Class 2: Model-Free Prediction

Sutton and Barto,
Chapters 5 and 6

David Silver

Course Outline, Silver

- Part I: Elementary Reinforcement Learning
  1. Introduction to RL
  2. Markov Decision Processes
  3. Planning by Dynamic Programming
  4. Model-Free Prediction
  5. Model-Free Control
- Part II: Reinforcement Learning in Practice
  1. Value Function Approximation
  2. Policy Gradient Methods
  3. Integrating Learning and Planning
  4. Exploration and Exploitation
  5. Case study - RL in games

Model-Free Reinforcement Learning

- Last lecture:
  - Planning by dynamic programming
  - Solve a *known* MDP
- This lecture:
  - Model-free prediction
  - Estimate the value function of an *unknown* MDP
- This lecture:
  - Model-free control
  - Optimise the value function of an *unknown* MDP

Monte-Carlo Reinforcement Learning

MC methods can solve the RL problem by averaging sample returns

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

MC is incremental episode by episode but not step by step

Approach: adapting general policy iteration to sample returns

First policy evaluation, then policy improvement, then control

Monte-Carlo Policy Evaluation

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

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

- Recall that the *return* is the total discounted reward:

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

- Recall that the value function is the expected return:

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

- Monte-Carlo policy evaluation uses *empirical mean* return instead of *expected* return, because we do not have the model

First-Visit Monte-Carlo 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 $N(s) \rightarrow \infty$

First-Visit MC Estimate

First-visit MC prediction, for estimating $V \approx v_\pi$

Initialize:
  $\pi \leftarrow$ policy to be evaluated
  $V \leftarrow$ an arbitrary state-value function
  $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:
  Generate an episode using π
  For each state s appearing in the episode:
    $G \leftarrow$ the return that follows the first occurrence of s
    Append G to $Returns(s)$
    $V(s) \leftarrow \text{average}(Returns(s))$

In this case each return is an independent, identically distributed estimate of v_pi(s) with finite variance. By the law of large numbers the sequence of averages of these estimates converges to their expected value. The average is an unbiased estimate. The standard deviation of its error converges as inverse square-root of n where n is the number of returns averaged.

Every-Visit Monte-Carlo 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 $V(s) = S(s)/N(s)$
- Again, $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$

