REINFORCEMENT LEARNING

Introduction

Informally, RL is about learning how to achieve a goal through interaction with an environment.

More formally, RL is concerned with mapping states to actions so as to maximize a numerical reward signal.

Example Chess

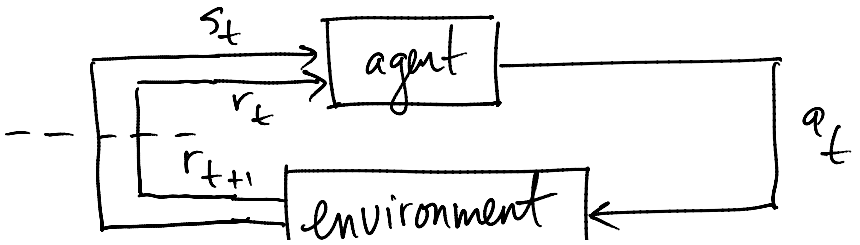
State s_t = current board configuration

Action a_t = your next move

Reward

$$r_t = \begin{cases} +1 & \text{if your move wins the game} \\ -1 & \text{if your move losses the game} \\ 0 & \text{if the game continues or is drawn} \end{cases}$$

In this example, the "environment" is your opponent.



Rewards communicate what you want to achieve, not how. E.g., we do not reward a chess program for capturing pieces, controlling the center, etc.

The output of an RL algorithm is a policy π, or a family of policies π_t if the environment is changing. A policy is a function

$$\pi_t(s, a) = \left(\text{probability that } a_t = a \text{ given } s_t = s\right)$$

Note that policies need not be deterministic.

Returns

What makes a good policy? Let's consider two cases.

1. Episodic tasks: Tasks that eventually terminate.

   Examples: plays of a game, trips through a maze.

   Essentially, and task that ends when a special state, called a terminal state (game over, maze exitted),

is reached.

If a task lasts for T steps, the <u>return</u> from time t to time T is

$$R_t = r_{t+1} + r_{t+2} + \cdots + r_T.$$

2. <u>Continuing tasks</u>: Tasks that continue indefinitely.

Examples: elevator dispatching, traffic planning, WALL-E

In this case, we define the <u>discounted return</u>

$$R_t = \sum_{k \geq 0} \gamma^k r_{t+k+1}$$

where $0 < \gamma < 1$ is the discount rate.

We can view an episodic task as a special case of a continuing task, where we view a terminal state as an <u>absorbing</u> state (once you enter, you can't leave). Then the return is

$$R_1 = \sum \gamma^k r_{t+k+1}$$

$$R_t = \sum_{k \geq 0} \gamma^k r_{t+k+1}$$

where $\gamma = 1$ and the reward for entering an absorbing state is always zero.

Value Functions

The return is random, as it depends on the policy and the environment. The value function is the expected return w.r.t. a given policy. We will work with two kinds of value functions.

1. State value function for π

$$V^\pi(s) = \mathbb{E}_\pi \{ R_t \mid s_t = s \}$$

$$= \mathbb{E}_\pi \left\{ \sum_{k \geq 0} \gamma^k r_{t+k+1} \mid s_t = s \right\}$$

2. Action-state value function for π:

$$Q^\pi(s,a) = \mathbb{E}_\pi \{ R_t \mid s_t = s, a_t = a \}$$

$$= \mathbb{E}_\pi \left\{ \sum_{k \geq 0} \gamma^k r_{t+k+1} \;\middle|\; s_t = s,\ a_t = a \right\}.$$

If we know the value functions for different policies, we could use them to choose the best one. Let we we see how to use value functions to find an optimal policy.

It's important to understand the distinction between reward and value.

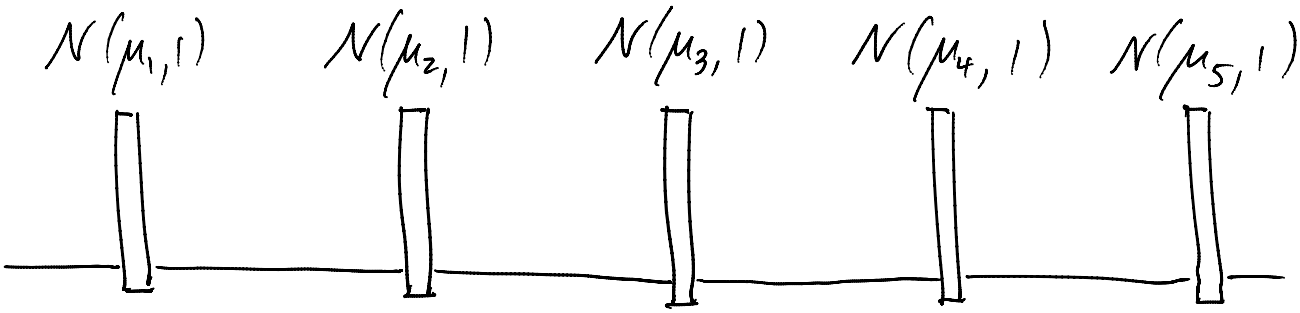
reward : short term desirability of a state,
easy to know,
provided by environment

value : long-term desirability of state,
usually must be estimated,
value function estimation is a central part of most RL algorithms.

The Multi-Arm Bandit Problem

An n-armed bandit is a game with n levers.

Every time you pull a lever, you receive a reward ($n=1 \iff$ slot machine). Assume the reward for a lever is a random variable whose distribution is specific to that lever. For example, suppose $n=5$ and



This is an episodic task with $T=1$. What policy should you adopt to maximize your return over repeated plays of the game? Assume the rewards for a given lever are iid.

