Carnegie Mellon
School of Computer Science

Deep Reinforcement Learning and Control

Monte Carlo Learning and Temporal Difference Learning

Fall 2021, CMU 10-703

Instructors
Katerina Fragkiadaki
Russ Salakhutdinov

Summary so far

- So far, to estimate value functions we have been using dynamic programming with known rewards and dynamics functions:

$$v_{[k+1]}(s) = \sum_a \pi(a|s)\left(r(s,a) + \gamma \sum_{s'} p(s'|s,a)v_{[k]}(s')\right), \forall s$$

$$v_{[k+1]}(s) = \max_{a \in \mathcal{A}}\left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a)v_{[k]}(s')\right), \forall s$$

Q: Was our agent interacting with the world? Was our agent exploring?

A: 1) No. 2) No, if you know everything, there is nothing to explore.

Coming up

- So far, to estimate value functions we have been using dynamic programming with known rewards and dynamics functions:

$$v_{\pi,[k+1]}(s) = \sum_a \pi(a \mid s)\left(r(s,a) + \gamma \sum_{s'} p(s' \mid s,a) v_{\pi,[k]}(s') \right), \forall s$$

$$v_{[k+1]}(s) = \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) v_{[k]}(s') \right), \forall s$$

- Next: estimate value functions and policies from interaction experience, without known rewards or dynamics.
- How? By sampling all the way. Instead of probabilities distributions to compute expectations, we will use empirical expectations by averaging sampled returns.

Monte Carlo (MC) Methods

- Monte Carlo methods are learning methods
  - Experience → values, policy
- Monte Carlo methods learn from complete sampled trajectories and their returns.
  - Only defined for episodic tasks .
  - All episodes must terminate.
- Monte Carlo uses the simplest possible idea: value = mean return

Monte-Carlo Policy Evaluation

- Goal: learn $v_\pi(s)$ from episodes of experience under policy π:

$$S_1, A_1, R_2, ..., S_k \sim \pi$$

- Remember that the return is the total discounted reward:

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

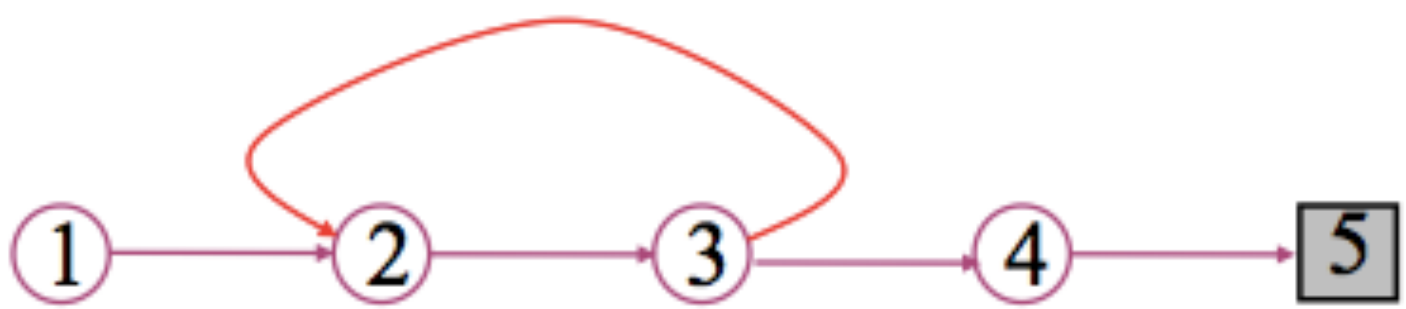
- Remember that the value function is the expected return:

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

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

Monte-Carlo Policy Evaluation

- Goal: learn $v_\pi(s)$ from episodes of experience under policy π:
- Idea: Average returns observed after visits to s:



- Every-Visit MC: average returns for every time s is visited in an episode
- First-visit MC: average returns only for first time s is visited in an episode
- Both converge asymptotically based on the law of large numbers

First-Visit MC Policy Evaluation

- To evaluate state s
- The first time-step t that state s is visited in an episode,
  - Increment counter: $N(s) \leftarrow N(s) + 1$
  - Increment total return: $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return $V(s) = S(s)/N(s)$
- By law of large numbers $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$

Law of large numbers

Every-Visit MC Policy Evaluation

- To evaluate state s
- Every time-step t that state s is visited in an episode,
  - Increment counter: $N(s) \leftarrow N(s) + 1$
  - Increment total return: $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return $V(s) = S(s)/N(s)$
- By law of large numbers $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$

Incremental Mean

- The mean μ_k of a sequence $x_1 \ldots x_k$ can be computed incrementally:

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

Monte Carlo Prediction

- Update V(s) incrementally after episode $S_1, A_1, R_2, ..., S_T$
- For each state St with return Gt

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

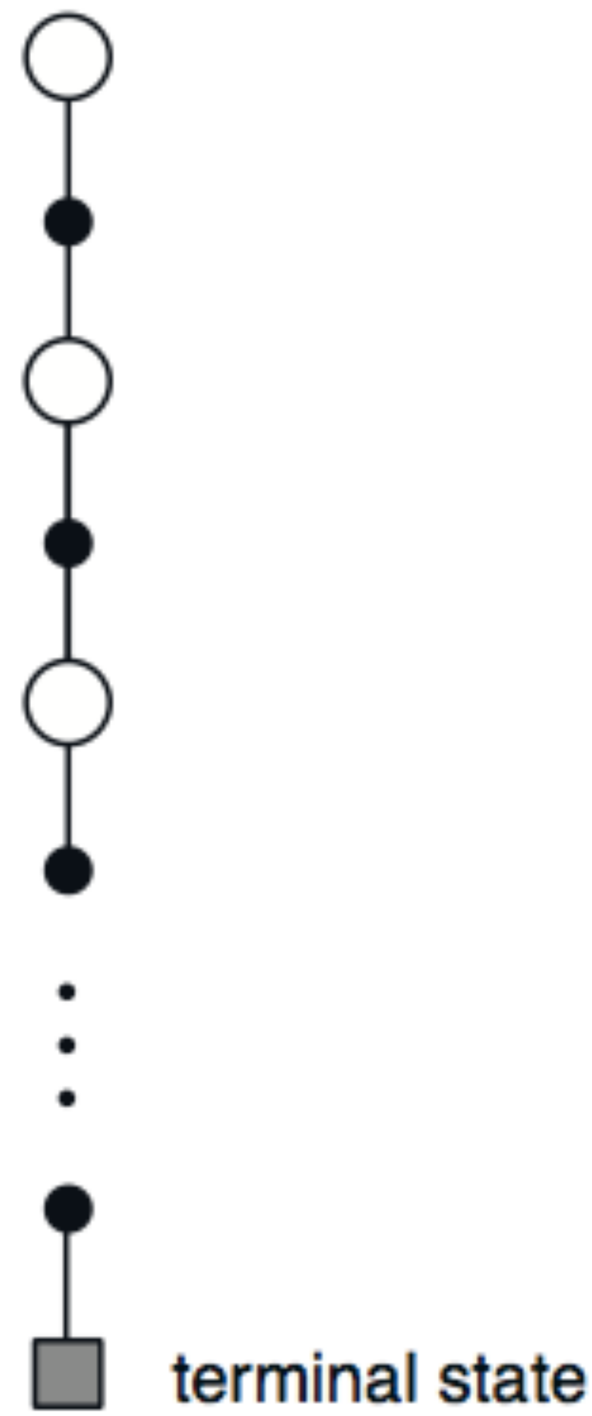
$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t))$$

- In non-stationary problems, it can be useful to track a running mean, i.e. forget old episodes.

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

Backup Diagram for Monte Carlo

- Entire rest of episode included
- Only one choice considered at each state (unlike DP)
- Does not bootstrap from successor state's values (unlike DP), i.e., the value estimates of later states are not used to inform the values of nearby states.
- Value is estimated by mean return.

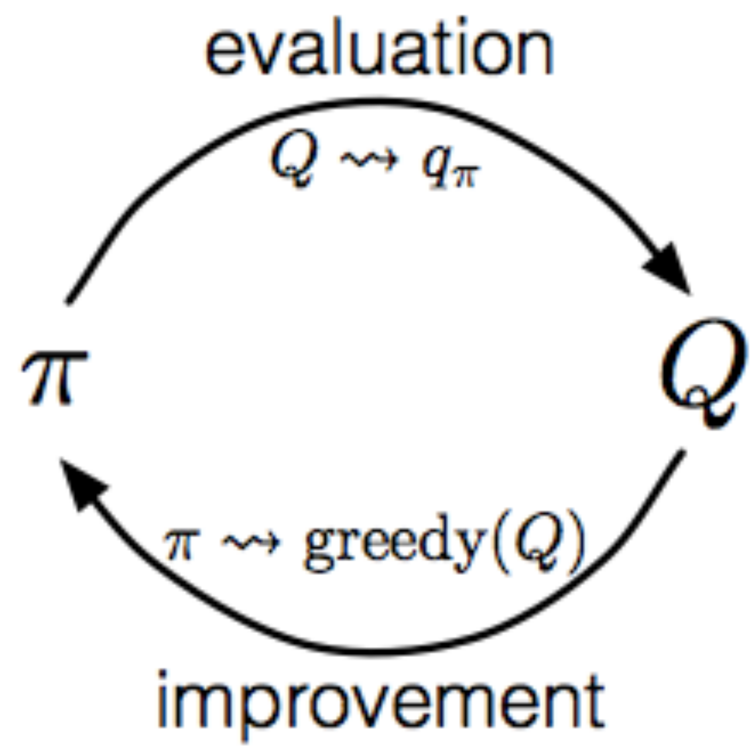


Summary so far

- Unknown dynamics: estimate value functions and optimal policies using Monte Carlo
  - Monte Carlo Prediction: estimate the value function of a given policy by deploying it, collect episodes and average their returns.
  - Next: Monte Carlo control: find optimal policies by interaction

Monte-Carlo Control

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



- MC policy iteration step: Policy evaluation using MC methods followed by policy improvement
- Policy improvement step: greedify with respect to value (or action-value) function

Greedy Policy

- For any action-value function q, the corresponding greedy policy is the one that:
  - For each s, deterministically chooses an action with maximal action-value:

$$\pi(s) \doteq \arg\max_a q(s, a).$$

- Policy improvement then can be done by constructing each π_{k+1} as the greedy policy with respect to $q_{\pi,k}$.

