Reinforcement Learning

Temporal-Difference Learning

U Kang
Seoul National University

In This Lecture

- Overview of TD Learning
- TD Prediction Methods
- TD Control Methods

Overview

- Temporal-Difference (TD) Learning
  - A central and novel idea to RL
  - A combination of MC and DP
  - Like MC methods, TD methods can learn directly from raw experience without a model of the environment
  - Like DP, TD methods update estimates based in part on other learned estimates, without waiting for a final outcome (they bootstrap)
  - The relationship between TD, DP, and Monte Carlo methods is a recurring theme in the theory of RL
    - Related topic: n-step TD and TD(λ)

Outline

TD Prediction

- Both TD and MC methods use experience to solve the prediction problem
- Given some experience following a policy π, both methods update their estimate V of v_π for the nonterminal states S_t occurring in that experience
- MC methods wait until the return following the visit is known, then use that return as a target for $V(S_t)$
- Constant-α MC: a simple every-visit MC method suitable for nonstationary environments

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

TD Prediction

- Whereas MC methods must wait until the end of the episode to determine the increment to $V(S_t)$ (only then is G_t known), TD methods need to wait only until the next time step
- At time t + 1 TD methods immediately form a target and make a useful update using the observed reward R_{t+1} and the estimate $V(S_{t+1})$
- The simplest TD method makes the update immediately on transition to S_{t+1} and receiving R_{t+1}

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

- The target for the MC update is G_t, whereas the target for the TD update is $R_{t+1} + \gamma V(S_{t+1})$
- This method is called TD(0), or one-step TD; this is a special case of n-step TD and TD(λ)

TD Prediction

Tabular TD(0) for estimating v_π

```
Input: the policy $\pi$ to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        $A \leftarrow$ action given by $\pi$ for $S$
        Take action $A$, observe $R$, $S'$
        $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
        $S \leftarrow S'$
    until $S$ is terminal
```

Sutton and Barto, Reinforcement Learning, 2018

TD Prediction

- TD(0) is a bootstrapping method (bases its update in part on an existing estimate)
- Value function

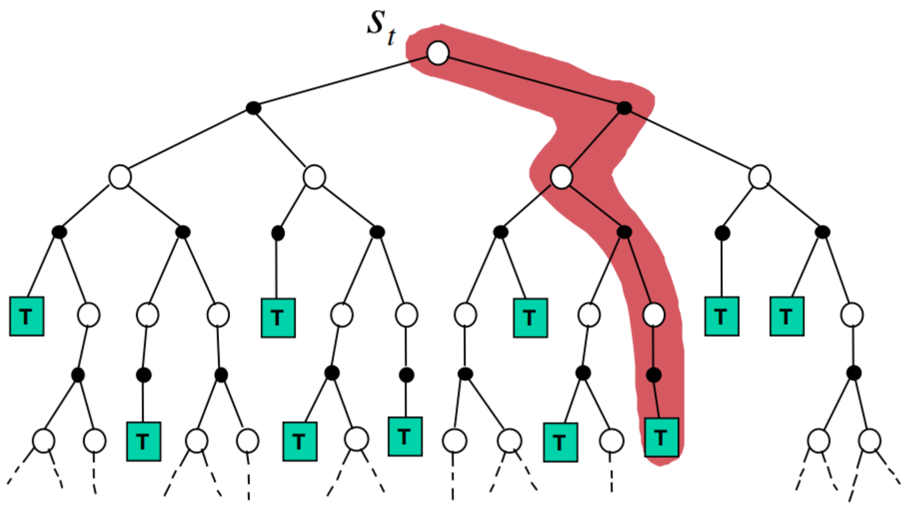
$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t | S_t = s] & \text{A} \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] & \text{B} \end{aligned}$$

- MC methods uses an estimate of A as a target, whereas TD methods use an estimate of B as a target
- The MC target is an estimate because the expected value in A is not known; a sample return is used in place of the real expected return
- The TD target is an estimate for two reasons: 1) it samples the expected values in B and 2) it uses the current estimate $V(S_{t+1})$ instead of the true $v_\pi(S_{t+1})$
- TD methods combine the sampling of MC with the bootstrapping of DP

TD vs MC

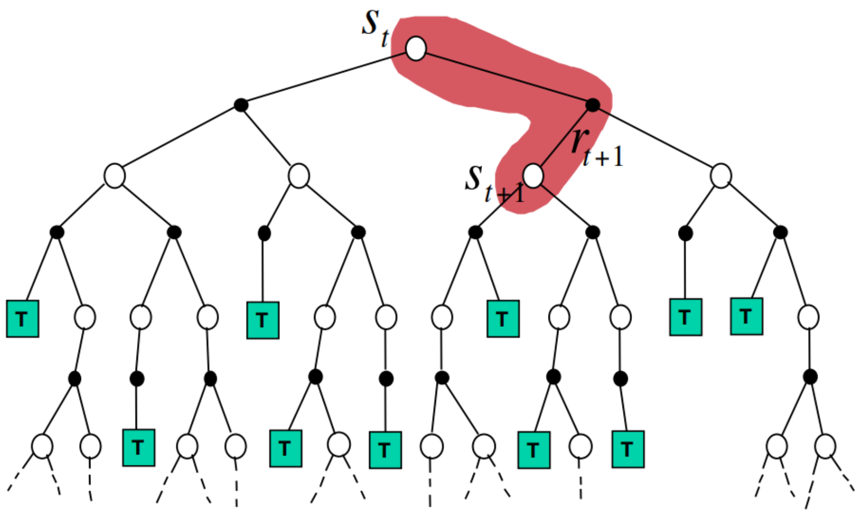
MC

$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t - V(S_t) \right)$$



TD

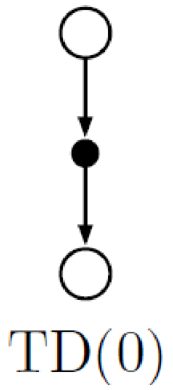
$$V(S_t) \leftarrow V(S_t) + \alpha \left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right)$$



https://www.davidsilver.uk/wp-content/uploads/2020/03/MC-TD.pdf

TD Prediction

- The value estimate for the state node at the top of the backup diagram is updated on the basis of the one sample transition from it to the immediately following state
- TD and MC updates are sample updates because they involve looking ahead to a sample successor state (or state–action pair), using the value of the successor and the reward along the way to compute a backed-up value, and then updating the value of the original state (or state–action pair) accordingly
- Sample updates differ from the expected updates of DP methods in that they are based on a single sample successor rather than on a complete distribution of all possible successors



Backup diagram for tabular TD(0)

Sutton and Barto, Reinforcement Learning, 2018

TD Prediction

- TD(0): $V(S_t) \leftarrow V(S_t) + \alpha\underbrace{[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]}_{\text{TD error } \delta_t}$

- Notice that the TD error at each time is the error in the estimate *made at that time*
- Because the TD error depends on the next state and next reward, it is not actually available until one time step later; δ_t is the error in $V(S_t)$, available at time t + 1

TD Prediction

- TD(0): $V(S_t) \leftarrow V(S_t) + \alpha[\underbrace{R_{t+1} + \gamma V(S_{t+1}) - V(S_t)}_{\text{TD error } \delta_t}]$

- If the array V does not change during the episode (as it does not in MC methods), then the MC error can be written as a sum of TD errors

$$
\begin{aligned}
G_t - V(S_t) &= R_{t+1} + \gamma G_{t+1} - V(S_t) + \gamma V(S_{t+1}) - \gamma V(S_{t+1}) \\
&= \delta_t + \gamma\big(G_{t+1} - V(S_{t+1})\big) \\
&= \delta_t + \gamma\delta_{t+1} + \gamma^2\big(G_{t+2} - V(S_{t+2})\big) \\
&= \delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2} + \cdots + \gamma^{T-t-1}\delta_{T-1} + \gamma^{T-t}\big(G_T - V(S_T)\big) \\
&= \delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2} + \cdots + \gamma^{T-t-1}\delta_{T-1} + \gamma^{T-t}\big(0 - 0\big) \\
&= \sum_{k=t}^{T-1} \gamma^{k-t}\delta_k.
\end{aligned}
$$

- This identity is not exact if V is updated during the episode (as it is in TD(0)), but if the step size is small then it may still hold approximately

Example: 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

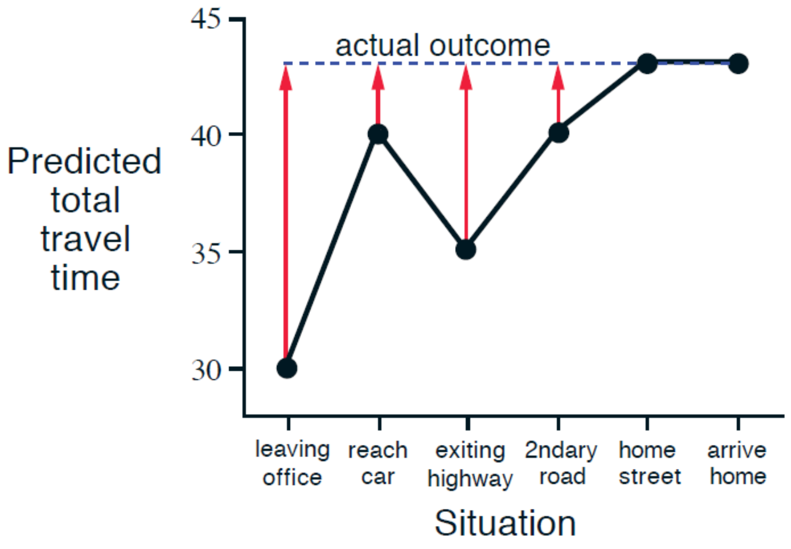
State	*Elapsed Time (minutes)*	*Predicted Time to Go*	*Predicted Total Time*
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43

