Robot Learning

Lecture 8 Model-Free RL -- Value-based

Lecture 5c:

Model-Free Reinforcement Learning -- Prediction and Control

Slides heavily adapted from David Silver (UCL)

Model-Free Reinforcement Learning

- So far: solve a known MDP (states, transition, rewards, actions)
- Model free
  - No environment model
  - No knowledge of MDP transition/rewards
- Model-free prediction - Estimate the value function of an unknown MDP
- Model-free control - Optimise the value function of an unknown MDP

Model-Free Prediction

Introduction

Outline

- Monte-Carlo approaches
- Temporal-Difference (TD) learning
- TD(λ)

Monte-Carlo

Monte-Carlo (MC) Reinforcement Learning

- MC methods learn directly from episodes of experience
- MC is model-free: no knowledge of MDP transitions/rewards
- MC learns from complete episodes: no bootstrapping

Text

- MC uses the simplest possible idea:

value = mean return across episodes

- Limitation: can only apply MC to episodic MDPs
  - All episodes must terminate

Monte-Carlo Policy Evaluation

- Goal: learn v_π from episodes of experience under policy π

$$S_1, A_1, R_1, \dots, R_k \sim \pi$$

- Recall that return is the total discounted reward

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

- Recall that value function is the expected return being in expected state in time s

$$v_\pi(s) = \mathbb{E}[G_t | S_t = s]$$

- Monte-Carlo policy evaluation uses empirical mean return instead of expected return

First-Visit Monte-Carlo Policy Evaluation

- ✓ To evaluate state s
- ✓ The first time-step t that state s is visited in an episode
  - I. Increment counter $\mathrm{N}(s) \leftarrow \mathrm{N}(s) + 1$
  - II. Increment total return $\mathrm{S}(s) \leftarrow \mathrm{S}(s) + G_t$

  Value is estimated by mean return $\mathrm{V}(s) = \mathrm{S}(s)/\mathrm{N}(s)$

  $\rightarrow$ update at the end of the episode
- ✓ By law of large numbers

$$\mathrm{V}(s) \rightarrow v_\pi(s) \text{ as } N(s) \rightarrow \infty$$

Every-Visit Monte-Carlo Policy Evaluation

- ✓ To evaluate state s
- ✓ Every time-step t that state s is visited in an episode
  - I. Increment counter $\mathrm{N}(s) \leftarrow \mathrm{N}(s) + 1$
  - II. Increment total return $\mathrm{S}(s) \leftarrow \mathrm{S}(s) + G_t$

  Value is estimated by mean return $\mathrm{V}(s) = \mathrm{S}(s)/\mathrm{N}(s)$

Update at the end

Blackjack Example

- States (200 of them):
  - Current sum (12-21)
  - Dealer's showing card (ace-10)
  - Do I have a useable ace? (yes-no)
- Reward for action stick (Stop receiving cards (and terminate)):
  - +1 if sum of cards > sum of dealer cards
  - 0 if sum of cards = sum of dealer cards
  - -1 if sum of cards < sum of dealer cards
- Reward for action twist (Take another card (no replacement)):
  - -1 if sum of cards > 21 (and terminate)
  - 0 otherwise
  - Transitions: automatically twist if sum of cards < 12

Blackjack Value Function after MC Learning

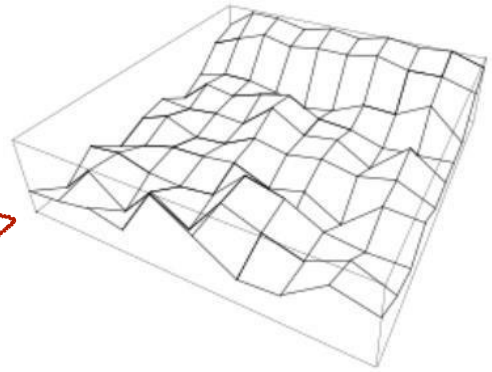
good estimation of the values

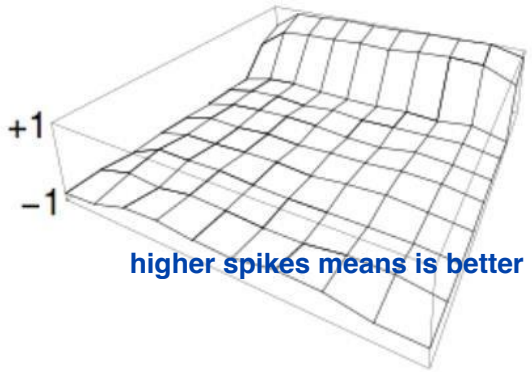
After 10K episodes

good estimation of the values

After 500K episodes

Usable ace



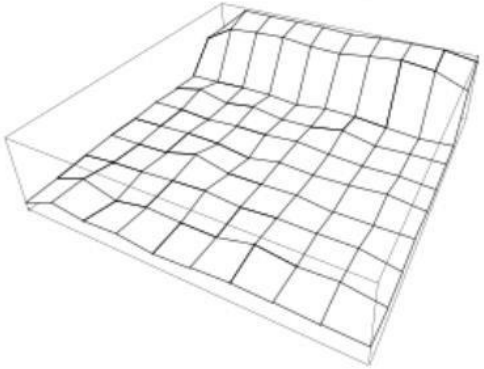


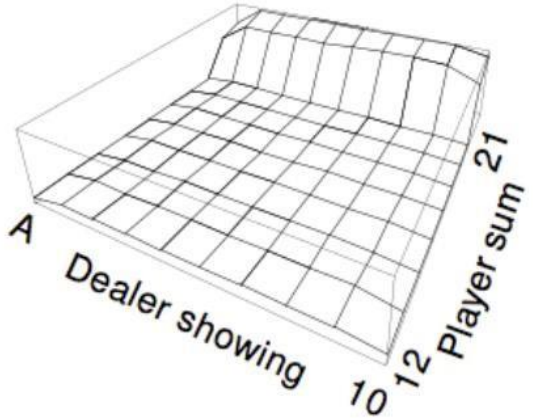
the state is the sum the cards

little noise because you do not have a lot of samples with usable ace

higher spikes means is better

No usable ace





Policy: stick if sum of cards ≥ 20, otherwise twist

Incremental Mean

The mean $\mu_1, \mu_2, \ldots$ of a sequence $x_1, x_2, \ldots$ can be computed incrementally

$$\mu_k = \frac{1}{k}\sum_{j=1}^{k} x_j = \frac{1}{k}\left(x_k + \sum_{j=1}^{k-1} x_j\right)$$

$$\mu_k = \frac{1}{k}(x_k + (k-1)\mu_{k-1}) = \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})$$

Incremental Mean MC Update

- Update $\mathrm{V}(s)$ incrementally after episode $S_1, A_1, R_1, \dots, R_T$
- For each state S_t with return G_t
  I. Increment counter $\mathrm{N}(s) \leftarrow \mathrm{N}(s) + 1$
  II. Update value function (with incremental mean)

$$\mathrm{V}(S_t) \leftarrow \mathrm{V}(S_t) + \frac{1}{N(S\,)}(G_t - V(S\,))$$

Spiegazione formula?

- In non-stationary problems track a running mean (forget old episodes)

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

Temporal-Difference Learning

Temporal-Difference (TD) Learning

- ✓TD methods learn directly from episodes of experience
- ✓TD is model-free: no knowledge of MDP transitions / rewards
- ✓TD learns from incomplete episodes, by bootstrapping
  - ✓While MC learns from complete ones

substitute the missing trajectory with an estimation

- ✓TD updates a guess towards a guess

MC Vs TD Learning

- Goal: learn v_π from episodes of experience under policy π
- Incremental every-visit MC
  - Update value $V(S_t)$ toward actual return G_t

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

update our value V(st) with the previous value with the estimated return (the immediate return plus the estimation

- Simplest temporal-difference learning algorithm (TD(0))
  - Update value $V(S_t)$ toward estimated return $R_{t+1} + \gamma V(S_{t+1})$

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

TD target is a guess

TD error δ_t

I could predict a crash without this happening
In monte carlo I have to crash to know there is a crash

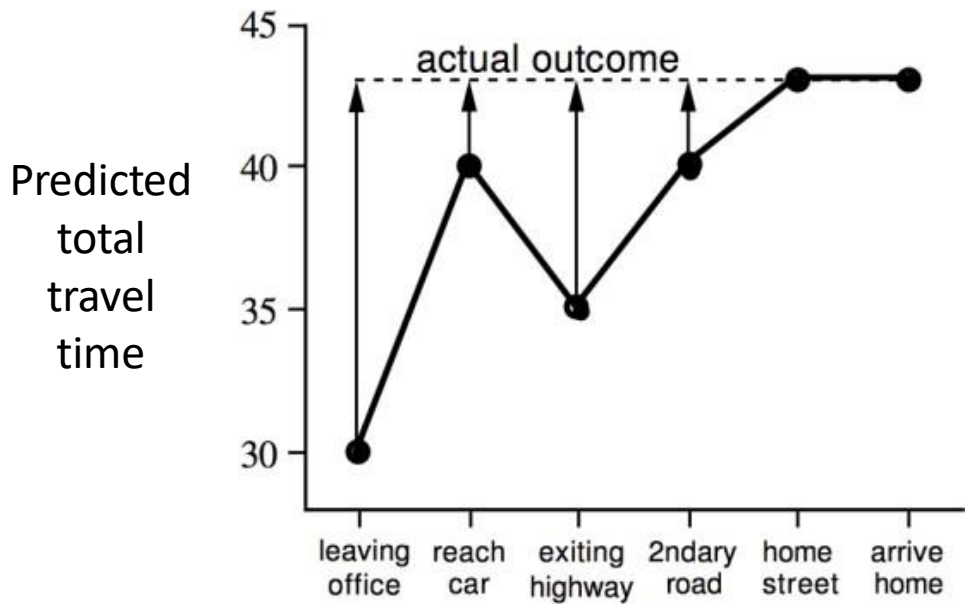
Driving Home Example

State	Elapsed Time (minutes)	Predicted Time to Go	Predicted Total Time
leaving office	0	30	30
reach car, raining	5	35	40
exit highway	20	15	35
behind truck	30	10	40
home street	40	3	43
arrive home	43	0	43

Driving Home Example – MC vs TD

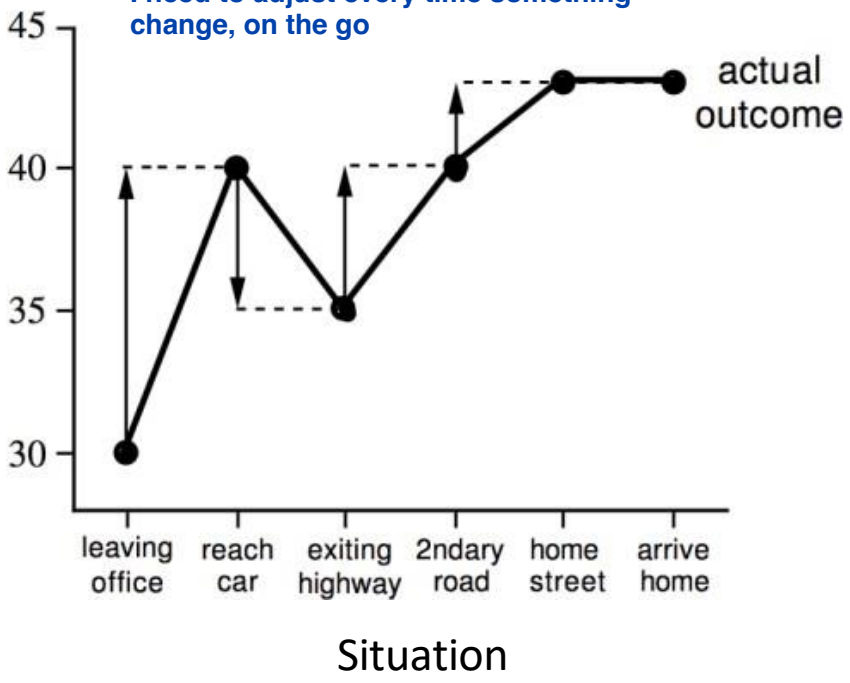
how update the value faction

Changes recommended by MC ($a = 1$)



