Chapter 5: Temporal-Difference Learning

- TD learning is a combination of Monte Carlo ideas and dynamic programming (Bellman equation).
- Can learn directly from raw experience without a model of environment's dynamics (i.e., without $p(s',r|s,a)$).
- TD learning applies to episodic and continual MDPs.
- First look at prediction/evaluation. Then look at control (finding an optimal policy).

TD Prediction/Evaluation

- Policy π is fixed.
  - As usual, will have a table for V(s), $s \in \mathcal{S}$.
- Recall Monte Carlo methods update estimates V(s) only at the ends of episodes.
- TD-methods make updates after every time step within episodes

Reminder: Polyak Averaging

- Let $x_1, x_2, x_3, \ldots$ be a sequence of numbers
- Let $y_n = \frac{1}{n}\sum_{i=1}^{n} x_i$ "average" or "arithmetic mean"
- Arithmetic mean puts the same weight on each of the x_i's. But may want to put more weight on more recent x_i's.

- Polyak average: $z_n = (1-\alpha)z_{n-1} + \alpha x_n$, $z_0 = 0$ $0 < \alpha < 1$ is a hyperparameter

- Estimator is updated by put weight 1-α on earlier estimate and weight α on most recent data
- Can show: $z_n = \alpha [\sum_{j=1}^{n} (1-\alpha)^{n-j} x_j]$
- More weight is placed on recent samples.

Reminder: unbiased estimator

- Suppose that the random variable X is an estimate for the real number a. We say X is **unbiased** if E[X] = a.

- Simple example: Suppose we toss an ordinary coin 10 times. If *i*th toss is heads, let $Y_i = 1$; if tails, $Y_i = 0$. Let $X = (Y_1+\ldots+Y_{10})/10$. Since E[X] = ½, X is an biased estimator for ½. However X may not be equal to ½.
- For same example, let $Z = \ln [\exp(Y_1) +\ldots+\exp(Y_{10})]/10$. Can be shown $E[Z] \neq ½$. Thus Z is a biased estimator or ½.

An alternative perspective on Bellman equation:

- Recall Bellman equation for fixed policy:

$v_\pi(s) = \sum_a \pi(a|s) [\, r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, v_\pi(s') \,]$ (1)

- An alternative way of writing this:

$= E_\pi[R_{t+1} + \gamma\, v_\pi(S_{t+1}) \mid S_t = s]$ (2)

- In class quiz: Show that (2) is equal to RHS of (1). Hint: Condition (2) on A_t and S_{t+1}.

- So if $S_t = s$, then $R_{t+1} + \gamma\, v_\pi(S_{t+1})$ is an unbiased estimator of $v_\pi(s)$

TD Prediction/Evaluation

- Let V(s) be our current estimate of $v_\pi(s)$.
- While running an episode with policy π, enter state S_t and consider updating $V(S_t)$. Use Polyak averaging to update V(s):

$$V(S_t) \leftarrow (1- \alpha)\, V(S_t) + \alpha \text{ (new unbiased estimator)}$$

- Take an action, get R_{t+1} and S_{t+1}.
- Just showed $R_{t+1} + \gamma\, v_\pi(S_{t+1})$ is an unbiased estimator of $v_\pi(S_t)$
- Ideally we use: $V(S_t) \leftarrow (1- \alpha)\, V(S_t) + \alpha\, [R_{t+1} + \gamma\, v_\pi(S_{t+1})\,]$
- Why can't we use this for our update?
- Replace $v_\pi(S_{t+1})$ with $V(S_{t+1})$:
  - $V(S_t) \leftarrow (1- \alpha)\, V(S_t) + \alpha\, [R_{t+1} + \gamma\, V(S_{t+1})\,]$
- New estimate of $V(S_t)$ is equal to a combination of old estimate of $V(S_t)$ and the "target" $R_{t+1} + \gamma\, V(S_{t+1})$

TD Prediction/Evaluation

- From previous slide: $V(S_t) \leftarrow (1- \alpha)\, V(S_t) + \alpha\, [R_{t+1} + \gamma\, V(S_{t+1})\,]$
- But $(1- \alpha)\, V(S_t) + \alpha\, [R_{t+1} + \gamma\, V(S_{t+1})\,] = V(S_t) + \alpha\, [R_{t+1} + \gamma\, V(S_{t+1}) - V(S_t)\,]$
- So equivalently:

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

- The term $R_{t+1} + \gamma\, V(S_{t+1}) - V(S_t)$ is referred to the "temporal difference" or the "temporal error".
- "Bootstrapping": The estimate $V(S_t)$ is updated using part of existing estimate, namely, using $V(S_{t+1})$. Thus if S_t= 7 and S_{t+1}=2, we are using V(2) to update v(7): bootstrapping.

