COL864: Special Topics in AI
Semester II, 2020-21

Reinforcement Learning II – Model-free Methods

Rohan Paul

Outline

- Last Class
  - Reinforcement Learning
  - Model-Based Reinforcement Learning
- This Class
  - Model-Free Reinforcement Learning
- Reference Material
  - Please follow the notes as the primary reference on this topic.

Acknowledgements

These slides are intended for teaching purposes only. Some material has been used/adapted from web sources and from slides by Nicholas Roy, Wolfram Burgard, Dieter Fox, Sebastian Thrun, Siddharth Srinivasa, Dan Klein, Pieter Abbeel, Max Likhachev and others.

Model-Based RL vs. Model-Free RL

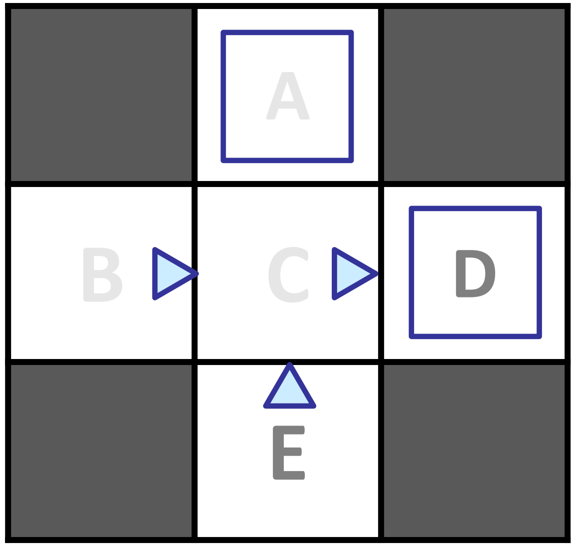
- Model-Based RL
  - Used data to infer a model and compute a policy
  - Problem
    - Storing the model (and computing the policy) can be difficult.
- Can we compute a policy in a way that “avoids” storing the model?
  - Note that we still need to store “some” statistics over our experience
  - Hence, maintain an estimate of the “value” function.
- Model-Free RL
  - Bypass explicit learning of the intermediate model.
  - Can sample trajectories from the world directly and estimate a value function without the model.

Monte Carlo Methods

- Learning the state-value function for a given policy
  - What is the value of a state?
    - Expected return – expected cumulative future discounted reward
- Key Idea
  - Sample trajectories from the world directly and estimate a value function without a model
  - Simply average the returns observed after visits to that state.
  - As more returns are observed the average should converge to the expected value.
  - Each occurrence of a state in an episode is a called a visit to the state.

Toy Example: Monte Carlo Method

Input Policy π



Assume: $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

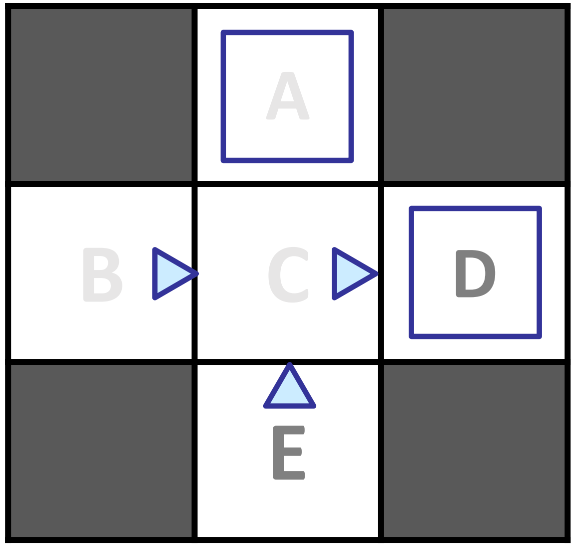
Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Value/utility of state c
$V^{\pi}(C) = ((9 + 9 + 9 + (-11))/4)$
$= 4$

Toy Example: Monte Carlo Method

Input Policy π



Assume: $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

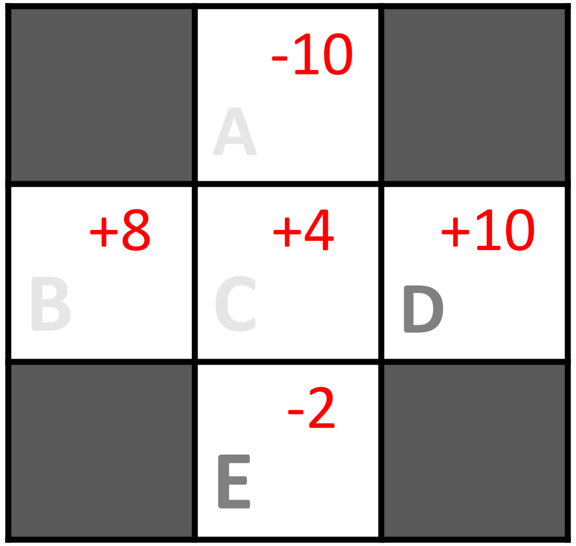
Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Value/utility of state c

