LEARNING POLICIES FROM EXPERIENCE

In these notes we will discuss strategies for evaluating and improving policies when the MDP parameters are unknown.

These methods attempt to approximate the optimal algorithms we just studied, kind of like how supervised learning methods approximate the Bayes classifier.

The methods we'll study fall under a general framework that we might call

Ⓐ

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

Monte Carlo Methods

Policy Evaluation

Let π be a policy for an episodic task.

Suppose we can easily perform the task many times using π. Then the MC estimates of π's value functions are:

$\hat{V}(s)$ = average of all returns following the first visit to s in an episode

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

Why would we only consider the first visits?

By the law of large numbers, as the number of episodes $\rightarrow \infty$, the estimates will converge to the true values.

Control

How can we use the MC estimates of the value function(s) to improve a given policy?

What are the issues?

In the case where the MDP parameters are known, a <u>greedy</u> policy improvement strategy was sufficient.

However, if we adopt a greedy update here,

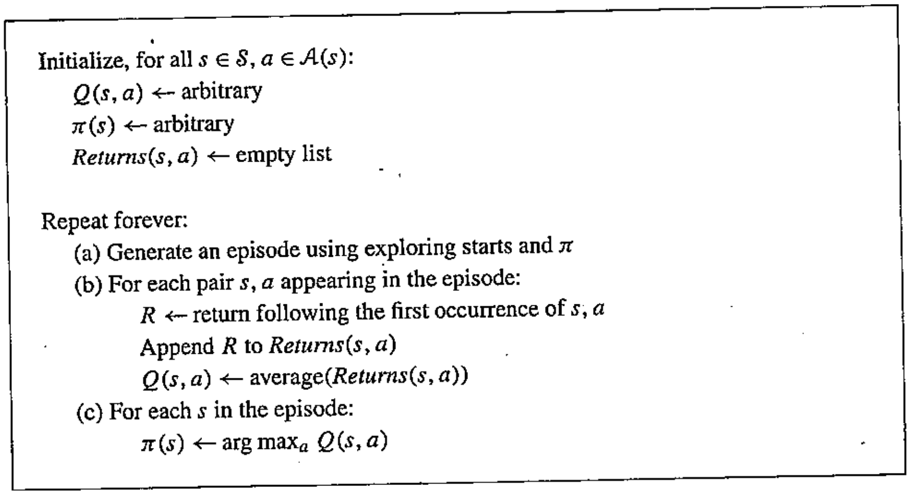
$$\pi(s) \leftarrow \arg\max_{a} \hat{Q}(s,a)$$

then some (s,a) pairs may never be <u>explored</u>, if π is deterministic.

What might we do to avoid this problem?

(B)

<u>Exploring Starts</u>



```
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:
            $\pi(s) \leftarrow \arg\max_a Q(s,a)$
```

Example | Blackjack

An ace is usable if the player can count it is 11 without going over 21.

Calculation of MDP parameters is hard, but simulation is easy!

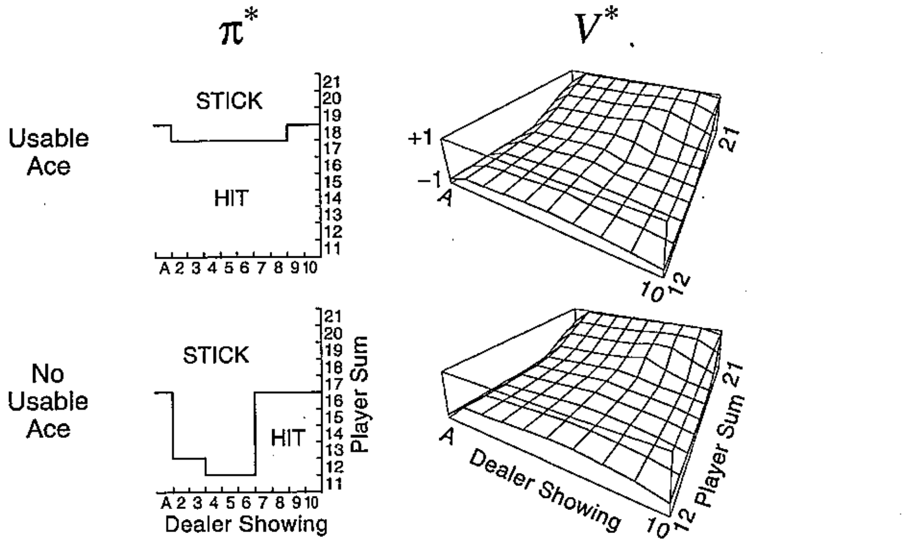


Figure 5.5 The optimal policy and state-value function for blackjack, found by Monte Carlo ES (Figure 5.4). The state-value function shown was computed from the action-value function found by Monte Carlo ES.