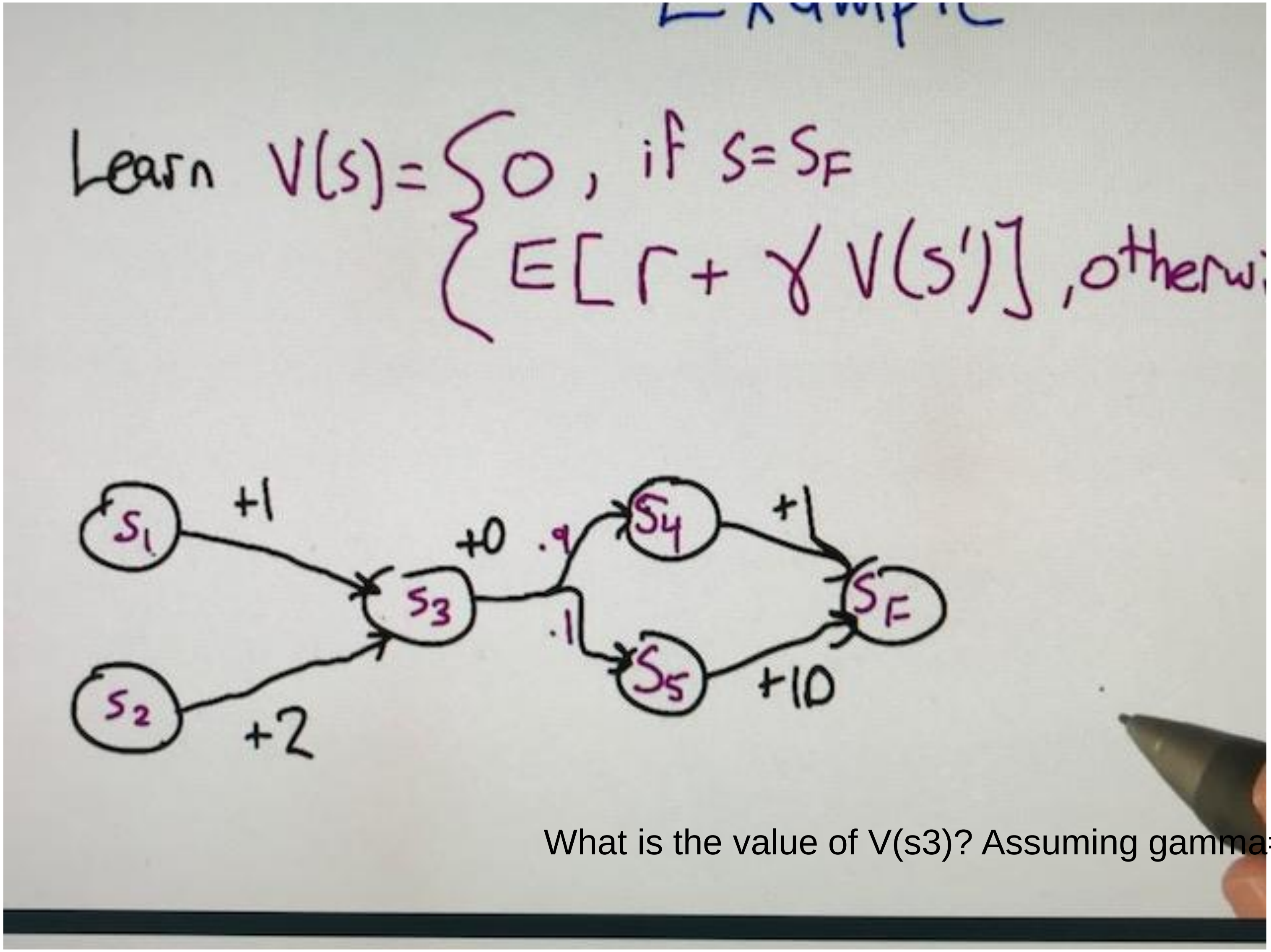
Can also be shown to converge

TD Lambda

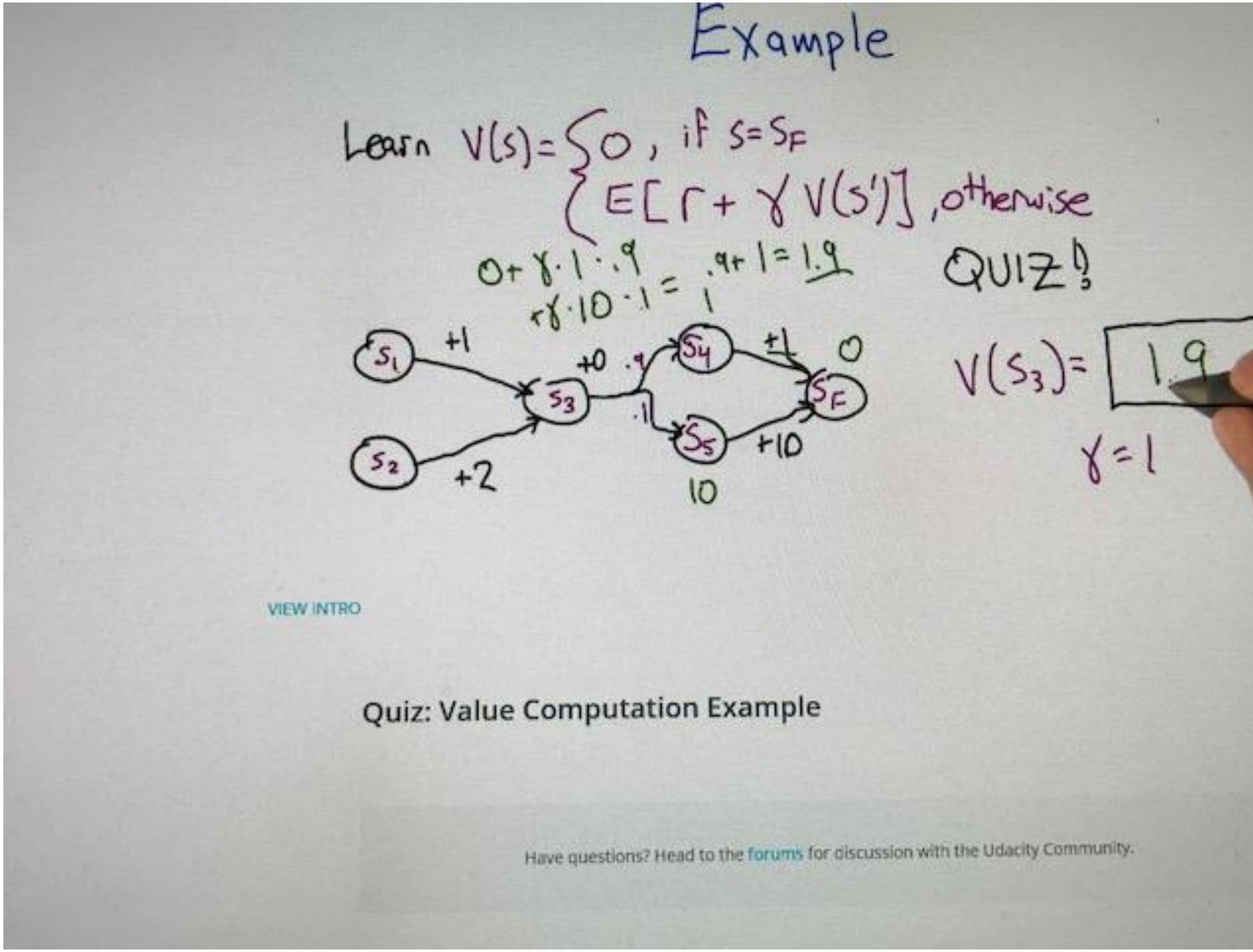
TD(λ)

