Temporal Difference Learning

Reinforcement learning – LM Artificial Iintelligence
(2022-23)

Alberto Castellini
University of Verona

Summary

- Introduction
- TD Prediction
- Advantages of TD Prediction Methods
- Batch Updating
- Q-Learning: Off-Policy TD Control
- Expected Sarsa (hints)
- Maximization Bias and Double Learning (hints)

Introduction

Introduction

- Temporal Difference (TD) is the most central and novel idea of RL

- TD is a combination of DP and MC ideas
  - **Like MC**, TD can learn from raw experience, **without a model of the dynamics**
  - **Like DP**, TD performs **bootstrapping**, i.e., updates estimates based on other learned estimates without waiting for final outcome

- **Relationships between DP, MC and TD** is a recurring theme in RL

- These ideas blend into each other and can be **combined** (see Chapters 7 and 12 of SutBar)

- **GPI** apprach: all DP, MC and TD use it but they **differentiate** in the approach used to solve the **prediction** problem

TD Prediction

TD Prediction

- Both TD and MC use **experience** to solve the prediction problem

- Given some **experience** following policy π both methods update their estimate V of v_π for the nonterminal states S_t occurring in the experience

- **MC** methods **wait** until the **return** following the visit is known. It uses the return as a target for $V(S_t)$.

- Every-visit MC update (*contstant-* α *MC*):

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

- where G_t is the **actual return** following time t

  α is a constant step-size parameter

TD Prediction

- **TD** methods instead need to **wait** only until the **next time step**. At time *t+1* they 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 (TD(0) or one-step TD)

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

  immediately on transition to S_{t+1}.

- The **target** for the MC update is G_t
- The **target** for TD update is $R_{t+1} + \gamma V(S_{t+1})$

- TD(0) is a **special case** of TD(λ) (Ch. 12 SutBar) and n-step TD (Ch. 7 SutBar)

TD Prediction

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

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

TD Prediction

- TD(0) is a **bootstrapping** method (as DP) since it bases its update in part on an existing estimate

- **MC** vs **DP**:

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

→ Target of **MC** methods

→ Target of **DP** methods

- The **MC target** is an **estimate** because the expected value in the first equation is unknown (a sample return is used in place of the real expected return)

- The **DP target** is an **estimate** because $v_\pi(S_{t+1})$ is not known and the current estimate $V(S_{t+1})$ is used instead (the expected values are completely provided by the model of the environment)

TD Prediction

- The **TD target** is an **estimate** for **both reasons**: it samples the expected values $\mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]$ and it uses the current estimate V instead of the true v_π

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

- TD **combines** the **sampling** of MC with the **bootstrapping** of DP, trying to get the advantages of both approaches

- Backup diagram of TD(0)

- **Sample updates** (used by MC and TD) **differ** from **expected updates** (used by DP) in that they are based on a **single sample** successor rather than a **complete distribution** of all possible successors

TD Prediction

- **TD error:** the quantity in brackets in the TD(0) update

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

  is a sort of error, measuring the difference between the estimated value of S_t (i.e., $V(S_t)$) and the better estimate $R_{t+1} + \gamma V(S_{t+1})$.

- This error is based **on the estimate made at time *t*** and it is not available until one time step later → Error in $V(S_t)$ available at time *t+1*

TD Prediction

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

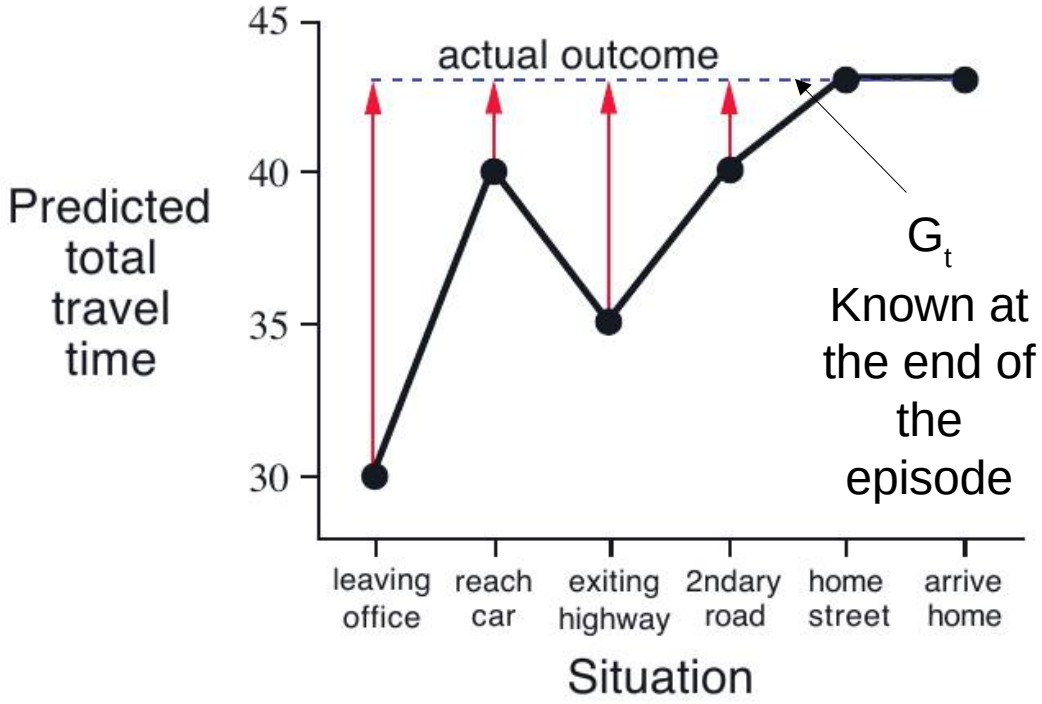
$$
\begin{aligned}
G_t - V(S_t) &= R_{t+1} + \gamma G_{t+1} - V(S_t) + \gamma V(S_{t+1}) - \gamma V(S_{t+1}) \\
&= \delta_t + \gamma\big(G_{t+1} - V(S_{t+1})\big) \\
&= \delta_t + \gamma\delta_{t+1} + \gamma^2\big(G_{t+2} - V(S_{t+2})\big) \\
&= \delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2} +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