Changes recommended by TD ($a = 1$)

I need to adjust every time something change, on the go



Advantages and Disadvantages of MC vs. TD (I)

- TD can learn before knowing the final outcome
  - TD can learn online after every step
  - MC must wait until end of episode before return is known
- TD can learn without the final outcome
  - TD can learn from incomplete sequences
  - MC can only learn from complete sequences
  - TD works in continuing (non-terminating) environments
  - MC only works for episodic (terminating) environments

Bias-Variance Tradeoff

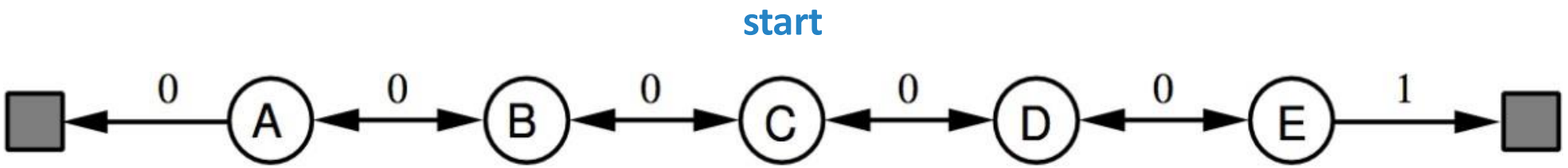
- Return $G_t = R_{t+1} + \gamma R_{t+2} + \cdots, \gamma^{T-1} R_T$ is unbiased estimate of $\mathrm{v}_\pi(S_t)$
- True TD target $R_{t+1} + \gamma v_\pi(S_{t+1})$ is unbiased estimate of $\mathrm{v}_\pi(S_t)$
- TD target $R_{t+1} + \gamma V(S_{t+1})$ is biased estimate of $\mathrm{v}_\pi(S_t)$
- TD target is much lower variance than the return:
  - Return depends on many random actions, transitions, rewards
  - TD target depends on one random action, transition, reward

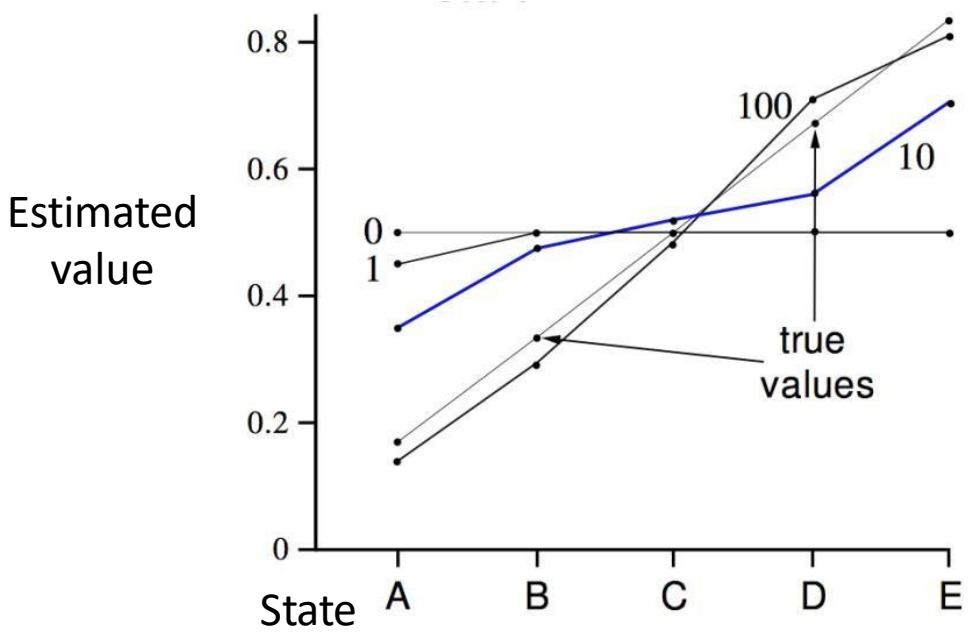
variance = noice, we have noise every time we make a prediction
In TD I consider only one transiction, is little bit more stable

Advantages and Disadvantages of MC vs. TD (II)

- MC has high variance, zero bias
  - Good convergence properties (even with function approximation)
  - Not very sensitive to initial value
  - Very simple to understand and use
- TD has low variance, some bias
  - Usually more efficient than MC
  - TD(0) converges to $v_\pi(s)$ (but not always with function approximation)
  - More sensitive to initial value

Random Walk Example





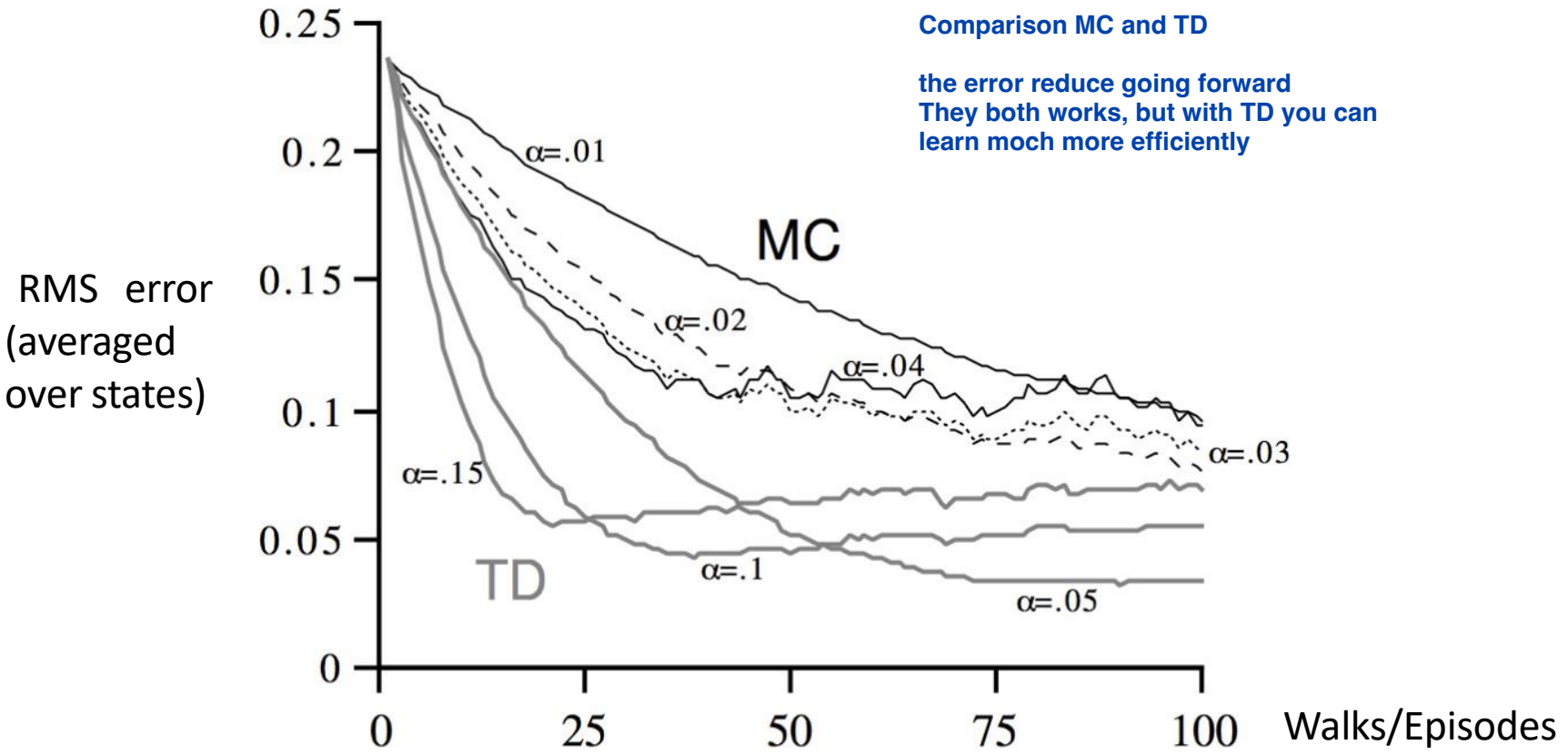
TD0

Actions = go right, go left

number of episodes

more episodes means better estimation

Random Walk Example – MC vs TD



Comparison MC and TD

the error reduce going forward
They both works, but with TD you can learn moch more efficiently

Batch MC and TD

- MC and TD converge: $\mathrm{V}(s) \rightarrow v_\pi(s)$ as experience $\rightarrow \infty$
- But what about batch solution for finite experience?

$$s_1^1, a_1^1, r_2^1, \dots, s_T^1$$

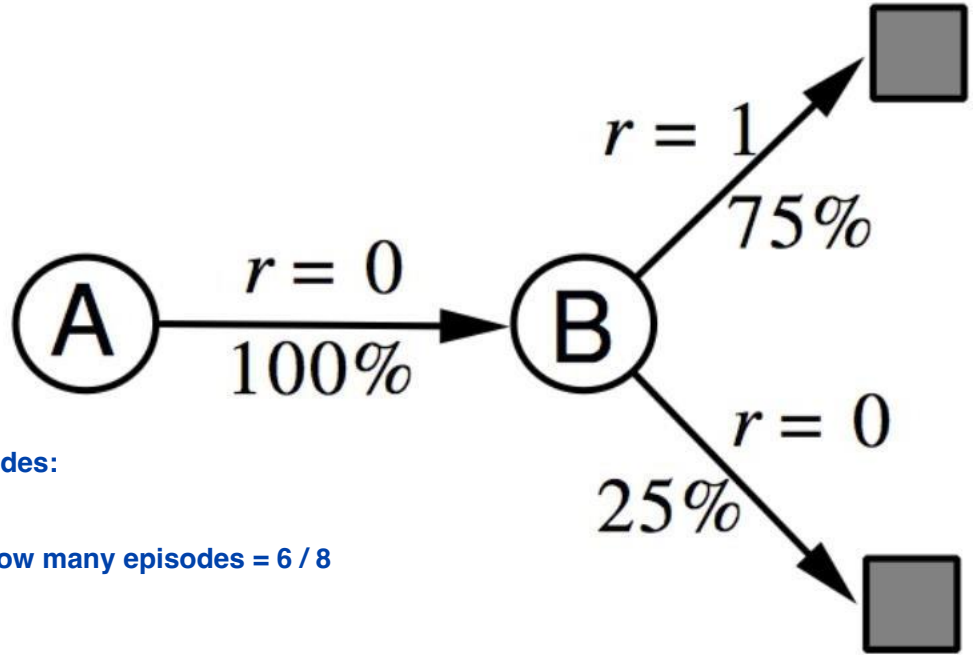
$$\vdots$$

$$s_1^K, a_1^K, r_2^K, \dots, s_T^K$$

  - e.g. repeatedly sample episode $k \in [1, K]$
  - Apply MC or TD(0) to episode k

A Simple Example

- Two states A; B; no discounting; 8 episodes of experience
  1. A, 0, B, 0
  2. B, 1
  3. B, 1
  4. B, 1
  5. B, 1
  6. B, 1
  7. B, 1
  8. B, 0
- What is V (A); V (B)?



