COMP 3200
Artificial Intelligence

Lecture 20

On/Off Policy Learning

Temporal-Difference Learning

SARSA, Q-Learning

On-Policy vs. Off-Policy

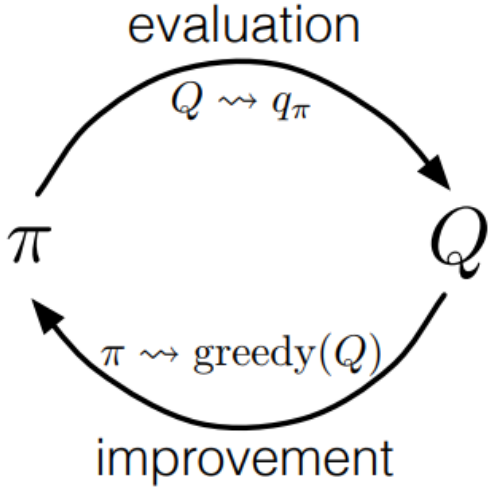
Chapter 5.4 to 5.6 in RL Textbook

http://incompleteideas.net/book/first/ebook/node54.html

On-Policy vs. Off-Policy

- Recall MC ES algorithm presented before
- We use the current policy to update values, then use those values to update the policy
- We are improving the same policy that is used to make decisions / guide learning (On-Policy)
- Off-Policy methods learn a different policy than the one that is used to generate data
  - Example: Use random/human actions to learn a policy

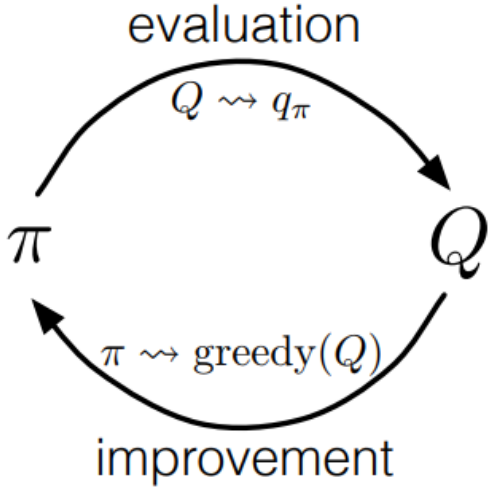
MC Policy Iteration (recall)



1. Function **MCPolicyIteration**()
2. q[s][a] = initial value estimates
3. p[s][a] = initially equiprobable policy
4. **while** true:
5. e = generate episode based on p
6. q = update value estimates based on episode returns
7. p = update policy to choose actions with max values

On-Policy: We generate episodes based on policy we update

Off-Policy Example



1. Function **RandomMCIteration**()
2. q[s][a] = initial value estimates
3. p[s][a] = initially equiprobable policy
4. **while** true:
5. e = generate episode with random actions
6. q = update value estimates based on episode returns
7. p = update policy to choose actions with max values

Off-Policy: Policy we form does not guide episode generation

On-Policy vs. Off-Policy

- On-Policy updates the same policy that is used to generate the actions
- Off-Policy updates a different policy than the one used to generate actions
- On-Policy learns policy that takes exploration into account (accounts for epsilon greedy, etc)
- Off-Policy can have poor online performance, but may result in better performance after learning

On-Policy Control Methods

- In On-P, the policy is generally soft
  - Chance to choose every action is > 0
  - $\pi(a|s) > 0$ for all s, a
- As it learns, the policy moves toward a deterministic optimal policy
- One example of on-policy methods is to use an ε-greedy policy to select actions

ε-Greedy On-Policy Control

- With probability ε choose random action
- With probability 1-ε choose greedy action
- In an ε-Greedy Policy:
  - Non-greedy actions have probability ε/|A(s)|
  - Greedy action has probability 1-ε+ε/|A(s)|
- This is an example of an ε-soft policy
  - $\pi(a|s) > \varepsilon$ for all s, a, and some $\varepsilon > 0$