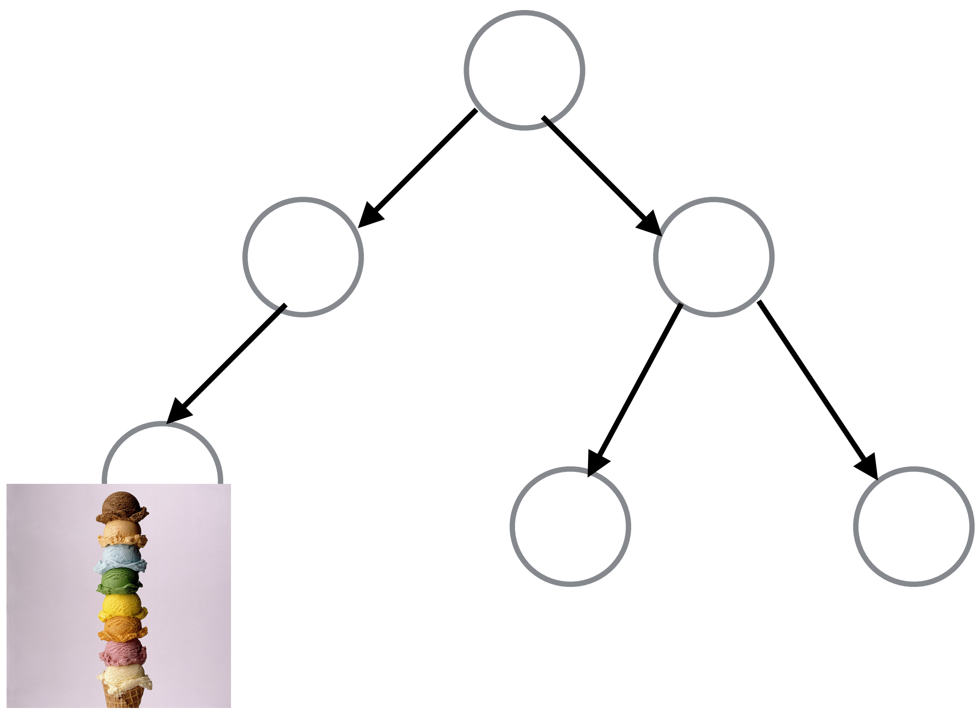
MC Estimation of Action Values (Q)

- Monte Carlo (MC) is most useful when a model is not available
  - We want to learn $q*(s, a)$ because then we can get an optimal policy without knowing dynamics.
- $q_\pi(s, a)$ - average return starting from state s and action a following π

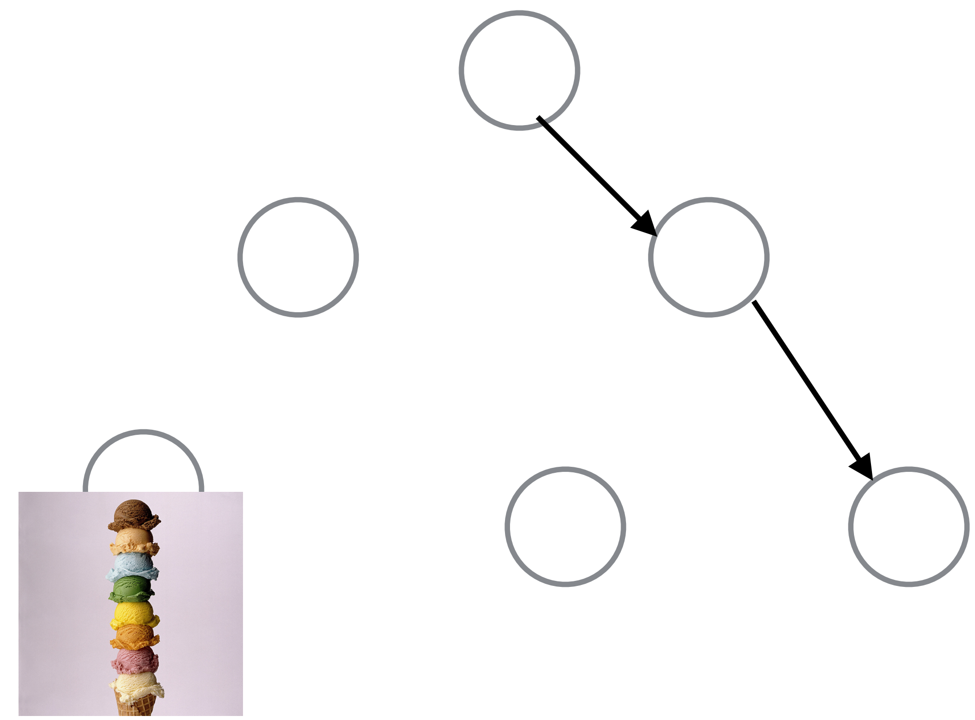
$$
\begin{aligned}
q_\pi(s,a) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_\pi(s')\Big].
\end{aligned}
$$

- Converges asymptotically if every state-action pair is visited.
  - Q: Is this possible if we are using a deterministic policy?

Dynamic Programming



Trial-and-error learning



In trial-and-error learning the state transitions are not available to you unless you visit them.

The Exploration problem

- If we always follow the deterministic policy to collect experience, we will never have the opportunity to see and evaluate (estimate q) of alternative actions…

- ALL learning methods face a dilemma: they seek to learn action values conditioned on subsequent optimal behaviour but they need to act suboptimally in order to explore all actions (to discover the optimal actions). The exploration-exploitation dilemma.

- Q: Does a learning algorithm know when the optimal policy has been reached to stop exploring?

The Exploration problem

- If we always follow the deterministic policy to collect experience, we will never have the opportunity to see and evaluate (estimate q) of alternative actions…

- ALL learning methods face a dilemma: they seek to learn action values conditioned on subsequent optimal behaviour but they need to act suboptimally in order to explore all actions (to discover the optimal actions). The exploration-exploitation dilemma.

- Solutions:

  1. exploring starts: Every state-action pair has a non-zero probability of being the starting pair

  2. Give up on deterministic policies and only search over ϵ-soft policies

  3. Off-policy: use a different policy to collect experience than the one you care to evaluate

Monte Carlo Exploring Starts

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

Fixed point is optimal policy π^*

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:
$G \leftarrow$ return following the first occurrence of s, a
Append G to $Returns(s,a)$
$Q(s,a) \leftarrow \text{average}(Returns(s,a))$
For each s in the episode:
$\pi(s) \leftarrow \arg\max_a Q(s,a)$

Convergence of MC Control

- Greedified policy meets the conditions for policy improvement:

$$\begin{aligned} q_{\pi_k}(s, \pi_{k+1}(s)) &= q_{\pi_k}(s, \arg\max_a q_{\pi_k}(s,a)) \\ &= \max_a q_{\pi_k}(s,a) \\ &\geq q_{\pi_k}(s, \pi_k(s)) \\ &\geq v_{\pi_k}(s). \end{aligned}$$

- And thus must be ≥ π_k.

- This assumes exploring starts and infinite number of episodes for MC policy evaluation

On-policy Monte Carlo Control

- On-policy: learn about policy currently executing
- How do we get rid of exploring starts?
  - The policy must be eternally soft: $\pi(a \mid s) > 0$ for all s and a.
- For example, for ε-soft policy, probability of an action, π(a|s),

$$= \frac{\epsilon}{|\mathcal{A}(s)|} \quad \text{or} \quad 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(s)|}$$

non-max max (greedy)

- Similar to GPI: move policy towards greedy policy
- Converges to the best ε-soft policy.

ϵ − soft Policies

- They keep choosing suboptimal actions even when the best one has been discovered.
- The second best action is as bad as the worst action.
- However, we will stick with them till we figure out better exploration methods later in the course.

On-policy Monte Carlo Control

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:

    $G \leftarrow$ return following the first occurrence of s, a

    Append G to $Returns(s,a)$

    $Q(s,a) \leftarrow$ average$(Returns(s,a))$

  (c) For each s in the episode:

    $A^* \leftarrow \arg\max_a Q(s,a)$

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

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

Off-policy methods

- Learn the value of the target policy π from experience due to behavior policy μ.

- For example, π is the greedy policy (and ultimately the optimal policy) while μ is exploratory (e.g., ε-soft) policy

- In general, we only require coverage, i.e., that μ generates behavior that covers, or includes, π:

$$\mu(a|s) > 0 \quad \text{for every } s, a \text{ at which} \quad \pi(a|s) > 0$$

- Q: can I average returns as before to obtain the value function of π ?

Off-policy methods

- Learn the value of the target policy π from experience due to behavior policy μ.

- For example, π is the greedy policy (and ultimately the optimal policy) while μ is exploratory (e.g., ε-soft) policy

- In general, we only require coverage, i.e., that μ generates behavior that covers, or includes, π:

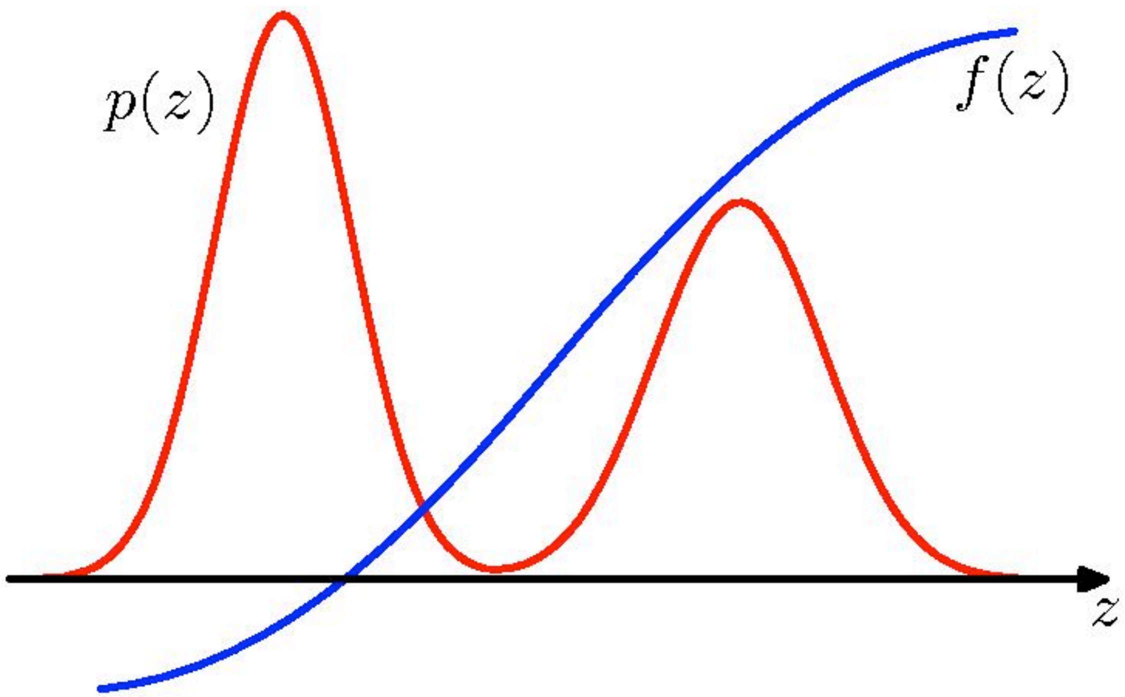
$$\mu(a|s) > 0 \quad \text{for every } s,a \text{ at which} \quad \pi(a|s) > 0$$

- Idea: Importance Sampling:
  - Weight each return by the ratio of the probabilities of the trajectory under the two policies.

Estimating Expectations

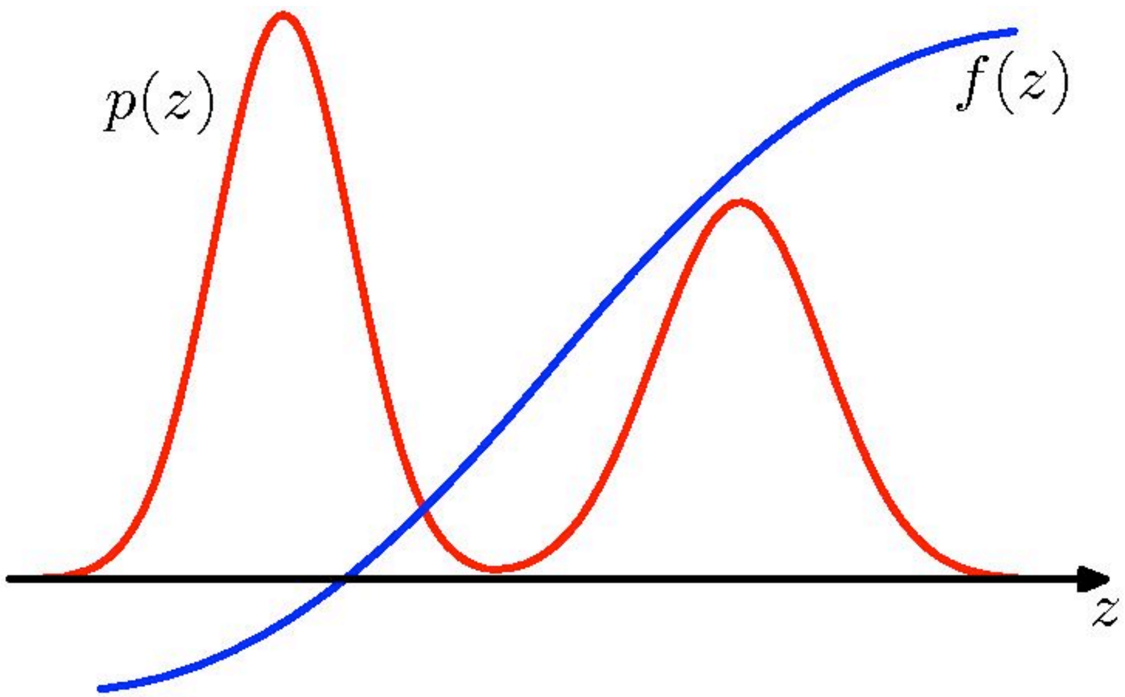
- General Idea: Draw independent samples $\{z^1, .., z^n\}$ from distribution p(z) to approximate expectation:

$$\mathbb{E}[f] = \int f(z)p(z)dz \approx \frac{1}{N}\sum_{n=1}^{N} f(z^n) = \hat{f}.$$



Estimating Expectations

- General Idea: Draw independent samples $\{z^1, .., z^n\}$ from distribution p(z) to approximate expectation:



$$\mathbb{E}[f] = \int f(z)p(z)dz \approx$$

$$\frac{1}{N}\sum_{n=1}^{N} f(z^n) = \hat{f}.$$

Note that: $\mathbb{E}[f] = \mathbb{E}[\hat{f}].$

so the estimator has correct mean (unbiased).