eight different episodes:
six with 1 reward

sum the rewards / how many episodes = 6 / 8

Value faction For A:
In MC ->

IN TD -> immediate + the following
0 + 6/8 = 6/8

Certainty Equivariance

- MC converges to solution with minimum mean-squared error
  - Best fit to the observed returns

$$\sum_{k=1}^{K}\sum_{t=1}^{T_k}\left(G_t^k - V(s_t^k)\right)^2$$

- TD(0) converges to solution of maximum likelihood Markov model
  - Solution to the MDP $\langle \mathsf{S}, P, \boldsymbol{P}, R, \gamma \rangle$ that best fits the data

tries to undestand the underlying MDP

$$\hat{P}_{ss'}^a = \frac{1}{N(s,a)}\sum_{k=1}^{K}\sum_{t=1}^{T_k}\mathbf{1}\left(s_t^k, a_t^k, s_{t+1}^k; s, a, s'\right)$$

$$\hat{\mathcal{R}}_s^a = \frac{1}{N(s,a)}\sum_{k=1}^{K}\sum_{t=1}^{T_k}\mathbf{1}\left(s_t^k, a_t^k; s, a\right) r_t^k$$

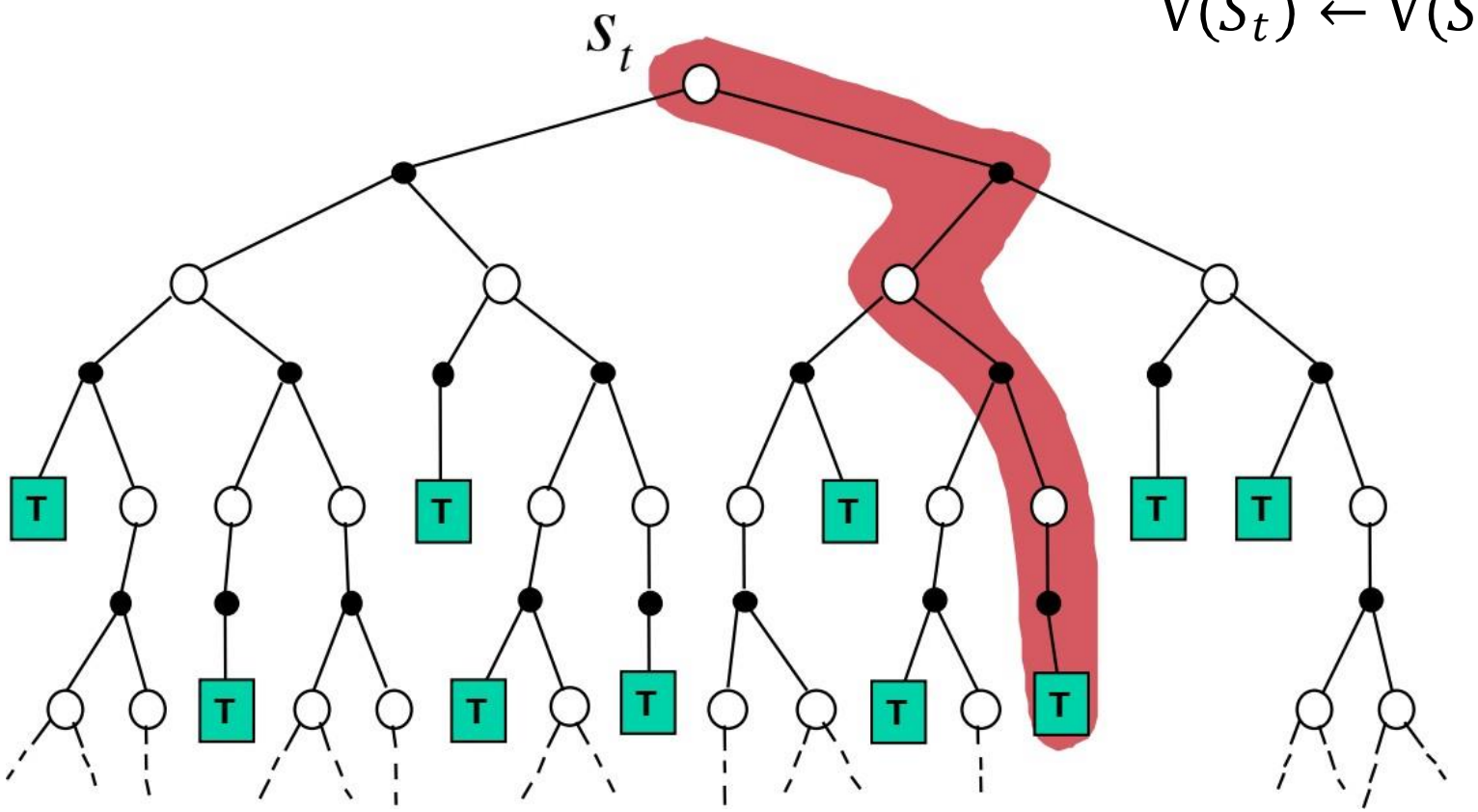
Advantages and Disadvantages of MC vs. TD (III)

- TD exploits Markov property
  - Usually more efficient in Markov environments
- MC does not exploit Markov property
  - Usually more effective in non-Markov environments