Sutton and Barto, Reinforcement Learning, 2018

- Rewards: elapsed times on each leg of the journey
- Return for each state: actual time to go from that state (no discounting)
- Value of each state: expected time to go ('Predicted Time to Go' column gives the current estimated value for each state)

Example: Driving Home

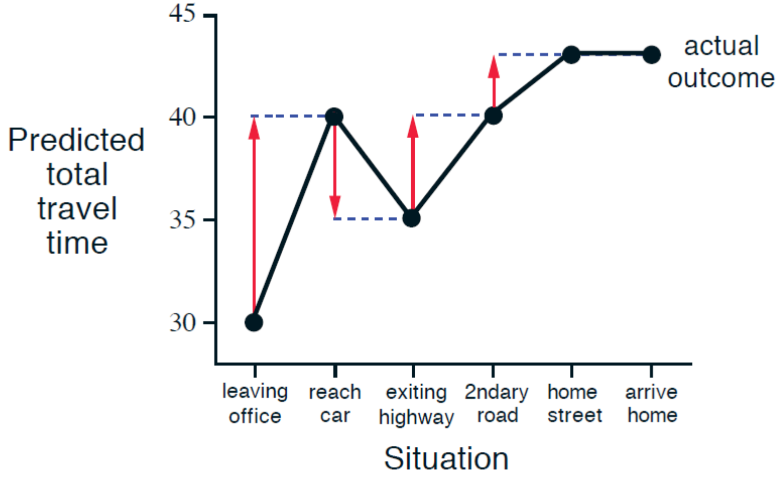
- MC methods
  - The red arrows show the changes in predictions recommended by the constant-α MC method where $\alpha = 1$
  - These are exactly the errors between the estimated value (predicted time to go) in each state and the actual return (actual time to go)
  - When you exited the highway you thought it would take only 15 minutes more to get home, but in fact it took 23 minutes; The error, Gt − V (St), at this time is eight minutes. If $\alpha = 0.5$, then the predicted time to go after exiting the highway would be revised upward by four minutes
  - This is probably too large a change in this case; the truck was probably just an unlucky break
  - In any event, the change can only be made offline, that is, after you have reached home



Sutton and Barto, Reinforcement Learning, 2018

Example: Driving Home

- TD methods
  - Is it necessary to wait until the final outcome is known before learning can begin?
  - Suppose on another day you estimate when leaving your office at 6:00 that it will take 30 minutes to drive home, but become stuck in a massive traffic jam in highway at 6:25; you now estimate that it will take another 25 minutes to get home, for a total of 50 minutes
  - TD allows to shift your initial estimate from 30 minutes toward 50
  - Each estimate would be shifted toward the estimate that immediately follows it; each error is proportional to the change over time of the prediction, that is, to the temporal differences in predictions



Sutton and Barto, Reinforcement Learning, 2018

Outline

- ☑ TD Prediction
- ➡ ☐ **Advantages of TD Prediction Methods**
- ☐ Optimality of TD(0)
- ☐ Sarsa: On-policy TD Control
- ☐ Q-learning: Off-policy TD Control
- ☐ Expected Sarsa
- ☐ Maximization Bias and Double Learning
- ☐ Games, Afterstates, and Other Special Cases
- ☐ Conclusion

Advantages of TD Prediction

- TD methods update their estimates based in part on other estimates; they learn a guess from a guess—they bootstrap

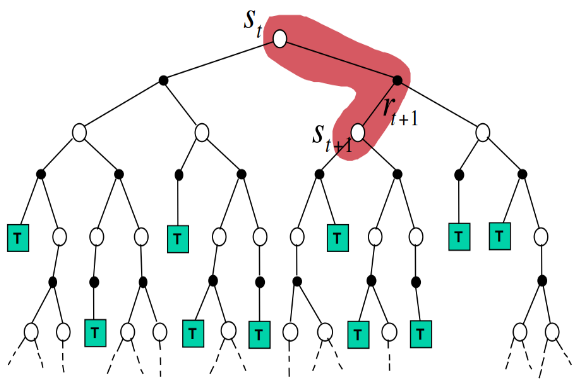
- Is this a good thing to do?

- What advantages do TD methods have over MC and DP methods?

TD vs MC vs DP

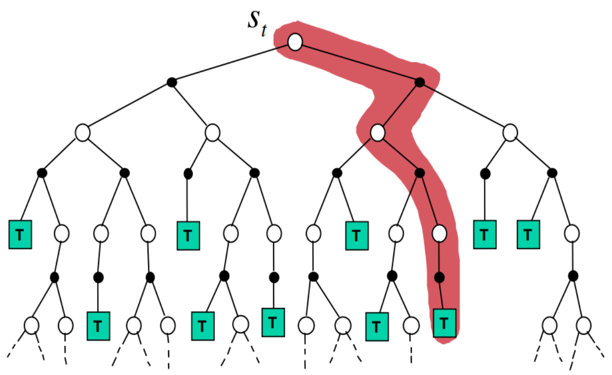
TD

$V(S_t) \leftarrow V(S_t) + \alpha\left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right)$



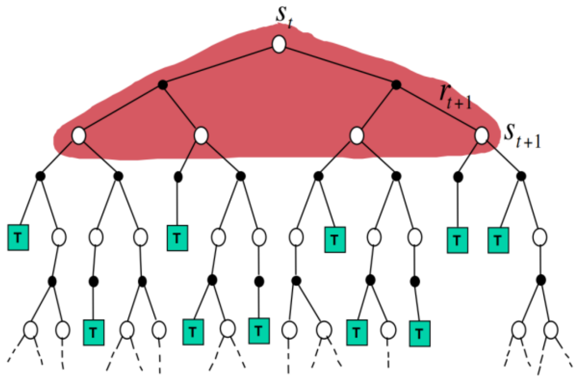
MC

$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t - V(S_t)\right)$



DP

$V(S_t) \leftarrow \mathbb{E}_\pi\left[R_{t+1} + \gamma V(S_{t+1})\right]$



https://www.davidsilver.uk/wp-content/uploads/2020/03/MC-TD.pdf

Advantages of TD Prediction

- Advantage 1 (TD vs. DP)
  - TD methods do not require a model of the environment, of its reward and next-state probability distributions (unlike DP)

Advantages of TD Prediction

- Advantage 2 (TD vs. MC)
  - TD methods are naturally implemented in an online, fully incremental fashion (unlike MC)
  - MC: one must wait until the end of an episode
  - TD: one need to wait only one time step
  - This is often critical - some applications have very long episodes, so that delaying all learning until the end of the episode is too slow; other applications are continuing tasks and have no episodes at all
- Advantage 3 (TD vs. MC)
  - Some MC methods (off-policy) must ignore or discount episodes on which experimental actions are taken, which can greatly slow learning
  - TD methods are much less susceptible to these problems because they learn from each transition regardless of what subsequent actions are taken

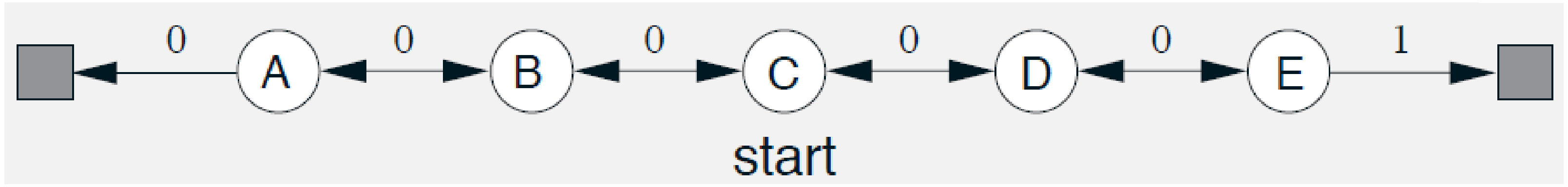
Advantages of TD Prediction

- But are TD methods sound? Can we still guarantee convergence to the correct answer?
- Yes! For any fixed policy π, TD(0) has been proved to converge to v_π on average for a constant step-size parameter if it is sufficiently small, and with probability 1 if the step-size parameter decreases according to the usual stochastic approximation conditions
- If both TD and MC methods converge asymptotically to the correct predictions, which method learns faster? Which makes the more efficient use of limited data?
- This is an open question in the sense that no one has been able to prove mathematically that one method converges faster than the other
- In practice, however, TD methods have usually been found to converge faster than constant-α MC methods on stochastic tasks

Example: Random Walk

- Comparison of the predictions of TD(0) and constant-α MC when applied to the following Markov Reward Process (MRP)

