(Planning with) Dynamic Programming

Introduction

Outline

- ✓ Introduction
- ✓ Dynamic programming
- ✓ Policy Evaluation
- ✓ Policy Iteration
- ✓ Value Iteration
- ✓ Advanced topics
  - ✓ Asynchronous update
  - ✓ Approximated approaches

What is dynamic programming

Dynamic $\mapsto$ problem with sequential or temporal component

Programming $\mapsto$ optimising a program, i.e. a policy

- ✓ A method for solving complex problems by breaking them down into subproblems
  - ✓ Solve the subproblems
  - ✓ Combine solutions to subproblems
- ✓ It is not divide-et-impera
  - ✓ Differentiates by overlapping breakdown

Requirements for dynamic programming

- ✓ Optimal substructure
  - ✓ Principle of optimality applies
  - ✓ Optimal solution can be decomposed into subproblems
- ✓ Overlapping subproblems
  - ✓ Subproblems recur many times
  - ✓ Solutions can be cached and reused

Markov decision processes satisfy both properties

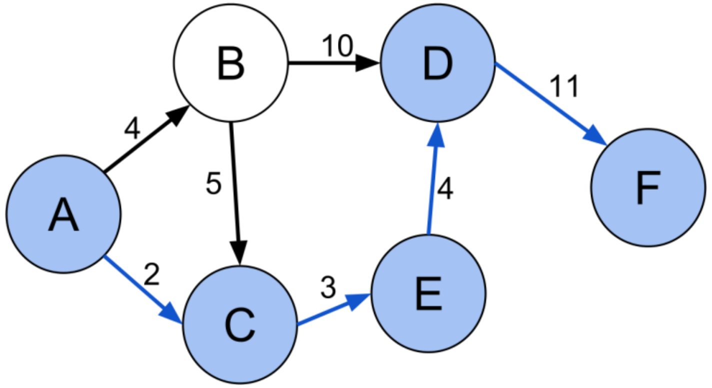
- ✓ Bellman equation gives recursive decomposition
- ✓ Value function stores and reuses solutions

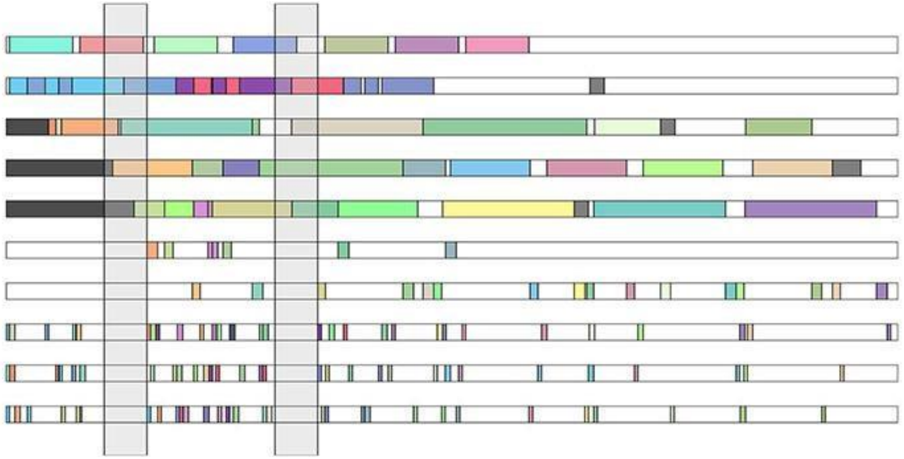
Planning by dynamic programming

- ✓ Dynamic programming assumes full knowledge of the MDP
- ✓ Planning in RL (repetita)
  - ✓ A model of the environment is known
  - ✓ The agent improves its policy

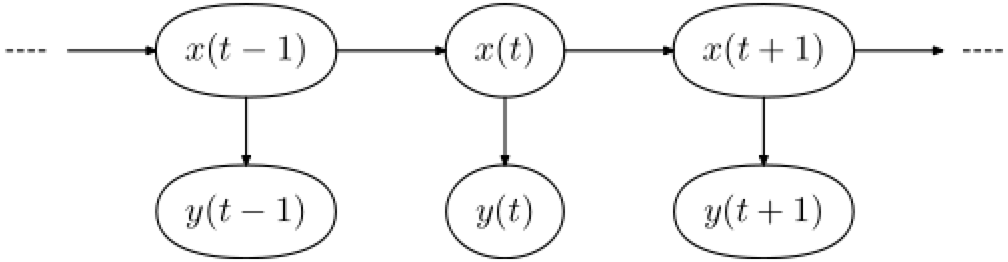
- ✓ Dynamic programming can be used for planning in RL
- ✓ Prediction
  - ✓ Input: MDP $\langle \mathcal{S}, \mathcal{A}, \boldsymbol{P}, \mathcal{R}, \gamma \rangle$ and policy π **or** MRP $\langle \mathcal{S}, \boldsymbol{P}, \mathcal{R}, \gamma \rangle$
  - ✓ Output: value function v_π
- ✓ Control
  - ✓ Input: MDP $\langle \mathcal{S}, \mathcal{A}, \boldsymbol{P}, \mathcal{R}, \gamma \rangle$
  - ✓ Output: optimal value function v_{π_*} **and** optimal policy π_*

Applications of Dynamic Programming





		G	C	C	C	T	A	G	C	G
	0	0	0	0	0	0	0	0	0	0
G	0	1	1	1	1	1	1	1	1	1
C	0	1	2	2	2	2	2	2	2	2
G	0	1	2	2	2	2	2	3	3	3
C	0	1	2	3	3	3	3	3	4	4
A	0	1	2	3	3	3	4	4	4	4
A	0	1	2	3	3	3	4	4	4	4
T	0	1	2	3	3	4	4	4	4	4
G	0	1	2	3	3	4	4	5	5	5



Policy Evaluation

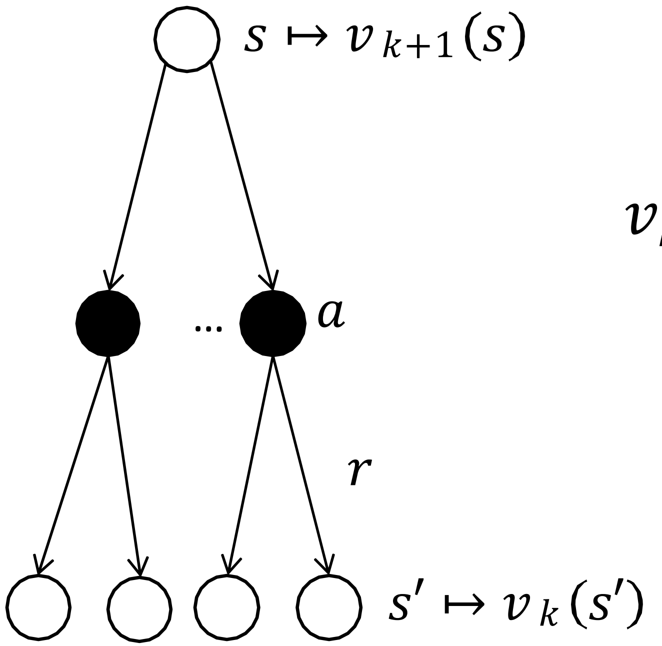
Iterative Policy Evaluation

- ✓ Problem: evaluate a given policy π
- ✓ Solution: iterative application of Bellman expectation backup

$$v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_\pi$$

- ✓ Using synchronous backups
  i. At each iteration $k + 1$
  ii. For all states $s \in \mathcal{S}$
  iii. Update $v_{k+1}(s)$ from $v_k(s')$ where s' is a successor state of s

Iterative Policy Evaluation - Formally



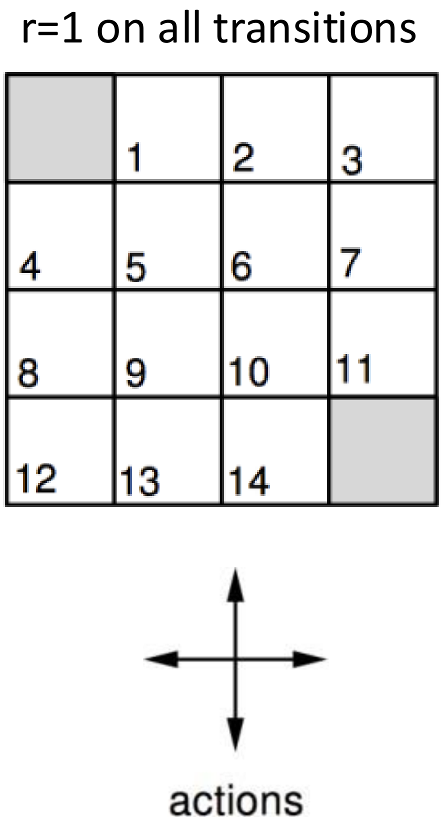
$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathrm{P}_{ss'}^a v_k(s') \right)$$

$$v_{k+1} = \mathcal{R}^\pi + \gamma \boldsymbol{P}^{\boldsymbol{\pi}} v_k$$

Evaluating a Random Policy in the Small Gridworld

- ✓ Undiscounted episodic MPD ($\gamma = 1$)
- ✓ Nonterminal states $1, \dots, 14$
- ✓ One terminal state (shown twice as shaded squares)
- ✓ Actions leading out of the grid leave state unchanged
- ✓ Reward is −1 until the terminal state is reached
- ✓ Agent follows uniform random policy

$$\pi(n|\cdot) = \pi(s|\cdot) = \pi(e|\cdot) = \pi(w|\cdot) = 0.25$$

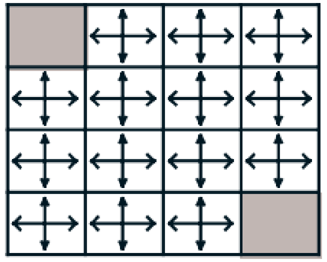


Iterative Policy Evaluation on Small Gridworld (I)

v_k — Greedy policy on v_k

$k = 0$

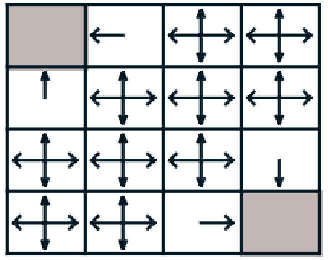
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



random policy

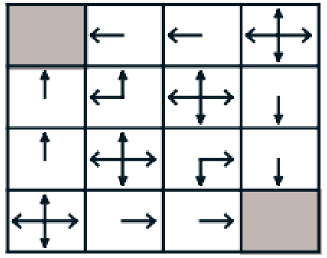
$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0



Iterative Policy Evaluation on Small Gridworld (I)

$k = 3$

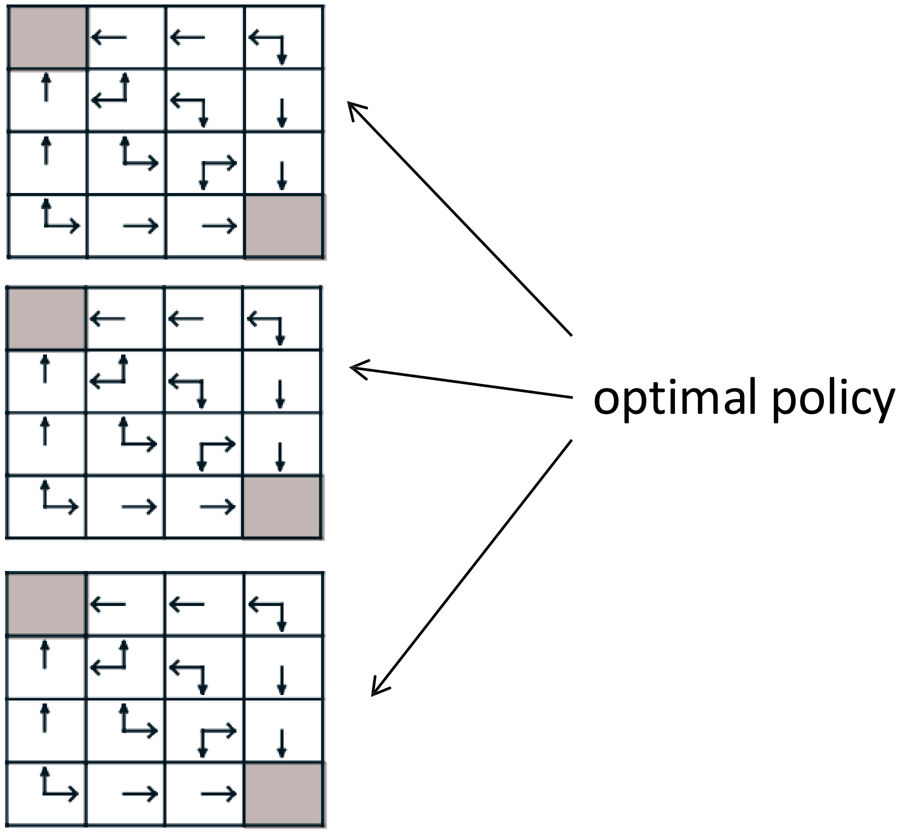
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



Policy Iteration

How to Improve a Policy

- ✓ Given policy π
  - ✓ Evaluate the policy π

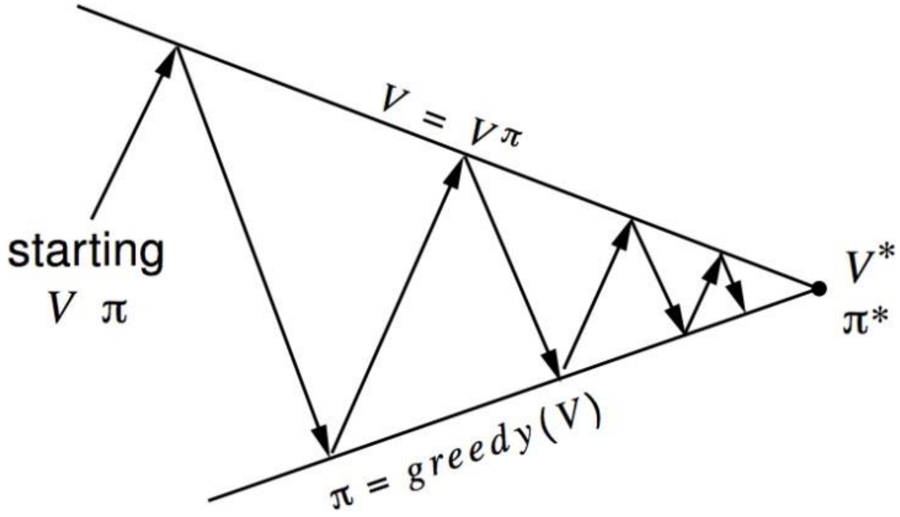
$$v_\pi(s) = \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} \ + \cdots | S_t = s\right]$$

