LSTM Cell

The *Long Short-Term Memory* (LSTM) cell was proposed in 1997[3] by Sepp Hochreiter and Jürgen Schmidhuber, and it was gradually improved over the years by several researchers, such as Alex Graves, Haşim Sak,[4] Wojciech Zaremba,[5] and many more. If you consider the LSTM cell as a black box, it can be used very much like a basic cell, except it will perform much better; training will converge faster and it will detect long-term dependencies in the data. In TensorFlow, you can simply use a `BasicLSTMCell` instead of a `BasicRNNCell`:

```
lstm_cell = tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons)
```

LSTM cells manage two state vectors, and for performance reasons they are kept separate by default. You can change this default behavior by setting `state_is_tuple=False` when creating the `BasicLSTMCell`.

So how does an LSTM cell work? The architecture of a basic LSTM cell is shown in Figure 14-13.

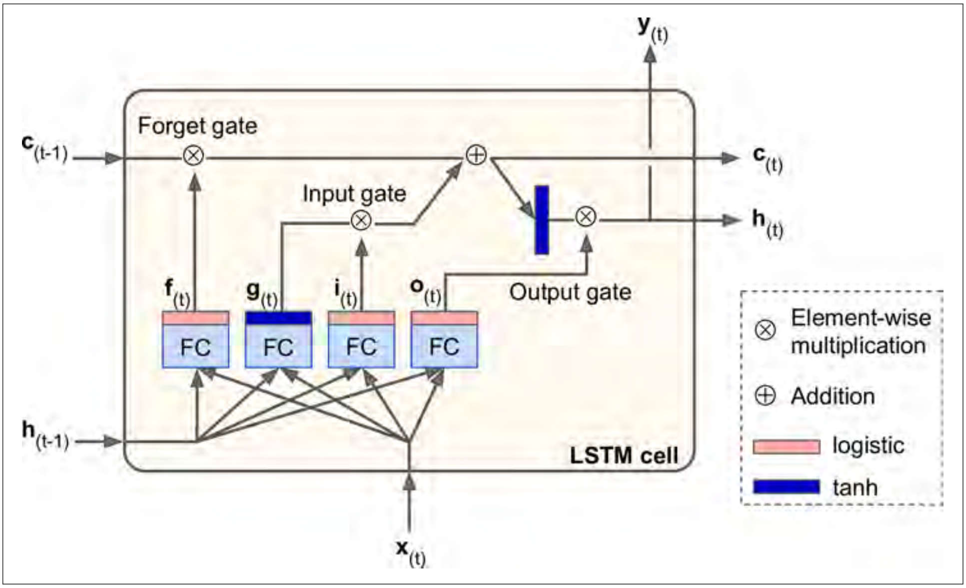

Figure 14-13. LSTM cell

3 "Long Short-Term Memory," S. Hochreiter and J. Schmidhuber (1997).

4 "Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling," H. Sak et al. (2014).

5 "Recurrent Neural Network Regularization," W. Zaremba et al. (2015).

If you don't look at what's inside the box, the LSTM cell looks exactly like a regular cell, except that its state is split in two vectors: $\mathbf{h}_{(t)}$ and $\mathbf{c}_{(t)}$ ("c" stands for "cell"). You can think of $\mathbf{h}_{(t)}$ as the short-term state and $\mathbf{c}_{(t)}$ as the long-term state.

Now let's open the box! The key idea is that the network can learn what to store in the long-term state, what to throw away, and what to read from it. As the long-term state $\mathbf{c}_{(t-1)}$ traverses the network from left to right, you can see that it first goes through a *forget gate*, dropping some memories, and then it adds some new memories via the addition operation (which adds the memories that were selected by an *input gate*). The result $\mathbf{c}_{(t)}$ is sent straight out, without any further transformation. So, at each time step, some memories are dropped and some memories are added. Moreover, after the addition operation, the long-term state is copied and passed through the tanh function, and then the result is filtered by the *output gate*. This produces the short-term state $\mathbf{h}_{(t)}$ (which is equal to the cell's output for this time step $\mathbf{y}_{(t)}$). Now let's look at where new memories come from and how the gates work.