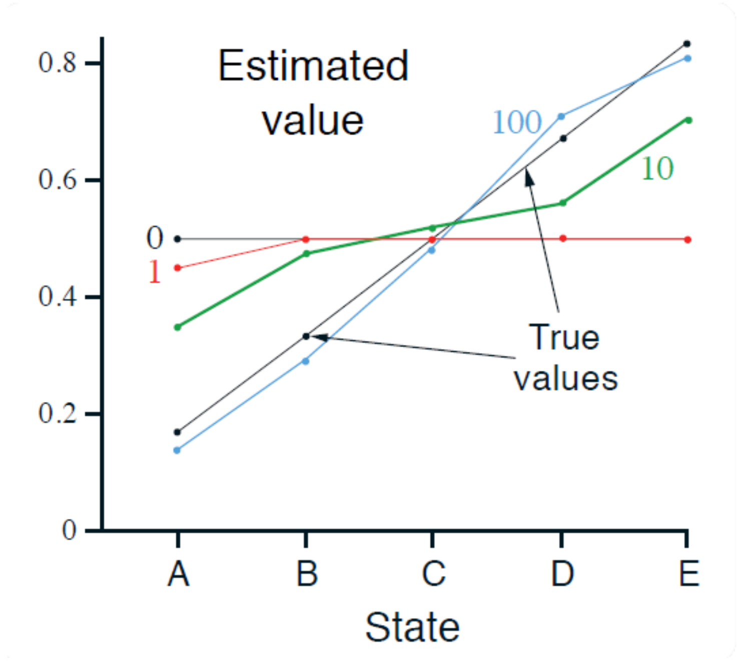
Sutton and Barto, Reinforcement Learning, 2018

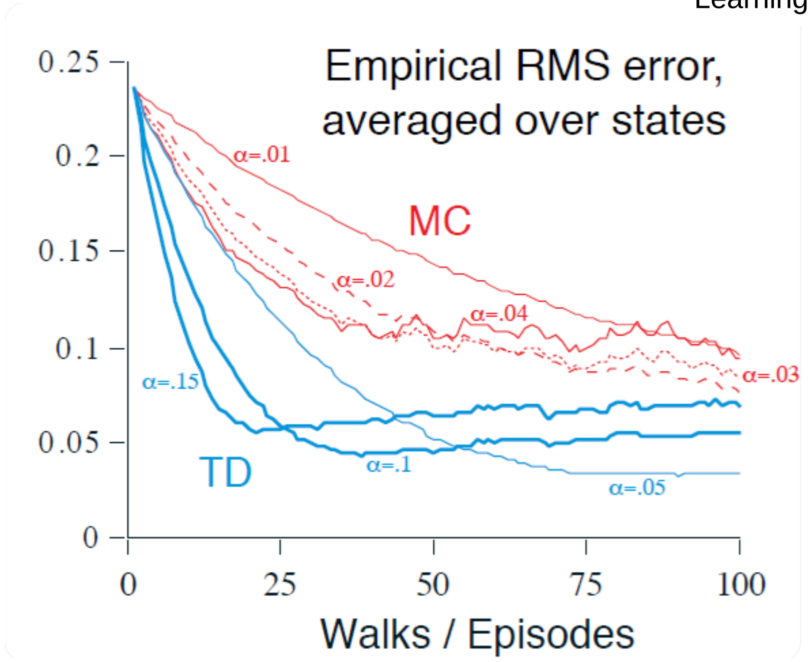


- MRP is an MDP w.o. actions
- All episodes start in the center state C, then proceed either left or right by one state on each step, with equal probability
- Episodes terminate either on the extreme left or the extreme right
- When an episode terminates on the right, a reward of +1 occurs; all other rewards are zero (rewards are undiscounted)
- E.g., an episode: C, 0,B, 0, C, 0,D, 0, E, 1.
- The true value of each state is the probability of terminating on the right if starting from that state; $v_\pi(A,B,C,D,E) = \frac{1}{6}, \frac{2}{6}, \frac{3}{6}, \frac{4}{6}, \frac{5}{6}$

Example: Random Walk

Sutton and Barto, Reinforcement Learning, 2018

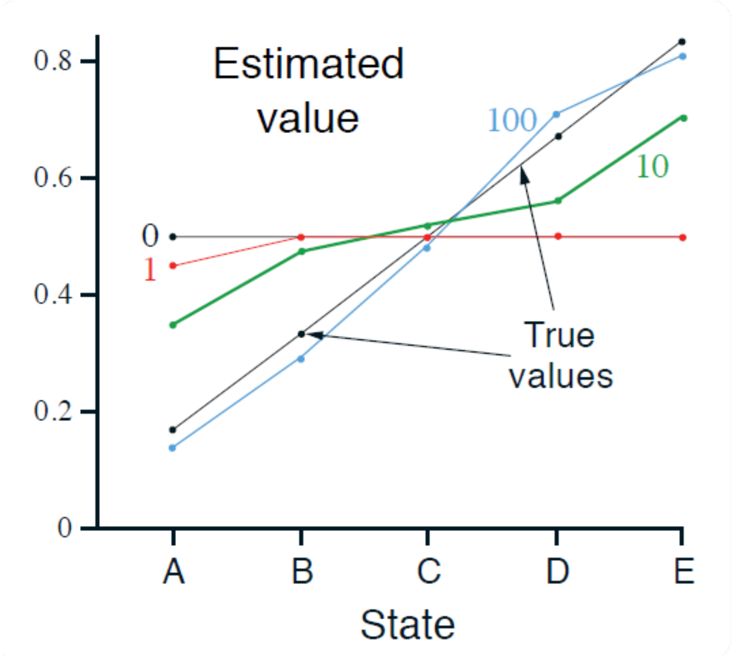


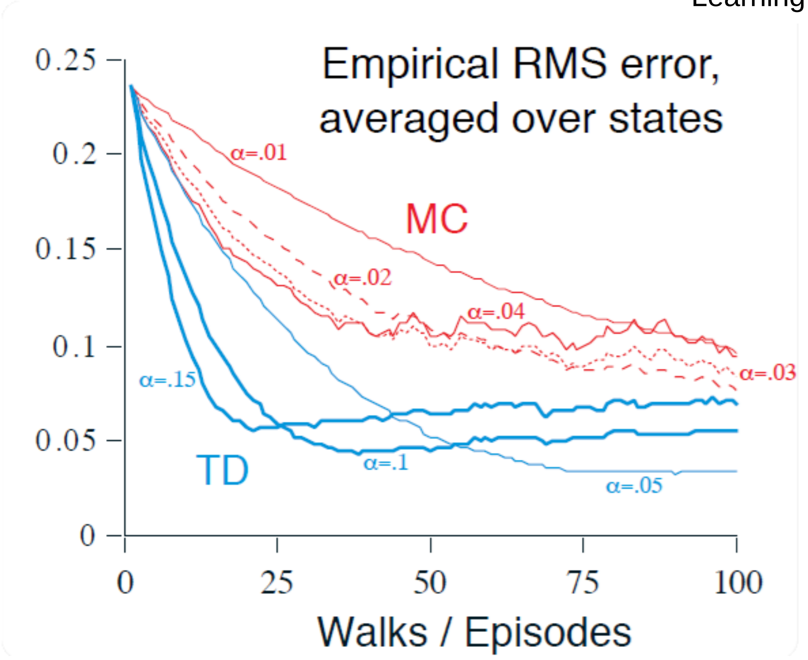


- The left graph above shows the values learned after various numbers of episodes on a single run of TD(0)
- The estimates after 100 episodes are about as close as they ever come to the true values—with a constant step-size parameter (α = 0.1), the values fluctuate indefinitely in response to the outcomes of the most recent episodes

Example: Random Walk

Sutton and Barto, Reinforcement Learning, 2018





- The right graph shows learning curves for the two methods
- The performance measure is the root mean-squared (RMS) error between the value function learned and the true value function, averaged over the five states, then averaged over 100 runs
- The approximate value function was initialized to the intermediate value V (s) = 0.5, for all s; TD method is consistently better than the MC method on this task

Outline

- ✓ TD Prediction
- ✓ Advantages of TD Prediction Methods
- ➡ ☐ **Optimality of TD(0)**
- ☐ Sarsa: On-policy TD Control
- ☐ Q-learning: Off-policy TD Control
- ☐ Expected Sarsa
- ☐ Maximization Bias and Double Learning
- ☐ Games, Afterstates, and Other Special Cases
- ☐ Conclusion

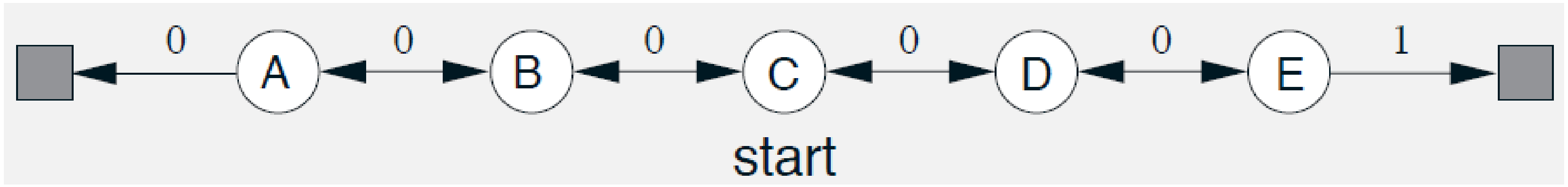
Optimality of TD(0)

- Suppose there is available only a finite amount of experience, say 10 episodes or 100 time steps
- Batch Update
  - In this case, a common approach with incremental learning methods is to present the experience repeatedly until the method converges upon an answer $\boxed{V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]}$
  - Given an approximate value function, V , the increments are computed for every time step t at which a nonterminal state is visited
  - Updates are made only after processing each complete batch of training data
  - Then all the available experience is processed again with the new value function to produce a new overall increment, and so on, until the value function converges

