Download from finelybook www.finelybook.com
including a discussion of *Markov decision processes* (MDP). We will use these techniques to train a model to balance a pole on a moving cart, and another to play Atari games. The same techniques can be used for a wide variety of tasks, from walking robots to self-driving cars.

Learning to Optimize Rewards

In Reinforcement Learning, a software *agent* makes *observations* and takes *actions* within an *environment*, and in return it receives *rewards*. Its objective is to learn to act in a way that will maximize its expected long-term rewards. If you don't mind a bit of anthropomorphism, you can think of positive rewards as pleasure, and negative rewards as pain (the term "reward" is a bit misleading in this case). In short, the agent acts in the environment and learns by trial and error to maximize its pleasure and minimize its pain.

This is quite a broad setting, which can apply to a wide variety of tasks. Here are a few examples (see Figure 16-1):

a. The agent can be the program controlling a walking robot. In this case, the environment is the real world, the agent observes the environment through a set of *sensors* such as cameras and touch sensors, and its actions consist of sending signals to activate motors. It may be programmed to get positive rewards whenever it approaches the target destination, and negative rewards whenever it wastes time, goes in the wrong direction, or falls down.
b. The agent can be the program controlling Ms. Pac-Man. In this case, the environment is a simulation of the Atari game, the actions are the nine possible joystick positions (upper left, down, center, and so on), the observations are screenshots, and the rewards are just the game points.
c. Similarly, the agent can be the program playing a board game such as the game of *Go*.
d. The agent does not have to control a physically (or virtually) moving thing. For example, it can be a smart thermostat, getting rewards whenever it is close to the target temperature and saves energy, and negative rewards when humans need to tweak the temperature, so the agent must learn to anticipate human needs.
e. The agent can observe stock market prices and decide how much to buy or sell every second. Rewards are obviously the monetary gains and losses.