Monte-Carlo GPI / ε-soft

- On-Policy MC is still GPI
- We use first-visit MC to perform policy evaluation and estimate current policy
- Without ES, if we simply make our policy greedy w.r.t. Q(s,a), we never explore, and many (s,a) pairs go unvisited
- ε-soft on-policy methods ensure that exploration occurs, and all (s,a) get visited
- However, the policy we learn is now only optimal within all ε-soft policies, but in practice it is still quite good

On-Policy Policy Optimality

- On-Policy methods update the policy with the randomness of ε-greedy included
- If the environment has hazardous features with strong negative rewards, ε-greedy will encounter them
- This results in a policy that can be overly cautious and not ‘globally optimal’

On-Policy First-Visit MC (ε-soft)

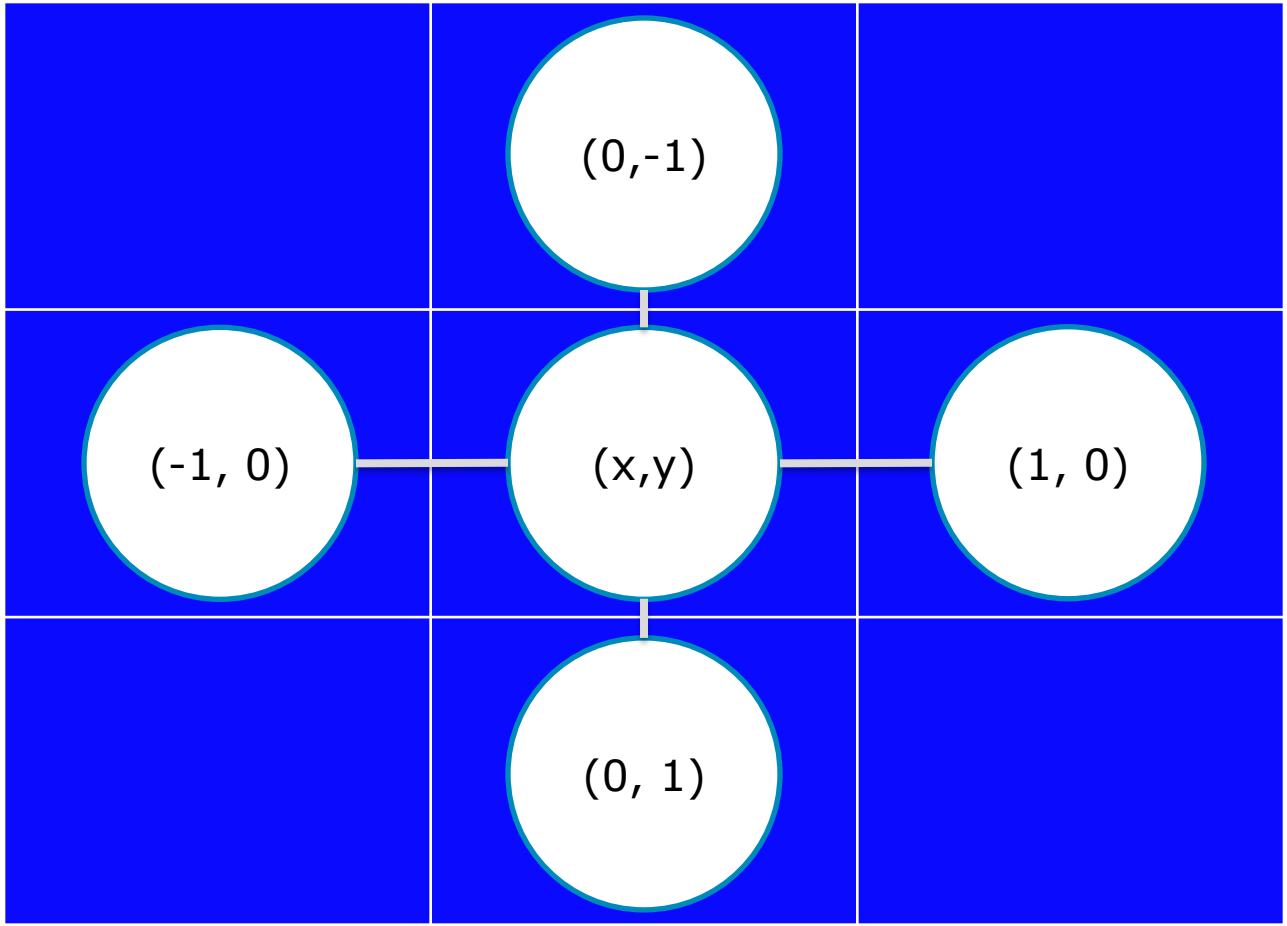
Initialization:

1. Q[s][a] = initial value for each state
2. P[s][a] = initially equiprobable policy

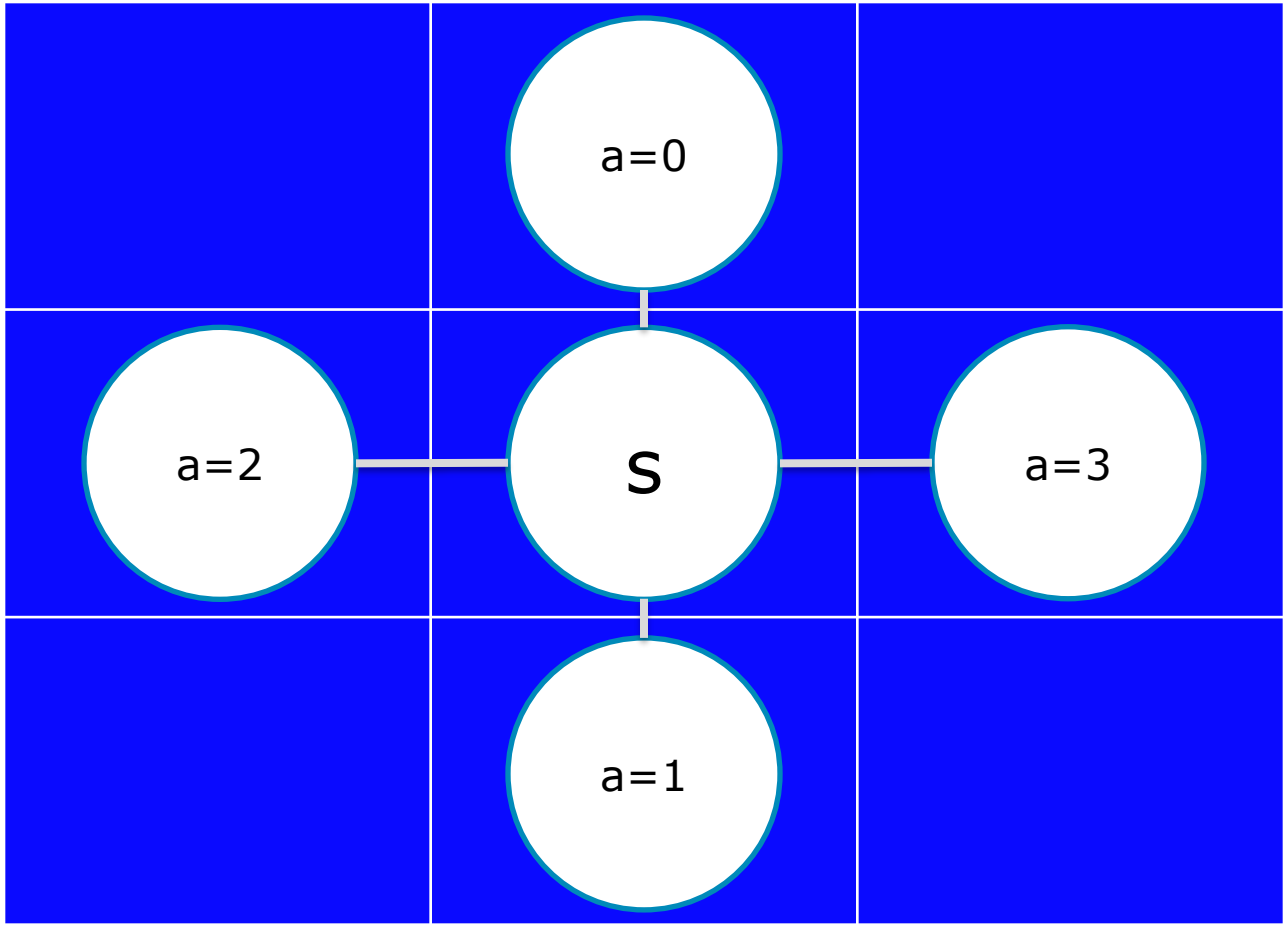
Repeat Forever (Learning):