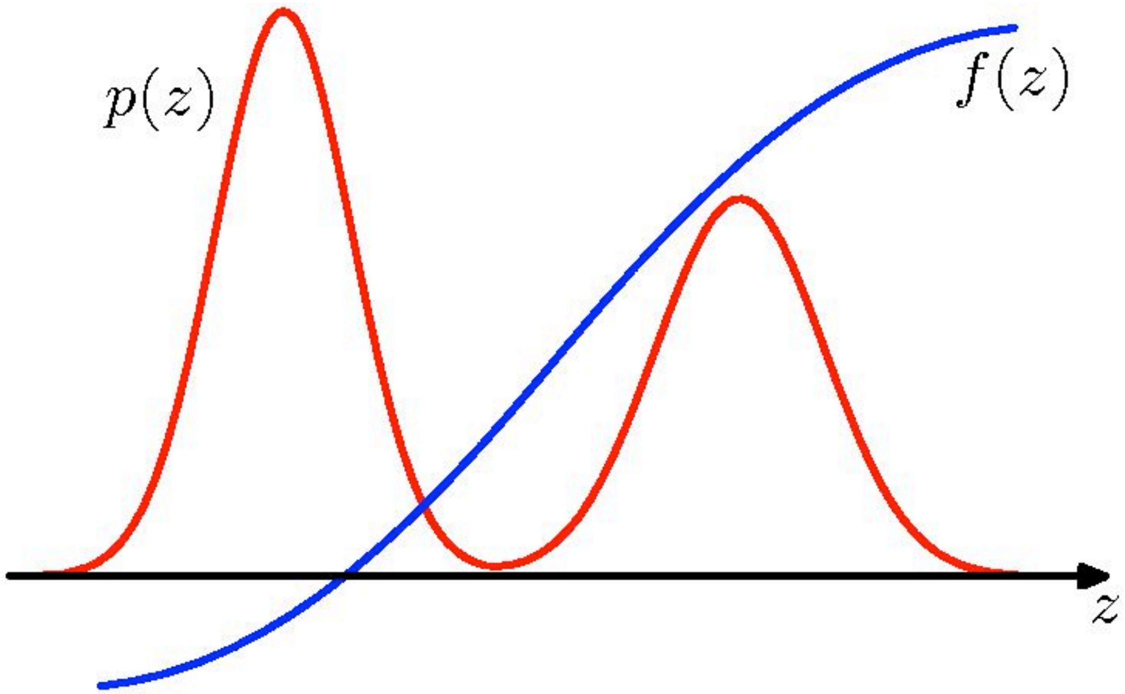
- The variance:

$$\mathrm{var}[\hat{f}] = \frac{1}{N}\mathbb{E}\left[(f - \mathbb{E}[f])^2\right].$$

- Variance decreases as 1/N.

Estimating Expectations

- General Idea: Draw independent samples $\{z^1, .., z^n\}$ from distribution p(z) to approximate expectation:



$$\mathbb{E}[f] = \int f(z)p(z)dz \approx \frac{1}{N}\sum_{n=1}^{N} f(z^n) = \hat{f}.$$

Note that: $\mathbb{E}[f] = \mathbb{E}[\hat{f}].$

so the estimator has correct mean (unbiased).

- The variance:

$$\text{var}[\hat{f}] = \frac{1}{N}\mathbb{E}\left[(f - \mathbb{E}[f])^2\right].$$

- Variance decreases as 1/N.

- Remark: The accuracy of the estimator does not depend on dimensionality of z.

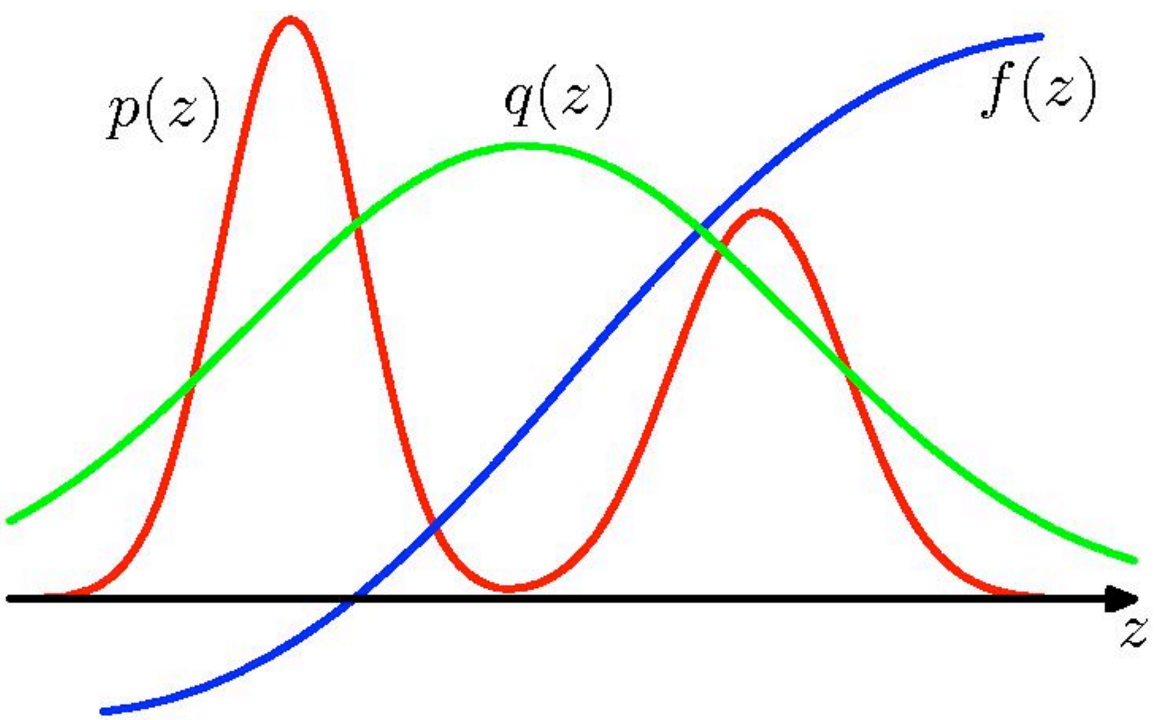
Importance Sampling

- Suppose we have an easy-to-sample proposal distribution q(z), such that $q(z) > 0 \text{ if } p(z) > 0.$

$$\mathbb{E}[f] = \int f(z)p(z)dz$$

$$= \int f(z)\frac{p(z)}{q(z)}q(z)dz$$

$$\approx \frac{1}{N}\sum_n \frac{p(z^n)}{q(z^n)}f(z^n), \quad z^n \sim q(z).$$



- The quantities

$$w^n = p(z^n)/q(z^n)$$

are known as importance weights.

- This is useful when we can evaluate the probability p but is hard to sample from it

Importance Sampling

- Let our proposal be of the form: $q(z) = \tilde{q}(z)/\mathcal{Z}_q$.

$$\mathbb{E}[f] = \int f(z)p(z)dz \quad = \int f(z)\frac{p(z)}{q(z)}q(z)dz = \frac{Z_q}{Z_p}\int f(z)\frac{\tilde{p}(z)}{\tilde{q}(z)}q(z)dz$$

$$\approx \frac{\mathcal{Z}_q}{\mathcal{Z}_p}\frac{1}{N}\sum_n \frac{\tilde{p}(z^n)}{\tilde{q}(z^n)}f(z^n) = \frac{\mathcal{Z}_q}{\mathcal{Z}_p}\frac{1}{N}\sum_n w^n f(z^n),$$

- But we can use the same weights to approximate $\mathcal{Z}_q/\mathcal{Z}_p$:

$$\frac{\mathcal{Z}_p}{\mathcal{Z}_q} = \frac{1}{\mathcal{Z}_q}\int \tilde{p}(z)dz = \int \frac{\tilde{p}(z)}{\tilde{q}(z)}q(z)dz \approx \frac{1}{N}\sum_n \frac{\tilde{p}(z^n)}{\tilde{q}^n(z)} = \frac{1}{N}\sum_n w^n.$$

- Hence:

$$\mathbb{E}[f] \approx \sum_{n=1}^{N} \frac{w^n}{\sum_{m=1}^{N} w^m} f(z^n), \qquad z^n \sim q(z).$$

Importance Sampling Ratio

- Probability of the rest of the trajectory, after S_t, under policy π:

$$
\begin{aligned}
\Pr\{A_t, S_{t+1}, A_{t+1}, \ldots, S_T \mid S_t, A_{t:T-1} \sim \pi\} \\
&= \pi(A_t|S_t)p(S_{t+1}|S_t, A_t)\pi(A_{t+1}|S_{t+1}) \cdots p(S_T|S_{T-1}, A_{T-1}) \\
&= \prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k),
\end{aligned}
$$

- Importance Sampling: Each return is weighted by the relative probability of the trajectory under the target and behavior policies

$$
\rho_t^T = \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)}{\prod_{k=t}^{T-1} \mu(A_k|S_k)p(S_{k+1}|S_k, A_k)} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{\mu(A_k|S_k)}
$$

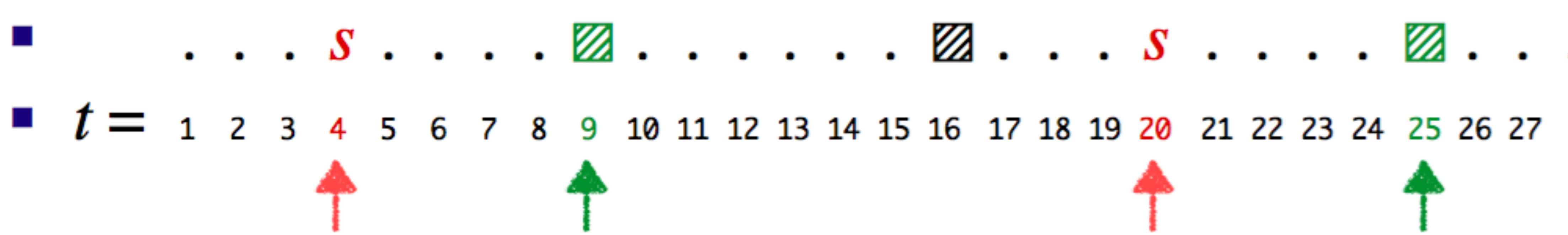
- This is called the Importance Sampling Ratio

Importance Sampling

- Ordinary importance sampling forms estimate

$$V(s) \doteq \frac{\sum_{t \in \mathfrak{T}(s)} \rho_t^{T(t)} G_t}{|\mathfrak{T}(s)|}.$$

- New notation: time steps increase across episode boundaries:



$\mathfrak{T}(s) = \{4, 20\}$ set of start times

$T(4) = 9 \quad T(20) = 25$ next termination times

Importance Sampling

- Ordinary importance sampling forms estimate

First time of termination following time t

return after t up through T(t)

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_t^{T(t)} G_t}{|\mathcal{T}(s)|}.$$

Every time: the set of all time steps in which state s is visited

Importance Sampling Ratio

- All importance sampling ratios have expected value 1:

$$\mathbb{E}_{A_k \sim \mu}\left[\frac{\pi(A_k|S_k)}{\mu(A_k|S_k)}\right] = \sum_a \mu(a|S_k)\frac{\pi(a|S_k)}{\mu(a|S_k)} = \sum_a \pi(a|S_k) = 1.$$

- Note: Importance Sampling can have high (or infinite) variance.

