Q & A - Reinforcement Learning

Q-Learning vs SARSA

Describe the differences existing between the Q-Learning and SARSA algorithm

SARSA and Q-Learning are two algorithm used to o control using the model free method called temporal difference.

Q-learning is an example of off-policy learning which means that it learns a target-policy (denoted by π) while following a behavioral policy $\overline{\pi}$. It means that we are, for example, using old policies to learn a new policy, or we are learning a new policy from observing other agents.

Q-Learning Update Function:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\big(R_{t+1} + \gamma \max_{a' \in A} Q(S_{t+1}, a') - Q(S_t, A_t)\big)$$

π is greedy (there is a max) and $\overline{\pi}$ is ϵ-greedy.

SARSA is an example of on-policy learning, so it learns the optimal policy based on the actions performed following its own policy. It samples A_{t+1} from its own policy.

SARSA Update Function:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$$

The two algorithms can perform differently in given situations, for example, in class, we have seen the cliff walking problem. Q-Learning learns an optimal policy along the edge of the cliff because the behavioral policy is ϵ-greedy; while SARSA learnt a safe non-optimal policy away from the edge. This means that if we adopt an ϵ-greedy behavioral policy for Q-Learning, and we use an ϵ-greedy policy for SARSA, we have that:

- If $\epsilon \neq 0$ SARSA performs better online
- if $\epsilon \to 0$ gradually both converge to the optimal policy.

Montecarlo vs Temporal Difference

Describe the differences existing between the Montecarlo and the Temporal Difference methods in the model-free estimation of a value function for a given policy.

Before diving into the differences let's show the update equations for both algorithms. TD(0) will be used as reference for TD algorithms

MC update equation

$$V(s_t) \leftarrow V(s_t) + \alpha\Big(v_t - V(s_t)\Big)$$

$v_t = G_t = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T$

TD(0) update equation

$$V(s_t) \leftarrow V(s_t) + \alpha\Big(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\Big)$$

The main difference between these algorithms is that **Monte Carlo** has **lower bias** and **higher variance**, while **TD** has **higher bias** but **lower variance**. This is due to the fact that **MC** return depends on **many** random actions, transitions, and rewards, while **TD**, the one-step version, depends only on **one** random actions, transition, reward (obviously if we considered the 2-step version we would have had more random variables and so on).
Since TD is much lower variance, it is **more efficient**.

One other important difference is that **MC** does **not** exploit the **Markov property**, while **TD does**. This is due to the fact that TD's value function update depends on the estimate of the return from neighbor states, while MC does not.

Moreover **MC** can learn only from **episodic** environments, **complete** sequences (it needs an episode to be over in order to update its value function wrt it), while **TD** can learn from **incomplete** and **continuing** environments. This means that **MC** must learn **offline**, and that **TD** can learn **online**.

Other less important differences are that MC is less sensitive to **value initialization** than TD and that MC performs better than TD with **function approximation**.

They both have good convergence properties but Monte Carlo requires a lot of episodes in order to converge.

On-Policy vs Off-Policy

Describe the difference between on-policy and off-policy reinforcement learning techniques. Make an example of an on-policy algorithm and an example of an off-policy algorithm.

The difference between Off and On policy techniques is the following:
On-policy learning "learns on the job". The policy that I'm following is the policy that I'm learning about.
It learns about policy π from experience sampled from π itself.

If with an on-policy algorithm we are following an ϵ-greedy policy with a constant ϵ we learn a non-optimal policy, while if ϵ decreases over time it converges to the optimal policy.

An example of on-policy technique is *SARSA Algorithm*.

SARSA update function (on-policy):
$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$

Off-policy learning "learns over someone's shoulders". It learns about the **target policy** $\pi(a|s)$ while following a **behavior policy** $\bar{\pi}(a|s)$.
Off policies learn from observing humans or other agents.
They re-use experience generated from old policies $\pi_1, \pi_2, \ldots, \pi_{t-1}$ in order to generate the new target policy π.

the best known example of why off-policy learning is used is the one regarding the exploration-exploitation tradeoff. We can follow an exploratory policy and at the same time learn about the optimal policy.
Another interesting use of off-policy learning is wanting to learn about multiple policies while following one: there might be many different behaviors we want to figure out.

If with an off-policy algorithm we adopt for the behavioral policy an ϵ-greedy policy (which is always the case) and ϵ is constant, we do learn the optimal policy. Obviously we do so even if ϵ decreases over time.

An example of off-policy technique is *Q-Learning*.

Q-Learning update function (off-policy) :
$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_{a' \in A} Q(S_{t+1}, a') - Q(S_t, A_t))$

UCB1 Algorithm

Describe the UCB1 algorithm. Is it a deterministic or a stochastic algorithm?

UCB1 is an algorithm to solve stochastic MAB problems through a frequentist approach.
In order to evaluate the expected reward of an arm, that we assume distributed as a Bernoulli, we do the following:

Evaluate for each arm

- its sample reward given its past rewards
- a bound

and then pick the arm whose $\hat{R} + B$ is the highest. Repeat the process.

In formulas:
For each time step t:

1. Compute $\hat{R}_t(a_i) = \frac{\sum_{i=1}^{t} r_{i,t} \mathbf{1}\{a_i = a_{i_t}\}}{N_t(a_i)} \; \forall a_i$
2. Compute $B_t(a_i) = \sqrt{\frac{2 \log t}{N_t(a_i)}} \; \forall a_i$
3. Play arm $a_{it} = \arg\max_{a_i \in A} \left(\hat{R}_t(a_i) + B_t(a_i) \right)$

Upperbound

Theorem:

At finite time T, the expected total regret of the UCB1 algorithm applied to a stochastic MAB problem is

$$L_t \leq 8 \log T \sum_{i | \Delta_i > 0} \frac{1}{\Delta_i} + \left(1 + \frac{\pi^2}{3}\right) \sum_{i | \Delta_i > 0} \Delta_i$$

where $\Delta_i = R^* - R(a_i)$, and R^* is the reward obtained by performing the best action.

the first term is our expected loss, and the second is our risk.

Since the choice of the arm to be pulled is deterministic, it's a deterministic algorithm.