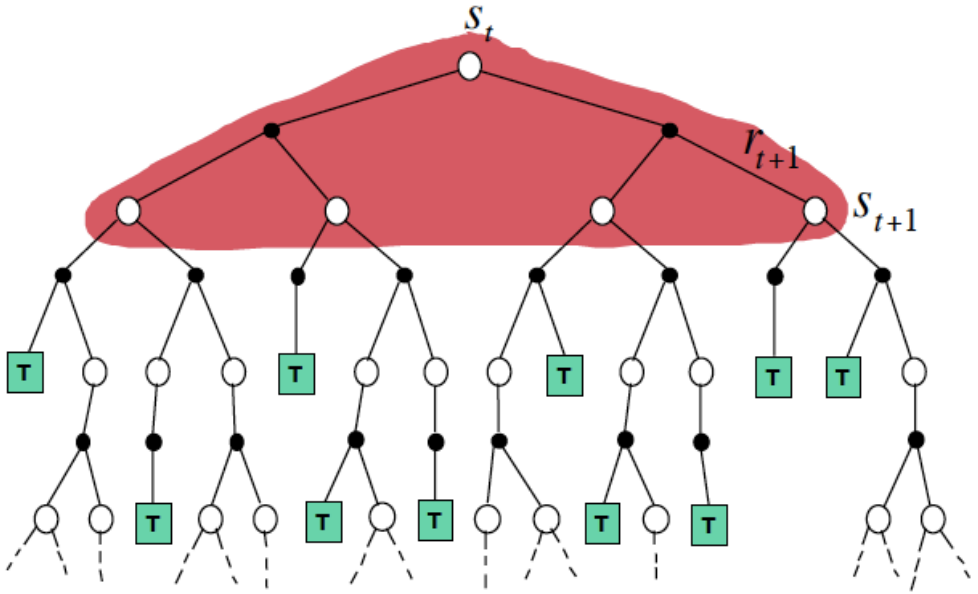
Model Free Control

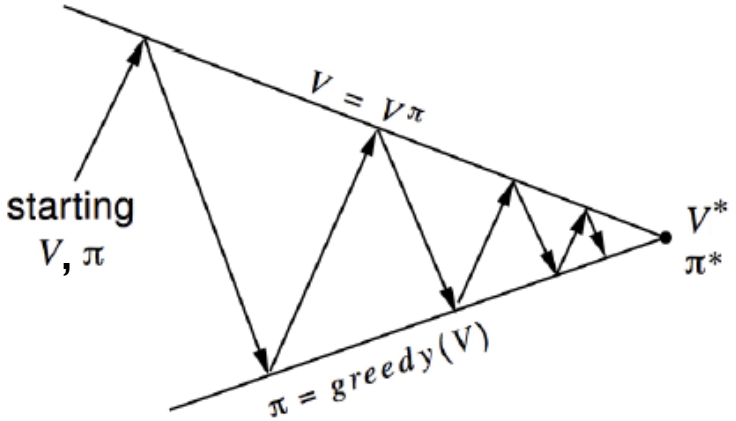
Lecture 5

Subir Varma

Model Based Policy Evaluation and Optimal Control (Lecture 3)

Model Based
Full Sweep
Full Backup





$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

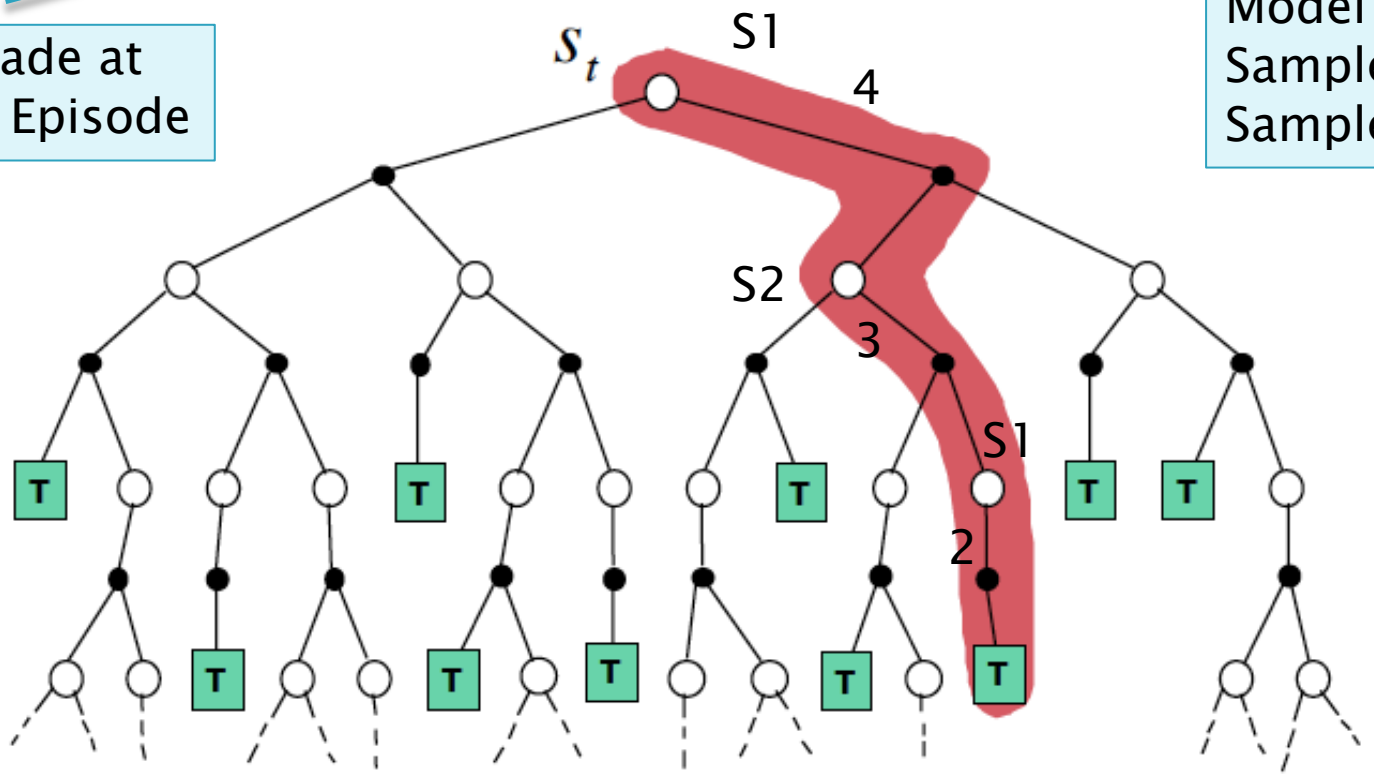
Full Policy Iteration
Generalized Policy Iteration

Model Free Policy Evaluation: Monte Carlo Learning (Lecture 4)

$$V(S_t) \leftarrow V(S_t) + \alpha\,(G_t - V(S_t))$$

Update made at end of an Episode

Model Free
Sample Sweep
Sample Backup

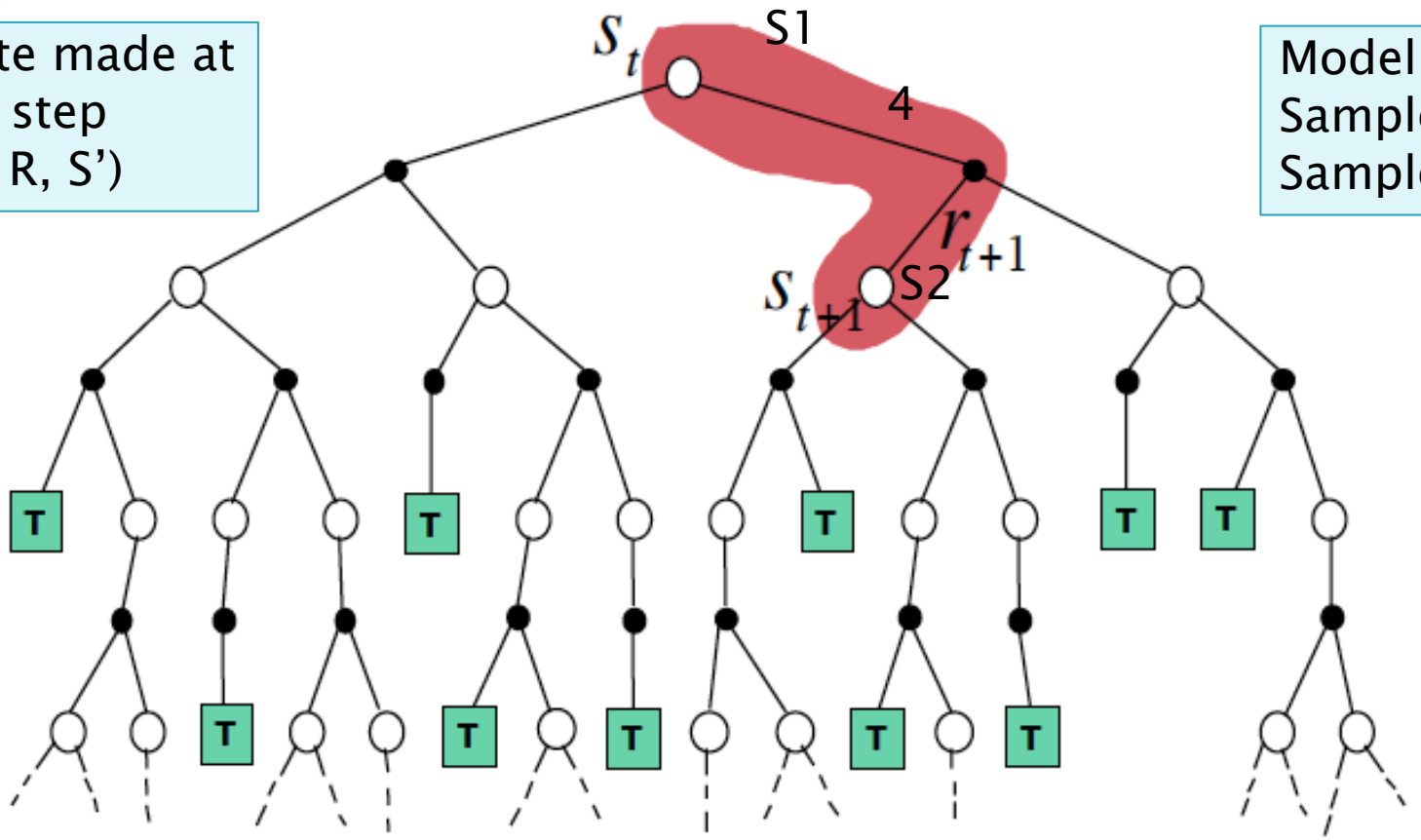


Instead of a Model, we now have sample episodes from the MDP

Model Free Policy Evaluation: Temporal-Difference (TD) Learning (Lecture 4)

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

Update made at every step
(S, A, R, S')



Model Free
Sample Sweep
Sample Backup

Instead of a Model, we now have sample episodes from the MDP

The Next Step..

