Reinforcement Learning **Spring Semester, 2018/9**

Lecture 6: April 3, 2019

Lecturer: Yishay Mansour *Scribe: ym*

In this and the next lecture we will look at model free learning algorithms. In this lecture we will look at the Q-learning algorithms, and its on-policy variant SARSA. We will also look at Monte-Carlo methods. In the next lecture we will look at temporal differences algorithms, $TD(\lambda)$, as well as evaluating one policy while following a different policy (using importance sampling). If we will have time next lecture, we will look also at actor-critic methodology.

6.1 Online approximation of mean

Before we start looking at model-free learning algorithms, we look at a very simple task, approximating the average of a random variable using samples. Our main goal would be to do it in an online way while maintaining minimal information, namely the average.

Assume we have a random variable $R \in [0,1]$ with $\mu = E[R]$. We observe T samples, $r_1, \ldots, r_T$. In the batch setting we simply compute the average of the samples, $\hat{\mu}_T = (1/T)\sum_{t=1}^{T} r_t$. We can bound $|\mu - \hat{\mu}_T|$ using Chernoff-Hoeffding concentration bounds, as we discussed last lecture, when considering model-based learning.

We can actually perform the computation of the average in an online way.

$$\hat{\mu}_T = \frac{1}{T}\sum_{t=1}^{T} r_t = \frac{T-1}{T}\hat{\mu}_{T-1} + \frac{1}{T}r_T = \hat{\mu}_{T-1} + \frac{1}{T}(r_T - \hat{\mu}_{T-1})$$

This lead naturally to the *exponential averaging* where we have a sequence of $\alpha_t \in (0,1)$ and

$$\bar{\mu}_T = \bar{\mu}_{T-1} + \alpha_T(r_T - \bar{\mu}_{T-1}) = \sum_{t=1}^{T} \beta_t r_t$$

where $\beta_t = \alpha_t \prod_{i=1}^{t-1}(1-\alpha_i)$. Note that $\sum_{t=1}^{T} \beta_t = 1$.

We would like to show that the approximation $\bar{\mu}$ concentrates around the mean μ. For this we will introduce the McDiarmid inequality.

The setting of the McDiarmid inequality is the following. There is a domain X, which can be for example $[0, 1]$. We have n random variables $X_i \in X$. There is a function f which maps the realization of the n random variables to a value. Namely, $f : X^n \to \mathbb{R}$.

We define c_i, the sensitivity of the function f to the i-th input, to be

$$c_i = \max_x \max_{a_1, a_0 \in X} |f(x_1, \ldots, x_{i-1}, a_0, x_{i+1}, \ldots, x_n) - f(x_1, \ldots, x_{i-1}, a_1, x_{i+1}, \ldots, x_n)|$$

The basic idea of McDimid's inequality is that with high probability the observed value of f is close to its expected value. Formally,

Lemma 6.1 (McDiarmid's inequality)**.**

$$\Pr[|f(x) - E[f(x)| \geq \epsilon] \leq e^{-2\epsilon^2/(\sum_i c_i^2)}$$

We can use McDirmid's inequality to recover the concentration bounds for a simple average. Define $avg(x_1, \ldots, x_n) = (1/n) \sum_{i=1}^n x_i$, where $x_i \in [0, 1]$. The sensitivity of avg to any input i is exactly $c_i = 1/n$. This implies that $\sum_i c_i^2 = \sum_i 1/n^2 = 1/n$. Therefore, McDiarmid's inequality gives us,

$$\Pr[|avg(x) - E[avg(x)]| \geq \epsilon] \leq e^{-2\epsilon^2 n}$$

Note that the bound is very similar to the Chernoff-Hoeffding bound.

We can now use the McDiarmid's inequality for the weighted average case. Define $wavg(x_1, \ldots, x_n) = \sum_{i=1}^n \beta_i x_i$, where $x_i \in [0, 1]$. The sensitivity of $wavg$ to any input i is exactly $c_i = \beta_i$. Therefore, McDiarmid's inequality gives us,

$$\Pr[|wavg(x) - E[wavg(x)]| \geq \epsilon] \leq e^{-\epsilon^2/(\sum_i \beta_i^2)}$$

For the case of exponential averaging, with a fixed parameter α, we have $\beta_t = \alpha(1 - \alpha)^{t-1}$. This implies that

$$\sum_{t=1}^T \beta_t^2 = \alpha^2 \frac{1 - (1 - \alpha)^T}{1 - (1 - \alpha)^2} \approx \frac{\alpha}{2 - \alpha}$$

This implies that if we like to have an accuracy ϵ with confidence $1 - \delta$ we can set $\alpha \approx \epsilon^2/(\log(1/\delta))$.