Learning to predict over time.

$$s_0 \xrightarrow{r_0} s_1 \xrightarrow{r_1} s_2 \xrightarrow{r_2} s_F$$

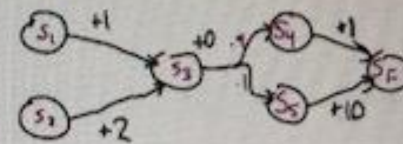


What is the value of V(s3)? Assuming gamma=1


Example
Learn V(s) = { 0 , if s = S_F
E[r + γ V(s')] , otherwise
0 + γ · 1 · .9
+ γ · 10 · .1 =
.9 + 1 = 1.9
1
QUIZ!
V(S_3) = 1.9
γ = 1
S_1
S_2
S_3
S_4
S_5
S_F
+1
+2
+0
.9
.1
+1
+10
0
10
VIEW INTRO
Quiz: Value Computation Example
Have questions? Head to the forums for discussion with the Udacity Community.

Estimating From Data

$\gamma = 1$



1. $s_1 \xrightarrow{+1} s_3 \xrightarrow{+0} s_4 \xrightarrow{+1} s_F$
2. $s_1 \xrightarrow{+1} s_3 \xrightarrow{+0} s_5 \xrightarrow{+10} s_F$
3. $s_1 \xrightarrow{+1} s_3 \xrightarrow{+0} s_4 \xrightarrow{+1} s_F$
4. $s_1 \xrightarrow{+1} s_3 \xrightarrow{+0} s_4 \xrightarrow{+1} s_F$
5. $s_2 \xrightarrow{+2} s_3 \xrightarrow{+0} s_5 \xrightarrow{+10} s_F$

QUIZ

What is an appropriate estimate for $V(s_1)$ after 3 episodes? 4?