So Far: We have algorithms to find the Value Function v_π, given a policy π (with or without a model)

But: We are really interested in finding the Optimal Policy π_*

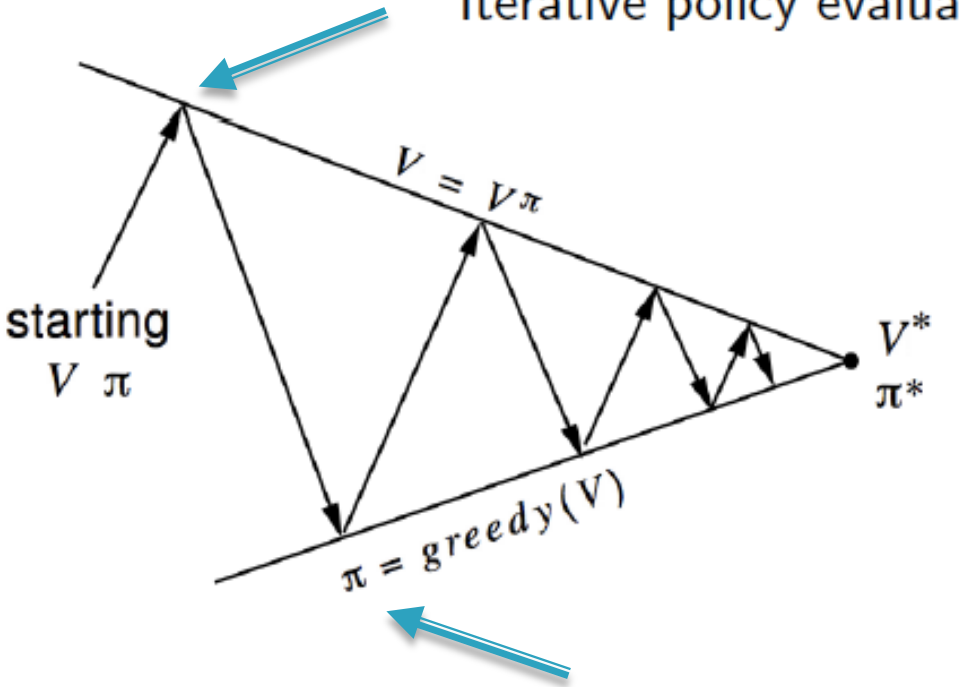
Model Free Reinforcement Learning

- Last lecture:
  - Model-free prediction
  - *Estimate* the value function of an *unknown* MDP
- This lecture:
  - Model-free control
  - *Optimise* the value function of an *unknown* MDP

From Lecture 3: Finding the Optimal Policy for the Model Based Case with Policy Iteration

Policy evaluation Estimate v_π

Iterative policy evaluation



This process is guaranteed to converge to optimal Value Function V* and thus the optimal policy

Policy improvement Generate $\pi' \geq \pi$

Greedy policy improvement

Will Policy Iteration work for the Model Free case?

Uses of Model-Free Control

Some example problems that can be modelled as MDPs

- Elevator
- Parallel Parking
- Ship Steering
- Bioreactor
- Helicopter
- Aeroplane Logistics
- Robocup Soccer
- Quake
- Portfolio management
- Protein Folding
- Robot walking
- Game of Go

❑ RL based on Human Feedback (RLHF)

For most of these problems, either:

- MDP model is unknown, but experience can be sampled
- MDP model is known, but is too big to use, except by samples

Model-free control can solve these problems

On and Off-Policy Learning

- On-policy learning
  - "Learn on the job"
  - Learn about policy π from experience sampled from π

Policy being used to generate episode is the same as the policy being learnt

- Off-policy learning
  - "Look over someone's shoulder"
  - Learn about policy π from experience sampled from μ

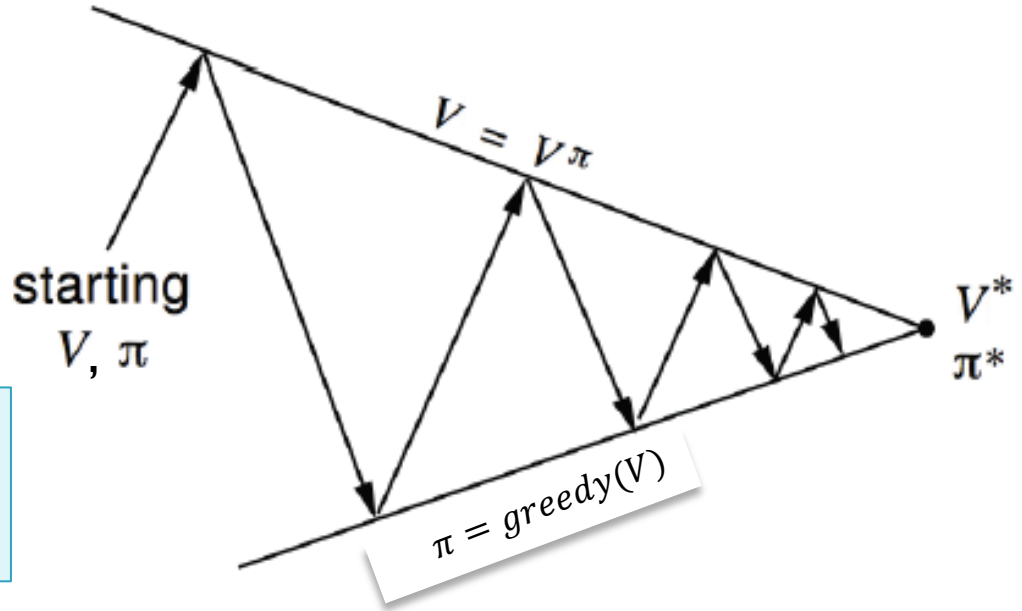
Policy being used to generate episode is the different than the policy being learnt

Monte Carlo Control

Generalized Policy Iteration

We computed v_π by using the Bellman Expectation Equation

Can we replace v_π by V_π obtained with Monte Carlo Evaluation?



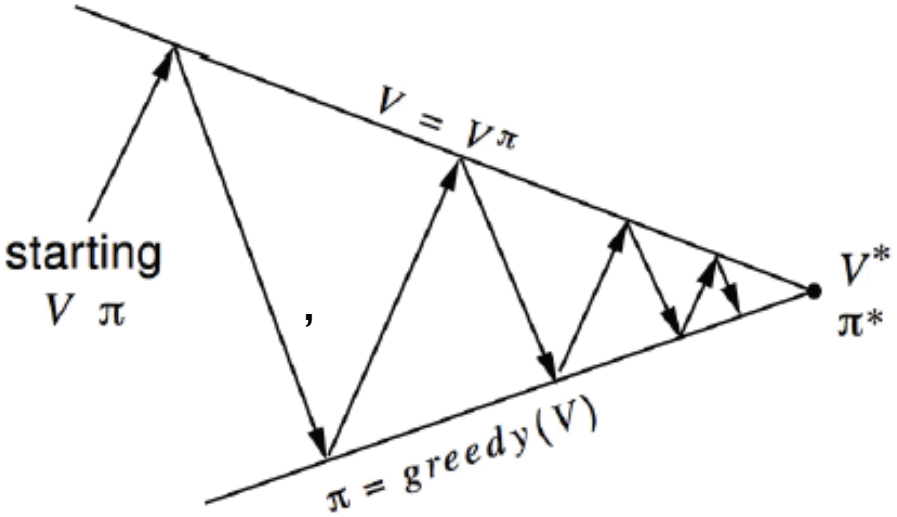
Policy evaluation Estimate V_π

e.g. Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$

e.g. Greedy policy improvement

Generalized Policy Iteration with Monte Carlo Evaluation



Instead of using Bellman Expectation Equation to compute v_π, we are using Monte Carlo Policy evaluation to estimate V_π

Policy evaluation Monte-Carlo policy evaluation, $V = v_\pi$?

Policy improvement Greedy policy improvement?

What is wrong with this approach?

Model-Free Policy Iteration Using Action-Value Function

- Greedy policy improvement over $V(s)$ requires model of MDP

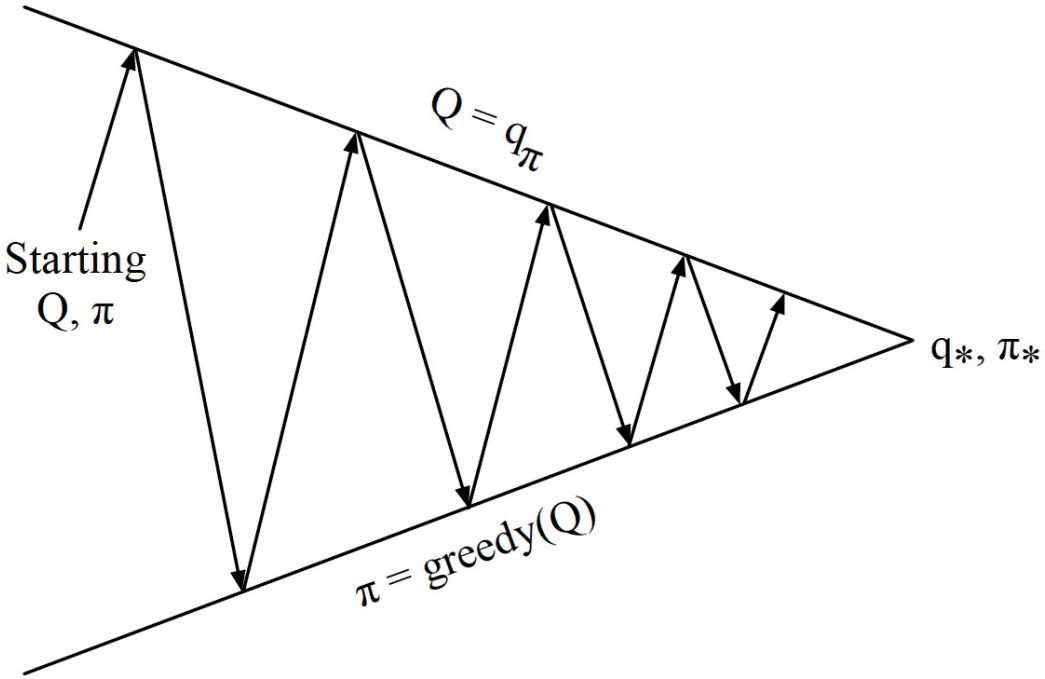
$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \left[\mathcal{R}_s^a + \sum \mathcal{P}_{ss'}^a V(s') \right]$$

- Greedy policy improvement over $Q(s, a)$ is model-free

$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ Q(s, a)$$

Estimating V_π(s) is not enough, we need to estimate Q_π(s,a)

Generalized Policy Iteration with Action-Value Function



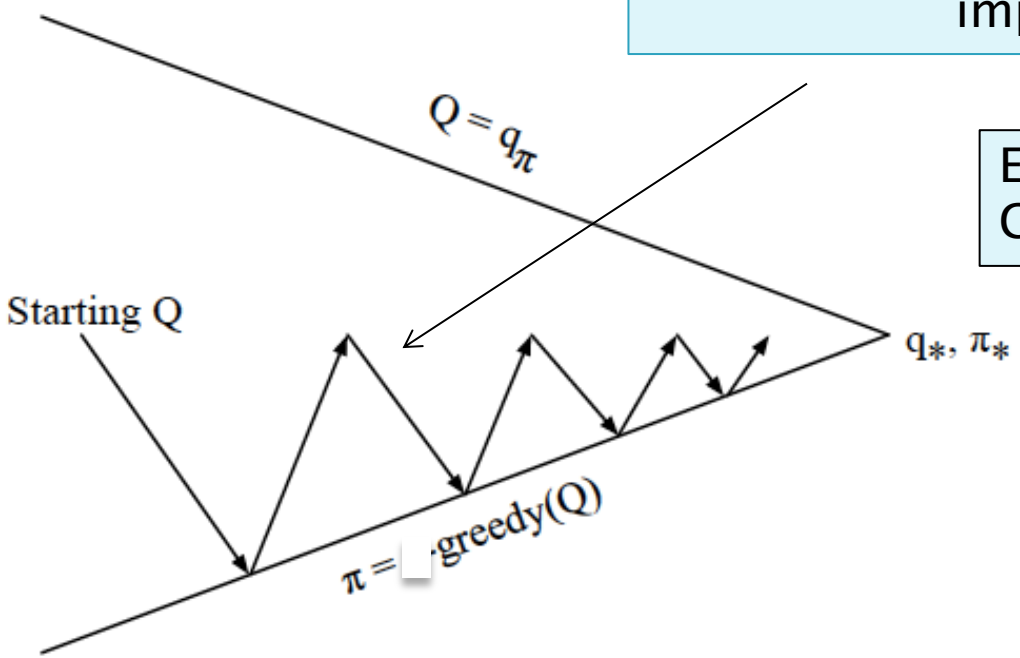
Evaluate new action VF Q_π(s,a)
Compute new policy

Policy evaluation Monte-Carlo policy evaluation, $Q = q_\pi$

Policy improvement Greedy policy improvement?

