

Temporal Difference Learning

Reading

- Sutton, Richard S., and Barto, Andrew G. Reinforcement learning: An introduction. MIT press, 2018.
 - <http://www.incompleteideas.net/book/the-book-2nd.html>
 - Chapters 6.1-6.4, 7
- David Silver lecture on Dynamic Programming
 - https://www.youtube.com/watch?v=PnHCvfgC_ZA&t=585s
 - Second part of the lecture (on TD learning)

Overview

- Dynamic programming (DP) requires full knowledge of the underlying MDP
 - Only possible for simple systems
- Monte Carlo methods require a lot of data to estimate state values
 - Also, not really online since need to wait for each episode to finish (and get the final reward)
 - Some tasks are continuous, others have very long episodes
 - Re-estimating state values for adapting policies requires yet more data
- Temporal Difference (TD) learning is the best of both worlds
 - Essentially a hybrid approach between the two
 - Main idea behind Q-learning also

Online State Value Estimation

- Suppose we have a fixed policy π and would like to estimate state values $v_{\pi}(s)$
- Suppose we start with some estimates $V(s)$ and would like to update them online
- With Monte Carlo methods, we need to wait until the end of each episode

– Already saw the incremental (off-policy) implementation:

$$V^k(s) = V^{k-1}(s) + \frac{\rho_{t:T,k}}{\sum_{j=1}^k \rho_{t:T,j}} \left(G_{t,k} - V^{k-1}(s) \right)$$

- Can replace the ρ term with a “learning rate” α

$$V^k(s) = V^{k-1}(s) + \alpha \left(G_{t,k} - V^{k-1}(s) \right)$$

– where α is now a hyperparameter

Online State Value Estimation

- Suppose we don't want to wait until the end of the episode
 - i.e., instead of using G_t , we'd like to use each R_t
 - Can update policy after every step (much more efficient)
- In Monte Carlo learning, the value estimate is updated based on the difference between new return and current estimate

$$\alpha \left(G_{t,k} - V^{k-1}(s) \right)$$

- How do we adapt this idea to the case of a one-step reward?
 - What property does v_π have?
 - Bellman equation: $v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s]$
 - Can replace the return with a bootstrapped estimate:

$$R_{t+1} + \gamma V^{k-1}(S_{t+1})$$

TD State Value Estimation

- Update $V(S_t)$ after each step in an episode

$$V'(S_t) = V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

- Called a bootstrapping method because it is based in part on our existing estimates

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated

Algorithm parameter: step size $\alpha \in (0, 1]$

Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\text{terminal}) = 0$

Loop for each episode:

 Initialize S

 Loop for each step of episode:

$A \leftarrow$ action given by π for S

 Take action A , observe R, S'

$V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$

$S \leftarrow S'$

 until S is terminal

TD as a hybrid method

- Similar to MC, TD updates estimates based on the difference between new and predicted data
 - A standard approach in general estimation theory
 - In some sense, this is a Bayesian method
 - Our bootstrapped estimate of $V(S_t)$ is the prior
- TD is also similar to DP, since it uses the Bellman equation
 - Makes use of the Markov property and the MDP structure
 - Implicitly estimating the MDP structure
- We'll see that TD has hyper-parameters that can shift it along the “line” between DP and MC

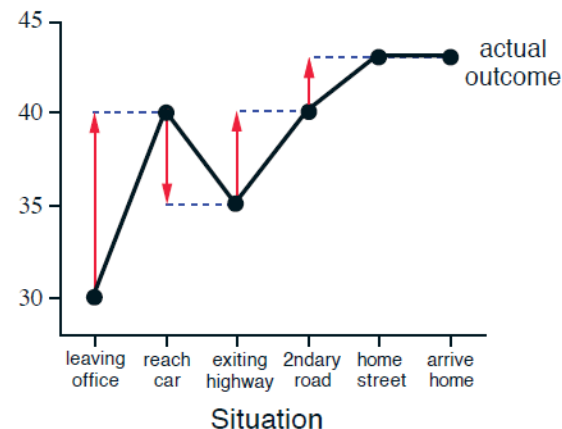
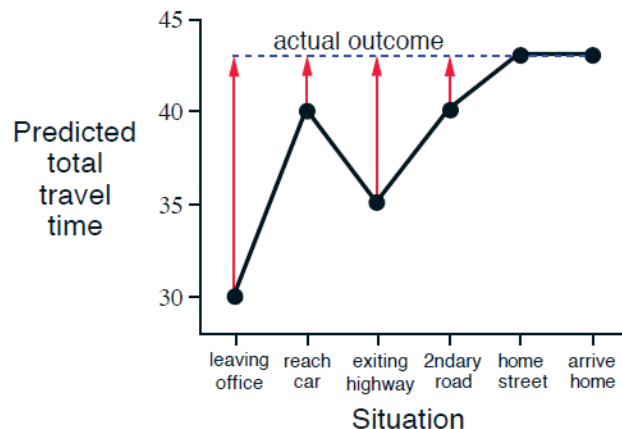
Example: Driving Home