Computing Estimates Incrementally

$V_{T-1}(s_1)$ $\quad R_T(s_1)$ $\quad V_T(s_1)$

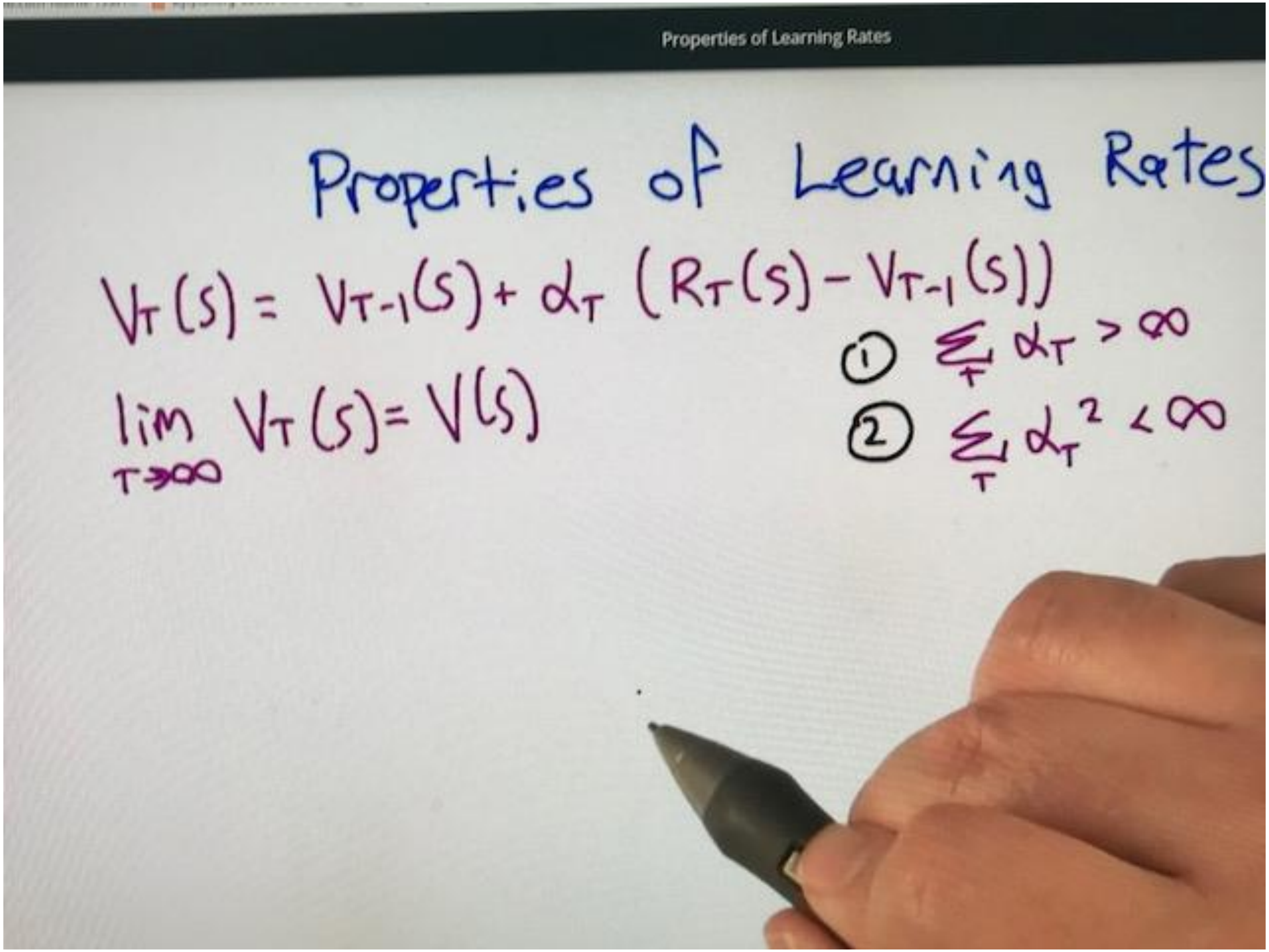
$$V_T(s_1) = \frac{(T-1)\,V_{T-1}(s_1) + R_T(s_1)}{T}$$

$$= \frac{T-1}{T}\,V_{T-1}(s_1) + \frac{1}{T}\,R_T(s_1)$$

$$= V_{T-1}(s_1) + \alpha_T\,(\underbrace{R_T(s_1) - V_{T-1}(s_1)}_{\text{error}})$$

where $\alpha_T = \frac{1}{T}$

T =number of episodes
Averaged over


Properties of Learning Rates
Properties of Learning Rates
$V_T(s) = V_{T-1}(s) + \alpha_T (R_T(s) - V_{T-1}(s))$
$\lim_{T \to \infty} V_T(s) = V(s)$
① $\sum_T \alpha_T > \infty$
② $\sum_T \alpha_T^2 < \infty$