Generalized Policy Iteration with Action-Value Function

Run a single episode, and then immediately improve the Policy!



Evaluate new action VF Q_π(s,a)
Compute new policy

Every episode:

Policy evaluation Monte-Carlo policy evaluation, $Q \approx q_\pi$

Policy improvement -greedy policy improvement

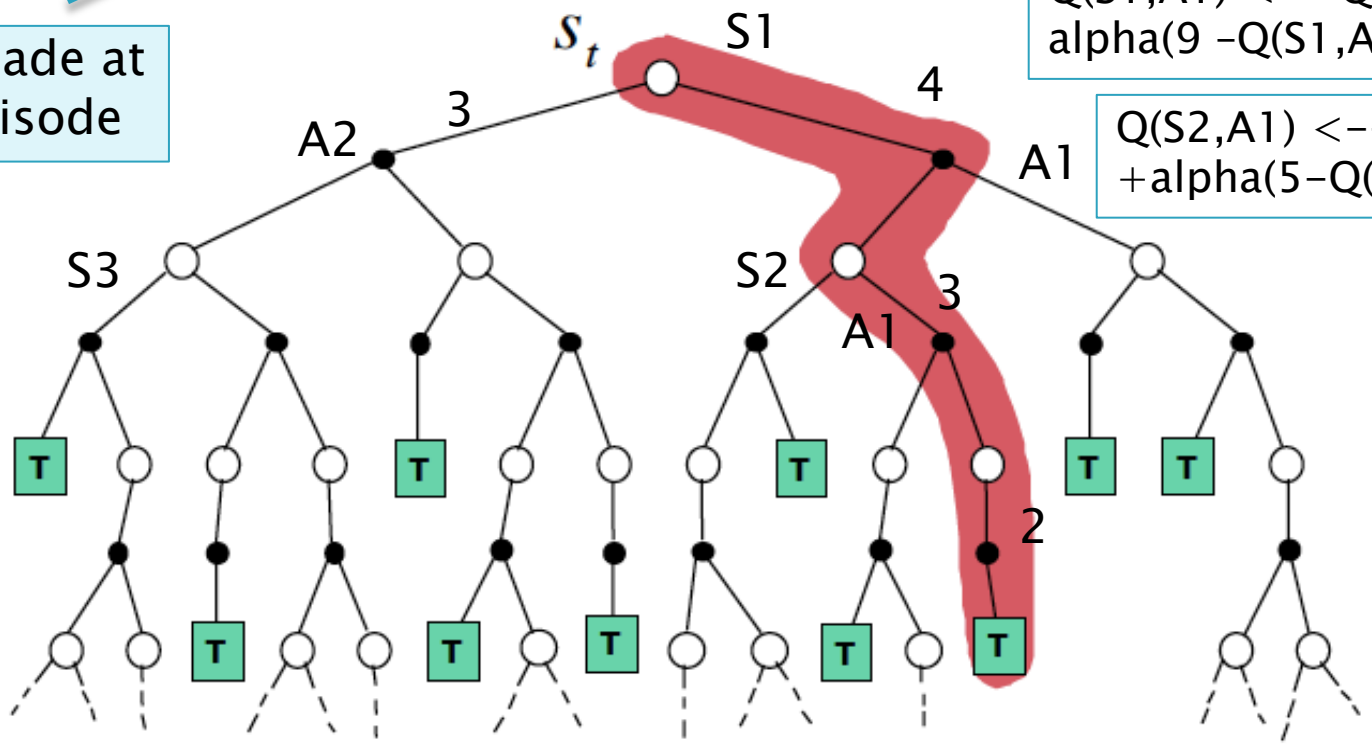
Monte Carlo Backup for Q

$$Q(S,A) \leftarrow Q(S,A) + \alpha(G - Q(S,A))$$

Update made at end of episode

Q(S1,A1) <-- Q(S1,A1)+ alpha(9 -Q(S1,A1))

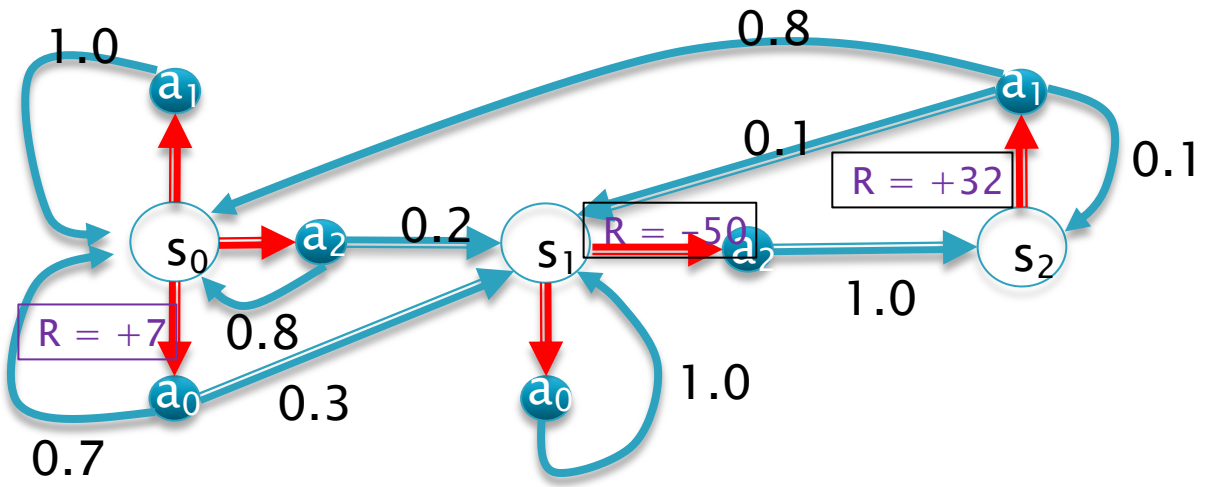
Q(S2,A1) <-- Q(S2,A1) +alpha(5-Q(S2,A1))



Another Problem: How to ensure that every (S,A) pair is visited?

Q(S,A) Update => Policy Update
Policy changes at end of every episode

Example: Q(S,A) Evaluation



$\pi(s_0) = a_0$

$\pi(s_1) = a_2$

$\pi(s_2) = a_1$

```
policy_fire
States (+rewards): 0 1 (-50) 2 (40) 0 (10) 0 (10) 0 (10) 0 1 (-50) 2 (40) 0 ... Total rewards = -220
States (+rewards): 0 (10) 0 (10) 0 (10) 0 (10) 0 (10) 0 (10) 0 (10) 0 (10) 0 ... Total rewards = 40
States (+rewards): 0 (10) 0 (10) 0 (10) 0 1 (-50) 2 (40) 0 (10) 0 1 (-50) 2 (40) ... Total rewards = 160
States (+rewards): 0 (10) 0 (10) 0 (10) 0 (10) 0 (10) 0 1 (-50) 2 (40) 0 (10) 0 (10) ... Total rewards = 280
States (+rewards): 0 (10) 0 1 (-50) 2 1 (-50) 2 (40) 0 (10) 0 (10) 0 (10) 0 (10) ... Total rewards = 190
Summary: mean=122.2, std=134.956674, min=-340, max=490
```

0 → 1 (−50) → 2 (32) → 0 (7) → 0 (7) → 0 (7) → 0 → 1 (−50) → 2 (32) → 0 …

Estimating V

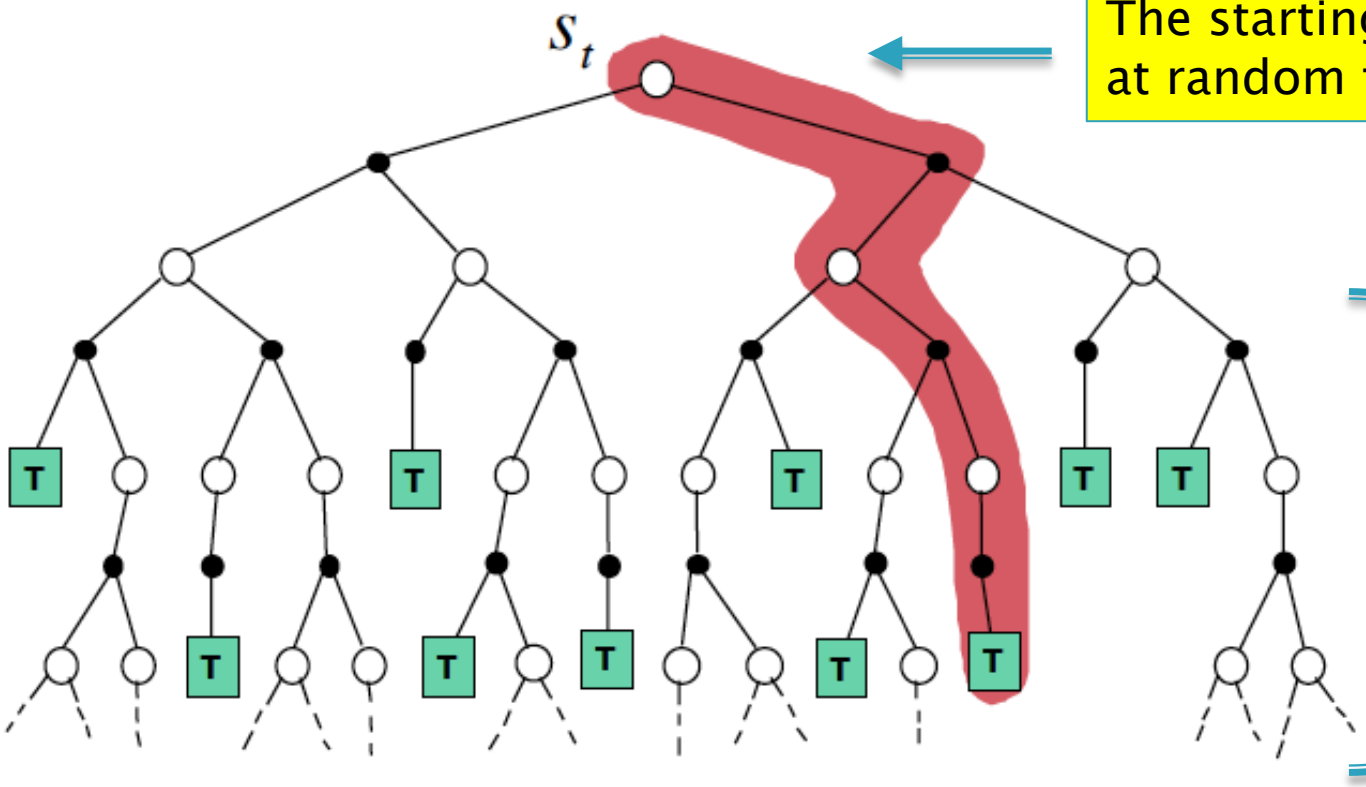
Estimating Q

0,2 → 1,2 (−50) → 2,1 (32) → 0,0 (7) → 0,0 (7) → 0,0 (7) → 0,0 → 1,2 (−50) → 2,1 (32) → 0,0 …