First, the current input vector $\mathbf{x}_{(t)}$ and the previous short-term state $\mathbf{h}_{(t-1)}$ are fed to four different fully connected layers. They all serve a different purpose:

- The main layer is the one that outputs $\mathbf{g}_{(t)}$. It has the usual role of analyzing the current inputs $\mathbf{x}_{(t)}$ and the previous (short-term) state $\mathbf{h}_{(t-1)}$. In a basic cell, there is nothing else than this layer, and its output goes straight out to $\mathbf{y}_{(t)}$ and $\mathbf{h}_{(t)}$. In contrast, in an LSTM cell this layer's output does not go straight out, but instead it is partially stored in the long-term state.
- The three other layers are *gate controllers*. Since they use the logistic activation function, their outputs range from 0 to 1. As you can see, their outputs are fed to element-wise multiplication operations, so if they output 0s, they close the gate, and if they output 1s, they open it. Specifically:
  — The *forget gate* (controlled by $\mathbf{f}_{(t)}$) controls which parts of the long-term state should be erased.
  — The *input gate* (controlled by $\mathbf{i}_{(t)}$) controls which parts of $\mathbf{g}_{(t)}$ should be added to the long-term state (this is why we said it was only "partially stored").
  — Finally, the *output gate* (controlled by $\mathbf{o}_{(t)}$) controls which parts of the long-term state should be read and output at this time step (both to $\mathbf{h}_{(t)}$) and $\mathbf{y}_{(t)}$.

In short, an LSTM cell can learn to recognize an important input (that's the role of the input gate), store it in the long-term state, learn to preserve it for as long as it is needed (that's the role of the forget gate), and learn to extract it whenever it is needed. This explains why they have been amazingly successful at capturing long-term patterns in time series, long texts, audio recordings, and more.

Equation 14-3 summarizes how to compute the cell's long-term state, its short-term state, and its output at each time step for a single instance (the equations for a whole mini-batch are very similar).

Equation 14-3. LSTM computations

$$
\begin{aligned}
\mathbf{i}_{(t)} &= \sigma\left(\mathbf{W}_{xi}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hi}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_{i}\right)\\
\mathbf{f}_{(t)} &= \sigma\left(\mathbf{W}_{xf}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hf}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_{f}\right)\\
\mathbf{o}_{(t)} &= \sigma\left(\mathbf{W}_{xo}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{ho}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_{o}\right)\\
\mathbf{g}_{(t)} &= \tanh\left(\mathbf{W}_{xg}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_{g}\right)\\
\mathbf{c}_{(t)} &= \mathbf{f}_{(t)} \otimes \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \otimes \mathbf{g}_{(t)}\\
\mathbf{y}_{(t)} &= \mathbf{h}_{(t)} = \mathbf{o}_{(t)} \otimes \tanh\left(\mathbf{c}_{(t)}\right)
\end{aligned}
$$

- $\mathbf{W}_{xi}$, $\mathbf{W}_{xf}$, $\mathbf{W}_{xo}$, $\mathbf{W}_{xg}$ are the weight matrices of each of the four layers for their connection to the input vector $\mathbf{x}_{(t)}$.
- $\mathbf{W}_{hi}$, $\mathbf{W}_{hf}$, $\mathbf{W}_{ho}$, and $\mathbf{W}_{hg}$ are the weight matrices of each of the four layers for their connection to the previous short-term state $\mathbf{h}_{(t-1)}$.
- $\mathbf{b}_{i}$, $\mathbf{b}_{f}$, $\mathbf{b}_{o}$, and $\mathbf{b}_{g}$ are the bias terms for each of the four layers. Note that TensorFlow initializes $\mathbf{b}_{f}$ to a vector full of 1s instead of 0s. This prevents forgetting everything at the beginning of training.

Peephole Connections

In a basic LSTM cell, the gate controllers can look only at the input $\mathbf{x}_{(t)}$ and the previous short-term state $\mathbf{h}_{(t-1)}$. It may be a good idea to give them a bit more context by letting them peek at the long-term state as well. This idea was proposed by Felix Gers and Jürgen Schmidhuber in 2000.[6] They proposed an LSTM variant with extra connections called *peephole connections*: the previous long-term state $\mathbf{c}_{(t-1)}$ is added as an input to the controllers of the forget gate and the input gate, and the current long-term state $\mathbf{c}_{(t)}$ is added as input to the controller of the output gate.

To implement peephole connections in TensorFlow, you must use the `LSTMCell` instead of the `BasicLSTMCell` and set `use_peepholes=True`:

```
lstm_cell = tf.contrib.rnn.LSTMCell(num_units=n_neurons, use_peepholes=True)
```

6 "Recurrent Nets that Time and Count," F. Gers and J. Schmidhuber (2000).