- Suppose you leave your office at 6 on Friday
 - You have an ETA from previous trips
 - However, unforeseen events delay you
- You have pre-existing predicted time-to-go's for each situation (state)

<i>State</i>	<i>Elapsed Time (minutes)</i>	<i>Predicted Time to Go</i>	<i>Predicted Total Time</i>
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

Example: Driving Home

- TD allows you to update your estimates after each event
 - E.g., you reach the car, and it is raining
 - You update your ETA based on the new information
- MC may introduce very large changes since it doesn't factor in intermediate states
 - Value updates may have great variance
 - An outlier data point, e.g., a slow truck, may cause a big change in the MC estimates

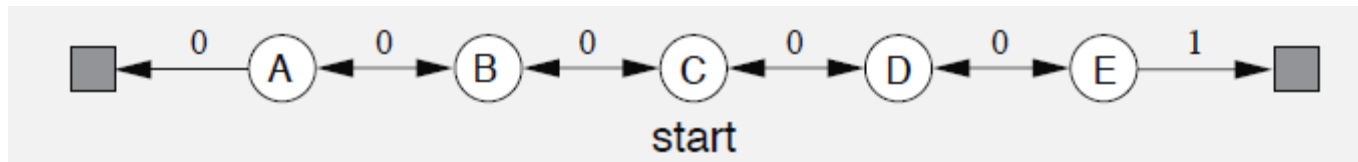


Advantages of TD Prediction

- No need to know the MDP, unlike DP
- Can be performed online, unlike Monte Carlo methods
- Usually more data-efficient than Monte Carlo methods since current estimate $V(S_t)$ can act as a prior
- Can update the state values after every step
 - May bring significant benefits if we also adapt the policy
 - A better policy may lead to seeing better rewards and faster learning overall
- MC minimizes the square error on the training set
 - TD converges to the maximum-likelihood estimate
- Both MC and TD converge to the true values given enough data
 - Proof is a bit technical, relies on stochastic processes

Advantages of TD Prediction, Example

- Consider the following Markov Reward Process
 - There's a 0.5 chance of taking each transition
 - Reward of 1 only when we reach the right square

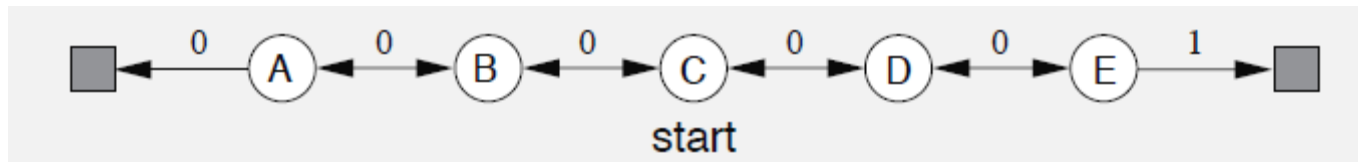


- What are the values of the various states?
 - Clearly, $v(C) = 0.5$ (equal chance of reaching each side)
 - Use matrix form of Bellman equation: $v(s) = \mathbf{R} + \mathbf{P}v(s)$

– Where $\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \mathbf{R} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0.5 \\ 0 \end{bmatrix}$

Advantages of TD Prediction, Example

- Consider the following Markov Reward Process
 - There's a 0.5 chance of taking each transition
 - Reward of 1 only when we reach the right square

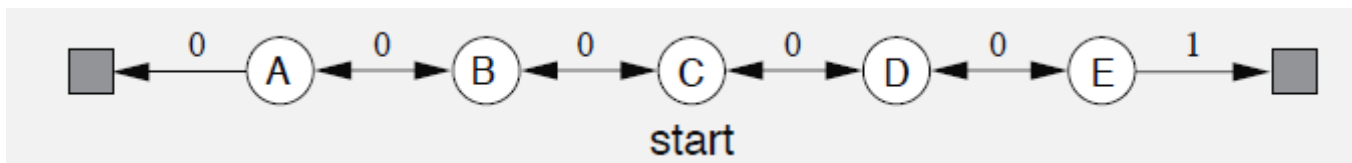


- Need to use iterative policy evaluation method
 - Why?
 - Matrix $I - P$ is not invertible
 - Has an eigenvalue of 0
- Without discounting, state values are:

$$v_{\pi}(C) = 0.5, v_{\pi}(A) = \frac{1}{6}, v_{\pi}(B) = \frac{2}{6}, v_{\pi}(D) = \frac{4}{6}, v_{\pi}(E) = \frac{5}{6}$$