Monte Carlo Backup for Action Value Functions with Exploring Starts

Update made at end of an Episode

$$Q(S,A) \leftarrow Q(S,A) + \alpha(G - Q(S,A))$$



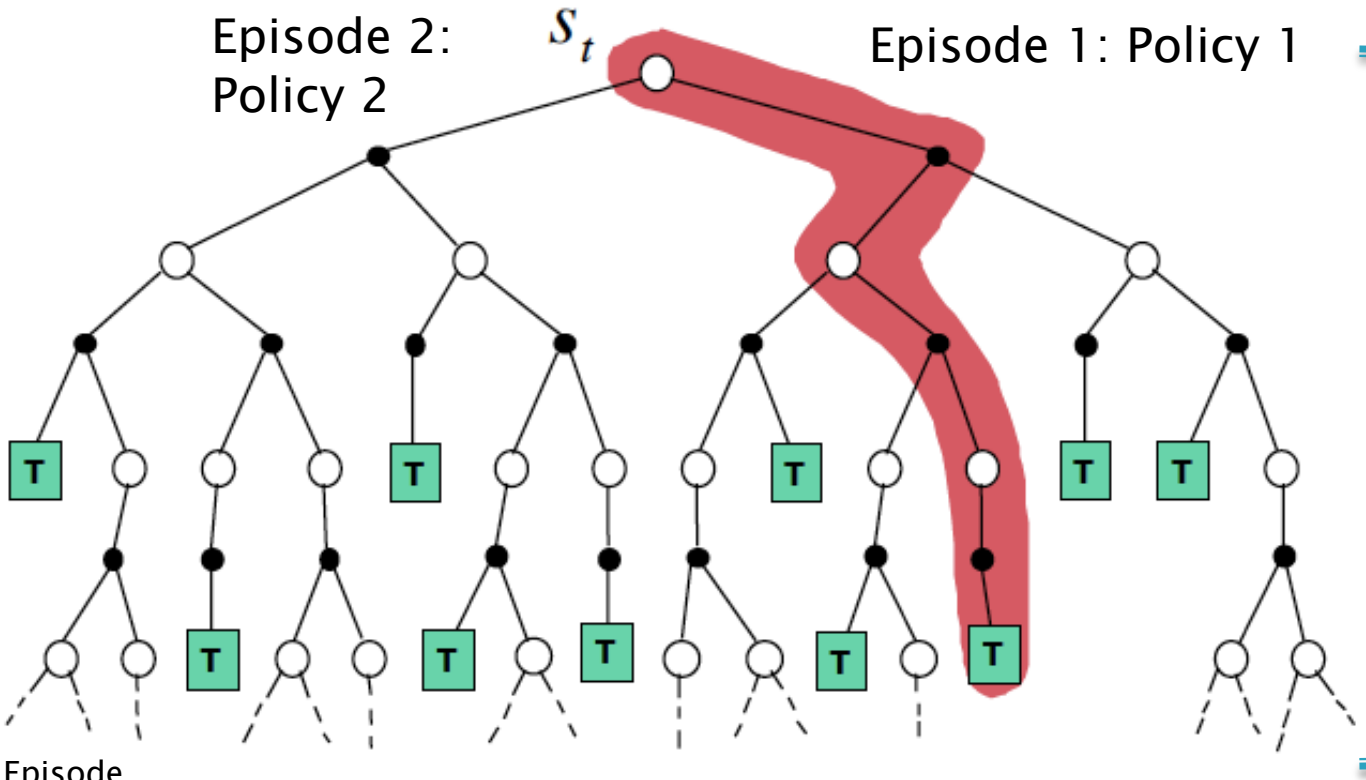
The starting (S,A) is chosen at random for each episode

Rest of the Actions are chosen according to Policy π

Policy Improvement Update made at end of an Episode

Model Free Monte Carlo Control

$$Q(S,A) \leftarrow Q(S,A) + \alpha(G - Q(S,A))$$



All of the Actions are chosen according to Policy π

Episode

0,0 → 1,2 (−50) → 2,1 (32) → 0,0 (7) → 0,0 (7) → 0,0 (7) → 0,0 → 1,2 (−50) → 2,1 (32) → 0,0

Policy Improvement Update made at end of an Episode

Monte Carlo Control with Exploring Starts

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
    $Q(s,a) \leftarrow$ arbitrary
    $\pi(s) \leftarrow$ arbitrary
    $Returns(s,a) \leftarrow$ empty list

Repeat forever:
    Choose $S_0 \in \mathcal{S}$ and $A_0 \in \mathcal{A}(S_0)$ s.t. all pairs have probability > 0
    Generate an episode starting from S_0, A_0, following π
    For each pair s,a appearing in the episode:
        $G \leftarrow$ the return that follows the first occurrence of s,a
        Append G to $Returns(s,a)$
        $Q(s,a) \leftarrow$ average$(Returns(s,a))$
    For each s in the episode:
        $\pi(s) \leftarrow \arg\max_a Q(s,a)$

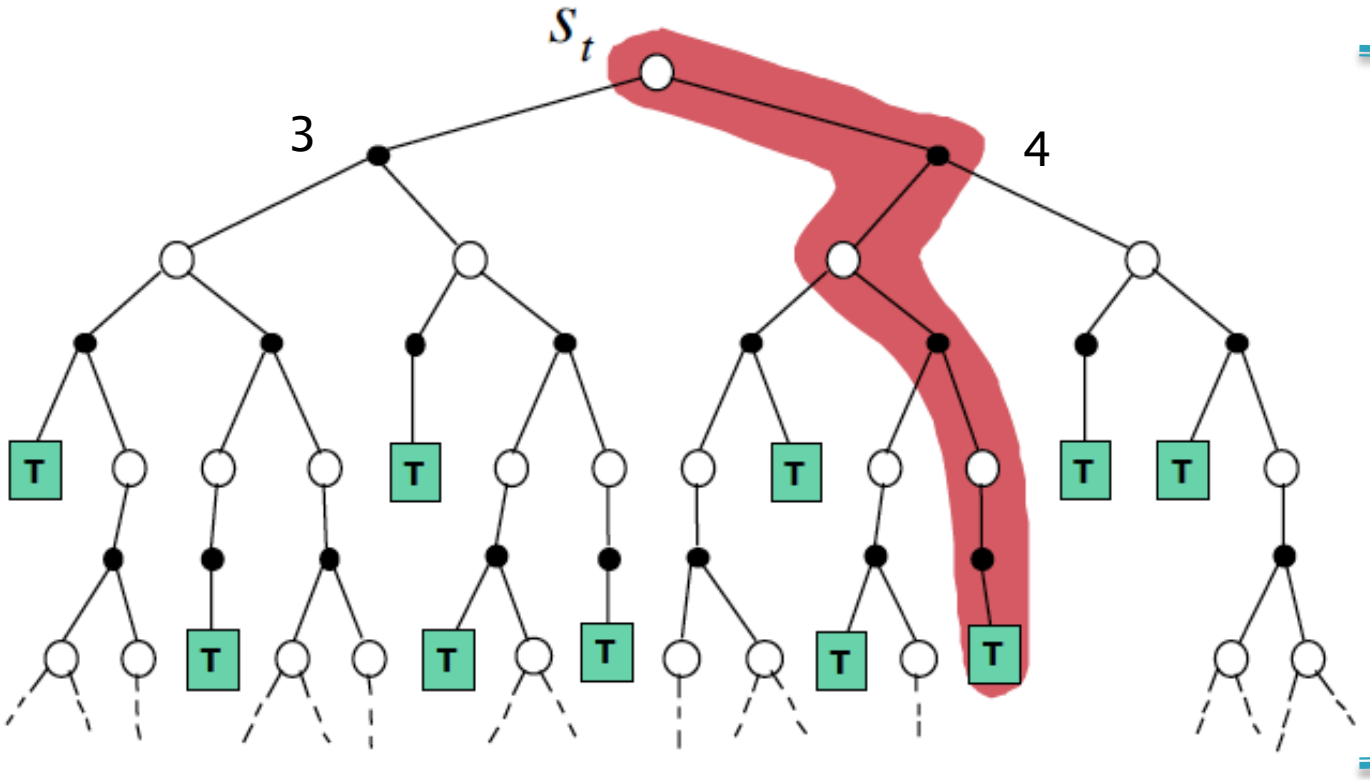
How to Avoid the Exploring Starts Assumption

General Strategy: Continue to select all possible Actions (even during an episode)

But: The agent is supposed to follow Policy π.

Idea: Randomize the Policy!

$$Q(S,A) \leftarrow Q(S,A) + \alpha(G - Q(S,A))$$



All of the Actions are chosen according to Policy π'

π' is a Random Policy

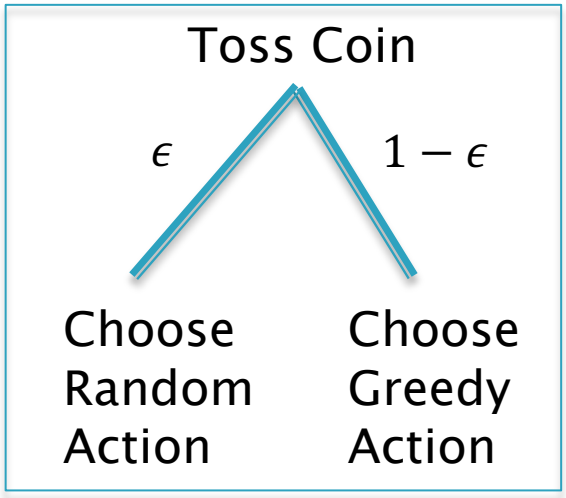
Policy Improvement Update made at end of an Episode

ε –Greedy Exploration

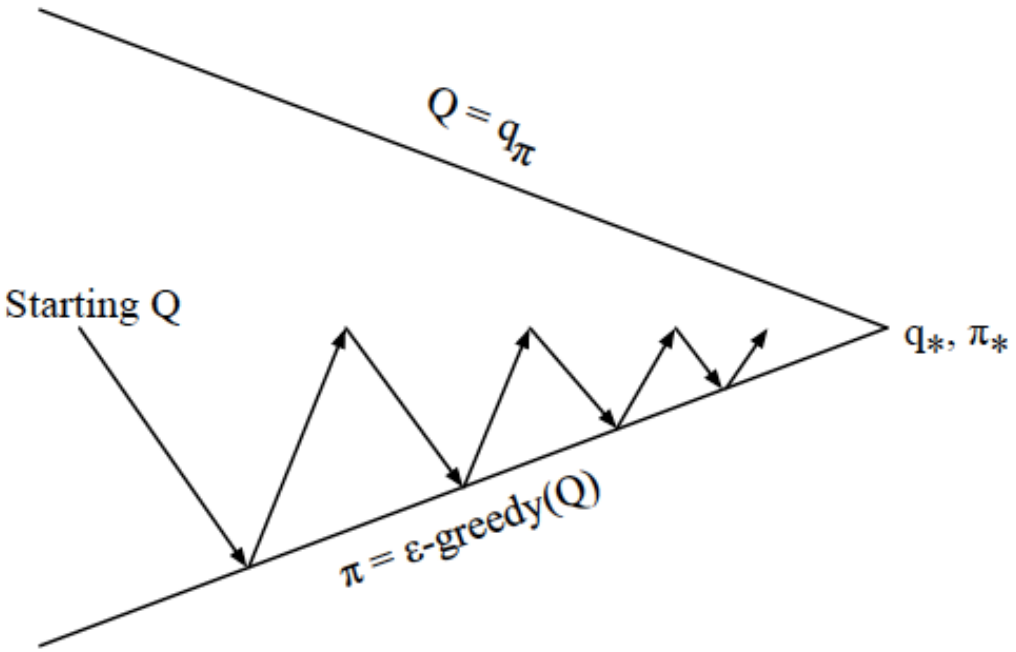
- Simplest idea for ensuring continual exploration
- All m actions are tried with non-zero probability
- With probability $1 - \epsilon$ choose the greedy action
- With probability ϵ choose an action at random