6.2 Q-Learning: Deterministic Decision Process

To demonstrate some key ideas of Q-learning, we start with a simplified learning algorithm that is suitable for a *Deterministic Decision Process (DDP)* model, namely:

$$s_{t+1} = f(s_t, a_t)$$
$$r_t = r(s_t, a_t)$$

We consider the discounted return criterion:

$$V^\pi(s) = \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\,, \quad \text{given } s_0 = s, a_t = \pi(s_t)$$
$$V^*(s) = \max_\pi V^\pi(s)$$

where V^* is the value function of the optimal policy.

Recall our definition of the Q-function (or *state-action value function*), specialized to the present deterministic setting:

$$Q^*(s,a) = r(s,a) + \gamma V^*(f(s,a))$$

The optimality equation is then

$$V^*(s) = \max_a Q^*(s,a)$$

or, in terms of Q^* only:

$$Q^*(s,a) = r(s,a) + \gamma \max_{a'} Q^*(f(s,a), a')$$

Our learning algorithm runs as follows:

- *Initialize:* Set $\hat{Q}(s,a) = Q_0(s,a)$, for all s, a.
- At each stage $t = 0, 1, \ldots$:
  - Observe s_t, a_t, r_t, s_{t+1}.
  - Update $\hat{Q}(s_t, a_t)$: $\hat{Q}_{t+1}(s_t, a_t) := r_t + \gamma \max_{a'} \hat{Q}_t(s_{t+1}, a')$

We note that this algorithm does not tell us how to choose the actions a_t.

Theorem 6.1 (Convergence of Q-learning for DDP)**.**
Assume a DDP model. If each state-action pair is visited infinitely-often, then $\lim_{t\to\infty} \hat{Q}_t(s,a) = Q^*(s,a)$*, for all* (s,a).

Proof. Let

$$\Delta_t \triangleq \|\hat{Q}_t - Q^*\|_\infty = \max_{s,a} |\hat{Q}_t(s,a) - Q^*(s,a)| \,.$$

Then at every stage t:

$$\begin{aligned} |\hat{Q}_{t+1}(s_t,a_t) - Q(s_t,a_t)| &= |r_t + \gamma \max_{a'} \hat{Q}_t(s_{t+1},a') - (r_t + \gamma \max_{a''} Q(s_{t+1},a''))| \\ &= \gamma |\max_{a'} \hat{Q}_t(s_{t+1},a') - \max_{a''} Q(s_{t+1},a'')| \\ &\le \gamma \max_{a'} |\hat{Q}_t(s_{t+1},a') - Q(s_{t+1},a')| \;\le\; \gamma \Delta_t \,. \end{aligned}$$

Consider now some interval $[t, t_1]$ over which all state-action pairs (s,a) appear at least once. Using the above relation and simple induction, it follows that $\Delta_{t_1} \le \gamma \Delta_t$. Since $\gamma < 1$ and since there is an infinite number of such intervals by assumption, it follows that $\Delta_t \to 0$, as t goes to infinity. □

6.3 Q-learning: Markov decision process

The Q-learning algorithm:

– initialize $\hat{Q}_0$.
– At time t: Observe (s_t, a_t, r_t, s_{t+1}), and let

$$Q_{t+1}(s_t,a_t) := Q_t(s_t,a_t) + \alpha_t(s_t,a_t)[r_t + \gamma \max_{a'} Q_t(s_{t+1},a') - Q_t(s_t,a_t)]$$

Let $\Gamma_t = r_t + \gamma \max_{a'} Q_t(s_{t+1},a') - Q_t(s_t,a_t)$. The basic intuition is that if $Q_t = Q^*$ then we have that $E[\Gamma_t] = 0$. The main challenge is that in the updates we use Q_t rather than Q^*. We also need to handle the stochastic nature of the updates, where there is both stochastic rewards and stochastic next state.