Importance Sampling

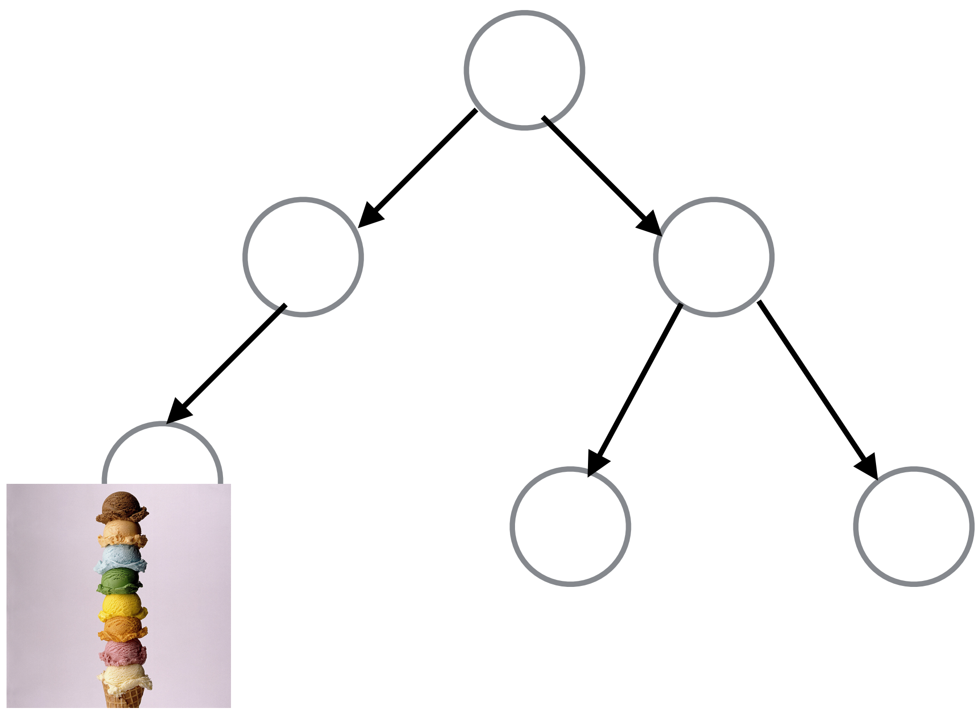
- Two ways of averaging weighted returns:
  - Ordinary importance sampling forms estimate:

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_t^{T(t)} G_t}{|\mathcal{T}(s)|}.$$

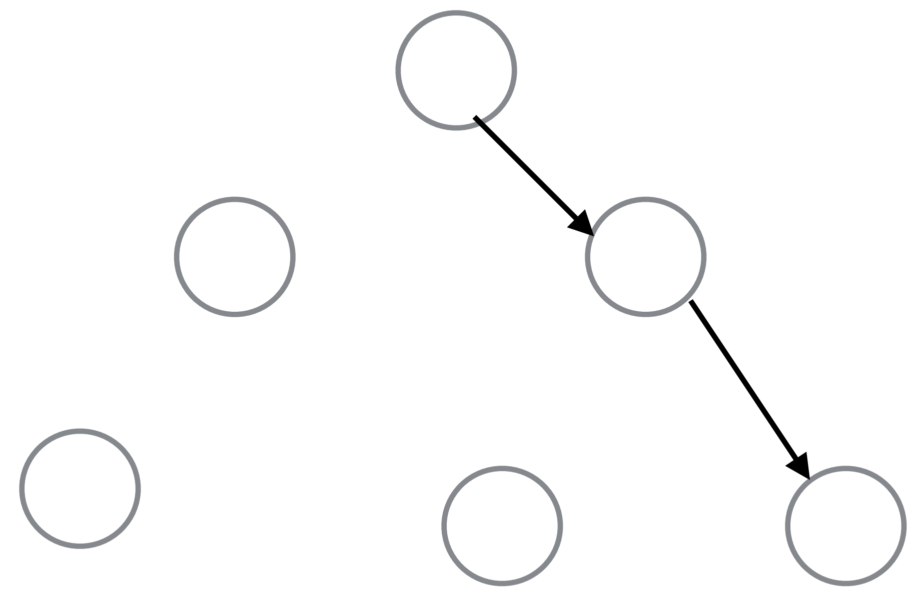
  - Weighted importance sampling forms estimate:

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_t^{T(t)} G_t}{\sum_{t \in \mathcal{T}(s)} \rho_t^{T(t)}}$$

Dynamic Programming



Trial-and-error learning



In trial-and-error learning the state transitions are not available to you unless you visit them.

So far

- MC has several advantages over DP:
  - Can learn directly from interaction with environment
  - No need for full models

- MC methods provide an alternate policy evaluation process
- One issue to watch for: maintaining sufficient exploration
- Looked at distinction between on-policy and off-policy methods

MC and TD Learning

- Goal: learn $v_\pi(s)$ from episodes of experience under policy π
- Incremental every-visit Monte-Carlo:
  - Update value $V(S_t)$ toward actual return G_t:

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

- Simplest Temporal-Difference learning algorithm: TD(0)
  - Update value V(St) toward estimated returns $R_{t+1} + \gamma V(S_{t+1})$

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

- $R_{t+1} + \gamma V(S_{t+1})$ is called the TD target
- $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the TD error.

DP vs. MC vs. TD Learning

- Remember:

MC: sample average return approximates expectation

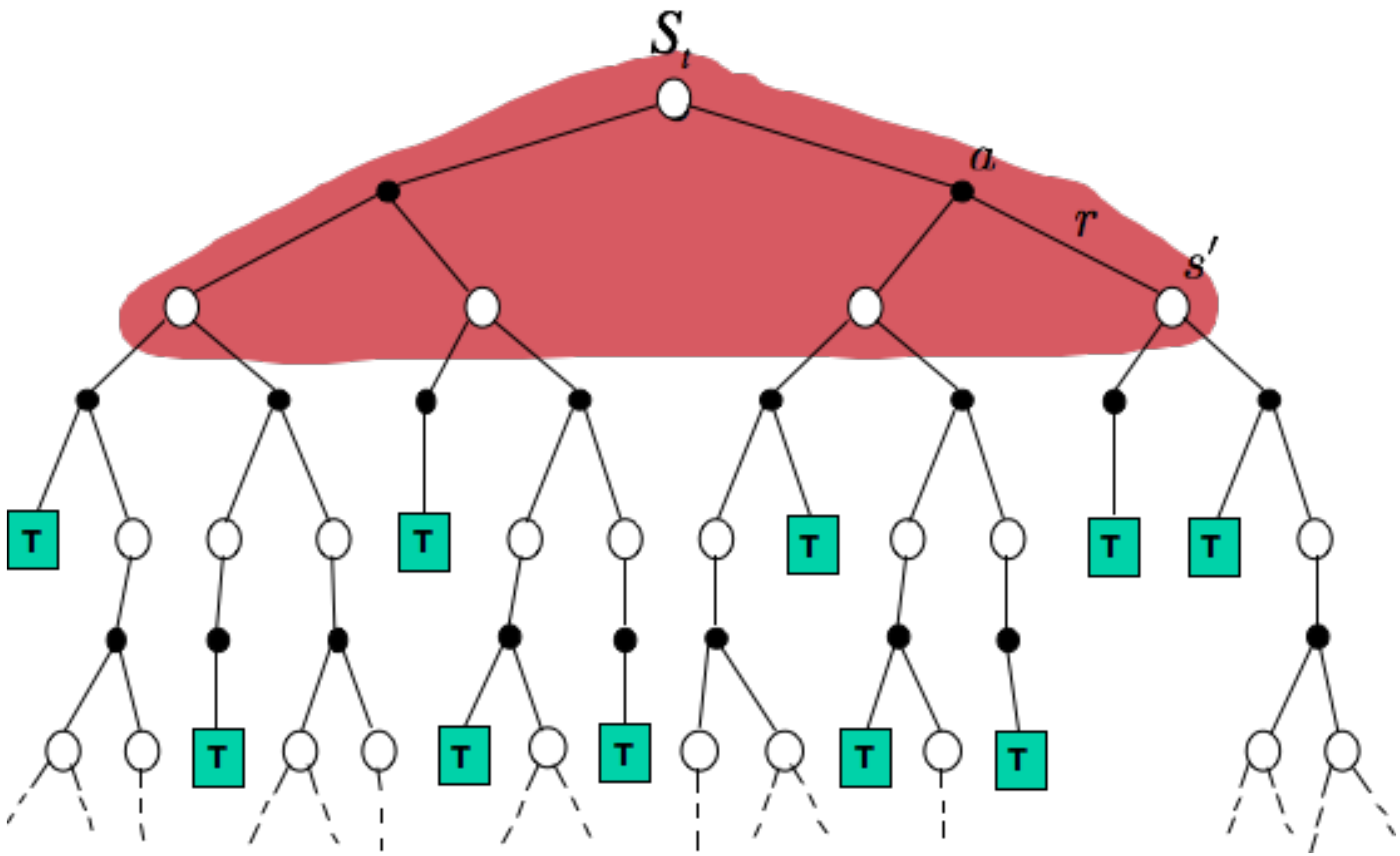
$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \,\middle|\, S_t = s\right] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] .
\end{aligned}
$$

TD: combine both: Sample expected values and use a current estimate $V(S_{t+1})$ of the true $v_\pi(S_{t+1})$

DP: the expected values are provided by a model. But we use a current estimate $V(S_{t+1})$ of the true $v_\pi(S_{t+1})$.

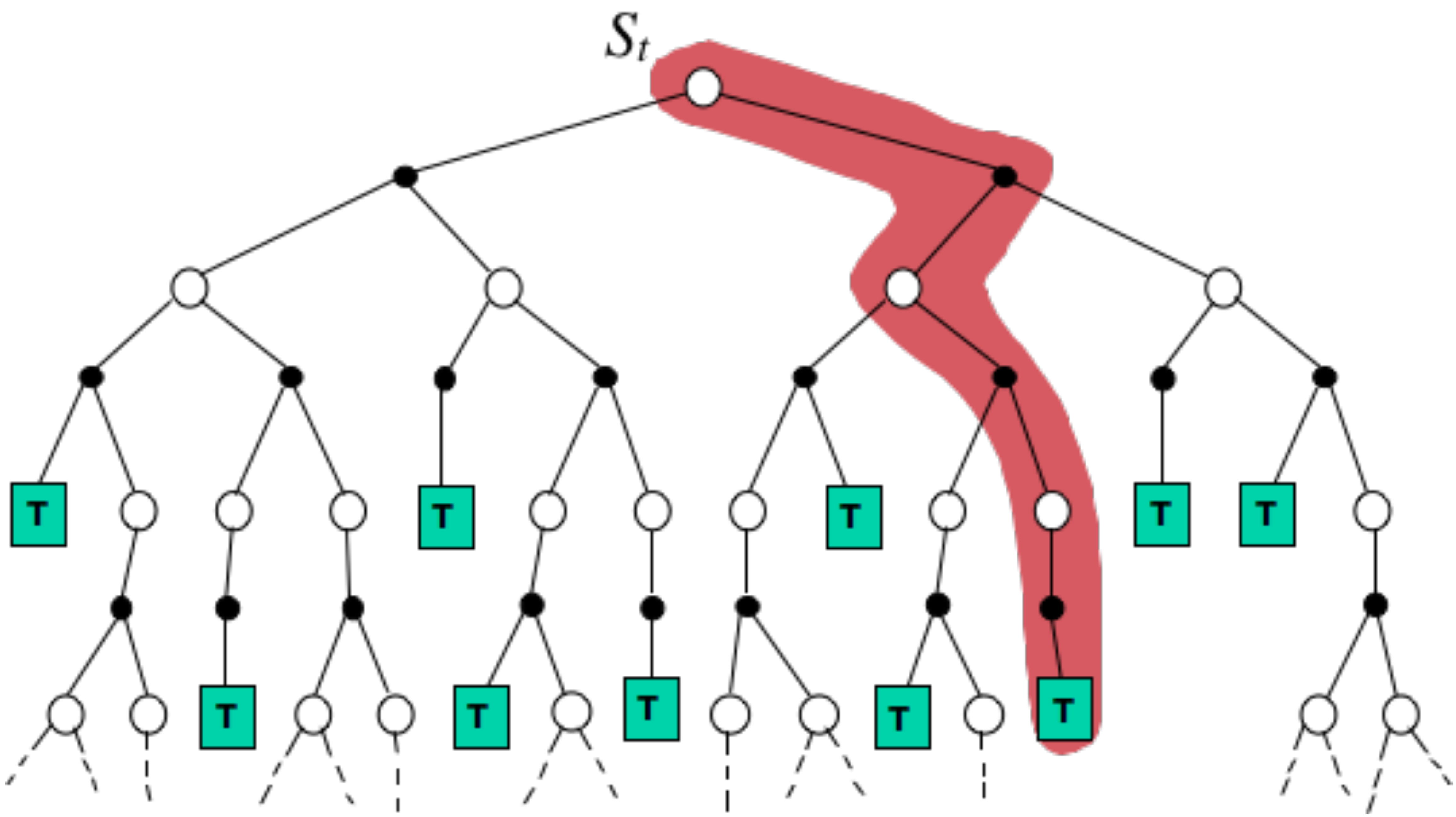
Dynamic Programming

$$V(S_t) \leftarrow E_\pi\left[R_{t+1} + \gamma V(S_{t+1})\right] = \sum_a \pi(a|S_t) \sum_{s',r} p(s',r|S_t,a)[r + \gamma V(s')]$$