Tabular TD 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 $V(terminal) = 0$

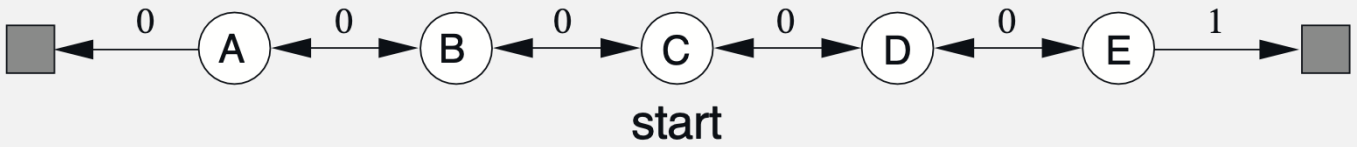
Loop for each episode:
    Initialize S
    Loop for each step of episode:
        $A \leftarrow$ action given by π for S
        Take action A, observe R, S'
        $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
        $S \leftarrow S'$
    until S is terminal

Comparison with Monte Carlo Prediction

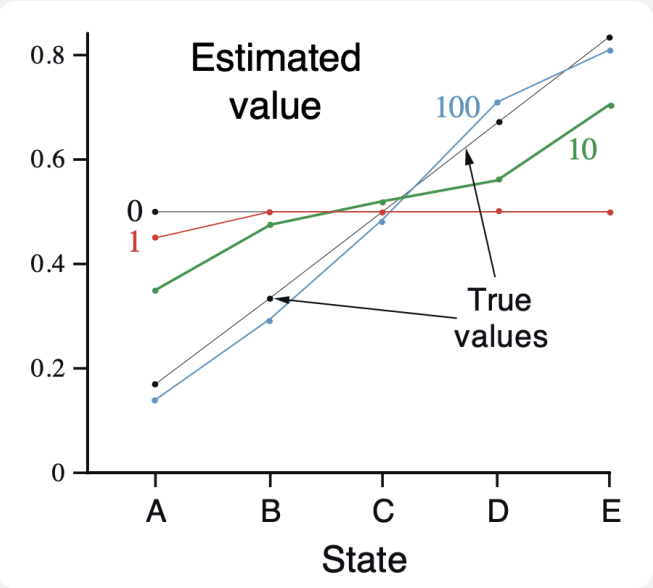
- Neither requires a model of the environment (unlike DP).
- TD can be employed for continuing and episodic MDPs.
- With Monte Carlo, V(s) will converge to $v_\pi(s)$. With TD, V(s) will fluctuate around $v_\pi(s)$. Can guarantee convergence by properly decreasing step size α.
- With TD, updates are done within episodes, potentially accelerating learning. With MC, need to wait until end of (possibly long) episode.

