Learning Policies
Monday, November 24, 2014
9:10 AM

LEARNING POLICIES FROM EXPERIENCE

Introduction

We will now study strategies for evaluating and improving policies when the MDP parameters are unknown, but instead we can learn about the MDP through experience.

The methods we'll study fall under a general framework called generalized policy iteration.

$$\begin{tikzcd}
\pi \arrow[r, bend left, "\text{evaluation}"] & V, Q \arrow[l, bend left, "\text{improvement}"]
\end{tikzcd}$$

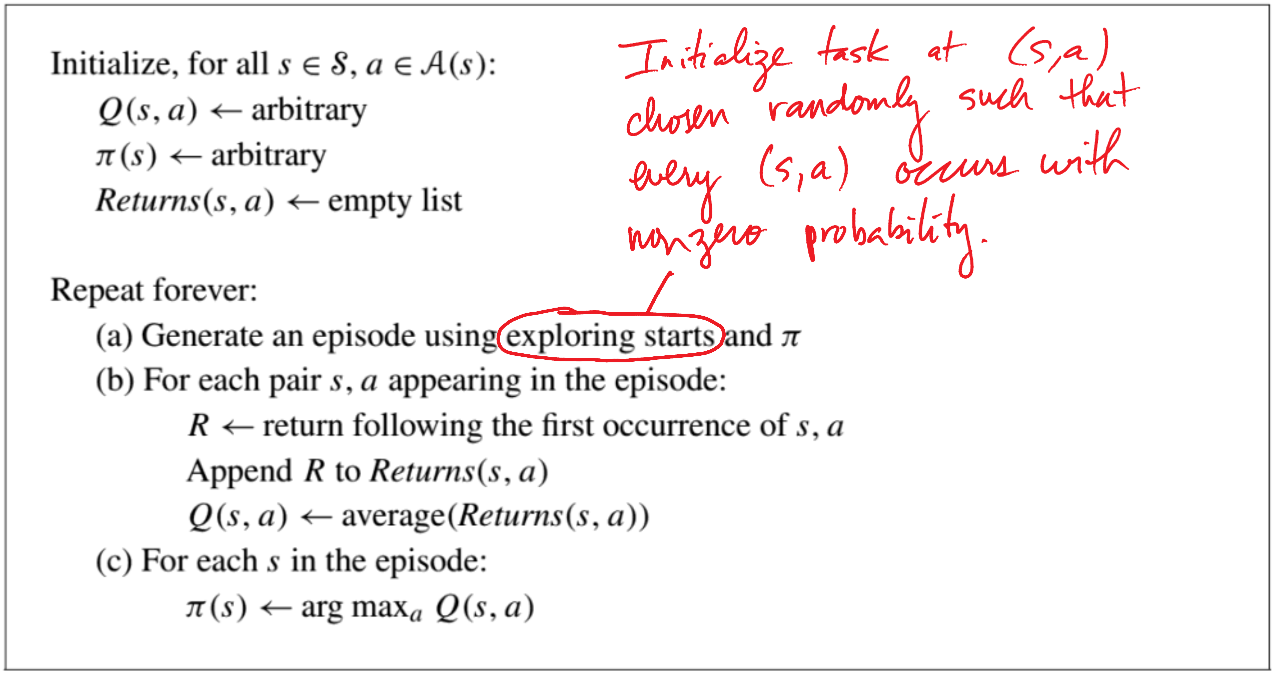
Monte Carlo Methods

Let π be a policy for an episodic task. Suppose that we can easily perform the task repeatedly. Then the

MC estimate of π's state-action value function is

$\hat{Q}(s,a)$ = average of all returns following the first occurrence of (s,a) in an episode.

Further suppose that an episode can be initialized at an arbitrary (s,a). This suggests the following GPI strategy:



```
Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
    $Q(s,a) \leftarrow$ arbitrary
    $\pi(s) \leftarrow$ arbitrary
    $Returns(s,a) \leftarrow$ empty list

Repeat forever:
    (a) Generate an episode using exploring starts and $\pi$
    (b) For each pair $s, a$ appearing in the episode:
            $R \leftarrow$ return following the first occurrence of $s, a$
            Append $R$ to $Returns(s,a)$
            $Q(s,a) \leftarrow$ average$(Returns(s,a))$
    (c) For each $s$ in the episode:
            $\pi(s) \leftarrow \arg\max_a Q(s,a)$
```

Initialize task at (s,a) chosen randomly such that every (s,a) occurs with nonzero probability.