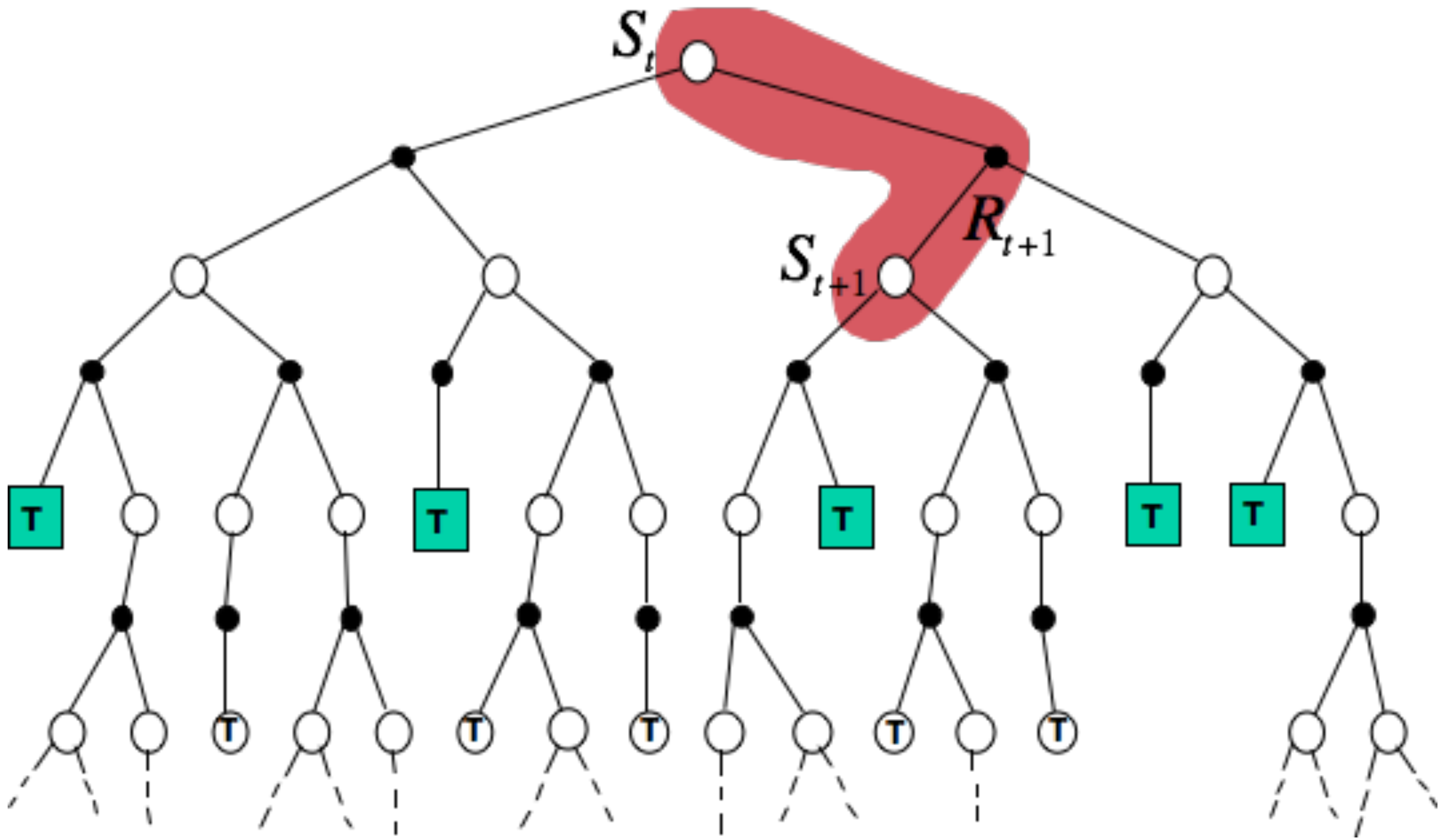
Monte Carlo

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



Simplest TD(0) Method

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



TD Methods Bootstrap and Sample

- Bootstrapping: update involves an estimate
  - MC does not bootstrap
  - DP bootstraps
  - TD bootstraps
- Sampling: update does not involve an expected value
  - MC samples
  - DP does not sample
  - TD samples

TD Prediction

- Policy Evaluation (the prediction problem):
  - for a given policy π, compute the state-value function v_π.
- Remember: Simple every-visit Monte Carlo method:

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

target: the actual return after time t

- The simplest Temporal-Difference method TD(0):

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

target: an estimate of the return

Example: Driving Home

State	*Elapsed Time (minutes)*	*Predicted Time to Go*	*Predicted Total Time*
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43

- Simple every-visit Monte Carlo method:

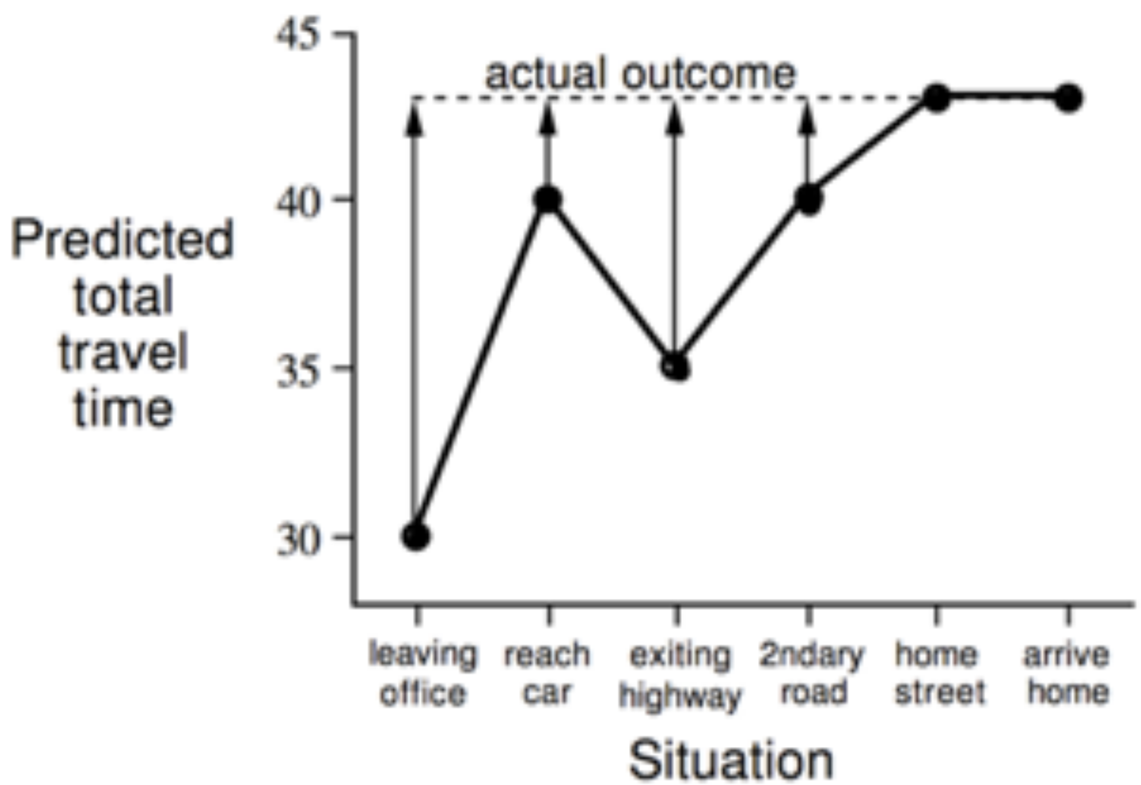
$$V(S_t) \leftarrow V(S_t) + \alpha\Big[G_t - V(S_t)\Big]$$

- The simplest Temporal-Difference method TD(0):

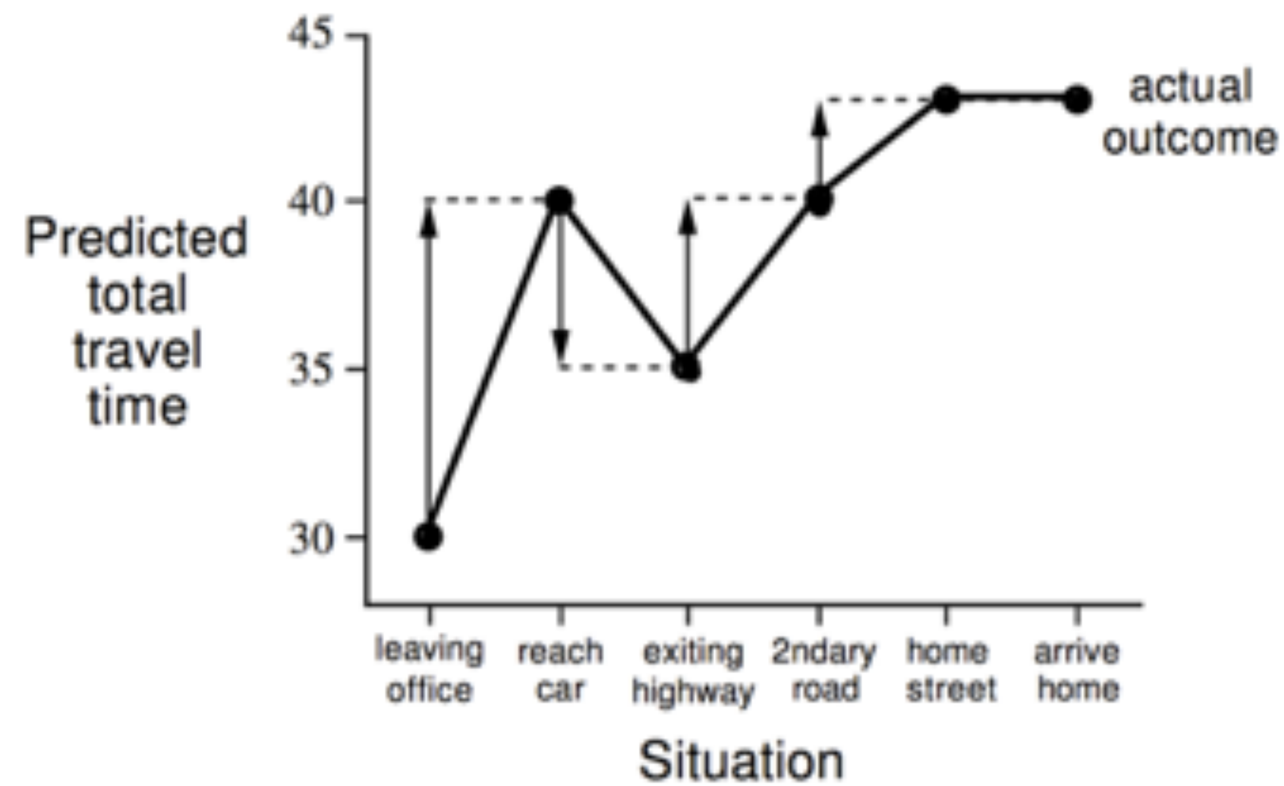
$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$

Example: Driving Home

Changes recommended by Monte Carlo methods (α=1)



Changes recommended by TD methods (α=1)



Advantages of TD Learning

- TD methods do not require a model of the environment, only experience
- TD, but not MC, methods can be fully incremental
- You can learn before knowing the final outcome
  - Less memory
  - Less computation
- You can learn without the final outcome
  - From incomplete sequences
- Both MC and TD converge (under certain assumptions to be detailed later), but which is faster?