$V^{\pi}(C) = ((9 + 9 + 9 + (-11))/4)$
$= 4$

Output Values



Monte Carlo Methods

- Advantage
  - Do not require the MDP dynamics or rewards
- Disadvantage
  - Can only be applied to episodic MDPs
    - Averaging over the returns from a complete episode
    - Requires each episode to terminate

First-Visit Monte Carlo (FVMC) Policy Evaluation

Initialize the value function arbitrarily.

Loop over the episode from the end till the start.

Account for the discounting of the reward

Except if (unless) the state has been visited from time 0 till (t-1) append and average out the results.
I.e., only update the value estimate if this is the first visit to the state.

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated
Initialize:
  $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$
  $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):
  Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:
      Append G to $Returns(S_t)$
      $V(S_t) \leftarrow \text{average}(Returns(S_t))$

Pseudo-code from Sutton and Barto (Reinforcement Learning) Ch 5 (Sec. 5.1)

First-Visit Monte Carlo (FVMC)

- First-Visit Monte Carlo (FVMC)
  - Averages the returns following the first visit to a state s in the episode.
- Every-visit Monte Carlo (EVMC)
  - Averages returns following all the visits to s.
- Convergence
  - FVMC – error falls as 1/N(s). Needs lots of data
  - EVMC - error falls quadratically, slightly better data efficiency.

So, now we know how to evaluate a policy.
What about improving the policy now?

Policy Improvement

Two problems in the last pseudo-code in doing policy improvement.

1. How to obtain the policy?
   - Note: we only stored the state value function. In the absence of a model, we cannot compute the policy.
   - *Solution: Store the state-action values.*
2. How to ensure the coverage of states?
   - *Solution: Use epsilon-greedy policies*
     - Most of the time select an action that has maximal estimated action value.
     - But with probability epsilon, instead, select an action at random.

Policy Improvement (Monte Carlo Control)

Estimate the Q-function.
Derive the policy from there.

All non-greedy actions have a small probability of being selected. The bulk of the likelihood is given to the greedy action.

```
Algorithm parameter: small ε > 0
Initialize:
    π ← an arbitrary ε-soft policy
    Q(s, a) ∈ ℝ (arbitrarily), for all s ∈ 𝒮, a ∈ 𝒜(s)
    Returns(s, a) ← empty list, for all s ∈ 𝒮, a ∈ 𝒜(s)

Repeat forever (for each episode):
    Generate an episode following π: S_0, A_0, R_1, ..., S_{T-1}, A_{T-1}, R_T
    G ← 0
    Loop for each step of episode, t = T-1, T-2, ..., 0:
        G ← γG + R_{t+1}
        Unless the pair S_t, A_t appears in S_0, A_0, S_1, A_1 ..., S_{t-1}, A_{t-1}:
            Append G to Returns(S_t, A_t)
            Q(S_t, A_t) ← average(Returns(S_t, A_t))
            A* ← argmax_a Q(S_t, a)                    (with ties broken arbitrarily)
            For all a ∈ 𝒜(S_t):
                π(a|S_t) ← { 1 - ε + ε/|𝒜(S_t)|   if a = A*
                           { ε/|𝒜(S_t)|           if a ≠ A*
```

In literature, a policy is called *soft* if $\pi(a|s) > 0$ for all $s \in S$ and all $a \in A(s)$.
An $\epsilon - soft$ policy is one for which $\pi(a|s) \geq \frac{\epsilon}{|A(s)|}$ for all states and actions for some ϵ

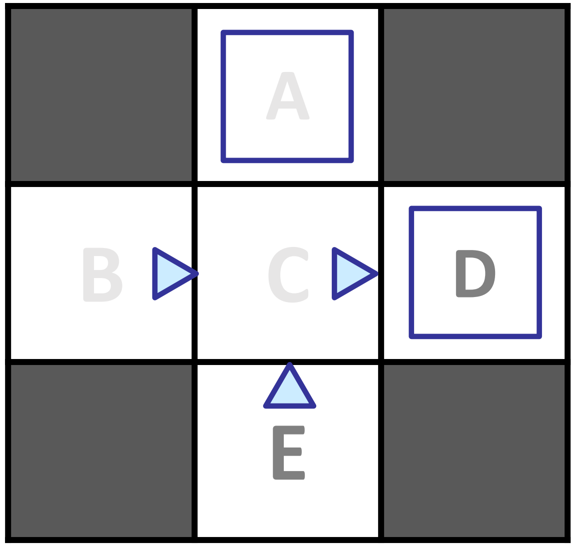
Pseudo-code from Sutton and Barto (Reinforcement Learning) Ch 5 (Sec. 5.4)

Problem with Monte Carlo

- Limitations
  - Each state must be learned separately, loses the state connection information.
    - Estimate of one state is not taking advantage of the estimates of the other states.
    - Note: Bellman equations tell us that value function for states has a recursive relationship.
  - Could only be used in an episodic setting.