Blackjack Example

Blackjack Example

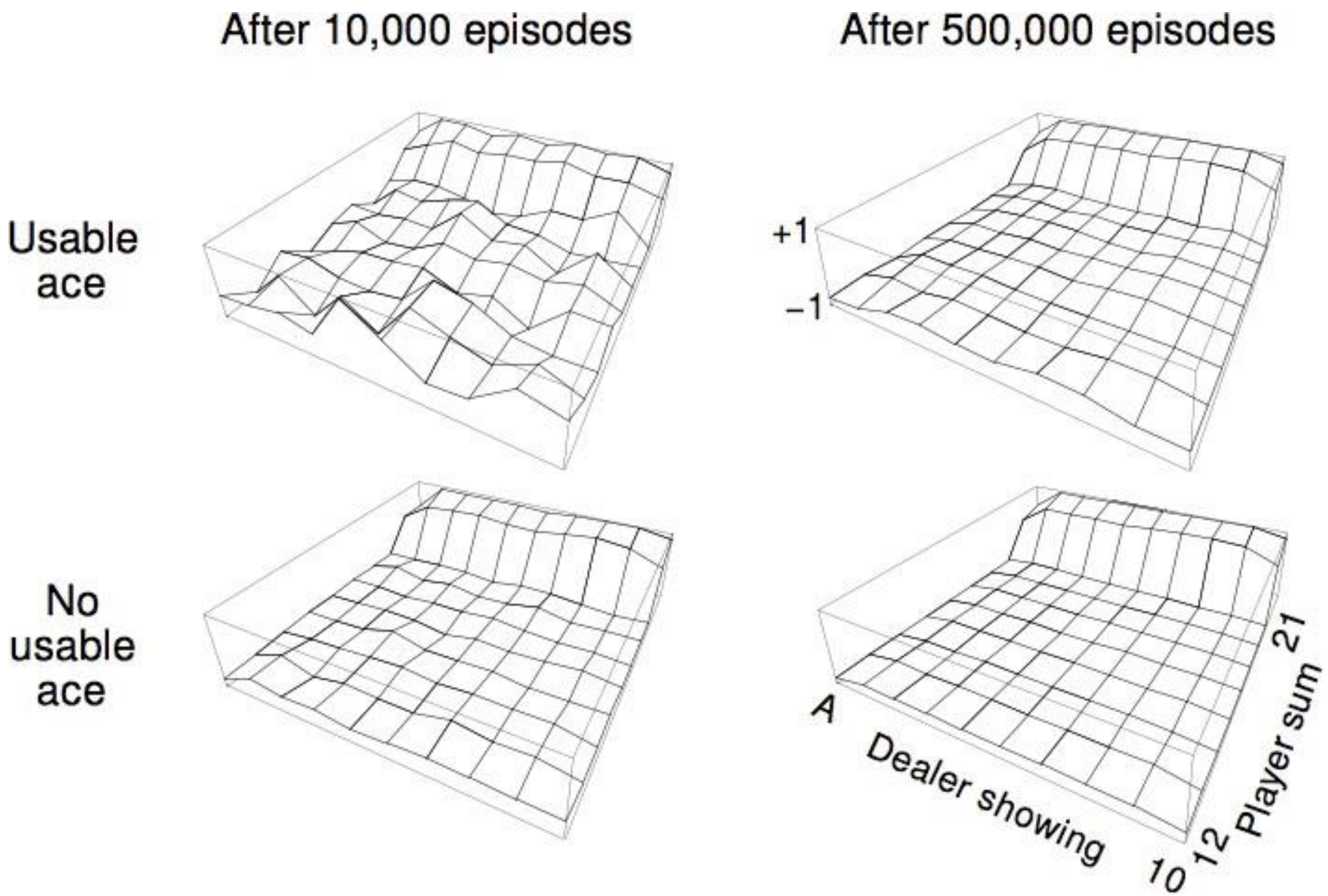
Each game is an episode
States: player cards and dealer's showing

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

Blackjack Value Function after Monte-Carlo Learning

Approximate state-value functions for the blackjack policy that sticks only on 20 or 21, computed by Monte Carlo policy evaluation.

Often; Monte Carlo methods are able to work with sample episodes alone which can be a significant advantage even when one has complete knowledge of the environment's dynamics.



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

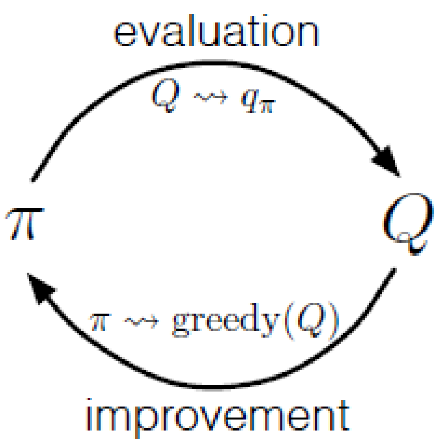
Monte-Carlo for Q(s,a)

- Same MC process but applied for each encountered (s,a).
- Problem: many Pairs may not be seen.
- Problem because we need to decide between all actions from a state.
- **Exploring starts**: requiring every (s,a) to be a start of an episode
- with positive probability