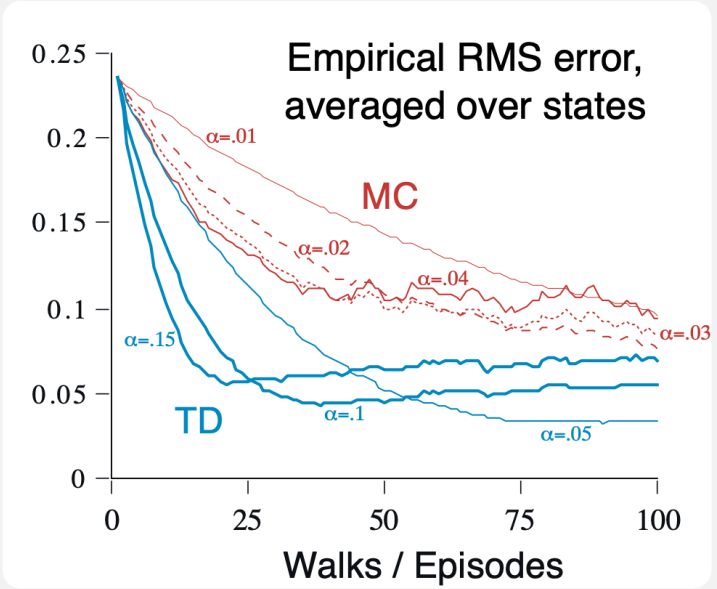
Example 6.2 Random Walk

In this example we empirically compare the prediction abilities of TD(0) and constant-α MC when applied to the following Markov reward process:



A *Markov reward process*, or MRP, is a Markov decision process without actions. We will often use MRPs when focusing on the prediction problem, in which there is no need to distinguish the dynamics due to the environment from those due to the agent. In this MRP, 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. For example, a typical episode might consist of the following state-and-reward sequence: C, 0, B, 0, C, 0, D, 0, E, 1. Because this task is undiscounted, the true value of each state is the probability of terminating on the right if starting from that state. Thus, the true value of the center state is $v_\pi(\mathsf{C}) = 0.5$. The true values of all the states, A through E, are $\frac{1}{6}, \frac{2}{6}, \frac{3}{6}, \frac{4}{6}$, and $\frac{5}{6}$.





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 ($\alpha = 0.1$ in this example), the values fluctuate indefinitely in response to the outcomes of the most recent episodes. The right graph shows learning curves for the two methods for various values of α. The performance measure shown 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. In all cases the approximate value function was initialized to the intermediate value $V(s) = 0.5$, for all s. The TD method was consistently better than the MC method on this task.

Here they are using a modified version of the MC algorithm:

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

In class do the following:

Exercise 6.3 From the results shown in the left graph of the random walk example it appears that the first episode results in a change in only $V(\mathsf{A})$. What does this tell you about what happened on the first episode? Why was only the estimate for this one state changed? By exactly how much was it changed? □

On-Policy versus Off-Policy Control Algorithms

- On-policy: update the current learned policy with data that was recently generated by the current learned policy. MCES is on-policy: use data from episode generated by current policy to update the policy.

- Off-policy: update current learned policy with data that is not necessarily from the current learned policy.

Q-Learning: Off Policy Data

- Recall that if we give the environment a state s and action a, it returns a reward r and a new state s', thereby generating a tuple (s,a,r,s').
- Let's generate lots of these tuples and create a dataset:

$$D = \{ (s_i, a_i, r_i, s'_i), i=0, 1, 2, ...\}$$

- We only require that every state-action pair (s,a) occurs infinitely often in the dataset
- The data set could be generated by a behavioral policy, which is different from the policy we are trying to learn. For example, it could be generated by the fixed policy π that chooses all possible actions with equal probability.

Q- Learning for Continuing Task

Inputs: step size α, dataset D

Initialize Q(s,a) for all s,a arbitrarily

Repeat:

- Select the next (s,a,r,s') from the dataset D
- $Q(s,a) \leftarrow Q(s,a) + \alpha [r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$

- Note, Q-learning is using Polyak averaging:

$$Q(s,a) \leftarrow (1- \alpha) Q(s,a) + \alpha [r + \gamma \max_{a'} Q(s',a')]$$

- Under certain conditions on α, Q(s,a) converges to $q^*(s,a)$
- Can use learned policy $\pi(s) = \text{argmax}_a Q(s,a)$ during inference

Q- Learning using a policy to generate the data

Inputs: step size α, stochastic behavioral policy π_b

Initialize Q(s,a) for all s,a arbitrarily

Initialize initial state s

Repeat:

- Select action from $\pi_b(.|s)$
- Take action a with environment in state s; observe r, s'
- $Q(s,a) \leftarrow Q(s,a) + \alpha [r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
- $s \leftarrow s'$

- Note, that in above version of the algorithm, the dataset D is generated by the behavioral policy π_b. We only need to require $\pi_b(a|s) > 0$ for all actions and states.
- Learned policy is $\pi(s) = \text{argmax}_a Q(s,a)$

