CHAPTER 8

Reinforcement Learning

The learning techniques we've covered so far in this book fall into the categories of supervised or unsupervised learning. In both cases, solving a given problem requires a data scientist to design a deep architecture that handles and processes input data and to connect the output of the architecture to a loss function suitable for the problem at hand. This framework is widely applicable, but not all applications fall neatly into this style of thinking. Let's consider the challenge of training a machine learning model to win a game of chess. It seems reasonable to process the board as spatial input using a convolutional network, but what would the loss entail? None of our standard loss functions such as cross-entropy or L^2 loss quite seem to apply.

Reinforcement learning provides a mathematical framework well suited to solving games. The central mathematical concept is that of the *Markov decision process*, a tool for modeling AI agents that interact with *environments* that offer *rewards* upon completion of certain *actions*. This framework proves to be flexible and general, and has found a number of applications in recent years. It's worth noting that reinforcement learning as a field is quite mature and has existed in recognizable form since the 1970s. However, until recently, most reinforcement learning systems were only capable of solving toy problems. Recent work has revealed that these limitations were likely due to the lack of sophisticated data intake mechanisms; hand-engineered features for many games or robotic environments often did not suffice. Deep representation extractions trained end-to-end on modern hardware seem to break through the barriers of earlier reinforcement learning systems and have achieved notable results in recent years.

Arguably, the first breakthrough in deep reinforcement learning was on ATARI arcade games. ATARI arcade games were traditionally played in video game arcades and offered users simple games that don't typically require sophisticated strategizing but might require good reflexes. Figure 8-1 shows a screenshot from the popular

ATARI game Breakout. In recent years, due to the development of good ATARI emulation software, ATARI games have become a testbed for gameplay algorithms. At first, reinforcement learning algorithms applied to ATARI didn't achieve superb results; the requirement that the algorithm understand a visual game state frustrated most attempts. However, as convolutional networks matured, researchers at DeepMind realized that convolutional networks could be combined with existing reinforcement learning techniques and trained end-to-end.

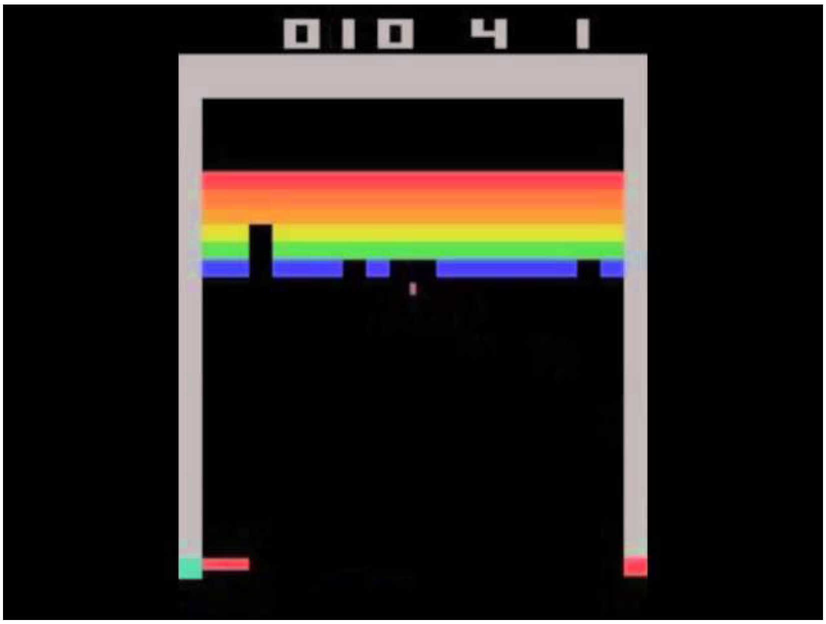

Figure 8-1. A screenshot of the ATARI arcade game Breakout. Players have to use the paddle at the bottom of the screen to bounce a ball that breaks the tiles at the top of the screen.