Recall our example

Input Policy π



Assume: $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

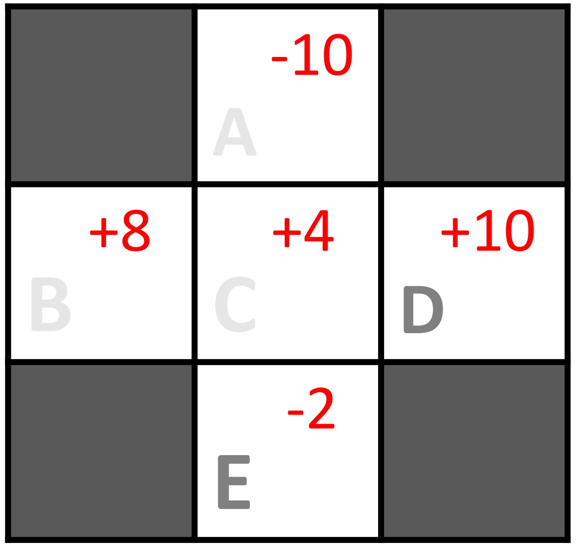
Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Output Values



Problem: we have lost the connection between states.
We go through state C, 4 times.
We use only 2 estimates for E.
But, C and E are adjacent!

Temporal Difference (TD) Learning

- Model-Free combination of
  - Monte Carlo (learning from sample trajectories/experience) and
  - Dynamic programming (via Bellman Equations)
- Incorporate Bootstrapping
  - Update value function estimates of a state based on others
    - Adjust the value function estimate using the Bellman Equation relationship between the value function of successor states.
    - More data-efficient than a Monte Carlo method (discussed previously)
- Setting
  - Can be used in an *episodic* or *infinite-horizon non-episodic* settings
    - Immediately updates the estimate of V(s) after each (s, a, s', r) tuple.

"If one had to identify one idea that is central and novel to reinforcement learning, it would undoubtedly be temporal-difference (TD) learning", Sutton and Barto 2017

TD Learning

- Aim: estimate $V\pi(s)$ given episodes generated under policy π
  - $G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \cdots$ in MDP M under policy π
  - $V^\pi(s) = E_\pi[G_t]$
- Bellman Operator (if we know MDP models)

$$B^\pi V(s) = r(s, \pi(s)) + \gamma \sum_{s'} p(s'|s, \pi(s)) V(s')$$

- In incremental every-visit MC, update estimate using 1 sample of return (for the current i^{th} episode)

$$V^\pi(s) = V^\pi(s) + \alpha(G_{i,t} - V^\pi(s))$$

- Insight: have an estimate of V^π, use to estimate expected return

$$V^\pi(s_t) = V^\pi(s_t) + \alpha\left(\underbrace{[r_t + \gamma V^\pi(s_{t+1})]}_{\text{TD target}} - V^\pi(s_t)\right)$$

- Can immediately update the value estimate after each (s, a, r, s') tuple. Do not need the episodic setting.

TD Methods

- The updates are based on the difference in value functions at each time step, the TD error,

$$\delta_t = r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)$$

  hence the name *temporal difference learning*.
- α is the learning rate.
- TD can be generalised to n-step returns:

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

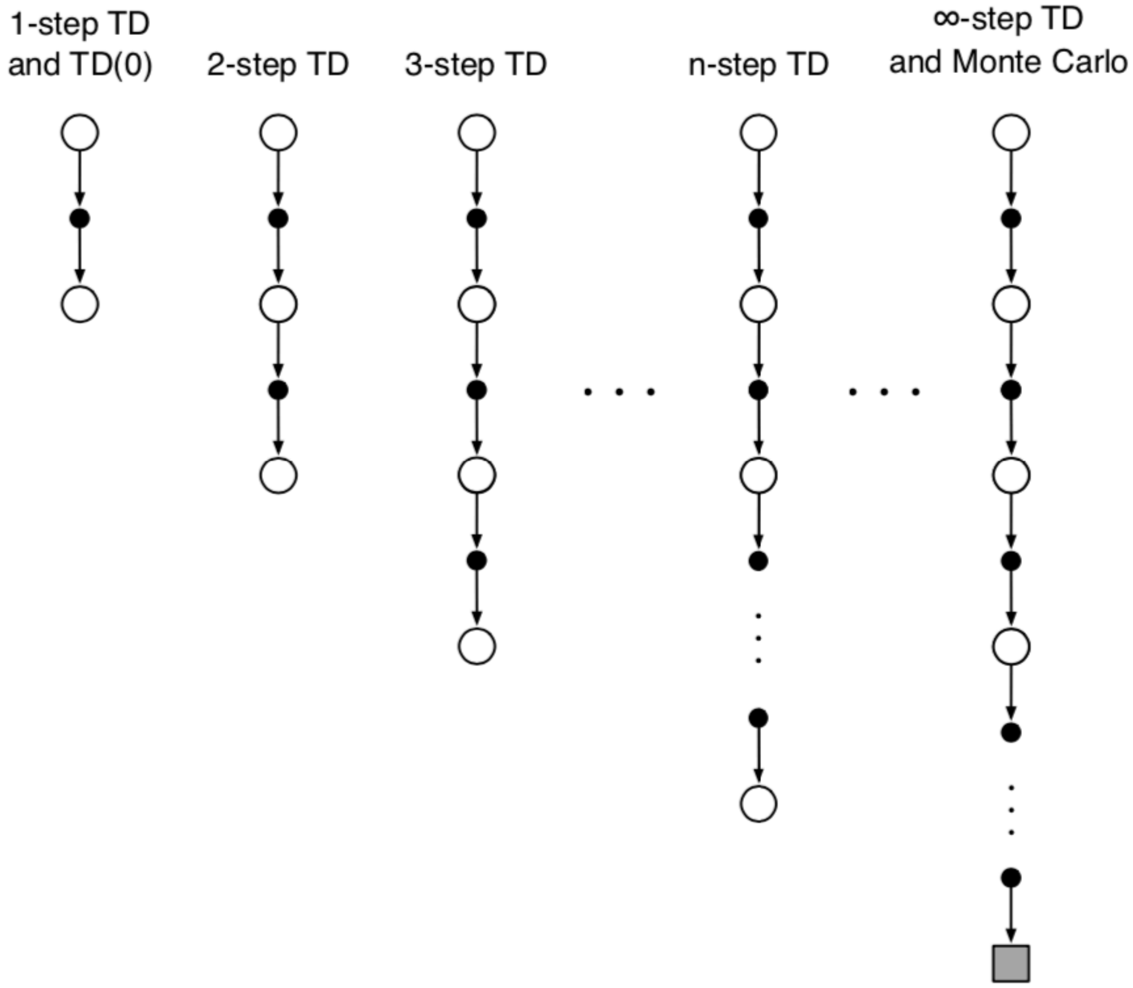
Note: earlier we used the 1-step return in the TD update. Now, we can generalize to how many steps in the future to update from.

$$V^\pi(s_t) = V^\pi(s_t) + \alpha\left(\underbrace{[r_t + \gamma V^\pi(s_{t+1})]}_{\text{TD target}} - V^\pi(s_t)\right)$$

TD Methods

TD can be generalised to n-step returns:

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



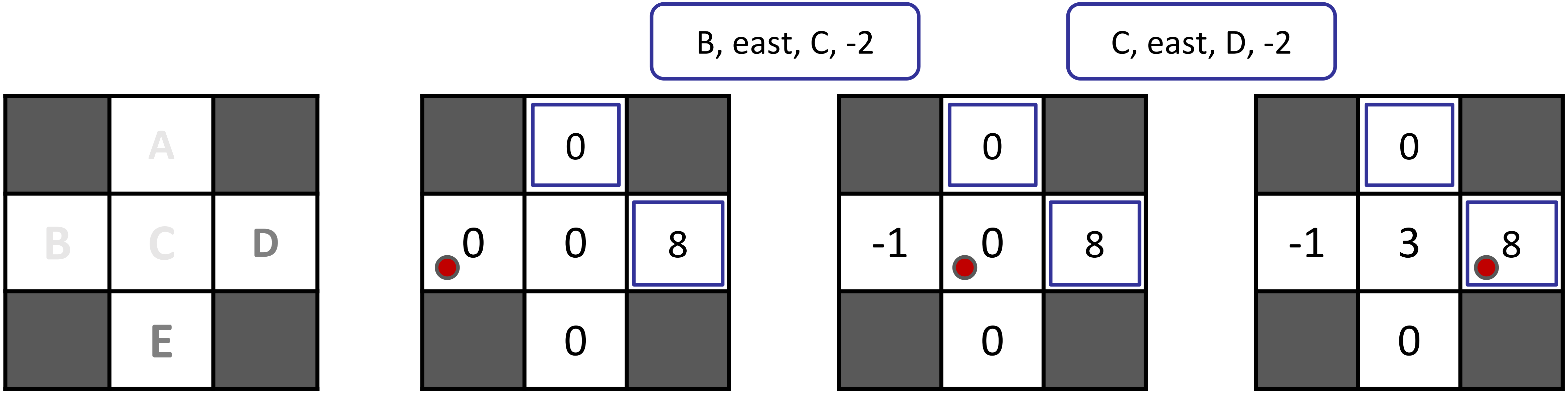
TD Learning (intuitively)

