

CHAPTER 15

Autoencoders

Autoencoders are artificial neural networks capable of learning efficient representations of the input data, called *codings*, without any supervision (i.e., the training set is unlabeled). These codings typically have a much lower dimensionality than the input data, making autoencoders useful for dimensionality reduction (see [Chapter 8](#)). More importantly, autoencoders act as powerful feature detectors, and they can be used for unsupervised pretraining of deep neural networks (as we discussed in [Chapter 11](#)). Lastly, they are capable of randomly generating new data that looks very similar to the training data; this is called a *generative model*. For example, you could train an autoencoder on pictures of faces, and it would then be able to generate new faces.

Surprisingly, autoencoders work by simply learning to copy their inputs to their outputs. This may sound like a trivial task, but we will see that constraining the network in various ways can make it rather difficult. For example, you can limit the size of the internal representation, or you can add noise to the inputs and train the network to recover the original inputs. These constraints prevent the autoencoder from trivially copying the inputs directly to the outputs, which forces it to learn efficient ways of representing the data. In short, the codings are byproducts of the autoencoder's attempt to learn the identity function under some constraints.

In this chapter we will explain in more depth how autoencoders work, what types of constraints can be imposed, and how to implement them using TensorFlow, whether it is for dimensionality reduction, feature extraction, unsupervised pretraining, or as generative models.

Efficient Data Representations

Which of the following number sequences do you find the easiest to memorize?

- 40, 27, 25, 36, 81, 57, 10, 73, 19, 68
- 50, 25, 76, 38, 19, 58, 29, 88, 44, 22, 11, 34, 17, 52, 26, 13, 40, 20

At first glance, it would seem that the first sequence should be easier, since it is much shorter. However, if you look carefully at the second sequence, you may notice that it follows two simple rules: even numbers are followed by their half, and odd numbers are followed by their triple plus one (this is a famous sequence known as the *hailstone sequence*). Once you notice this pattern, the second sequence becomes much easier to memorize than the first because you only need to memorize the two rules, the first number, and the length of the sequence. Note that if you could quickly and easily memorize very long sequences, you would not care much about the existence of a pattern in the second sequence. You would just learn every number by heart, and that would be that. It is the fact that it is hard to memorize long sequences that makes it useful to recognize patterns, and hopefully this clarifies why constraining an autoencoder during training pushes it to discover and exploit patterns in the data.

The relationship between memory, perception, and pattern matching was **famously studied by William Chase and Herbert Simon in the early 1970s**.¹ They observed that expert chess players were able to memorize the positions of all the pieces in a game by looking at the board for just 5 seconds, a task that most people would find impossible. However, this was only the case when the pieces were placed in realistic positions (from actual games), not when the pieces were placed randomly. Chess experts don't have a much better memory than you and I, they just see chess patterns more easily thanks to their experience with the game. Noticing patterns helps them store information efficiently.

Just like the chess players in this memory experiment, an autoencoder looks at the inputs, converts them to an efficient internal representation, and then spits out something that (hopefully) looks very close to the inputs. An autoencoder is always composed of two parts: an *encoder* (or *recognition network*) that converts the inputs to an internal representation, followed by a *decoder* (or *generative network*) that converts the internal representation to the outputs (see **Figure 15-1**).

As you can see, an autoencoder typically has the same architecture as a Multi-Layer Perceptron (MLP; see **Chapter 10**), except that the number of neurons in the output layer must be equal to the number of inputs. In this example, there is just one hidden

¹ "Perception in chess," W. Chase and H. Simon (1973).