Advantages of TD Prediction, Example

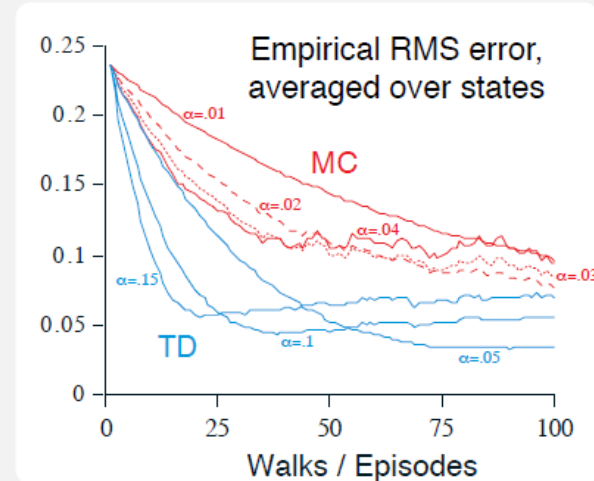
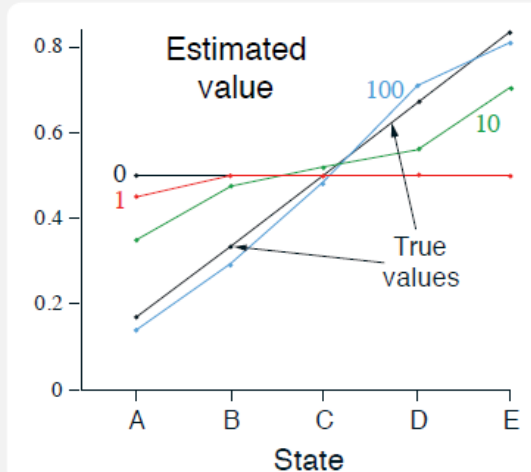
- Consider the following Markov Chain (with 0.5 chance of taking each transition)



– Without discounting, state values are:

$$v_{\pi}(C) = 0.5, v_{\pi}(A) = \frac{1}{6}, v_{\pi}(B) = \frac{2}{6}, v_{\pi}(D) = \frac{4}{6}, v_{\pi}(E) = \frac{5}{6}$$

TD converges
after ~100
episodes



TD Convergence Properties

- How does the learning rate α affect convergence?
 - Smaller α means slower convergence
 - Larger α means faster convergence but algorithm converges with bigger estimation error
- In order to truly converge to the optimal values, one needs to decrease α progressively
 - Will discuss in more detail when we get to Q learning

TD Batch Estimation

- Suppose we have fixed data from N episodes
 - Each episode is the usual $S_1, A_1, R_2, \dots$
- Can apply TD estimation in batch fashion
 - Iterate through the episodes until convergence
 - TD guaranteed to converge
 - Guaranteed to converge to the true values as $N \rightarrow \infty$

Example: Difference between MC and TD

- Suppose we have an unknown MDP and observe the following 8 episodes with rewards
 - $A, 0, B, 0$ $B, 1$
 - $B, 1$ $B, 1$
 - $B, 1$ $B, 1$
 - $B, 1$ $B, 0$
- What is your guess for the value at B ?
 - A guess of $6/8$ seems reasonable
- What is your guess for the value at A ?
 - Both $6/8$ and 0 seem reasonable

Example: Difference between MC and TD

- Suppose we have an unknown MDP and observe the following 8 episodes with rewards
 - $A, 0, B, 0$ $B, 1$
 - $B, 1$ $B, 1$
 - $B, 1$ $B, 1$
 - $B, 1$ $B, 0$
- Both MC and TD output $6/8$ for $v(B)$
 - Why?
 - When we have a terminal state, TD is essentially MC
- If you use a constant α , you won't actually converge
 - Will bounce around $6/8$
 - In true MC, $\alpha = 1/(1 + k)$

Example: Difference between MC and TD, cont'd

- Suppose we have an unknown MDP and observe the following 8 episodes with rewards
 - $A, 0, B, 0$ $B, 1$
 - $B, 1$ $B, 1$
 - $B, 1$ $B, 1$
 - $B, 1$ $B, 0$
- MC will output 0 for $v(A)$
 - Why?
 - Only run that visited A had a return of 0

