REINFORCEMENT LEARNING

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

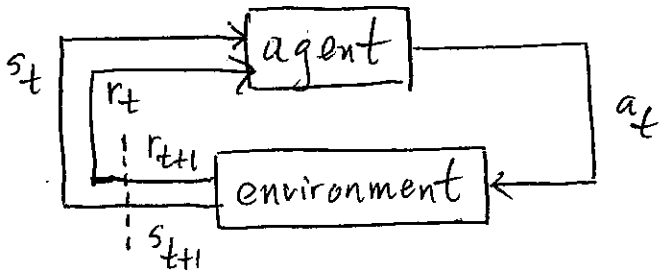
More formally, RL is about learning how to map states to actions, in order to maximize a numerical reward signal.

Example | Chess

state : the current board configuration (s_t)

action : your next move (a_t)

reward : $r_{t+1} = \begin{cases} +1 & \text{if your move wins the game} \\ -1 & \text{" " " losses " "} \\ 0 & \text{if the game continues, or ends in a draw} \end{cases}$



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

Rewards communicate what you want the agent to achieve, not how

E.g., we do not reward a chess program for capturing pieces or controlling the center.

The goal of a RL algorithm is to produce a policy π, or family of policies π_t (since they might evolve over time)

$$\pi_t(s,a) =$$

(A)

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 with probability 1.

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

We define the <u>discounted return</u>

$$R_t =$$

(B)

where $0 < \gamma < 1$ is the

<u>Convention</u> | we can view an episodic task as a special case of a continuing task that has entered an _____ state. Then we can always write

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

where for episodic tasks, $\gamma = 1$ and the reward for entering an absorbing state is always 0.

What are some examples of continuing tasks?

(c)

Value Functions

The return is random, and it depends on the policy/policies adopted.

The <u>value function</u> is the expected return w.r.t a given policy. We will work with two kinds of value functions:

1) <u>state value function</u> for π:

$$V^{\pi}(s) = E_{\pi}\{R_t \mid s_t = s\}$$

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

2) <u>action-state value function</u> for π:

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

$$= E_{\pi}\left\{\sum_{k \geq 0} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\}$$

If we knew the value functions, we could choose the best action for a given policy, and even choose the best of competing policies.

reward vs. value

reward = short-term desirability of a state

value = long-term desirability of a state

reward: usually easy to know; provided by the environment

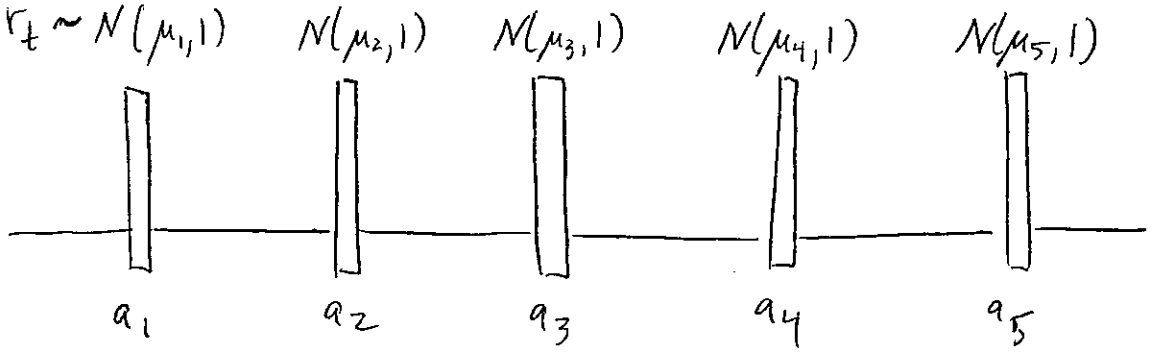
value: usually must be estimated; a central part of most RL algs. is a method for value estimation

The Multi-Armed Bandit Problem

Let's consider an RL problem that is simple yet illustrates some of the key issues in RL.

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

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



These distributions are unknown to you. What policy should you adopt to maximize your **return** over many plays of the **game**?