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \underset{a \in \mathcal{A}}{\text{argmax}}\ Q(s,a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

Works very well in practice
Guarantees that you continue to explore everything
Guarantees that you improve your policy



Generalized Policy Iteration with Action-Value Function and ϵ Greedy Exploration



Every episode:

Policy evaluation Monte-Carlo policy evaluation, $Q \approx q_\pi$

Policy improvement ϵ-greedy policy improvement

On-Policy First Visit MC Control with ε Greedy Policies

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
$Q(s,a) \leftarrow$ arbitrary
$Returns(s,a) \leftarrow$ empty list
$\pi(a|s) \leftarrow$ an arbitrary ε-soft policy

Repeat forever:
(a) Generate an episode using π
(b) For each pair s, a appearing in the episode:
$G \leftarrow$ the return that follows the first occurrence of s, a
Append G to $Returns(s,a)$
$Q(s,a) \leftarrow \text{average}(Returns(s,a))$
(c) For each s in the episode:
$A^* \leftarrow \arg\max_a Q(s,a)$ (with ties broken arbitrarily)
For all $a \in \mathcal{A}(s)$:

$$\pi(a|s) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq A^* \end{cases}$$

On Policy: Agent uses Policy to generate Episode and then used that Episode to improve the same Policy

Exploration and Exploitation

- *Exploration* finds more information about the environment
- *Exploitation* exploits known information to maximise reward
- It is usually important to explore as well as exploit

ε –Greedy Policy Improvement

Is the greedy ε policy π' actually better than the old policy π?

Theorem

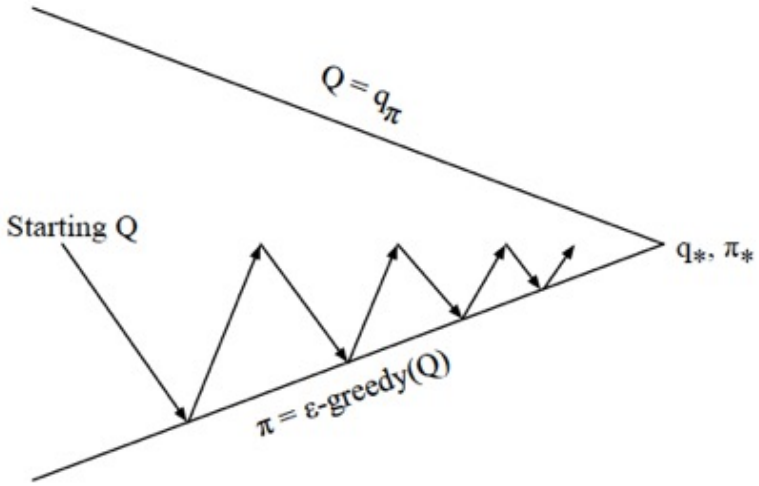
For any ϵ-greedy policy π, the ϵ-greedy policy π' with respect to q_π is an improvement, $v_{\pi'}(s) \geq v_\pi(s)$

$$
\begin{aligned}
v_{\pi'}(s) &= \sum_{a \in \mathcal{A}} \pi'(a|s) q_\pi(s,a) \\
&= \epsilon/m \sum_{a \in \mathcal{A}} q_\pi(s,a) + (1-\epsilon) \max_{a \in \mathcal{A}} q_\pi(s,a) \\
&\geq \epsilon/m \sum_{a \in \mathcal{A}} q_\pi(s,a) + (1-\epsilon) \sum_{a \in \mathcal{A}} \frac{\pi(a|s) - \epsilon/m}{1-\epsilon} q_\pi(s,a) \\
&= \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a) = v_\pi(s)
\end{aligned}
$$

Therefore from policy improvement theorem, $v_{\pi'}(s) \geq v_\pi(s)$

One More Problem …

- We know that the Optimal Policy is NOT Random
- We need a way to gradually reduce the randomness in the Policy



Solution: GLIE

Definition

Greedy in the Limit with Infinite Exploration (GLIE)

- All state-action pairs are explored infinitely many times,

$$\lim_{k \to \infty} N_k(s, a) = \infty$$

- The policy converges on a greedy policy,

$$\lim_{k \to \infty} \pi_k(a|s) = \mathbf{1}(a = \operatorname*{argmax}_{a' \in \mathcal{A}} Q_k(s, a'))$$

- For example, ϵ-greedy is GLIE if ϵ reduces to zero at $\epsilon_k = \frac{1}{k}$

GLIE Monte-Carlo Control

- Sample kth episode using π: $\{S_1, A_1, R_2, ..., S_T\} \sim \pi$
- For each state S_t and action A_t in the episode,

$$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(G_t - Q(S_t, A_t)\right)$$

- Improve policy based on new action-value function

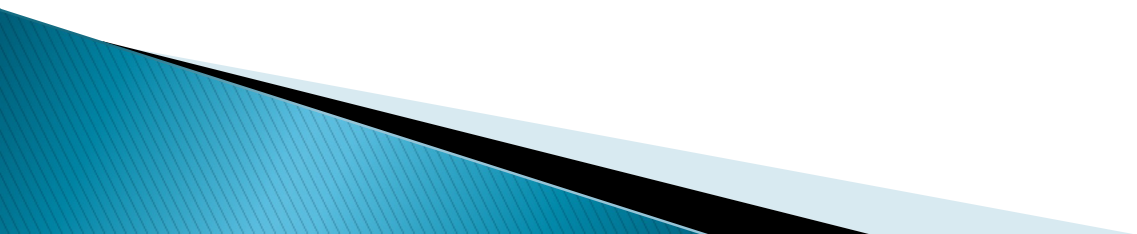
$$\epsilon \leftarrow 1/k$$

$$\pi \leftarrow \epsilon\text{-greedy}(Q)$$

Theorem

GLIE Monte-Carlo control converges to the optimal action-value function, $Q(s, a) \rightarrow q_*(s, a)$

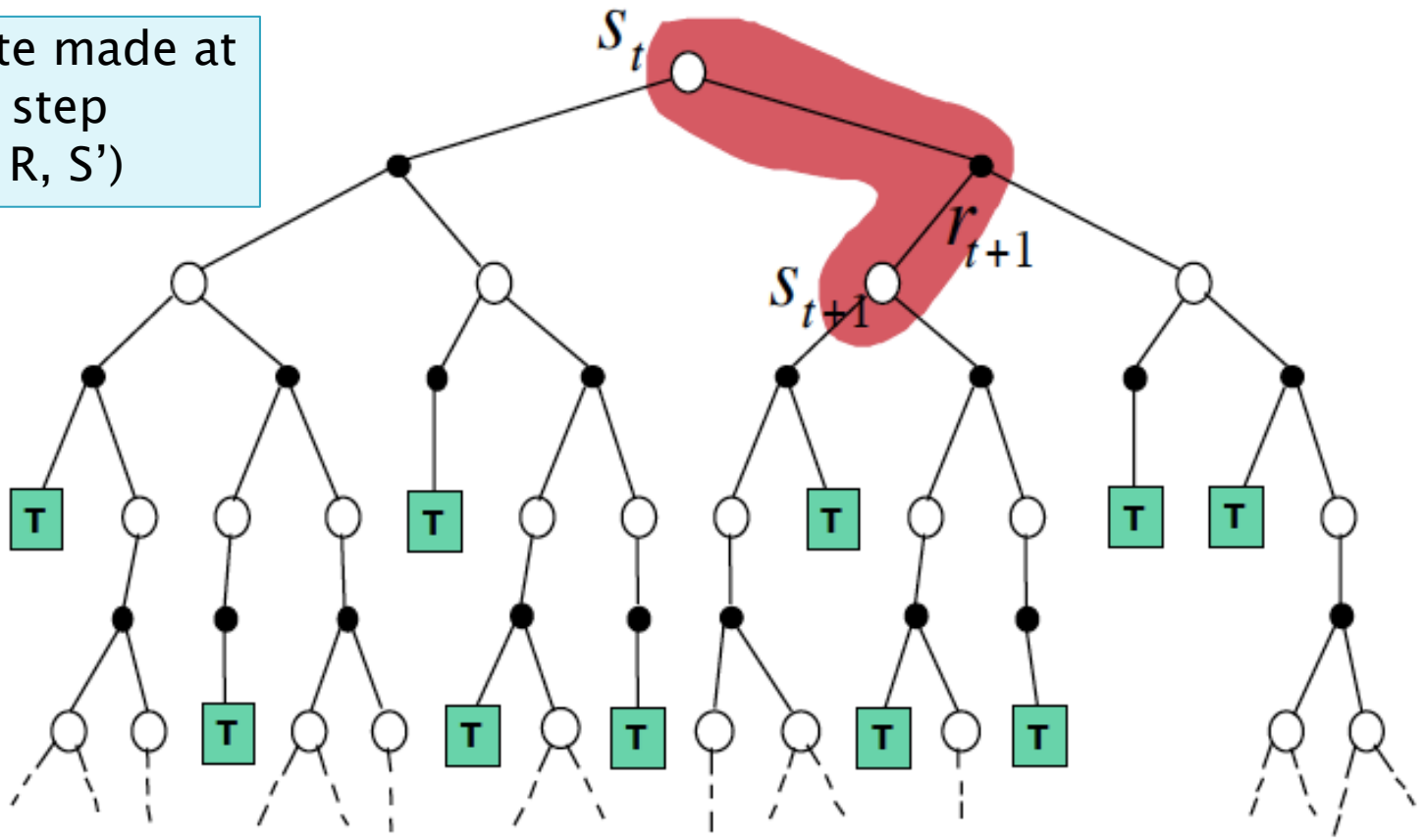
On Policy TD Control: The SARSA Algorithm



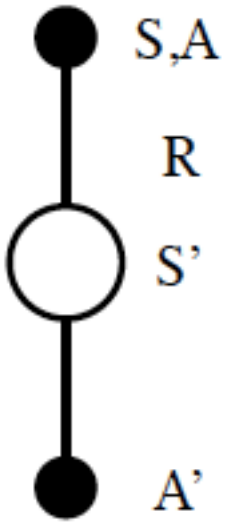
Recall: Temporal-Difference (TD) Learning

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

Update made at every step (S, A, R, S')



Updating Q Functions with SARSA



Start from (S,A)

Sample S'

Sample A'