1. Generate episode using policy P
2. **for** each tuple (s,a,r) in the episode: // value updates
3. G = the return (sum of r) following first occurrence of (s,a) in E
4. Q[s][a] = Q[s][a] + stepSize * (G – Q[s][a]) // any update rule
5. **for** each state s in the episode: // policy updates
6. $A^* = \text{argmax}_a(Q[s][a])$ // tiebreak arbitrary
7. **for** each action a in A(s):
8. **if** (a == A^*) P[s][a] = 1 – ε + (ε / |A(s)|) // greedy action
9. **else** P[s][a] = ε / |A(s)| // non-greedy action

Action [x,y]



Action
N



Example ε-soft calculation

1. **for** each state s in the episode: // policy updates
2. A* = argmax_a(Q[s][a]) // tiebreak arbitrary
3. **for** each action a in A(s):
4. **if** (a == A*) P[s][a] = 1 – ε + (ε / |A(s)|) // greedy action
5. **else** P[s][a] = ε / |A(s)| // non-greedy action

- Example: ε = 0.1, |A(s)| = 5
- Q[s][a] = [10, **20**, 15, 5, 10]
- A* = 1 (index of best action)
- P[s][a] = [0.1/5, 1-0.1+0.1/5, 0.1/05, 0.1/05, 0.1/05]
- P[s][a] = [0.02, 0.92, 0.02, 0.02, 0.02]

Off-Policy Methods

- Learning Control Methods Dilemma:
  - We wish to learn action values based on what we currently believe to be optimal behavior
  - BUT, we need to behave non-optimally in order to explore the (s,a) space and actually FIND the optimal behavior
- On-Policy ε-soft methods compromise by finding a near-optimal policy that still explores
- Another approach is to use two policies
  - Target Policy – Policy being learned over time
  - Behavior Policy – Used to generate episodes / explore
  - Learning from data 'off' the target policy = Off-Policy Learning

Off-Policy Methods

- On-Policy methods are simpler
- Off-Policy methods require additional concepts, higher variance, and converge slower than on-policy methods
- Off-Policy are more powerful in general
- On-Policy methods can be considered a special case of Off-Policy where both policies are same

Temporal Difference Learning

Chapter 6 in RL Textbook

http://incompleteideas.net/book/first/ebook/node60.html

Temporal-Difference Learning

- Central idea of reinforcement learning
- Combines MC and DP ideas
- Like MC: TD methods learn directly from experience without a model of the environment
- Like DP: TD method update incorporates other learned estimates without waiting for final outcome of an episode (bootstrapping)
- Relationship between TD, MC, DP is important

TD Prediction

- Both TD and MC use experience to predict the value of a policy π
- MC methods wait for an episode to finish, then update values based on final return
- TD methods do not wait, and instead update values after each time step

TD Prediction Target

- MC updates use episode return
  $V(S_t) = V(S_t) + \alpha*[G_t - V(S_t)]$
- TD uses next reward + next state value
  $V(S_t) = V(S_t) + \alpha*[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$
- TD updates immediately on transitioning to S_{t+1} and receiving reward R_{t+1}
- TD's target for update is $R_{t+1} + \gamma V(S_{t+1})$
  - This is called TD(0), or one-step TD

Tabular TD(0) for estimating v_π

Initialization (Input Policy P):

1. V[s] = initial value for each state

Repeat Forever (for each episode):

1. S = initialize starting state for episode
2. **Repeat** (for each step of episode)
3. A = action given by P for S
4. S’ = do action A at S
5. R = reward from doing A at S
6. $V(S) = V(S) + \alpha*[R + \gamma V(S') - V(S)]$
7. S = S’

TD Value Estimate

- TD uses next state value in estimate
- Recall from Dynamic Programming

$$v_\pi(s) = E_\pi[G_t] \quad (1)$$
$$= E_\pi[R_{t+1} + \gamma G_{t+1}] \quad (2)$$
$$= E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})] \quad (3)$$