Here's an example from Sutton and Barto

Example Blackjack

Blackjack is a game played with standard playing cards.

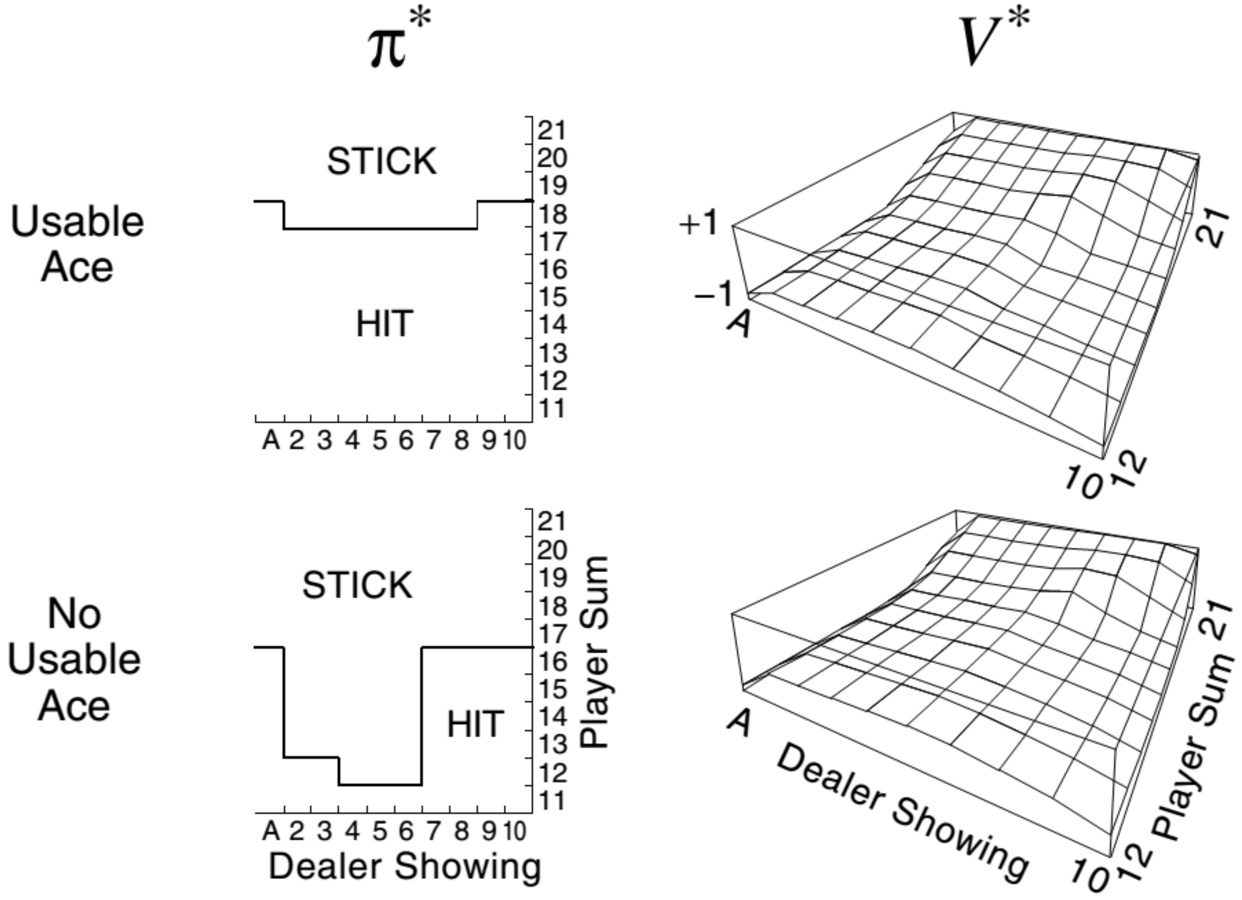
Example 5.1 *Blackjack* is a popular casino card game. The object is to obtain cards the sum of whose numerical values is as great as possible without exceeding 21. All face cards count as 10, and the ace can count either as 1 or as 11. We consider the version in which each player competes independently against the dealer. The game begins with two cards dealt to both dealer and player. One of the dealer's cards is faceup and the other is facedown. If the player has 21 immediately (an ace and a 10-card), it is called a *natural*. He then wins unless the dealer also has a natural, in which case the game is a draw. If the player does not have a natural, then he can request additional cards, one by one (*hits*), until he either stops (*sticks*) or exceeds 21 (*goes bust*). If he goes bust, he loses; if he sticks, then it becomes the dealer's turn. The dealer hits or sticks according to a fixed strategy without choice: he sticks on any sum of 17 or greater, and hits otherwise. If the dealer goes bust, then the player wins; otherwise, the outcome—win, lose, or draw—is determined by whose final sum is closer to 21.