The next theorem states the main convergence property of Q-leanring.

Theorem 6.2 (Q-learning convergence)**.**
Assume every state-action pair (s,a) occurs infinitely often, and the step size $\alpha_t(s,a)$ has the properties: (1) $\sum_t \alpha_t(s,a) I(s_t = s, a_t = a) = \infty$, and (2) $\sum_t \alpha_t^2(s,a) I(s_t = s, a_t = a) = O(1)$,
Then, Q_t converges with probability 1 to Q^*

We will not give the details of the proof, but will only outline the main building blocks and the methodology.

6.3.1 Stochastic Approximation

There is a general framework of stochastic approximation algorithms. We will outline the main results of that literature, as we need it to show convergence of the learning algorithms.

The iterative algorithm takes the following general form:

$$X_{t+1} = (1 - \alpha_t(s))X_t(s) + \alpha_t(s)((HX_t)(s) + w_t(s))$$

where H is a (possibly non-linear) operator.

We will mainly look at (B, γ) *well behaved* iterative algorithms, where $B > 0$ and $\gamma \in (0, 1)$, which have the following properties:

1. Step size: (1) $\sum_t \alpha_t(s, a)I(s_t = s, a_t = a) = \infty$, and (2) $\sum_t \alpha_t^2(s, a)I(s_t = s, a_t = a) = O(1)$,
2. Noise: $E[w_t(s)|h_{t-1}] = 0$ and $|w_t(s)| \leq B$, where h_{t-1} is the history up to time t.
3. Contraction: There exists X^* such that for any X we have $\|HX - X^*\|_\infty \leq \gamma\|X - X^*\|_\infty$. (This implies that $HX^* = X^*$. Note that if H is a contracting operator, this property is guarantee to hols.)

The following is the convergence theorem for well behaved iterative algorithms.

Theorem 6.3 (Iterative Stochastic Approximation: convergence)**.**
Let X_t be a sequence that is generated by a (B, γ) well behaved iterative algorithm. Then X_t converges with probability 1 to X^*.

We will not give a proof of this important theorem, but we will try to sketch the main proof methodology.

There are two distinct parts to the iterative algorithms. The part (HX_t) is contracting, in a deterministic manner. If we had only this part (say, $w_t = 0$ always) then the contraction property of H will give the convergence (as we saw before). The main challenge is the addition stochastic noise w_t. The noise is unbiased, so on average the expectation is zero. Also, the noise is bounded by a constant B. This implies that if we average the noise, then the average should be very close to zero.

The proof works in phases. In each phase we have a deterministic contraction using the operator H. This implies that the deterministic contraction implies that the space contracts by $\gamma < 1$. We have to take care of the stochastic noise. We make the phase long enough so that the average of the noise is less than $(1 - \gamma)/2$ factor. This implies that the space contracts by $(1 + \gamma)/2 < 1$. This implies that each phase contracts by a constant and eventually we have convergence.

6.3.2 Q-learning as an iterative stochastic approximation algorithm

We first define the operator H.

$$(Hq)(s,a) = \sum_{s'} p(s'|s,a)[r(s,a) + \gamma \max_{a'} q(s',a')]$$

The contraction of H is established as follows,

$$\begin{aligned}
\|Hq_1 - Hq_2\|_\infty &= \gamma \max_{s,a} |\sum_{s'} p(s'|s,a)[\max_{b_1} q_1(s',b_1) - \max_{b_2} q_2(s',b_2)| \\
&\leq \gamma \max_{s,a} \max_{b,s'} |q_1(s',b) - q_2(s',b)| \\
&\leq \gamma \|q_1 - q_2\|_\infty
\end{aligned}$$

In this section we re-write the Q-learning algorithm to follow the iterative stochastic approximation algorithms, so that we will be able to apply Theorem 6.3.