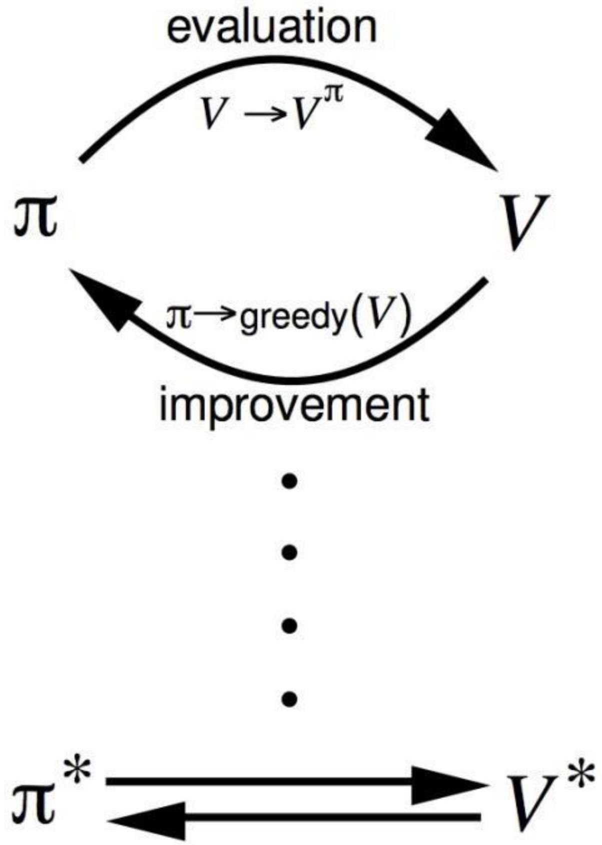
  - ✓ Improve the policy by acting greedily with respect to v_π

$$\pi^{'} = greedy(\pi)$$

- ✓ In Small Gridworld improved policy was optimal, $\pi^{'} = \pi_*$
- ✓ In general, need more iterations of improvement / evaluation
- ✓ But this process of policy iteration always converges to π_*

Policy Iteration

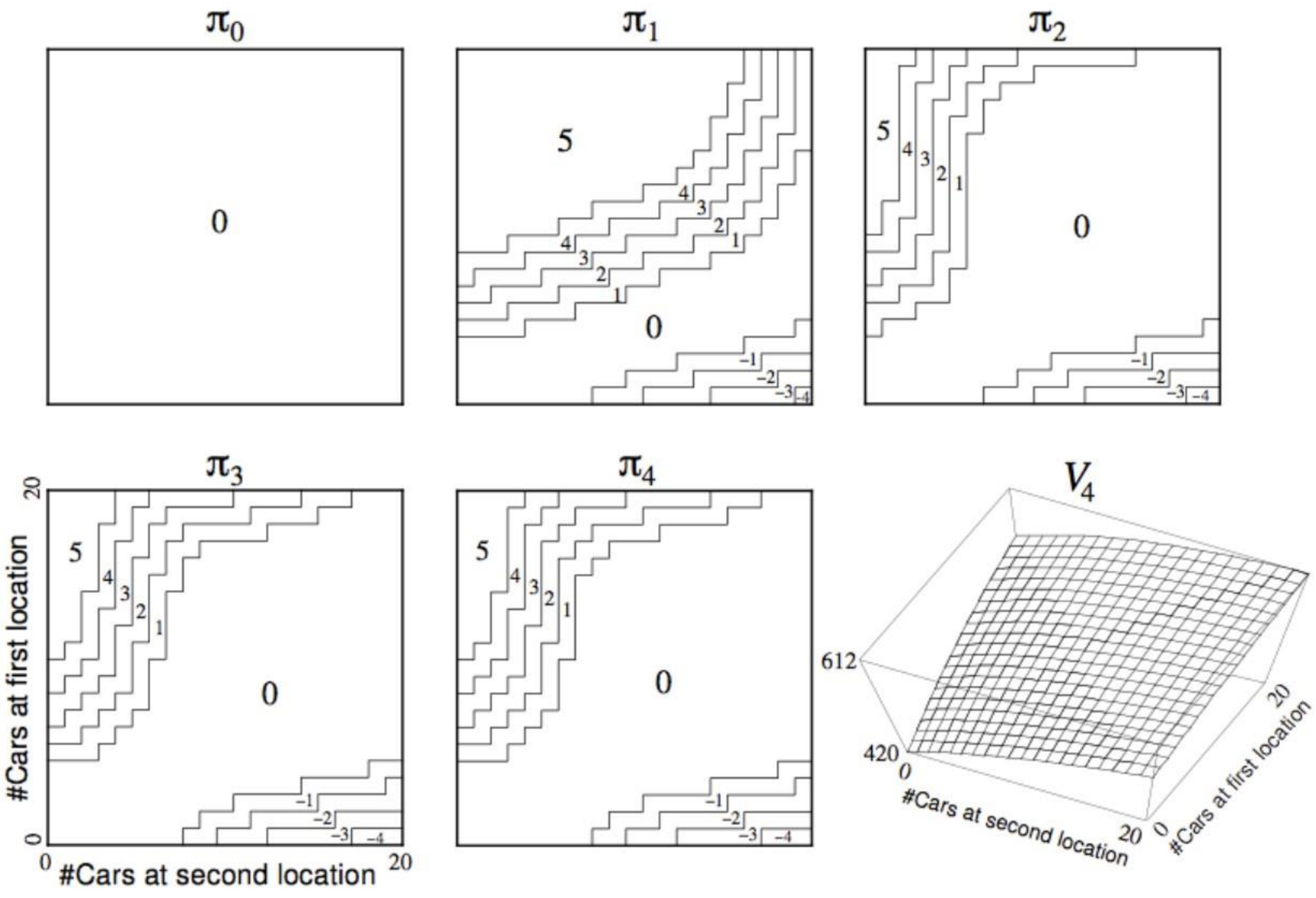




- ✓ Policy evaluation - Estimate v_π
  - ✓ Iterative policy evaluation
- ✓ Policy improvement - Generate $\pi' \geq \pi$
  - ✓ Greedy policy improvement

Jack's Car Rental

- ✓ States - Two locations, maximum of 20 cars at each
- ✓ Actions - Move up to 5 cars between locations overnight
- ✓ Reward - $10 for each car rented (must be available)
- ✓ Transitions - Cars returned and requested randomly
  - ✓ Poisson distribution, n returns/requests $\sim \frac{\lambda^n}{n!} e^{-\lambda}$
  - ✓ 1st location: average requests = 3, average returns = 3
  - ✓ 2nd location: average requests = 4, average returns = 2



Policy Iteration in Jack’s Car Rental

Policy Improvement (I)

Consider a deterministic policy $a = \pi(s)$

We can improve the policy by acting greedily

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

This improves the value from any state s over one step

$$q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) \geq q_\pi\big(s, \pi(s)\big) = v_\pi(s)$$

Therefore improving the value function $v_{\pi'}(s) \geq v_\pi(s)$

Policy Improvement (II)

If improvement stops

$$q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) = q_\pi\big(s, \pi(s)\big) = v_\pi(s)$$

We satisfy Bellman optimality

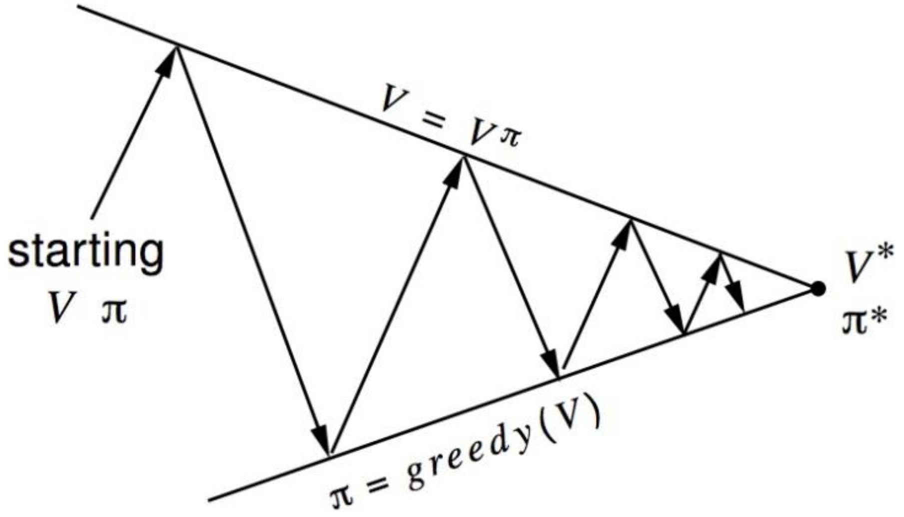
$$v_\pi(s) = \max_{a \in \mathcal{A}} q_\pi(s, a)$$

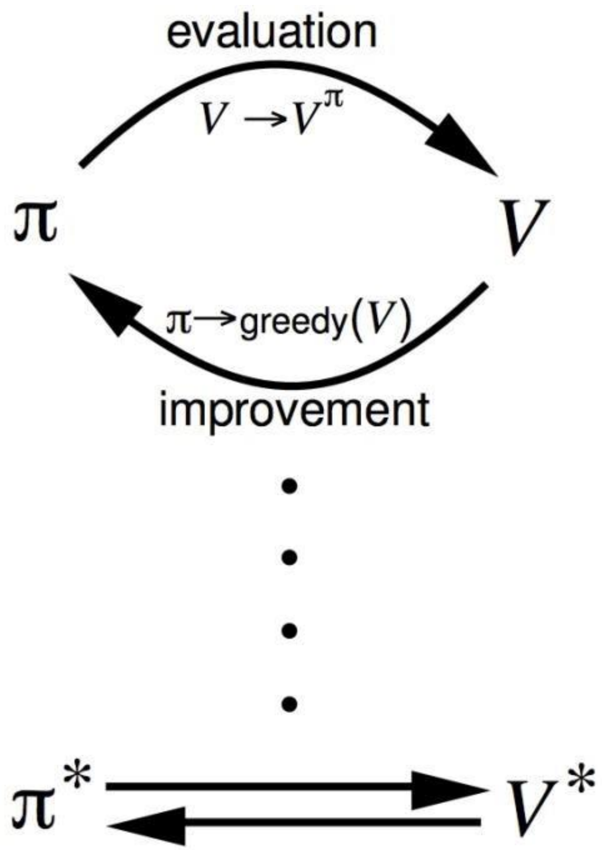
Therefore $v_\pi(s) = v_*(s), \forall s \in \mathcal{S}$, and π is an optimal policy

Modified Policy Improvement

- ✓ Does policy evaluation need to converge to v_{π^*} ?
  - ✓ Introduce a stopping condition, e.g. ϵ-convergence of value function
  - ✓ Stop after k iterations of iterative policy evaluation, e.g. k=3 was sufficient in small gridworld
- ✓ Why update policy every iteration?
  - ✓ Stop after k = 1
  - ✓ This is equivalent to value iteration (coming up)

Generalized Policy Iteration





- ✓ Policy evaluation - Estimate v_π
  - ✓ Any policy evaluation
- ✓ Policy improvement - Generate $\pi' \geq \pi$
  - ✓ Any policy improvement algorithm

Value Iteration

Optimality Principle

Any optimal policy can be subdivided into two components

- ✓ An optimal first action a^*
- ✓ Followed by an optimal policy from successor state s'

Theorem (Principle of Optimality)

A policy $\pi(a|s)$ achieves the optimal value from state s' (i.e. $v_\pi(s) = v_*(s)$) if and only if for any state s' reachable from s

- π achieves the optimal value from state s', $v_\pi(s') = v_*(s')$

Deterministic Value Iteration

- ✓ If we know the solution to subproblems $v_*(s')$
- ✓ Then solution $v_*(s)$ can be found by one-step lookahead

$$v_*(s) \leftarrow \max_{a \in \mathcal{A}} \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathrm{P}_{ss'}^a v_*(s')$$

- ✓ Value iteration applies these updates iteratively
- ✓ Intuition: start with final rewards and work backwards
  - ✓ Still works with loopy, stochastic MDPs

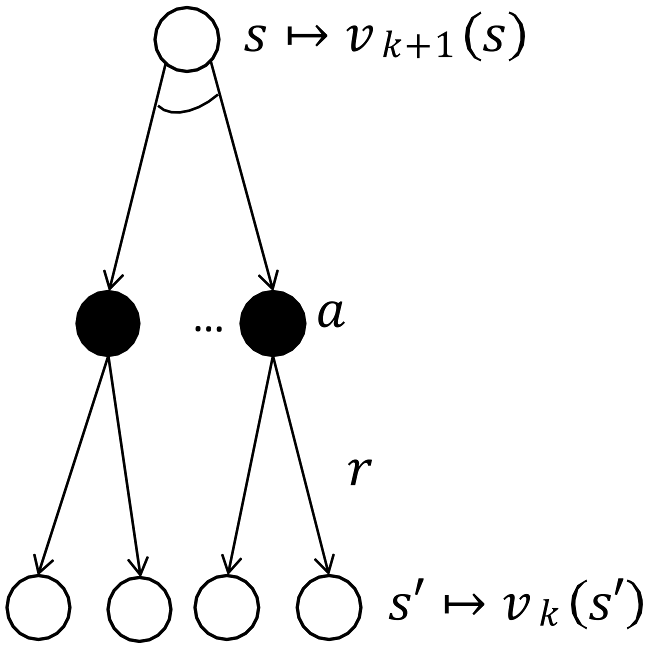
Value Iteration

- ✓ Problem: find optimal policy π
- ✓ Solution: iterative application of Bellman optimality backup

$$v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_\pi$$

- ✓ Using synchronous backups
  i. At each iteration $k + 1$
  ii. For all states $s \in \mathcal{S}$
  iii. Update $v_{k+1}(s)$ from $v_k(s')$
- ✓ Unlike policy iteration, there is no explicit policy
- ✓ Intermediate value functions may not correspond to any policy