Unified View

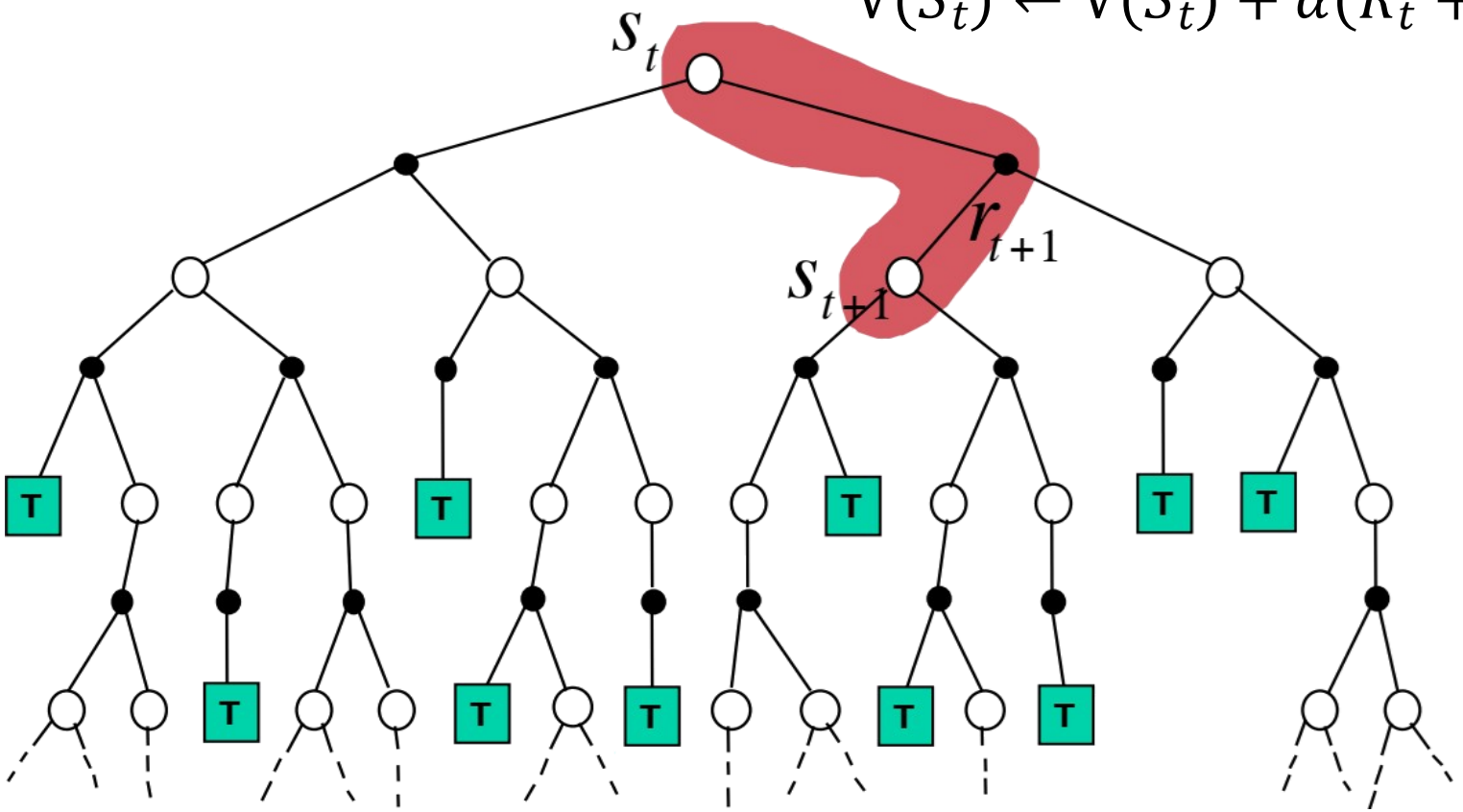
MC Update

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

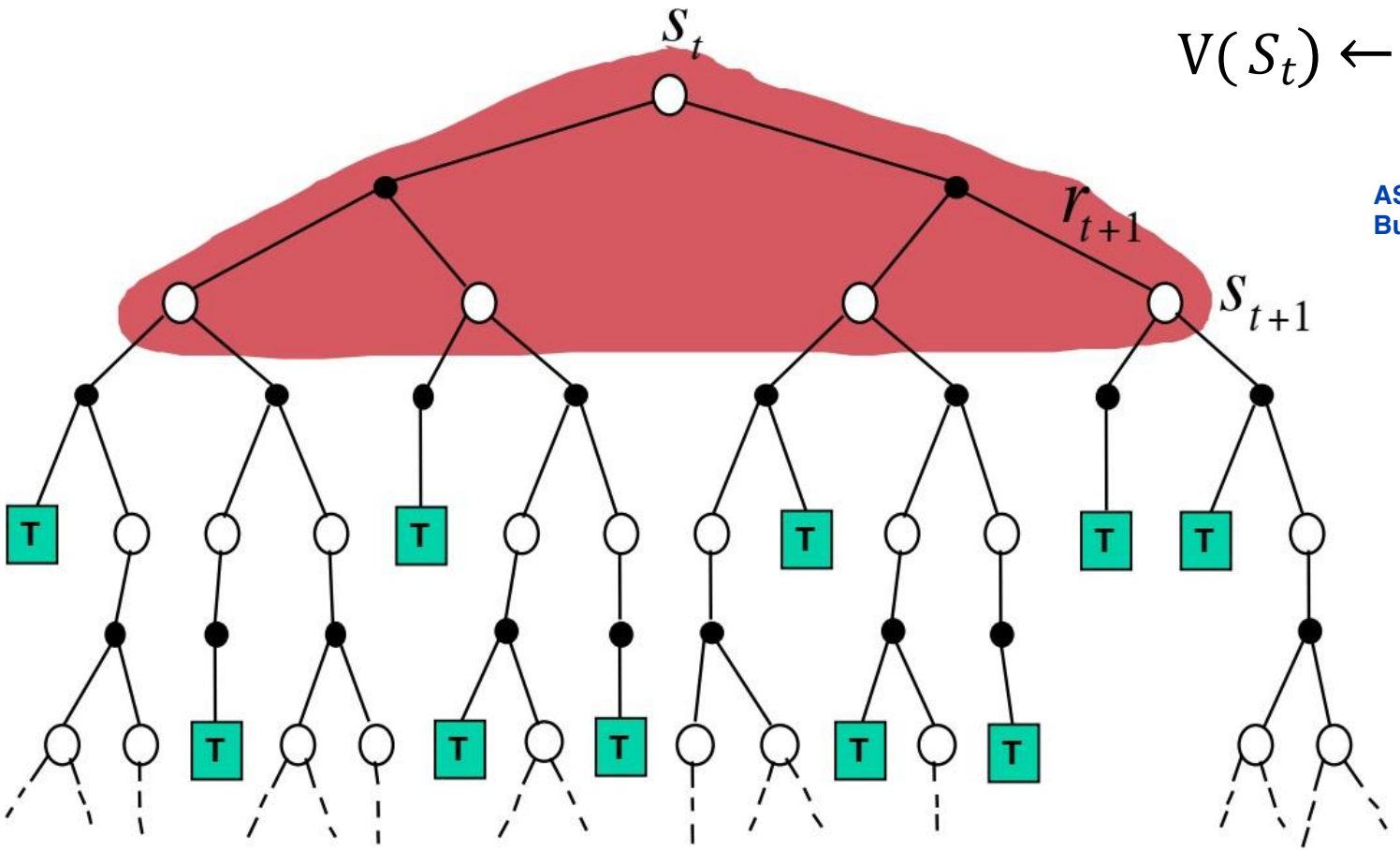


TD Update

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



Dynamic Programming



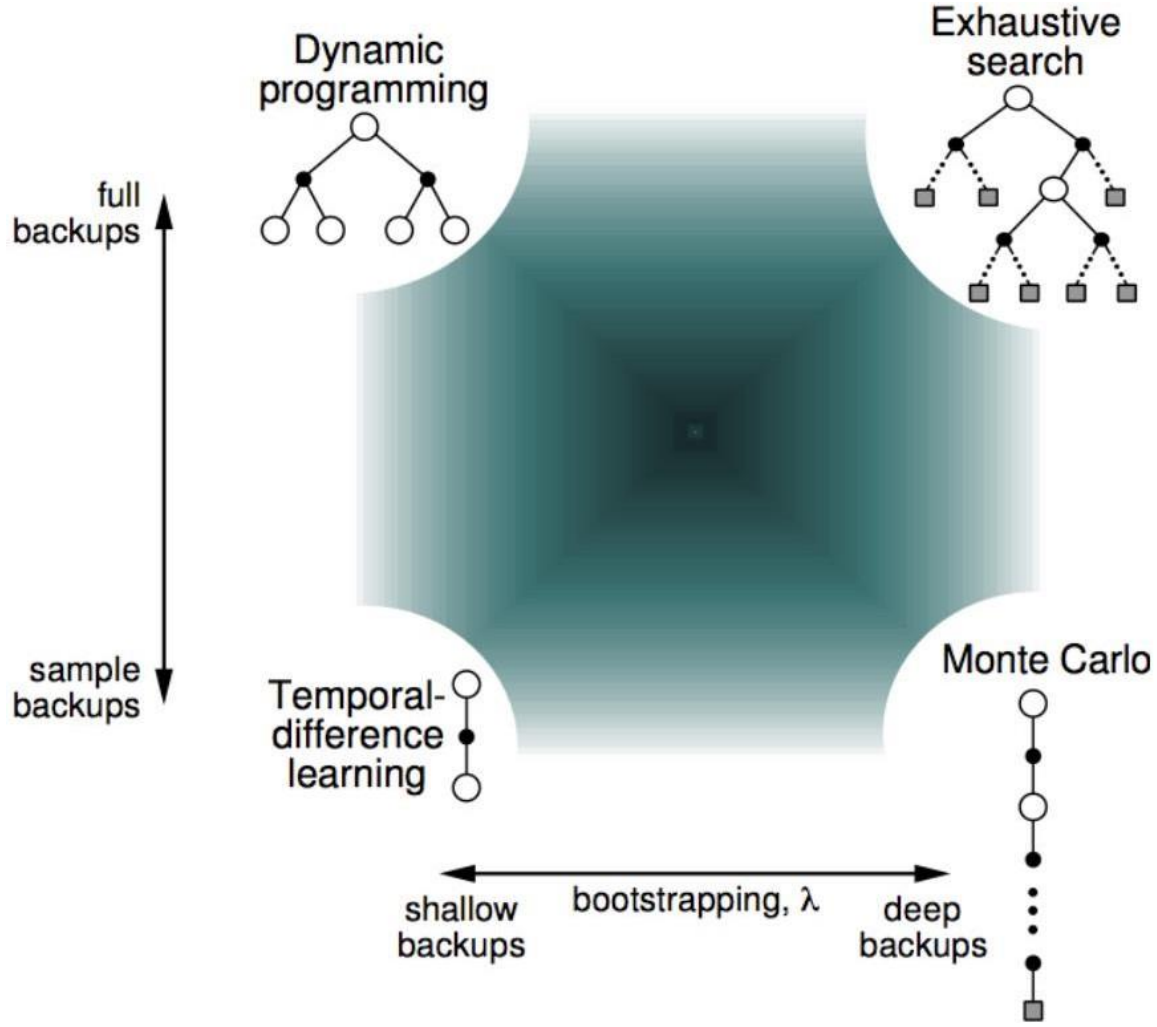
$$V(S_t) \leftarrow \mathbb{E}\left[R_{t+1} + \gamma V(S_{t+1})\right]$$

AS TD, only one step ahead
But I will find all the possible states

Bootstrapping and Sampling

- Bootstrapping - Update involves an estimate
  - MC does not bootstrap
  - DP bootstraps
  - TD bootstraps
- Sampling - Update samples an expectation
  - MC samples
  - DP does not sample
  - TD samples

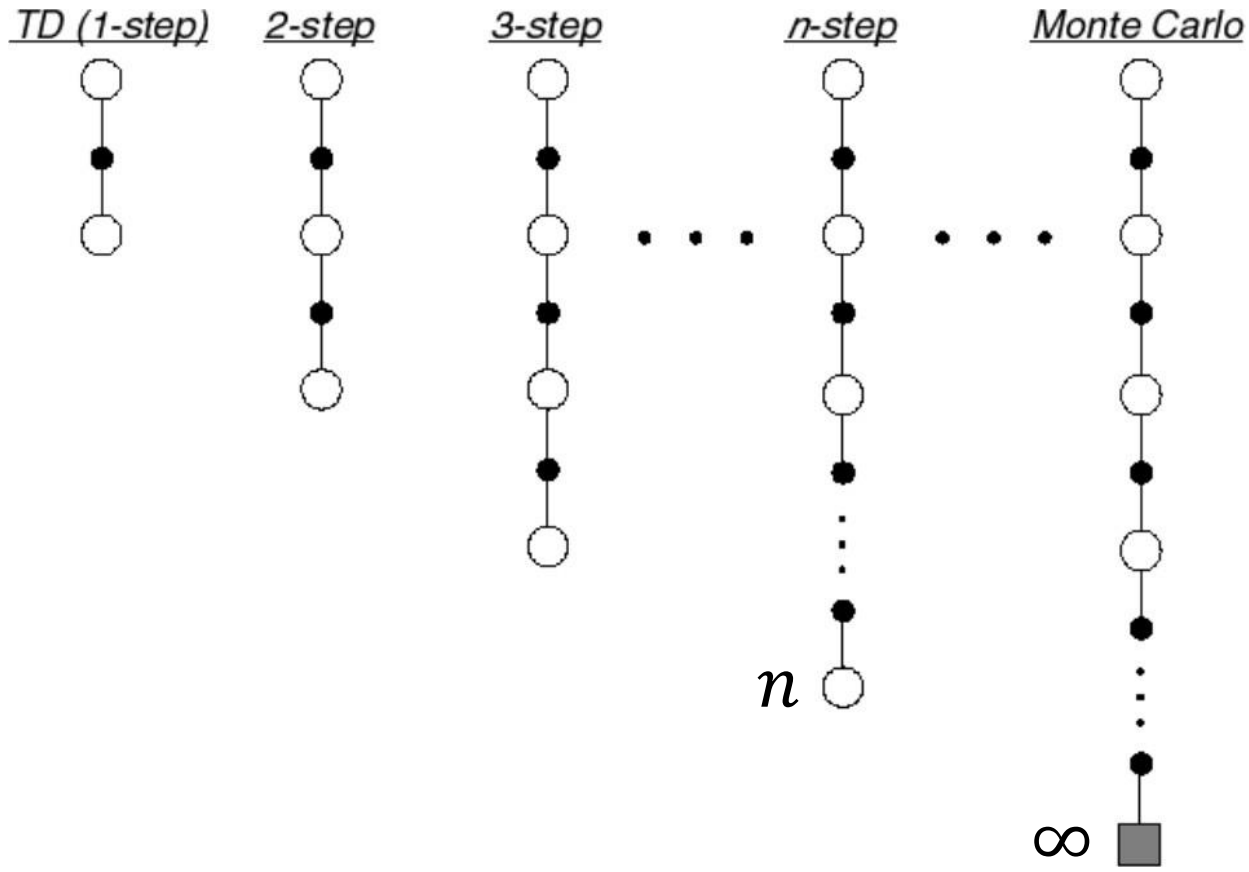
Unified View of RL



Generalizing TD

n-step Prediction

Have TD look and target *n* steps in the future



n-step Return

- Consider the following n -step returns for $n = 1,2, \dots, \infty$

$$n = 1 \quad (\text{TD}) \quad G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1})$$

slightly more accurated than n=1

$$n = 2 \qquad G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2})$$

discounted estimated value

$$\dots$$

$$n = \infty \quad (\text{MC}) \quad G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$$

- Define the n-step return

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

- Learn based on the n-step difference

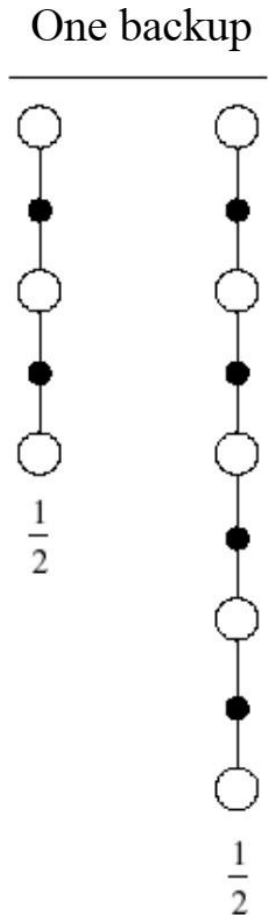
$$\mathrm{V}(S_t) \leftarrow \mathrm{V}(S_t) + \alpha \left(G_t^{(n)} - V(S_t) \right)$$

Averaging n-step Returns

- We can average n-step returns over different n
  - E.g.: Average the 2-step and 4-step returns

$$\frac{1}{2}G^{(2)} + \frac{1}{2}G^{(4)}$$

- Combines information from two different time-steps
- Can we efficiently combine information from all time-steps?



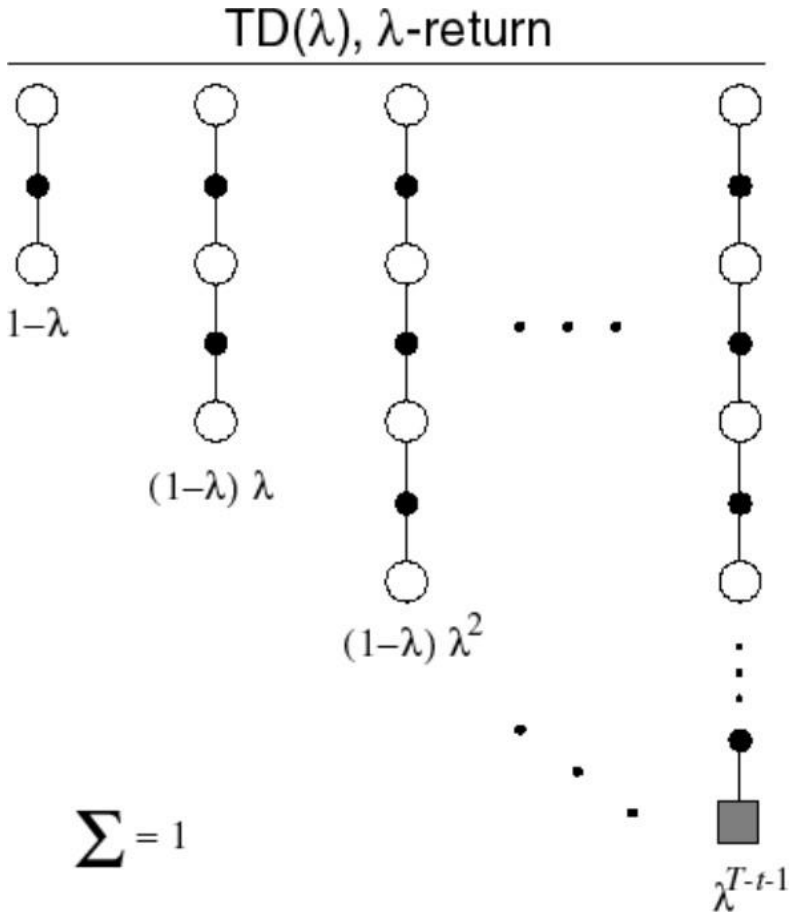
λ-returns

- The λ-return G_t^{λ} combines all n-step returns $G_t^{(n)}$
- Using weight $(1-\lambda)\lambda^{n-1}$