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

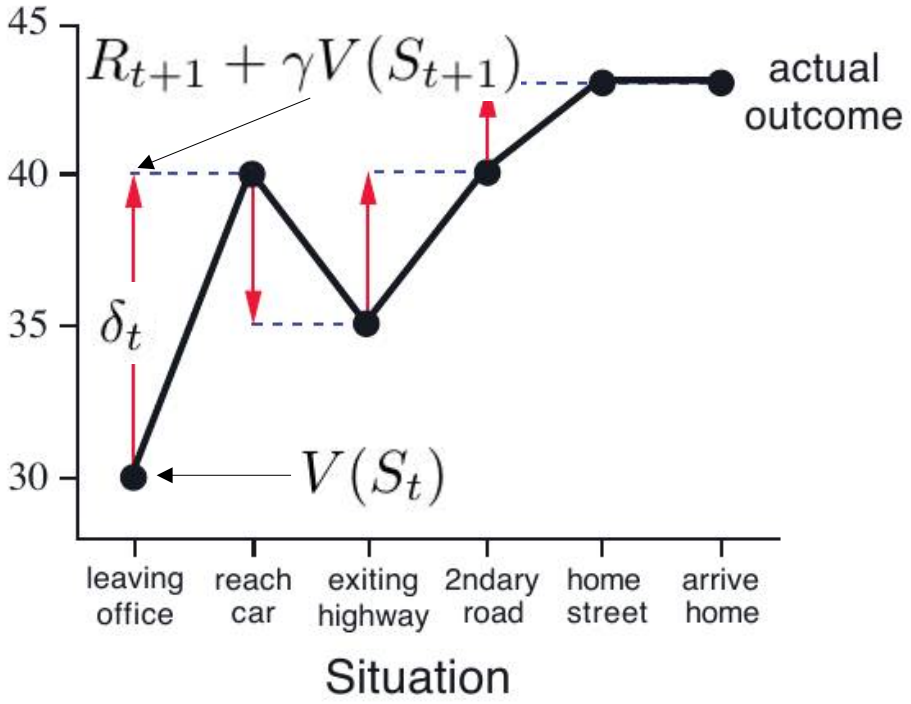
Example: Driving home

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

MC approach



TD approach



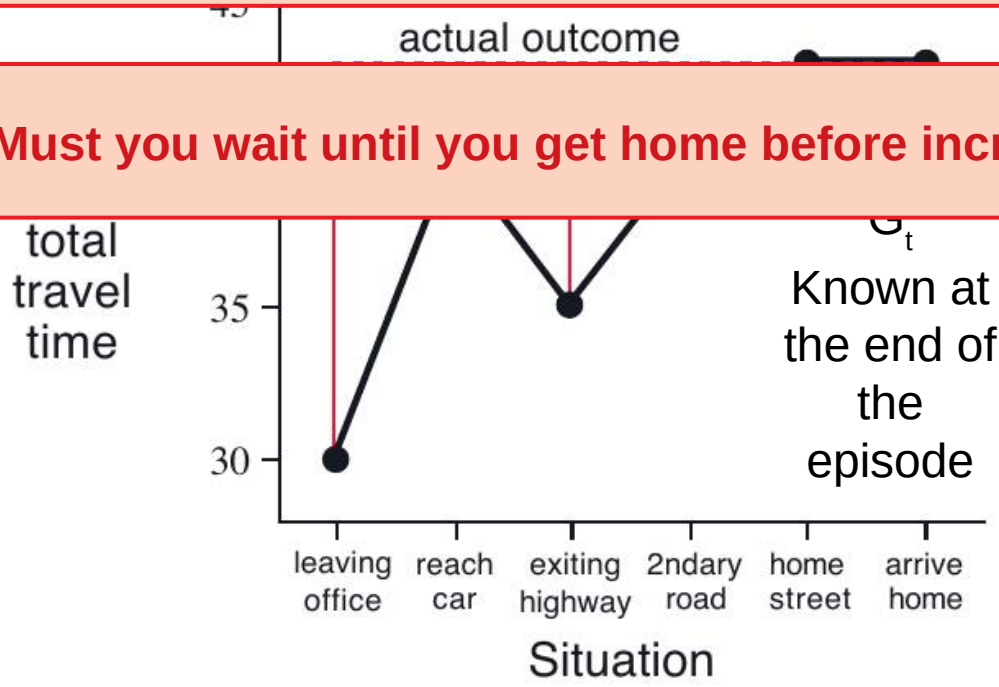
$$\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

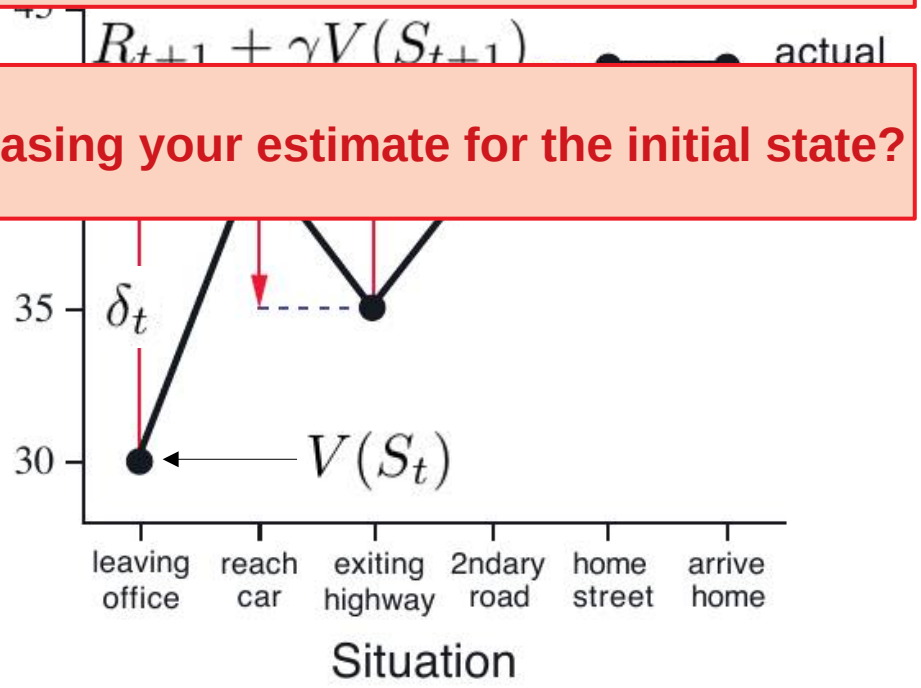
Example: Driving home

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

Is it necessary to wait until the final outcome is known before learning can begin?

Must you wait until you get home before increasing your estimate for the initial state?