Value Iteration - Formally



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

$$v_{k+1} = \max_{a \in \mathcal{A}} (\boldsymbol{\mathcal{R}}^a + \gamma \boldsymbol{P}^a v_k)$$

The algorithm:

Initialize $V(s)$ arbitrarily, for all $s \in \mathcal{S}$
Initialize θ to a small positive value

Loop:
$\quad \Delta \leftarrow 0$
$\quad$ Loop for each $s \in \mathcal{S}$:
$\quad\quad v \leftarrow V(s)$
$\quad\quad V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
$\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
Until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi*$, such that
$\pi(s) \leftarrow \text{argmax}_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

DP Example

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

Synchronous Dynamic Programming Wrap-up

Problem	Bellman Equation	Algorithm
Prediction	Bellman Expectation Equation	Iterative Policy Evaluation
Control	Bellman Expectation Equation + Greedy Policy Improvement	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

- ✓ Algorithms are based on state-value function $\mathrm{v}_\pi(\mathrm{s})$ or $\mathrm{v}_*(\mathrm{s})$
  - ✓ Complexity is $O(mn^2)$ per iteration ($m = |\mathcal{A}|$ and $n = |\mathcal{S}|$)
- ✓ Could also apply to action-value function $\mathrm{q}_\pi(\mathrm{s}, \mathrm{a})$ or $\mathrm{q}_*(\mathrm{s}, \mathrm{a})$
  - ✓ Complexity is $O(m^2n^2)$) per iteration

Take (stay) home messages

- ✓ Dynamic Programming - Method for solving complex problems by breaking them down into subproblems
  - ✓ Use recursive formulation founded in return nested definition
- ✓ Policy iteration - Re-define the policy at each step and compute the value according to this new policy until the policy converges
- ✓ Value iteration - Computes the optimal state value function by iteratively improving the estimate of V(s)
- ✓ Policy vs Value iteration
  - ✓ Policy can converge quicker (agent is interested in optimal policy)
  - ✓ Value iteration is computationally cheaper (per iteration)

Next Lecture

Model-Free Prediction

- ✓ Estimate the value function of an unknown MDP
- ✓ Monte-Carlo approaches
- ✓ Temporal-Difference learning
- ✓ TD(λ)

Before start…

Primo appello sessione estiva: 16/06

Secondo appello sessione estiva: 01/07 (NEW!)

Terzo appello sessione estiva: 21/07

Domani lezione speciale AIRC campus: uso degli strumenti computazionali per la lo studio del cancro.

Model Free Prediction

Introduction

Outline

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

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

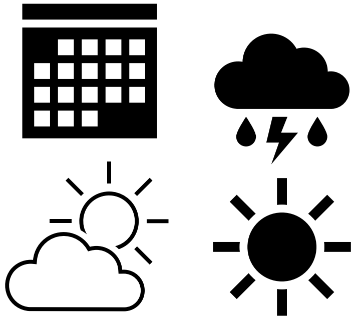
Monte-Carlo

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

Bootstrapping in RL

Model that, on Monday, predicts the meteo for Sunday.



- **No bootstrap:** Takes only info from Monday and Sunday

- **Bootstrap:** Uses information from the week meteo (induces bias)

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(*)
- ✓ MC uses the simplest possible idea: value = mean return across episodes
- ✓ Caveat: 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

$$S_1, A_1, R_2, \dots, R_k \sim \pi$$

- ✓ Recall that return is the total discounted reward

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

- ✓ Recall that value function is the expected return

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

- ✓ 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 $\mathrm{T}(s) \leftarrow \mathrm{T}(s) + G_t$
  III. Value is estimated by mean return $\mathrm{V}(s) = \mathrm{T(s)}/\mathrm{N}(s)$
- By law of large numbers

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

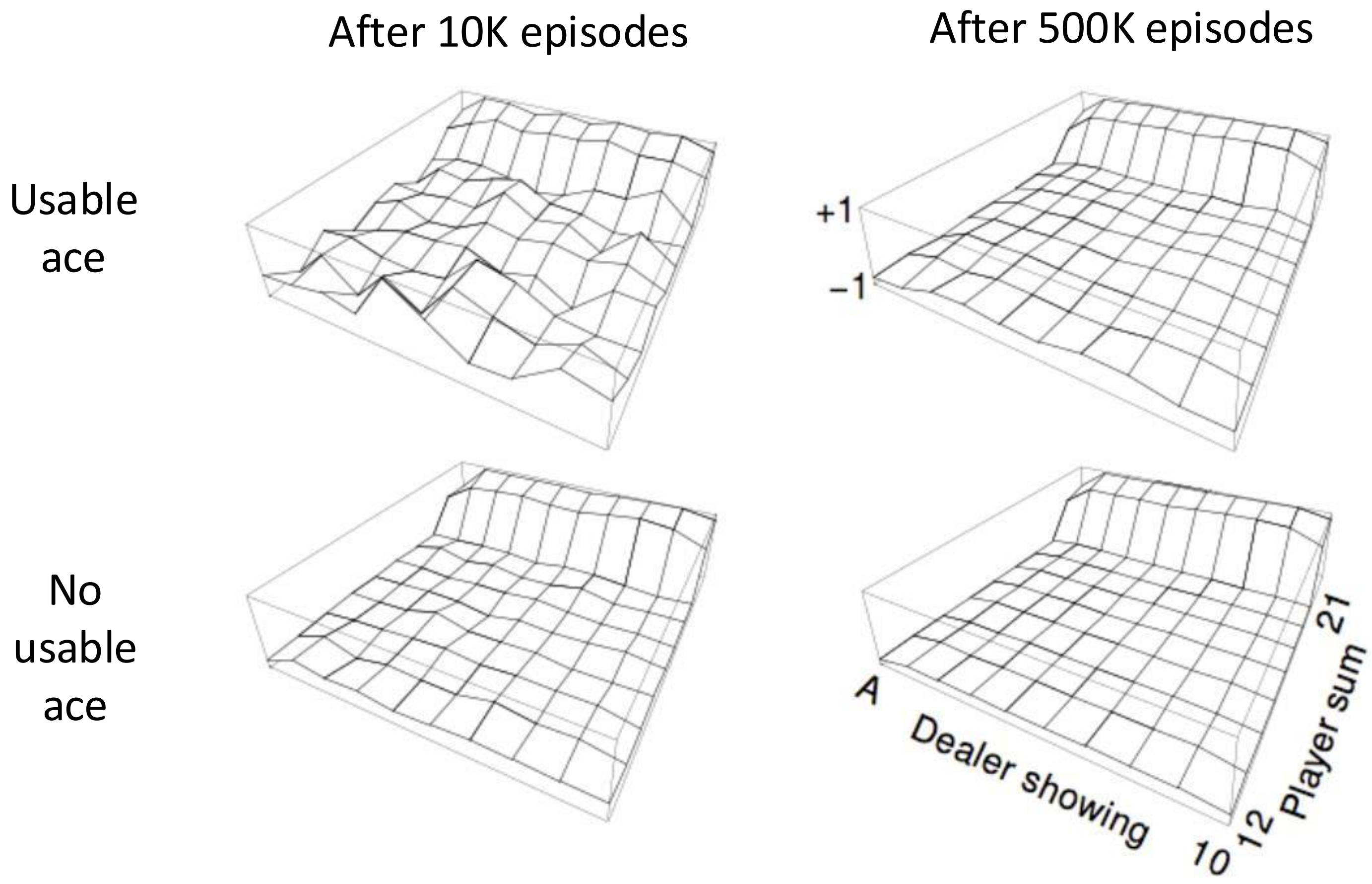
Every-Visit Monte-Carlo Policy Evaluation

- ✓ 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 $\mathrm{T}(s) \leftarrow \mathrm{T}(s) + G_t$
  III. Value is estimated by mean return $\mathrm{V}(s) = \mathrm{T(s)}/\mathrm{N}(s)$

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



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

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

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

Incremental Mean MC Update

- ✓ Update $\mathrm{V}(s)$ incrementally after episode $S_1, A_1, R_2, \dots, R_T$
- ✓ 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)

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

- ✓ 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
- ✓ 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

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

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

$$V(S_t) \leftarrow V(S_t) + \alpha(\underbrace{\underbrace{R_t + \gamma V(S_{t+1})}_{\text{TD target}} - V(S_t)}_{\text{TD error } \delta_t})$$

TD(0) Learning algorithm

Algorithm 1 Tabular TD(0) for estimating v_π

Input: Policy π to be evaluated **Parameters:** Learning rate $\alpha \in (0, 1]$

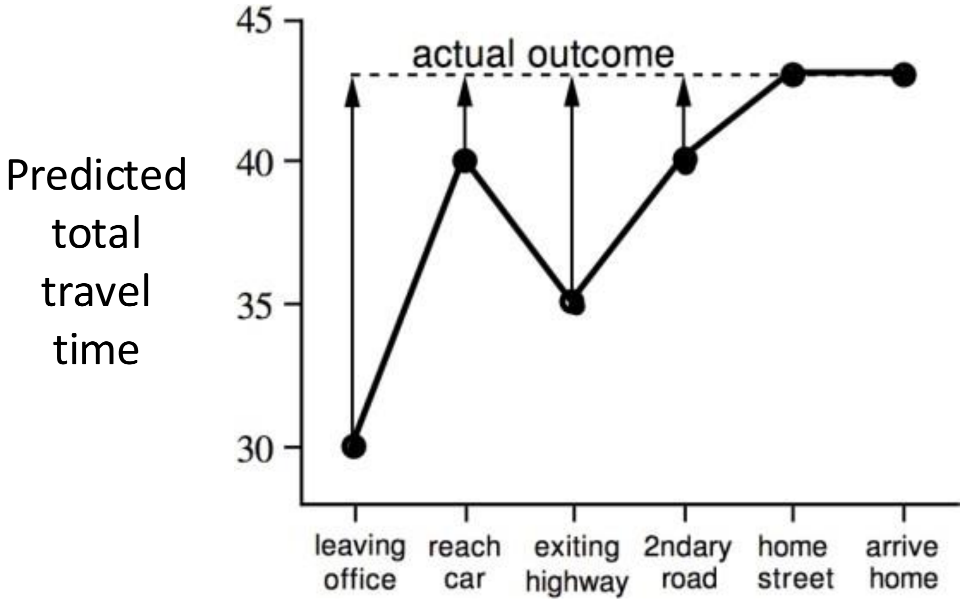
```
for each episode: do
    Initialize S
    while S is not terminal: do
        Take action A given by π(a|S)
        Observe R, S'
        Update V(S) ← V(S) + α[R + γV(S') − V(S)]
        S ← S'
    end while
end for
```

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

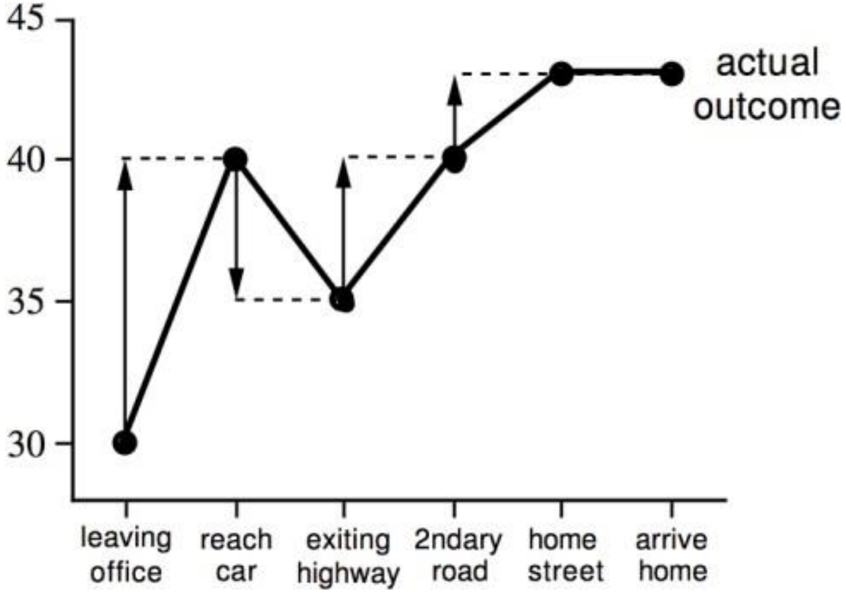
Driving Home Example – MC vs TD

Changes recommended by MC ($\alpha = 1$)



Situation

Changes recommended by TD ($\alpha = 1$)



Situation

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

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

Batch MC and TD

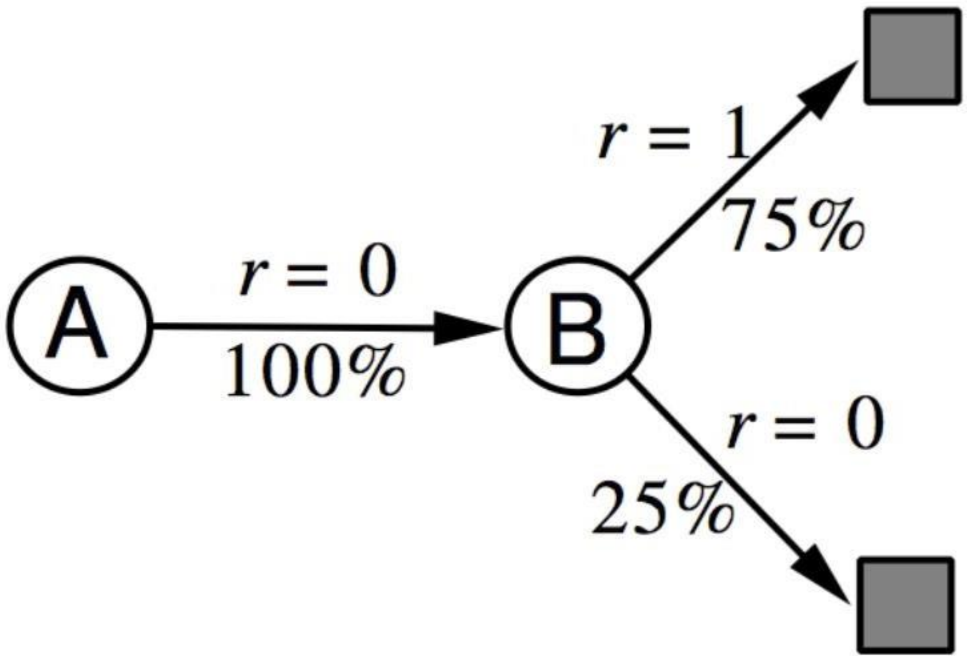
- ✓ MC and TD converge: $V(s) \rightarrow v_\pi(s)$ as experience $\rightarrow \infty$
- ✓ But what about batch solution for finite experience?

$$
\begin{gathered}
s_1^1, a_1^1, r_2^1, \dots, s_{T_1}^1 \\
\vdots \\
s_1^K, a_1^K, r_2^K, \dots, s_{T_K}^K
\end{gathered}
$$

  - ✓ 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)?