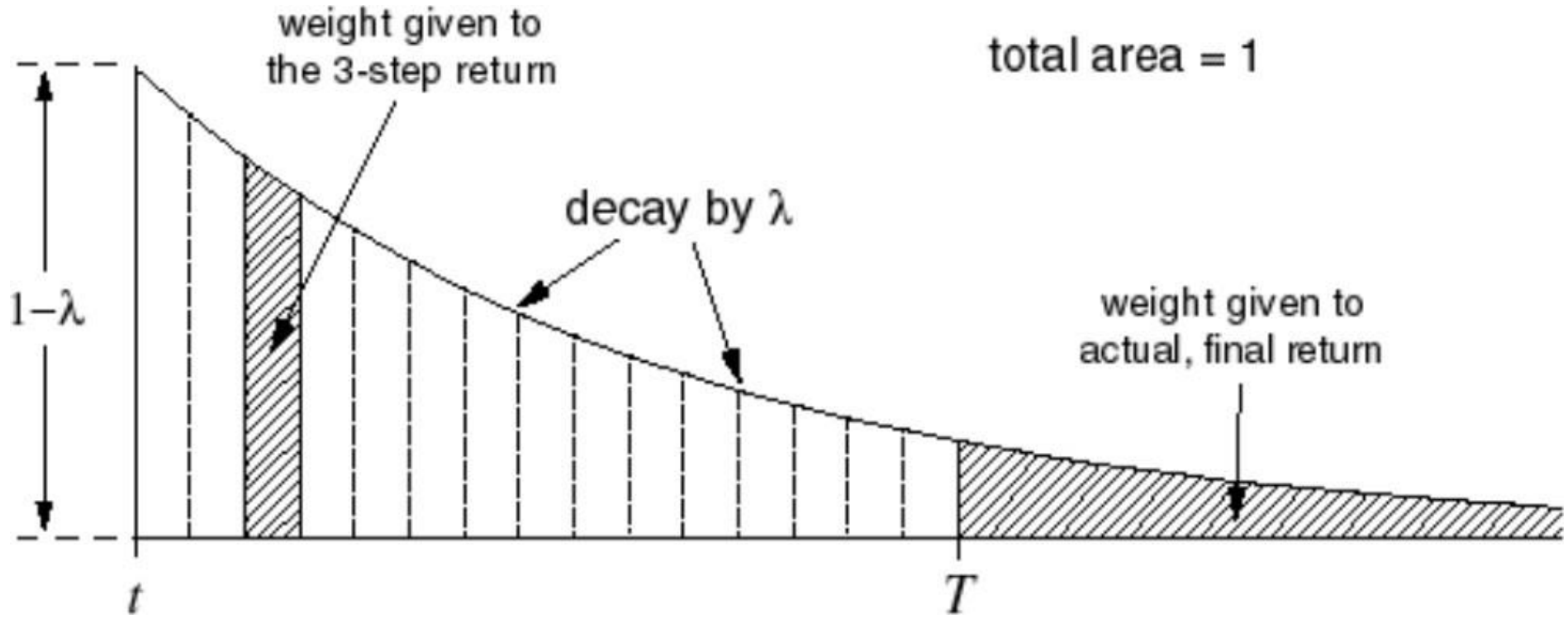
$$G_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{(n)}$$

- Update as appropriate (TD(λ))

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t^{\lambda} - V(S_t))$$

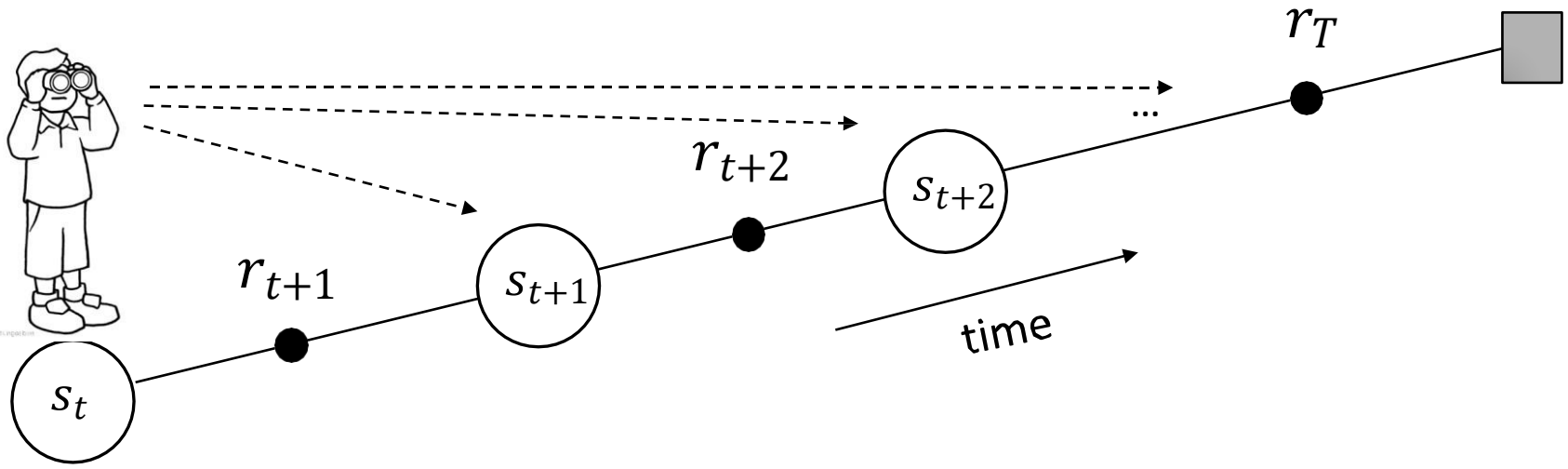


TD(λ) Weight Function



$$G_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$$

Forward View TD(λ)



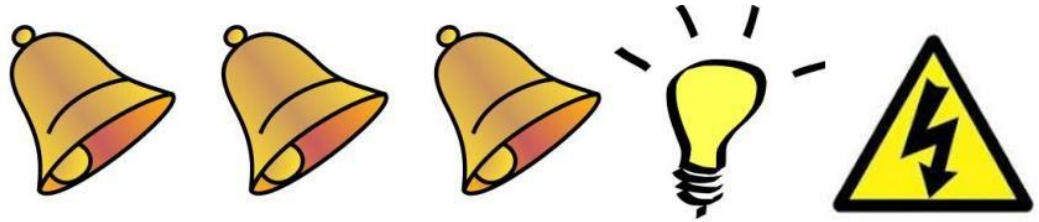
- Update value function towards the λ-return
- Forward-view looks into the future to compute G_t^{λ}
- Like MC, can only be computed from complete episodes

Backward View TD(λ)

- ✓ Forward view provides theory
- ✓ Backward view provides mechanism
- ✓ Update online, every step, from incomplete sequences

Eligibility Traces

euristic measure how much a state can cause

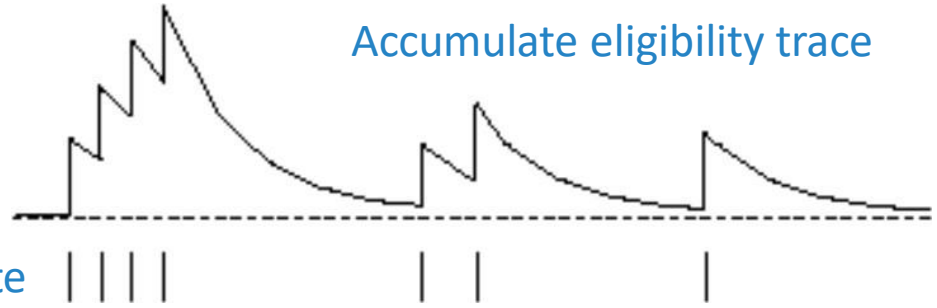


- Credit assignment problem: what caused shock?
  - Frequency heuristic: assign credit to most frequent states the bells
  - Recency heuristic: assign credit to most recent states the light
- Eligibility traces combine both heuristics

In all the sates I've seen have something to do with results

$$E_0(s) = 0$$

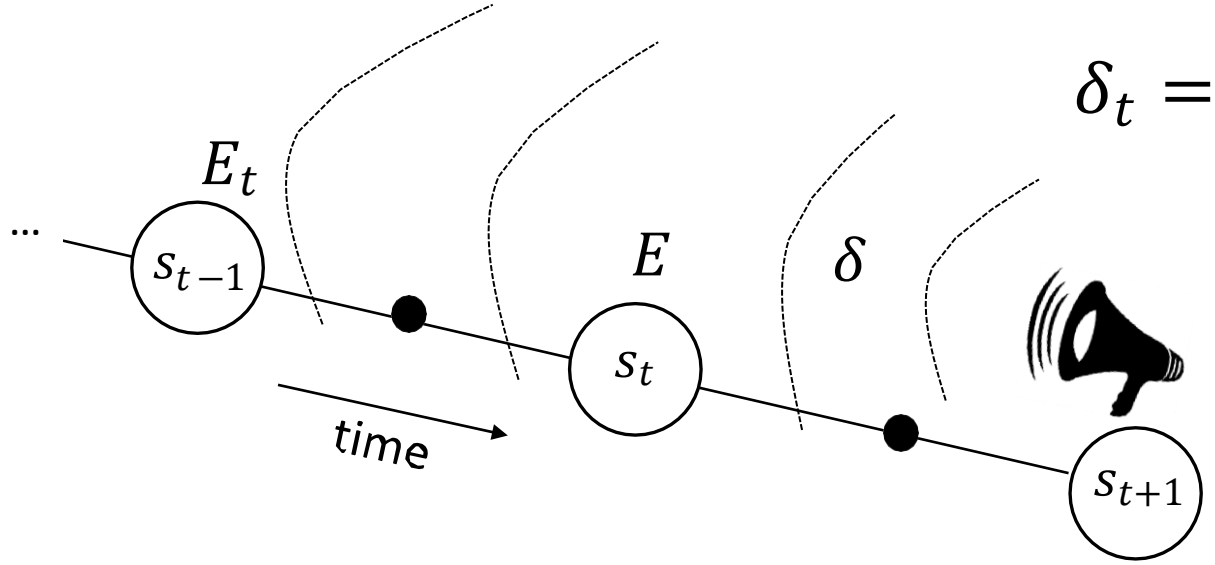
$$E_t(s) = \gamma\lambda E_{t-1}(s) + \mathbf{1}(S_t; s)$$



se non ho viistato quello stato da molto tempo allora prende meno importanza

Backward View TD(λ)

- Keep an eligibility trace for every state s
- Update value $V(s)$ for every state s
- In proportion to TD-error δ_t and eligibility trace $E_t(s)$



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

$$V(s) = V(s) + \alpha \delta_t E\ (s)$$

TD(λ) and TD($\mathbf{0}$)

- When $\lambda = 0$ only current state is updated

$$E_t(s) = \mathbf{1}(S_t; s)$$

$$V(s) \leftarrow V(s) + \alpha\delta_t E_t(s)$$

- Equivalent to TD(0) update

$$V(S_t) \leftarrow V(S_t) + \alpha\delta_t$$

TD(λ) and MC

- When $\lambda = 1$ credit is deferred until end of episode
- Consider episodic environments with offline updates
- Over the course of an episode, total update for TD(1) is the same as total update for MC