- Nudge our prior estimate of the value function for a state using the given experience.
- Shift the estimate based on the error in what we are experiencing and what estimate we had before
- Weighted by the learning rate.

Temporal Difference Learning: Example

States

Observed Transitions



Assume: $\gamma = 1$, $\alpha = 1/2$

$$V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + \alpha\left[R(s,\pi(s),s') + \gamma V^{\pi}(s')\right]$$

SARSA

- The TD algorithm describes how to evaluate a policy, but does not describe how to improve the policy.
- Can be done very easily with SARSA.
- Rather than learning the value function $V(s)$, we will learn the state-action value $Q(s, a)$, learning from the tuple $\{s_t, a_t, r_t, s_{t+1}, a_{t+1}\}$, hence the name SARSA or "state action reward state action".
- If we generalize the TD rule to state-action functions, such that

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha\left[(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})) - Q(s_t, a_t)\right],$$

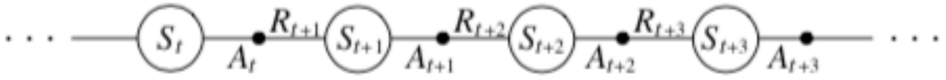
  can then choose a new policy by maximizing over the next action we might take, as in

$$\pi(s) = \max_a Q(s, a)$$

- We modify this policy slightly, using a policy $\pi_\epsilon(s)$ such that

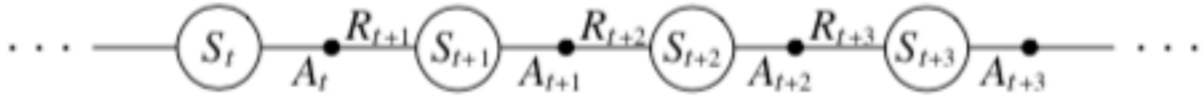
$$\pi_\epsilon(s) = \begin{cases} \max_a Q(s,a) & \text{with probability}(1-\epsilon) \\ \text{RAND}(A) & \text{otherwise} \end{cases}$$

  in order to ensure that we visit all state-action pairs sufficiently often.
- The parameters ϵ can be lowered as $\epsilon_t = 1/t$ so that the agent eventually stops exploring and converges to the optimal policy.



Update using state, action, reward, state, action.

SARSA



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

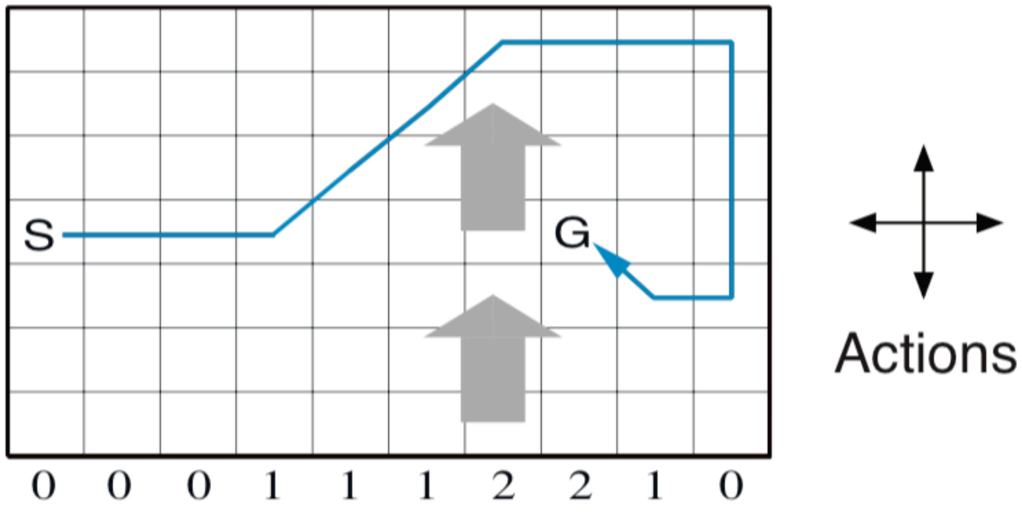
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