Policy evaluation by MC with ES

Follow GPI idea

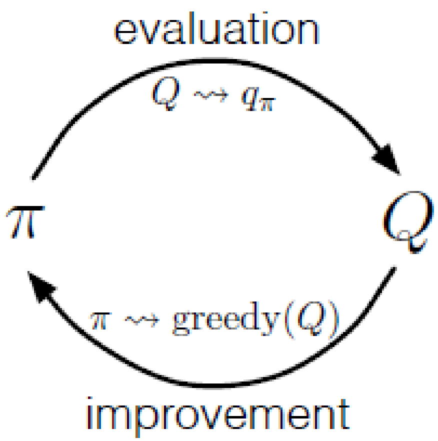
$$\pi_0 \xrightarrow{E} q_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} q_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots \xrightarrow{I} \pi_* \xrightarrow{E} q_*,$$



We can do full MC wit ES (Exploring starts) for each policy evaluation
Then do improvement. But this is not practical to have infinite
iterations

In black jack ES is reasonable. We can simulate a game from any initial set of cards

Monte Carlo Control



For Monte Carlo policy evaluation alternate between evaluation and improvement on an episode-by-episode basis. After each episode, the observed returns are used for policy evaluation, and then the policy is improved at all the states visited in the episode.

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

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

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

In Monte Carlo ES, all the returns for each state-action pair are accumulated and averaged, irrespective
of what policy was in force when they were observed. It is easy to see that Monte Carlo ES
cannot converge to any suboptimal policy. If it did, then the value function would eventually converge
to the value function for that policy, and that in turn would cause the policy to change. Stability is
achieved only when both the policy and the value function are optimal. Convergence to this optimal
fixed point seems inevitable as the changes to the action-value function decrease over time, but has not
yet been formally proved. In our opinion, this is one of the most fundamental open theoretical questions
in reinforcement learning (for a partial solution, see Tsitsiklis, 2002).

Epsilon-greedy and epsilon-soft policies

A policy is e-greedy relative to Q is in
(1-e)+1/number of actions
of the time. We choose a greedy action and otherwise unfirmly at random (of a total of e)

E-soft policy gives a positive probability to every action and does so unfirmly.

probability, $1 - \varepsilon + \frac{\varepsilon}{|\mathcal{A}(s)|}$, is given to the greedy action. The ε-greedy policies are examples of ε-*soft* policies, defined as policies for which $\pi(a|s) \geq \frac{\varepsilon}{|\mathcal{A}(s)|}$ for all states and actions, for some $\varepsilon > 0$. Among ε-soft policies, ε-greedy policies are in some sense those that are closest to greedy.

Monte-Carlo without exploring starts

On-policy vs off-policy methods:

- on-policy evaluates or improve the policy that is being used to make the decisions
- Off-policy: evaluates and improve policy that is different than the one generating the data.

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

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

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

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

Off-policy Prediction via Importance Sampling

In this section we begin the study of off-policy methods by considering the *prediction* problem, in which both target and behavior policies are fixed. That is, suppose we wish to estimate v_π or q_π, but all we have are episodes following another policy b, where $b \neq \pi$. In this case, π is the target policy, b is the behavior policy, and both policies are considered fixed and given.

For more on off-policy based on importance sampling read section 5.5

Incremental Mean

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

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

Incremental Monte-Carlo Updates

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

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

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

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

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

Temporal Difference

Sutton and Barto,
Chapters 6

TD learning is the central idea for RL.
It combines MC with DP David Silver

Temporal-Difference Learning, Chapter 6

- 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

The general idea

- TD learning is a combination of Monte Carlo ideas and dynamic
- programming (DP) ideas.
- Like Monte Carlo methods, TD methods can learn directly from raw experience without a model of the environment's dynamics.
- Like DP, TD methods update estimates based in part on other learned estimates, without waiting for a final outcome (they bootstrap).
- The relationship between TD, DP, and Monte Carlo methods is a recurring theme in the theory of reinforcement learning.
- The focus is on policy evaluation, or the prediction problem on one hand and the problem of estimating

  the value function on the other.

For the *control* problem (finding an optimal policy), DP, TD, and Monte Carlo methods all use some variation of generalized policy iteration (GPI). The differences in the methods are primarily differences in their approaches to the prediction problem.

MC and TD

- Goal: learn v_π online from experience under policy π
- **Incremental** every-visit Monte-Carlo
  - Update value $V(S_t)$ toward *actual* return G_t

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

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

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

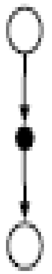
  - $R_{t+1} + \gamma V(S_{t+1})$ is called the *TD target*
  - $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the *TD error*

Tabular TD(0) for value prediction

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0$, for all $s \in \mathcal{S}^+$)
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
    $S \leftarrow S'$
  until S is terminal

v(St+1) which is an estimate is used instead of the return .


