Stacked convolutional autoencoders[9]
: Autoencoders that learn to extract visual features by reconstructing images processed through convolutional layers.

Generative stochastic network (GSN)[10]
: A generalization of denoising autoencoders, with the added capability to generate data.

Winner-take-all (WTA) autoencoder[11]
: During training, after computing the activations of all the neurons in the coding layer, only the top *k*% activations for each neuron over the training batch are preserved, and the rest are set to zero. Naturally this leads to sparse codings. Moreover, a similar WTA approach can be used to produce sparse convolutional autoencoders.

Adversarial autoencoders[12]
: One network is trained to reproduce its inputs, and at the same time another is trained to find inputs that the first network is unable to properly reconstruct. This pushes the first autoencoder to learn robust codings.

Exercises

1. What are the main tasks that autoencoders are used for?
2. Suppose you want to train a classifier and you have plenty of unlabeled training data, but only a few thousand labeled instances. How can autoencoders help? How would you proceed?
3. If an autoencoder perfectly reconstructs the inputs, is it necessarily a good autoencoder? How can you evaluate the performance of an autoencoder?
4. What are undercomplete and overcomplete autoencoders? What is the main risk of an excessively undercomplete autoencoder? What about the main risk of an overcomplete autoencoder?
5. How do you tie weights in a stacked autoencoder? What is the point of doing so?
6. What is a common technique to visualize features learned by the lower layer of a stacked autoencoder? What about higher layers?
7. What is a generative model? Can you name a type of generative autoencoder?

9 "Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction," J. Masci et al. (2011).

10 "GSNs: Generative Stochastic Networks," G. Alain et al. (2015).

11 "Winner-Take-All Autoencoders," A. Makhzani and B. Frey (2015).

12 "Adversarial Autoencoders," A. Makhzani et al. (2016).

8. Let's use a denoising autoencoder to pretrain an image classifier:

   - You can use MNIST (simplest), or another large set of images such as CIFAR10 if you want a bigger challenge. If you choose CIFAR10, you need to write code to load batches of images for training. If you want to skip this part, TensorFlow's model zoo contains tools to do just that.
   - Split the dataset into a training set and a test set. Train a deep denoising autoencoder on the full training set.
   - Check that the images are fairly well reconstructed, and visualize the low-level features. Visualize the images that most activate each neuron in the coding layer.
   - Build a classification deep neural network, reusing the lower layers of the autoencoder. Train it using only 10% of the training set. Can you get it to perform as well as the same classifier trained on the full training set?

9. *Semantic hashing*, introduced in 2008 by Ruslan Salakhutdinov and Geoffrey Hinton,[13] is a technique used for efficient *information retrieval*: a document (e.g., an image) is passed through a system, typically a neural network, which outputs a fairly low-dimensional binary vector (e.g., 30 bits). Two similar documents are likely to have identical or very similar hashes. By indexing each document using its hash, it is possible to retrieve many documents similar to a particular document almost instantly, even if there are billions of documents: just compute the hash of the document and look up all documents with that same hash (or hashes differing by just one or two bits). Let's implement semantic hashing using a slightly tweaked stacked autoencoder:

   - Create a stacked autoencoder containing two hidden layers below the coding layer, and train it on the image dataset you used in the previous exercise. The coding layer should contain 30 neurons and use the logistic activation function to output values between 0 and 1. After training, to produce the hash of an image, you can simply run it through the autoencoder, take the output of the coding layer, and round every value to the closest integer (0 or 1).
   - One neat trick proposed by Salakhutdinov and Hinton is to add Gaussian noise (with zero mean) to the inputs of the coding layer, during training only. In order to preserve a high signal-to-noise ratio, the autoencoder will learn to feed large values to the coding layer (so that the noise becomes negligible). In turn, this means that the logistic function of the coding layer will likely saturate at 0 or 1. As a result, rounding the codings to 0 or 1 won't distort them too much, and this will improve the reliability of the hashes.

13 "Semantic Hashing," R. Salakhutdinov and G. Hinton (2008).

- Compute the hash of every image, and see if images with identical hashes look alike. Since MNIST and CIFAR10 are labeled, a more objective way to measure the performance of the autoencoder for semantic hashing is to ensure that images with the same hash generally have the same class. One way to do this is to measure the average Gini purity (introduced in Chapter 6) of the sets of images with identical (or very similar) hashes.
- Try fine-tuning the hyperparameters using cross-validation.
- Note that with a labeled dataset, another approach is to train a convolutional neural network (see Chapter 13) for classification, then use the layer below the output layer to produce the hashes. See Jinma Gua and Jianmin Li's 2015 paper.[14] See if that performs better.

10. Train a variational autoencoder on the image dataset used in the previous exercises (MNIST or CIFAR10), and make it generate images. Alternatively, you can try to find an unlabeled dataset that you are interested in and see if you can generate new samples.

Solutions to these exercises are available in Appendix A.

14 "CNN Based Hashing for Image Retrieval," J. Gua and J. Li (2015).

CHAPTER 16
Reinforcement Learning

Reinforcement Learning (RL) is one of the most exciting fields of Machine Learning today, and also one of the oldest. It has been around since the 1950s, producing many interesting applications over the years,[1] in particular in games (e.g., *TD-Gammon*, a *Backgammon* playing program) and in machine control, but seldom making the headline news. But a revolution took place in 2013 when researchers from an English startup called DeepMind demonstrated a system that could learn to play just about any Atari game from scratch,[2] eventually outperforming humans[3] in most of them, using only raw pixels as inputs and without any prior knowledge of the rules of the games.[4] This was the first of a series of amazing feats, culminating in March 2016 with the victory of their system AlphaGo against Lee Sedol, the world champion of the game of *Go*. No program had ever come close to beating a master of this game, let alone the world champion. Today the whole field of RL is boiling with new ideas, with a wide range of applications. DeepMind was bought by Google for over 500 million dollars in 2014.

So how did they do it? With hindsight it seems rather simple: they applied the power of Deep Learning to the field of Reinforcement Learning, and it worked beyond their wildest dreams. In this chapter we will first explain what Reinforcement Learning is and what it is good at, and then we will present two of the most important techniques in deep Reinforcement Learning: *policy gradients* and *deep Q-networks* (DQN),

1 For more details, be sure to check out Richard Sutton and Andrew Barto's book on RL, *Reinforcement Learning: An Introduction* (MIT Press), or David Silver's free online RL course at University College London.

2 "Playing Atari with Deep Reinforcement Learning," V. Mnih et al. (2013).

3 "Human-level control through deep reinforcement learning," V. Mnih et al. (2015).

4 Check out the videos of DeepMind's system learning to play *Space Invaders*, *Breakout*, and more at *https://goo.gl/yTsH6X*.