Recall that

$$Q_{t+1}(s_t,a_t) := (1-\alpha_t(s_t,a_t))Q_t(s_t,a_t) + \alpha_t(s_t,a_t)[r_t + \gamma \max_{a'} Q_t(s_{t+1},a') - Q_t(s_t,a_t)]$$

Let $\Phi_t = r_t + \gamma \max_{a'} Q_t(s_{t+1},a')$. This implies that $E[\Phi_t] = (HQ_t)(s_t,a_t)$. We can define the noise term as $w_t(s_t,a_t) = \Phi_t - (HQ_t)(s_t,a_t)$ and have $E[w_t(s_t,a_t)] = 0$. In addition $|w_t(s_t,a_t)| \leq \frac{R_{max}}{1-\gamma}$.

We can now rewrite the Q-learning, as follows,

$$Q_{t+1}(s_t,a_t) := (1-\alpha_t(s_t,a_t))Q_t(s_t,a_t) + \alpha_t(s_t,a_t)[(HQ_t)(s_t,a_t) + w_t(s_t,a_t)]$$

In order to apply Theorem 6.3, we have the properties on the noise w_t, and of the contraction of H. Therefore, we can derive Theorem 6.2, since the step size requirement is part of the theorem.

6.3.3 Step size

We need for the step size to have two properties. The first is that $\sum_t \alpha_t(s,a) I(s_t = s, a_t = a) = \infty$, which intuitively implies that we can potentially reach any value. This is important, since we might have errors on the way, and this guarantees that the step sizes are large enough to possibly reach correct any error. (It does not guarantee that it will correct the errors, only that the step size is large enough to allow it.)

The second requirement from the step size is that $\sum_t \alpha_t^2(s,a) I(s_t = s, a_t = a) = O(1)$. This requirement is that the step size are not too large. It will guarantee that once we are close to the correct value, the step size will be small enough that we actually converge, and not bounce around.

Think of the following experiment. Suppose from some time t we update using only Q^*, then we clearly would like to converge. The large enough step size will guarantee that we will reach Q^* neighborhood. The small enough step size will guarantee that we will converge inside the neighborhood.

Many times step size are simply a function of the number of visits to (s,a), which we denote by $n(s,a)$, and this widely used in practice. Two leading examples are:

1. *Linear step size:* $\alpha_t(s,a) = 1/n(s,a)$. We have that $\sum_{n=1}^{N} 1/n = \ln(N)$ and therefore $\sum_{n=1}^{\infty} 1/n = \infty$. Also, $\sum_{n=1}^{\infty} 1/n^2 = \pi^2/6$

2. *Polynomial step size*: For $\theta \in (1/2, 1)$ we have $\alpha_t(s,a) = 1/(n(s,a))^{\theta}$. We have that $\sum_{n=1}^{N} 1/n^{\theta} \approx (1-\theta)^{-1} N^{1-\theta}$ and therefore $\sum_{n=1}^{\infty} 1/n^{\theta} = \infty$. Also, $\sum_{n=1}^{\infty} 1/n^{2\theta} \leq \frac{1}{2\theta - 1}$, since $2\theta > 1$.

The linear step size, although many times popular in practice, might lead to slow converges. Here is a simple example. We have a single state s and single action a and $r(s,a) = 0$. However, we start with $Q_0(s,a) = 1$. We will analyze the convergence with the linear step size. Our update is,

$$Q_t = (1 - \frac{1}{t}) Q_{t-1} + \frac{1}{t}[0 + \gamma Q_{t-1}] = (1 - \frac{1-\gamma}{t}) Q_{t-1}$$

When we solve the recursion we get that $Q_t = \Theta(1/t^{1-\gamma})$. [1] This implies that for $t \leq c(1/\epsilon)^{1/(1-\gamma)}$ we have $Q_t \geq \epsilon$.

In contrast, if we use a polynomial step size, we have,

$$Q_t = (1 - \frac{1}{t^{\theta}}) Q_{t-1} + \frac{1}{t^{\theta}}[0 + \gamma Q_{t-1}] = (1 - \frac{1-\gamma}{t^{\theta}}) Q_{t-1}$$

When we solve the recursion we get that $Q_t = \Theta(e^{-(1-\gamma)t^{1-\theta}})$. This implies that for $t \geq c\frac{1}{1-\gamma} \log^{1/(1-\theta)}(1/\epsilon)$ we have $Q_t \leq \epsilon$. This is a poly-logarithmic dependency on ϵ, which is much better. Also, not that θ is in our control, and we can set for example $\theta = 2/3$. The setting of γ has a huge influence on the objective function and the effective horizon.