TD(0)

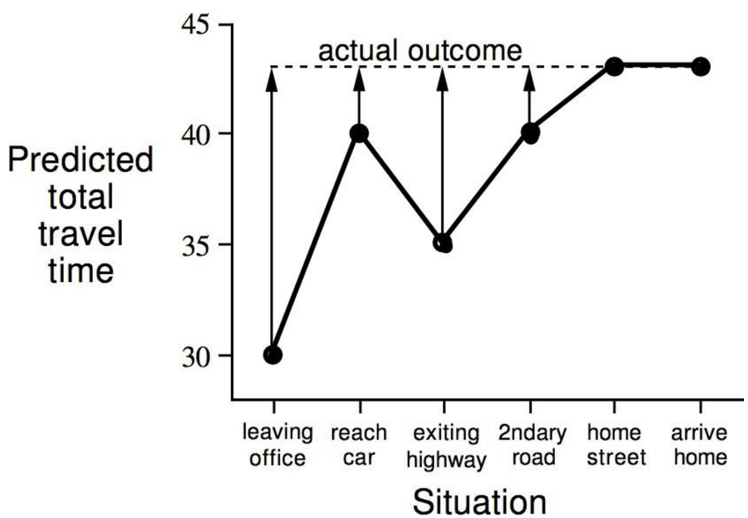
It is bootstrapping

Driving Home Example

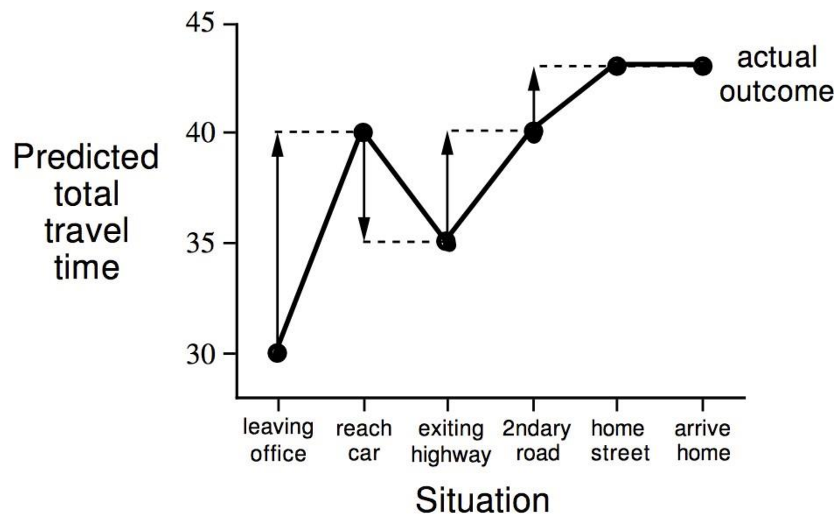
State	Elapsed Time (minutes)	Predicted Time to Go	Predicted Total Time
leaving office	0	30	30
reach car, raining	5	35	40
exit highway	20	15	35
behind truck	30	10	40
home street	40	3	43
arrive home	43	0	43

Driving Home Example: MC vs. TD

Changes recommended by Monte Carlo methods (=1)



Changes recommended by TD methods (!=1)



Advantages and Disadvantages of MC vs. TD

- 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

Advantages and Disadvantages of MC vs. TD (2)

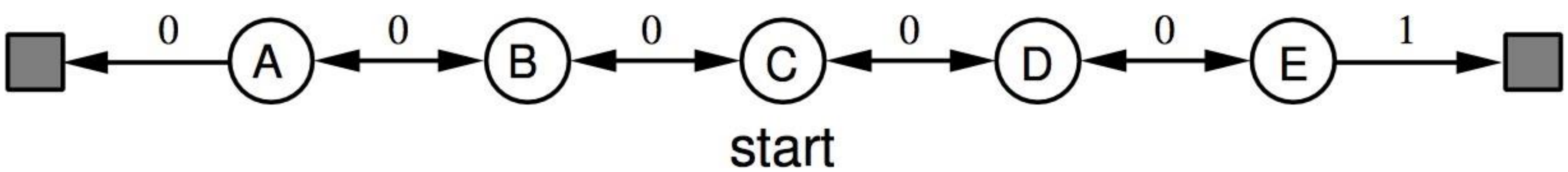
- MC has high variance, zero bias
  - Good convergence properties
  - (even with function approximation)
  - Not very sensitive to initial value
  - Very simple to understand and use
- TD has low variance, some bias
  - Usually more efficient than MC
  - TD(0) converges to $v_\pi(s)$
  - (but not always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