State: your cards, dealer's visible card, and whether you have a usuable ace - an ace that could be counted as 1 or 11 without going over 21

Action: Hit or stand

$$\text{Reward} = \begin{cases} +1 & \text{win} \\ -1 & \text{lose} \\ 0 & \text{draw} \end{cases}$$

Here is the result of the above MC algorithm:



This MC strategy is sometimes called MC with exploring starts. If it is not possible to initialize a task arbitrarily, it may be necessary to incorporate exploration into the policy update, such as an ε-greedy policy.

MC methods are the heart of state-of-the-art programs for playing Go.

Temporal Difference Methods

Unlike MC methods, TD methods update before the end of a task, and are suitable for continuing tasks. Consider an update of the form

$$V(s_t) \longleftarrow V(s_t) + \alpha\,[\,\text{TARGET} - V(s_t)\,]$$

Monte Carlo has this form, where $\alpha = \frac{1}{k+1}$ and TARGET is the $(k+1)^{st}$ return observed after a first occurrence of state s_t.

To motivate an alternative update, consider

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

$$= \mathbb{E}_{\pi}\{r_{t+1} + \gamma R_{t+1} \mid s_t = s\}$$

$$= \mathbb{E}_{\pi}\{r_{t+1} + \gamma V^{\pi}(s_{t+1}) \mid s_t = s\}$$

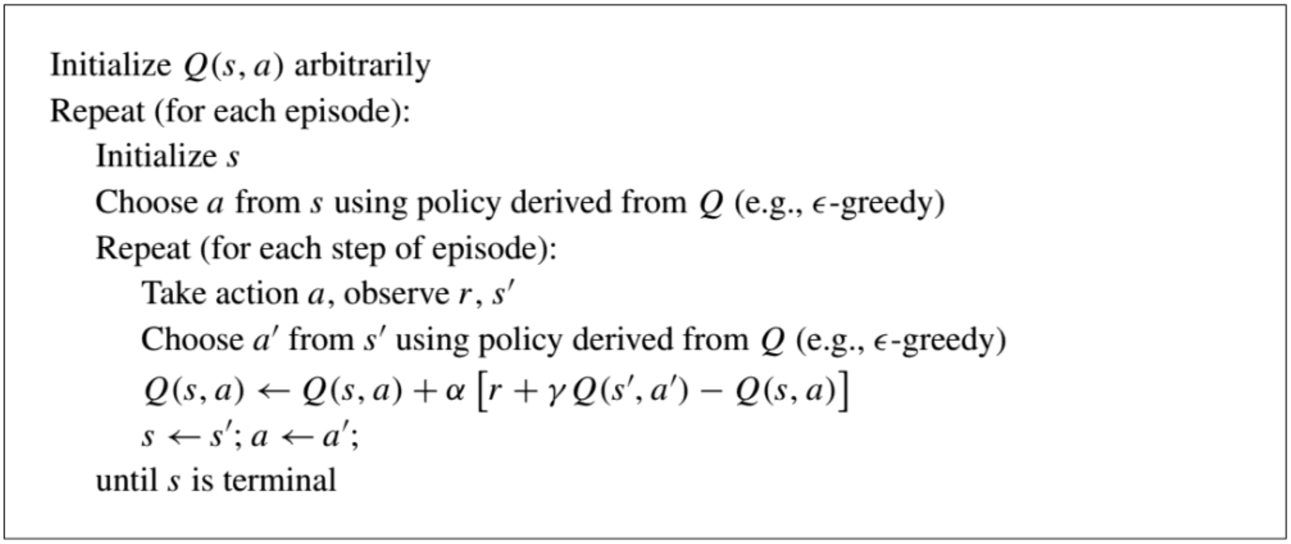
The <u>TD(0) update</u> uses

$$\text{TARGET} = r_{t+1} + \gamma V(s_{t+1})$$

leading to the value function update

$$V(s_t) \longleftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right]$$

It can be shown that when following a fixed π, this converges to V^{π}. TD(0) can also be combined with policy improvement to learn a policy.



```
Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
    Initialize $s$
    Choose $a$ from $s$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
    Repeat (for each step of episode):
        Take action $a$, observe $r$, $s'$
        Choose $a'$ from $s'$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
        $Q(s,a) \leftarrow Q(s,a) + \alpha \left[ r + \gamma Q(s',a') - Q(s,a) \right]$
        $s \leftarrow s'$; $a \leftarrow a'$;
    until $s$ is terminal
```