The resulting system achieved superb results, and learned to play many ATARI games (especially those dependent on quick reflexes) at superhuman standards. Figure 8-2 lists ATARI scores achieved by DeepMind's DQN algorithm. This breakthrough result spurred tremendous growth in the field of deep reinforcement learning and inspired legions of researchers to explore the potential of deep reinforcement learning techniques. At the same time, DeepMind's ATARI results showed reinforcement learning techniques were capable of solving systems dependent on short-term movements. These results didn't demonstrate that deep reinforcement learning systems were capable of solving games that required greater strategic planning.

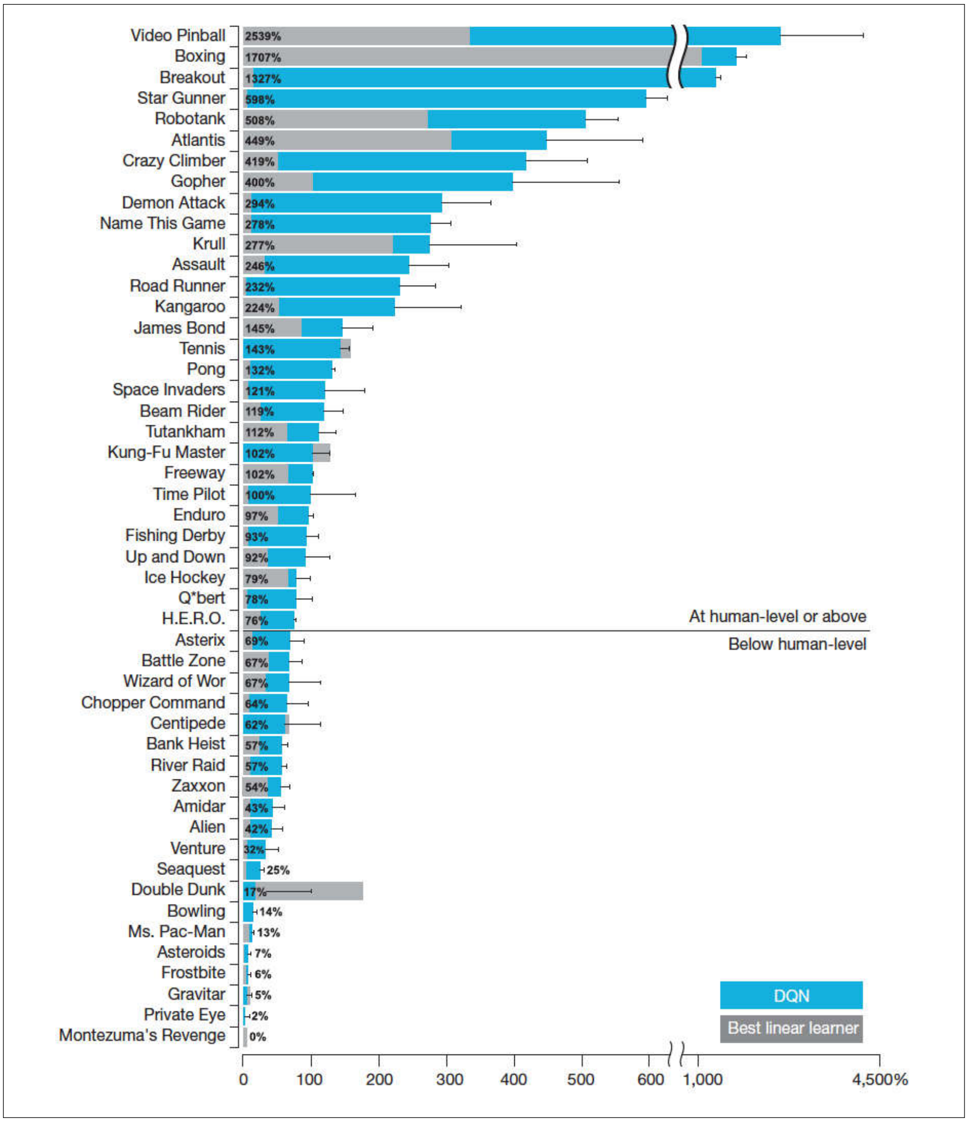

Figure 8-2. Results of DeepMind's DQN reinforcement learning algorithm on various ATARI games. 100% is the score of a strong human player. Note that DQN achieves superhuman performance on many games, but is quite weak on others.