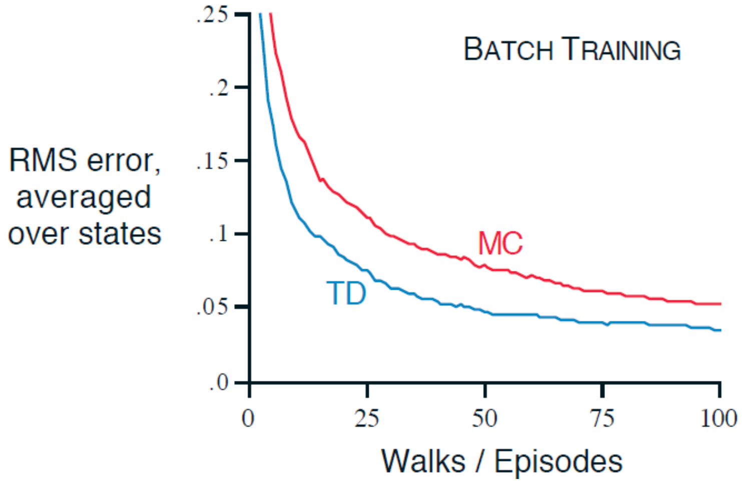
Optimality of TD(0)

- Batch Update
  - Under batch updating, TD(0) converges deterministically to a single answer independent of the step-size parameter α, as long as α is chosen to be sufficiently small
  - Constant-α MC method also converges deterministically under the same conditions, but to a different answer
  - Under normal updating the methods do not move all the way to their respective batch answers, but in some sense they take steps in these directions

Example: Random Walk Under Batch Updating

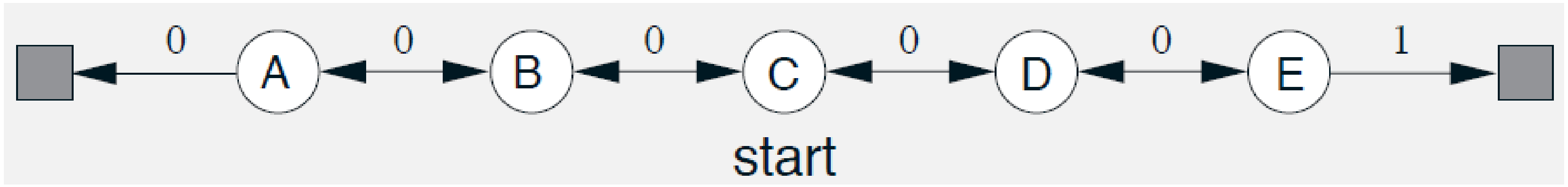


- Batch-updating versions of TD(0) and constant-α MC for the random walk prediction
- After each new episode, all episodes seen so far were treated as a batch
- They were repeatedly presented to either TD(0) or constant-α MC, with α sufficiently small that the value function converged
- The resulting value function was then compared with v_π, and the average RMSE across the five states (and across 100 independent repetitions of the whole experiment) was plotted
- The batch TD is consistently better than the batch MC

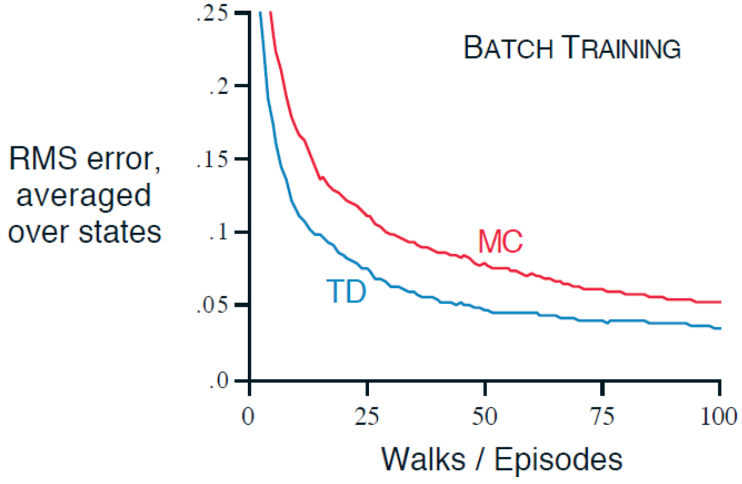


Sutton and Barto, Reinforcement Learning, 2018

Example: Random Walk Under Batch Updating



- Under batch training, constant-α MC converges to values, V (s), that are sample averages of the actual returns experienced after visiting each state s
- These are optimal estimates in the sense that they minimize RMSE from the actual returns in the training set
- Surprisingly, the batch TD method was able to perform better according to RMSE
- Why was batch TD able to perform better than this optimal method?
- MC method is optimal only in a limited way, and that TD is optimal in a way that is more relevant to predicting returns



Sutton and Barto, Reinforcement Learning, 2018

Example: You are the Predictor

- Assume you are predicting returns for an unknown MRP
- Suppose you observe the following eight episodes

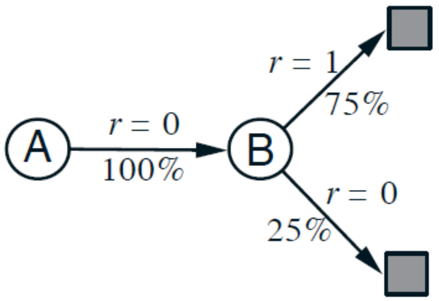
  A, 0, B, 0
  B, 1
  B, 1
  B, 1
  B, 1
  B, 1
  B, 1
  B, 0

- The first episode started in state A, transitioned to B with a reward of 0, and then terminated from B with a reward of 0
- The other seven episodes started from B and terminated immediately
- What would be the best estimates of V(A) and V(B)?
- Everyone would probably agree that the optimal value for V (B) is ¾, by averaging rewards (6 out of 8 cases the rewards were 1)

Example: You are the Predictor

A, 0, B, 0	B, 1
B, 1	B, 1
B, 1	B, 1
B, 1	B, 0

- What is the optimal estimate of V(A)? Two possible answers
- Answer 1 (by TD(0))
  - Observe that 100% of the times the process was in state A it traversed immediately to B (with a reward of 0); and because we have already decided that B has value ¾, A must have value ¾ as well
  - One way of viewing this answer is that it is based on first modeling the Markov process, and then computing the correct estimates given the model, which indeed in this case gives V(A) = ¾
  - This is the answer given by TD(0)



Sutton and Barto, Reinforcement Learning, 2018

Example: You are the Predictor

$A, 0, B, 0$
$B, 1$
$B, 1$
$B, 1$

$B, 1$
$B, 1$
$B, 1$
$B, 0$

- What is the optimal estimate of V(A)? Two possible answers
- Answer 2 (by MC)
  - The other reasonable answer is simply to observe that we have seen A once and the return that followed it was 0; we therefore estimate V (A) as 0
  - This is the answer that batch MC methods give
  - It is also the answer that gives minimum RMSE on the training data (it gives zero error on the data)
  - But still we expect the answer 1 to be better; if the process is Markov, we expect that the answer 1 will produce lower error on future (test) data, even though the MC answer is better on the existing (training) data

Optimality of TD(0)

- Batch TD(0) vs. batch MC
  - Batch MC always finds the estimates that minimize mean-squared error on the training set, whereas batch TD(0) always finds the estimates that would be exactly correct for the maximum-likelihood model of the Markov process
  - The maximum-likelihood estimate (MLE) of a parameter is the parameter value whose probability of generating the data is greatest
  - In this case, the MLE is the model of the Markov process formed from the observed episodes: the estimated transition probability from i to j is the fraction of observed transitions from i that went to j, and the associated expected reward is the average of the rewards observed on those transitions
  - Given this model, we can compute the estimate of the value function that would be exactly correct if the model were exactly correct
  - This is called the certainty-equivalence estimate because it is equivalent to assuming that the estimate of the underlying process was known with certainty rather than being approximated
  - In general, batch TD(0) converges to the certainty-equivalence estimate

Optimality of TD(0)

- Batch TD(0) vs. batch MC
  - Why TD methods converge more quickly than MC methods?
  - In batch form, TD(0) is faster than MC methods because it computes the true certainty-equivalence estimate
  - Although the nonbatch methods do not achieve either the certainty-equivalence or the minimum squared-error estimates, they can be understood as moving roughly in these directions
  - Nonbatch TD(0) may be faster than constant-α MC because it is continuously moving toward a better estimate

Outline

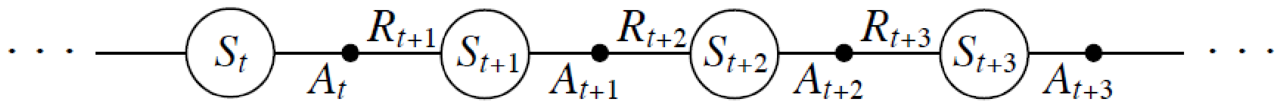
- ✓ TD Prediction
- ✓ Advantages of TD Prediction Methods
- ✓ Optimality of TD(0)
- ☐ **Sarsa: On-policy TD Control**
- ☐ Q-learning: Off-policy TD Control
- ☐ Expected Sarsa
- ☐ Maximization Bias and Double Learning
- ☐ Games, Afterstates, and Other Special Cases
- ☐ Conclusion

Sarsa: On-policy TD Control

- Goal: use of TD prediction methods for the control problem
- Approach
  - Generalized policy iteration (GPI), using TD methods for the evaluation or prediction part
  - As with MC methods, we face the need to trade off exploration and exploitation, and approaches fall into on-policy and off-policy ones

Sarsa: On-policy TD Control

- Sarsa: On-policy TD control method
  - We aim to learn an action-value function rather than a state-value function
  - In particular, for an on-policy method we must estimate $q_\pi(s, a)$ for the current behavior policy π, and for all states s and actions a
  - This can be done using essentially the same TD method for learning v_π
  - An episode consists of an alternating sequence of states and state–action pairs:

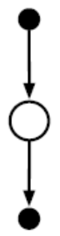


Sutton and Barto, Reinforcement Learning, 2018

  - TD(0) algorithm for action values

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

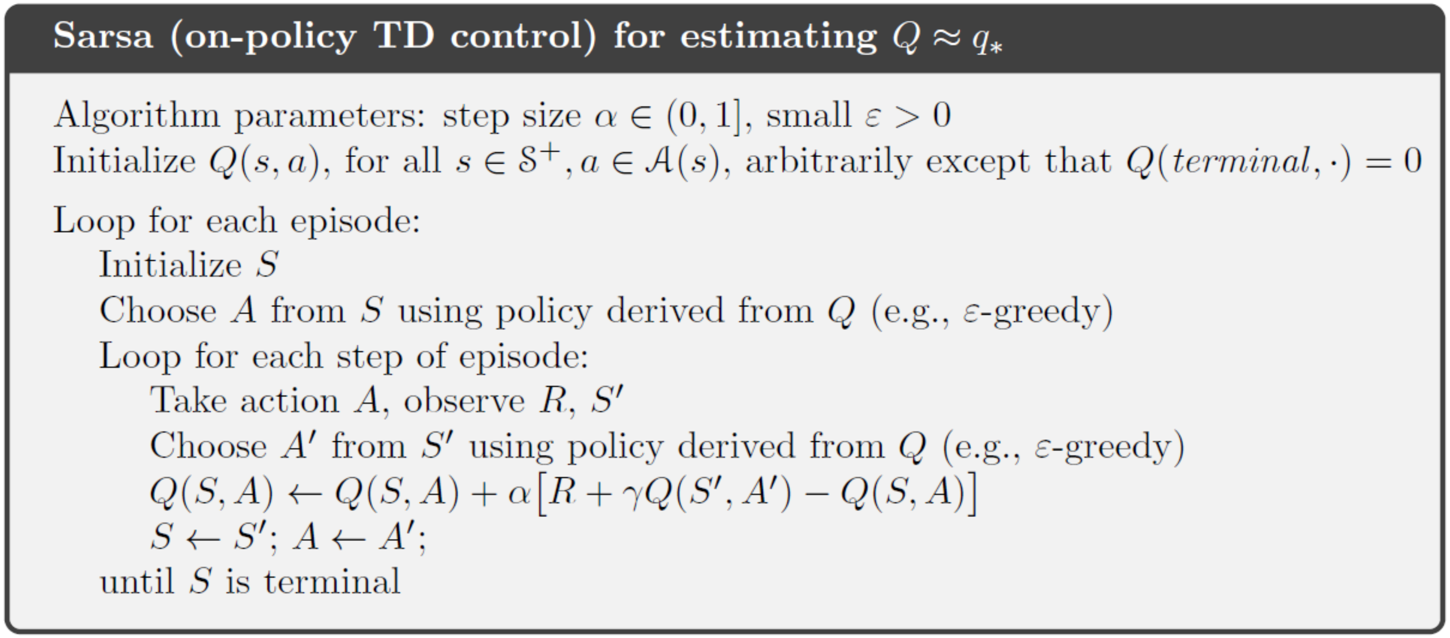
  - This update is done after every transition from a nonterminal state S_t; if S_{t+1} is terminal, then $Q(S_{t+1}, A_{t+1})$ is defined as zero
  - This rule uses every element of the quintuple of events, $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$, that make up a transition from one state–action pair to the next



Sarsa

Sarsa: On-policy TD Control

- Sarsa: On-policy TD control method
  - Continually estimate q_π for the behavior policy π, and at the same time change π toward greediness with respect to q_π



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
```

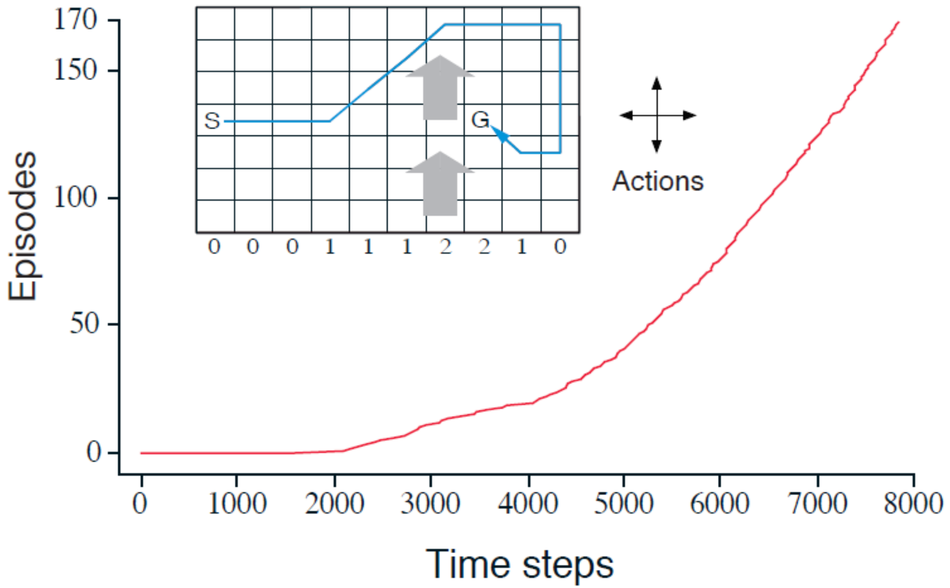
Sutton and Barto, Reinforcement Learning, 2018

Sarsa: On-policy TD Control

- Convergence of Sarsa
  - The convergence properties of the Sarsa algorithm depend on the nature of the policy's dependence on Q
  - For example, one could use ϵ-greedy or ϵ-soft policies
  - Sarsa converges with probability 1 to an optimal policy and action-value function as long as all state–action pairs are visited an infinite number of times and the policy converges in the limit to the greedy policy (e.g., ϵ-greedy policies with ϵ = 1/t)

Example: Windy Gridworld

- A standard gridworld with a crosswind running upward through the middle of the grid
- Actions: up, down, right, and left
- The strength of the wind is given below each column
- An undiscounted episodic task, with constant rewards of −1 until the goal state is reached

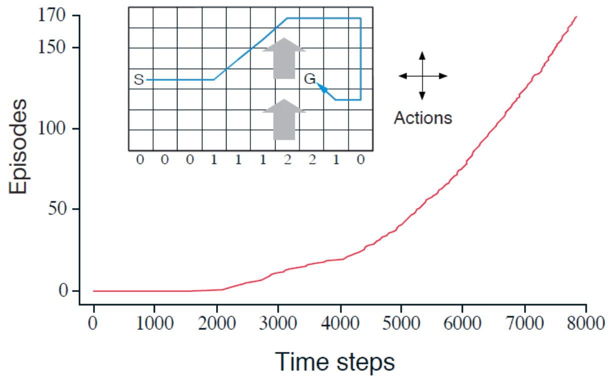


Applying ϵ-greedy Sarsa

Sutton and Barto, Reinforcement Learning, 2018

Example: Windy Gridworld

- The increasing slope of the graph shows that the goal was reached more quickly over time; by 8000 time steps, the greedy policy was long since optimal
- Continued ϵ-greedy exploration kept the average episode length at about 17 steps, two more than the minimum of 15
- MC methods cannot easily be used here because termination is not guaranteed for all policies; if a policy was ever found that caused the agent to stay in the same state, then the next episode would never end
- Online learning methods such as Sarsa do not have this problem because they quickly learn during the episode that such policies are poor, and switch to something else



Sutton and Barto, Reinforcement Learning, 2018

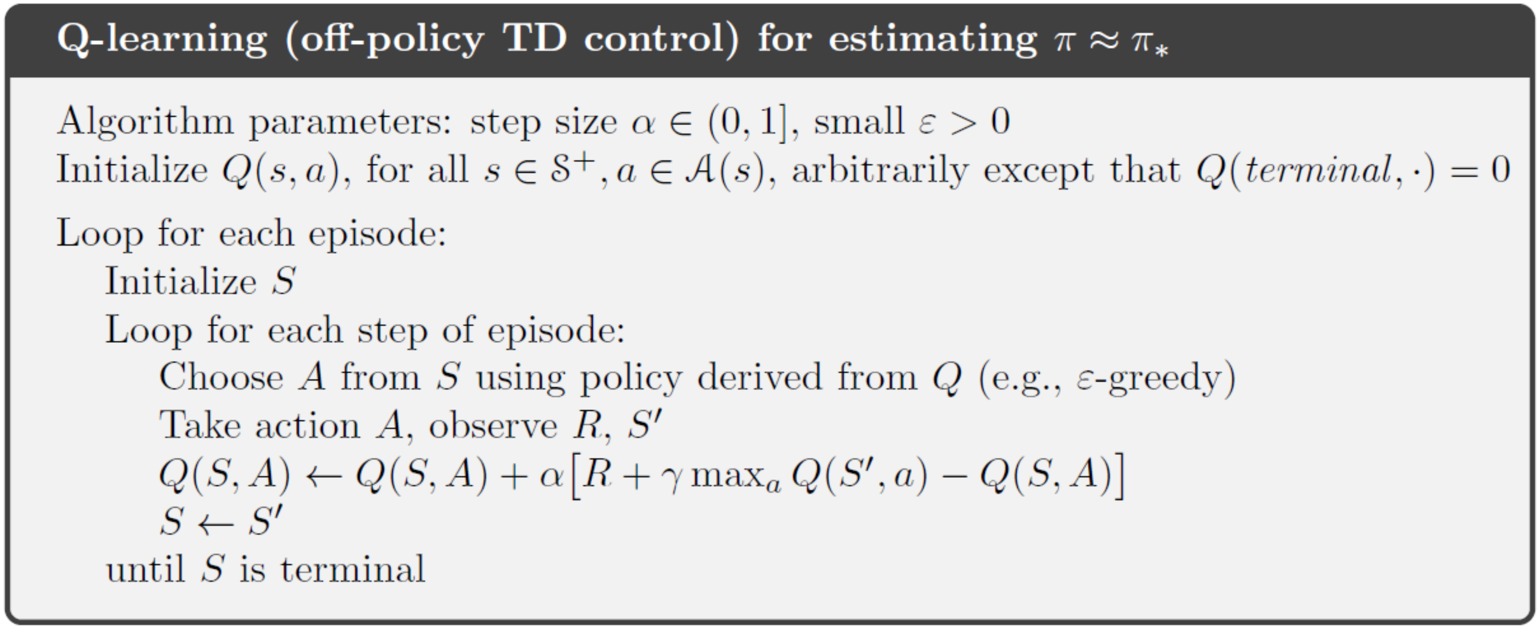
Outline

- ✓ TD Prediction
- ✓ Advantages of TD Prediction Methods
- ✓ Optimality of TD(0)
- ✓ Sarsa: On-policy TD Control
- ☐ **Q-learning: Off-policy TD Control**
- ☐ Expected Sarsa
- ☐ Maximization Bias and Double Learning
- ☐ Games, Afterstates, and Other Special Cases
- ☐ Conclusion

Q-learning: Off-policy TD Control

- Q-learning: one of the early breakthroughs in RL
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$
- The learned action-value function, Q, directly approximates q_*, the optimal action-value function, independent of the policy being followed
- This dramatically simplifies the analysis of the algorithm and enabled early convergence proofs
- The policy still has an effect in that it determines which state–action pairs are visited and updated; however, all that is required for correct convergence is that all pairs continue to be updated
- Under this assumption and a variant of the usual stochastic approximation conditions on the sequence of step-size parameters, Q has been shown to converge with probability 1 to q_*

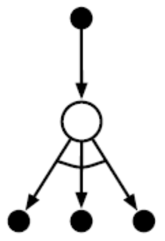
Q-learning: Off-policy TD Control



