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: \[ r_{t+1} = \begin{cases} +1 & \text{if your move wins the game} \\ -1 & \text{if you lose} \\ 0 & \text{if the game continues, or ends in a draw} \end{cases} \]

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 $\pi$, or family of policies $\pi_t$ (since they might evolve over time)

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

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 return from time \( t \) to time \( T \) is

\[
R_t = r_{t+1} + r_{t+2} + \cdots + r_T
\]

2) Continuing tasks: Tasks that continue indefinitely

We define the discounted return

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

where \( 0 < \gamma < 1 \) is the convention. We can view an episodic task as a special case of a continuing task that has entered an absorbing state. Then we can always write

where for episodic tasks, \( \gamma = 1 \) and the reward for entering an absorbing state is always 0.
What are some examples of continuing tasks?

Value Functions

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

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 $\pi$:

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

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

2) action-state value function for $\pi$:

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

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

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 is usually easy to know; provided by the environment  
value is usually must be estimated; a central part of most RL algo. 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

\[ t \sim N(\mu_1) \quad N(\mu_2,1) \quad N(\mu_3,1) \quad N(\mu_4,1) \quad N(\mu_5,1) \]

---

\[ q_1 \quad q_2 \quad q_3 \quad q_4 \quad q_5 \]
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

\[ r^*(a) = \text{expected reward of level } a \]
\[ \hat{r}(a) = \text{estimate of } r(a) \]

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 \( \hat{V}(a) \) is largest.
- **Exploration**: choose some other action \( a' \), to improve the accuracy of \( \hat{V}(a) \).

In RL, a key issue is striking the right balance between these two.

Consider the following two policies:

1) **greedy**:

   \[ q_t = \arg \max_a \hat{V}(a) \]

2) **\( \epsilon \)-greedy**:

   \[ q_t = \begin{cases} 
   \arg \max_a \hat{V}(a) & \text{with prob } 1-\epsilon \\
   \text{random action} & \text{with prob } \epsilon 
   \end{cases} \]

Unlike the greedy policy, the \( \epsilon \)-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.

Additional Issues

- What if the expected rewards are drifting?

- Softmax action selection; select a with probability

- Initialization: \( \hat{\pi}_0(a) \) large \( \Rightarrow \) 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. \( R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \) \( \gamma \) = discount rate, absorbing

C. Elevator dispatching, WALL-E

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

E. Sample average:

\[
\hat{V}(a) \leftarrow \hat{V}(a) + \frac{1}{k+1} [r_{\text{new}} - \hat{V}(a)]
\]

where \( r_{\text{new}} \) is the \( k \)th reward observed for action \( a \)

Weighted average:

\[
\bar{V}(a) \leftarrow \bar{V}(a) + \alpha [r_{\text{new}} - \bar{V}(a)]
\]

\[
\Rightarrow \hat{V}(a) = (1-\alpha)^k \bar{V}_0(a) + \sum_{i=1}^{k} \alpha (1-\alpha)^{k-i} r_{a,i}
\]

reward from \( i \)th selection of action \( a \)

F. \( e^{\hat{V}(a)/\tau} / \sum_b e^{\hat{V}(b)/\tau} \) \( \tau \) = temperature

\( \tau \rightarrow \infty \) random

\( \tau \rightarrow 0 \) greedy