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

State-action values updated using the SARSA Learning Rule

Example: Windy Grid World

- Setup
  - Standard grid world with start and goal state.
  - Crosswind running upward through the middle of the grid.
  - Actions: Up, down, left, right
  - Wind strength varies from column to column (written below) in number of grid cells shifted upwards.
  - E.g., if you are one cell to the right of the goal, then the action left takes you to the cell just above the goal.
  - Undiscounted episodic task with constant rewards of -1 till the goal state is reached.
- SARSA with epsilon-greedy does well on this control task.
  - Can learn that the blue path is good.
- Monte Carlo methods cannot easily be used for this task because termination is not guaranteed for all policies.
  - If a policy was found that caused the agent to stay in the same state, then the next episode would never end.
- Step-by-step learning methods such as SARSA do not have this problem because they quickly learn *during the episode.*



Sutton and Barto Ch 6.5 Example 6.5

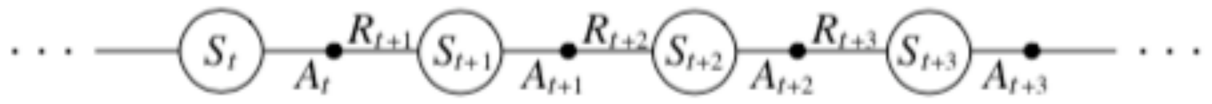
Q-Learning

- One disadvantage to SARSA is that it is an on-policy algorithm
  - That is, we only get an estimate of the Q function for tuples that are directly experienced by executing the policy
  - What if our initial policy is really poor?
  - In that case could take a long time to find the optimal policy.
- The update rule can be modified to improve the *best* state-action tuple, rather than the *experienced* tuple by putting the max operator directly inside the update rule, as in

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left[(r_t + \gamma \max_{a'} Q(s_{t+1}, a')) - Q(s_t, a_t) \right].$$

- Q-LEARNING is one of the core algorithms of reinforcement learning.
- It is an off-policy algorithm, in that the Q function can be shown to converge, regardless of the underlying policy, so long as the underlying policy is guaranteed to visit all state-action pairs infinitely often.

Q-Learning



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

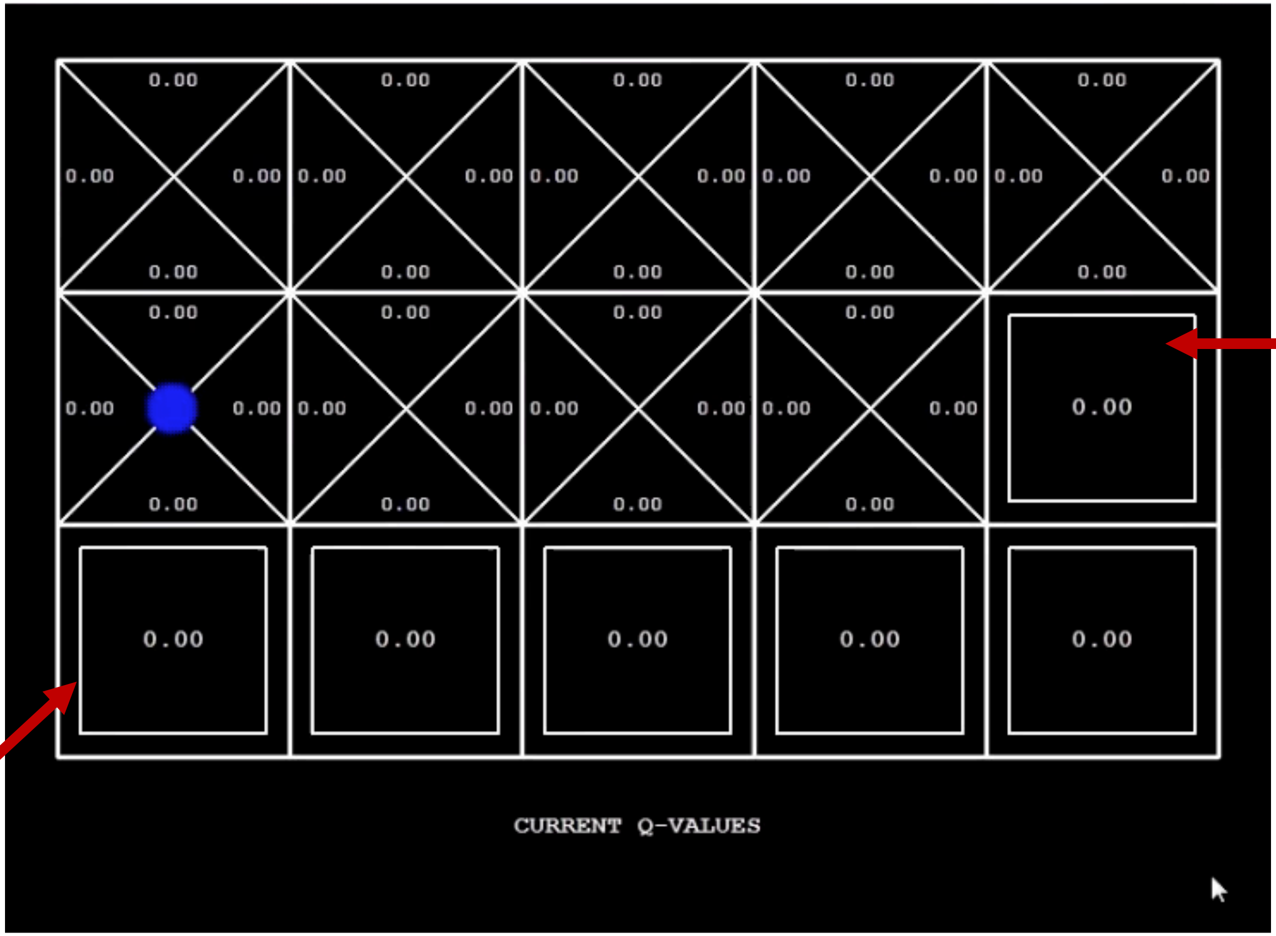
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