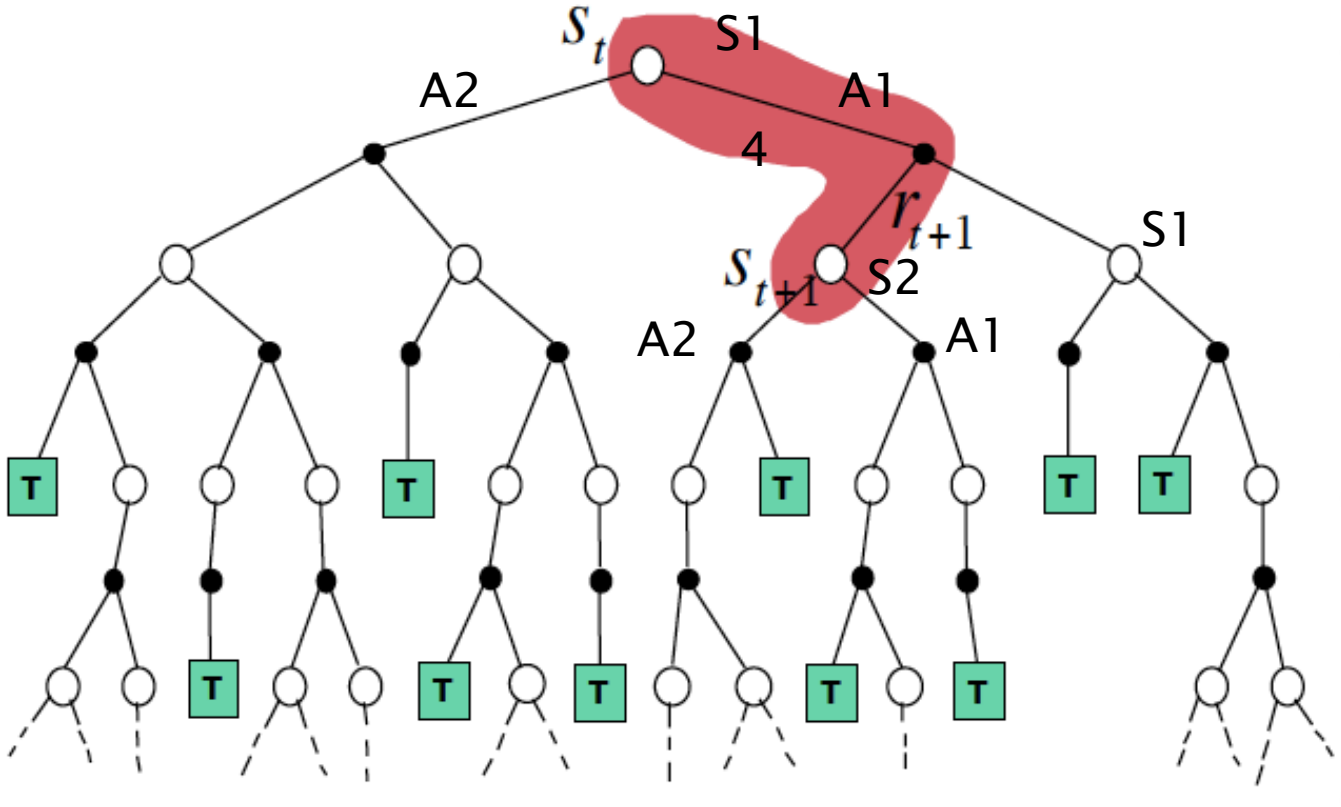
$$Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma Q(S',A') - Q(S,A)\right)$$

For Value Functions:

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

Model Free On Policy Temporal-Difference Algorithm: SARSA

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma Q(S',A') - Q(S,A)\right)$$

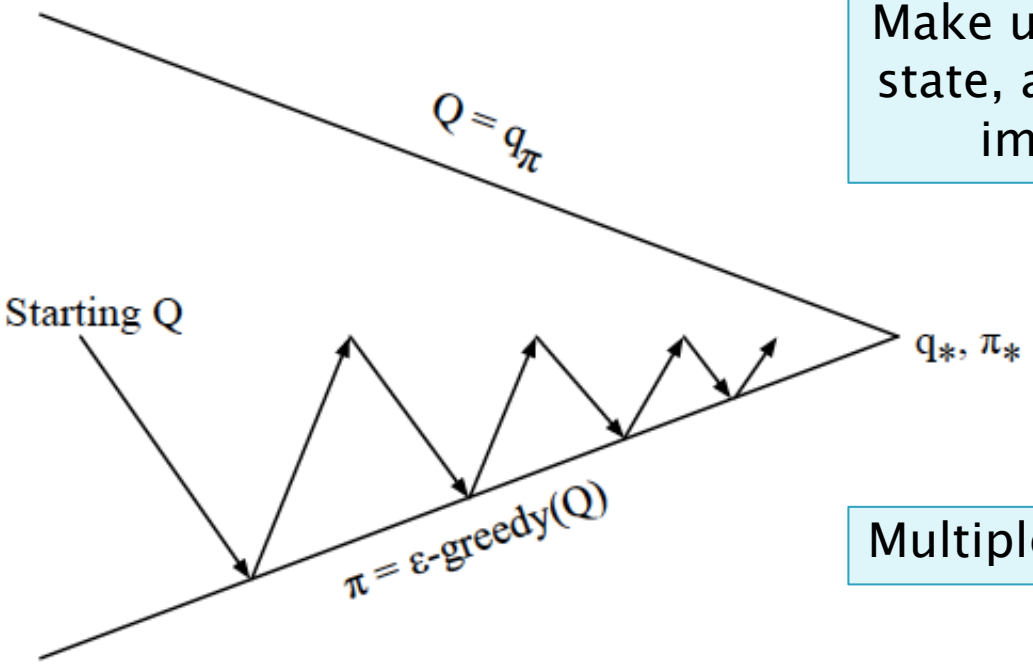


Actions are chosen according to Policy $\pi 1$

Actions are chosen according to Improved Policy π

Policy Improvement Update made after each step!

On-Policy Control with SARSA



Make update to just a single state, and then immediately improve the Policy!

Multiple updates per episode

Every time-step:

Policy evaluation Sarsa, $Q \approx q_\pi$

Policy improvement ϵ-greedy policy improvement

SARSA Algorithm for On-Policy Control

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (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

MC vs TD Control

- Temporal-difference (TD) learning has several advantages over Monte-Carlo (MC)
  - Lower variance
  - Online
  - Incomplete sequences

Convergence of SARSA

Theorem

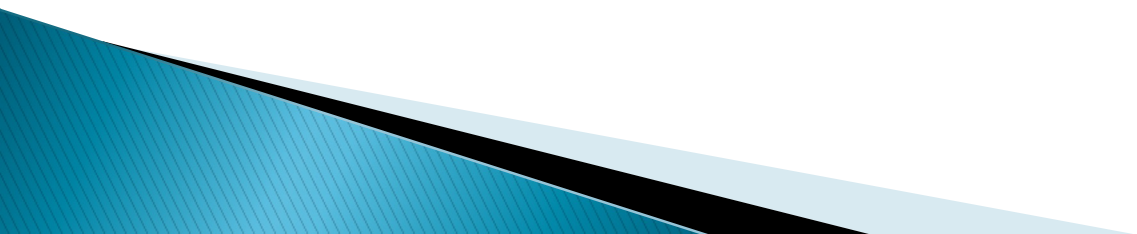
Sarsa converges to the optimal action-value function, $Q(s,a) \rightarrow q_*(s,a)$, under the following conditions:

- *GLIE sequence of policies $\pi_t(a|s)$*
- *Robbins-Monro sequence of step-sizes α_t*

$$\sum_{t=1}^{\infty} \alpha_t = \infty$$

$$\sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

Off Policy Temporal Difference Control: The Q Learning Algorithm



On and Off-Policy Learning

- On-policy learning
  - "Learn on the job"
  - Learn about policy π from experience sampled from π

- Off-policy learning
  - "Look over someone's shoulder"
  - Learn about policy π from experience sampled from μ

Off Policy Control with Q Learning

$$Q(S,A) \leftarrow Q(S,A) + \alpha(R + \gamma Q(S', argmax_{a'} Q(S',a')) - Q(S,A))$$

Behavior Policy

Target Policy

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma \max_{a'} Q(S',a') - Q(S,A)\right)$$

Next Action chosen according to a randomized Behavior Policy
This ensures Exploration of the State Space

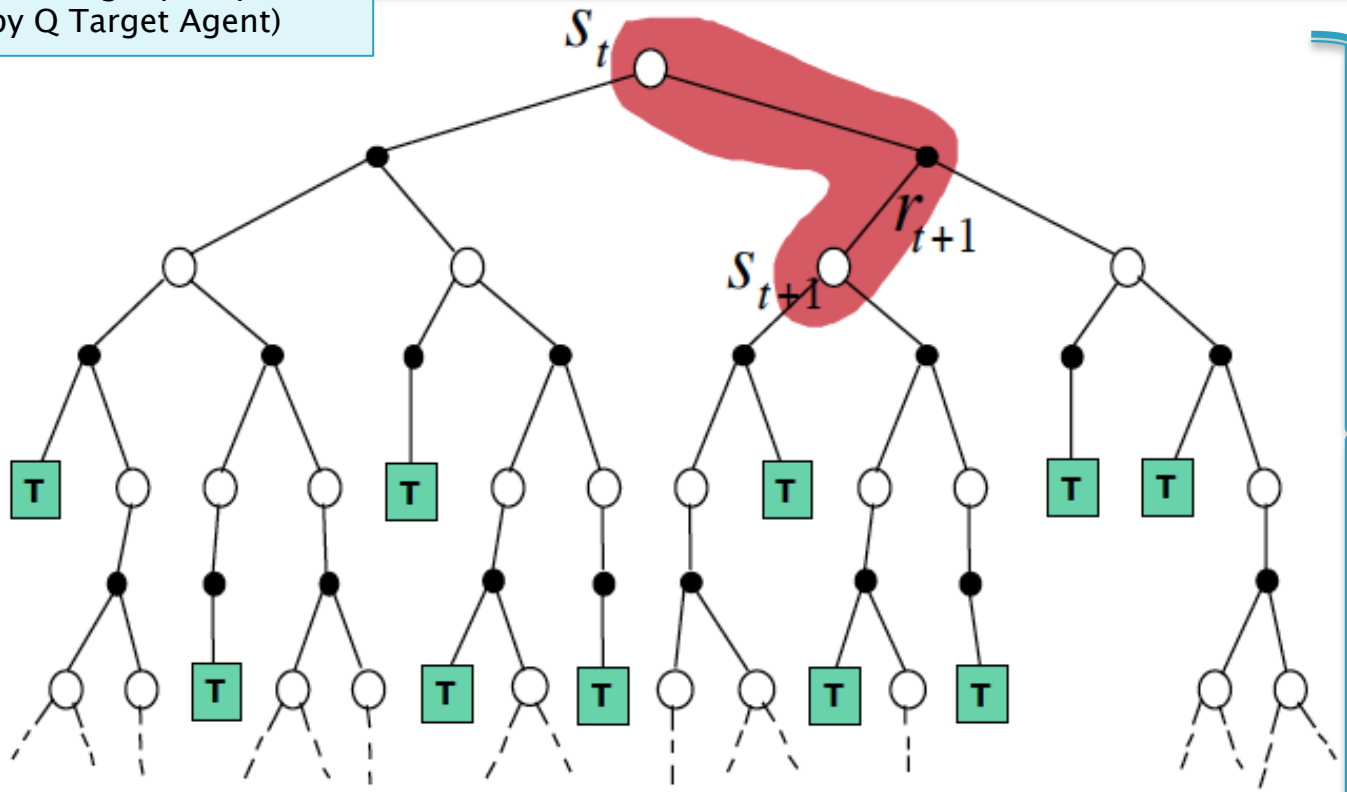
But: Q-Value Update made according to the 'Optimal' Target Policy

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma Q(S',A') - Q(S,A)\right)$$

Model Free Off Policy Temporal-Difference Algorithm – Q Learning

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma \max_{a'} Q(S',a') - Q(S,A)\right)$$

The Q value updates made at each step according to Target policy (by Q Target Agent)



All of the Actions chosen according to Behavior Policy $\pi 1$ (by Behavior Agent