ϵ-greedy MC

Oftentimes it is not possible to initialize an episode arbitrarily. In this case we can explore by occaisionaly choosing actions at random:

$$\pi(s) = \begin{cases} \arg\max_a \hat{Q}(s,a) & \text{w.p. } 1-\epsilon \\ \text{random } a & \text{w.p. } \epsilon \end{cases}$$

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

Repeat forever:
    (a) Generate an episode using $\pi$
    (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:
            $a^* \leftarrow \arg\max_a Q(s,a)$
            For all $a \in \mathcal{A}(s)$:
                $\pi(s,a) \leftarrow \begin{cases} 1 - \epsilon + \epsilon/|\mathcal{A}(s)| & \text{if } a = a^* \\ \epsilon/|\mathcal{A}(s)| & \text{if } a \neq a^* \end{cases}$
```

$A(s)$ = set of possible actions in state s

Greedy: pick best action with prob $1-\varepsilon$, else pick action at random

(c) This is called an ______ control algorithm because we are exploring using the same policy we are learning

______ control algorithms learn (evaluate and improve) a different policy than what is used for exploration. For example, we might want to use an ϵ-greedy policy for exploration, but a greedy policy for actual planning. For such algorithms, the value function must be updated differently. See Sutton & Barto Ch 5 for details.

Pros and Cons of MC methods

Pros: no knowledge of MDP parameters required.
doesn't even assume process is Markovian
can learn from experience, simulated or real

Cons: doesn't apply to continuing tasks
only updates policy at the end of an episode

Temporal Difference Methods

Like MC, TD methods don't require knowledge of MDP parameters
Unlike MC, TD methods update before the end of episodes,
and are suitable for continuing tasks.

Policy Evaluation

Like MC, TD is an approach to policy evaluation

Updates have the form

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

where TARGET is an estimate of the expected return following state s_t.

The MC approach is to take TARGET $= R_t$, which means the update cannot be computed until the end of the episode.

Recall that for a policy π,

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\{R_t \mid s_t = s\} \\ &= E_{\pi}\{r_{t+1} + \gamma R_{t+1} \mid s_t = s\} \\ &= E_{\pi}\{r_{t+1} + \gamma V^{\pi}(s_{t+1}) \mid s_t = s\} \end{aligned}$$

This motives

Ⓓ TARGET =

and the TD(0) policy eval. rule:

$V(s_t) \leftarrow$

Note that TD(0) is a <u>bootstrapping</u> method: it computes new estimates from old ones. MC, on the other hand, does not bootstrap.

For fixed π, TD(0) converges to V^{π}. In practice, it usually converges faster than MC.

Sarsa : On Policy TD Control

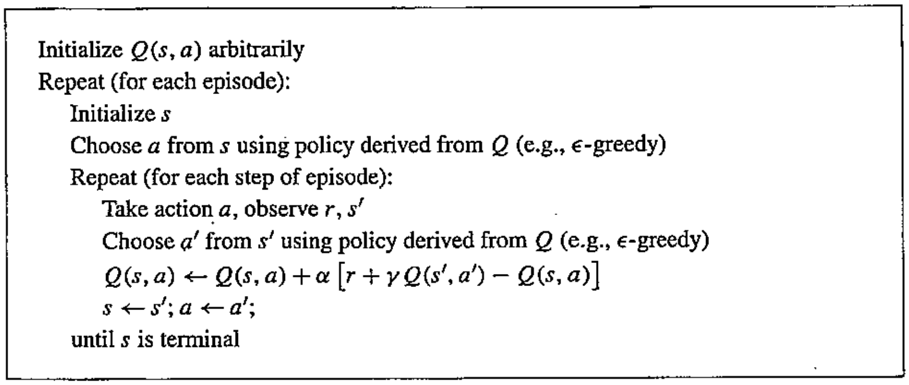
Ⓔ $Q(s_t, a_t) \leftarrow$

Uses the variables $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$, hence the name.

If s_{t+1} is terminal, $Q(s_{t+1}, a_{t+1}) = 0$.

Sarsa combines TD(0) action-value update with an ϵ-greedy policy, updated after each <u>step</u>.

It converges to π^* provided $\epsilon = 1/t$.