- MC methods use (1) as the target
- DP / TD methods use (3) as the target
- TD combines MC sampling with DP update

Example 6.1: Driving Home Each day as you drive home from work, you try to predict how long it will take to get home. When you leave your office, you note the time, the day of week, the weather, and anything else that might be relevant. Say on this Friday you are leaving at exactly 6 o'clock, and you estimate that it will take 30 minutes to get home. As you reach your car it is 6:05, and you notice it is starting to rain. Traffic is often slower in the rain, so you reestimate that it will take 35 minutes from then, or a total of 40 minutes. Fifteen minutes later you have completed the highway portion of your journey in good time. As you exit onto a secondary road you cut your estimate of total travel time to 35 minutes. Unfortunately, at this point you get stuck behind a slow truck, and the road is too narrow to pass. You end up having to follow the truck until you turn onto the side street where you live at 6:40. Three minutes later you are home. The sequence of states, times, and predictions is thus as follows:

State	*Elapsed Time (minutes)*	*Predicted Time to Go*	*Predicted Total Time*
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43

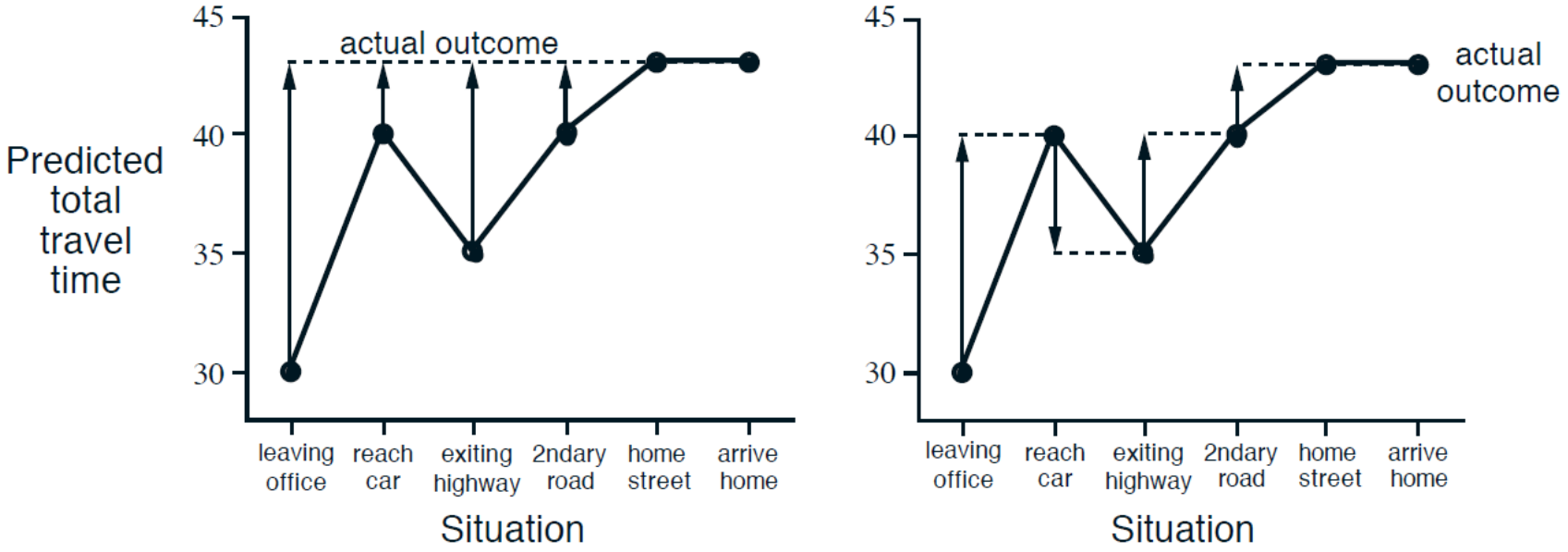


Figure 6.1: Changes recommended in the driving home example by Monte Carlo methods (left) and TD methods (right).