```
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
```

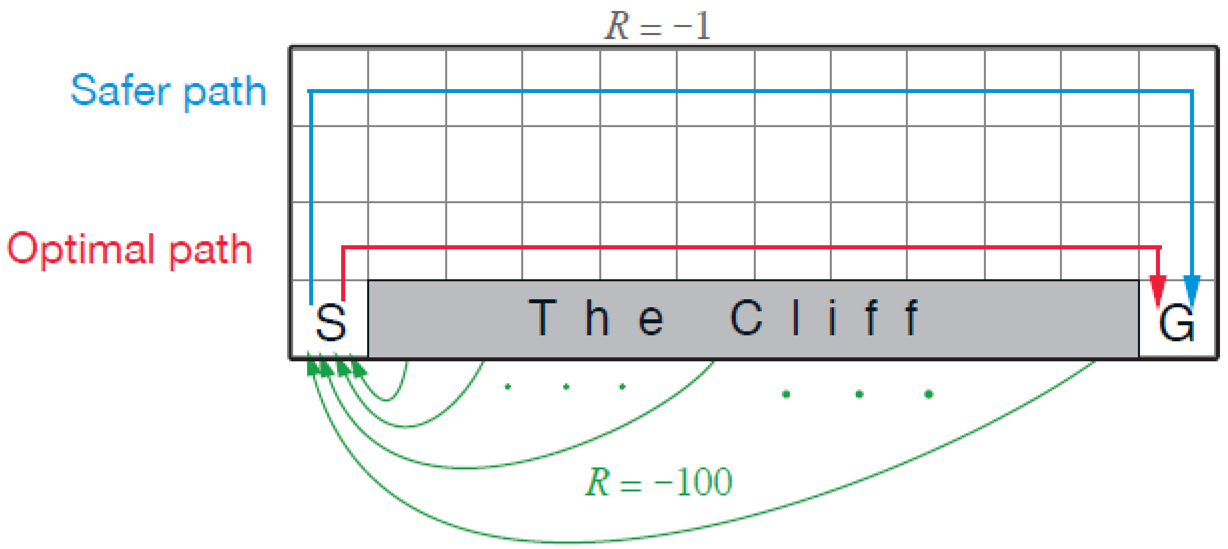


Q-learning

Sutton and Barto, Reinforcement Learning, 2018

Example: Cliff Walking

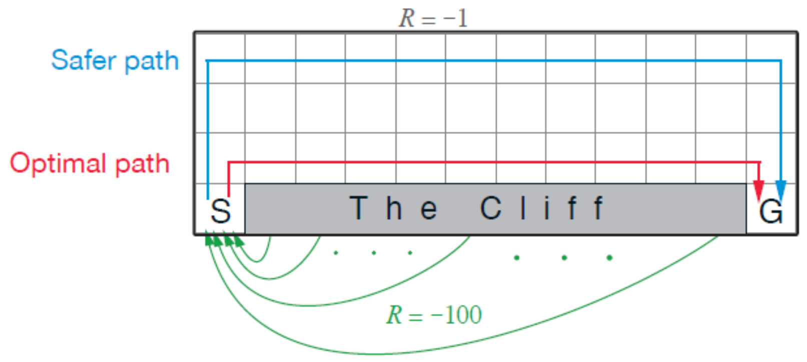
- A standard undiscounted, episodic task, with start and goal states, and the usual actions causing movement up, down, right, and left
- Reward: −1 on all transitions except those into the region marked "The Cliff"; stepping into this region incurs a reward of −100 and sends the agent instantly back to the start

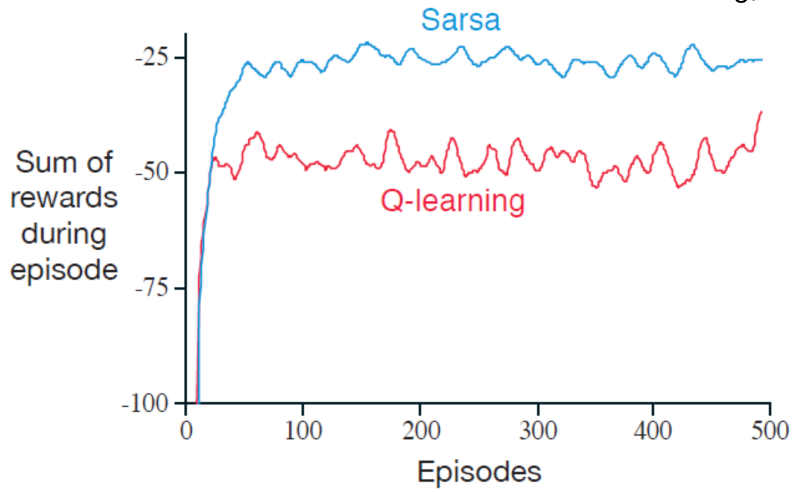


Sutton and Barto, Reinforcement Learning, 2018

Example: Cliff Walking

Sutton and Barto, Reinforcement Learning, 2018





- Performance of Sarsa and Q-learning with ϵ-greedy action selection, ϵ=0.1
- Q-learning learns values for the optimal policy, that travels right along the edge of the cliff; unfortunately, this results in its occasionally falling off the cliff because of the ϵ-greedy action selection
- Sarsa takes the action selection into account and learns the longer but safer path through the upper part of the grid
- Although Q-learning actually learns the values of the optimal policy, its online performance is worse than that of Sarsa, which learns the roundabout policy
- If ϵ were gradually reduced, then both methods would asymptotically converge to the optimal policy

Outline

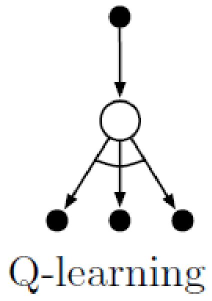
Expected Sarsa

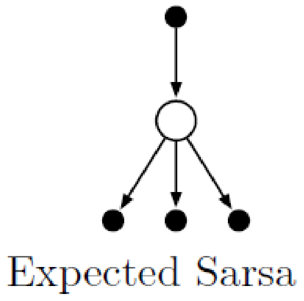
- Consider the learning algorithm that is just like Q-learning except that instead of the maximum over next state–action pairs it uses the expected value, taking into account how likely each action is under the current policy
- This algorithm "Expected Sarsa" uses the update rule

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \mathbb{E}_\pi[Q(S_{t+1}, A_{t+1}) | S_{t+1}] - Q(S_t, A_t)]$$

$$\leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t A_t)]$$

- Given the next state S_{t+1}, this algorithm moves deterministically in the same direction as Sarsa moves in expectation


Q-learning

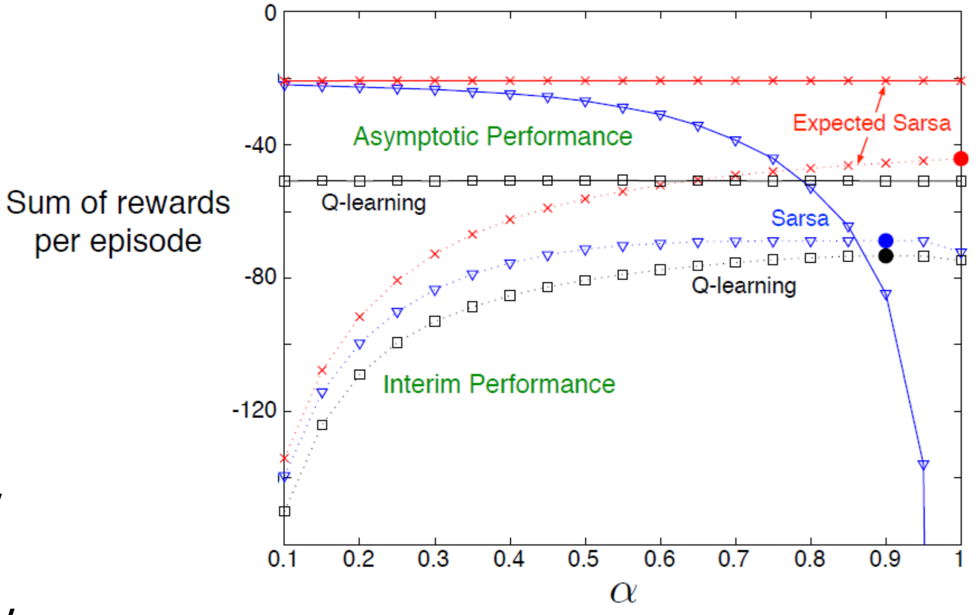

Expected Sarsa

Sutton and Barto, Reinforcement Learning, 2018

Expected Sarsa

- Expected Sarsa is more computationally complex than Sarsa, but in return, it eliminates the variance due to the random selection of A_{t+1}
- Given the same amount of experience, Expected Sara performs slightly better than Sarsa
- In cliff-walking the state transitions are all deterministic and all randomness comes from the policy; in such cases, Expected Sarsa can safely set α = 1 without suffering any degradation of asymptotic performance, whereas Sarsa can only perform well in the long run at a small value of α, at which short-term performance is poor

Cliff-Walking Task



Sutton and Barto, Reinforcement Learning, 2018

Expected Sarsa

- In these cliff walking results, Expected Sarsa was used on-policy, but in general it might work as an off-policy method: it might use a behavior policy different from the target policy π
- For example, suppose π is the greedy policy while behavior is more exploratory; then Expected Sarsa is exactly Q-learning
- In this sense Expected Sarsa subsumes and generalizes Q-learning while reliably improving over Sarsa

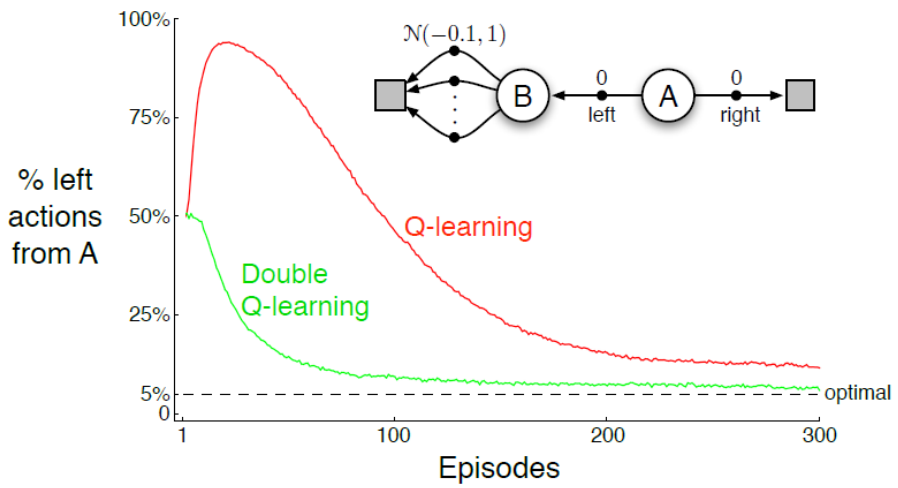
Outline

- ☑ TD Prediction
- ☑ Advantages of TD Prediction Methods
- ☑ Optimality of TD(0)
- ☑ Sarsa: On-policy TD Control
- ☑ Q-learning: Off-policy TD Control
- ☑ Expected Sarsa
- ➡ ☐ **Maximization Bias and Double Learning**
- ☐ Games, Afterstates, and Other Special Cases
- ☐ Conclusion

Maximization Bias and Double Learning

- All the control algorithms that we have discussed so far involve maximization in the construction of their target policies
  - In Q-learning the target policy is the greedy policy given the current action values, which is defined with a max
  - In Sarsa, the policy is often ϵ-greedy, which also involves a maximization operation