Computer Go

In 1994, IBM revealed the system Deep Blue, which later succeeded in defeating Garry Kasparov in a highly publicized chess match. This system relied on brute force computation to expand the tree of possible chess moves (with some help from handcrafted chess heuristics) to play master-level chess.

Computer scientists attempted to apply similar techniques to other games such as Go. Unfortunately for early experimenters, Go's 19 × 19 game board is significantly larger than chess's 8 × 8 board. As a result, trees of possible moves explode much more quickly than for chess, and simple back-of-the-envelope calculations indicated that Moore's law would take a very long time to enable brute force solution of Go in the style of Deep Blue. Complicating matters, there existed no simple heuristic for evaluating who's winning in a half-played Go game (determining whether black or white is ahead is a notoriously noisy art for the best human analysts). As a result, until very recently, many prominent computer scientists believed that strong computer Go play was a decade away at the least.

To demonstrate the prowess of its reinforcement learning algorithms, DeepMind took on the challenge of learning to play Go, a game that requires complex strategic planning. In a tour-de-force paper, DeepMind revealed its deep reinforcement learning engine, AlphaGo, which combined convolutional networks with tree-based search to defeat the human Go master Lee Sedol (Figure 8-3).

Figure 8-3. Human Go champion Lee Sedol battles AlphaGo. Lee Sedol eventually lost the match 1–4, but succeeded in winning one game. It's unlikely that this victory can be replicated against the vastly improved successors of AlphaGo such as AlphaZero.

AlphaGo convincingly demonstrated that deep reinforcement learning techniques were capable of learning to solve complex strategic games. The heart of the breakthrough was the realization that convolutional networks could learn to estimate whether black or white was ahead in a half-played game, which enabled game trees to be truncated at reasonable depths. (AlphaGo also estimates which moves are most fruitful, enabling a second pruning of the game tree space.) AlphaGo's victory really launched deep reinforcement learning into prominence, and a host of researchers are working to transform AlphaGo-style systems into practical use.

In this chapter, we discuss reinforcement learning algorithms and specifically deep reinforcement learning architectures. We then show readers how to successfully apply reinforcement learning to the game of tic-tac-toe. Despite the simplicity of the game, training a successful reinforcement learner for tic-tac-toe requires significant sophistication, as you will soon see.

The code for this chapter was adapted from the DeepChem reinforcement learning library, and in particular from example code created by Peter Eastman and Karl Leswing. Thanks to Peter for debugging and tuning help on this chapter's example code.

Markov Decision Processes

Before launching into a discussion of reinforcement learning algorithms, it will be useful to pin down the family of problems that reinforcement learning methods seek to solve. The mathematical framework of Markov decision processes (MDPs) is very useful for formulating reinforcement learning methods. Traditionally, MDPs are introduced with a battery of Greek symbols, but we will instead try to proceed by providing some basic intuition.

The heart of MDPs is the pair of an *environment* and an *agent*. An environment encodes a "world" in which the agent seeks to act. Example environments could include game worlds. For example, a Go board with master Lee Sedol sitting opposite is a valid environment. Another potential environment could be the environment surrounding a small robot helicopter. In a prominent early reinforcement learning success, a team at Stanford led by Andrew Ng trained a helicopter to fly upside down using reinforcement learning as shown in Figure 8-4.