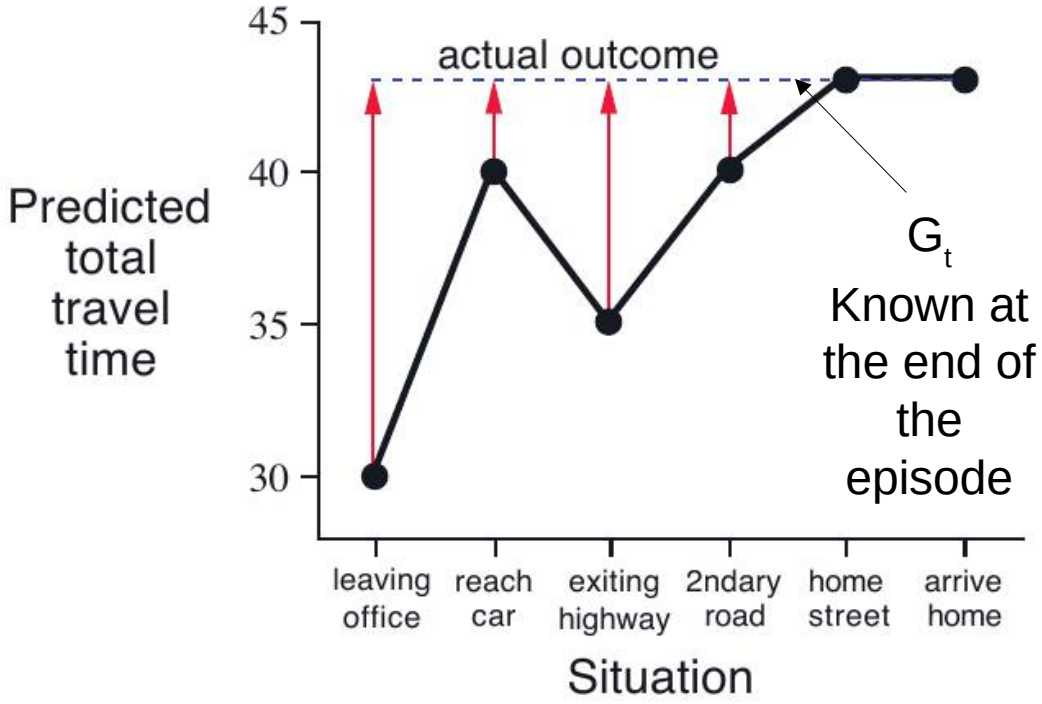


$$\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

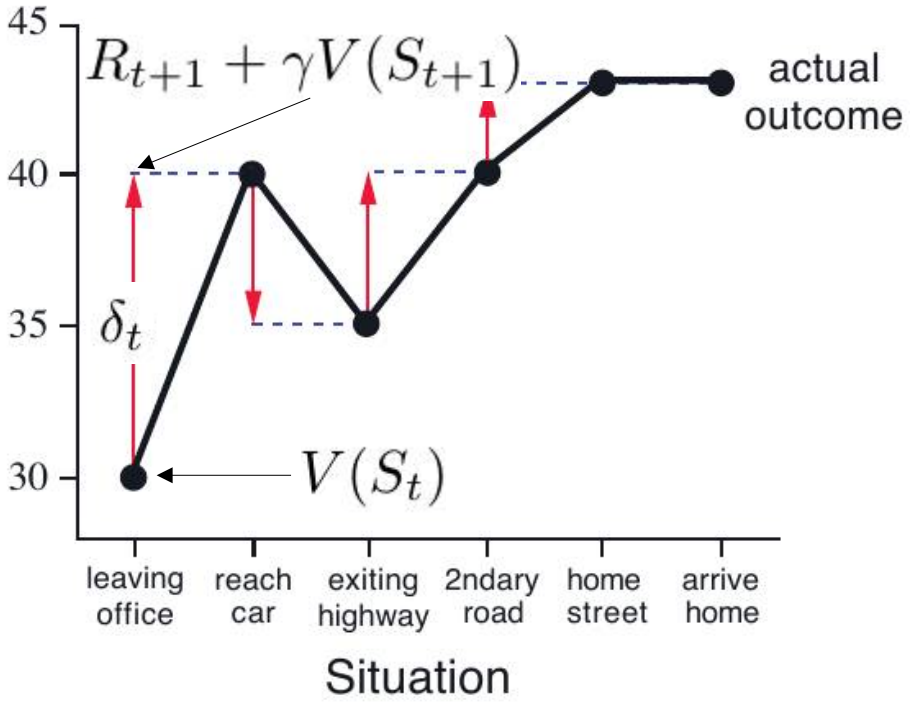
Example: Driving home

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

MC approach

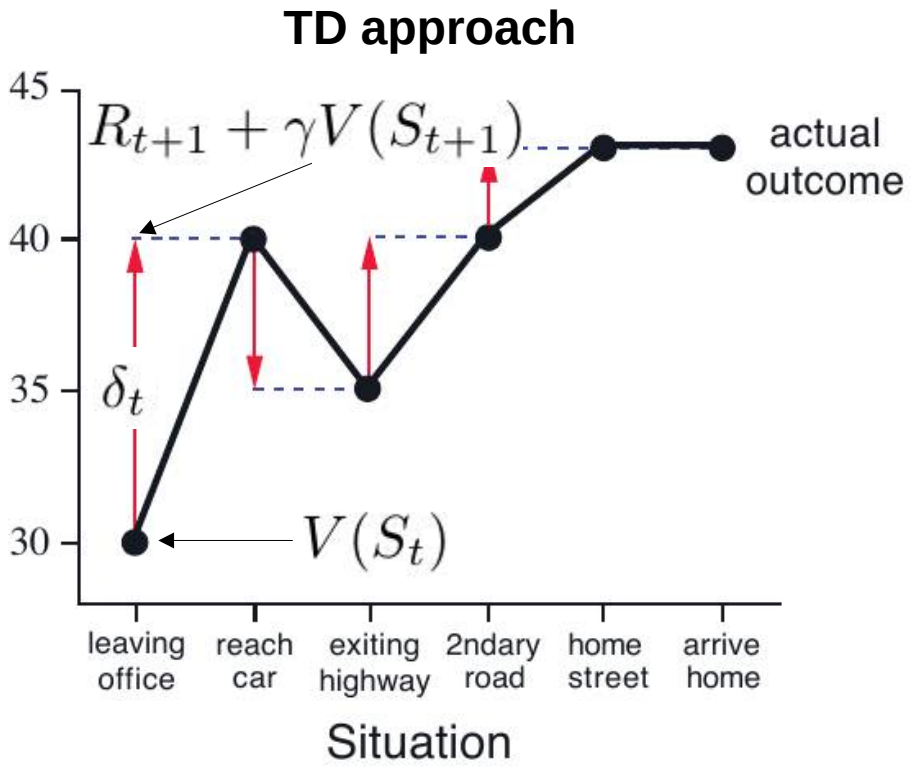


TD approach



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

Example: Driving home



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

Each error is proportional to the change over time of the prediction, i.e., to the Temporal Difference in predictions

Advantages of TD Prediction Methods

Advantages of TD Prediction Methods (over MC and DP)

- 1. **TD** methods do **not** require a **model of the environment** as DP methods

- 2. **TD** methods are naturally implemented in an **online** and fully **incremental** way, unlike MC methods
  - With **MC** methods one has to **wait** until the end of the **episode** to make updates, while **TD** methods **wait** only **one time step**
    - Some applications have very **long episodes** hence the wait is too long
    - Other applications are **continuing tasks** and have no episode at all
  - **MC** methods have **slow learning** in some conditions, while **TD** methods are **less susceptible** to these problems because they learn from each transition regardles of subsequent actions

Are TD methods sound? Convergence

For any fixed policy π, **TD(0) has been proved to converge** to v_π

- In the mean for a **constant step-size parameter** $\alpha_n = \alpha$ if it is sufficiently small;

- With probability 1 if the step-size parameter decreases according to **stochastic approximation conditions** (Sec. 2.5 SutBar)

$$\sum_{n=1}^{\infty} \alpha_n(a) = \infty \qquad \text{and} \qquad \sum_{n=1}^{\infty} \alpha_n^2(a) < \infty$$

where $\alpha_n(a)$ is the step-size parameter at the *n*-th selection of action *a*.