- In these algorithms, a maximum over estimated values is used implicitly as an estimate of the maximum value, which can lead to a significant positive bias
- Consider a single state s where there are many actions a whose true values, q(s, a), are all zero but whose estimated values, Q(s, a), are uncertain and thus distributed some above and some below zero
- Maximization bias: the maximum of the true values is zero, but the maximum of the estimates is positive, a positive bias

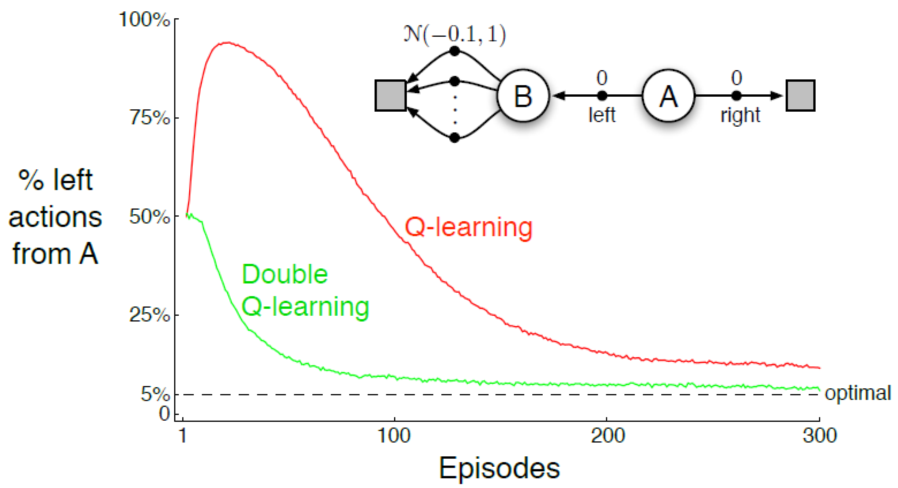
Example: Maximiztion Bias



Sutton and Barto, Reinforcement Learning, 2018

- The MDP has two non-terminal states A and B; episodes always start in A
- The right action transitions immediately to the terminal state with a reward and return of zero; the left action transitions to B, also with a reward of zero, from which there are many possible actions all of which cause immediate termination with a reward drawn from a normal distribution with mean −0.1 and variance 1.0
- The expected return for any trajectory starting with left is −0.1, and thus taking left in state A is always a mistake

Example: Maximiztion Bias



Sutton and Barto, Reinforcement Learning, 2018

- Nevertheless, our control methods may favor left because of maximization bias making B appear to have a positive value
- Q-learning with ϵ-greedy action selection initially learns to strongly favor the left action on this example
- Even at asymptote, Q-learning takes the left action about 5% more often than is optimal at our parameter settings (ϵ = 0.1, α = 0.1, and γ = 1)

Maximization Bias and Double Learning

- Are there algorithms that avoid maximization bias?
- Consider a bandit case in which we have noisy estimates of the value of each of many actions, obtained as sample averages of the rewards received on all the plays with each action
- There will be a positive maximization bias if we use the maximum of the estimates as an estimate of the maximum of the true values
- It is due to using the same samples (plays) both to determine the maximizing action and to estimate its value

Maximization Bias and Double Learning

- Double Learning
  - Suppose we divided the plays in two sets and used them to learn two independent estimates $Q_1(a)$ and $Q_2(a)$, each an estimate of the true value $q(a)$, for all $a \in A$
  - We could then use one estimate, say Q_1, to determine the maximizing action $A^* = argmax_a Q_1(a)$, and the other Q_2, to provide the estimate of its value $Q_2(A^*) = Q_2(argmax_a Q_1(a))$
  - This estimate will then be unbiased in the sense that $\mathrm{E}\big(Q_2(A^*)\big) = q(A^*)$
  - We can also repeat the process with the role of the two estimates reversed to yield a second unbiased estimate $Q_1(argmax_a Q_2(a))$
  - Although we learn two estimates, only one estimate is updated on each play; double learning doubles the memory requirements, but does not increase the amount of computation per step

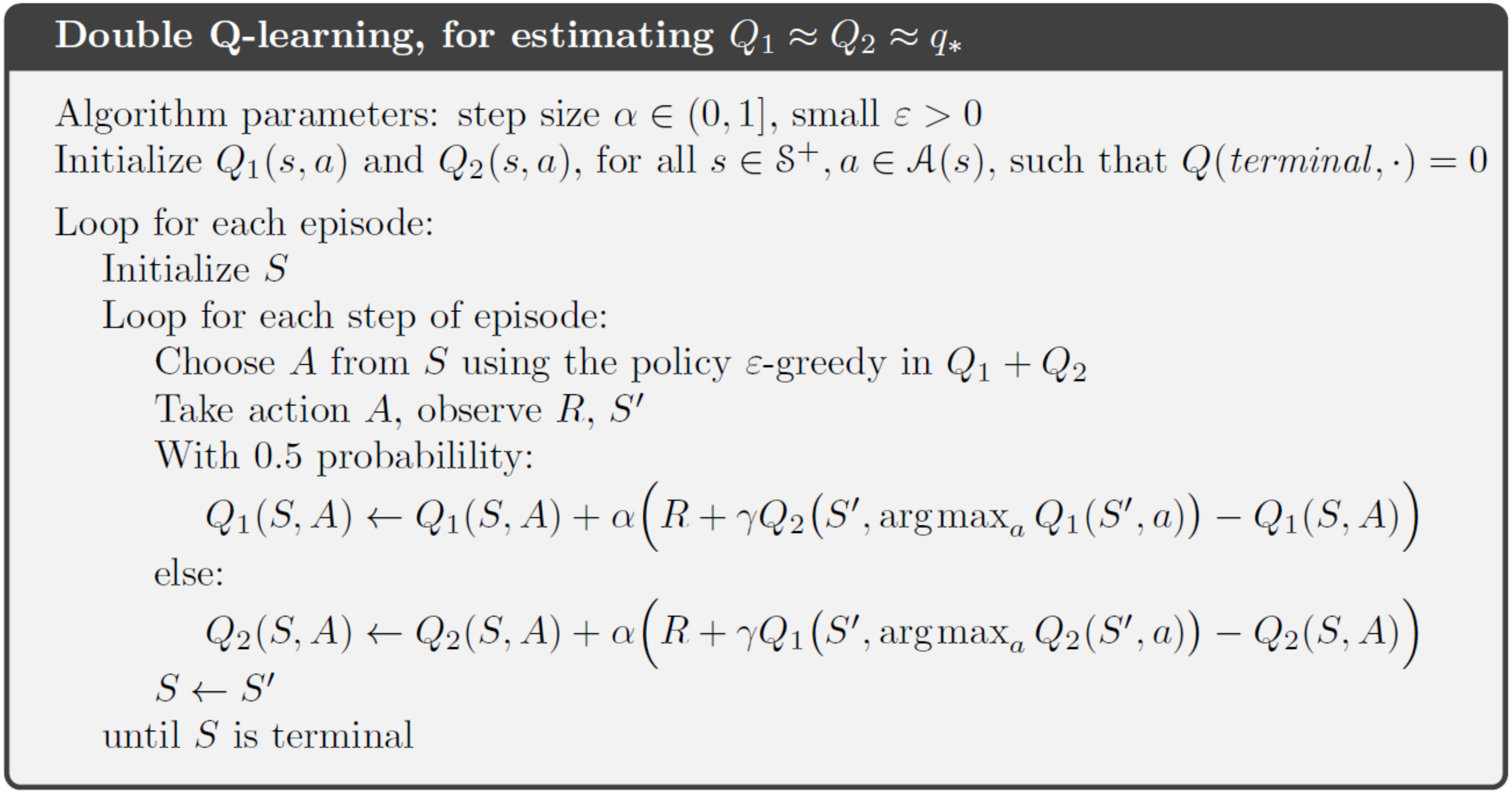
Maximization Bias and Double Learning

- Double Learning for full MDP
  - Double Q-learning is a double learning algorithm analogous to Q-learning
  - Double Q-learning divides the time steps in two, perhaps by flipping a coin on each step
  - If the coin comes up heads, the update is

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha[R_{t+1} + \gamma Q_2\left(S_{t+1}, \operatorname*{argmax}_a Q_1(S_{t+1}, a)\right) - Q_1(S_t, A_t)]$$

  - If the coin comes up tails, then the same update is done with Q_1 and Q_2 switched, so that Q_2 is updated; the two approximate value functions are treated completely symmetrically
  - The behavior policy can use both action-value estimates. For example, an ϵ-greedy policy for Double Q-learning could be based on the average (or sum) of the two action-value estimates

Maximization Bias and Double Learning



Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