Theorem

The sum of offline updates is identical for forward-view and backward-view TD(λ)

$$\sum_{t=1}^{T} \alpha \delta_t E_t(s) = \sum_{t=1}^{T} \alpha \left(G_t^{\lambda} - V(S_t) \right) \mathbf{1}(S_t; s)$$

Telescoping in TD(1)

- When $\lambda = 1$ sum of TD errors telescopes into MC error

$$\dots (proof\ in\ book\ if\ interested) \dots$$

- TD(1) is roughly equivalent to every-visit Monte-Carlo
- Error is accumulated online, step-by-step
- If value function is only updated offline at end of episode, then total update is the same as MC

Wrap-up

Take home messages

- Model-free prediction is value function estimation of an unknown MDP
  - Based on sample-updates
- Monte Carlo methods
  - Estimating value function by averaging sample returns
  - Only for episodic tasks (eventually terminate no matter what actions are taken)
- Temporal-Difference (TD) learning
  - Learn from existing (biased) estimates of future return (bootstrapping)
  - Explore the future until n-th step

Model-Free Control

Outline

- Introduction
- On-policy Vs Off-Policy
- On-policy Monte-Carlo
- On-policy Temporal-Difference learning (SARSA)
- Off-policy Temporal-Difference (Q-learning)

Introduction

Model Free Control – Where to find it

- ✓Elevator
- ✓Robot walking
- ✓Vehicle Steering
- ✓Bioreactor
- ✓Molecule engineering
- ✓Robocup Soccer
- ✓Quake
- ✓Portfolio management
- ✓Protein Folding
- ✓Game of Go

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

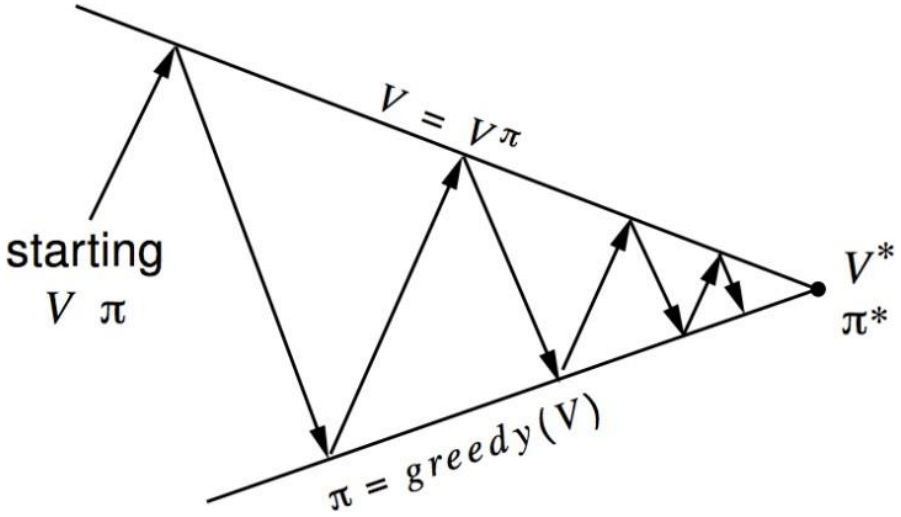
Model-free control can solve these problems

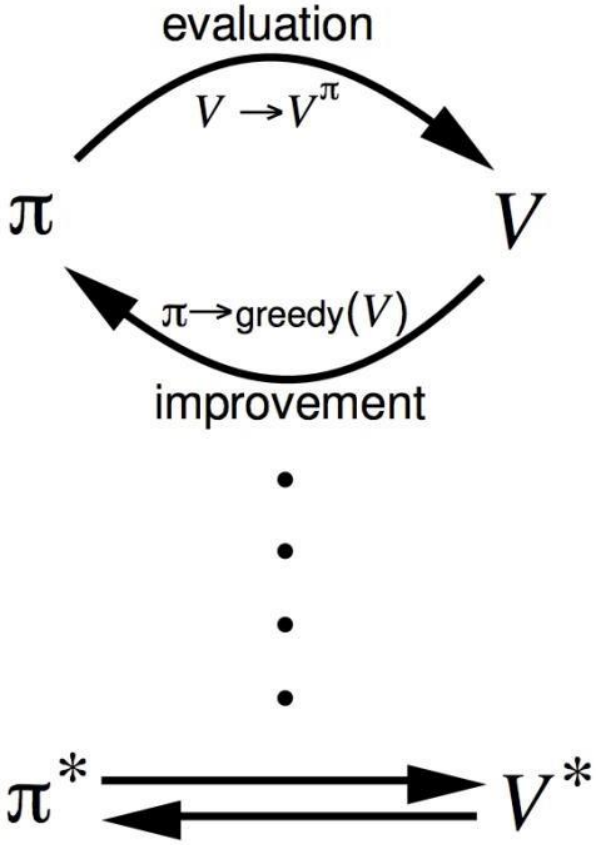
On-policy & Off-policy Learning

- 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

On-Policy Monte-Carlo

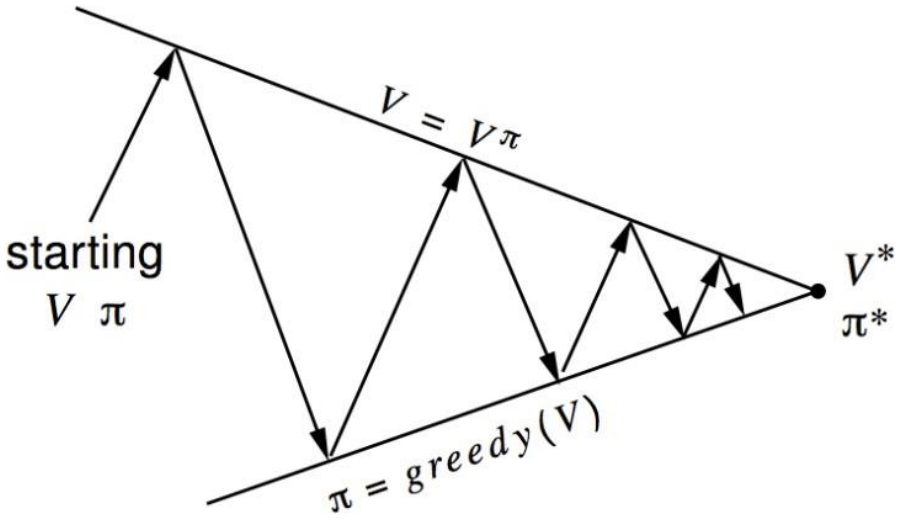
Generalized Policy Iteration (Lecture 5b)





- Policy evaluation - Estimate v_{π}
  - Any policy evaluation
- Policy improvement - Generate $\pi' \geq \pi$
  - Any policy improvement algorithm

Generalized Policy Iteration with On-policy MC



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

Model-Free Policy Iteration Using Action-Value Function

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

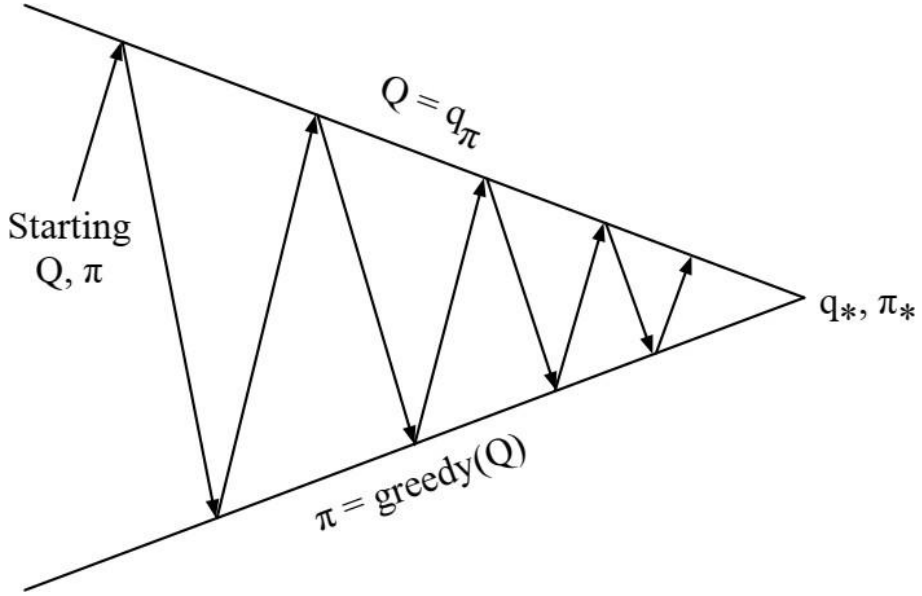
$$\pi'(s) = arg\max_{a \in \mathcal{A}} R_s^a + P_{ss'}^a V(s')$$

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

we need to evaluate a pair of states and action

$$\pi'(s) = arg\max_{a \in \mathcal{A}} Q(s,a)$$

Generalized Policy Iteration with Action-Value Function



- Policy evaluation - Monte-Carlo policy evaluation, $Q = q_\pi$
- Policy improvement - Generate Greedy policy improvement?

Example of Greedy Action Selection

There are two doors in front of you.

- You open the left door and get reward 0 – V (left) = 0
- You open the right door and get reward +1 - V (right) = +1
- You open the right door and get reward +3 - V (right) = +2
- You open the right door and get reward +2 - V (right) = +2

...

- Are you sure you've chosen the best door?

scegli sempre quella che ti ha dato il reward maggiore quindi qui vai a scegliere sempre la destra

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

Qualche volta, randomicamente, scegli un'altra possibilità invece che quella migliore per evitare di rimanere bloccato in un loop sempre ugualòe

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

ϵ-greedy Policy Improvement

Theorem

For any ϵ-greedy policy π, the ϵ-greedy policy π' with respect to q_π is an improvement