Reward Estimation

Since the game is memoryless, there is only one state. In addition, because of the memoryless property, all we need to know to compute the value of a policy is the expected reward of each action.

Notation

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

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

Initially, we must guess $\hat{r}(a)$, i.e.

$$\hat{r}(a) = \hat{r}_0(a) \quad \forall a$$

After playing the game for a while, suppose we played $a_1, \ldots, a_T$ and observed rewards $r_1, \ldots, r_T$. The sample average estimate is

$$\hat{r}(a) =$$

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

Exploration vs. Exploitation

There is a fundamental tradeoff when choosing the next action in a multi-armed bandit problem

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

exploration: choose some other action 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

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

Unlike the greedy policy, the ϵ-greedy policy is guaranteed to try each action infinitely often in the long run.

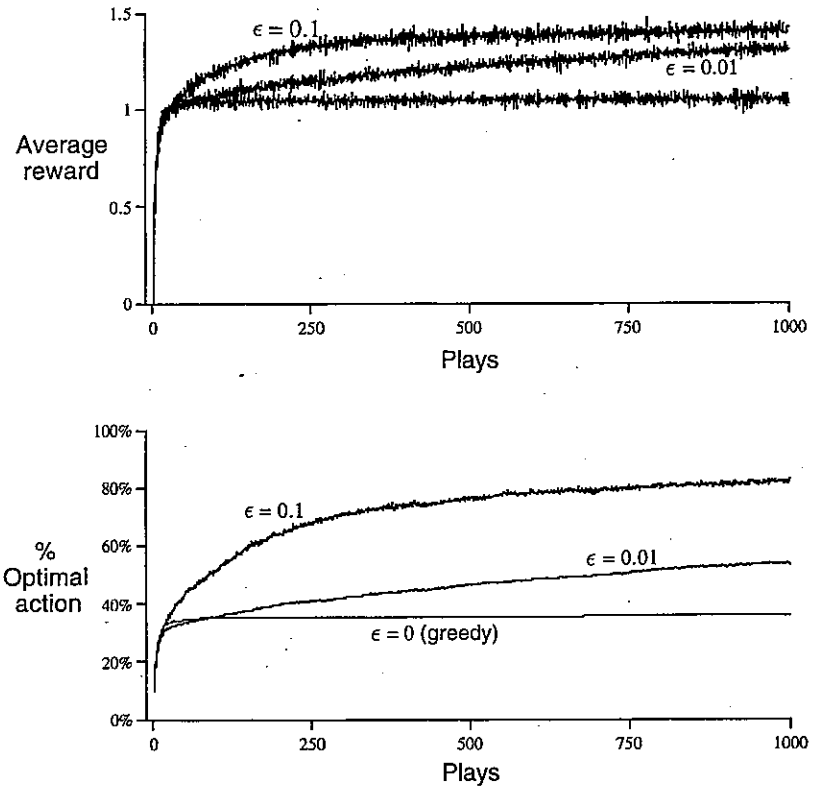


Figure 2.1 Average performance of ϵ-greedy action-value methods on the 10-armed testbed. These data are averages over 2000 tasks. All methods used sample averages as their action-value estimates.

← Sutton and Barto

Additional Issues

- What if the expected rewards are drifting?

(E)

- Softmax action selection: select a with probability

(F)

- Initialization: $\hat{r}_0(a)$ large ⇒ more exploration by greedy alg.

References

Sutton & Barto, Reinforcement Learning: An Introduction
- good intuition, not much math; best place to start

Bertsekas & Tsitsiklis, Neuro-Dynamic Programming
- more mathematical

Key

A: $\pi_t(s,a)$ = probability that $a_t = a$ given $s_t = s$

B. $R_t = \sum_{k \geq 0} \gamma^k r_{t+k+1}$, γ = discount rate, absorbing

C. Elevator dispatching, WALL-E

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

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

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

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

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

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

F. $e^{\hat{r}(a)/\tau} \Big/ \left(\sum_b e^{\hat{r}(b)/\tau}\right)$ τ = temperature

$\tau \to \infty$ random

$\tau \to 0$ greedy