```
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q_1(s, a)$ and $Q_2(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, such that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        Choose $A$ from $S$ using the policy $\varepsilon$-greedy in $Q_1 + Q_2$
        Take action $A$, observe $R$, $S'$
        With 0.5 probabilility:
            $Q_1(S, A) \leftarrow Q_1(S, A) + \alpha\Big(R + \gamma Q_2\big(S', \arg\max_a Q_1(S', a)\big) - Q_1(S, A)\Big)$
        else:
            $Q_2(S, A) \leftarrow Q_2(S, A) + \alpha\Big(R + \gamma Q_1\big(S', \arg\max_a Q_2(S', a)\big) - Q_2(S, A)\Big)$
        $S \leftarrow S'$
    until $S$ is terminal
```

Sutton and Barto, Reinforcement Learning, 2018

Outline

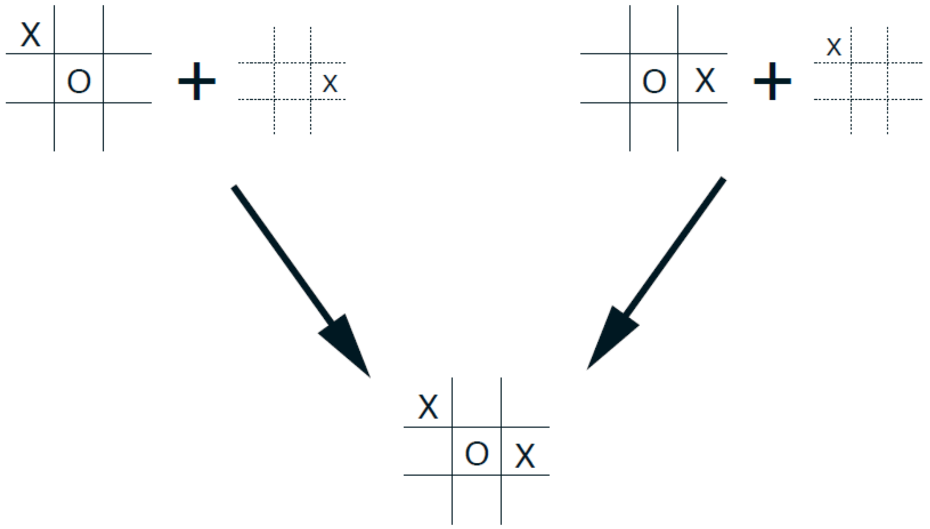
- ☑ TD Prediction
- ☑ Advantages of TD Prediction Methods
- ☑ Optimality of TD(0)
- ☑ Sarsa: On-policy TD Control
- ☑ Q-learning: Off-policy TD Control
- ☑ Expected Sarsa
- ☑ Maximization Bias and Double Learning
- ➡ ☐ **Games, Afterstates, and Other Special Cases**
- ☐ Conclusion

Games, Afterstates, and Other Special Cases

- There are exceptional tasks that are better treated in a specialized way
- Example: Tic-Tac-Toe
  - The TD method for learning to play tic-tac-toe (in Ch. 1) learns something much more like a state-value function, although in general we prefer an action-value function
  - The function learned there is neither an action-value function nor a state-value function in the usual sense
  - A conventional state-value function evaluates states in which the agent has the option of selecting an action, but the state-value function used in tic-tac-toe evaluates board positions *after* the agent has made its move
  - These are called *afterstates*, and value functions over these are called *afterstate value functions*
  - Afterstates are useful when we have knowledge of an initial part of the environment's dynamics but not necessarily of the full dynamics
  - For example, in games we typically know the immediate effects of our moves; we know for each possible chess move what the resulting position will be, but not how our opponent will reply
  - Afterstate value functions are a natural way to take advantage of this kind of knowledge and thereby produce a more efficient learning method

Games, Afterstates, and Other Special Cases

- Why are algorithms using afterstates more efficient?
- A conventional action-value function would map from positions and moves to an estimate of the value; but many position–move pairs produce the same resulting position
- A conventional action-value function would have to separately assess both pairs, whereas an afterstate value function would immediately assess both equally; any learning about the position–move pair on the left would immediately transfer to the pair on the right



Sutton and Barto, Reinforcement Learning, 2018

Games, Afterstates, and Other Special Cases

- Afterstates arise in many tasks, not just games
- For example, in queuing tasks there are actions such as assigning customers to servers, rejecting customers, or discarding information; in such cases the actions are in fact defined in terms of their immediate effects, which are completely known

- In general, many principles of RL apply widely to specialized problems and algorithms
- For example, afterstate methods are still aptly described in terms of generalized policy iteration, with a policy and (afterstate) value function interacting in essentially the same way

Outline

- ✓ TD Prediction
- ✓ Advantages of TD Prediction Methods
- ✓ Optimality of TD(0)
- ✓ Sarsa: On-policy TD Control
- ✓ Q-learning: Off-policy TD Control
- ✓ Expected Sarsa
- ✓ Maximization Bias and Double Learning
- ✓ Games, Afterstates, and Other Special Cases
- ☐ **Conclusion**

Conclusion

- TD methods are alternatives to MC methods for solving the prediction problem
- In both cases, the extension to the control problem is via the idea of generalized policy iteration (GPI) that we abstracted from dynamic programming
- This is the idea that approximate policy and value functions should interact in such a way that they both move toward their optimal values

Conclusion

- One of the two processes making up GPI drives the value function to accurately predict returns for the current policy; this is the prediction problem
- The other process drives the policy to improve locally (e.g., to be ϵ-greedy) with respect to the current value function
- When the first process is based on experience, a complication arises concerning maintaining sufficient exploration
- We can classify TD control methods according to whether they deal with this complication by using an on-policy or off-policy approach
- Sarsa is an on-policy method, and Q-learning is an off-policy method; Expected Sarsa is also an off-policy method

Conclusion

- The methods presented in this chapter are today the most widely used RL methods
- This is probably due to their great simplicity: they can be applied online, with a minimal amount of computation, to experience generated from interaction with an environment; they can be expressed nearly completely by single equations that can be implemented with small computer programs

Questions?