Prof. Alfio Ferrara

Lab of Reinforcement Learning

Coding classes

Introduction

In this class, we explore Reinforcement Learning algorithms. The general goal of RL is to learn the optimal policy without knowing the MDP, in particular without any knowledge about:

- Transition probabilities: $p(\cdot \mid s, a)$
- Reward function: $r(s, a, s')$

Practically speacking this means the the only way we have to guess the optimal policy is by interacting with the environment.

Motivating example

The dungeon game. We are moving in a dungeon and we need to learn what's the best option to gain the exit (and gain a lot of gold!).

Q-learning and SARSA

Q-learning

The goal of Q-learning is learn $Q^*(s, a)$ by interacting with the environment and observing the reward. Note that, having $Q^*(s, a)$ implies also an optimal policy, since, at each state s, we can choose the best action by $a^* = \arg\max_{a \in A} Q^*(s, a)$.

The idea is to start from an initial stage where $Q(s, a) = 0 \; \forall (s, a) : a \in A(s)$ and $Q(s, a) = -\inf \; \forall (s, a) : a \notin A(s)$. The we interact with the environment and we update $Q(s, a)$ as follows:

$$\hat{Q}_{t+1} \leftarrow (1 - \eta)\, \hat{Q}_t(s, a) + \eta \left[r_t(s, a, s') + \gamma \hat{V}_t(s') \right] \tag{1}$$

recall that

$$\hat{V}_t(s') = \max_{a' \in A(s')} \hat{Q}_t(s', a') \tag{2}$$

note that

We observe (s, a, r, s'). Thus, we do not need a policy for choosing a', since we exploit what we have learned so far on all the possible $a' \in A(s')$ (**off-policy**).

Coding

```
Define Q(S, A) as a |S| x |A| table of -inf
Set Q(s, a) = 0 for all s, a such that a in A(s)
for each episode
  set start state s
  for each step of the episode
    choose action a using any policy derived from Q
    take action a and observe s' and r(s, a, s')
    update Q(s, a)
    s = s'
  break if converge
```

SARSA

With Q-learning we do not have a reference policy to update, but we just look at the value of our options along the episodes (analogous to value iteration). With SARSA, we look at the policy, try to figure out how to incrementally improve such a policy by interacting with the environment (analogous to policy iteration). That is:

$$\hat{Q}^{\pi}_{t+1} \leftarrow (1-\eta)\,\hat{Q}^{\pi}_{t}(s,\pi(s)) + \eta\left[r(s,\pi(s),s') + \gamma\hat{Q}^{\pi}_{t}(s',\pi(s'))\right] \tag{3}$$

where $a_t \sim \pi_t(s_t)$.

recall that

Having $Q(s,a)$ is like having a *greedy policy* where $\pi_t(s) = \arg\max_{a\in A(s)} Q_t(s,a)$

note that

We now observe (s,a,r,s',a'). In order to do this observation, we need a current policy. (**on-policy**).

Coding

```
Define Q(S, A) as a |S| x |A| table of -inf
Set Q(s, a) = 0 for all s, a such that a in A(s)
for each episode
  set start state s
  choose action a using the policy derived from Q
  for each step of the episode
    take action a and observe s' and r(s, a, s')
    choose action a' from state s' using the policy derived from Q
    update Q(s, a)
    s = s', a = a'
  break if converge
```

Epsilon-greedy policy

The crucial step in SARSA is that we choose the actions using Q. Now, if we always use a greedy approach in doing this, (i.e., $\pi_t(s) = \arg\max_{a \in A(s)} Q_t(s,a)$), we may never explore other potentially valuable actions available from s. In other terms, it would be a good idea to explore the environment sometimes. We can do this by exploiting an ϵ-greedy policy, defined as follows:

$$\pi(s) = \begin{cases} \arg\max_{a \in A(s)} Q(s,a) & \textit{with probability } 1-\epsilon \\ \text{random from } A(s) & \textit{with probability } \epsilon \end{cases} \tag{4}$$

A suitable choice for ϵ is

$$\epsilon_t(s) = \frac{1}{N_t(s)}, \tag{5}$$

where $N_t(s) \geq 1$ is a counter of how many times we visited s.

An example with Gym

See Taxi-driver (implementation with SARSA)

TD(λ)

In temporal difference learning, we use our predictions on $V(s_i)$ to estimate remaining reward without observing it (like in Monte Carlo).

Monte Carlo: $R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots$

TD: $R_t^{(1)} = r_{t+1} + \gamma V_t(s_{t+1})$

N-step TD: $R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V_t(s_{t+n})$

We can not compute

$$\Delta V_t(s_t) = \eta \left[R_t^{(n)} - V_t(s_t) \right] \tag{6}$$

We can combine different n rewards by taking their average, e.g.

$$R_t^{avg} = \frac{1}{2}\left(R_t^{(2)} + R_t^{(4)} \right) \tag{7}$$

By TD(λ) we take the average of all the n-steps

- Weight by λ^{n-1}
- Return $R_t^\lambda = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)}$
- $\Delta V_t(s_t) = \eta \left[R_t^\lambda - V_t(s_t) \right]$

Note that

- If $\lambda = 1$, we have Monte Carlo: $R_t^\lambda = (1-1)(1-\lambda) \sum_{n=1}^{\tau-t-1} 1^{n-1} R_t^{(n)} + 1^{\tau-t-1} R_t = R_t$

- If $\lambda = 0$, we have TD(0): $R_t^\lambda = (1-0)(1-\lambda)\sum_{n=1}^{\tau-t-1} 0^{n-1} R_t^{(n)} + 0^{\tau-t-1} R_t = R_t^1$

Backward mechanism

We define a variable $e_t(s)$ called **eligibility trace**

- On each stay, we reduce all the traces by $\gamma\lambda$ and we increment it for the current state by 1
- We accumulate the trace

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & if\ s \neq s_t \\ \gamma\lambda e_{t-1}(s) + 1 & if\ s = s_t \end{cases} \tag{8}$$

Use eligibility trace to compute $V(s)$ given a policy π

```
Initialize V(s) arbitrarily and e(s) = 0 for all s in S
For each episode:
  Init choose s
  For each step:
    a = policy(s)
    Take action a and observe s' and r(s, a, s')
    Update Delta(V(s))
    e(s) = e(s) + 1 # here we visited s
    For all s:
      Update V(s) = V(s) + eta * Delta * e(s)
      Update e(s) = gamma * lambda * e(s)
    s = s'
```

Using this mechanism, we can also implement a solution that does not require π:

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a) + 1 & if\ s = s_t \wedge a = a_t \\ \gamma\lambda e_{t-1}(s,a) & otherwise \end{cases} \tag{9}$$

Now, we can implement SARSA by:

$Q_{t+1}(s,a) = Q_t(s,a) + \eta\Delta_t e_t(s,a)$

$\Delta_t = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)$

See code for SARSA(λ) on the repository.