This algorithm is called Sarsa by Sutton + Barto because of the variables used: s, a, r, s', a'

Example 6.5: Windy Gridworld Figure 6.10 shows a standard gridworld, with start and goal states, but with one difference: there is a crosswind upward through the middle of the grid. The actions are the standard four—`up`, `down`, `right`, and `left`—but in the middle region the resultant next states are shifted upward by a "wind," the strength of which varies from column to column. The strength of the wind is given below each column, in number of cells shifted upward. For example, if you are one cell to the right of the goal, then the action `left` takes you to the cell just above the goal. Let us treat this as an undiscounted episodic task, with constant rewards of -1 until the goal state is reached. Figure 6.11 shows the result of applying ϵ-greedy Sarsa to this task, with $\epsilon = 0.1$, $\alpha = 0.1$, and the initial values $Q(s, a) = 0$ for all s, a. The increasing slope of the graph shows that the goal is reached more and more quickly over time. By 8000 time steps, the greedy policy (shown inset) was long since optimal; continued ϵ-greedy exploration kept the average episode length at about 17 steps, two less than the minimum of 15. Note that Monte Carlo methods cannot easily be used on this task because termination is not guaranteed for all policies. If a policy was ever found that caused the agent to stay in the same state, then the next episode would never end. Step-by-step learning methods such as Sarsa do not have this problem because they quickly learn *during the episode* that such policies are poor, and switch to something else. ■