  - **First condition:** guarantees that the steps are large enough to overcome any initial condition or random fluctuation
  - **Second condition:** guarantees that eventually the steps become small enough to assure convergence

Are TD methods sound? Convergence

- Most convergence proofs apply only to the **table-based case** of the algorithm

- Some proofs also apply to the case of general **linear function approximations** (Ch. 9 SutBar)

Convergence speed

- Both **TD** and **MC** methods **converge asymptotically** to the correct predictions.
- **Which get first?**
- **Which uses more efficiently limited data?**

- This is an **open question**. No mathematical proof of faster convergence

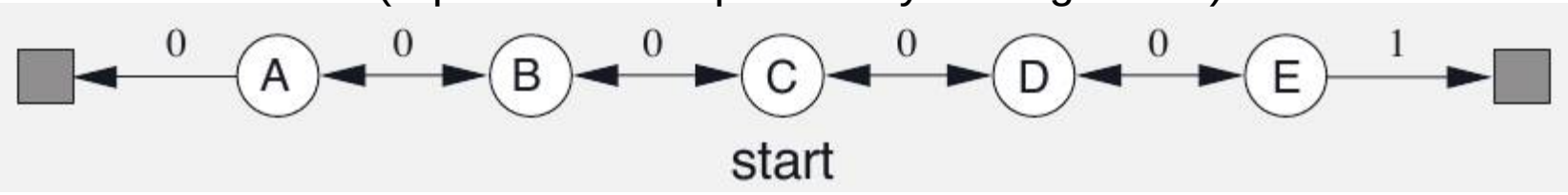
- **In practice**, **TD** methods usually **converge faster** than constant-α MC methods on stochastic tasks

Example: Random Walk

Comparison between TD(0) and constant-α MC

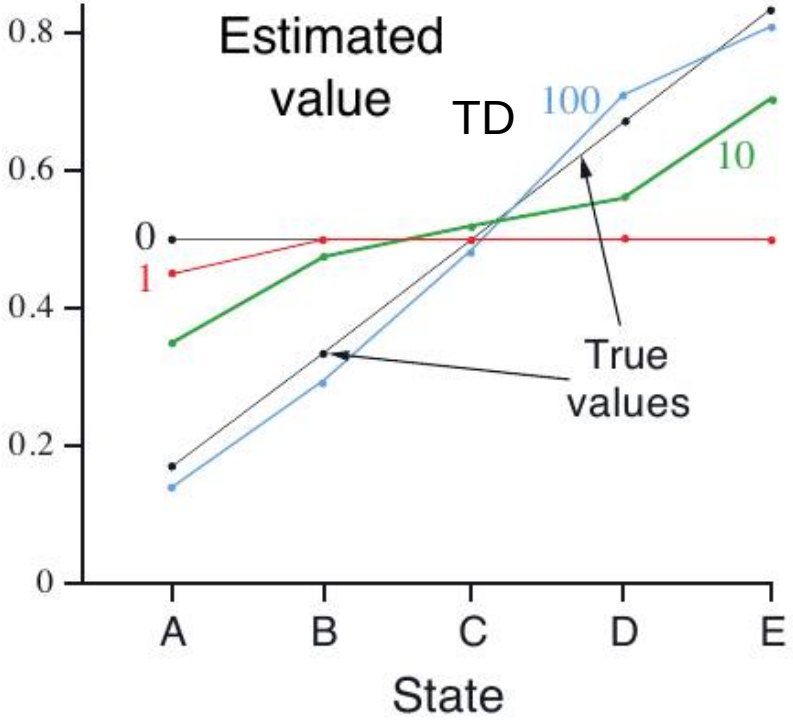
Markov Reward Process (MRP)

(equal transition probability among states)

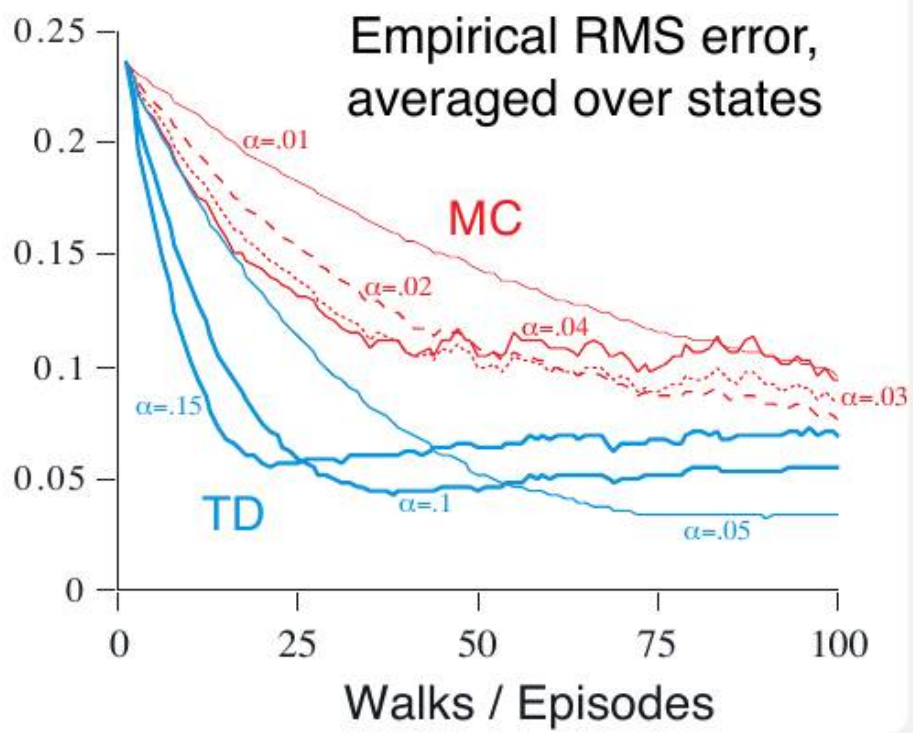


1/6 **2/6** **3/6** **4/6** **5/6** **True values**

Values learned by TD(0), α=0.1



TD(0) vs MC (100 episodes)



Batch Updating

Batch updating

- **Batch updating:** suppose a **finite amount of experience is available**, e.g., 10 or 100 episodes, a **common approach** with **incremental learning** is to **present the experience repeatedly** until **convergence**

- Given an approximate value function *V* the **increments** are computed for **every time step** *t* in which a nonterminal state is visited

  but

  the **value function** is **updated** only **once** (after processing the complete batch) by the **sum of all increments**

  then

  the experience is **processed again** with the new value function to produce a new overall increment

  until

  the value function **converges**

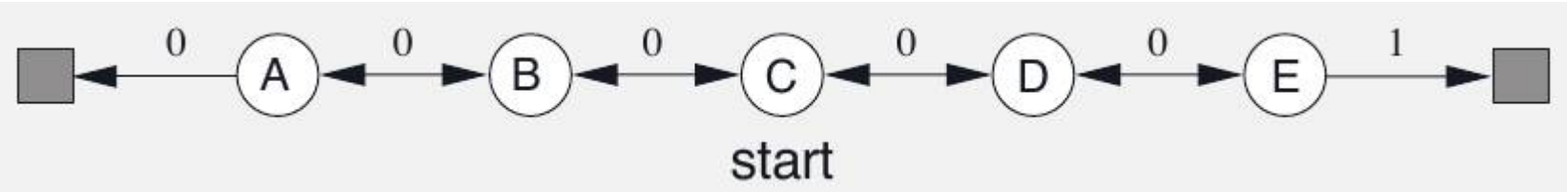
Batch updating: difference between MC and TD