TD Advantages

- DP requires model, MC/TD do not
- MC waits until episode finished to do update, TD is on-line, incremental
  - Sometimes episodes are very long
  - Continuing tasks have no episodes
- TD methods tend to converge faster than fixed-size α MC methods, but not proven

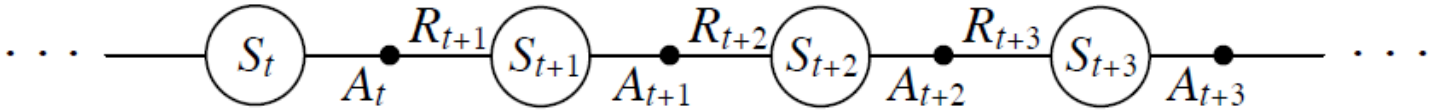
Temporal Difference Control

- We have discussed TD for prediction: Given a policy π, what is $V_{\pi}(s)$?
- How do we now update the policy?
- We will follow GPI
- Like MC, we have the same exploitation vs. exploration trade-off
- Like MC, we have On-Policy and Off-Policy

On-Policy TD Control

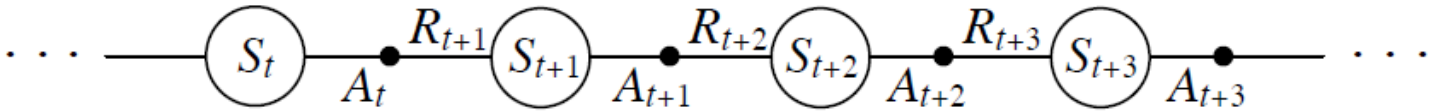
- On-Policy: Need to estimate $Q_\pi(s,a)$ for the current behaviour policy π and for all state-action pairs (s,a)
- Can use the same TD method discussed previously for estimating $v_\pi(s)$
- We can then update the policy π to be greedy w.r.t. $Q_\pi(s,a)$

SARSA: On-Policy TD Control



- Episodes consist of State, Action, Reward
- Previously we learned for TD to transition between episode states and update V(s)
  $V(S_t) = V(S_t) + \alpha*[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$
- SARSA transitions between (s,a) pairs:
  $Q(S_t,A_t) = Q(S_t,A_t) + \alpha*[R_{t+1} + \gamma Q(S_{t+1},A_{t+1}) - Q(S_t,A_t)]$

SARSA: On-Policy TD Control



$$Q(S_t,A_t) = Q(S_t,A_t) + \alpha*[R_{t+1}+ \gamma Q(S_{t+1},A_{t+1}) - Q(S_t,A_t)]$$

- If S_{t+1} is terminal, $Q(S_{t+1},A_{t+1})$ is 0
- Policy update is greedy w.r.t. Q(S,A)
- Formula makes use of the following data: (S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}) = SARSA

SARSA using ε-greedy

Initialization ():

1. Q[s][a] = initial value for each (s,a), Q[terminal] = 0

Repeat Forever (for each episode):

1. S = initialize starting state for episode
2. A = choose action at S using ε-greedy on Q[S][:]
3. **Repeat** (for each step of episode)
4. R, S’ = Take action A at state S
5. A’ = Choose A’ from S’ using ε-greedy on Q[S’][:]
6. Q[S][A] = Q[S][A] + α*(R + γQ[S’][A’] – Q[S][A])
7. S = S’, A = A'

Q-Learning: Off-Policy TD Control

- 1989, early breakthrough in RL
- Off-Policy TD control algorithm

$$Q(S_t,A_t) = Q(S_t,A_t) + \alpha*[R_{t+1}+ \gamma max_a Q (S_{t+1},a) - Q(S_t,A_t)]$$

- The learned Q directly approximates q*, the optimal action-value function, no matter which behaviour policy is being followed

Q-Learning using ε-greedy

Initialization ():