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 $\langle \mathcal{S}, \mathcal{A}, \boldsymbol{P}, \mathcal{R}, \gamma \rangle$ that best fits the data

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

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

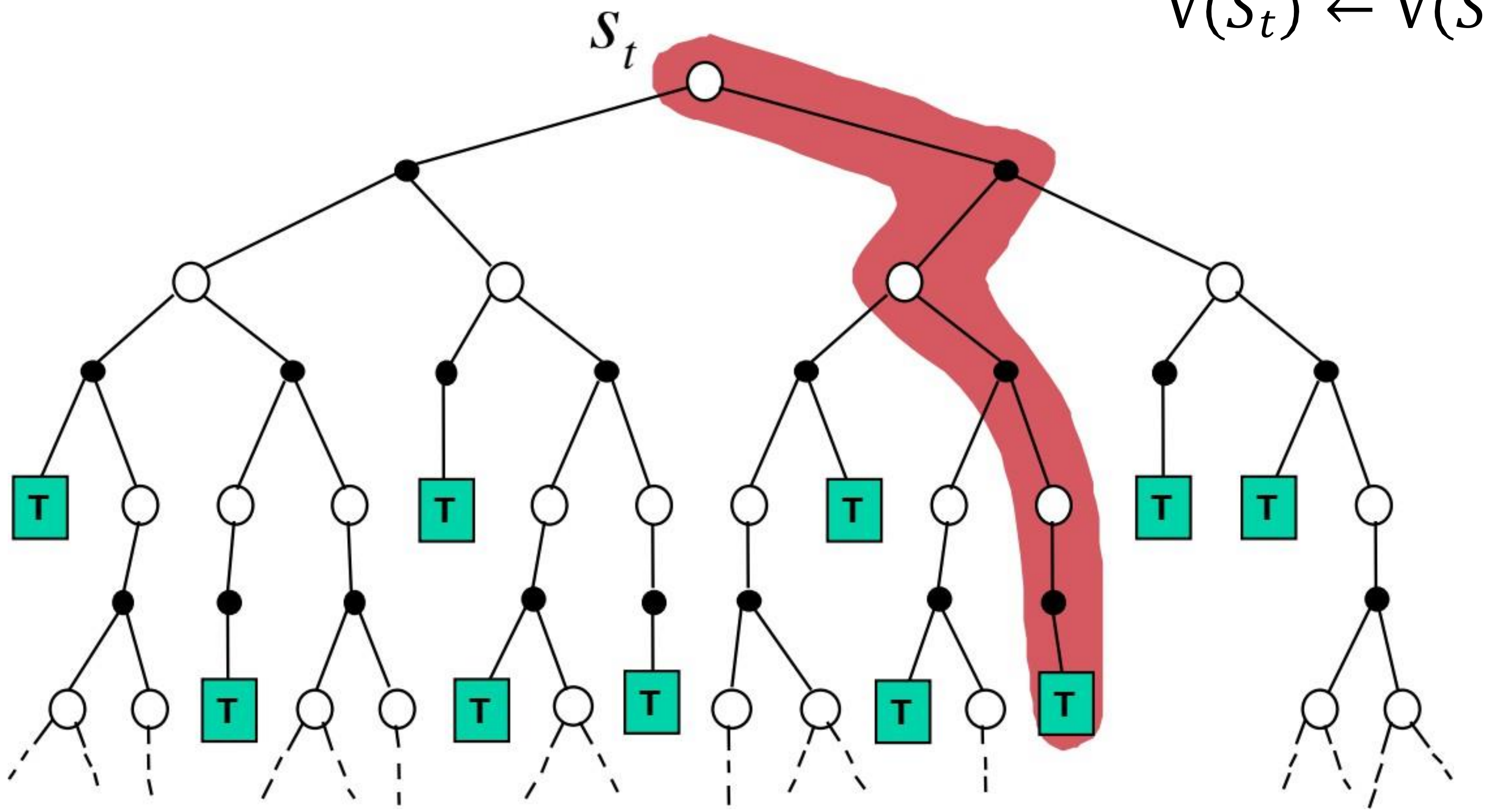
Advantages and Disadvantages of MC vs. TD (III)

- ✓ 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

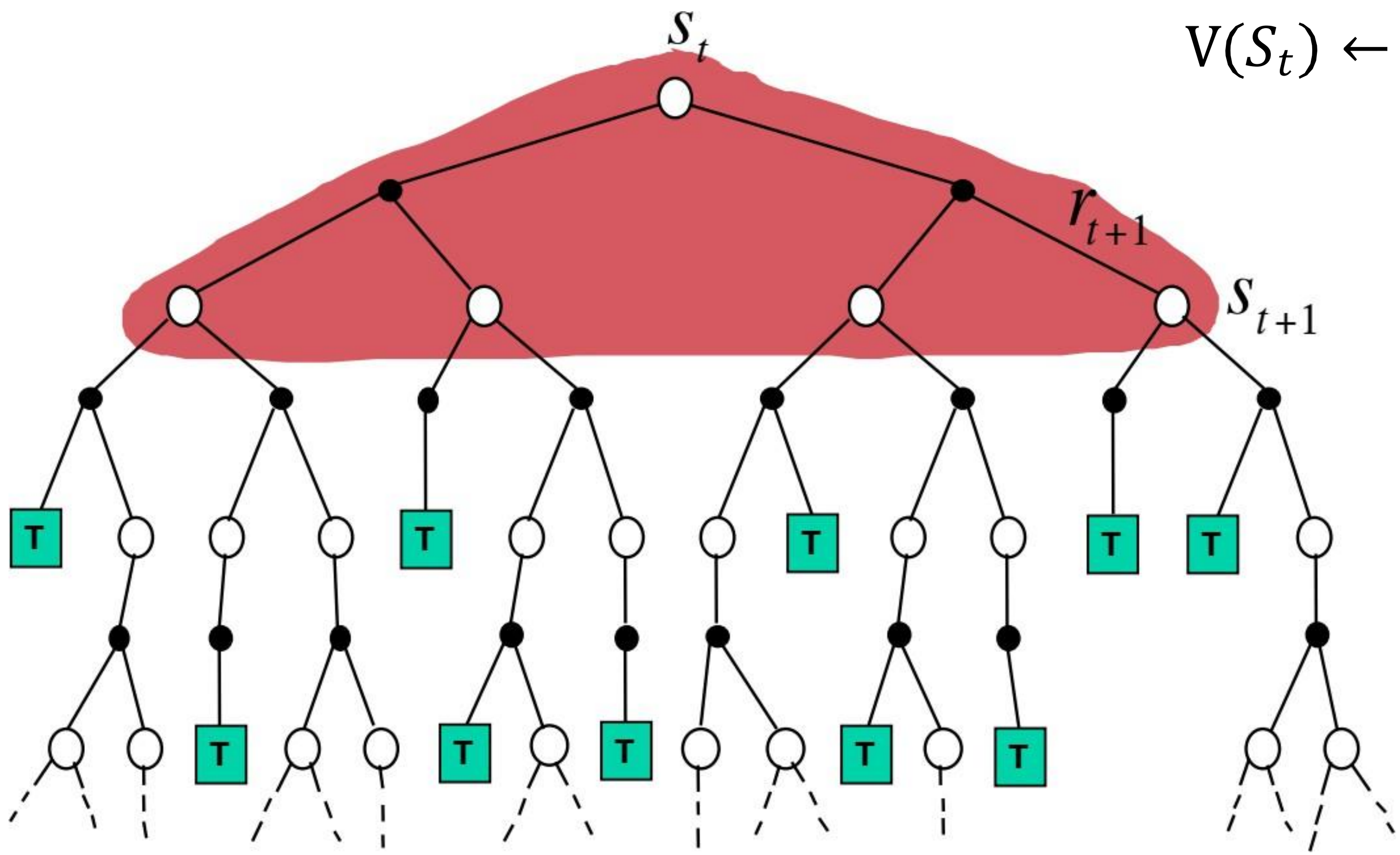
$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$