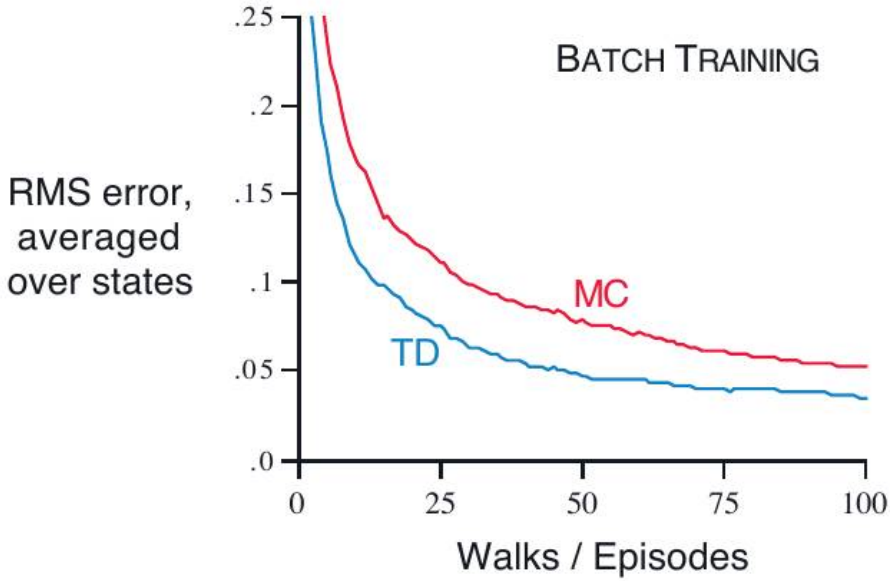
- Under batch updating **TD(0) converges** deterministically to a single answer **indipendent of the step-size parameter** α , as long as it is choosen to be sufficiently small

- The **constant-α MC** method also **converges** deterministically under the same conditions, but to a **different answer**

- **Why the answers provided by the two methods is different?** Let's understand it from two examples

Example: Random walk under batch updating

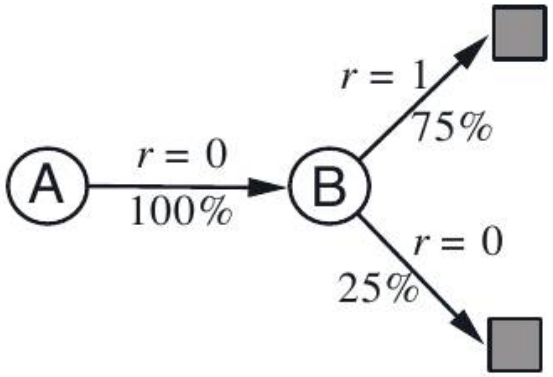




- **Batch TD was consistently better than batch MC**
- **Constant-α MC** converges to values V(S) that are sample averages of the actual returns experienced after visiting each state.
- They are **optimal estimates**, they minimize the MSE from actual returns in the training set
- **How is that batch TD performed better than this optimal method?**

Example: You are the predictor

Unknown Markov random process



Eight episodes observed

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

- **Given this batch of data, what are the best estimates for V(A) and V(B)?**
- Is the optimal value for V(B) ***3/4***?
- What is the optimal value for the estimate of V(A)?
  - **Answer 1 (MC)**: we have seen A once and the return that followed was 0 → The estimated V(A) is **0 (notice: this answer has minimum squared error on the training data!)**
  - **Answer 2 (TD)**: 100% times the process was in A it moved to B with reward 0 → The value of A is ¾.
  - This answer is given by first modeling the Markov process (left)

Batch updating: difference between MC and TD

- **If the process is Markovian, we expect that the second answer (provided by TD) will produce lower error on future data, even though the MC answer is better on the observed data**

General difference between batch TD(0) and batch MC:

- **Batch TD(0)** finds the estimate that would be exactly correct for the **maximum-likelihood model** of the Markov process
- **Batch MC** finds the estimate that **minimize mean-squared error** on the training set, without considering the Markov propery of the process

- The **Maximum-likelihood estimate** of a parameter is the parameter value whose probability of generating the data is greater. This estimate considers the **model** of the Markov process, not considered by MC

Batch updating: difference between MC and TD

- **Certainty-equivalence estimate:** it is the estimate of the parameter obtained assuming that the estimate of the underlying process was known with certainty

- **Batch TD(0) converges** to the **certainty-equivalence estimate**

- This helps esplaining why **Batch TD methods converge more quickly than Batch MC methods**

- **Nonbatch methods** do not achieve the certainty-equivalence or the minimum squared-error estimate **but** they **move roughly in these directions**.

- This **empirically** motivates why **also nonbatch TD(0) is usually faster than nonbatch constant-α MC.** At the moment nothing more definite can be said abou the relative efficiency of online TD and MC

Batch updating: difference between MC and TD

- Notice: If *n* is the number of states, then conventionally computing the **certainty-equivalence estimate** may require
  - the **order of n^2 memory**
  - the **order of n^3** computational steps

- **TD methods** can **approximate** the same solution using
  - **memory no more than order *n***
  - repeated computations over the training set

- On tasks with **large state space TD** may be the **only feasible way** of approximating the **certainty-equivalence estimate** solution

Sarsa: On-Policy TD Control

Sarsa: On-Policy TD Control

- We use **TD prediction** for the **control** problem
- According to the **GPI approch**, we use TD methods for evaluation
- As with MC we need to **trade off exploration and exploitation**
  → **On-policy** vs Off-policy methods
- We estimate an action-value function $q_\pi(s,a)$ for the policy π

Sarsa: On-Policy TD Control

- We can **adapt** the TD method for learning v_π considering transactions from **state-action** to state-action, obtaining the **update rule**:

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

  performed after every transition from a nonterminal state S_t. Notice that $Q(S_{t+1}, A_{t+1})=0$ if S_{t+1} is a terminal state

- The rule is called **Sarsa** because it uses elements S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}

- The **backup diagram** for Sarsa is the following 

- The on-policy control algorithm based on Sarsa prediction continually **estimates** q_π for the behavior policy π and at the same time **changes** π towards greedyness w.r.t. q_π

Sarsa: On-Policy TD Control

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) // Exploration policy
  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) // Exploration policy
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$ // It learns Q values of the exploration policy
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Notice: there is **no explicit representation of the policy**. It is implicitly **derived from the Q-function** (i.e., selecting the action greedly on Q)