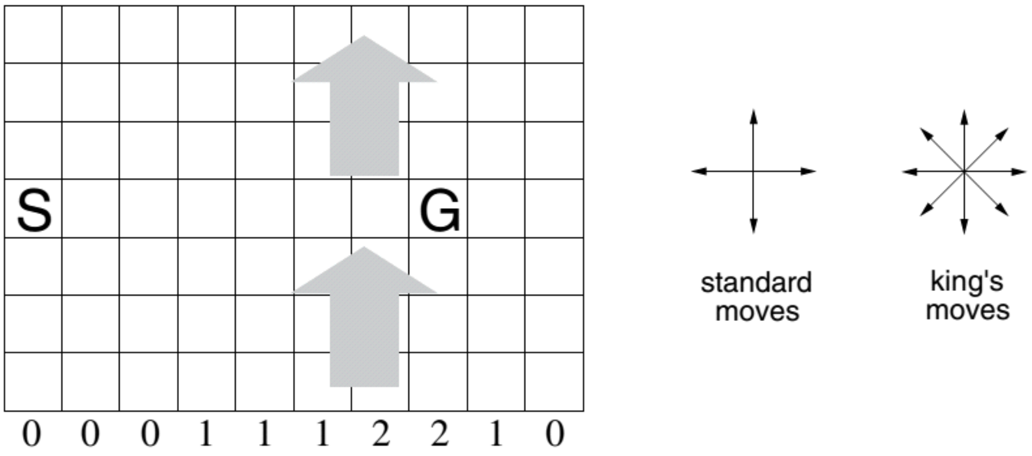
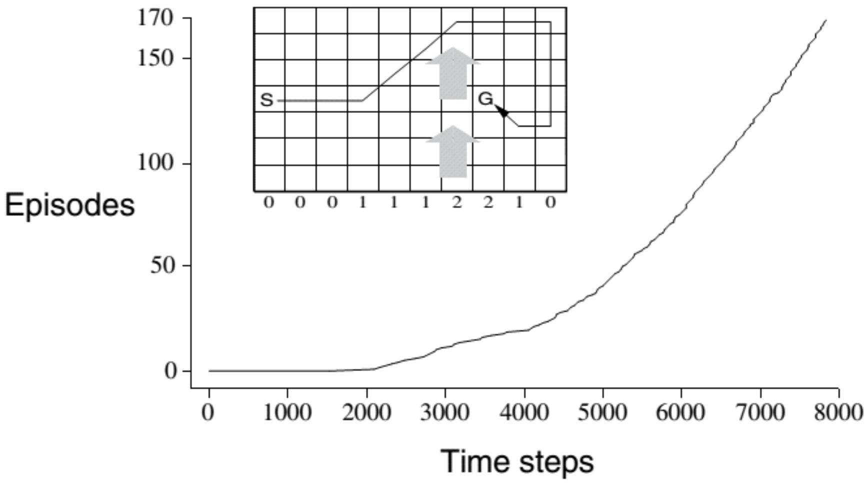
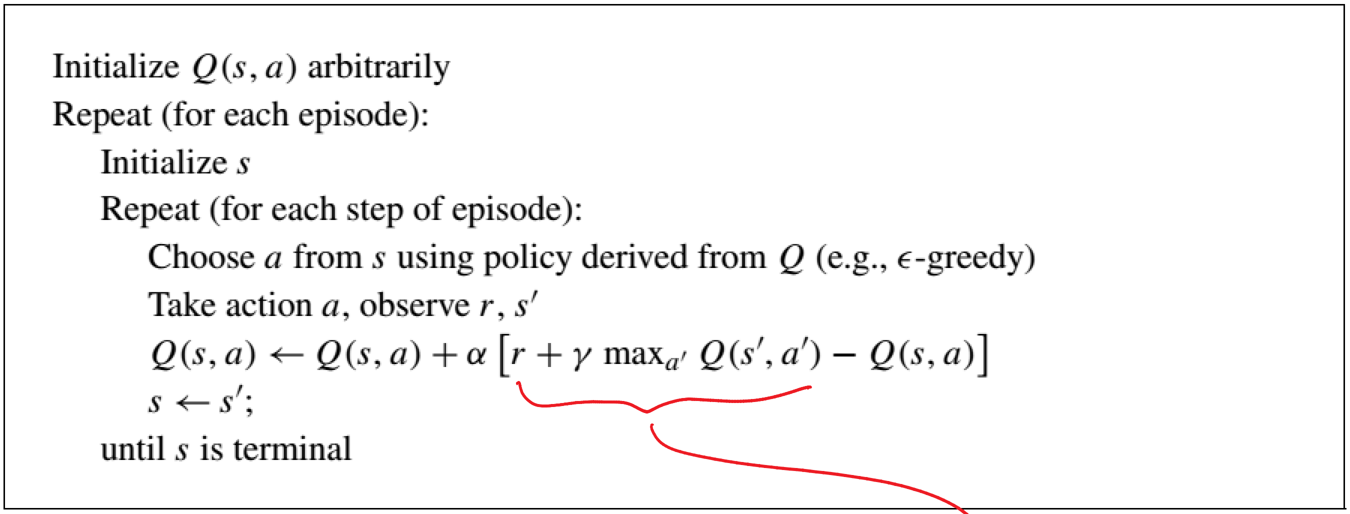


Figure 6.10 Gridworld in which movement is altered by a location-dependent, upward "wind."



Sarsa is an example of an on-policy algorithm, meaning that the policy that is used to generate behavior is the same policy that is evaluated and improved. If this is not the case, the algorithm is called off policy. There is a well know off-policy counterpart to Sarsa called Q-Learning.



```
Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
    Initialize $s$
    Repeat (for each step of episode):
        Choose $a$ from $s$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
        Take action $a$, observe $r$, $s'$
        $Q(s,a) \leftarrow Q(s,a) + \alpha \left[\underbrace{r + \gamma \max_{a'} Q(s',a')}_{\text{directly approximates optimal } Q^*} - Q(s,a)\right]$
        $s \leftarrow s'$;
    until $s$ is terminal
```

Figure 6.12 Q-learning: An off-policy TD control algorithm.

Q-Learning explores with one policy, such as ε-greedy based on the current estimate of Q, but it converges to the optimal Q^* (under certain assumptions), so the policy be learned is an optimal policy.

- each state-action pair visited infinitely often
- $\alpha = \alpha_t \to 0$ at a certain rate

There is a generalization of TD(0) called <u>TD(λ)</u>, $\lambda \in [0,1]$. This may be thought of as a compromise between MC and TD(0) that often outperforms both. There are on-policy and off-policy learning algorithms for TD(λ), similar to TD(0).