TD Update

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

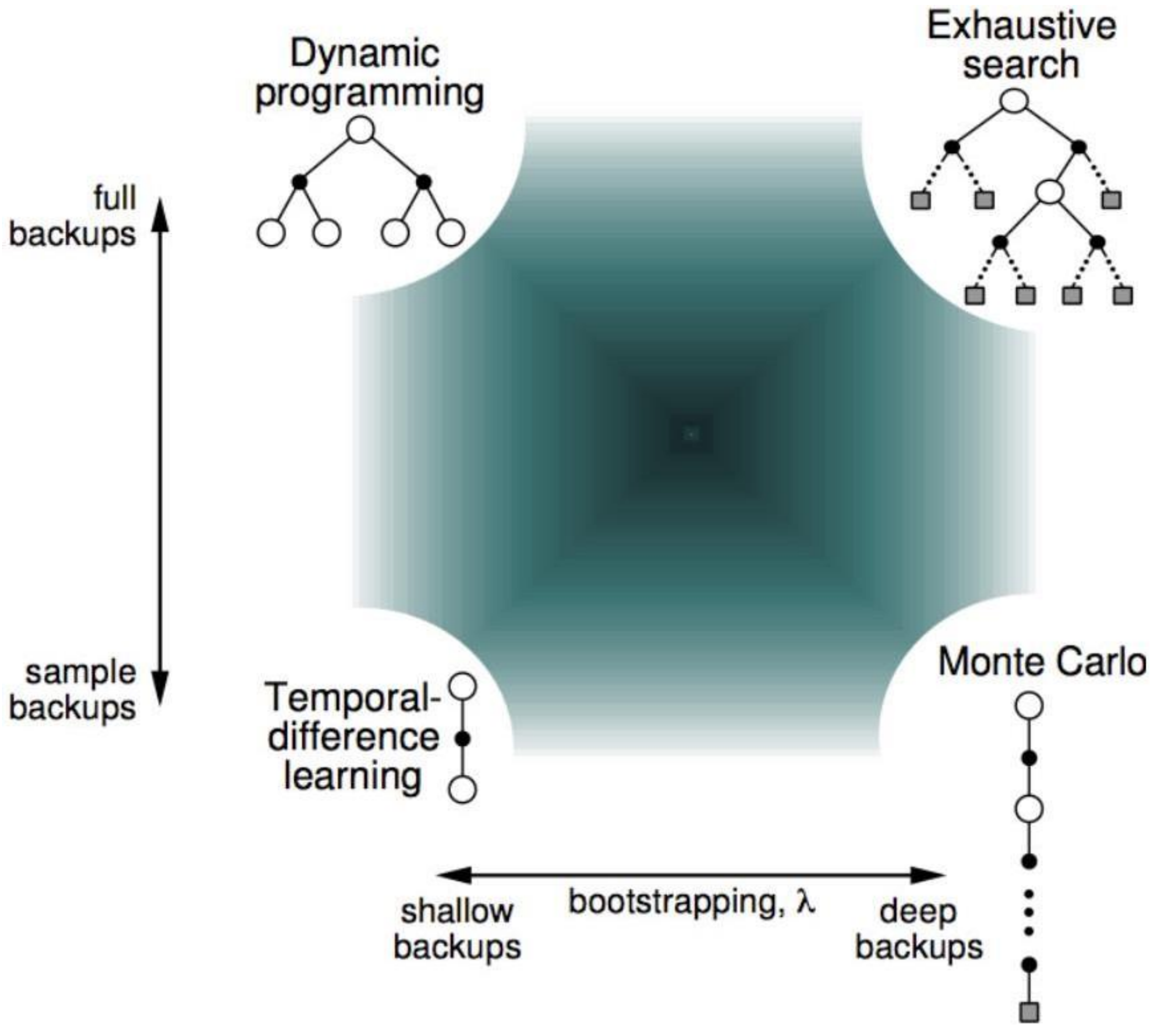
Dynamic Programming



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

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

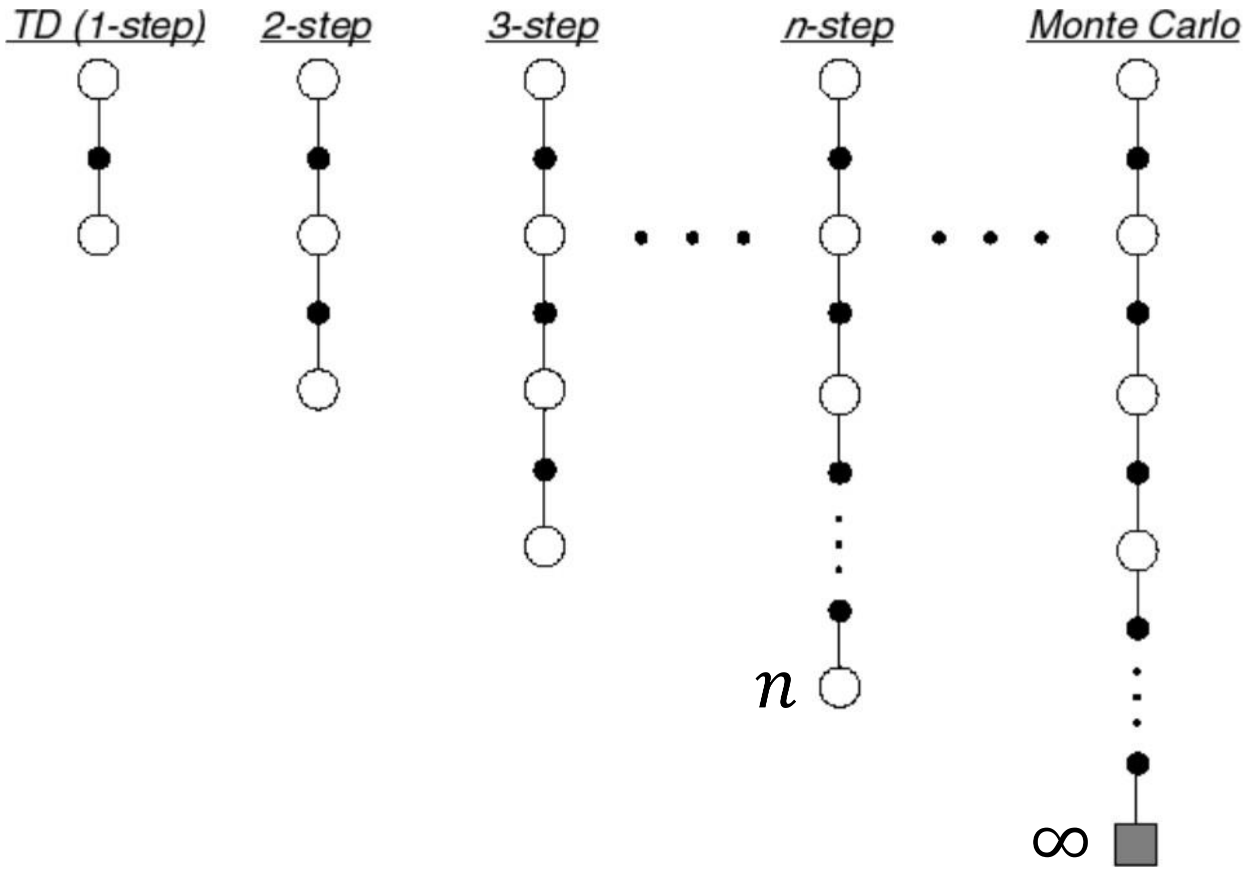


Unified View of RL

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 $n = 1,2,\dots,\infty$

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

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

$$\dots$$

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

✓ 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

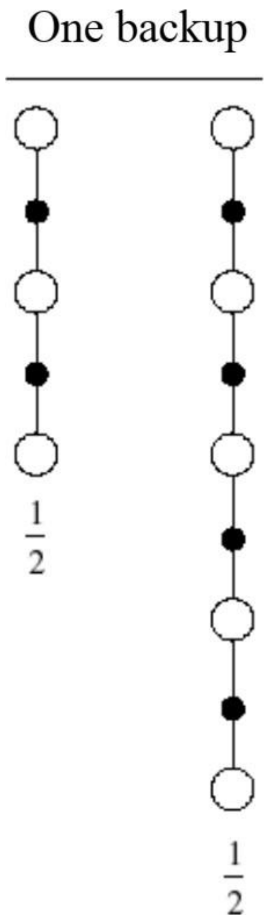
$$\mathrm{V}(S_t) \leftarrow \mathrm{V}(S_t) + \alpha\left(G_t^{(n)} - V(S_t)\right)$$

Averaging n-step Returns

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

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

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



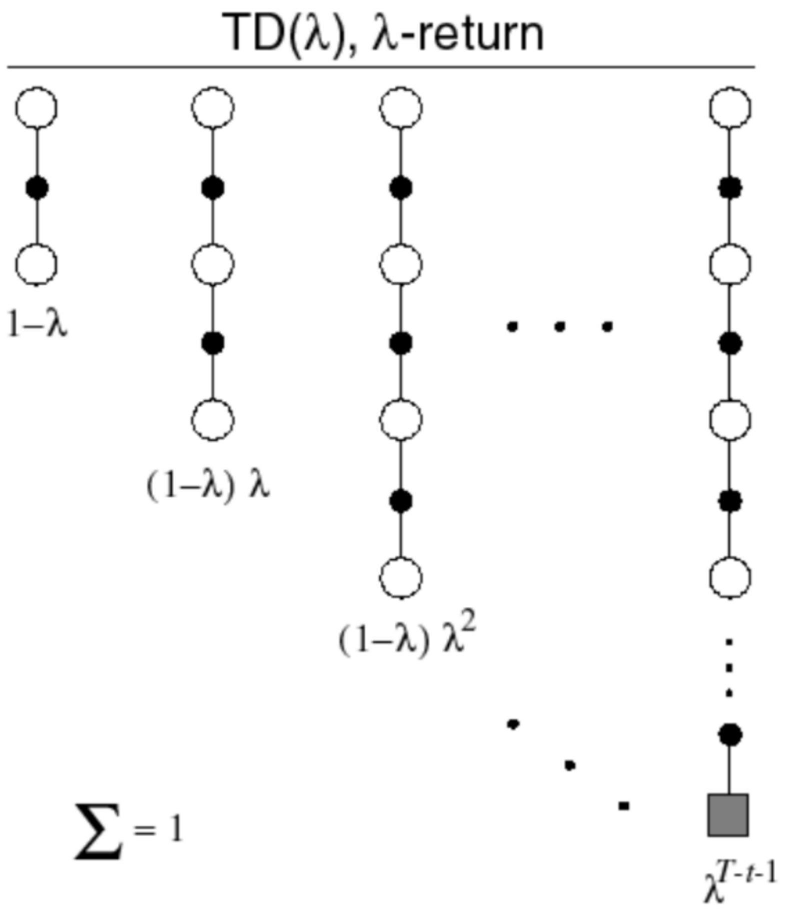
λ-returns

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

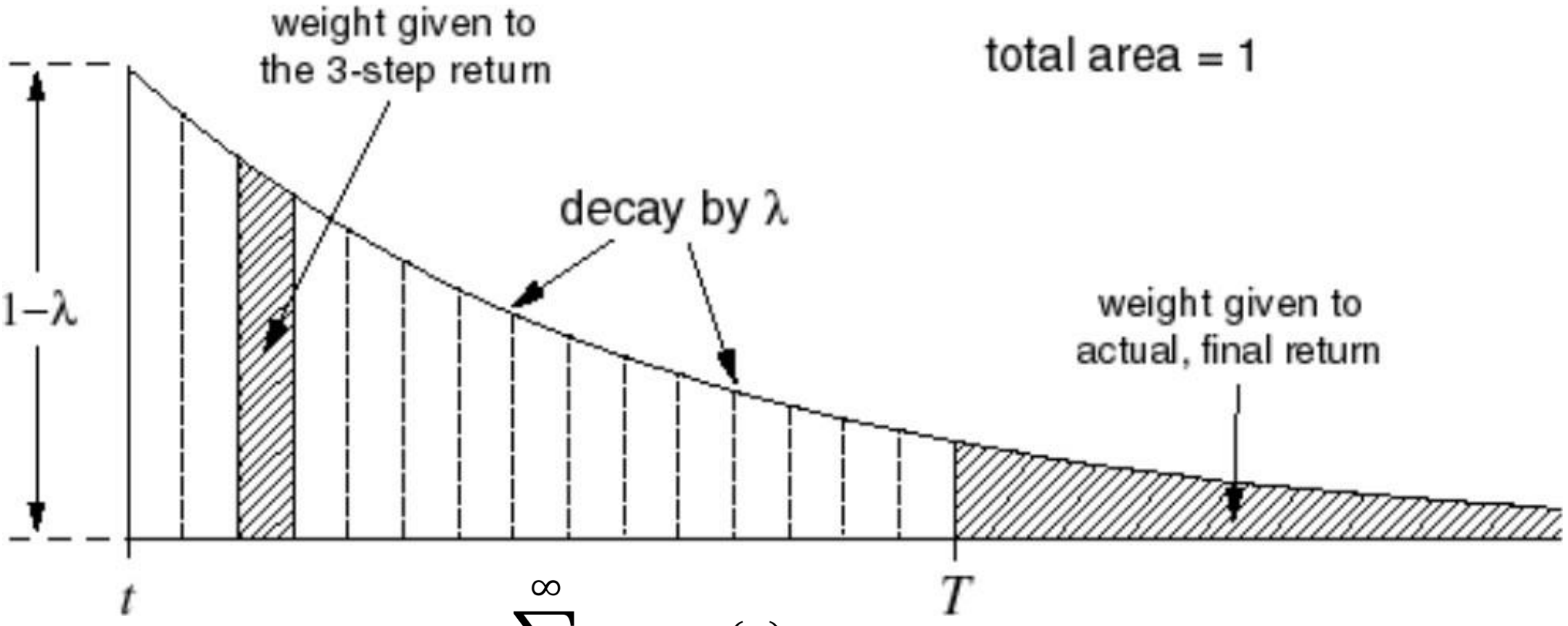
$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{(n)}$$

- ✓ Update as appropriate (TD(λ))

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t^\lambda - V(S_t)\right)$$

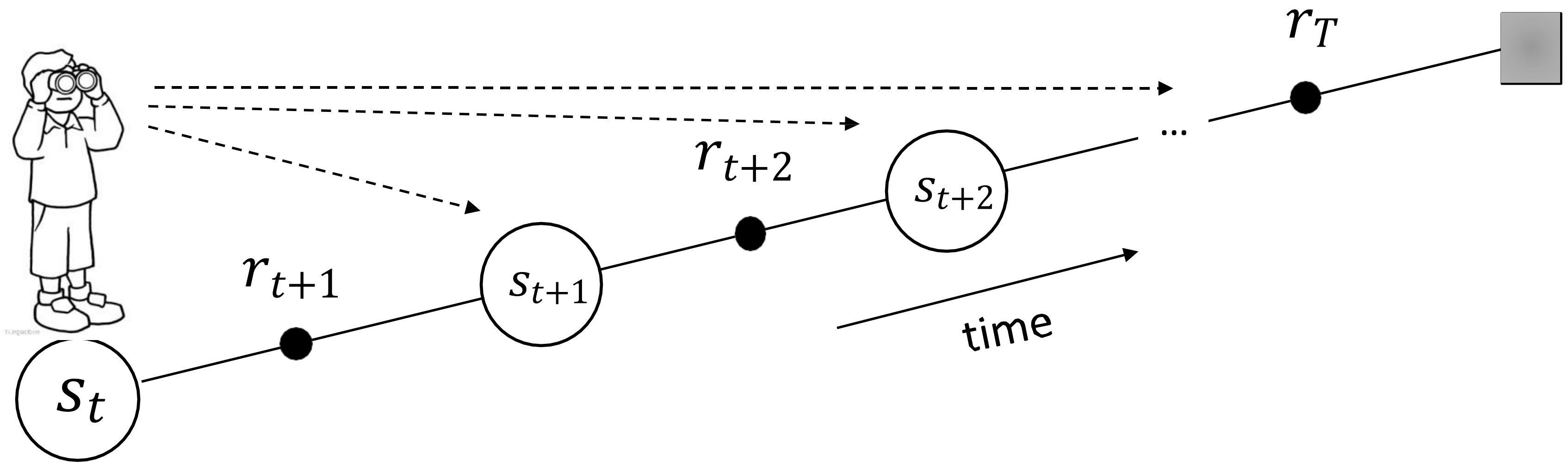


TD(λ) Weight Function



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

Forward View TD(λ)

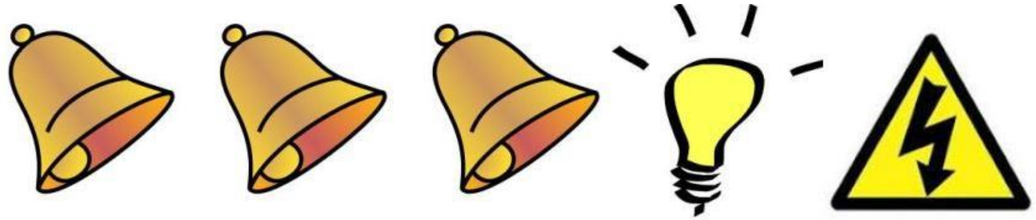


- ✓ Update value function towards the λ-return
- ✓ Forward-view looks into the future to compute G_t^λ
- ✓ Like MC, can only be computed from complete episodes

Backward View TD(λ)

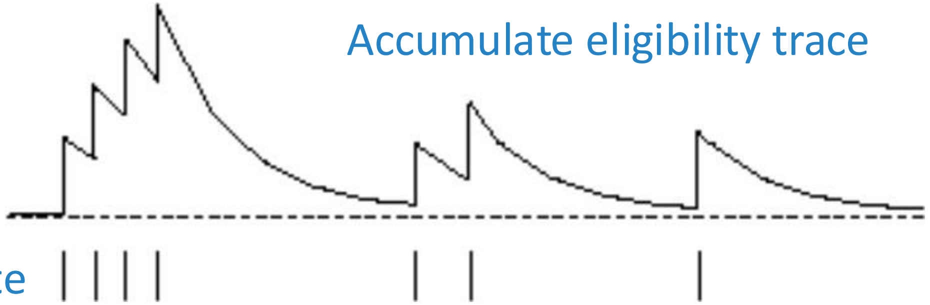
- ✓ Forward view provides theory
- ✓ Backward view provides mechanism
- ✓ Update online, every step, from incomplete sequences