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

Q-Learning update rule that uses the max of the next state-action tuples to update.

Example: Cliff Walking

- The agent does not know the rewards a-priori.
  - Learns the effect of the east action over time.
  - Only the actions taken by the agent contribute to updates.
- Occasionally falls in the cliff and gets the negative reward.
- Note that the max of the Q values is propagated (green values) to other states as it is approximating the optimal Q value.



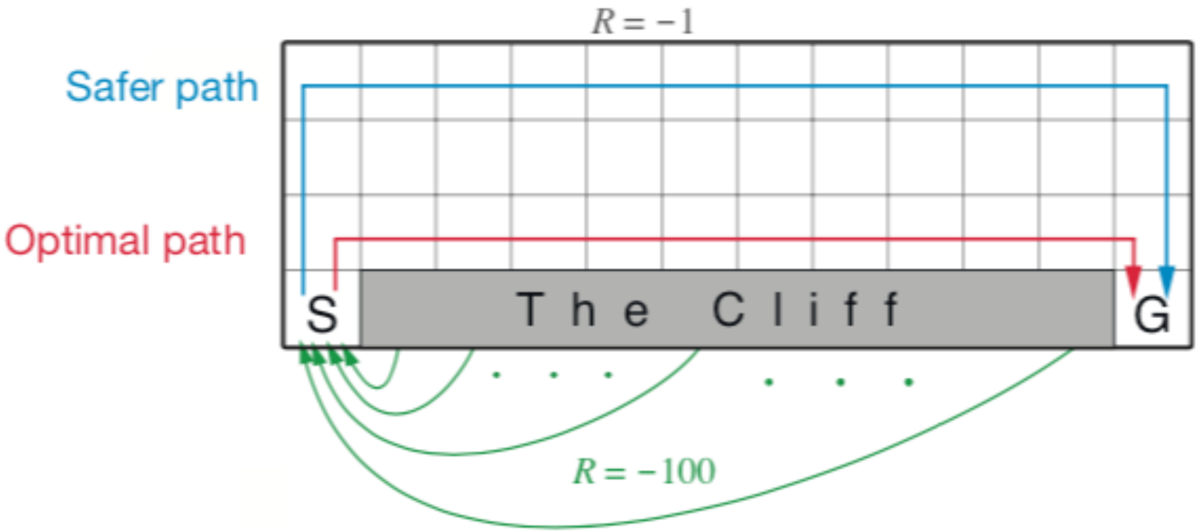
Exit with +10 reward

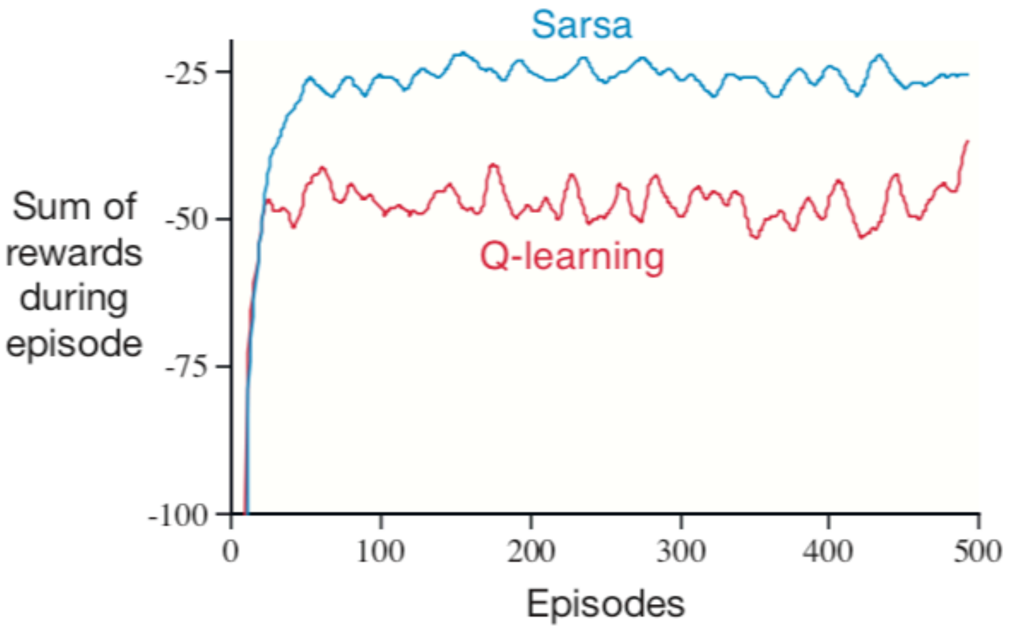
Cliff at the bottom, negative reward here.

The Off-Policy Nature of Q-Learning

- Off-policy methods
  - Learns the optimal state-action value function, independent of the policy being followed.
- Q-learning converges to optimal policy
  - Even if the agent is acting *sub-optimally.*
  - Under conditions: Exploration is enough. The learning rate becomes small enough
- The algorithm learns about the value of the optimal policy without knowing or using the optimal policy.
  - Example: sporadic falls in the cliff does not affect the knowledge that going right leads to high reward.