```
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
```

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. ■

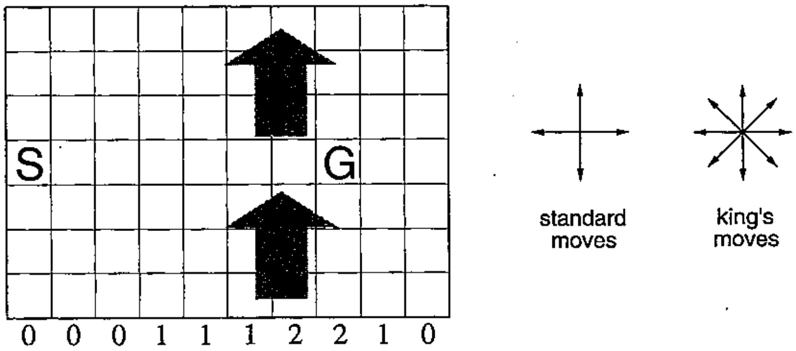


Figure 6.10 Gridworld in which movement is altered by a location-dependent, upward "wind."

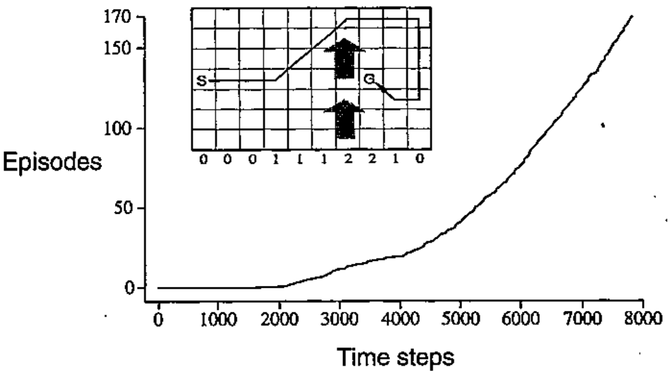


Figure 6.11 Results of Sarsa applied to the windy gridworld.

Q-Learning : Off-Policy TD Control

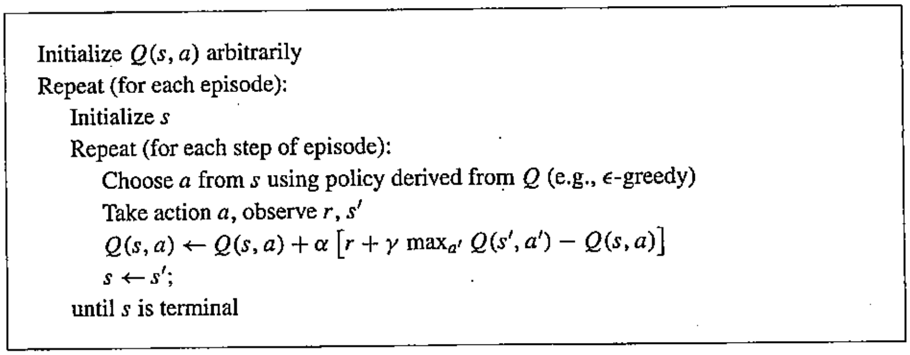
One-step Q-learning:

Ⓕ $Q(s_t, a_t) \longleftarrow$

Approximates Bellman optimality equation for Q^*

Analysis is relatively clean: Converges to Q^* w.p. 1 provided

- each state-action pair visited infinitely often (e.g. ϵ-greedy)
- $\alpha = \alpha_t \to 0$ at certain rate