Eligibility Traces



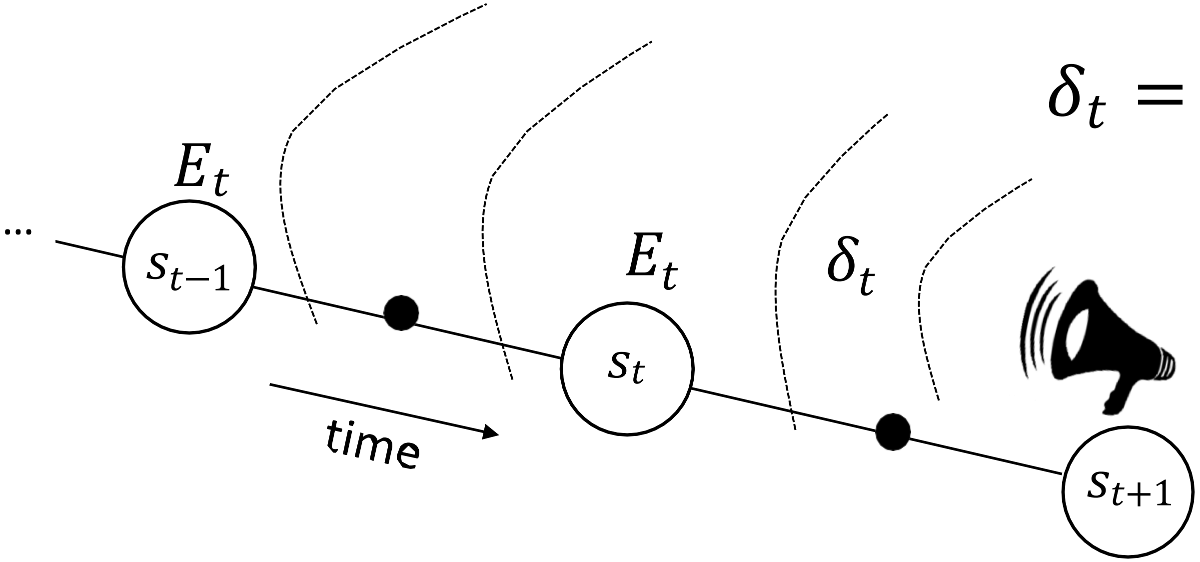
- Credit assignment problem: what caused shock?
  - Frequency heuristic: assign credit to most frequent states
  - Recency heuristic: assign credit to most recent states
- Eligibility traces combine both heuristics

$$E_0(s) = 0$$
$$E_t(s) = \gamma\lambda E_{t-1}(s) + \mathbf{1}(S_t; s)$$



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 $E_t(s)$



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

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

TD(λ) and TD($\mathbf{0}$)

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

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

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

- ✓ Equivalent to TD(0) update

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

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

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

- ✓ 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)
- ✓ TD learning
  - ✓ Learn from existing (biased) estimates of future return (bootstrapping)
  - ✓ Explore the future until n-th step

Next Lecture

Model-Free Control

- ✓ Optimise the value function of an unknown MDP
  - ✓ Generalised Policy Iteration
- ✓ Monte Carlo Control
- ✓ TD learning
- ✓ On-policy Vs Off-policy

Model Free Control

Outline

- ✓ Introduction
- ✓ On-policy Vs Off-Policy
- ✓ On-policy Monte-Carlo
- ✓ On-policy TD learning (SARSA)
- ✓ Off-policy TD (Q-learning)

Introduction

Today’s focus

- ✓ Last lecture
  - ✓ Model-free prediction
  - ✓ Estimate the value function of an unknown MDP
- ✓ Today’s lecture
  - ✓ Model-free control
  - ✓ Optimise the value function of an unknown MDP

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

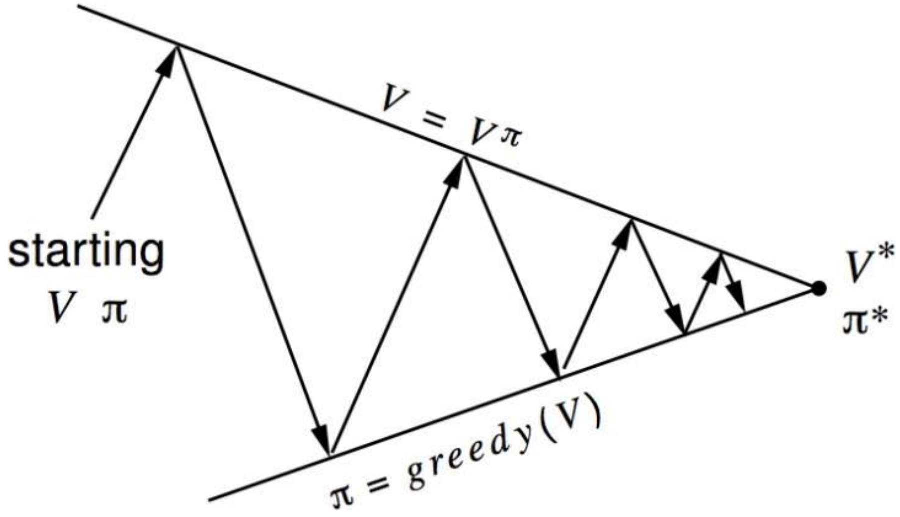
Model-free control can solve these problems

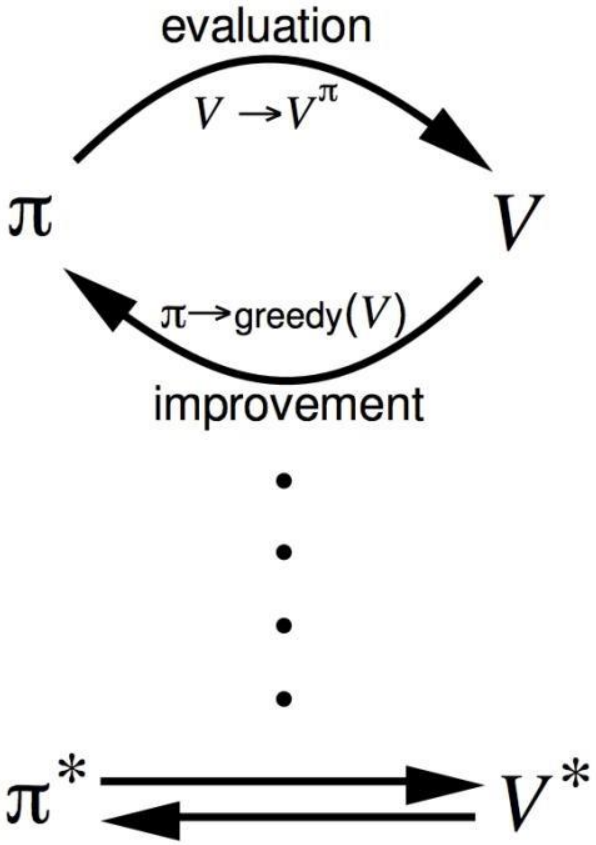
On-policy & Off-policy Learning

- 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 MC

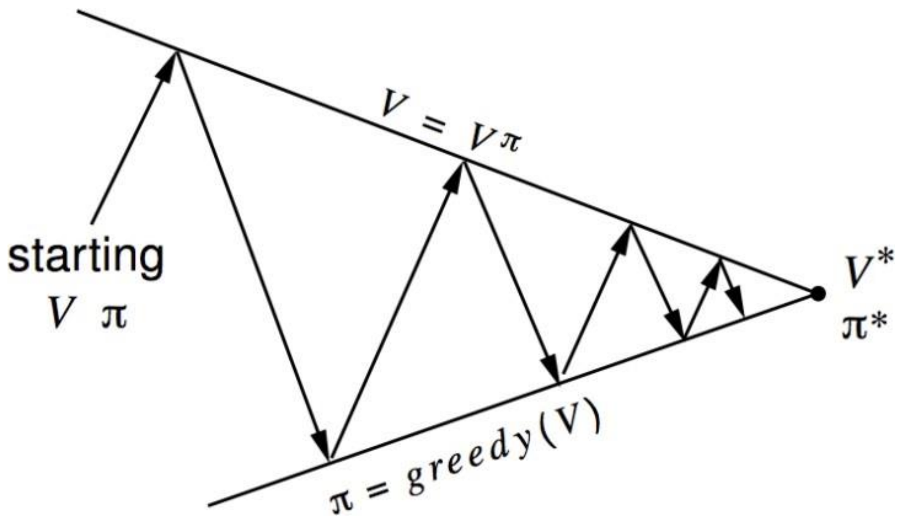
Generalized Policy Iteration (Lecture 3)





- ✓ 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

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

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

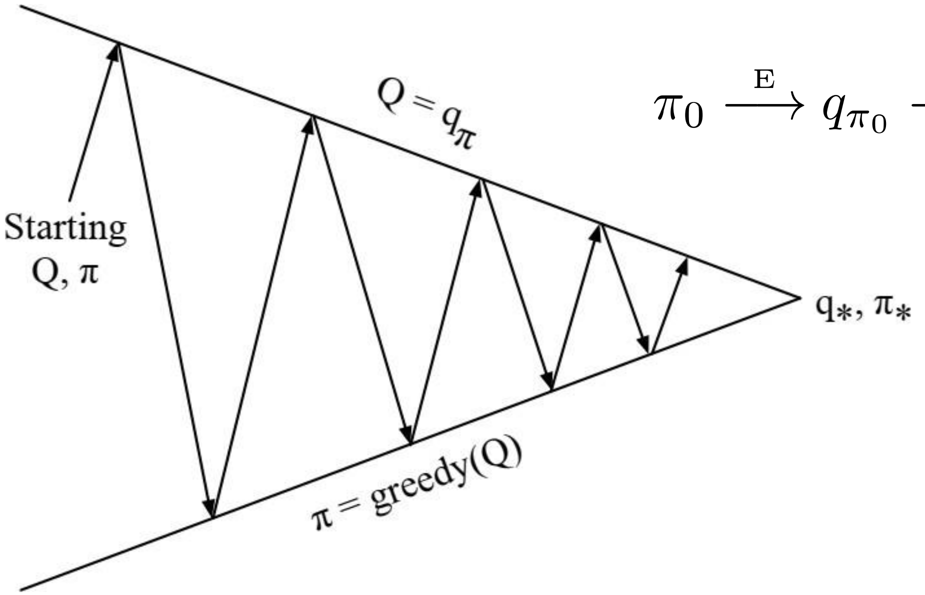
$$\pi'(s) = \arg\max_{a \in \mathcal{A}} Q(s,a)$$

Evaluating Action- Value Function with MC

$$q_\pi(s, a) \quad = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

- ✓ REMINDER: First-visit and every-visit variants of MC
- ✓ Take averages for state-action pairs

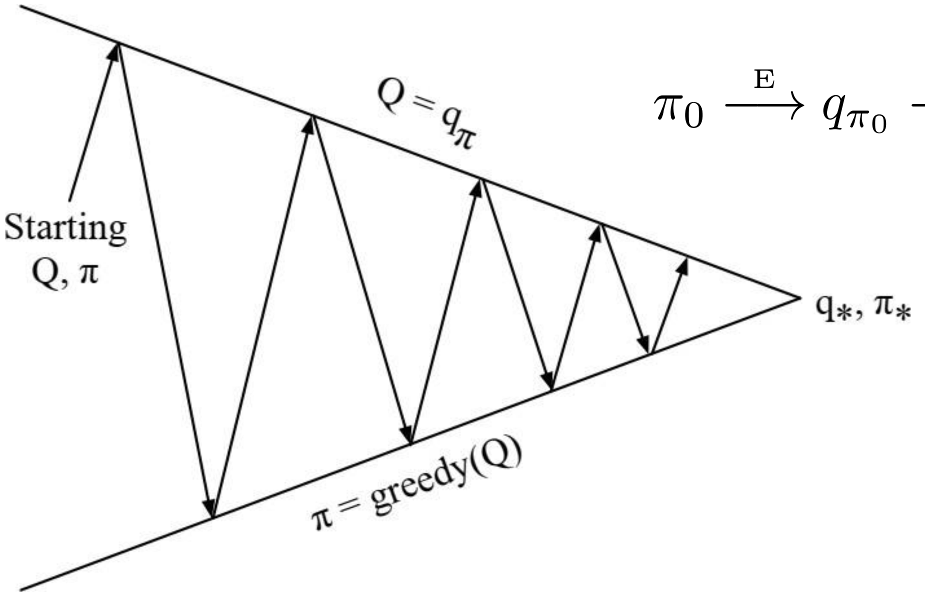
Generalized Policy Iteration with Action-Value Function



$$\pi_0 \xrightarrow{E} q_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} q_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots \xrightarrow{I} \pi_* \xrightarrow{E} q_*,$$

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

Generalized Policy Iteration with Action-Value Function



$$\pi_0 \xrightarrow{E} q_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} q_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots \xrightarrow{I} \pi_* \xrightarrow{E} q_*,$$

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

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

Evaluating Action- Value Function with MC

$$q_\pi(s, a) \quad = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

- ✓ REMINDER: First-visit and every-visit variants of MC
- ✓ Take averages for state-action pairs
- ✓ **What happens if π is deterministic?**

Two approaches to ensure exploration in MC-control

- ✓ Exploring Starts (enforce pair state-action visiting).
- ✓ Soft policy

Exploring Starts MC-control

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

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

Loop forever (for each episode):
  Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0
  Generate an episode from S_0, A_0, following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
      Append G to $Returns(S_t, A_t)$
      $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
      $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$

Exploring Starts MC-control

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

Initialize:
$\pi(s) \in \mathcal{A}(s)$ (arbitrarily), $\boxed{\text{for all } s \in \mathcal{S}}$
$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Loop forever (for each episode):
Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0
Generate an episode from S_0, A_0, following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
$G \leftarrow 0$
Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
$G \leftarrow \gamma G + R_{t+1}$
Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
Append G to $Returns(S_t, A_t)$
$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
$\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$

What happens if we don't know how to explore the starts?