- Note that SARSA was an on-policy algorithm

Example: Another Cliff Walking





Task: Undiscounted, episodic task with start and goal states.

- Online performance of Q-learning is worse than of SARSA
- SARSA learns the blue path.
- Q-learning learns the red path occasionally falling into the cliff due to epsilon greedy action selection and gets lower rewards.

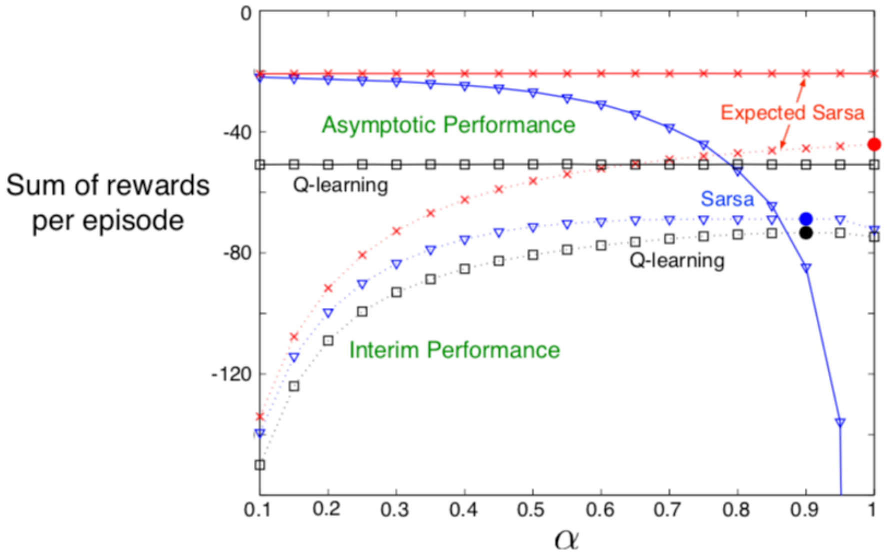
Sutton and Barto Ch 6.5 Example 6.6

Expected SARSA

- Expected SARSA just like Q-learning except takes and expectation instead of the max.
- Considers how likely the action is under the current policy.
- Expected SARSA is more complex computationally but eliminates the variance due to the random selection of actions.

$$
\begin{aligned}
Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \boxed{\alpha\Big[R_{t+1} + \gamma \mathbb{E}_\pi[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t)\Big]} \\
&\leftarrow Q(S_t, A_t) + \boxed{\alpha\Big[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)\Big]},
\end{aligned}
$$

- Expected SARSA performs better than both SARSA and Q-learning on the cliff walking task.



Sutton and Barto Ch 6.5 Example 6.6