[1] $Q_t = \prod_{i=1}^{t}(1 - (1-\gamma)/i) \approx \prod_{i=1}^{t} e^{-(1-\gamma)/i} = e^{-\sum_{i=1}^{t}(1-\gamma)/i} \approx e^{-(1-\gamma)\ln t} = t^{-(1-\gamma)}$.

6.4 SARSA: on-policy Q-learning

We would like to have an on-policy variant of Q-learning, which is called SARSA. The name comes from the fact that the feedback we observe $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$, ignoring the subscripts we have SARSA. Note that in this case the actions are actually under the control of the algorithm.

When designing the algorithm we need to think of two contradicting objectives in selecting the actions. The first is the need to explore, perform each action infinitely often. This implies that we need, for each state s, to have that $E[\sum_t \pi_t(a|s)] = \infty$. Then by Borel-Cantelli lemma we have with probability 1 an infinite number of times that we select action a in state s (actually, we need independence of the events, or at least a Martingale property, which holds in our case). On the other hand we would like not only our estimates to converge, as done in Q-learning, but also the return to be near optimal. For this we need the action selection to converge to being greedy with respect to the Q function.

The SARSA algorithm

– initialize $\hat{Q}_0$.
– At time t: Observe $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$, and let

$$Q_{t+1}(s_t, a_t) := Q_t(s_t, a_t) + \alpha_t(s_t, a_t)[r_t + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)]$$

where $a_{t+1} = \pi(s_{t+1}; Q_t)$, namely, our policy selects an action a_{t+1} for state s_{t+1}, dependent on the values of Q_t.

Selecting the action: We give two simple ways to select the action by $\pi(s; Q)$. Let

$$\bar{a} = \arg\max Q(s, a)$$

The ϵ_t-greedy, has as a parameter a sequence of ϵ_t and selects either: (1) with probability $1 - \epsilon_t$ sets $\pi(s; Q) = \bar{a}$, or (2) with probability $\epsilon_t/|A|$, selects $\pi(s; Q) = a$, for each $a \in A$. Common values for ϵ_t are linear, $\epsilon_t = 1/t$, or polynomial, $\epsilon_t = 1/t^\theta$.

The *soft-max*, has as a parameter a sequence of $\beta_t \geq 0$ and select $\pi(s; Q) = a$, for each $a \in A$, with probability $\frac{e^{\beta_t Q(s,a)}}{\sum_{a' \in A} e^{\beta_t Q(s,a')}}$. Note that for $\beta_t = 0$ we get the uniform distribution and for $\beta_t \to \infty$ we get the maximum. We would like the schedule of the β_t to go to infinity (become greedy) but need it to be slow enough (so that each action appears infinitely often).

SARSA convergence: Convergence of the Q_t to Q^* would follow immediately from the basic properties of the Q learning, we just need to guarantee that we explore each action infinitely often. Then, the result will follow immediately from the convergence of Q-learning.

The convergence of the return of SARSA to that of Q^* is more delicate. Recall that we do not have such a claim about Q-learning, since it is an off-policy method. For the convergence of the return we need to make our policy 'greedy enough', in the sense that it has enough exploration, but guarantees a high return through the greedy actions.

Expected SARSA algorithm: In SARSA there are two sources of errors, or stochastic behavior. One is from the environment, through the selection of the next state and the rewards, both not under our control. The other is from our policy, which select a stochastic action. We will attempt to reduce that part of the noise in the following variation of the SARSA algorithm.

The Expected SARSA algorithm:
– initialize $\hat{Q}_0$.

– At time t: Observe (s_t, a_t, r_t, s_{t+1}), and let

$$Q_{t+1}(s_t, a_t) := Q_t(s_t, a_t) + \alpha_t(s_t, a_t)[r_t + \gamma E_{a\sim\pi(s_{t+1};Q_t)} Q_t(s_{t+1}, a) - Q_t(s_t, a_t)]$$

Select $a_{t+1} = \pi(s_{t+1}; Q_t)$

The min idea is to replace the sample a_{t+1} in the estimate of Q_{t+1}, by an averaging over a_{t+1}. We can do it, since we define the policy $\pi(s; Q)$. By the averaging, Expected SARSA reduces the variance of the estimates and hopefully converges faster.