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

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

ϵ-greedy Policy Improvement

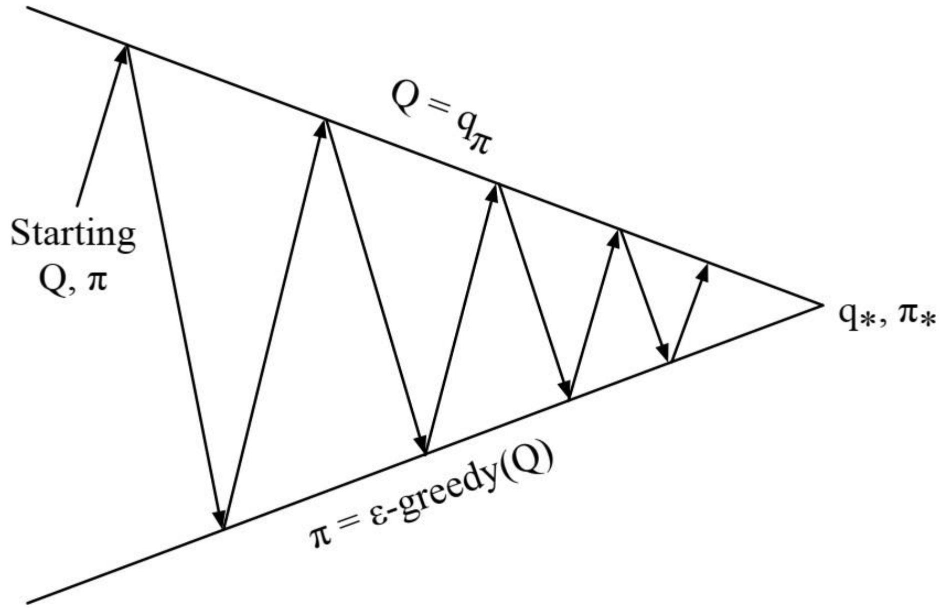
Theorem

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

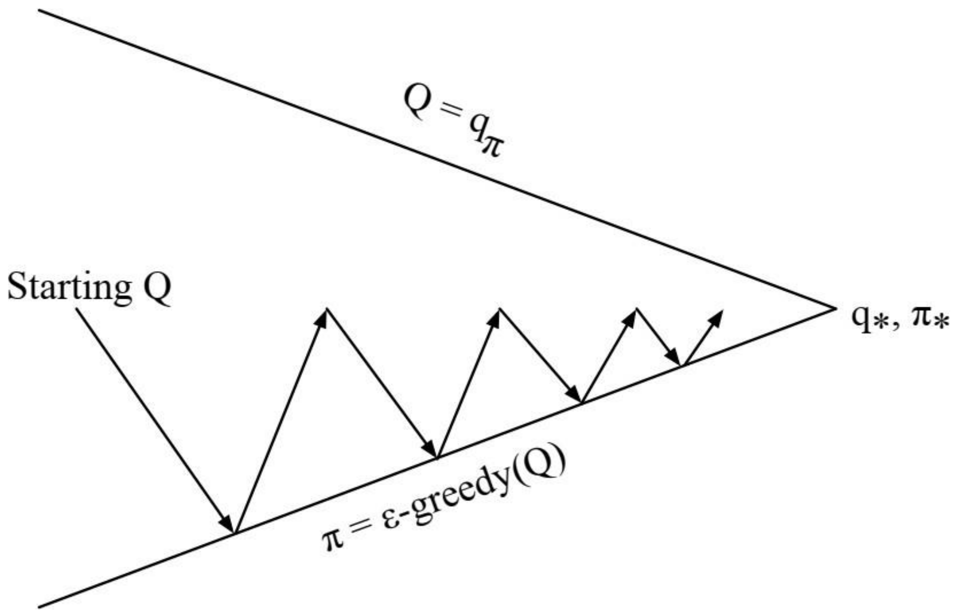
Monte-Carlo Policy Iteration



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

✓Policy improvement - ϵ-greedy policy improvement

Monte-Carlo Control



Every Episode

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

✓ Policy improvement - ϵ-greedy policy improvement

Monte-Carlo Control

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

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\quad \pi \leftarrow$ an arbitrary ε-soft policy

$\quad Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$\quad Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

$\quad$ Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$\quad G \leftarrow 0$

$\quad$ Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$\quad\quad G \leftarrow \gamma G + R_{t+1}$

$\quad\quad$ Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

$\quad\quad\quad$ Append G to $Returns(S_t, A_t)$

$\quad\quad\quad Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

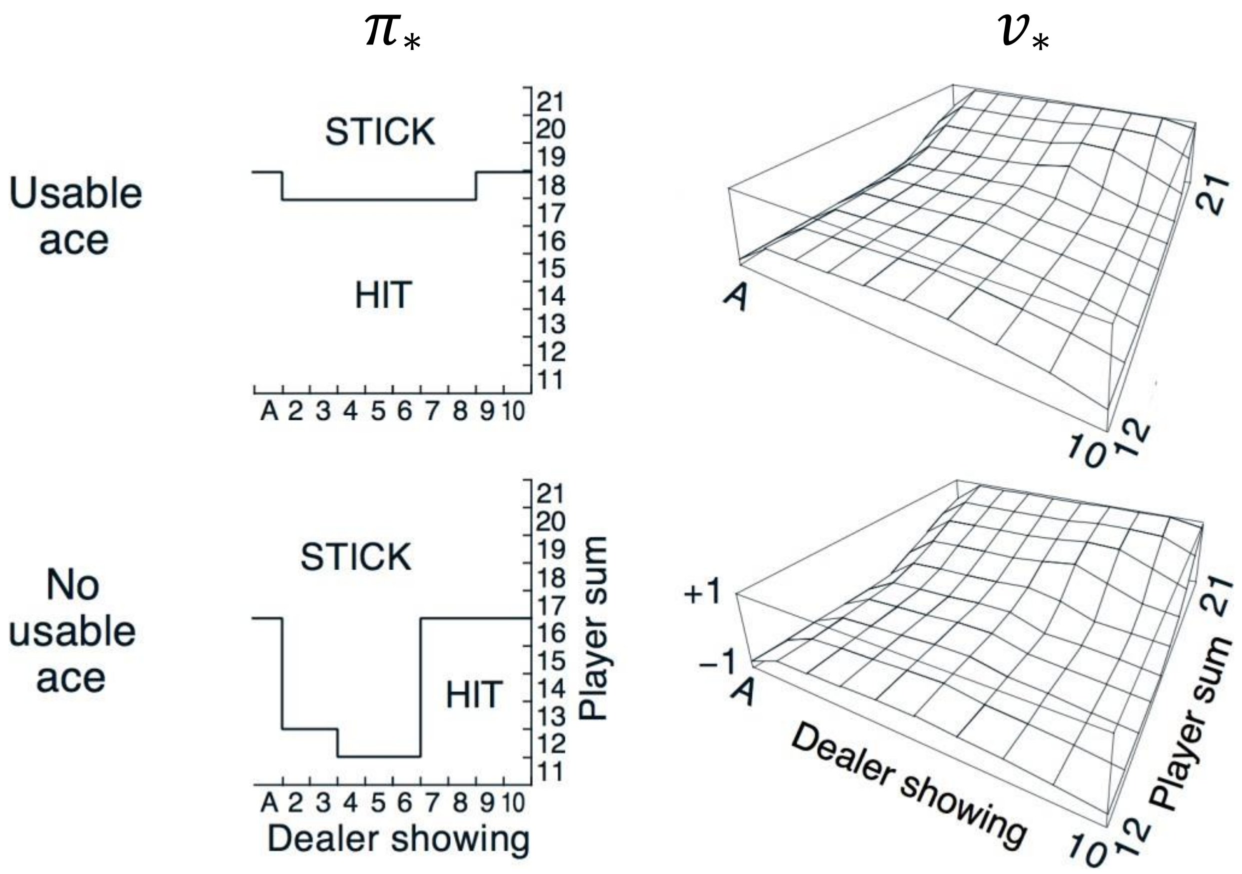
$\quad\quad\quad A^* \leftarrow \arg\max_a Q(S_t, a)$ $\quad$ (with ties broken arbitrarily)

$\quad\quad\quad$ For all $a \in \mathcal{A}(S_t)$:

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

Blackjack Example

Blackjack – MC Control

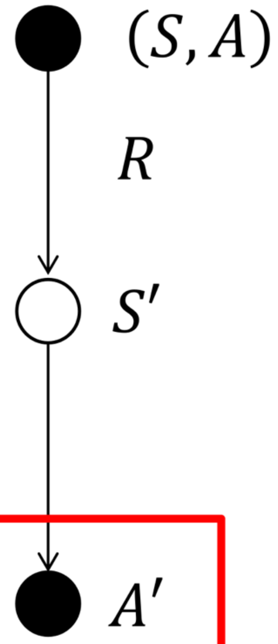


On-Policy TD Control

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 (State–action–reward–state–action)