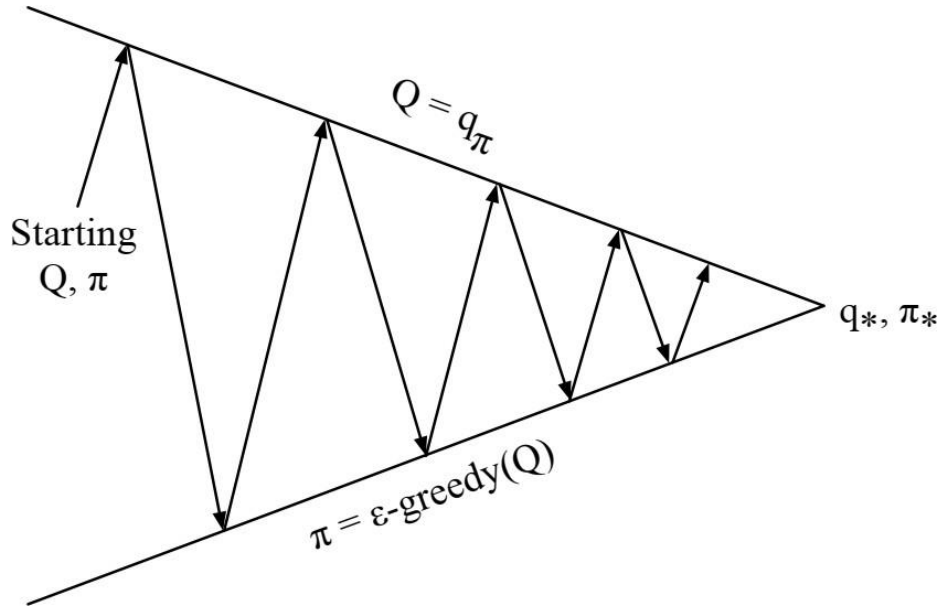
$$v_{\pi'}(s) \geq v_\pi(s)$$

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

Therefore from policy improvement theorem

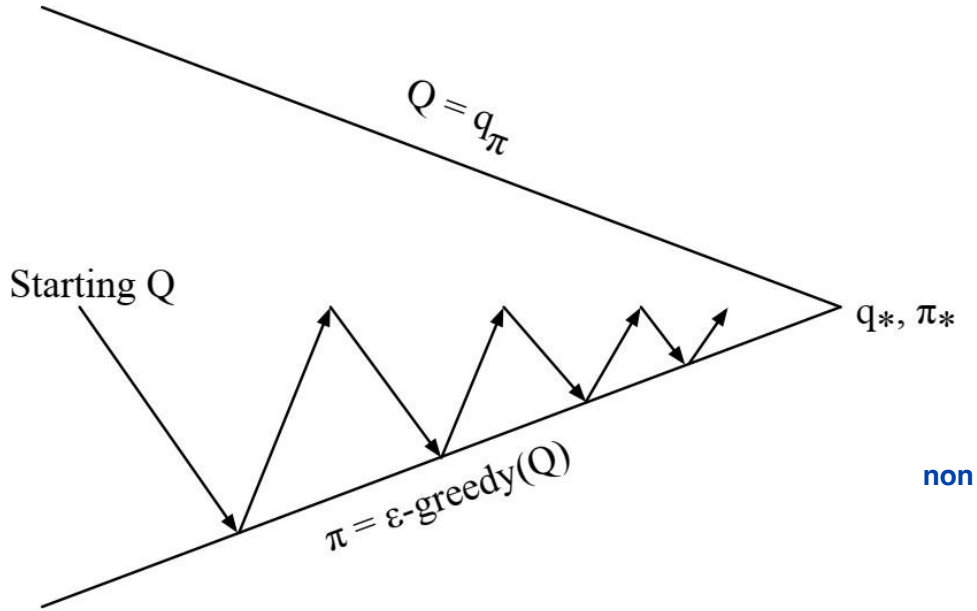
$$v_{\pi'}(s) \geq v_\pi(s)$$

Monte-Carlo Policy Iteration



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

Monte-Carlo Control



non è necessario controllare ogni state?

Every Episode

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

Greedy in the Limit with Infinite Exploration (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

$$\lim_{k\to\infty} \pi_k(a|s) = \mathbf{1}\left(a; \arg\max_{a'\in\mathcal{A}} Q_k(s,a')\right)$$

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

GLIE Monte Carlo Control

if you have enought episodes with statistics you will converge

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

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

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)}\left(G_t - Q(S_t, A_t)\right)$$

- Improve policy based on new action-value function

$$\epsilon \leftarrow \frac{1}{k}$$

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

Theorem

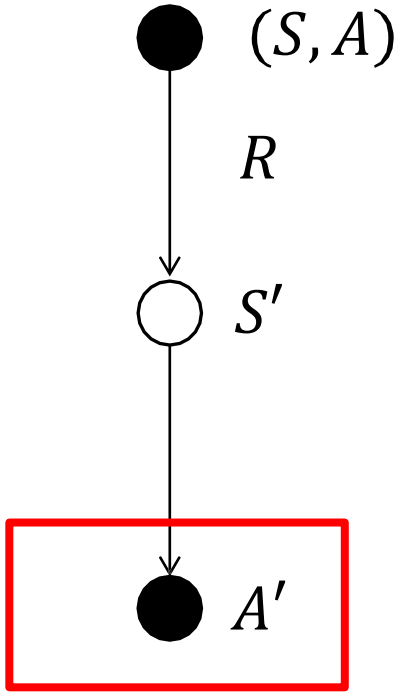
GLIE Monte-Carlo control converges to the optimal action-value function $Q(s, a) \rightarrow q_*(s, a)$

On-Policy TD Control

MC Vs TD Control

- TD learning has several advantages over MC
  - Lower variance
  - Online
  - Incomplete sequences
- Straightforward intuition - Use TD instead of MC in our control loop
  - Apply TD to $Q(s, a)$
  - Use ϵ-greedy policy improvement
  - Update every time-step

Updating Action-Value Functions with SARSA



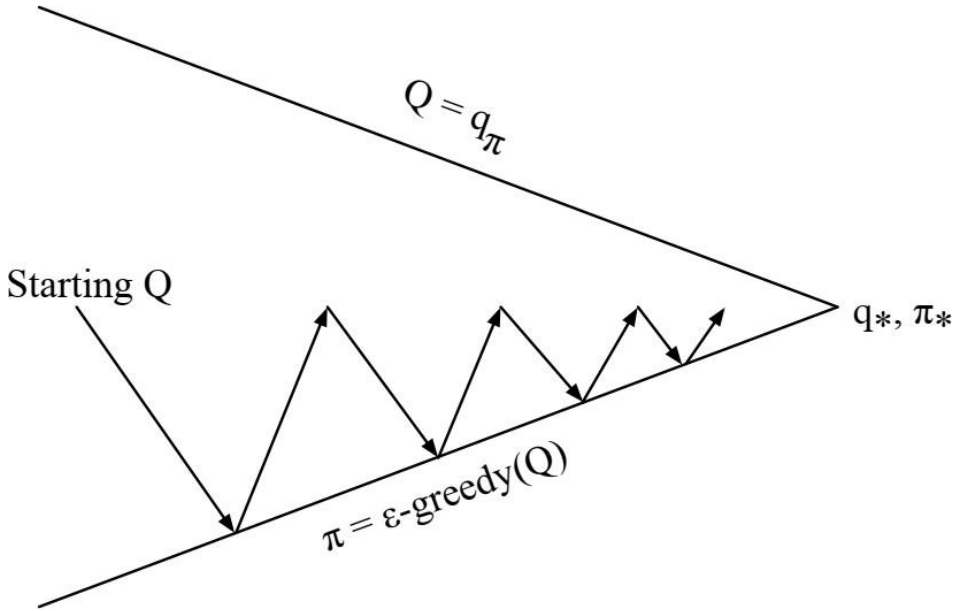
Expected SARSA

$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{a' \in \mathcal{A}} \pi(a'|s') q_*(s',a')$$

We sample also future action A'
(instead of leveraging policy to compute expectation)

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

On-Policy Control with SARSA



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 $Q \approx q_*$

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 of SARSA

Limitazioni:

step size devono essere sufficientemente grandi

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

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

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

Time for TD Demo

https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_td.html

SARSA(λ)

n-step SARSA

- Consider the following n -step returns for $n = 1,2, \dots, \infty$

$$n = 1 \quad \text{(SARSA)} \quad q_t^{(1)} = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$$
$$n = 2 \qquad q_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 Q(S_{t+2}, A_{t+2})$$
$$\dots$$
$$n = \infty \quad \text{(MC)} \quad q_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$$

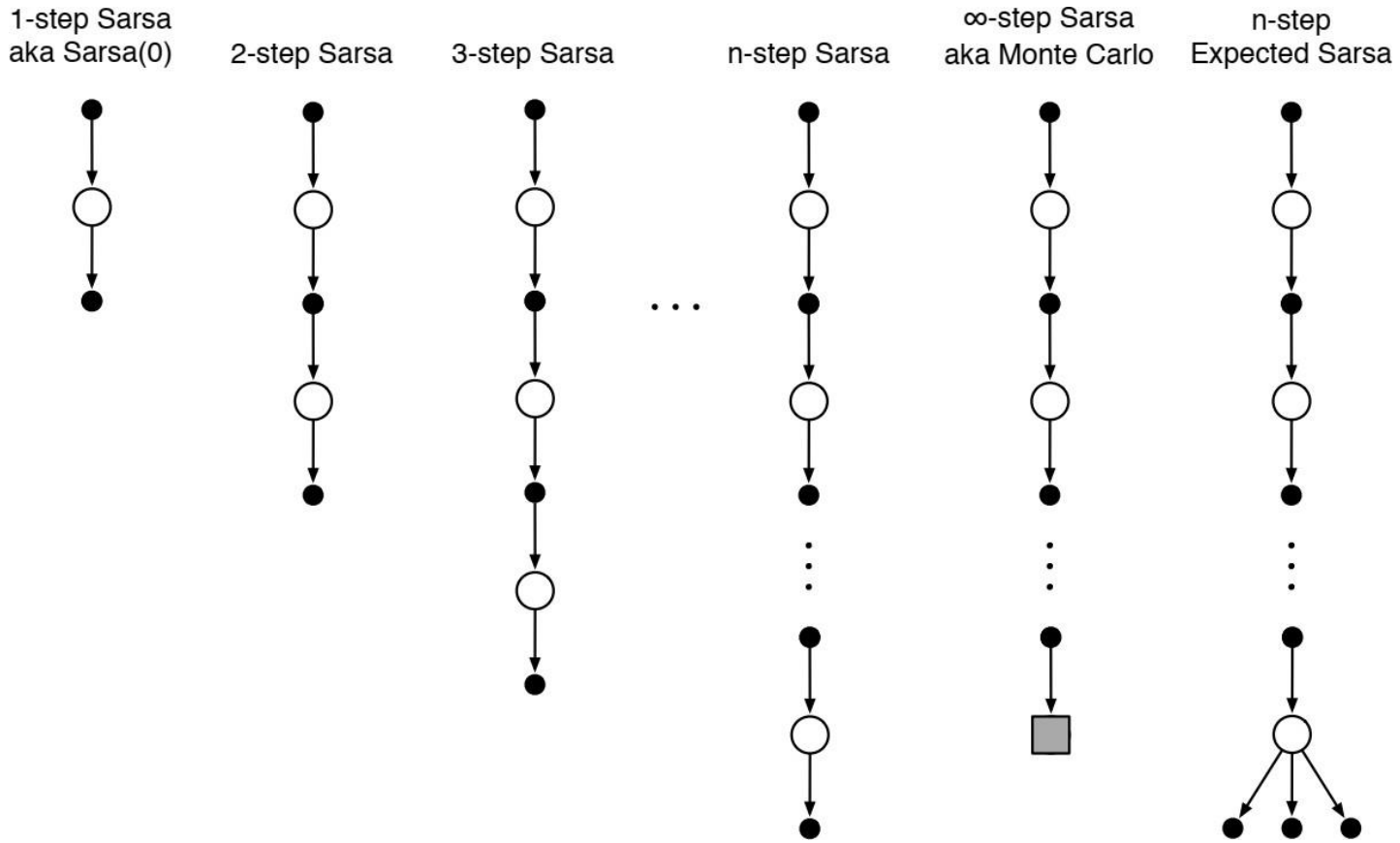
- Define the n-step Q-return

$$q_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n}, A_{t+n})$$

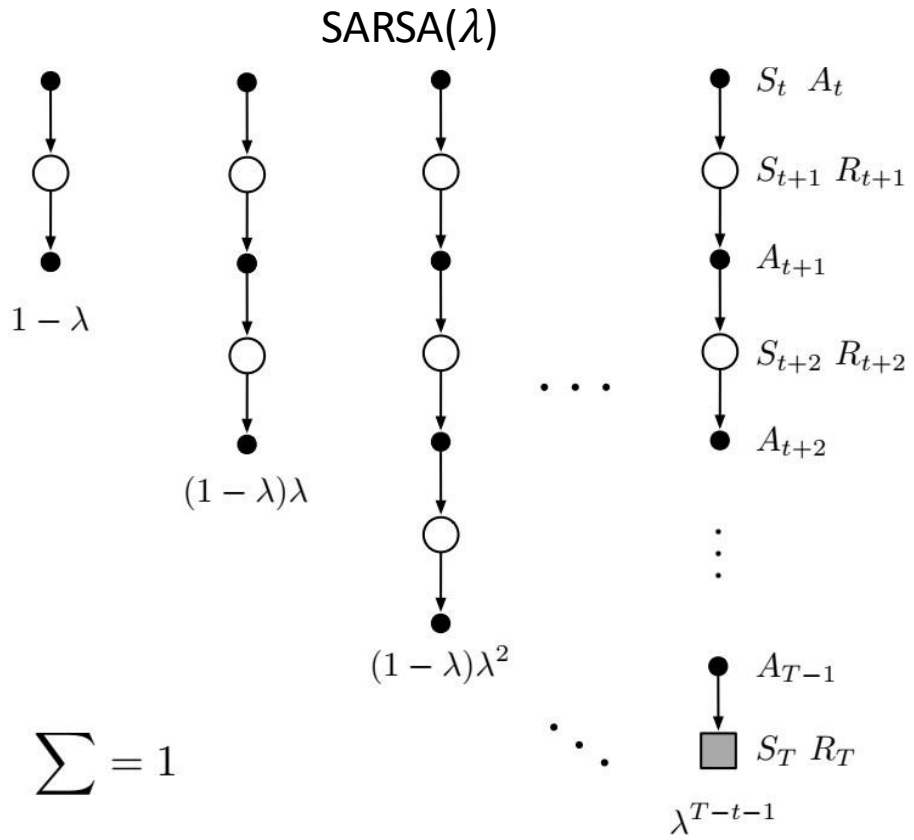
- n-step SARSA updates $Q(S, A)$ towards the n-step Q-return