6.5 Monte-Carlo Algorithm

Monte-Carlo methods learn directly from experience in a model free way. The idea is very simple. In order to estimate the value of a state under a given policy, i.e., $V^\pi(s)$, we consider runs of the policy π from state s and average them. the method does not assume any dependency between the different states, and does not even assume a Markovian environment, which is both a plus (less assumptions) and a minus (longer time to learn). We will concentrate on the case of an episodic MDP, namely, generating finite length episodes in each run. A special case of an episodic MDP is a finite horizon return, where all the episodes can be viewed as having the same length.

Assume we have a fixed policy π, which for each state s selects action a with probability $\pi(a|s)$. Using π we generate an episode $(s_1, a_1, r_1, \ldots, s_k, a_k, r_k)$. The observed return of the episode is $G = \sum_{i=1}^{k} r_i$. We are interested in the expected return of an episode conditioned on the initial state, i.e., $V^\pi(s) = E[\sum_{i=1}^{k} r_i | s_1 = s]$. Note that k is a random variable, which is the length of the episode.

Fix a state s, and assume you observed returns $G_1, \ldots, G_m$, all starting at state s. The Monte-Carlo estimate for the state s would be $\hat{V}^\pi(s) = \frac{1}{m} \sum_{i=1}^{m} G_i$. The main issue that remains is how do we generate the samples G_i for a state s. Clearly, if we assume we can reset the MDP to any state, we are done, However, we do not want to assume that we can reset the MDP to any state s and start an episode from there.

6.5.1 Generating the samples

Initial state only We use only the initial state of the episode. Namely, give an episode $(s_1, a_1, r_1, \ldots, s_k, a_k, r_k)$ we update only $\hat{V}^\pi(s_1)$. This is clearly an unbiased estimate, but has many drawbacks. First, most likely it is not the case that every state can be an initial state, what do we do with such states. Second, it seems very wasteful, updating only a single state per episode.

First visit We update every state that appears in the episode, but update it only once. Namely, give an episode $(s_1, a_1, r_1, \ldots, s_k, a_k, r_k)$ for each state s that appear in the episode, we consider the first appearance of s, say s_j, and update $\hat{V}^\pi(s)$ using $G = \sum_{i=j}^{k} r_i$. Namely, we compute the actual return from the first visit to state s, and use it to update our approximation.

Every visit We do an update during each step of in the episode. Namely, give an episode $(s_1, a_1, r_1, \ldots, s_k, a_k, r_k)$ for each state s_j that appear in the episode, we update each $\hat{V}^\pi(s_j)$ using $G = \sum_{i=j}^{k} r_i$. Namely, we compute the actual return from every state s_j to the end, and use it to update our approximation. Note that a state can be updated multiple times using this approach.

6.5.2 First versus Every visit

We consider a simple test case. We have a two state MDP, actually a Markov Chain. In the initial state s_1 we have a reward of 1 and with probability $1 - p$ we stay in that state and with probability p move to the terminating state s_2. See Figure 6.1.

The expected value is $V(s_1) = 1/p$, which is the expected length of an episode. (Note that the return of an episode is its length, since all the rewards are 1.) Assume

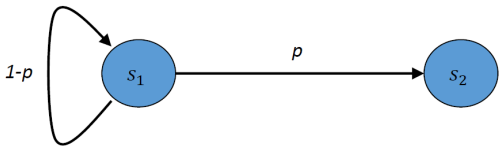

Figure 6.1: The situated agent

we observe a single trajectory, $(s_1, s_1, s_1, s_1, s_2)$, and all the rewards are 1. What would be a reasonable estimate for the expected return from s_1.

`First visit` take the naive approach, considers the return from the first occurrence of s_1, which is 4, and uses this as an estimate. `Every visit` considers three runs from state s_1, we have: $(s_1, s_1, s_1, s_1, s_2)$ with return 4, (s_1, s_1, s_1, s_2) with return 3, (s_1, s_1, s_2) with return 2, and (s_1, s_2) with return 1. `Every visit` averages the four and has $G = (4 + 3 + 2 + 1)/4 = 2.5$. On the face of it, the estimate of 4 seems to make more sense.