Download from finelybook www.finelybook.com

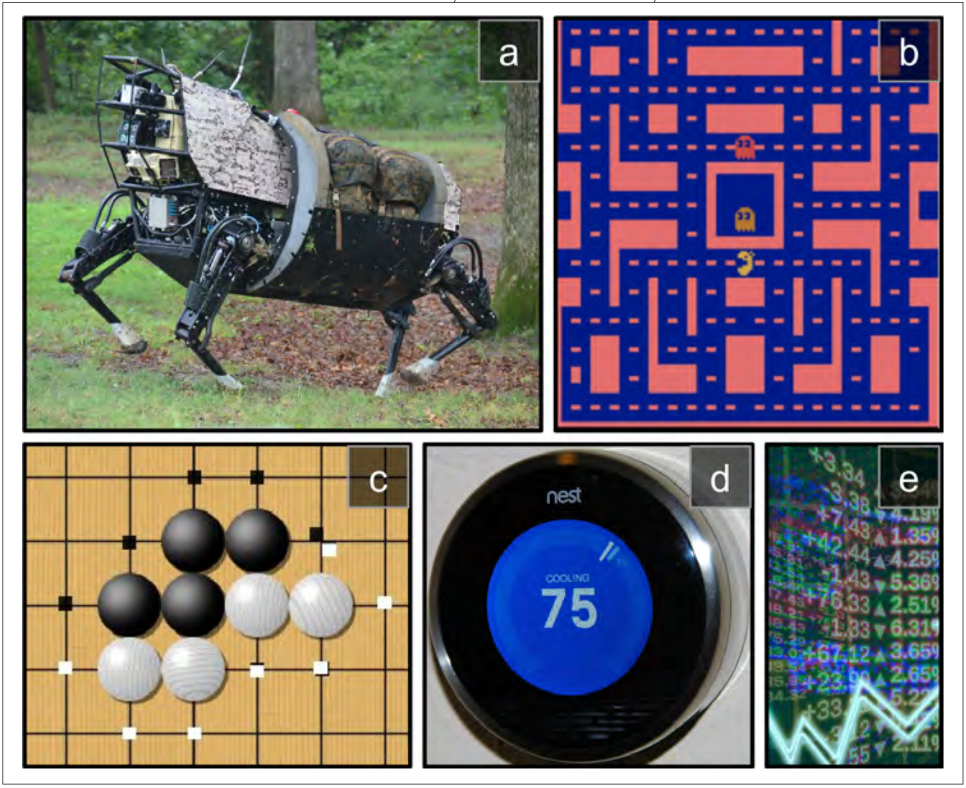

Figure 16-1. Reinforcement Learning examples: (a) walking robot, (b) Ms. Pac-Man, (c) Go player, (d) thermostat, (e) automatic trader[5]

Note that there may not be any positive rewards at all; for example, the agent may move around in a maze, getting a negative reward at every time step, so it better find the exit as quickly as possible! There are many other examples of tasks where Reinforcement Learning is well suited, such as self-driving cars, placing ads on a web page, or controlling where an image classification system should focus its attention.

5 Images (a), (c), and (d) are reproduced from Wikipedia. (a) and (d) are in the public domain. (c) was created by user Stevertigo and released under Creative Commons BY-SA 2.0. (b) is a screenshot from the Ms. Pac-Man game, copyright Atari (the author believes it to be fair use in this chapter). (e) was reproduced from Pixabay, released under Creative Commons CC0.

Download from finelybook www.finelybook.com

Policy Search

The algorithm used by the software agent to determine its actions is called its *policy*. For example, the policy could be a neural network taking observations as inputs and outputting the action to take (see Figure 16-2).

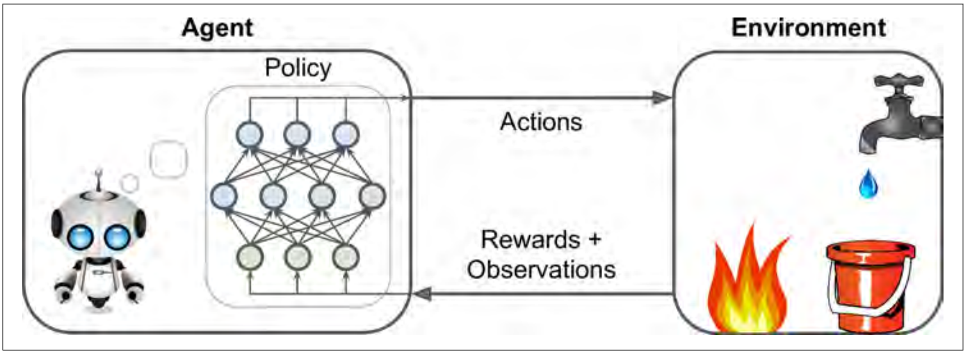

Figure 16-2. Reinforcement Learning using a neural network policy

The policy can be any algorithm you can think of, and it does not even have to be deterministic. For example, consider a robotic vacuum cleaner whose reward is the amount of dust it picks up in 30 minutes. Its policy could be to move forward with some probability p every second, or randomly rotate left or right with probability $1 - p$. The rotation angle would be a random angle between –r and +r. Since this policy involves some randomness, it is called a *stochastic policy*. The robot will have an erratic trajectory, which guarantees that it will eventually get to any place it can reach and pick up all the dust. The question is: how much dust will it pick up in 30 minutes?

How would you train such a robot? There are just two *policy parameters* you can tweak: the probability p and the angle range r. One possible learning algorithm could be to try out many different values for these parameters, and pick the combination that performs best (see Figure 16-3). This is an example of *policy search*, in this case using a brute force approach. However, when the *policy space* is too large (which is generally the case), finding a good set of parameters this way is like searching for a needle in a gigantic haystack.

Another way to explore the policy space is to use *genetic algorithms*. For example, you could randomly create a first generation of 100 policies and try them out, then "kill" the 80 worst policies[6] and make the 20 survivors produce 4 offspring each. An off-

6 It is often better to give the poor performers a slight chance of survival, to preserve some diversity in the "gene pool."