Temporal Difference (TD) Learning

- a combination of MC & DP ideas
  - → like MC : TD methods learn from raw experience w/o a model for the environment
  - → like DP : TD methods update estimates based on other learned estimates, w/o waiting for the final outcome in an episode (bootstrap)

- last time : MC methods (constant-α)
  - wait until the return following the visit of S_t is known; then use it as a target for $V(S_t)$

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

step size, learning rate

requires end of episode

- TD methods :
  - wait until next step : at step t+1 they form a target based not R_{t+1} and the (previous) estimate $V(S_{t+1})$

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

TD(0) update rule [special case of TD(λ)]
↳ Chapter 7 → HW

Def: TD error / Bellman error:

$$\delta_t := R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

available first at step $t+1$

– Policy Evaluation using TD-Learning:
fix a policy π, want estimate $V \approx v_\pi$

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 $V(\textit{terminal}) = 0$

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'
    $V(S) \leftarrow V(S) + \alpha\left[R + \gamma V(S') - V(S)\right]$
    $S \leftarrow S'$
  until S is terminal

→ recall :

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] \quad \leftarrow \text{MC methods}$$

$$= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s]$$

$$= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \leftarrow \text{TD methods}$$

- MC : have estimate b/c $\mathbb{E}_\pi[G_t \mid S_t = s]$ is not known; estimated from a sample

TD(λ)

- TD(0): have astimate b/c of
  i) expectation under π : $\mathbb{E}_\pi$
  ii) we use current estimate $V(S_{t+1})$ instead of true value $v_\pi(S_{t+1})$

– Example : driving home (Example 6.1.)

State	Elapsed Time (minutes)	value 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

→ as an RL problem:

reward: elapsed times on each leg of journey

$\gamma = 1$ (undiscounted)

state: actual time to go from that state

value of each state: expected time to go

MC (offline) TD (online)

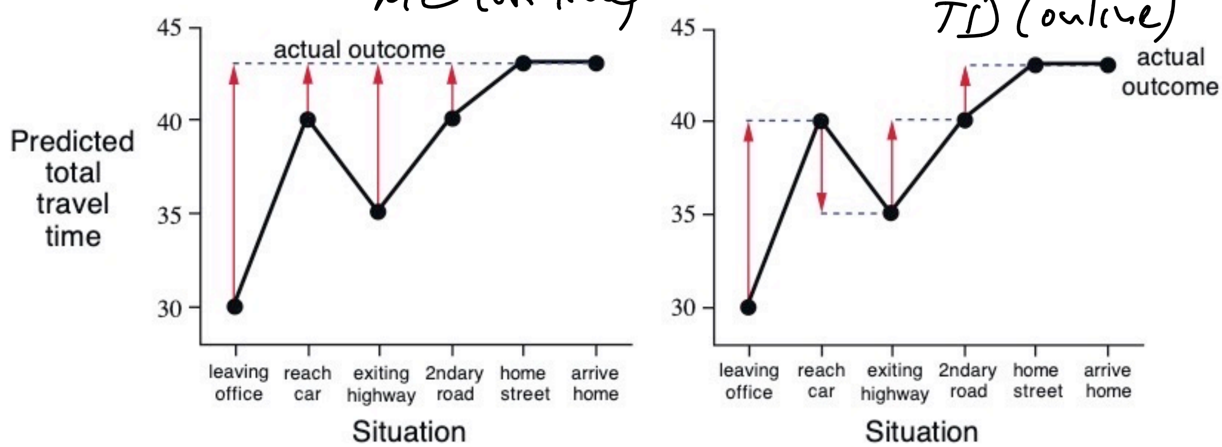


Figure 6.1: Changes recommended in the driving home example by Monte Carlo methods (left) and TD methods (right).

- advantages of TD methods:
  - over DP: do not require env. dynamics (no need for $p(s', r \mid s, a)$)
  - over MC: no need to wait until end of episode (TD is online); better for tasks with long episodes, or non-episodic tasks

<u>in practice</u>: TD methods may work better on stochastic tasks with finite data

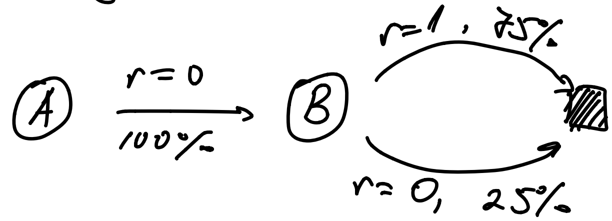
<u>Batch Updating</u> (→ important, esp. for Deep Learning)

→ suppose we have available a finite amount of experience/data (called batch)

<u>idea</u>: present to agent experience repeatedly until the algorithm converges to an answer (→ replay buffer)

<u>observation</u>: under batch updating TD & MC <u>may</u> converge to different answers

<u>Example</u>:



observed following data/batch:

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

what are the optimal predictions based on this data for $V(A)$ and $V(B)$?