1. Q[s][a] = initial value for each (s,a), Q[terminal] = 0

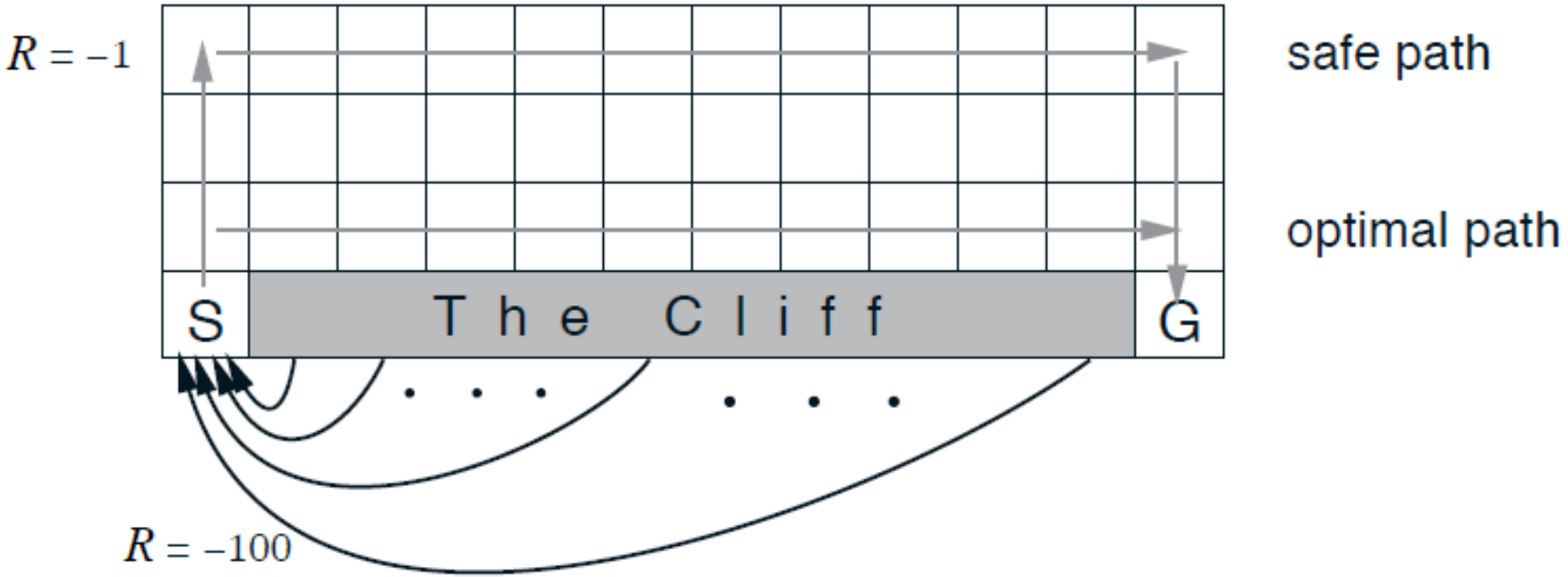
Repeat Forever (for each episode):

1. S = initialize starting state for episode
2. **Repeat** (for each step of episode)
3. A = choose action at S using ε-greedy on Q[S][:]
4. R, S’ = take action A at state S
5. $Q[S][A] = Q[S][A] + \alpha*(R + \gamma \max_a Q[S'][a] - Q[S][A])$
6. S = S’

Q-Learning: Off Policy

- Why is Q-Learning considered off-policy?
- If the algorithm estimates the value function of the policy generating the data, the method is called on-policy
- SARSA learns value of its policy, which does some sort of exploration
- Q-Learning learns value of a policy which does not explore, since it takes the max action value

