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.

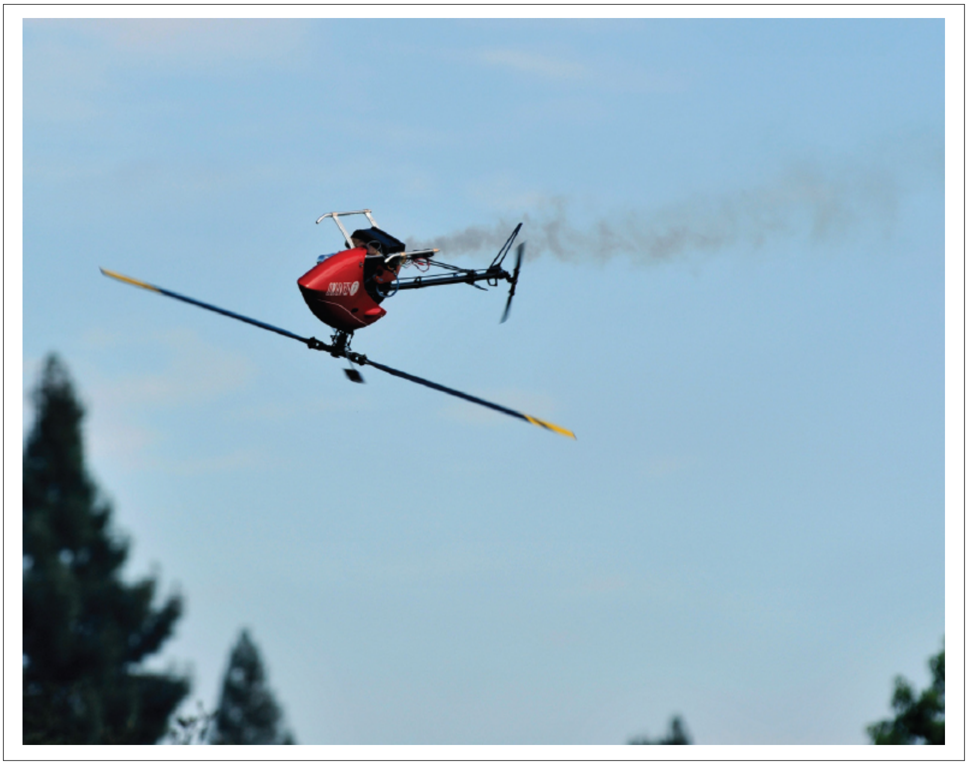

Figure 8-4. Andrew Ng's team at Stanford, from 2004 to 2010, trained a helicopter to learn to fly upside down using reinforcement learning. This work required the construction of a sophisticated physically accurate simulator.

The agent is the learning entity that acts within the environment. In our first example, AlphaGo itself is the agent. In the second, the robot helicopter (or more accurately, the control algorithm in the robot helicopter) is the agent. Each agent has a set of actions that it can take within the environment. For AlphaGo, these constitute valid Go moves. For the robot helicopter, these include control of the main and secondary rotors.

Actions the agent takes are presumed to have an effect on the environment. In the case of AlphaGo, this effect is deterministic (AlphaGo deciding to place a Go stone results in the stone being placed). In the case of the helicopter, the effect is likely probabilistic (changes in helicopter position may depend on wind conditions, which can't be modeled effectively).

The final piece of the model is the notion of reward. Unlike supervised learning where explicit labels are present to learn from, or unsupervised learning where the challenge is to learn the underlying structure of the data, reinforcement learning operates in a setting of partial, sparse rewards. In Go, rewards are achieved at the end of the game upon victory or defeat, while in helicopter flight, rewards might be presented for successful flights or completion of trick moves.

Reward Function Engineering Is Hard

One of the largest challenges in reinforcement learning is designing rewards that induce agents to learn desired behaviors. For even simple win/loss games such as Go or tic-tac-toe, this can be surprisingly difficult. How much should a loss be punished and how much should a win be rewarded? There don't yet exist good answers.

For more complex behaviors, this can be extremely challenging. A number of studies have demonstrated that simple rewards can result in agents learning unexpected and even potentially damaging behaviors. These systems spur fears of future agents with greater autonomy wreaking havoc when unleashed in the real world after having been trained to optimize bad reward functions.

In general, reinforcement learning is less mature than supervised learning techniques, and we caution that decisions to deploy reinforcement learning in production systems should be taken very carefully. Given uncertainty over learned behavior, make sure to thoroughly test any deployed reinforcement learned system.

Reinforcement Learning Algorithms

Now that we've introduced you to the core mathematical structures underlying reinforcement learning, let's consider how to design algorithms that learn intelligent behaviors for reinforcement learning agents. At a high level, reinforcement learning algorithms can be separated into the buckets of *model-based* and *model-free* algorithms. The central difference is whether the algorithm seeks to learn an internal model of how its environment acts. For simpler environments, such as tic-tac-toe, the model dynamics are trivial. For more complex environments, such as helicopter flight or even ATARI games, the underlying environment is likely extraordinarily complex. Avoiding the construction of an explicit model of the environment in favor of an implicit model that advises the agent on how to act may well be more pragmatic.