Example: Difference between MC and TD, cont'd

- Suppose we have an unknown MDP and observe the following 8 episodes with rewards
 - $A, 0, B, 0$ $B, 1$
 - $B, 1$ $B, 1$
 - $B, 1$ $B, 1$
 - $B, 1$ $B, 0$
- TD will output $6/8$ for $v(A)$
 - Why?
 - Eventually, $V(B)$ will converge to $6/8$
 - When we process episode 1 after that (with $\alpha = 0.1$):
 - (Assuming currently $V(A) = 0$ for simplicity)
$$V'(A) = V(A) + \alpha(0 + V(B) - V(A)) = 3/40$$
 - Keep iterating and $V(A)$ will bounce around $V(B)$

SARSA: On-policy TD Control

- On-policy control idea is the same as before
 - Estimate the state/action values
 - For each state, choose the action with the highest value
- Action value recursion is the same as the state one:
$$Q'(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$
 - Requires also knowing the next action A_{t+1}
 - This makes it on-policy
 - Uses every quintuplet $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$
 - Hence the name
- To ensure exploration, still need ϵ -greedy policies

SARSA: On-policy TD Control, cont'd

- After every step, alternate between policy evaluation and policy iteration
- Guaranteed to converge
 - (will see a sketch of the proof for Q-learning)

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 $Q(\text{terminal}, \cdot) = 0$

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)

$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$

$S \leftarrow S'; A \leftarrow A';$

 until S is terminal

n -step bootstrapping

- MC methods wait for the return at the end of the episode
- TD updates its estimates and policies after every step
- TD learning may be noisy as it updates estimate too frequently
 - Smoothing over multiple steps would bring better results
- Can we have something in between?
 - Something that updates estimates/policies after n steps?
 - Yes, n -step bootstrapping!
 - Same idea as TD learning but applied over n steps

***n*-step return**

- The MC return is

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-t-1} R_T$$

- The TD return is just R_{t+1}

- What is the n -step TD return?

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n}$$

- The MC evaluation recursion is

$$V'(S_t) = V(S_t) + \alpha(G_t - V(S_t))$$

- The TD recursion is

$$V'(S_t) = V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

- What is the n -step recursion?

$$V'(S_t) = V(S_t) + \alpha(G_{t:t+n} + \gamma^n V(S_{t+n}) - V(S_t))$$

n -step TD Prediction

- What is the n -step recursion?

$$V'(S_t) = V(S_t) + \alpha(G_{t:t+n} + \gamma^n V(S_{t+n}) - V(S_t))$$

- Note that we need to wait for n steps to receive $G_{t:t+n}$
 - As $n \rightarrow \infty$, n -step TD converges to MC
- Larger n allow us to get a better estimate of $v_\pi(s)$
 - Adding stability at the expense of slower convergence