Two Agents

Behavior Agent

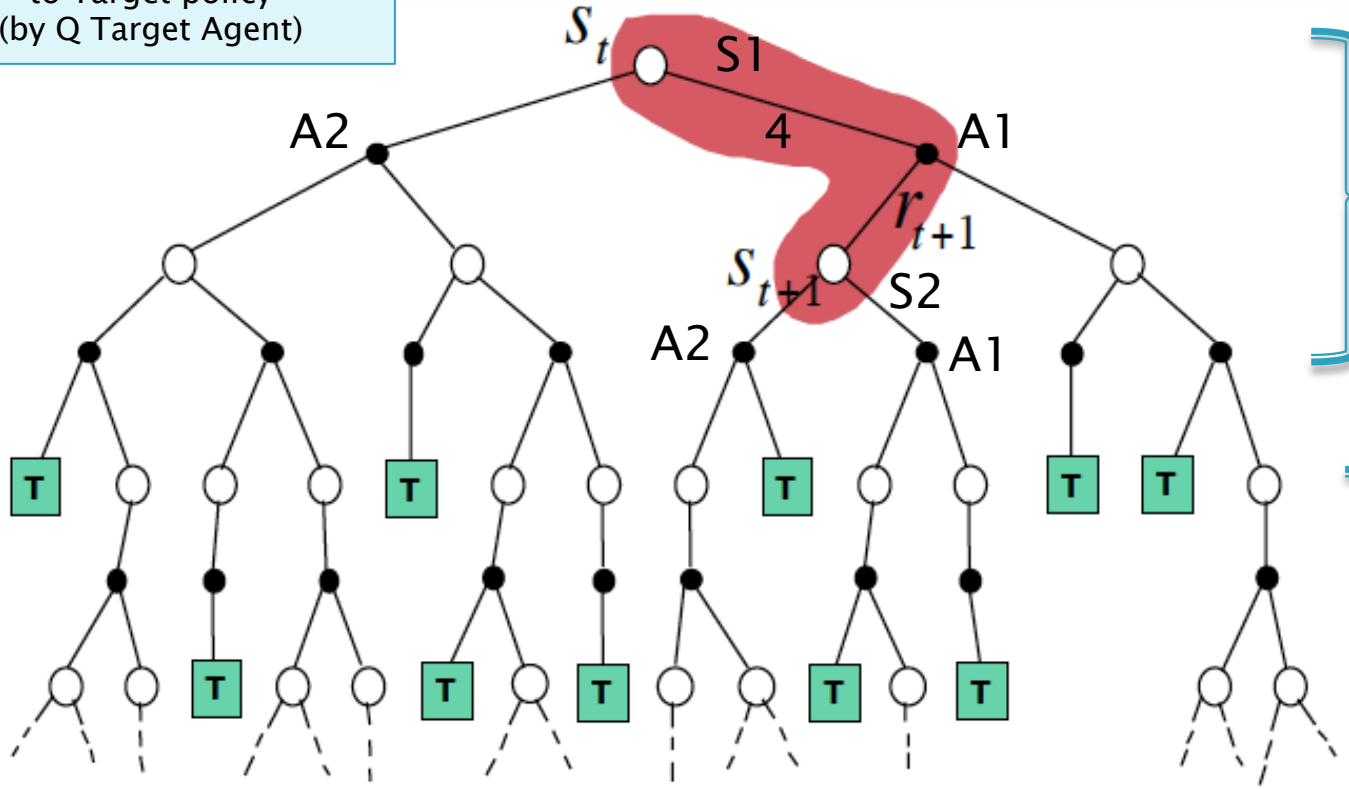
Target Agent

Q Learning: Behavior Policy also Evolves

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma \max_{a'} Q(S',a') - Q(S,A)\right)$$

The Q value updates made at each step according to Target policy (by Q Target Agent)

Use ϵ Greedy to Choose Behavior Policy



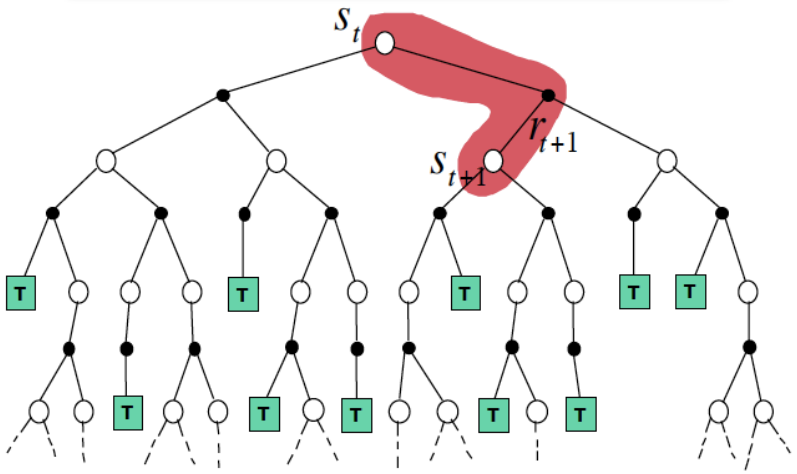
Actions chosen according to Behavior Policy $\pi 1$ (by Behavior Agent)

Actions chosen according to improved Behavior Policy $\pi 2$ (by Behavior Agent)

Behavior Agent uses the same Q value to choose actio

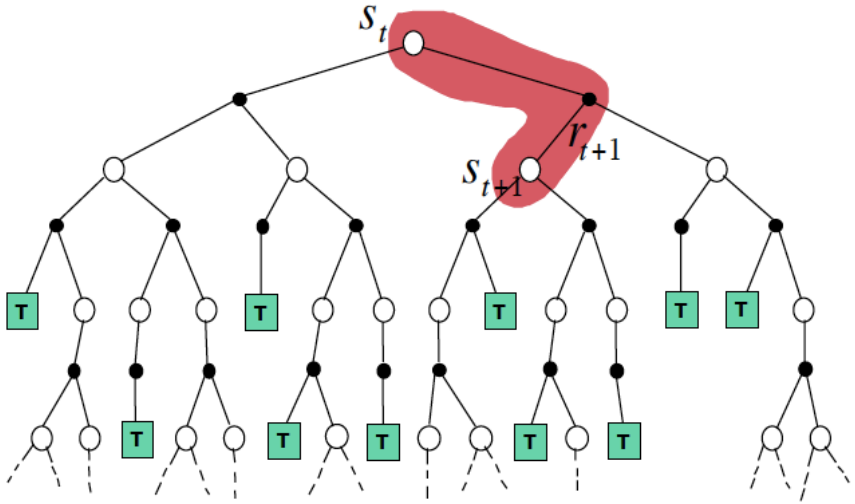
General Off Policy Learning

Behavior Agent chooses actions
Based on its own policy. For example
It can simply choose action with
Equal probability



Behavior Agent

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma \max_{a'} Q(S',a') - Q(S,A)\right)$$



Target Agent

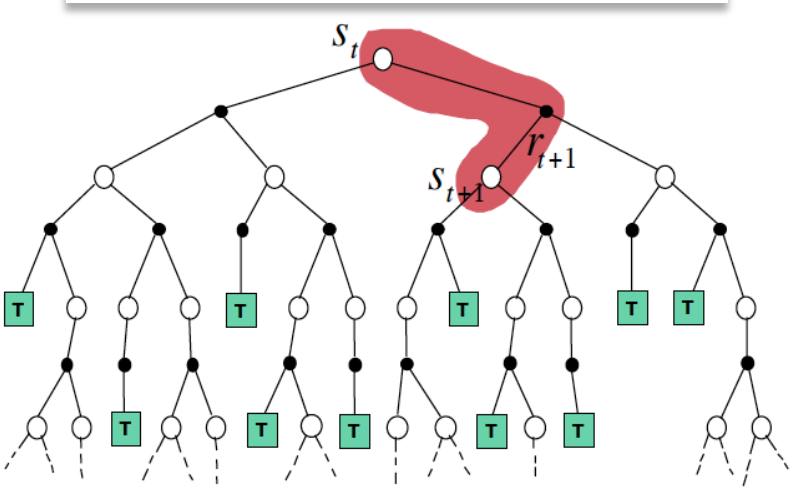
Follows Behavior Agent
AND In Parallel
Computes Best Possible Action

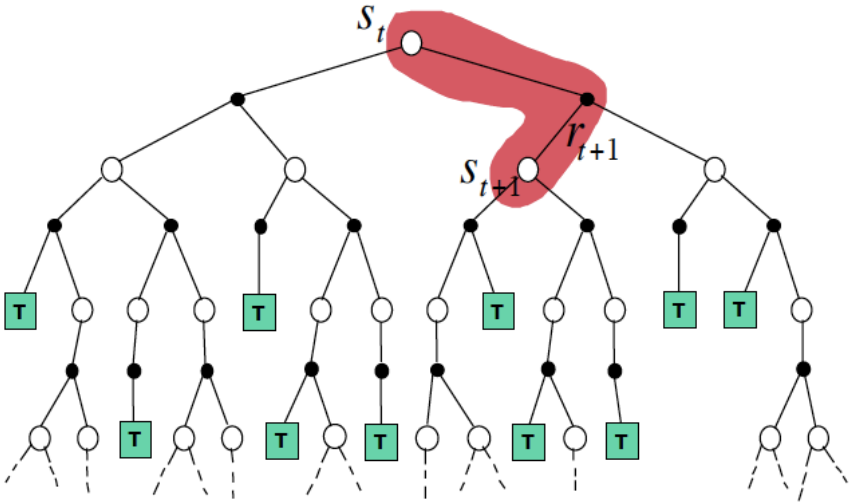
Q Learning

A special case of
Off Policy Learning

Behavior Agent chooses actions using
The Q values that the Target Agent
computes

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma \max_{a'} Q(S',a') - Q(S,A)\right)$$





Behavior Agent

Controls All Actions Actually Taken
Using epsilon-greedy algo

Target Agent

Follows Behavior Agent
AND In Parallel
Computes Best Possible Action

Off Policy Control with Q Learning

- We now allow both behaviour and target policies to improve
- The target policy π is greedy w.r.t. $Q(s, a)$

$$\pi(S_{t+1}) = \underset{a'}{\text{argmax}}\ Q(S_{t+1}, a')$$

- The behaviour policy μ is e.g. ϵ-greedy w.r.t. $Q(s, a)$
- The Q-learning target then simplifies:

$$\begin{aligned} & R_{t+1} + \gamma Q(S_{t+1}, A') \\ =& R_{t+1} + \gamma Q(S_{t+1}, \underset{a'}{\text{argmax}}\ Q(S_{t+1}, a')) \\ =& R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a') \end{aligned}$$

Q Learning Algorithm for Off Policy Control

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

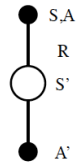
Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$
  until S is terminal

Behavior Policy

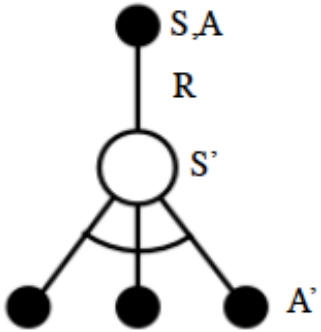
Target Policy

Q Learning Control Algorithm

SARSA



$$Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma Q(S',A') - Q(S,A)\right)$$



$$Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma \max_{a'} Q(S',a') - Q(S,A)\right)$$

Theorem

Q-learning control converges to the optimal action-value function, $Q(s,a) \rightarrow q_*(s,a)$

Other Uses of Off Policy Learning

- Learn about *optimal* policy while following *exploratory* policy
- Learn from observing humans or other agents
- Learn about *multiple* policies while following *one* policy
- Re-use experience generated from old policies $\pi_1, \pi_2, \ldots, \pi_{t-1}$