Q-Learning with behavioral policy similar to learned policy

Inputs: step size α, randomization parameter ε

Initialize Q(s,a) for all s,a arbitrarily

Initialize initial state s

Repeat:

- Select a = $\text{argmax}_{a'}$ Q(s,a') with prob (1-ε) ; select random a with prob ε
- Take action a with environment in state s; observe r, s'
- $Q(s,a) \leftarrow Q(s,a) + \alpha [r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
- $s \leftarrow s'$

- Note that in above version of the algorithm, the dataset D is generated by the the ε-greedy behavioral policy
- Learned policy is $\pi(s) = \text{argmax}_a Q(s,a)$ is the greedy policy
- This is the version of the algorithm that is in the textbook.

On Convergence of Q-Learning

- Instead of $Q(s,a) \leftarrow Q(s,a) + \alpha [r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$, use
  $Q_{n+1}(s,a) \leftarrow Q_n(s,a) + \alpha_n [r + \gamma \max_{a'} Q_n(s',a') - Q_n(s,a)]$
  with $\sum_{n=1}^{\infty} \alpha_n = \infty$, $\sum_{n=1}^{\infty} \alpha_n^2 < \infty$. Then $Q_n(s,a) \rightarrow q^*(s,a)$

- Can then obtain optimal policy $\pi^*(s) = \text{argmax}_a\, q^*(s,a)$

- Proof is based on "stochastic approximations theorem," which is a generalization of the contraction mapping theorem.

Maximization Bias and Double Learning

- Q-Learning: $Q(s,a) \leftarrow Q(s,a) + \alpha [r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
- But the Q function is an inaccurate estimate, particularly at the beginning of training.
- The maximization in $\max_{a'} Q(s',a')$ can exploit where the estimate is way off, making $\max_{a'} Q(s',a')$ much bigger than it should be.
- Toy Example: Suppose for some state s, the current estimate is $Q(s,a) = q(s,a) + \varepsilon_a$ where the ε_a's are iid $N(0,\sigma^2)$ errors. For simplicity, further suppose $q(s,a) = 0$ for all a. Then $\max_a Q(s,a) = \max_a \varepsilon_a >> 0 = \max_a q(s,a)$
  - Thus $\max_a Q(s,a)$ is a biased estimator for $\max_a q(s,a)$
- In summary, vanilla Q-learning will over estimate q(s,a), leading to slower convergence and less sample inefficiency.