6.5.3 Maximum likelihood MDP model

We can consider what is the maximum likelihood model for the MDP. Namely, what value of p would maximize the probability of observing the sequence $(s_1, s_1, s_1, s_1, s_2)$. The likelihood of the sequence is, $(1 - p)^3 p$. We like to solve for

$$p^* = \arg\max (1 - p)^3 p$$

Taking the derivative we have $(1 - p)^3 - 3(1 - p)^2 p = 0$, which give $p^* = 1/4$.

For the maximum likelihood model ML we have $p^* = 1/4$ and therefore $V(s_1; ML) = 4$.

In general the Maximum Likelihood model value does not always coincide with the `First Visit` Monte-carlo estimate. However we can make the following connection.

Clearly, when updating state s using `First Visit`, we ignore all the episodes that do not include s, and also for each the remaining episode, that do include s, we ignore the prefix until the fist appearance of s. Lets modify the sample by deleting those parts (episodes in which s does not appear, and for each episode that s appears, start it at the first appearance of s). Call this the *reduced sample*.

Theorem 6.4. *Let M be the maximum likelihood MDP for the reduced sample. The expected value of s in M, i.e., $V(s; M)$, is identical to the* `First Visit` *estimate of s.*

6.5.4 Every visit minimizes MSE

Recall that the MSE is summing over all the observation the squared error. Assume we have $s_{i,j}$ as the j-th state in the i-th episode, and it has reward $r_{i,j}$. Let $\hat{V}(s_{i,j})$ be the estimate that `Every Visit` for state $s_{i,j}$. (Note that states $s_{i,j}$ are not unique, and we have $s = s_{i_1,j_1} = s_{i_2,j_2}$.) The square error is

$$SE = \sum_{i,j} (\hat{V}(s_{i,j}) - r_{i,j})^2$$

For a fixed state s we have

$$SE(s) = \sum_{i,j:s=s_{i,j}} (\hat{V}(s) - r_{i,j})^2$$

To minimize the square error of s we have

$$V^{se}(s) = \frac{\sum_{i,j:s=s_{i,j}} r_{i,j}}{|(i,j) : s = s_{i,j}|},$$

which is exactly the `Every Visit` Monte-carlo estimate.

6.5.5 First versus Every visit

The `First Visit` updates are unbiased, since the different trajectories are from different episodes. For each episode that update is an independent sample of the return.

For `Every Visit` the situation is more complicated, since there are different updates from the same episode, and therefore they are dependent. Consider the case of Figure 6.1. For a single episode of length k we have that the sum of the rewards is $k(k+1)/2$, since there are updates of lengths $k, \ldots, 1$. The number of updates is k, so we have that the estimate of a single episode is $(k+1)/2$. When we take the expectation we have that $E[(k+1)/2] = (1/p+1)/2$ which is different from the expected value of $1/p$. (Note that the `Every Visit` updates using k and $E[k] = 1/p$ which is also the expected value.)

When we have multiple episodes, the situation gets better. The reason is that we sum separately the rewards, and the number of occurrences. This implies that we have

$$E[V^{ev}(s_1) = \frac{E[k(k+1)/2]}{E[k]} = \frac{1}{p},$$

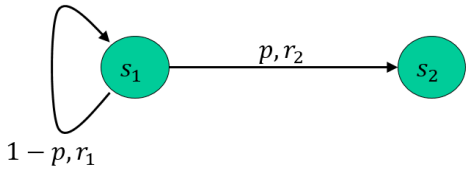

Figure 6.2: The situated agent

since $E[k^2] = 2/p^2 - 1/p$. This implies that if we average many episodes we will get an almost unbiased estimate using `Every Visit`.

We did all this on the example of Figure 6.1, but this indeed generalizes. Given an arbitrary episodic MDP, consider the following mapping. For each episode, mark the places where state s appears (the state we want to approximate its value). We now have a distribution of rewards from going from s back to s. Since we are in an episodic MDP, we also have to terminate, and for this we can add another state, from which we transition from state s and have the reward distribution as the rewards from the last appearance of s until the end of the episode.

This implies that we have two states MDP as described in Figure 6.2.