Critical Idea used in
Deep Reinforcement Learning

Q Learning in Batch Mode

$$Q(S,A) \leftarrow Q(S,A) + \alpha \left(R + \gamma \max_{a'} Q(S',a') - Q(S,A) \right)$$

(S1,A1,R1,S1')

(S2,A2,R2,S2')

(S3,A3,R3,S3')

(S4,A4,R4,S4')

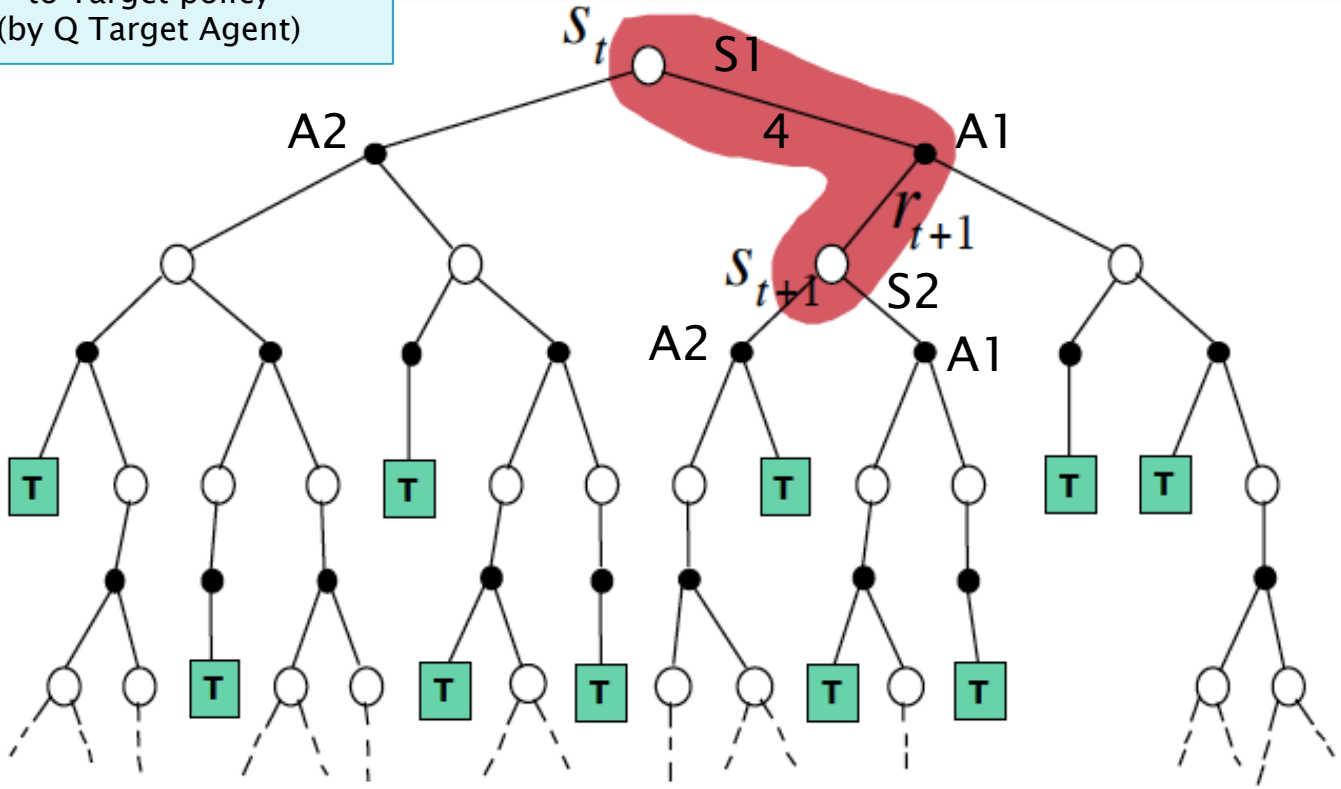
A Collection of
1-Step Transitions

Q Learning: Behavior Policy also Evolves

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma \max_{a'} Q(S',a') - Q(S,A)\right)$$

The Q value updates made at each step according to Target policy (by Q Target Agent)

Use ϵ Greedy to Choose Behavior Policy

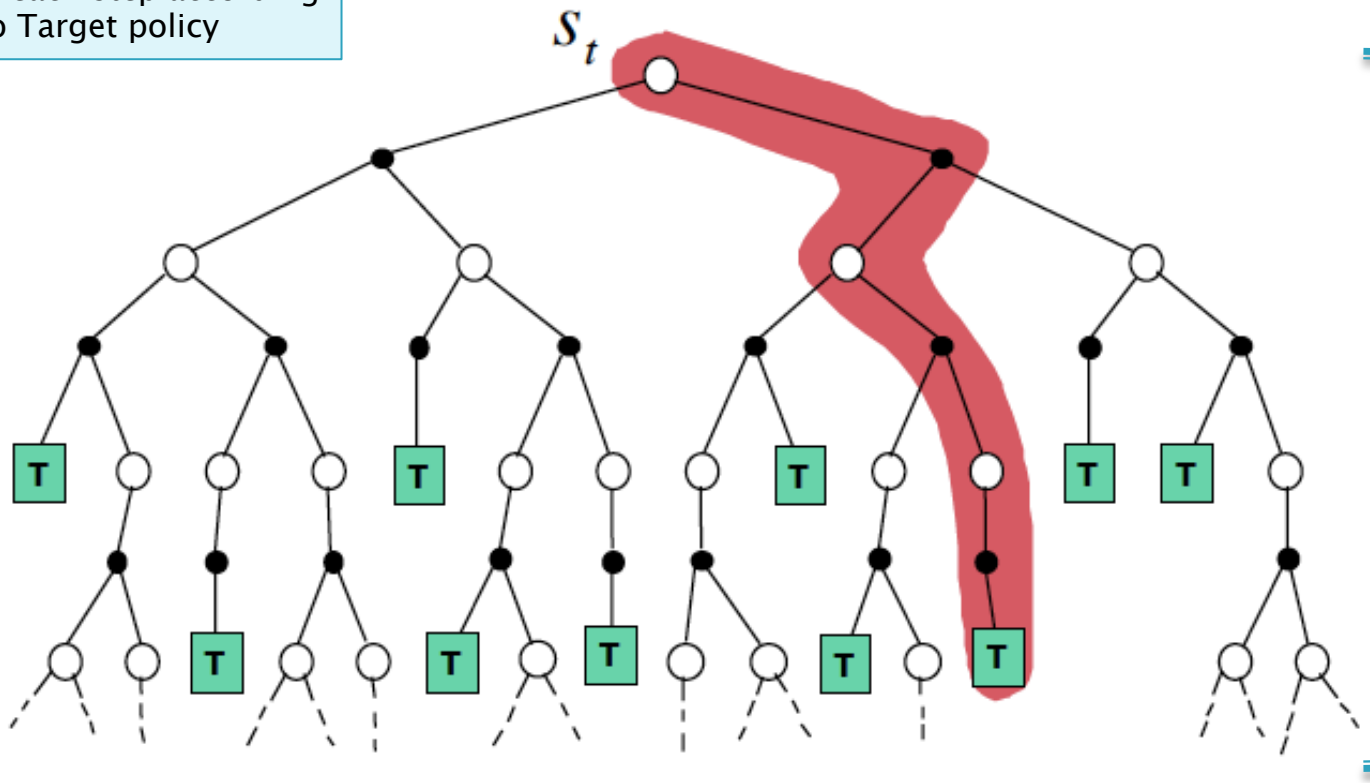


Allows reuse of a single transition multiple times

Q Learning in Batch Mode

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma \max_{a'} Q(S',a') - Q(S,A)\right)$$

The Q value updates made at each step according to Target policy



All of the Actions in the episode are fixed

Allows reuse of a single episode multiple times

Summary

Summary

- On Policy Monte Carlo Control: This involves a Policy Evaluation step follows by a Policy Improvement Step. Policy Evaluation is done based on the data from a complete episode of the MDP. This is followed by Policy Improvement using the new Q values, and the new policy is used to generate the next episode.

- On Policy Temporal Difference Control (SARSA): This also involves Policy Evaluation followed by Policy Improvement. However the Policy Evaluation is based on the data from a single step of the MDP. This is immediately followed by Policy Improvement, and the improved policy is used to generate the next step of the MDP.

- Off Policy Temporal Difference Control (Q Learning): In this case the Agent taking the Actions (using the Behavior Policy) is different from the Agent computing the optimal Q function (using the Target Policy). Behavior Policy uses some randomness to traverse the MDP, and Target Policy uses the data generated from this traversal to compute the optimal Q function.

Further Reading

Sutton and Barto:

- Chapter 5: Sections 5.3 – 5.4
- Chapter 6: Sections 6.4 – 6.5

Example: Monte Carlo

$\alpha = 0.8, \gamma = 1$

- **Given the following episode:**

  (s1,a0) (r = 3) → (s0,a0) (r = 2) → (s2,a1) (r = -1) → (s0,a0)

  assume that the Q values in the starting iteration are given by the following table:

Q(s,a)	a0	a1
s0	2	-1
s1	4	3
s2	0	5

(c) Monte Carlo $Q(S,A) \leftarrow Q(S,A) + \alpha(G - Q(S,A))$

$q(s_1, a_0) = 4 + 0.8 \times ((3+2-1) - 4) = 4$

$q(s_0, a_0) = 2 + 0.8 \times ((2-1) - 2) = 1.2$

$q(s_2, a_1) = 5 + 0.8 \times (-1 - 5) = 0.2$

Example: SARSA

$\alpha = 0.8, \gamma = 1$

Given the following episode:

(s1,a0) (r = 3) → (s0,a0) (r = 2) → (s2,a1) (r = -1) → (s0,a0)

assume that the Q values in the starting iteration are given by the following table:

Q(s,a)	a0	a1
s0	2	-1
s1	4	3
s2	0	5

(a) SARSA: $Q(S,A) \leftarrow Q(S,A) + \alpha (R + \gamma Q(S',A') - Q(S,A))$

$q(s_1,a_0) = q(s_1,a_0) + 0.8 \times (3 + q(s_0,a_0) - q(s_1,a_0))$

$= 4 + 0.8 \times (3+2-4) = 4.8$

$q(s_0,a_0) = q(s_0,a_0) + 0.8 \times (2 + q(s_2,a_1) - q(s_0,a_0))$

$= 2 + 0.8 \times (2+5-2) = 6$

$q(s_2,a_1) = q(s_2,a_1) + 0.8 \times (-1 + q(s_0,a_0) - q(s_2,a_1))$

$= 5 + 0.8 \times (-1 + 6 - 5) = 5$

Example: Q Learning

$\alpha = 0.8, \gamma = 1$

- **Given the following episode:**

  (s1,a0) (r = 3) → (s0,a0) (r = 2) → (s2,a1) (r = -1) → (s0,a0)

  assume that the Q values in the starting iteration are given by the following table:

Q(s,a)	a0	a1
s0	2	-1
s1	4	3
s2	0	5

(b) Q-learning: $Q(S,A) \leftarrow Q(S,A) + \alpha(R + \gamma \max_{a'} Q(S',a') - Q(S,A))$

$Q(S_1, a_0) = 4 + 0.8 \times (3 + \max(2, -1) - 4) = 4.8$

$Q(S_0, a_0) = 2 + 0.8 \times (2 + \max(0, 5) - 2) = 6$

$Q(S_2, a_1) = 5 + 0.8 \times (-1 + \max(6, -1) - 5) = 5$