```
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 $r, s'$
        $Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$
        $s \leftarrow s'$;
    until $s$ is terminal
```

Bridging MC and TD(0)

A natural extension of TD(0) is

$$V(s_t) \longleftarrow V(s_t) + \alpha \left[R_t^{(n)} - V(s_t)\right]$$

where

Ⓖ $R_t^{(n)} =$

Ⓗ Notice $n = 1 \Rightarrow$

$n = \infty \Rightarrow$

The larger n, the more we rely on the data, and the less we rely on the MDP model (from which we get $R_{t+n-1} \approx r_{t+n} + \gamma V(s_{t+n})$)

Also, the larger n, the longer we have to wait before updating.

λ – Returns

Taking it one step further, our TARGET could be any weighted combination of $R_t^{(n)}$s, e.g.

$$\text{TARGET} = 0.5\ R_t^{(1)} + .3\ R_t^{(3)} + .2\ R_t^{(6)}$$

For $0 \leq \lambda \leq 1$, let us define the $\underline{\lambda\text{-return}}$

Ⓘ $$R_t^{\lambda} :=$$

If T is the episode length, and $n \geq T-t$, then $R_t^{(n)} = R_T$ and

$$R_t^{\lambda} =$$

Notice $\lambda = 1 \Rightarrow$

$\lambda = 0 \Rightarrow$

The λ-return algorithm for policy evaluation is

$$V(s_t) \leftarrow V(s_t) + \alpha \left[R_t^\lambda - V(s_t) \right]$$

TD (λ)

The λ-return algorithm is, like MC, noncausal, and can only be performed at the end of an episode. (practically, we could treat large powers of λ as 0, but still we have to wait a few steps).

TD(λ) is an algorithm that is essentially a more efficient version of the λ-return algorithm. In fact, the two algorithms are the same in a certain setting (episodic tasks with batch updates).

TD(λ) performs

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

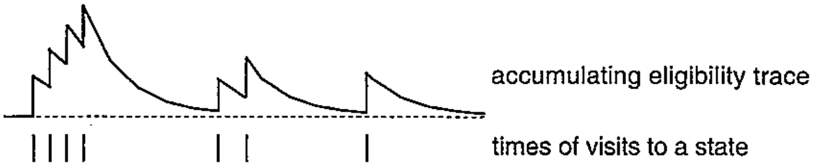
for all $s \in S$

where

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{if } s \neq s_t \\ \gamma\lambda e_{t-1}(s) + 1 & \text{if } s = s_t \end{cases}$$

Ⓘ is called the ______________ ____________ for state s at time t. Initialize $e_0(s) = 0$.

Unlike previous algorithm, we update $V(s)$ for all states at each t, but the weight given to a state depends on how often it has occurred recently.



Note that $\lambda = 0 \Rightarrow TD(0)$, while $\lambda = 1$ is "MC-like," in that more weight is given to $r_{t+2}, r_{t+3}, \ldots$ (because $e_t(s)$ decays more slowly). Thus, larger λ offers more "immunity" to non Markov processes. Furthermore, TD(1) applies to continuing tasks!

TD(λ) is suited to situations where data are scarce (states are rarely repeated). It is a compromise between

- TD (0) : looks only one step ahead
- MC : requires many examples of each state/action

Control : Can be extended to Sarsa (λ) or Q(λ)

$$\underset{\text{on policy}}{\overset{\uparrow}{\text{Sarsa}}}\ (\lambda) \quad \text{or} \quad \underset{\text{off-policy}}{\overset{\uparrow}{Q}}(\lambda)$$

Implementation: if $e_t(s) \approx 0$, which it will be for most states, don't need to update value function

$\boxed{\text{Key}}$

A. generalized policy iteration (GPI)

B. Initialize so that each (s,a) occurs with prob > 0 (exploring starts)

Randomized policies, such as ϵ-greedy

C. on-policy, off-policy

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

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

E. $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$

F. $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$

G. $R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V(s_{t+n})$

H. $n=1 \Rightarrow$ TD(0), $n=\infty \Rightarrow$ MC

I. $R_t^\lambda = (1-\lambda) \sum_{n=1}^{\infty} \lambda^n R_t^{(n)} = (1-\lambda) \sum_{n=1}^{T-t-1} \lambda^n R_t^{(n)} + \lambda^{T-t-1} R_T$

$\lambda = 1 \Rightarrow$ MC, $\lambda = 0 \Rightarrow$ TD(0)

J. Eligibility trace