-> start with state B

- $V(B) = \frac{6}{8} = \frac{3}{4}$

- $V(A) = ?$

-> MC : single data point for $A \Rightarrow V(A) = 0$

-> TD : $A \to B$ & we know that $V(B) = 3/4$

$\Rightarrow V(A) = 3/4$

note : MC gives zero training error (zero error on the training data) but TD generalizes better to future data

[temporal difference -> causality ex. B comes after A]

=> batch MC methods minimize mean-squared error on the training set

=> batch TD methods find estimates consistent with the maximum likelihood model of the underlying Markov process

TD Control / Policy Improvement

like before: trade-off exploration and exploitation

Def: Exploration - Exploitation Dilemma:

i) agent wants to exploit its knowledge
→ trying out new actions may lead occasionally to worse (w.r.t. the current) returns
→ agent wants to be greedy

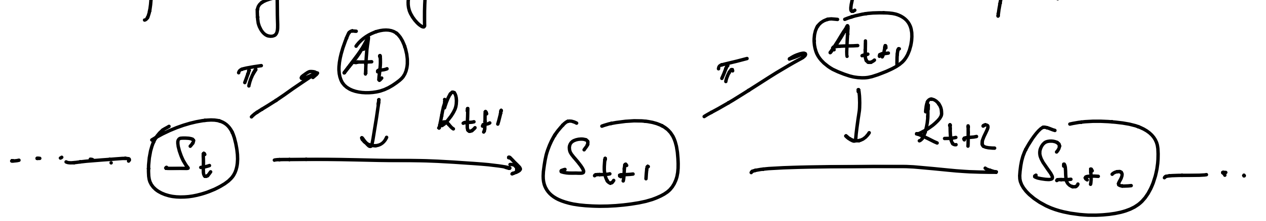
ii) agent needs to explore new actions
→ there might exist actions which result in a higher/better (wrt the current) returns

Dilemma:
- too much exploitating & agent cannot improve return
- too much exploration & agent cannot learn (lose correlations btw good & bad actions)

Def: in RL there are two classes of algorithms:
- on-policy: exploration & policy improvement are done using the same policy
- off-policy: exploration is done with a different policy (behavior policy) than the policy being learned/optimized

SARSA: on-policy TD-control

- on-policy algo to learn $q_* = q_{\pi_*}$



want to do policy evaluation for $q(s,a)$

$$Q_{new}(S_t, A_t) = Q_{old}(S_t, A_t) + \alpha\left[R_{t+1} + \gamma Q_{old}(S_{t+1}, A_{t+1}) - Q_{old}(S_t, A_t)\right]$$

α: learning rate / step size , $\alpha \in [0, 1]$

→ need quintuple: $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$

→ name of algorithm

- Bellman error:

$$\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$$

– Algorithm (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)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

- convergence to $q_*(s,a)$ is guaranteed, provided all (s,a)-pairs are visited an infinite number of times

Watkin's Q-Learning: Off-policy TD Control

– update rule:

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha\left[R_{t+1} + \gamma \max_{a \in A(S_{t+1})} Q(S_{t+1},a) - Q(S_t,A_t)\right]$$

→ learned Q-function directly approximates q_* independently of the policy being followed

→ behavior policy still has an effect: it determines which (s,a)-pairs are visited & updated

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'
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
    $S \leftarrow S'$
  until S is terminal

→ Bellman error:

$$\delta = R_{t+1} + \gamma \max_{a \in A(S_{t+1})} Q(S_{t+1},a) - Q(S_t,A_t)$$

- convergence to $q_*(s,a)$ is guaranteed, provided all (s,a)-pairs are visited an infinite number of times

Expected SARSA

- at step $t+1$: take into account how likely each action is, under the behavior policy

any policy is OK

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

$\rightarrow$ algorithm is same as Q learning

Remarks:

- here, we use the target policy π to generate the behavior (on-policy), but we can use any other policy!
  - $\Rightarrow$ expected Sarsa is off-policy!
- special case: π is the greedy policy wrt Q

$$\Rightarrow \sum_a \pi(a|S_{t+1})\, Q(S_{t+1}, a) = \max_{a \in A(S_{t+1})} Q(S_{t+1}, a)$$