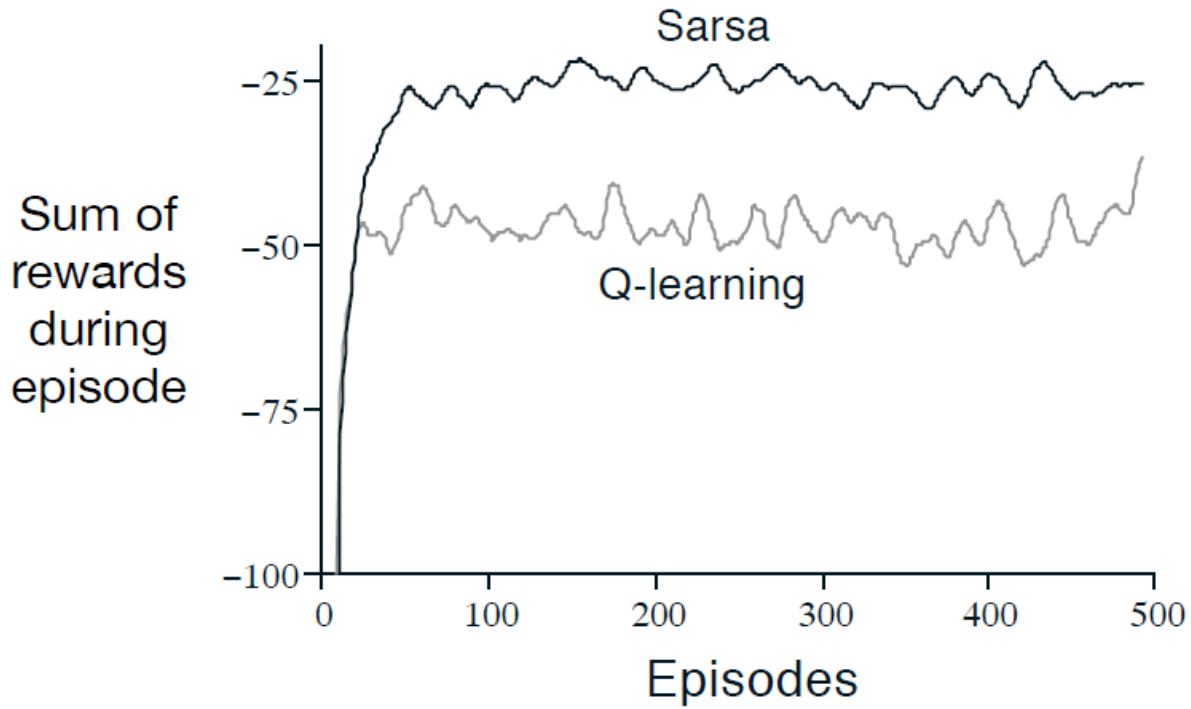
Example: The Cliff



The Cliff

- Actions: Up, Down, Left, Right
- -1 reward at each time step
- If you step over ‘The Cliff’ you get a reward of -100 and go back to start
- Designed to show the difference between Q-Learning and SARSA methods on-line performance

Cliff: SARSA vs. Q-Learning



Epsilon Greedy
E = 0.1

Cliff: SARSA vs. Q-Learning

- Q-Learning learns values for the optimal policy, which travels at the edge of the cliff
- This results in it sometimes falling off the cliff in training because of e-greedy action selection
- SARSA takes e-greedy action selection into account and learns the longer but 'safer' path
  - On-Policy: Learns values of the policy that guides it
- Even though Q-Learning learns the values of the optimal policy, its online performance is worse

State

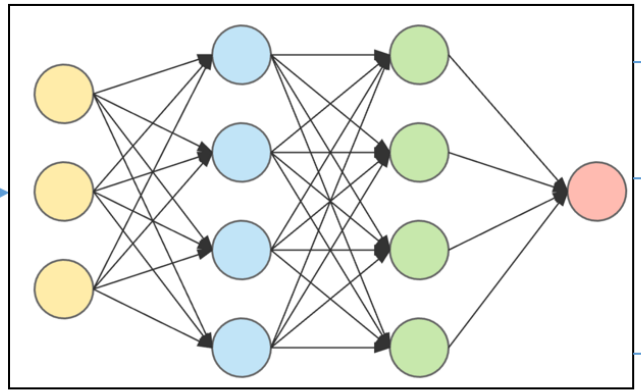
Action

Q Table	
State-Action	Value
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0

Q-Value

Q Learning

State



Q-Value Action 1

Q-Value Action 2

-

-

Q-Value Action N

Deep Q Learning

Exam Questions

- What does it mean to be an on-policy or an off-policy RL method?
- Why is SARSA considered on-policy while Q-Learning is considered off-policy
- Why do SARSA and Q-Learning produce different paths through the example of The Cliff?
- TD methods borrow ideas from both MC and DP, what are they and why?
- SARSA + Q-Learning algorithms