Sarsa: On-Policy TD Control

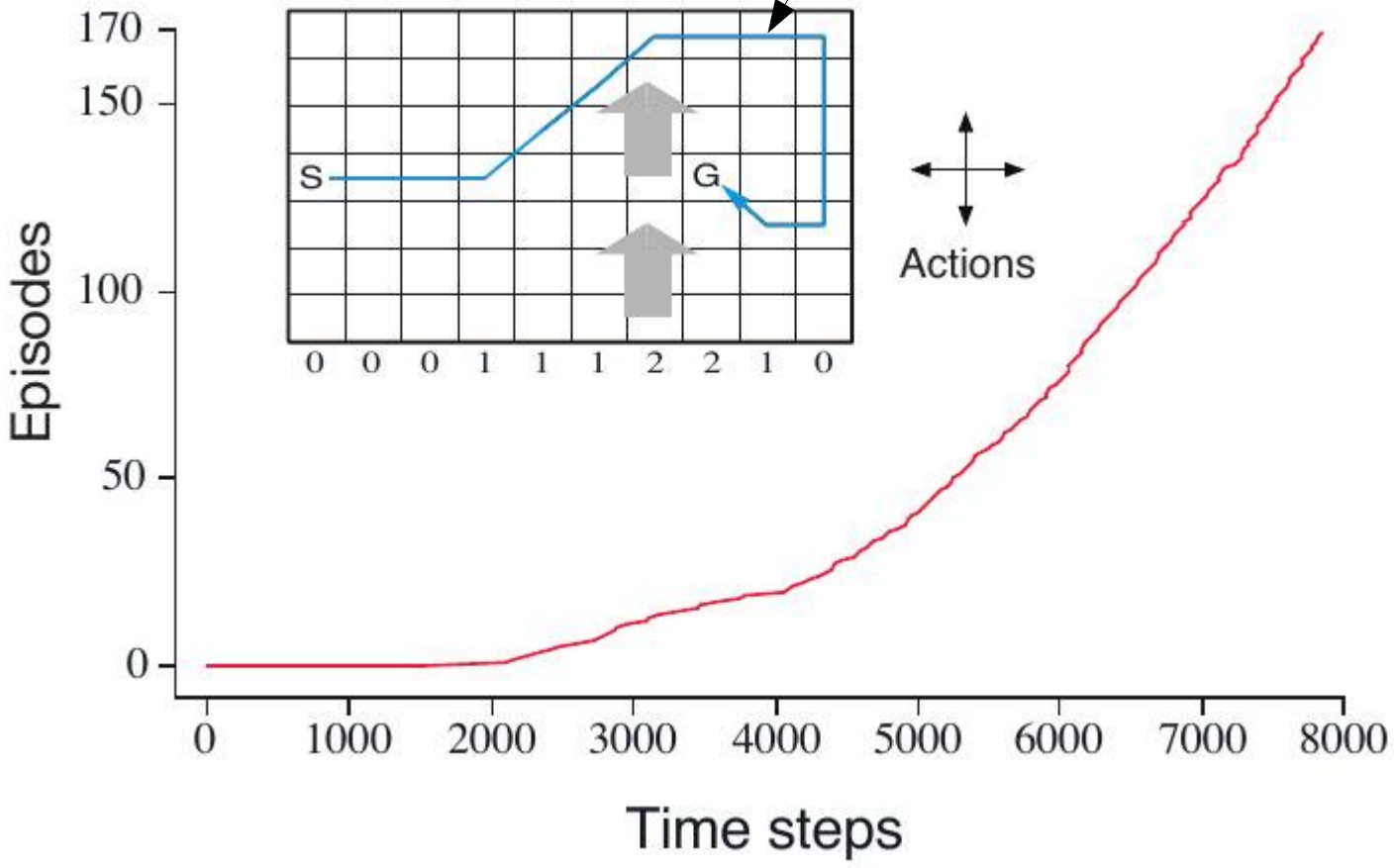
- The **convergence** properties of Sarsa depend on the **nature of the policy dependence on Q**

- Sarsa **converges** with **probability 1** to an optimal policy and action-value function as long as
  - all state-value pairs are **visited an infinite number of times**
  - the policy converges in the limit to the **greedy policy** (e.g., using ε-greedy policies with $\varepsilon = 1/t$)

Windy Gridworld

Results with ε-greedy Sarsa with ε =0.1, α =0.5

Trajectory using a policy generated after 8000 time steps (not optimal, it takes 17 steps while the optimal policy takes 15)



Termination cannot be guaranteed → MC methods cannot easily be used in this task

Q-Learning: Off-Policy TD Control

Q-Learning: Off-Policy TD Control

- Proposed by **Watkins** in **1989**, **Q-learning** is one of the early breakthroughs in RL, defined by the update rule

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

- The **learned action-value function Q directly approximate q_***, the optimal action-value function, **indipendent of the policy being followed (this makes the method off-policy).** The policy however determines which state-action pairs are visited and updated

- This strategy simplifies the analysis of the algorithm and **enables early convergence proofs**

- For correct convergence it is only **required** that the policy followed ensures **all pairs continue to be updated** (as in any other method).

- Under this assumption and a variant of **stochastic approximation conditions** on step-size, Q is shown to **converge with prob. 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., ε-greedy) // Exploration policy

    Take action A, observe R, S'

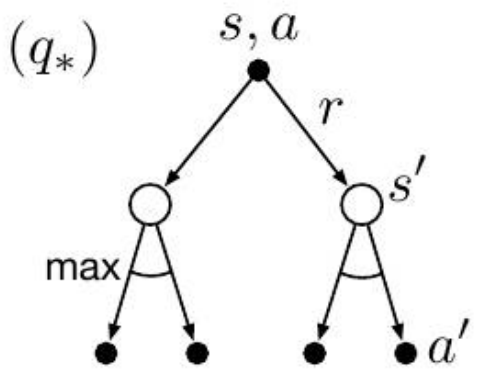
    $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$ // It learns Q values of policy q_* (not exploratory policy)

    $S \leftarrow S'$

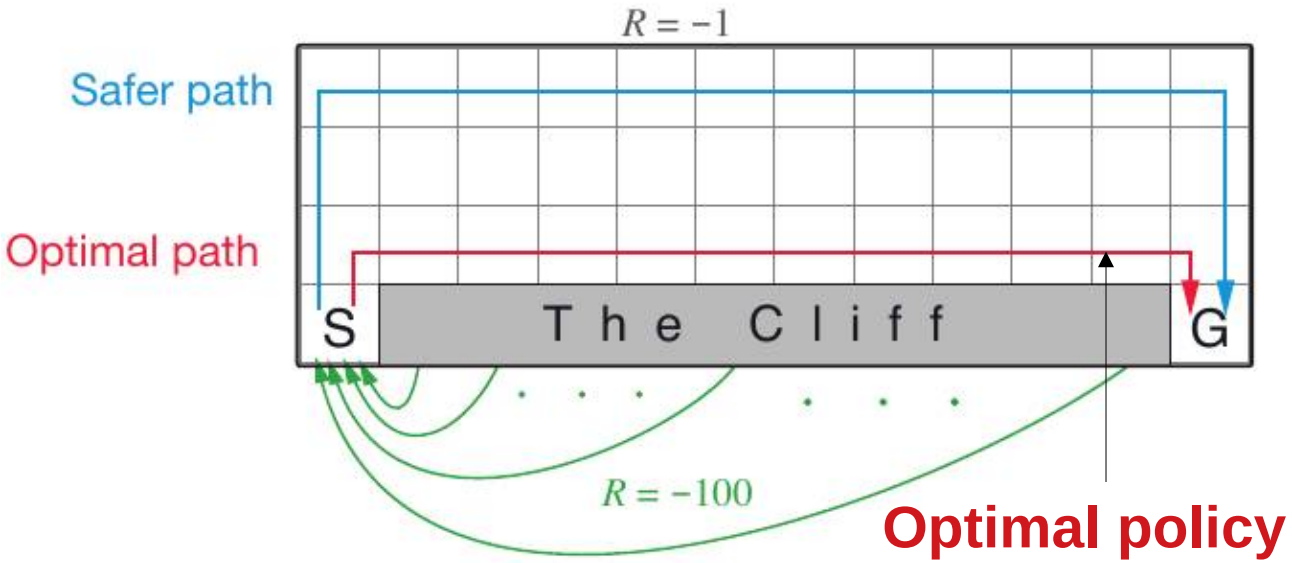
  until S is terminal

Notice: a is not selected by the ε-greedy policy as in Sarsa

The backup diagram for Q-learning is reported in the following:



Cliff Walking (comparing Sarsa and Q-learning)

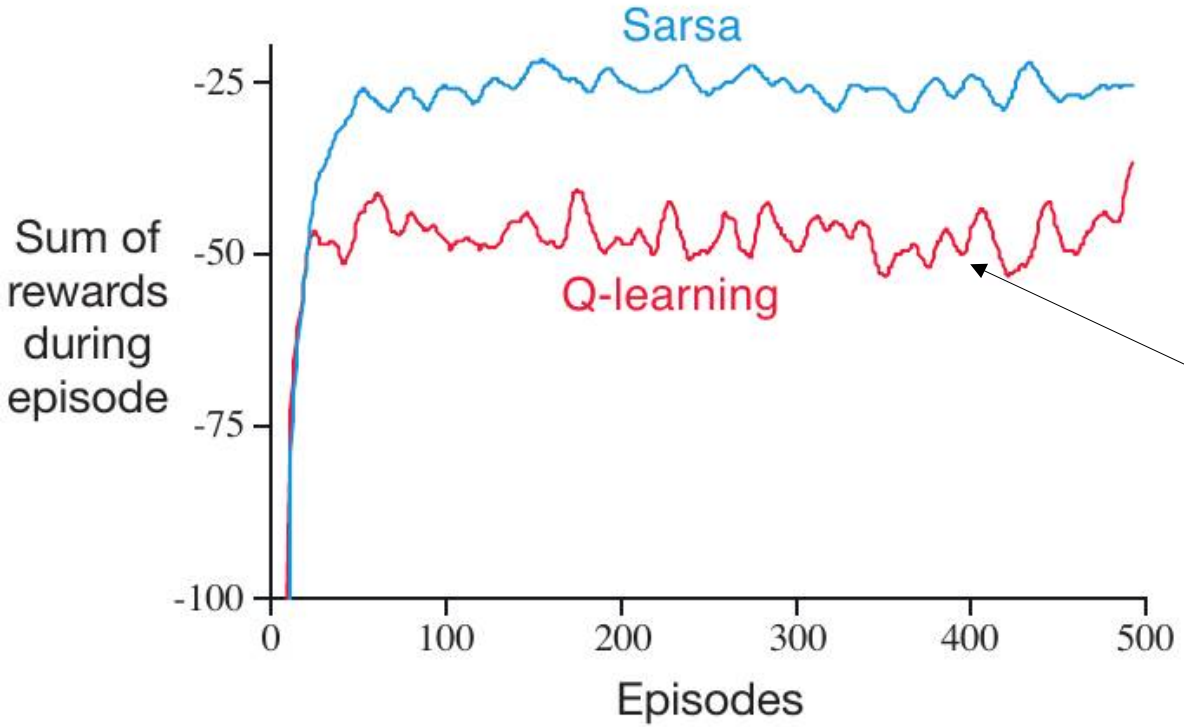


Optimal policy

Actions: up, down, left, right

Reward:

- -1 on all transitions
- -100 stepping into the cliff



Sarsa and Q-learning used ε-greedy action selection with $\varepsilon = 0.1$

Worse online performance

If ε is gradually reduced than both methods converge to q_*

Expected Sarsa

Expected Sarsa

- Consider **Q-learning** with **expected value** instead of the maximum over next state-action pairs

- **Update rule**:

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

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

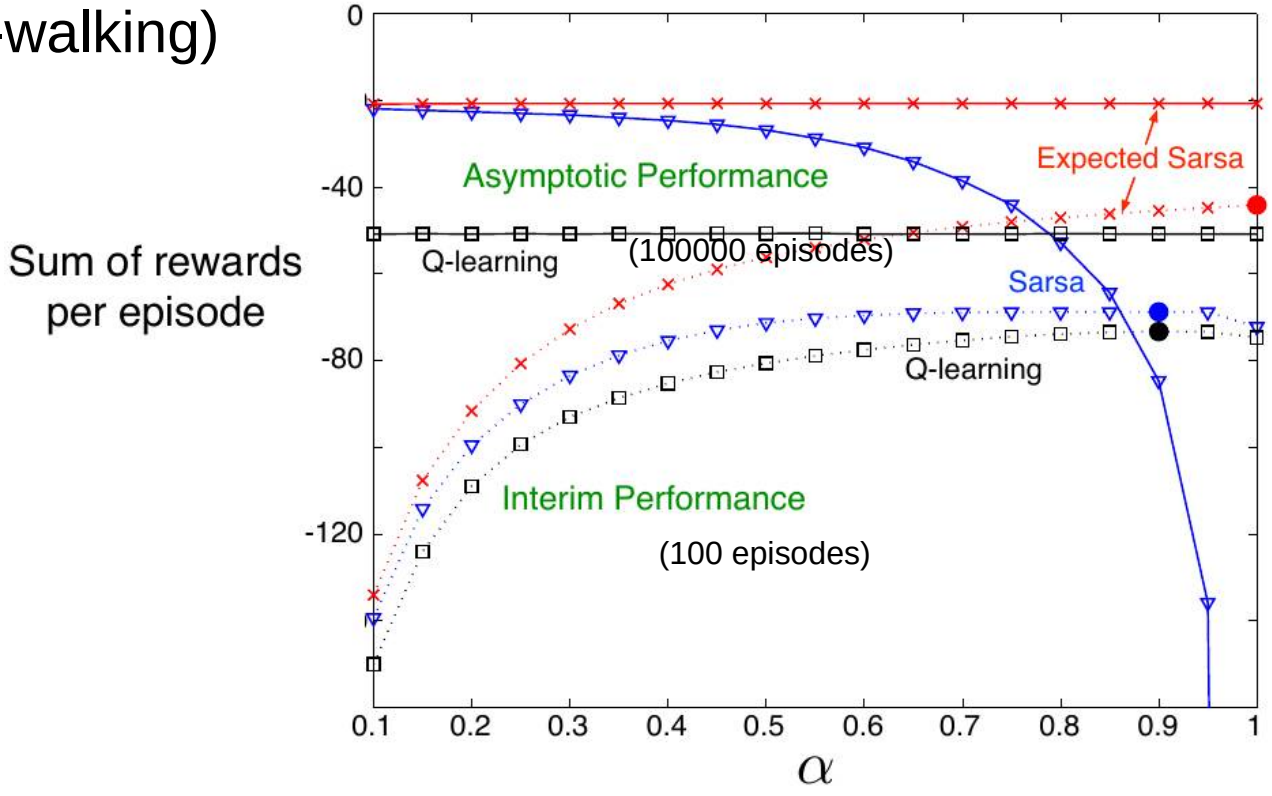
- The **backup diagrams** of Q-learning and Expected Sarsa are compared here:

Expected Sarsa

- **Advantage**: expected Sarsa **eliminates the variance** due to the random selection of A_{t+1} made by Sarsa

- **Disadvantage**: expected Sarsa is **computationally more complex** than Sarsa

- Given the same amount of experience we might expect **Expected Sarsa to perform better than Sarsa** and **Q-learning** (see results for cliff-walking)



Expected Sarsa

- Expected Sarsa can be used both **on-policy** and **off-policy**. In the second case a policy different from the target policy π is used to generate the behavior

