CMSC 352: Machine Learning
Q-Learning Exercise

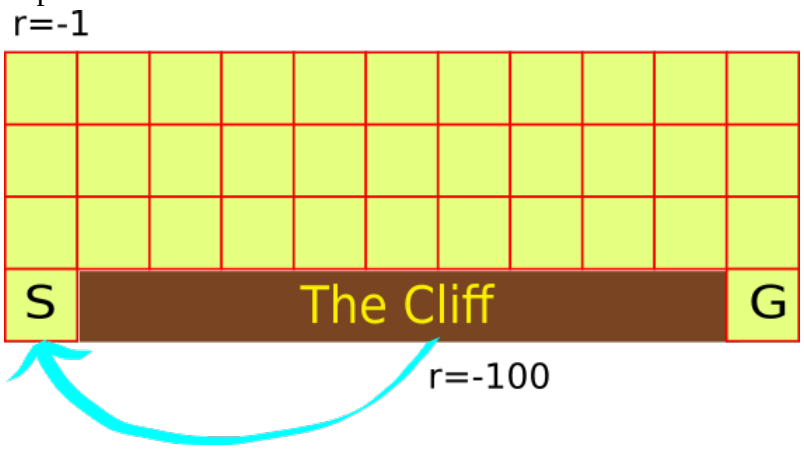
Q-learning is an off-policy algorithm, since it updates the Q-value at a state-action pair (s,a) using Q(s,a) and Q(s',a') where a' is the optimal action leaving state s'. This is not necessarily the action it will actually take when leaving s'! A common policy is epsilon-greedy, meaning that the algorithm takes the best current action (1-epsilon) of the time and a random action epsilon% of the time.

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Choose a from s using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ε-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

Figure 6.9: Sarsa: An on-policy TD control algorithm.

An on-policy algorithm known as Sarsa is subtly different (Fig 6.9 above). Rather than using the best action from state s', it uses whatever the policy dictates as its **actual** next action.

I have implemented the cliff-world problem of Sutton & Barto shown below. This is an undiscounted, episodic task with a start and goal states. The actions are up, down, right, and left. Reward is -1 for all transitions except those marked "Cliff" and "G". Stepping into the cliff region yields reward -100 and sends the agent back to the start, S, terminating the episode. Moving to the G state has a reinforcement of 0 and terminates the episode.



The `cliff.py` code provided implements a Q-learning solution to this problem. Note that the optimal actions derived from the Q-table are inverted relative to the figure here.

Directions

Modify `cliff.py` to implement the Sarsa solution to this problem. I recommend you find the TODO comment in the code and implement the solutions they suggest. You may like to revisit Figure 6.9 above to help you.

1. If you leave epsilon=-0.1, what are the two optimal policies found by the two algorithms? Note that getting optimal solutions can be tricky with reinforcement learning, so you may need to make numerous runs with slightly different parameters. Explain how they differ and why they differ.

2. If epsilon is slowly decreased from episode to episode, do the two algorithms yield the same optimal sequence of actions from S to G? Discuss.