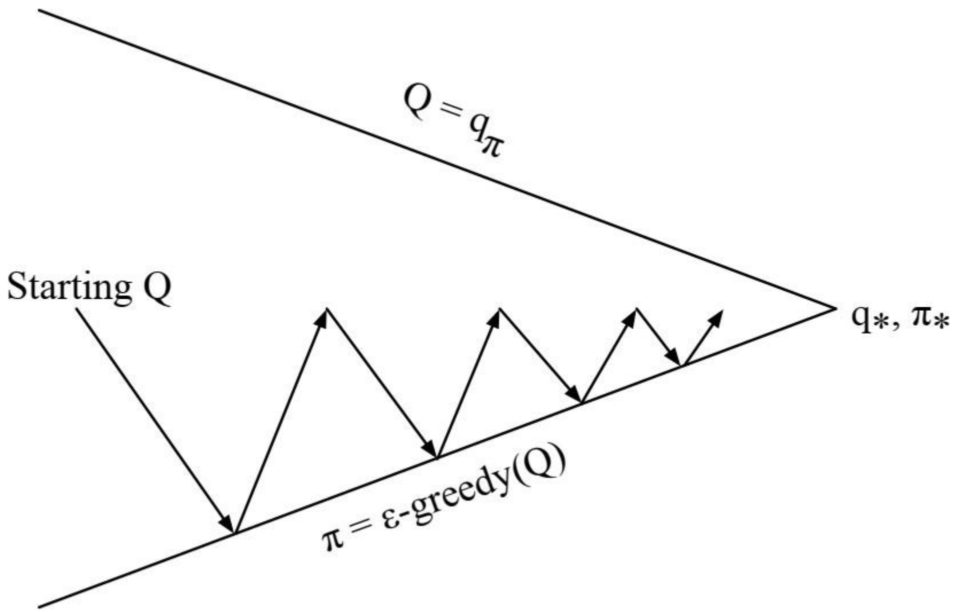
Expected SARSA

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

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

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

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

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 $n = 1,2, \dots, \infty$

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

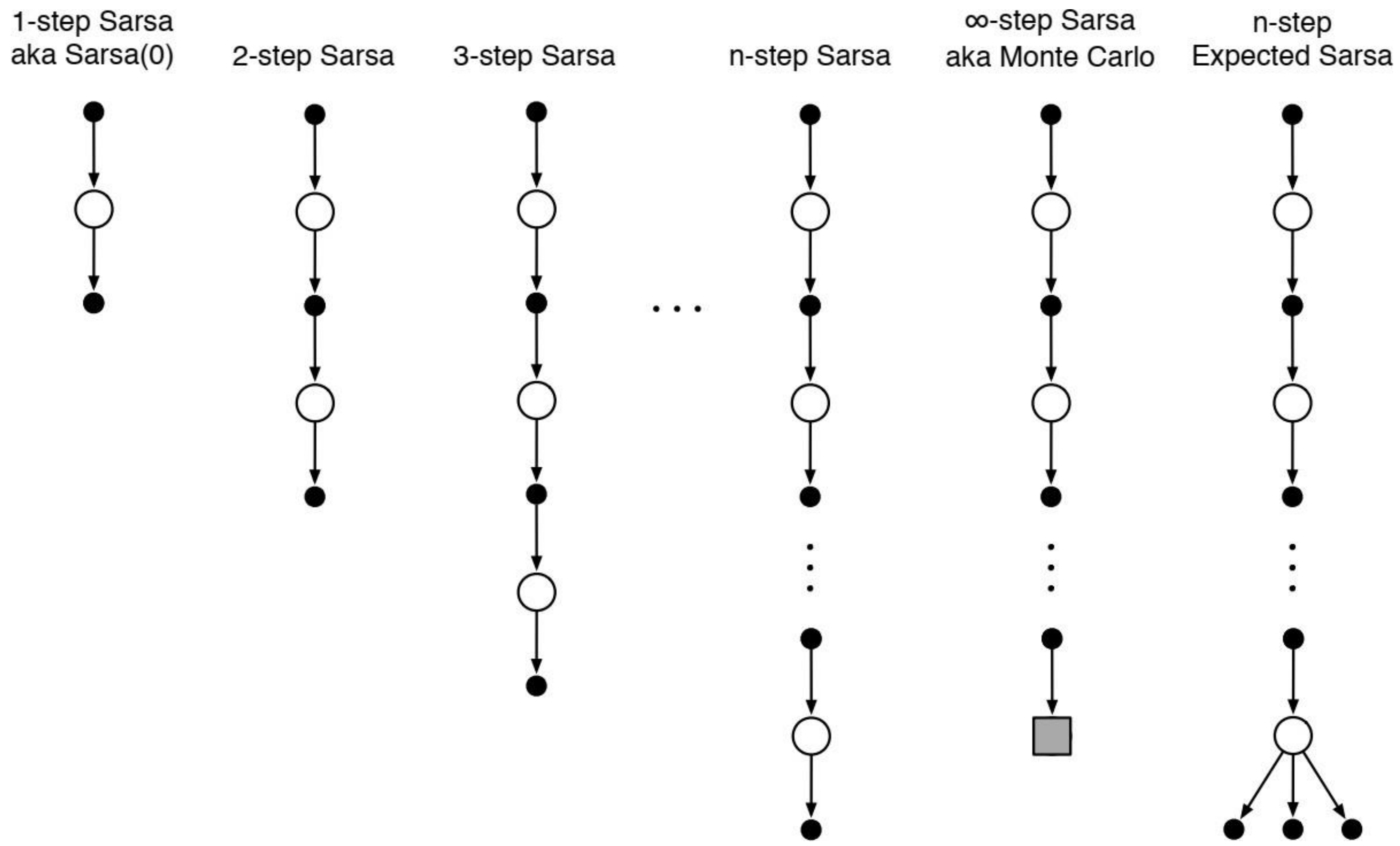
- Define the n-step Q-return

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

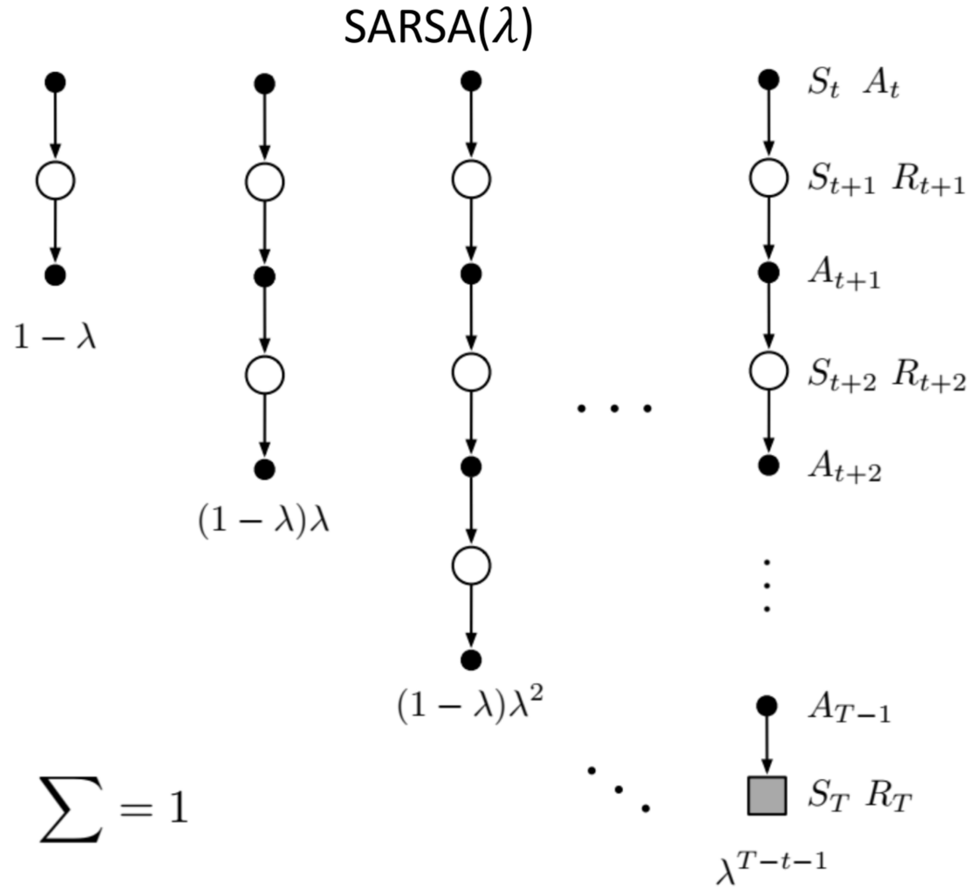
- n-step SARSA updates $Q(S, A)$ towards the n-step Q-return

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

SARSA backups



SARSA(λ) - Forward View



✓The q^λ return combines all n-step Q-returns $q_t^{(n)}$

✓Using weight $(1-\lambda)\lambda^{n-1}$

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

✓Forward SARSA update

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

SARSA(λ) - Backward View

- ✓The return of eligibility traces
- ✓ SARSA(λ) needs one eligibility trace for each state-action pair

$$E_0(s,a) = 0$$
$$E_t(s,a) = \gamma\lambda E_{t-1}(s,a) + \mathbf{1}(S_t, A_t; s, a)$$

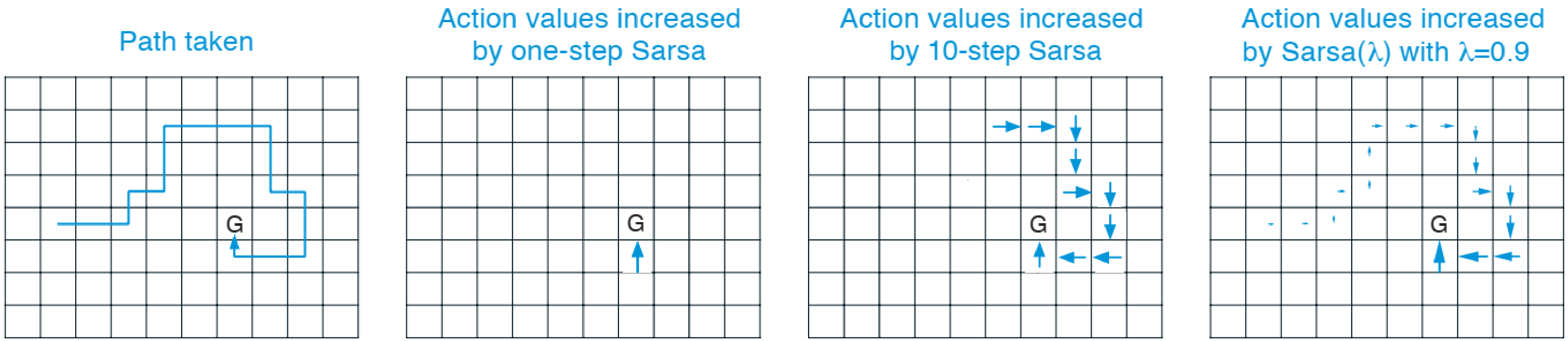
- ✓$Q(s,a)$ is updated for every state s and action a in proportion to TD-error δ_t and eligibility trace $E_t(s,a)$

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

SARSA(λ) Algorithm

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



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 $\mu(a|s)$

$$\{S_1, A_1, R_2, \dots, S_T\} \sim \mu$$

- ✓ Why is this important?
  - ✓ Learn from imitation (humans, other agents,…)
  - ✓ Re-use experience generated from old policies $\pi_1, \pi_2, \dots, \pi_{\mathrm{t}-1}$
  - ✓ 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

$$
\begin{aligned}
\mathbb{E}_{X\sim P}[f(X)] &= \sum P(X)f(X) \\
&= \sum Q(X)\frac{P(X)}{Q(X)}f(X) \\
&= \boxed{\mathbb{E}_{X\sim Q}}\left[\boxed{\frac{P(X)}{Q(X)}}f(X)\right]
\end{aligned}
$$

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

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

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

- ✓ 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

Importance Sampling for Off-Policy TD

- ✓ 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 $A_{t+1} \sim \mu(\cdot \mid S_t)$
- ✓ But we consider alternative successor action $A' \sim \pi(\cdot \mid S_t)$
- ✓ And update $Q(S_t, A_t)$ towards value of alternative action

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

Off-policy Control by Q-Learning

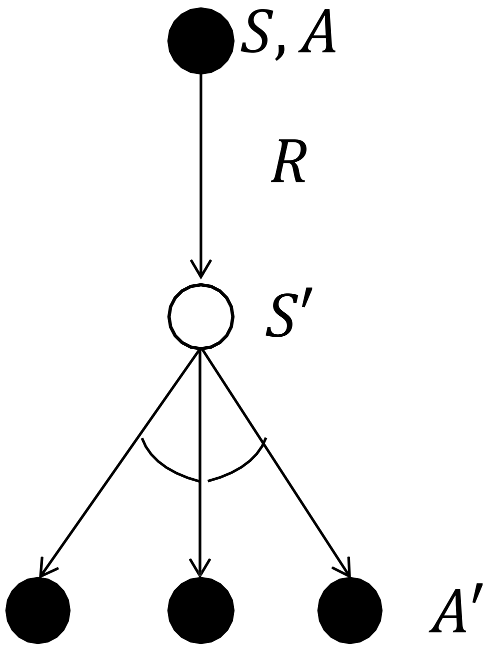
- ✓ Allow both behaviour and target policies to improve
- ✓ The target policy π is greedy w.r.t. $Q(S_t, A_t)$

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

- ✓ 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

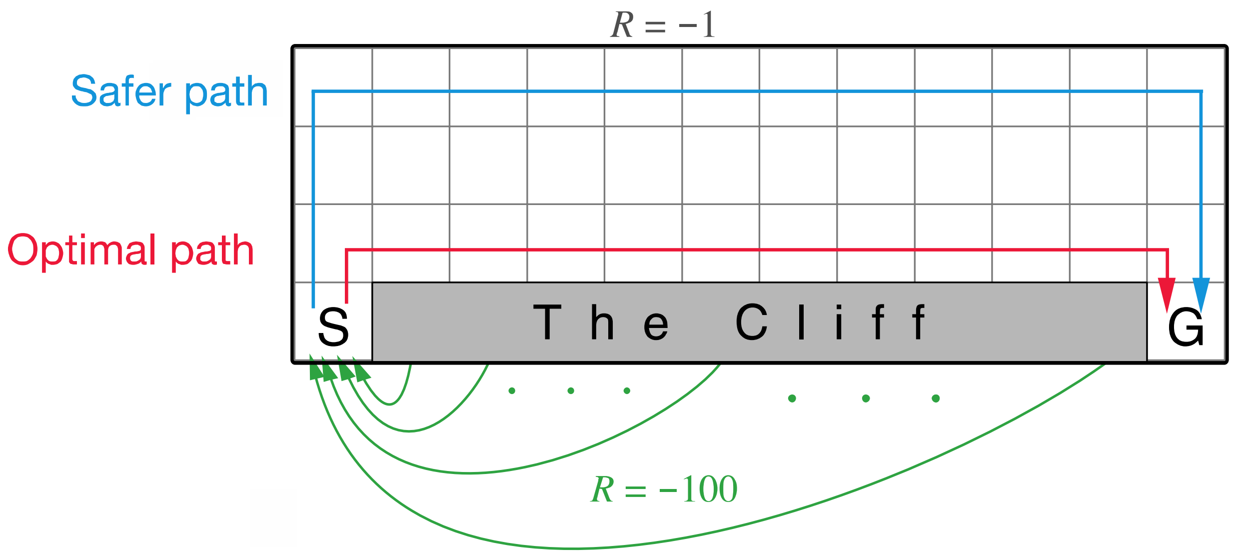
Q-learning control converges to the optimal action-value function, $Q(s,a) \to q_*(s,a)$

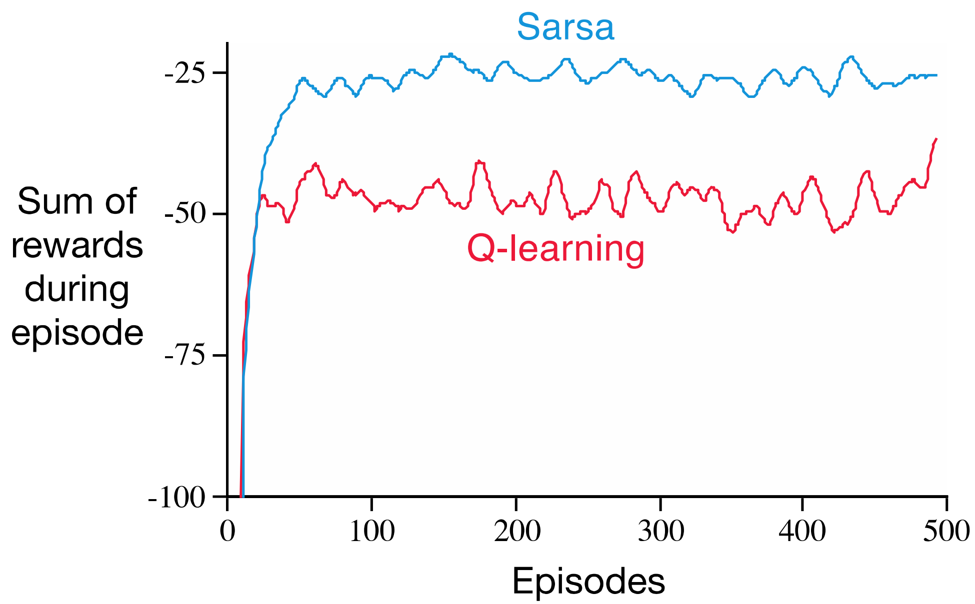
$$Q(S,A) \leftarrow Q(S,A) + \alpha \left(R + \max_{a'} \gamma Q(S', a') - Q(S,A) \right)$$

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

Q-Learning vs SARSA





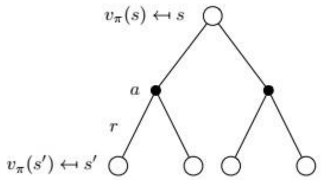
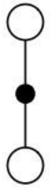
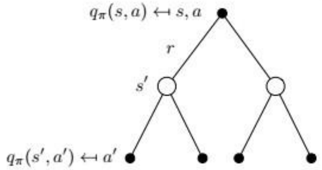
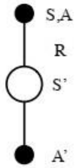
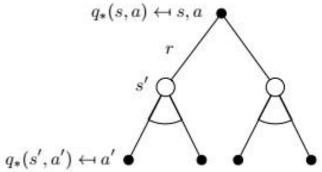
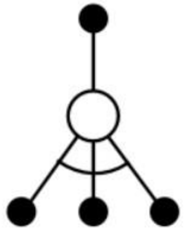
OPTIMAL PATH vs ON-LINE PERFORMANCE

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

Q-learning & Exploration Demo

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 (stay) 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