For this MDP, the value is $V(s_1) = \frac{1-p}{p} r_1 + r_2$. The single episode expected estimate of `Every Visit` is $V(s_1) = \frac{1-p}{2p} r_1 + r_2$. The m episodes expected estimate of `Every Visit` is $V(s_1) = \frac{m}{m+1} \frac{1-p}{p} r_1 + r_2$. This implies that if we have a large number of episodes the bias of the estimate becomes negligible.

6.5.6 Monte-Carlo control

The main idea is to learn the Q function. Simply we do an update for every (s, a). (The updates can be either `Every Visit` or `First Visit`.) The problem is that we need the policy to be "exploring", otherwise we will not have enough information about the actions the policy does not perform.

For the control, we can maintain an estimates of the Q function. Each time we reach a state s, we select a "near-greedy" action, for example, use ϵ-greedy.

We will show that updating from one ϵ-greedy policy to another ϵ-greedy policy, using policy improvement, does increase the value of the policy.

Recall that an ϵ-greedy policy, can be define in the following way. For every state s there is an action $\bar{a}_s$, which is the preferred action. The policy does the following: (1) with probability $1-\epsilon$ selects action $\bar{a}$. (2) for each action $a \in A$, with probability $\epsilon/|A|$, select action a.

Assume we have an ϵ-greedy policy π_1. We compute Q^{π_1} and define π_2 to be ϵ-greedy with respect to Q^{π_1}.

Theorem 6.5. *For any ϵ-greedy policy π_1, the ϵ-greedy improvement policy π_2 has $V^{\pi_2} \geq V^{\pi_1}$.*

Proof. Let $\bar{a}_s = \arg\max_a Q^{\pi_1}(s,a)$.

$$\begin{aligned} E_{a\sim\pi_1(\cdot|s)}[Q^{\pi_1}(s,a)] &= \sum_{a\in A} \pi_2(a|s) Q^{\pi_1}(s,a) \\ &= \frac{\epsilon}{|A|} \sum_{a\in A} Q^{\pi_1}(s,a) + (1-\epsilon) Q^{\pi_1}(s,\bar{a}_s) \\ &\geq \frac{\epsilon}{|A|} \sum_{a\in A} Q^{\pi_1}(s,a) + (1-\epsilon) \sum_{a\in A} \frac{\pi_1(a|s) - \epsilon/|A|}{1-\epsilon} Q^{\pi_1}(s,a) \\ &= \sum_{a\in A} \pi_1(a|s) Q^{\pi_1}(s,\bar{a}) = V^{\pi_1}(s) \end{aligned}$$

The inequality follows, since we are essentially concentrating of the action that $\pi^1(\cdot|s)$ selects with probability $1-\epsilon$, and clearly $\bar{a}_s$, by definition, guarantees a higher value.

From the basic policy improvement properties we have that

$$V^{\pi_2} \geq \max_a Q^{\pi_1}(s,a) \geq E_{a\sim\pi_1(\cdot|s)}[Q^{\pi_1}(s,a)] \geq V^{\pi_1}.$$

□

6.5.7 Monte-Carlo: pros and cons

The main benefits of the Monte-Carlo updates are:

1. Very simple and intuitive
2. Does not assume the environment is Markovian
3. Extends naturally to function approximation (more in future lectures)
4. Unbiased updates (using `First Visit`).

The main drawback of the Monte-Carlo updates are:

1. Suited mainly for episodic environment
2. Need to wait for the end of episode to update.
3. Biased updates (using `Every Visit`).

6.6 Bibliography Remarks

The Q-learning outline of the asymptotic convergence and the step size analysis follows [1].

Expected SARSA was presented in [4]

The comparison of the `First Visit` and `Every Visit` is based on [2].

Part of the outline borrows from David Silver class notes and the the book of Sutton and Barto [3].

Bibliography

[1] Eyal Even-Dar and Yishay Mansour. Learning rates for q-learning. *Journal of Machine Learning Research*, 5:1–25, 2003.

[2] Satinder P. Singh and Richard S. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22(1-3):123–158, 1996.

[3] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press, 1998.

[4] Harm van Seijen, Hado van Hasselt, Shimon Whiteson, and Marco A. Wiering. A theoretical and empirical analysis of expected sarsa. In *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning, ADPRL 2009, Nashville, TN, USA, March 31 - April 1, 2009*, pages 177–184, 2009.