EE675A

Lecture 17: SARSA, Expected SARSA, Q-learning

22/03/2023

Lecturer: Subrahmanya Swamy Peruru   Scribe: Lucky Kant Nayak and Prajeeth Babu Kodru

1 TD based control

TD based control algorithm is an algorithm that learns to control an agent's behavior by updating its policy based on the difference between the expected and actual reward obtained in each time step. It is used to find optimal policy.

Previously in DP based methods we've seen the generalized policy iteration where we can approximately compute V_π and improve on the current policy.

Given a policy π, we can estimate q_π using DP by the equation

$$q_\pi(s,a) = R_s^a + \gamma\Sigma_{s'} V_\pi(s') P_{ss'}^a = R_s^a + \gamma\Sigma_{s'}\Sigma_a \pi(a|s') q_\pi(s'|a') P_{ss'}^a \tag{1}$$

We dont know the dynamics of the MDP since we are working in model-free setting. So we work with Q-functions instead of value functions.

2 SARSA

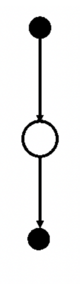
The SARSA (State-Action-Reward-State-Action) algorithm is a TD-based control algorithm that learns an action-value function Q(s,a) for each state-action pair.

TD target is a combination of the current reward and the estimated value of the next state or state-action pair, depending on whether we are estimating the value function or the action-value function. The TD target for SARSA can be expressed as:

$$\text{TD target} = R + \gamma Q(s', a') \tag{2}$$

Where TD target is the target value for the current state-action pair (s, a), R is the immediate reward obtained from the current state-action pair, gamma is the discount factor, s' is the next state, and a' is the next action that the agent will take based on the policy being followed. SARSA finds an optimal policy by updating the value function of a state-action pair based on the rewards received for a sequence of state-action pairs(n state-action pairs for n-step SARSA). The algorithm updates Q(s,a) based on the difference between the estimated value of the current state-action pair and the value of the next state-action pair, using the following update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(R + \gamma Q(s', a') - Q(s,a)\right) \tag{3}$$


Sarsa

SARSA is an on-policy algorithm, which means that it updates the action-value function and chooses the next action using the same policy. "TD target" is the expected future reward estimated by the agent. By iteratively updating the Q-value function using the TD target, SARSA learns to choose the best action in a given state, under the current policy being followed. The algorithm is given as follows:

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

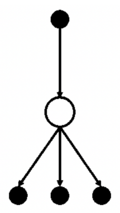
```
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Loop for each step of episode:
        Take action $A$, observe $R$, $S'$
        Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until $S$ is terminal
```

3 Expected SARSA

Expected SARSA is an algorithm that learns an optimal policy by iteratively updating an estimate of the action-value function, similar to the SARSA algorithm. However, instead of using the observed next action to compute the target value, it computes the expected value of the action over all possible actions, weighted by their probabilities under the current policy. The Expected SARSA algorithm updates the estimate of the action-value function $Q(s, a)$ using the following update rule:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \sum_{a'} \pi(a'|s')Q(s', a') - Q(s, a)) \tag{4}$$


Expected Sarsa

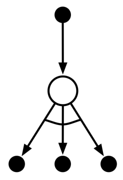
The key difference between SARSA and Expected SARSA is the way in which the target is selected and next (s', a') pair to be updated . SARSA chooses the next action according to the behaviour policy, while Expected SARSA computes the expected value of the action over all possible actions, weighted by their probabilities under the current policy. SARSA chooses the next action a' in the next state s' (next (s', a') pair to be updated) before updating the current $Q(s, a)$ value but in Expected SARSA the next action a' is choosen after updating the $Q(s, a)$ value .

4 Q-Learning

Q-learning is a sample-based method, off policy that estimates the optimal action-value function $Q^*(s,a)$ through an iterative update process, and can be seen as a variant of value iteration. In contrast to value iteration, which computes the value function by performing a complete backup of all states and actions at each iteration, Q-learning learns by updating the action-values based on sample transitions obtained by interacting with the environment.

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right) \tag{5}$$

where $Q(s,a)$ is the action-value of taking action a in state s at time t, α is the learning rate , r is the reward received at time $t+1$, γ is the discount factor, and s' is the state transitioned to from s by taking action a. The Q-value of a state-action pair is updated using the difference between Q-learning target and current estimate of that pair:


Q-learning

$$\text{Q-learning Target} = r + \gamma \max_{a'} Q(s',a') \tag{6}$$

In Q-learning, the agent follows a behavior policy, which is used for generating sample transitions and is designed for exploration. The behavior policy can be any policy that covers the entire state-action space, such as an ϵ-greedy policy, where with probability ϵ the agent chooses a random action and with probability $1-\epsilon$ it chooses the action with the highest Q-value.

The target policy, on the other hand, is the policy being learned and is used to update the value function. In Q-learning, the target policy is the greedy policy that selects the action with the highest Q-value for a given state. The target policy can be different from the behavior policy, which can lead to better exploration and more accurate value estimates. This approach is known as off-policy learning.

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

```
Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
        $S \leftarrow S'$
    until $S$ is terminal
```

Algorithm 1 Q-Learning Algorithm

```
 1: Initialize the Q-table with zeros for all state-action pairs
 2: Set the learning rate: $0 < \alpha \leq 1$
 3: Set the discount factor: $0 \leq \gamma \leq 1$
 4: Set the exploration rate: $0 < \epsilon \leq 1$
 5: for each episode do
 6:     Initialize the current state: $s \in S$
 7:     while the episode has not ended do
 8:         Choose the next action $a$ using an epsilon-greedy policy based on the Q-table and the
    exploration rate $\epsilon$
 9:         Take the action $a$ and observe the next state $s'$
10:         Receive the reward $r$
11:         Update the Q-value for the current state-action pair:
12:             $Q(s,a) \leftarrow Q(s,a) + \alpha\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$
13:         Set the current state to the next state: $s \leftarrow s'$
14:     end while
15:     Update the exploration rate: $\epsilon \leftarrow \epsilon \times \epsilon_{decay}$
16: end for
```

Note that S is the set of all possible states, A is the set of all possible actions, and $Q(s, a)$ is the Q-value for state s and action a. The epsilon-greedy policy selects the action with the highest Q-value with probability $1 - \epsilon$ and a random action with probability ϵ. The parameter ϵ_{decay} is used to decrease the exploration rate over time.

The material is primarily based on the content from [1]

References

[1] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press, second edition, 2018.