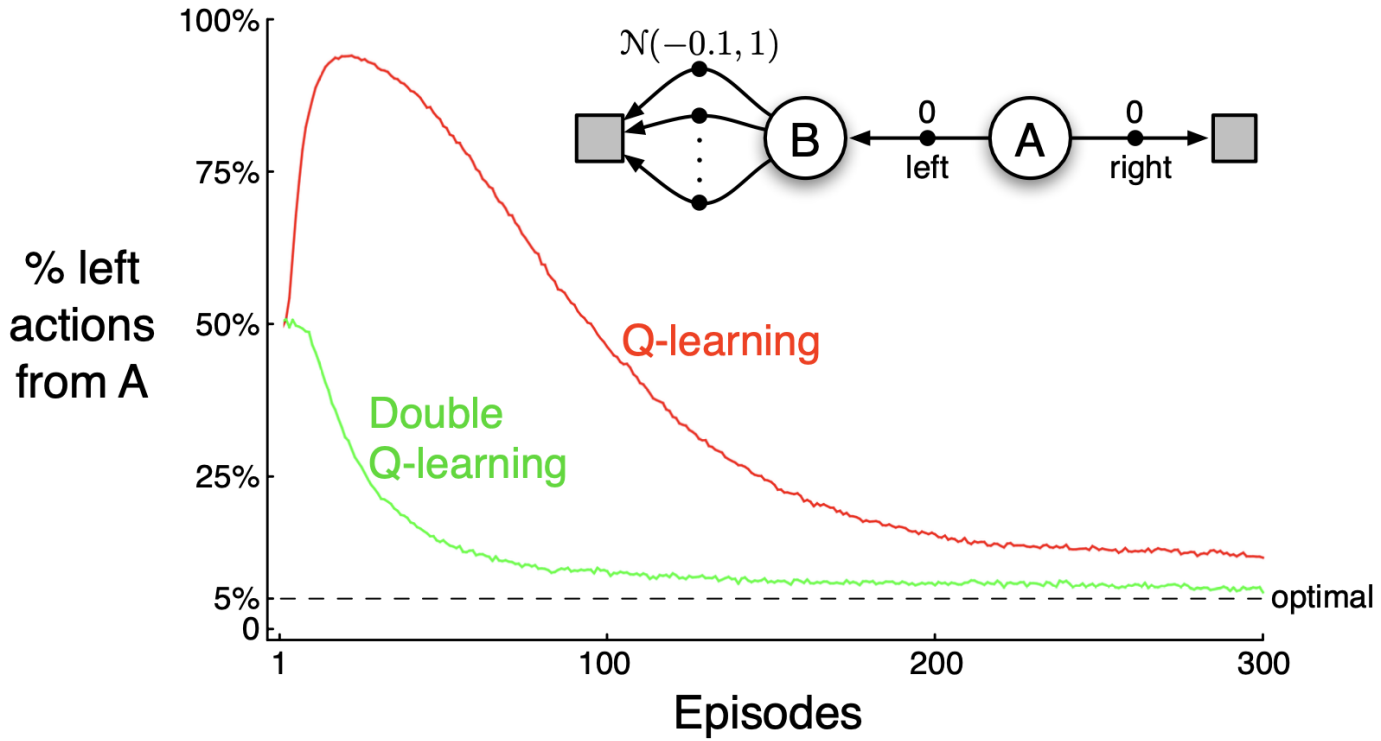


Figure 6.5: Comparison of Q-learning and Double Q-learning on a simple episodic MDP (shown inset). Q-learning initially learns to take the left action much more often than the right action, and always takes it significantly more often than the 5% minimum probability enforced by ε-greedy action selection with $\varepsilon = 0.1$. In contrast, Double Q-learning is essentially unaffected by maximization bias. These data are averaged over 10,000 runs. The initial action-value estimates were zero. Any ties in ε-greedy action selection were broken randomly.

Episode always starts in state A. The return for choosing a=right is 0, whereas for a=left the expected return is -0.1. So the optimal policy is to always choose a=right.

However, with Q(B,a) initially zero, we will update to $Q(B,a) = \alpha\, R$ were $R \sim N(-.1,1)$. Some of these R's will likely be positive for some of the a's, so some of the Q(B,a)'s will be positive. So $\max_a Q(B,a)$ will be positive. So $Q(A,\text{left}) = a[0 + \max_a Q(B,a)] > 0$, and Q(A,right) = 0. So action left will be chosen by the greedy policy.

Double Learning Theory

- Create two Q functions: $Q_1(s,a)$ and $Q_2(s,a)$ with independent errors.
  - Where the error is big for $Q_1(s,a)$ it is unlikely to be big for $Q_2(s,a)$
- Toy example:
  - Suppose for some state s, $q(s,a) = 0$ for all a, and the current estimates are $Q_1(s,a) = q(s,a) + \varepsilon_a$ and $Q_2(s,a) = q(s,a) + \delta_a$ where the ε_a's and δ_a's are independent and iid $N(0,\sigma^2)$ errors.
  - Let $a_1 = \text{argmax}_a\, Q_1(s,a)$
  - $Q_1(s,a_1) = \max_a Q_1(s,a) = \max_a \varepsilon_a >> 0$
  - But $Q_2(s,a_1) = q(s,a_1) + \delta_{a1} = \delta_{a1} \approx 0 = E[\delta_{a1}]$
  - In other words, $\max_a Q_1(s,a)$ is a biased over estimator for $\max_a q(s,a)$ whereas $Q_2(s, \text{argmax}_a\, Q_1(s,a))$ is an unbiased estimator.

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

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using the policy ε-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

Tabular RL is done!

- We will now move onto Deep Reinforcement Learning (DRL).
  - Already saw one example: the continuous bandit problem in HW
- Typically the state space will be huge:
  - Finite but astronomically large as in alpha-Go
  - Or continuous and multi-dimensional as in robotics
- Action space:
  - For some problems finite and small, like Atari and Go
  - For some problems, continuous and multidimensional, as in robotics