Well, if the μ_i are known, just play the i^{th} lever all the time.

Now consider the more interesting case where the μ_i are unknown. The name of the game is value function estimation. Denote

$r^*(a)$ = expected reward of lever a

$\hat{r}(a)$ = estimate of $r^*(a)$

Let $\hat{r}_0(a)$ be an initial guess of $r^*(a)$. After playing the game for a while, suppose we play $a_1, \ldots, a_T$ and observe rewards $r_1, \ldots, r_T$. The sample average estimate is

$$\hat{r}(a) = \frac{1}{|I_a|} \sum_{t \in I_a} r_t$$

where

$$I_a = \{t : a_t = a\}.$$

Policy design must address the exploration-exploitation tradeoff:

exploitation: choose a for which $\hat{r}(a)$ is largest

exploration: choose some other a to improve the accuracy of $\hat{r}(a)$

In RL, a key issue is striking the right balance between these two. Consider the following two policies:

1. greedy:

$$a_t = \arg\max_a \hat{r}(a)$$

2. ε-greedy:

$$a_t = \begin{cases} \arg\max_a \hat{r}(a) & \text{with prob } 1-\varepsilon \\ \text{random action} & \text{with prob } \varepsilon \end{cases}$$

Unlike the greedy policy, the ε-greedy policy is guaranteed to try each action infinitely often, and therefore will ultimately have very accurate estimates of $r^*(a)$.

From Sutton and Barto

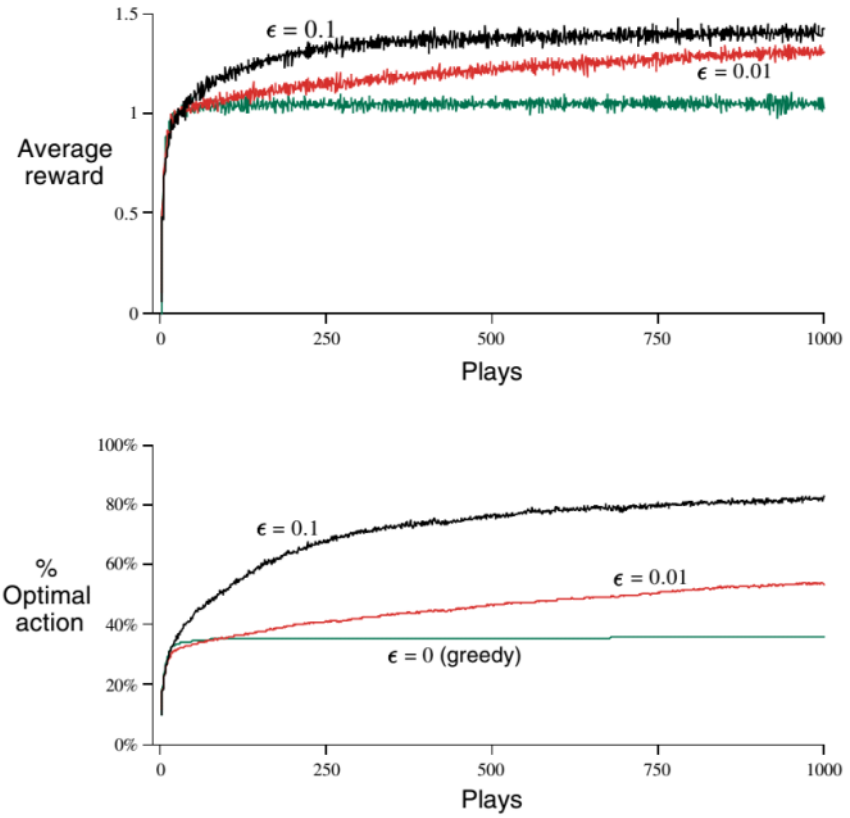


Figure 2.1 Average performance of ϵ-greedy action-value methods on the 10-armed testbed. These data are averages over 2000 tasks. All methods used sample averages as their action-value estimates.

↳ i.e., 2000 draws of

$\mu_1, \ldots, \mu_{10} \overset{iid}{\sim} N(0,1)$

Additional considerations:

1. What if the rewards are drifting? Notice that the sample average estimate is updated according to

$$\hat{r}(a) \leftarrow \hat{r}(a) + \frac{1}{k+1}\left[r_{new} - \hat{r}(a)\right]$$

where r_{new} is the k^{th} reward observed for action a.

For drifting rewards, use

$$\hat{r}(a) \leftarrow \hat{r}(a) + \alpha\left[r_{new} - \hat{r}(a)\right]$$

where $0 < \alpha < 1$ is fixed. This implies

$$\hat{r}(a) = (1-\alpha)^k \hat{r}_0(a) + \sum_{i=1}^{k} \alpha(1-\alpha)^{k-i} r_{a,i}$$

$r_{a,i}$: reward from i^{th} selection of a

2. The <u>softmax</u> action selection rule is

select $a_t = a$ with probability $\dfrac{e^{\hat{r}(a)/\tau}}{\sum_b e^{\hat{r}(b)/\tau}}$

where $\tau > 0$ is called the <u>temperature</u> parameter. Note

$\tau \to \infty$ random

$\tau \to 0$ greedy

Like ε-greedy, this policy explores the action space, but has the advantage that it wastes less time on really bad actions.

3. By setting $\hat{r}_0(a)$ to be very large, we can ensure that even the greedy policy explores each action at least once. This idea may be generalized using Bayesian estimation.

Reference

Sutton and Barto, Reinforcement Learning: An Introduction