The case study for this chapter trains a recurrent neural network language model on the Penn Treebank corpus, a body of sentences extracted from *Wall Street Journal* articles. This tutorial was adapted from the TensorFlow official documentation tutorial on recurrent networks. (We encourage you to access the original tutorial on the TensorFlow website if you're curious about the changes we've made.) As always, we recommend that you follow along with the code in the GitHub repo associated with this book.

Overview of Recurrent Architectures

Recurrent architectures are useful for modeling very complex time varying datasets. Time varying datasets are traditionally called *time-series*. Figure 7-1 displays a number of time-series datasets.

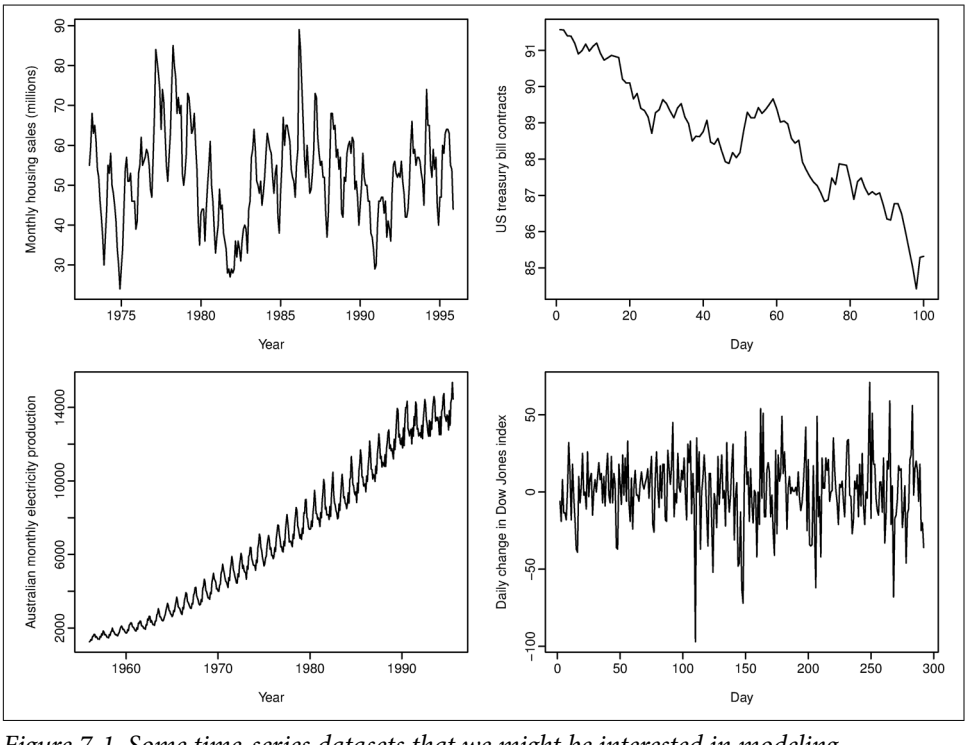

Figure 7-1. Some time-series datasets that we might be interested in modeling.

In time-series modeling, we design learning systems that are capable of learning the evolution rule that models how the future of the system at hand evolves depending on the past. Mathematically, let's suppose that at each time step, we receive a datapoint x_t where t is the current time. Then, time-series methods seek to learn some function f such that

$$x_{t+1} = f(x_1, \cdots, x_t)$$

The idea is that f encodes the underlying dynamics of the system well and learning it from data would enable a learning system to predict the future of the system at hand. In practice, it's too unwieldy to learn a function that depends on all past inputs, so learning systems often assume that all information about last datapoints $x_1, \cdots, x_{t-1}$ can be encoded into some fixed vector h_t. Then, the update equation simplifies into the format

$$x_{t+1}, h_{t+1} = f(x_t, h_t)$$

Notice that we assume that the same function f here applies for all timesteps t. That is, we assume the time-series to be *stationary* (Figure 7-2). This assumption is broken for many systems, notably including the stock market where today's rules need not hold tomorrow.