$\rightarrow$ back to Q-learning

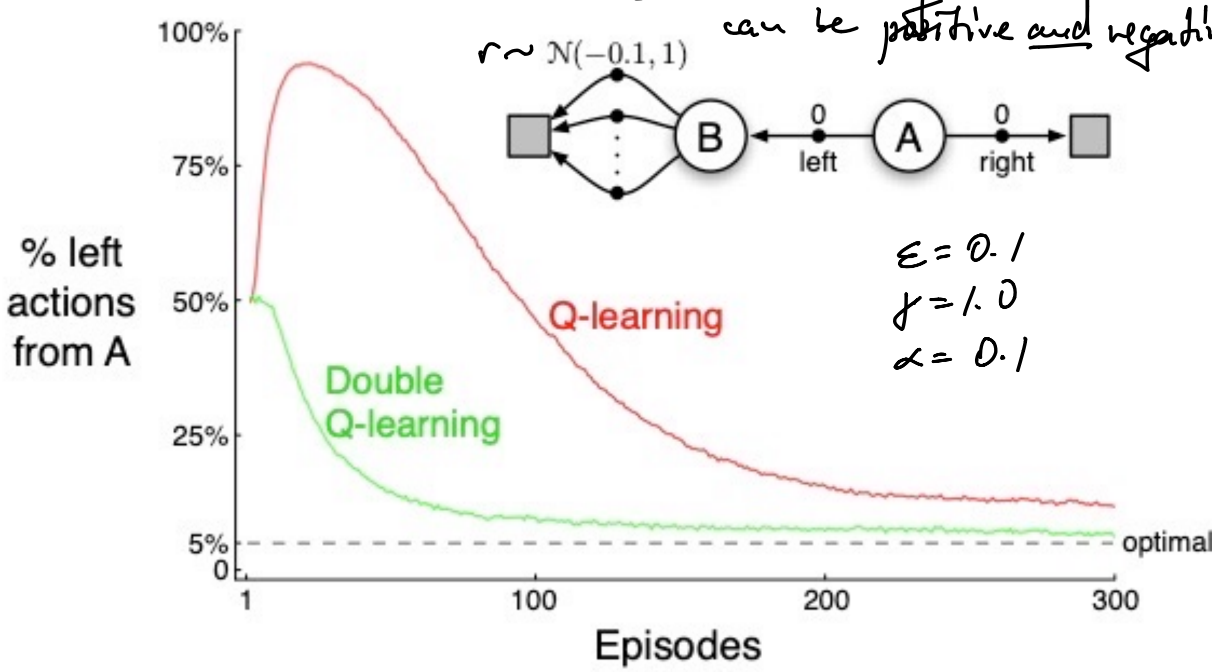
Maximization Bias & Double Learning

→ all control algorithms so far involve some maximization procedure, e.g. argmax, max, π_ε, etc....

→ can lead to a significant bias

- Example:

reward normally distr.
can be positive and negative



$\varepsilon = 0.1$

$\gamma = 1.0$

$\alpha = 0.1$

$Q(A, \text{left}) = -0.1 < 0$ (expected return starting from A)

$Q(A, \text{right}) = 0$

issue: agent may be fooled to take left b/c of some positive rewards occurring there

How do we fix this issue?

idea: we can use two indep. Q functions:

1) $A^* = \underset{a \in A}{\mathrm{argmax}}\ Q_1(a)$: Q_1 selects action

$Q_2(A^*) = Q_2(\underset{a}{\mathrm{argmax}}\ Q_1(a))$: Q_2 evaluates action

$\Rightarrow$ unbiased estimator for q_*, since
$\mathbb{E}[Q_2(A^*)] = q_*(A^*)$

2) interchange the roles of Q_1 & Q_2 in a symmetric fashion

$\Rightarrow$ <u>double learning</u> (extremely important for Deep Q-learning)

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

& $1 \Leftrightarrow 2$

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

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:
            $Q_1(S,A) \leftarrow Q_1(S,A) + \alpha\Big(R + \gamma Q_2\big(S', \operatorname{argmax}_a Q_1(S',a)\big) - Q_1(S,A)\Big)$
        else:
            $Q_2(S,A) \leftarrow Q_2(S,A) + \alpha\Big(R + \gamma Q_1\big(S', \operatorname{argmax}_a Q_2(S',a)\big) - Q_2(S,A)\Big)$
        $S \leftarrow S'$
    until S is terminal

$\rightarrow$ double Q-learning doubles the memory requirements but does not increase the CPU time/runtime.

HW : Exercise 6.13 : write down algorithm for Double Expected Sarsa

RL Alg.'s (not complete)

online
- DP
- TD
  - on-policy: Sarsa
  - off-policy: Q-learning, Expected Sarsa

offline
- MC methods
  $\rightarrow$ off-policy

Ex: ES, policy gradient (can be online)