CHAPTER 14
Recurrent Neural Networks

The batter hits the ball. You immediately start running, anticipating the ball's trajectory. You track it and adapt your movements, and finally catch it (under a thunder of applause). Predicting the future is what you do all the time, whether you are finishing a friend's sentence or anticipating the smell of coffee at breakfast. In this chapter, we are going to discuss *recurrent neural networks* (RNN), a class of nets that can predict the future (well, up to a point, of course). They can analyze *time series* data such as stock prices, and tell you when to buy or sell. In autonomous driving systems, they can anticipate car trajectories and help avoid accidents. More generally, they can work on *sequences* of arbitrary lengths, rather than on fixed-sized inputs like all the nets we have discussed so far. For example, they can take sentences, documents, or audio samples as input, making them extremely useful for natural language processing (NLP) systems such as automatic translation, speech-to-text, or *sentiment analysis* (e.g., reading movie reviews and extracting the rater's feeling about the movie).

Moreover, RNNs' ability to anticipate also makes them capable of surprising creativity. You can ask them to predict which are the most likely next notes in a melody, then randomly pick one of these notes and play it. Then ask the net for the next most likely notes, play it, and repeat the process again and again. Before you know it, your net will compose a melody such as the one produced by Google's Magenta project. Similarly, RNNs can generate sentences, image captions, and much more. The result is not exactly Shakespeare or Mozart yet, but who knows what they will produce a few years from now?

In this chapter, we will look at the fundamental concepts underlying RNNs, the main problem they face (namely, vanishing/exploding gradients, discussed in Chapter 11), and the solutions widely used to fight it: LSTM and GRU cells. Along the way, as always, we will show how to implement RNNs using TensorFlow. Finally, we will take a look at the architecture of a machine translation system.

Recurrent Neurons

Up to now we have mostly looked at feedforward neural networks, where the activations flow only in one direction, from the input layer to the output layer (except for a few networks in Appendix E). A recurrent neural network looks very much like a feedforward neural network, except it also has connections pointing backward. Let's look at the simplest possible RNN, composed of just one neuron receiving inputs, producing an output, and sending that output back to itself, as shown in Figure 14-1 (left). At each *time step t* (also called a *frame*), this *recurrent neuron* receives the inputs $\mathbf{x}_{(t)}$ as well as its own output from the previous time step, $y_{(t-1)}$. We can represent this tiny network against the time axis, as shown in Figure 14-1 (right). This is called *unrolling the network through time*.

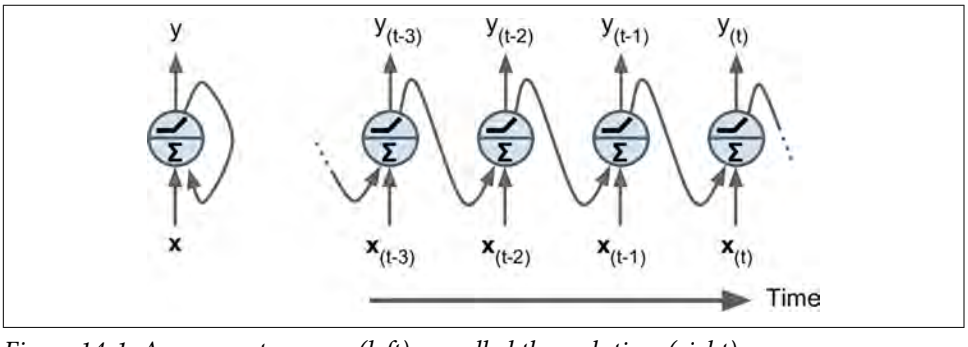

Figure 14-1. A recurrent neuron (left), unrolled through time (right)

You can easily create a layer of recurrent neurons. At each time step *t*, every neuron receives both the input vector $\mathbf{x}_{(t)}$ and the output vector from the previous time step $\mathbf{y}_{(t-1)}$, as shown in Figure 14-2. Note that both the inputs and outputs are vectors now (when there was just a single neuron, the output was a scalar).

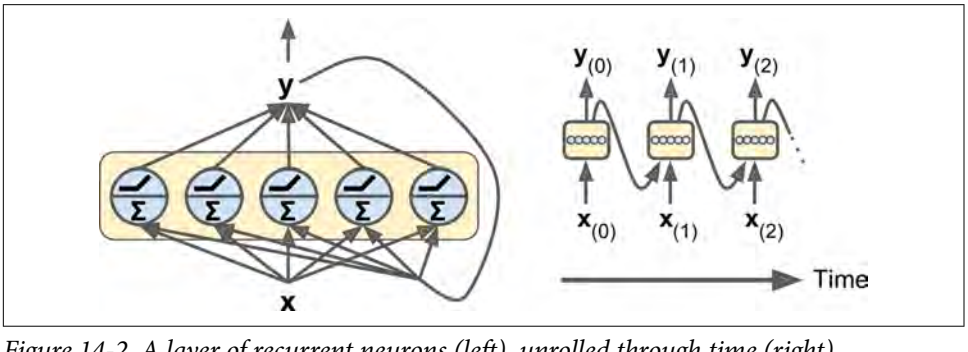

Figure 14-2. A layer of recurrent neurons (left), unrolled through time (right)

Each recurrent neuron has two sets of weights: one for the inputs $\mathbf{x}_{(t)}$ and the other for the outputs of the previous time step, $\mathbf{y}_{(t-1)}$. Let's call these weight vectors $\mathbf{w}_x$ and $\mathbf{w}_y$.