$.
- A single gate controller controls both the forget gate and the input gate. If the gate controller outputs a 1, the input gate is open and the forget gate is closed. If

7 "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation," K. Cho et al. (2014).

8 A 2015 paper by Klaus Greff et al., "LSTM: A Search Space Odyssey," seems to show that all LSTM variants perform roughly the same.

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 n^{th} word would be represented as a 50,000-dimensional vector, full of 0s except for a 1 at the n^{th} 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,"