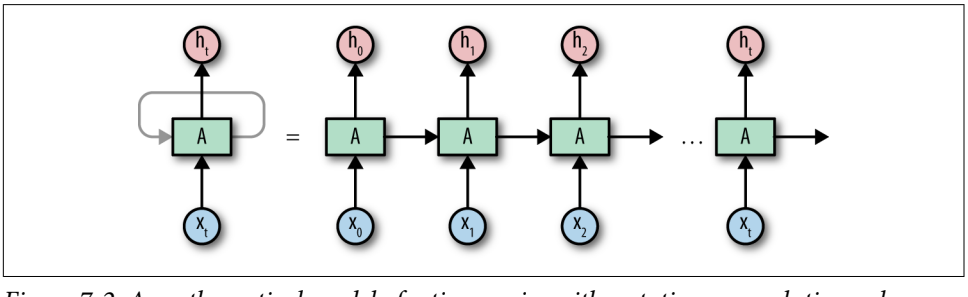

Figure 7-2. A mathematical model of a time-series with a stationary evolution rule. Recall that a stationary system is one whose underlying dynamics don't shift over time.

What does this equation have to do with recurrent neural nets? The basic answer derives from the universal approximation theorem that we introduced in Chapter 4. The function f can be arbitrarily complex, so using a fully connected deep network to learn f seems like a reasonable idea. This intuition essentially defines the RNN. A simple recurrent network can be viewed as a fully connected network that is applied repeatedly to each time step of the data.

In fact, recurrent neural networks really become interesting only for complex high-dimensional time-series. For simpler systems, there are classical signal processing time-series methods that often do a good job of modeling time dynamics. However, for complex systems, such as speech (see the speech spectrogram in Figure 7-3), RNNs come into their own and offer capabilities that other methods can't.

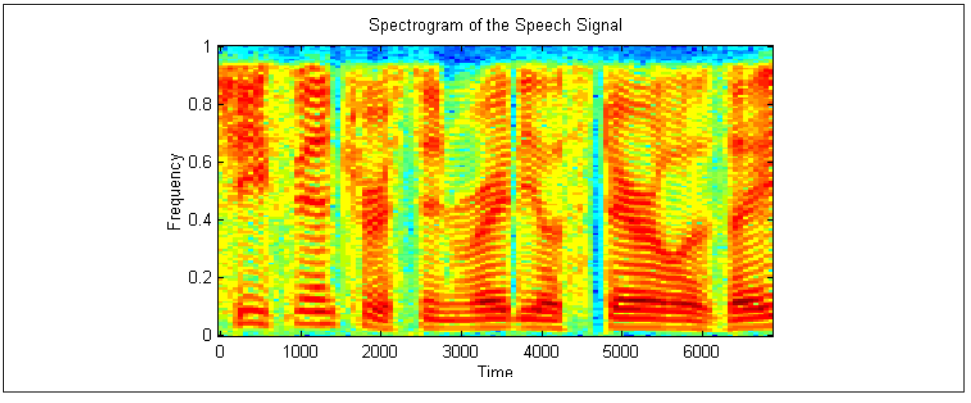

Figure 7-3. A speech spectrogram representing the frequencies found in a speech sample.

Recurrent Cells

Gradient Instability

Recurrent networks tend to degrade signal over time. Think of it as attenuating a signal by a multiplicative factor at each timestep. As a result, after 50 timesteps, the signal is quite degraded.

As a result of this instability, it has been challenging to train recurrent neural networks on longer time-series. A number of methods have arisen to combat this instability, which we will discuss in the remainder of this section.

There are a number of elaborations on the concept of a simple recurrent neural network that have proven significantly more successful in practical applications. In this section, we will briefly review some of these variations.

Long Short-Term Memory (LSTM)

Part of the challenge with the standard recurrent cell is that signals from the distant past attenuate rapidly. As a result, RNNs can fail to learn models of complex dependencies. This failure becomes particularly notable in applications such as language modeling, where words can have complex dependencies on earlier phrases.

One potential solution to this issue is to allow states from the past to pass through unmodified. The long short-term memory (LSTM) architecture proposes a mechanism to allow past state to pass through to the present with minimal modifications. Empirically using an LSTM "cell" (shown in Figure 7-4) seems to offer superior learning performance when compared to simple recurrent neural networks using fully connected layers.