- E.g., if π is **greedy** and behavior is **exploratory** then Expected Sarsa is exactly **Q-learning**

- **Expected Sarsa subsumes and generalizes Q-learning while reliably improving over Sarsa**

Maximization Bias and Double Learning

Maximization Bias and Double Learning

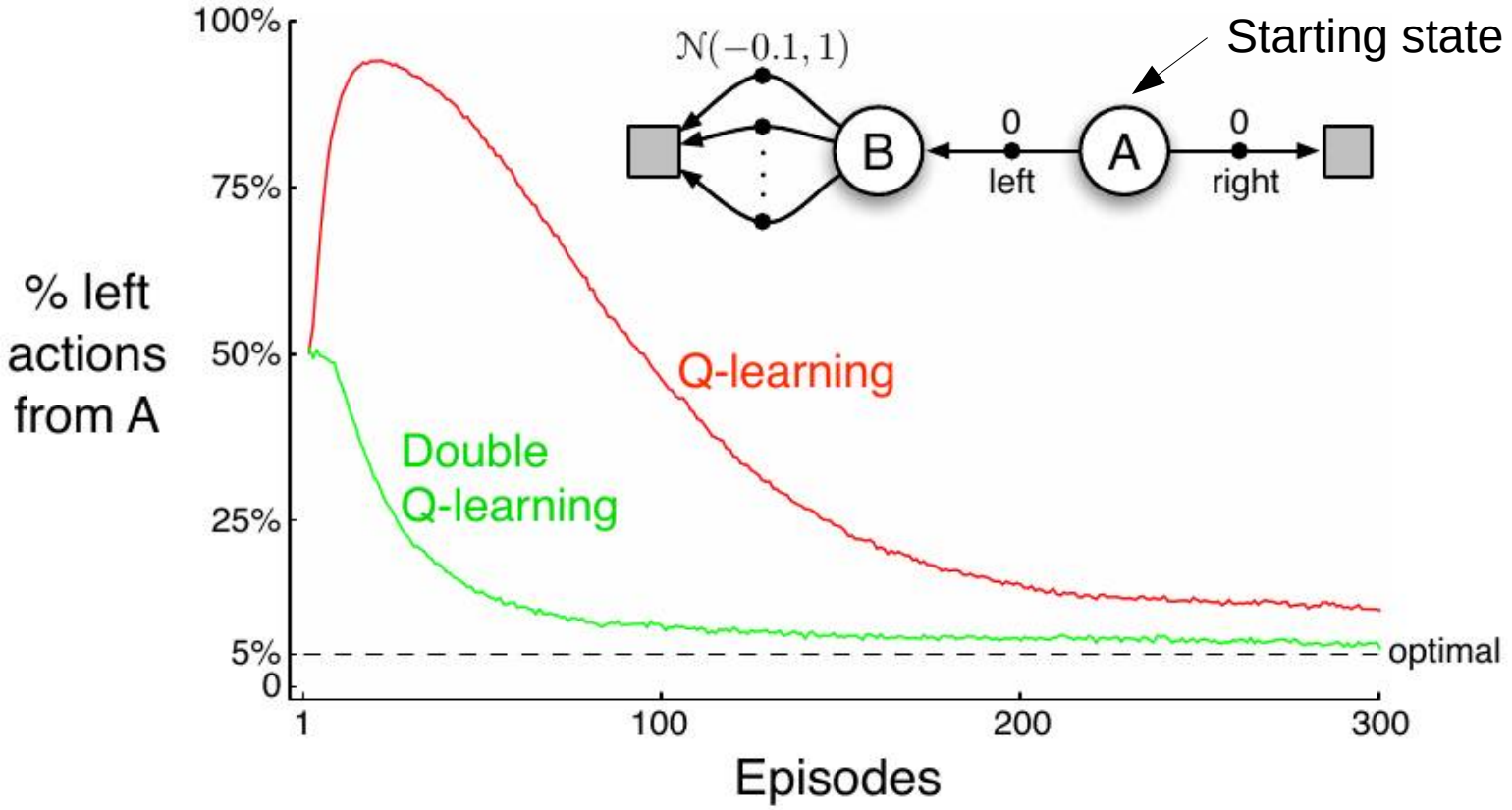
- **Control algorithms involve maximization** in the construction of the target policy (e.g., max in Q-learning, ε -greedy in Sarsa)

- **Problem:** a **maximum over estimated values** is used implicitly as an **estimate of the maximum value**

- This can lead to a positive bias called ***maximization bias***

- **Example:** state *s*, actions with true values *q(s,a)=0*, estimated values *Q(s,a)* with uncertainty (some above and some below zero)
  - The maximum of the true values q(s,a) is 0
  - The maximum of the estimates is positive → **Positive bias**

Maximization Bias Example



- Expected return for trajectories starting from left: -0.1
- Best policy: always select right from state A
- **Problem: TD methods may favor the left action from state A because of the maximization bias making B appear to have positive value**

Maximization Bias and Double Learning

- **How to avoid maximization bias?**
- Consider the actions connected to state B in the previous example
- Suppose we **divide the plays of each actions in two sets** and obtain **two independent estimates of *q(B,a)* for each action *a***, namely, $Q_1(B,a)$ and $Q_2(B,a)$ for all *a*.
- We can then use one estimate (e.g., $Q_1(B,a)$) to determine the maximizing action $A^* = \text{argmax}_a Q_1(B,a)$ and the other (e.g., $Q_2(B,a)$) to provide an estimate of its value $Q_2(B,A^*) = Q_2(B,\text{argmax}_a Q_1(B,a))$
- **This estimate is unbiased,** namely, $E[Q_2(B,A^*)] = q(B,A^*)$
- This is the idea of **Double Learning**

Maximization Bias and Double Learning

- **Obs 1:** we can also repeat the process with the role of the two estimetes reversed to obtain a secon unbiased estimate

- **Obs 2:** Although we learn two estimates, only one estimate is updated on each play

- **Obs 3:** Double learning doubles the memory requirements, but it does not increase the amount of computation per step.

Maximization Bias and Double Learning

- The double learning analogous to Q-learning is called **Double Q-learning**.
  - It divides the time steps in two, flipping a coin on each step
  - If the coin comes up heads, the update is

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma Q_2\big(S_{t+1}, \arg\max_a Q_1(S_{t+1}, a)\big) - Q_1(S_t, A_t) \Big]$$

  - 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 behavior policy can use both action-values estimates

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

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:

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

    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

- There exist also double versions of Sarsa and Expected Sarsa

Summary

- We introduced Temporal-Difference learning (**TD**)
- TD is an **alternative to MC** for solving the prediction problem in GPI

- **On-policy** TD: Sarsa
- **Off-policy** TD: Q-learning, Expected Sarsa

- Third way to use TD in control: **Actor-Critic methods** (Ch. 13 SutBar). They explicitly represent also the policy which is instead inferred from the value functions in standard TD

- TD methods are the **most widely used** RL methods

- In this lecture we analyzed **one-step, tabular, model-free** TD methods
  - Ch 7: **n-steps** (link to MC methods that perform all episode steps)
  - Ch 8: **model-based** RL methods (link to planning)
  - Part II: **function approximation** (link to deep RL)

References

- R. S. Sutton, A. G. Barto. Reinforcement learning, An Introduction. Second edition. Chapter 6