Bias-Variance Trade-Off

- Monte-Carlo: Update value $V(S_t)$ toward actual return G_t

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

- Return $G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T$ is unbiased estimate $v_\pi(S_t)$

- TD: Update value V(St) toward estimated returns $R_{t+1} + \gamma V(S_{t+1})$

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

  - True TD target: $R_{t+1} + \gamma v_\pi(S_{t+1})$ is unbiased estimate ($v_\pi(S_t)$

    - TD target: $R_{t+1} + \gamma V(S_{t+1})$ is biased estimate of $v_\pi(S_t)$.

- 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

AB Example

- Suppose you observe the following 8 episodes:

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

$V(\mathrm{B})$? 0.75

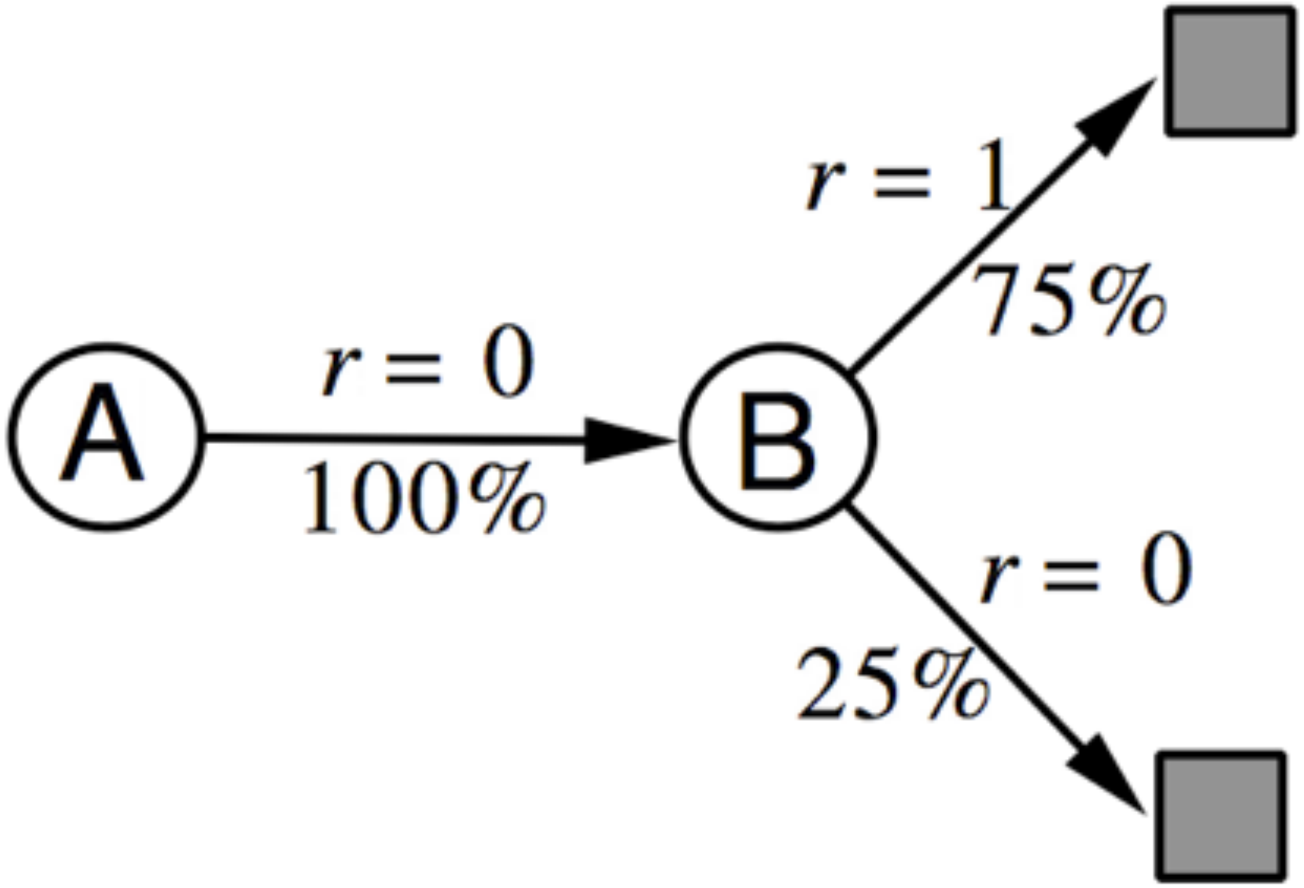
$V(\mathrm{A})$? 0

- Assume Markov states, no discounting (γ = 1)

AB Example

- The prediction that best matches the training data is V(A)=0
  - This minimizes the mean-square-error on the training set
  - This is what a batch Monte Carlo method gets

AB Example



$V(\mathrm{A})$? 0.75

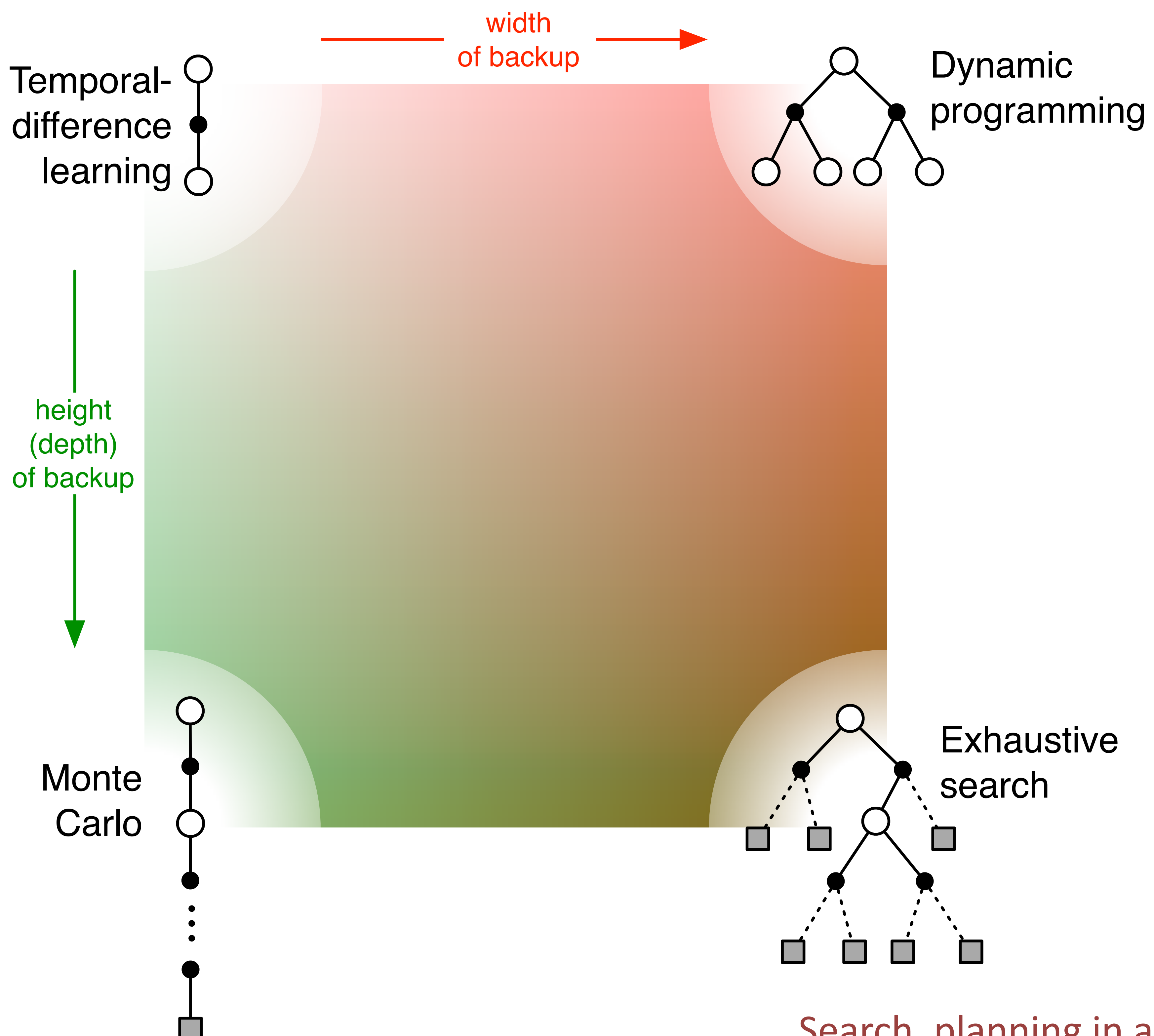
AB Example

- The prediction that best matches the training data is V(A)=0
  - This minimizes the mean-square-error on the training set
  - This is what a batch Monte Carlo method gets
- If we consider the sequentiality of the problem, then we would set V(A)=.75
  - This is correct for the maximum likelihood estimate of a Markov model generating the data
  - i.e, if we do a best fit Markov model, and assume it is exactly correct, and then compute what it predicts.
  - This is called the certainty-equivalence estimate
  - This is what TD gets

Summary so far

- Introduced one-step tabular model-free TD methods
- These methods bootstrap and sample, combining aspects of DP and MC methods

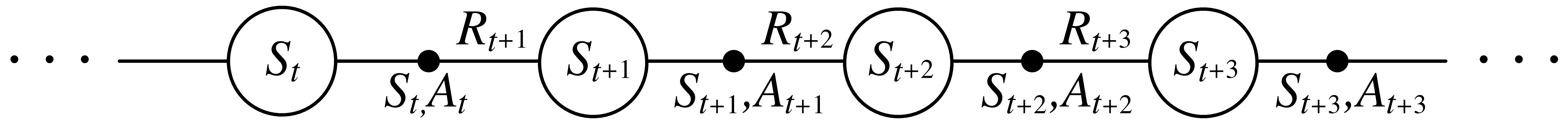
Unified View



Search, planning in a later lecture!

Learning An Action-Value Function

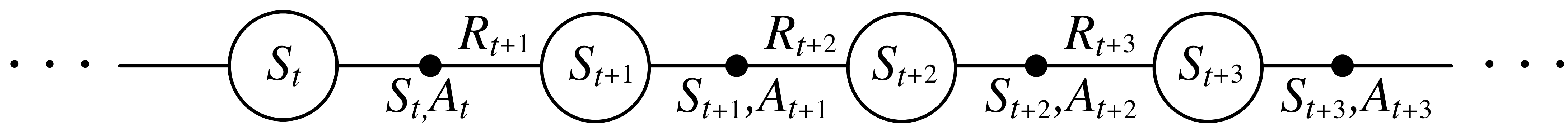
- Estimate q_π for the current policy π



- Can we come up with the TD update equation for Q values?