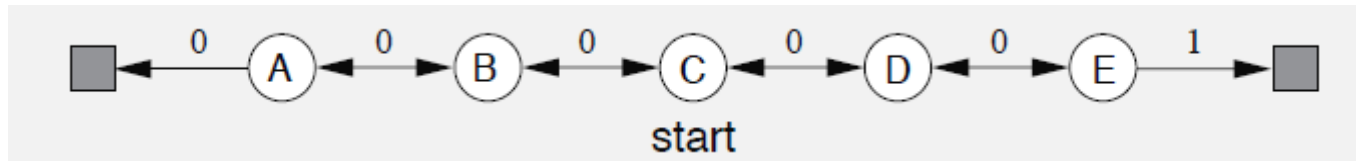
n -step TD for estimating $V \approx v_\pi$

```
Input: a policy  $\pi$ 
Algorithm parameters: step size  $\alpha \in (0, 1]$ , a positive integer  $n$ 
Initialize  $V(s)$  arbitrarily, for all  $s \in \mathcal{S}$ 
All store and access operations (for  $S_t$  and  $R_t$ ) can take their index mod  $n + 1$ 

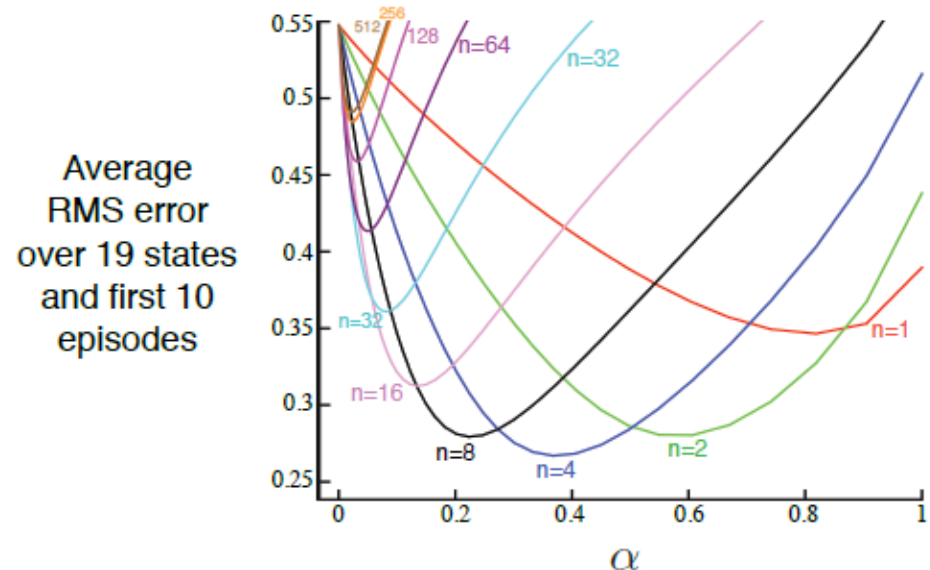
Loop for each episode:
  Initialize and store  $S_0 \neq \text{terminal}$ 
   $T \leftarrow \infty$ 
  Loop for  $t = 0, 1, 2, \dots$ :
    If  $t < T$ , then:
      Take an action according to  $\pi(\cdot | S_t)$ 
      Observe and store the next reward as  $R_{t+1}$  and the next state as  $S_{t+1}$ 
      If  $S_{t+1}$  is terminal, then  $T \leftarrow t + 1$ 
       $\tau \leftarrow t - n + 1$  ( $\tau$  is the time whose state's estimate is being updated)
      If  $\tau \geq 0$ :
         $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n, T)} \gamma^{i-\tau-1} R_i$ 
        If  $\tau + n < T$ , then:  $G \leftarrow G + \gamma^n V(S_{\tau+n})$  ( $G_{\tau:\tau+n}$ )
         $V(S_\tau) \leftarrow V(S_\tau) + \alpha [G - V(S_\tau)]$ 
      Until  $\tau = T - 1$ 
```

Example: Random Walk

- Recall random walk example



- Suppose we have 19 states instead of 5, i.e., 9 on each side
- And suppose reward is -1 on the left
- Lowest error for $n = 4$
 - Best trade-off in this case
- MC ($n = \infty$) is pretty much the worst



n -step SARSA: On-policy Control

- Same idea as before
 - Estimate the state/action values
 - For each state, choose the action with the highest value

- Action value recursion is the same as the state one:

$$Q'(S_t, A_t) = Q(S_t, A_t) + \alpha[G_{t:t+n} + \gamma^n Q(S_{t+n}, A_{t+n}) - Q(S_t, A_t)]$$

n -step Sarsa for estimating $Q \approx q_*$ or q_π

```
Initialize  $Q(s, a)$  arbitrarily, for all  $s \in \mathcal{S}, a \in \mathcal{A}$ 
Initialize  $\pi$  to be  $\varepsilon$ -greedy with respect to  $Q$ , or to a fixed given policy
Algorithm parameters: step size  $\alpha \in (0, 1]$ , small  $\varepsilon > 0$ , a positive integer  $n$ 
All store and access operations (for  $S_t$ ,  $A_t$ , and  $R_t$ ) can take their index mod  $n + 1$ 

Loop for each episode:
  Initialize and store  $S_0 \neq \text{terminal}$ 
  Select and store an action  $A_0 \sim \pi(\cdot | S_0)$ 
   $T \leftarrow \infty$ 
  Loop for  $t = 0, 1, 2, \dots$ :
    If  $t < T$ , then:
      Take action  $A_t$ 
      Observe and store the next reward as  $R_{t+1}$  and the next state as  $S_{t+1}$ 
      If  $S_{t+1}$  is terminal, then:
         $T \leftarrow t + 1$ 
      else:
        Select and store an action  $A_{t+1} \sim \pi(\cdot | S_{t+1})$ 
     $\tau \leftarrow t - n + 1$  ( $\tau$  is the time whose estimate is being updated)
    If  $\tau \geq 0$ :
       $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n, T)} \gamma^{i-\tau-1} R_i$ 
      If  $\tau + n < T$ , then  $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$  ( $G_{\tau:\tau+n}$ )
       $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha [G - Q(S_\tau, A_\tau)]$ 
      If  $\pi$  is being learned, then ensure that  $\pi(\cdot | S_\tau)$  is  $\varepsilon$ -greedy wrt  $Q$ 
    Until  $\tau = T - 1$ 
```