$$Q(S, A) \leftarrow Q(S, A) + \alpha \left(q_t^{(n)} - Q(S, A) \right)$$

SARSA backups



SARSA(λ) - Forward View



- ✓ The q^λ return combines all n-step Q-returns $q_t^{(n)}$
- ✓ Using weight $(1-\lambda)\lambda^{n-1}$

$$q_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}q_t^{(n)}$$

- ✓ Forward SARSA update

$$Q(S,A) \leftarrow Q(S,A) + \alpha(q_t^\lambda - Q(S,A))$$

SARSA(λ) - Backward View

- ✓ The return of eligibility traces

Which actions are responsible for a behaviour

- ✓ SARSA(λ) needs one eligibility trace for each state-action pair

$$E_0(s,a) = 0$$

$$E_t(s,a) = \gamma\lambda E_{t-1}(s,a) + \mathbf{1}(S_t, A_t; s, a)$$

- ✓ $Q(s,a)$ is updated for every state s and action a in proportion to TD-error δ_t and eligibility trace $E_t(s,a)$

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

$$Q(s,a) \leftarrow Q(s,a) + \alpha\delta_t E_t(s,a)$$

SARSA(λ) Algorithm

enviroments will give me the next state and the reward

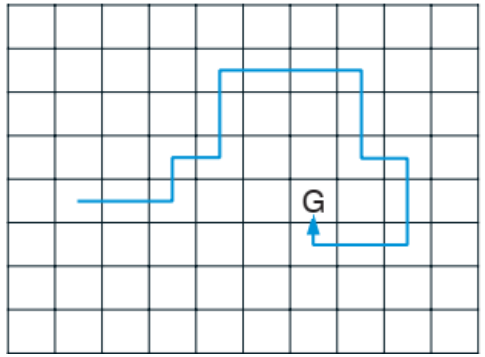
Initialize $Q(s, a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Repeat (for each episode):
  $E(s, a) = 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
  Initialize S, A
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$
    $E(S, A) \leftarrow E(S, A) + 1$
    For all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:
      $Q(s, a) \leftarrow Q(s, a) + \alpha \delta E(s, a)$
      $E(s, a) \leftarrow \gamma \lambda E(s, a)$
    $S \leftarrow S'$; $A \leftarrow A'$
  until S is terminal

SARSA(λ) on Gridworld

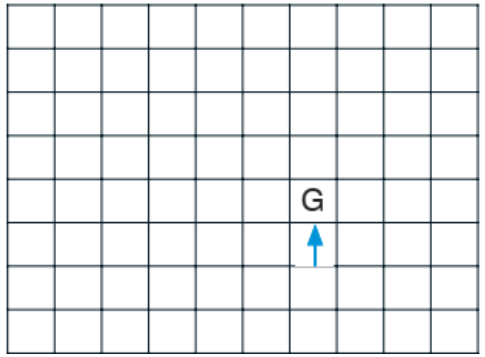
Reward of 1 only when go to G, 0 otherwise

propagate the information thanks to lamba until is 0

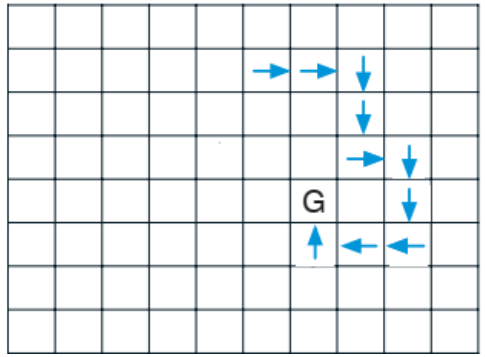
Path taken



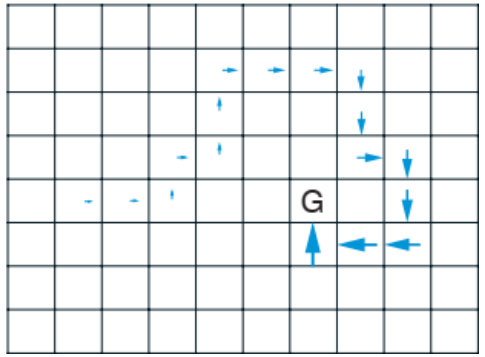
Action values increased by one-step Sarsa



Action values increased by 10-step Sarsa



Action values increased by Sarsa(λ) with λ=0.9



Off-policy TD Learning

Off-Policy Learning

- Evaluate target policy $\pi(a|s)$ to compute $\mathrm{v}_\pi(s)$ or $\mathrm{q}_\pi(s,a)$
- While following behaviour policy $\mu(a|s)$
  - $\{S_1, A_1, R_2, \dots, S_T\} \sim \mu$

policy that somebody else gave us

- Why is this important?
  - Learn from imitation (humans, other agents,…)
  - Re-use experience generated from old policies $\pi_1, \pi_2, \dots, \pi_{t-1}$
  - Learn about optimal policy while following exploratory policy
  - Learn about multiple policies while following one policy

Importance Sampling

Estimate the expectation leveraging an external (importance) distribution

$$\mathbb{E}_{X\sim P}[f(X)] = \sum P(X)f(X)$$

$$= \sum Q(X)\frac{P(X)}{Q(X)}f(X)$$

$$= \mathbb{E}_{X\sim Q}\left[\frac{P(X)}{Q(X)}f(X)\right]$$

Draw samples from importance distribution $Q(X)$ rather than from $P(X)$

Assign weights such that the empirical expectation (on $Q(X)$ samples) matches the expectation under $P(X)$

Importance Sampling for Off-Policy Monte Carlo

two policy

- Use returns generated from μ to evaluate π
- Weight return G_t according to similarity between policies
- Multiply importance sampling corrections along whole episode

$$G_t^{\pi/\mu} = \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)} \frac{\pi(A_{t+1}|S_{t+1})}{\mu(A_{t+1}|S_{t+1})} \cdots \frac{\pi(A_T|S_T)}{\mu(A_T|S_T)} G_t$$

- Update value towards corrected return

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t^{\pi/\mu} - V(S_t))$$

- Importance sampling can dramatically increase variance

Not used

Importance Sampling for Off-Policy TD

only one transiction for TD so is feasible

- Use TD targets generated from μ to evaluate π
- Weight TD targets $R + \gamma V(S')$ by importance sampling
- Only need a single importance sampling correction

$$V(S_t) \leftarrow V(S_t) + \alpha \left(\frac{\pi(A_t|S_t)}{\mu(A_t|S_t)} (R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right)$$

- Much lower variance than MC
- Policies only need to be similar over a single step

Q-Learning

Off-policy learning of action-values $Q(s, a)$

- No importance sampling is required
- Next action is chosen using behaviour policy $A_{t+1} \sim \mu(\cdot\,|S_t)$
- But we consider alternative successor action $A' \sim \pi(\cdot\,|S_t)$
- And update $Q(S_t, A_t)$ towards value of alternative action

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t))$$

when I need to compute the TD error

Off-policy Control by Q-Learning

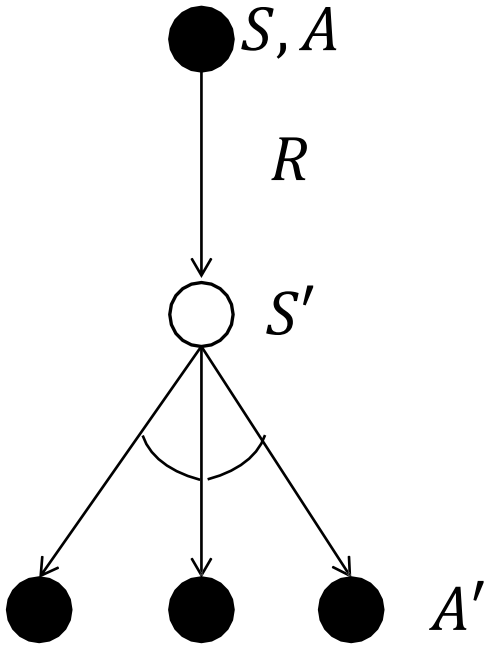
- Allow both behaviour and target policies to improve
- The target policy π is greedy w.r.t. $Q(S_t, A_t)$

$$\pi(S_{t+1}) = \arg\max_{a'} Q(S_{t+1}, a')$$

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

$$\begin{aligned} R_{t+1} + \gamma Q(S_{t+1}, A') &= R_{t+1} + \gamma Q\left(S_{t+1}, \arg\max_{a'} Q(S_{t+1}, a')\right) \\ &= R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a') \end{aligned}$$

Q-Learning Control Algorithm



Theorem

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

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

Q-Learning Algorithm for Off-policy Control

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, 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'

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

    $S \leftarrow S'$;

  until S is terminal

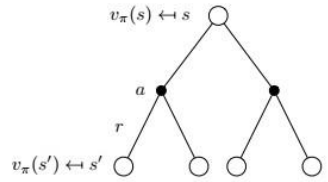
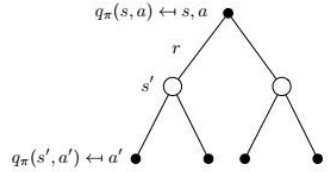
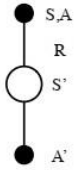
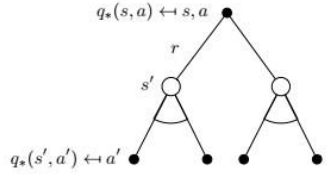
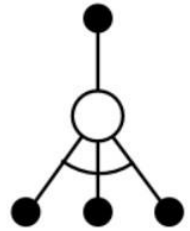
use different according to what to do
when update use the maximum
in altro caso boh :::::::(((((((

Q-learning & Exploration Demo

https://www.aslanides.io/aixijs/demo.html

Wrap-up

Dynamic Programming Vs Temporal Difference Learning

	Full Backup (DP)	*Sample Backup (TD)*
Bellman Expectation Equation for $v_\pi(s)$	 Iterative Policy Evaluation	 TD Learning
Bellman Expectation Equation for $q_\pi(s, a)$	 Q-Policy Iteration	 Sarsa
Bellman Optimality Equation for $q_*(s, a)$	 Q-Value Iteration	 Q-Learning

Take home messages

- Model-Free control leverages action-value function
  - Greedy policy improvement does not need MDP
  - Generalized policy iteration
- Need to maintain sufficient exploration (ϵ-greedy)
- Off-policy control
  - Learning value function of a target policy from data generated by a different behaviour policy
  - Importance sampling to match the expectations of two policies
- TD control
  - On-policy: SARSA(λ)
  - Off-policy: Q-learning

Next Lecture

Value-function approximation

- Leave aside tabular environments
- Estimate value function with function approximation
- Linear models & neural networks
- MC & TD with Stochastic Gradient
- Experience replay buffers