Learning An Action-Value Function

- Estimate q_π for the current policy π



After every transition from a nonterminal state, S_t, do this:

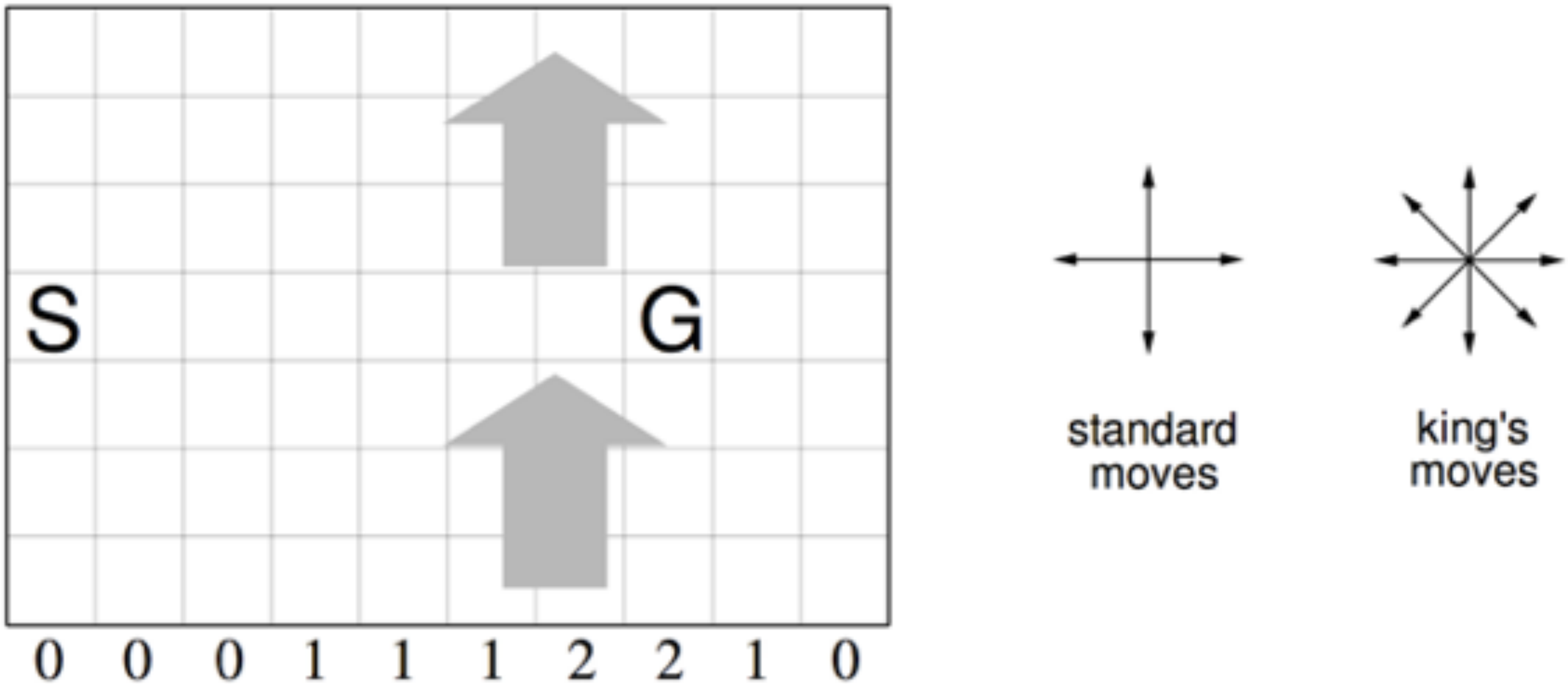
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right]$$

If S_{t+1} is terminal, then define $Q(S_{t+1}, A_{t+1}) = 0$

SARSA: On-Policy TD Control

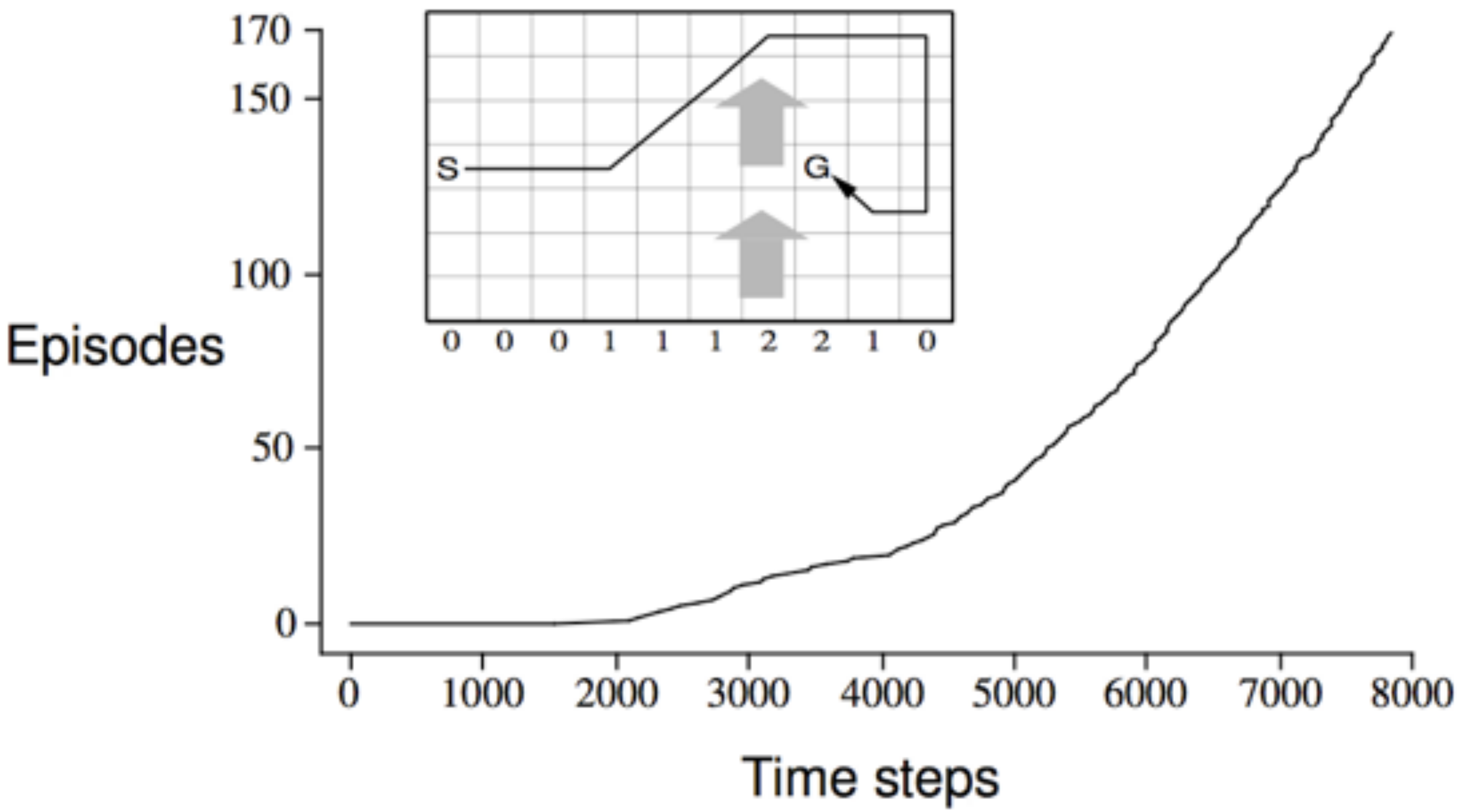
Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Repeat (for each step of episode):
        Take action A, observe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

Windy Gridworld



- undiscounted, episodic, reward = –1 until goal

Results of SARSA on the Windy Gridworld



Q: Can a policy result in infinite loops? What will MC policy iteration do then?

- If the policy leads to infinite loop states, MC control will get trapped as the episode will not terminate.
- Instead, TD control can update continually the state-action values and switch to a different policy.

Q-Learning: Off-Policy TD Control

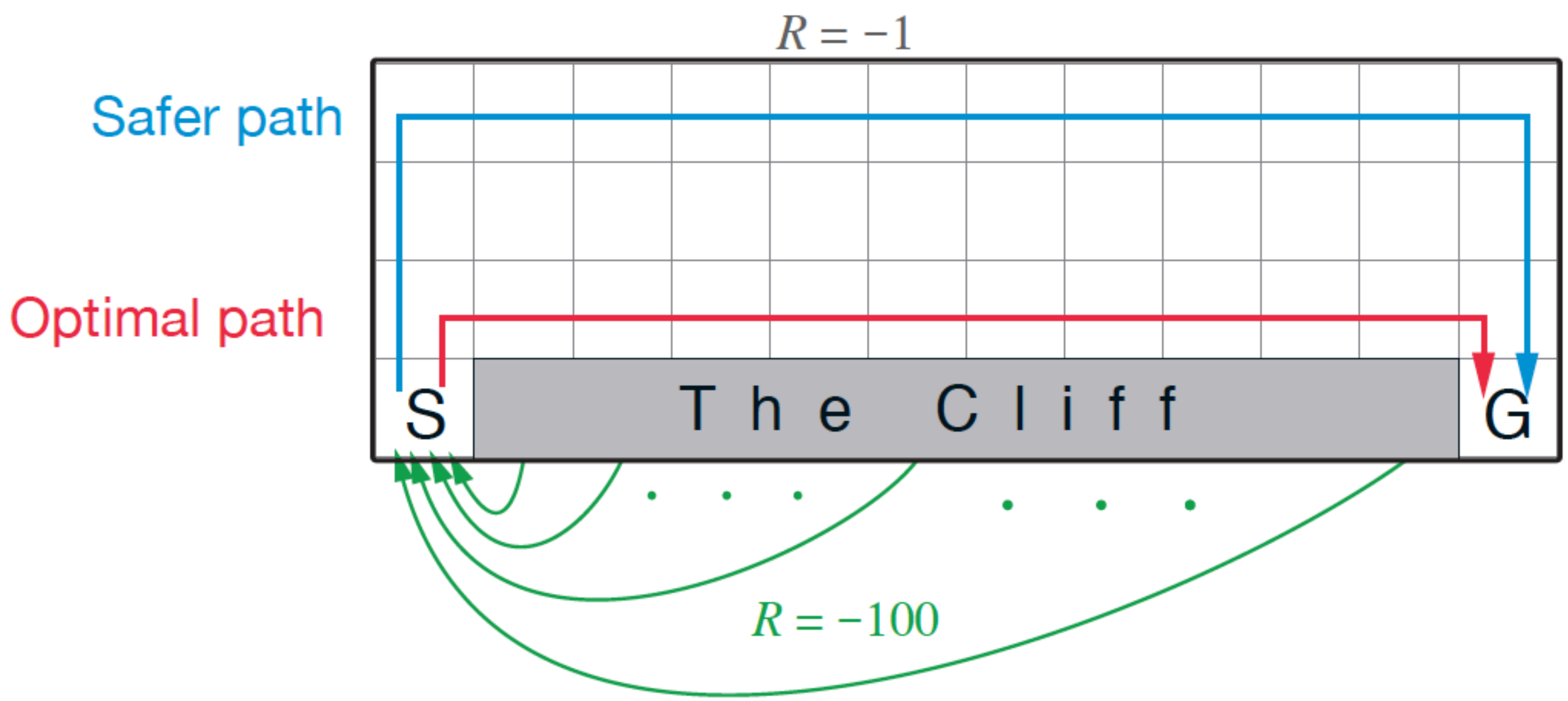
- One-step Q-learning:

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

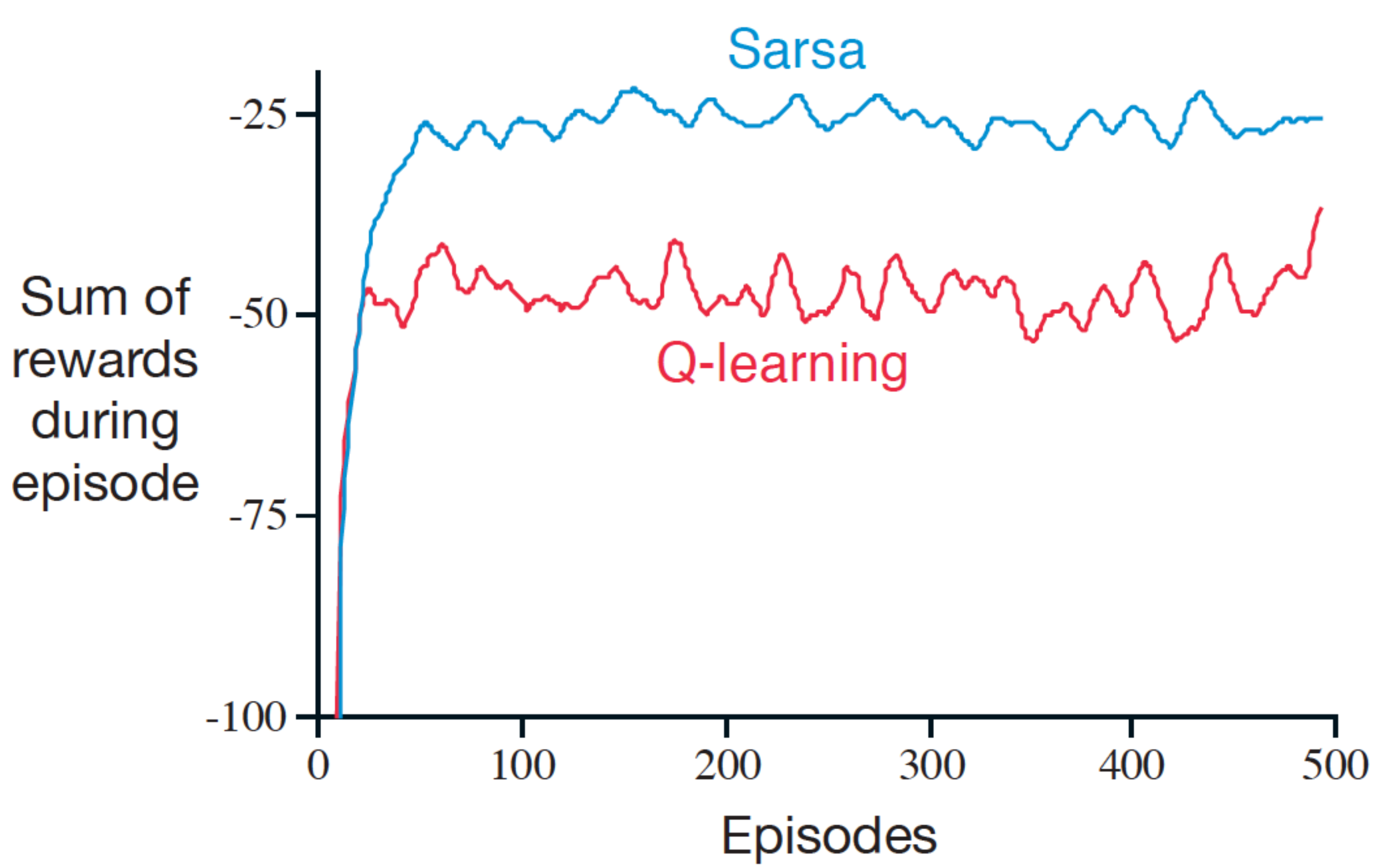
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R,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

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

Cliff-walking



ϵ − greedy, $\epsilon = 0.1$



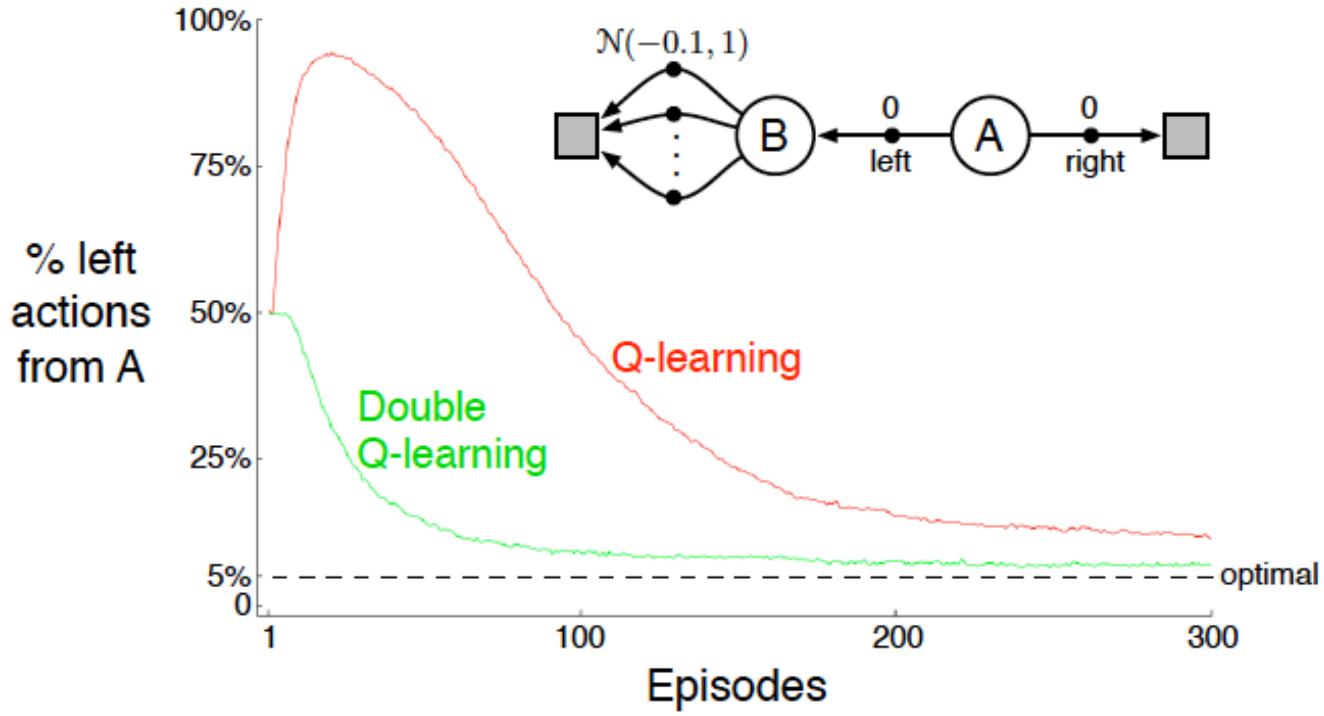
Maximization Bias

- We often need to maximize over our value estimates. The estimated maxima suffer from maximization bias

- Consider a state for which all action a, $q_*(s,a) = 0$. Our estimates $Q(s,a)$ are uncertain, some are positive and some negative.

- $Q(s, \underset{a}{\operatorname{argmax}}\, Q(s,a)) > 0$ while $q_*(s, \underset{a}{\operatorname{argmax}}\, q_*(s,a)) = 0$.

- This is because we use the same estimate Q both to choose the argmax and to evaluate it.



Double Q-Learning

- Train 2 action-value functions, Q1 and Q2
- Do Q-learning on both, but
  - never on the same time steps (Q1 and Q2 are independent)
  - pick Q1 or Q2 at random to be updated on each step
- If updating Q1, use Q2 for the value of the next state:

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

- Action selections are ε-greedy with respect to the sum of Q1 and Q2

Double Tabular Q-Learning

Initialize $Q_1(s, a)$ and $Q_2(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily
Initialize $Q_1(\textit{terminal-state}, \cdot) = Q_2(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        Choose A from S using policy derived from Q_1 and Q_2 (e.g., ε-greedy in $Q_1 + Q_2$)
        Take action A, observe R, S'
        With 0.5 probabilility:

$$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha\Big(R + \gamma Q_2\big(S', \arg\max_a Q_1(S', a)\big) - Q_1(S, A)\Big)$$

        else:

$$Q_2(S, A) \leftarrow Q_2(S, A) + \alpha\Big(R + \gamma Q_1\big(S', \arg\max_a Q_2(S', a)\big) - Q_2(S, A)\Big)$$

        $S \leftarrow S'$;
    until S is terminal

Hado van Hasselt 2010

Expected Sarsa

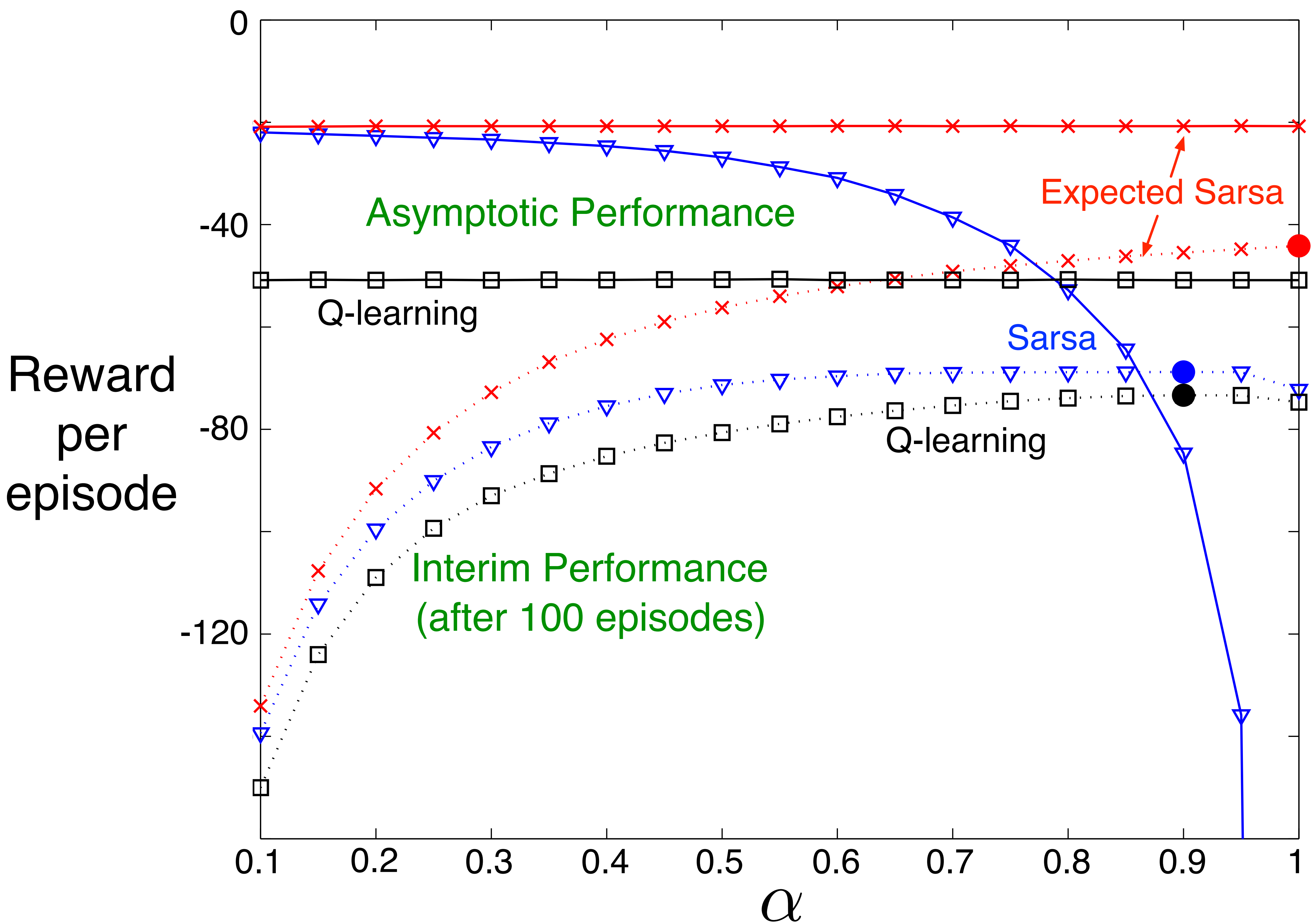
- Instead of the sample value-of-next-state, use the expectation!

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

- Expected Sarsa performs better than Sarsa (but costs more)
  - Q: why?

Q: Is expected SARSA on policy or off policy?
What if π is the greedy deterministic policy?

Performance on the Cliff-walking Task



Summary

- Introduced one-step tabular model-free TD methods
- These methods bootstrap and sample, combining aspects of DP and MC methods
- TD methods are computationally congenial

- Extend prediction to control by employing some form of GPI
    - On-policy control: Sarsa
    - Off-policy control: Q-learning