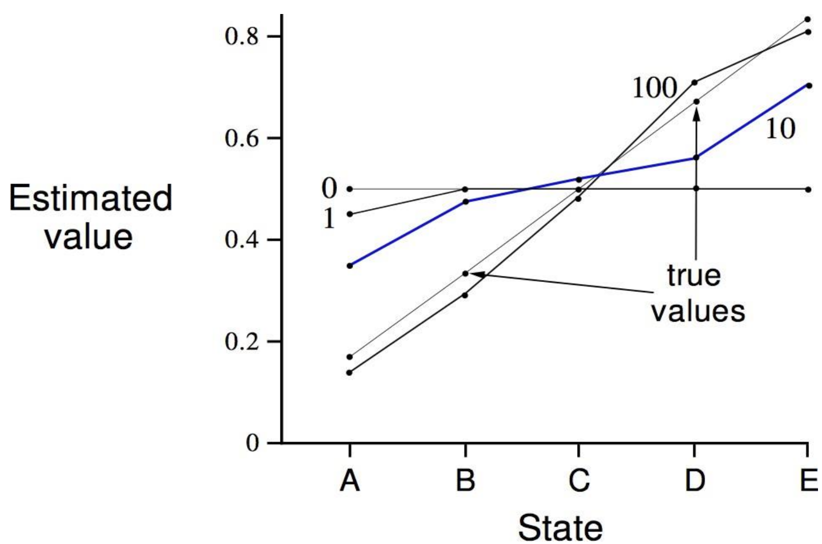
Does TD(0) converges to v? If so, How fast?

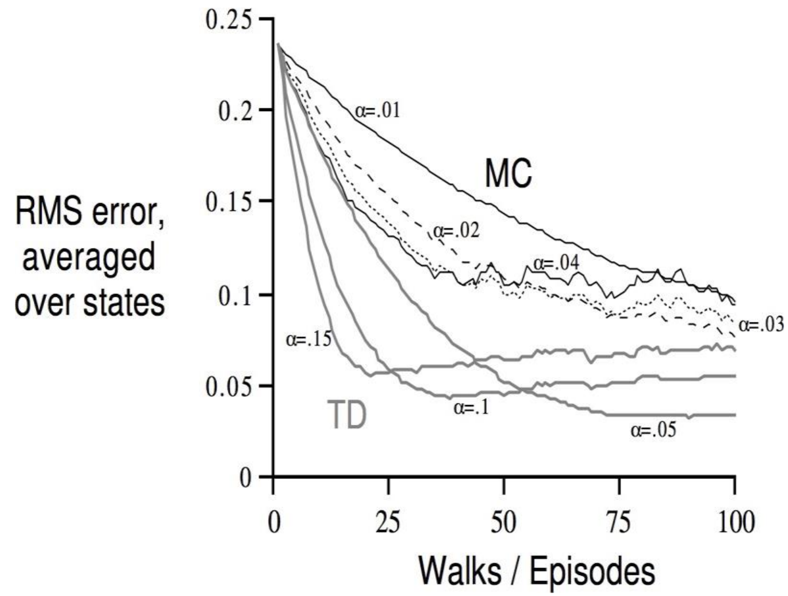
- Yes. For any fixed policy , TD(0) has been proved to converge to v, in the mean for a constant step-size parameter if it is sufficiently small, and with probability 1 if the step-size parameter decreases according to the usual stochastic approximation conditions (2.7).
- Most convergence proofs apply only to the table-based case of the algorithm presented above (6.2), but some also apply to the case of general linear function approximation.
- Which is faster convergence? MC or TD?
- At the current time this is an open question in the sense that no one has been able to prove mathematically that one method converges faster than the other. In fact, it is not even clear what is the most appropriate formal way to phrase this question! In practice,
- however, TD methods have usually been found to converge faster than constant- MC methods on stochastic tasks, as illustrated in the random walk example.

Random Walk Example

$$V(S_t) \leftarrow V(S_t) + \alpha \Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \Big] \tag{6.2}$$







Optimality of TD(0)

Batch MC and TD

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

$$s_1^1, a_1^1, r_2^1, \ldots, s_{T_1}^1$$

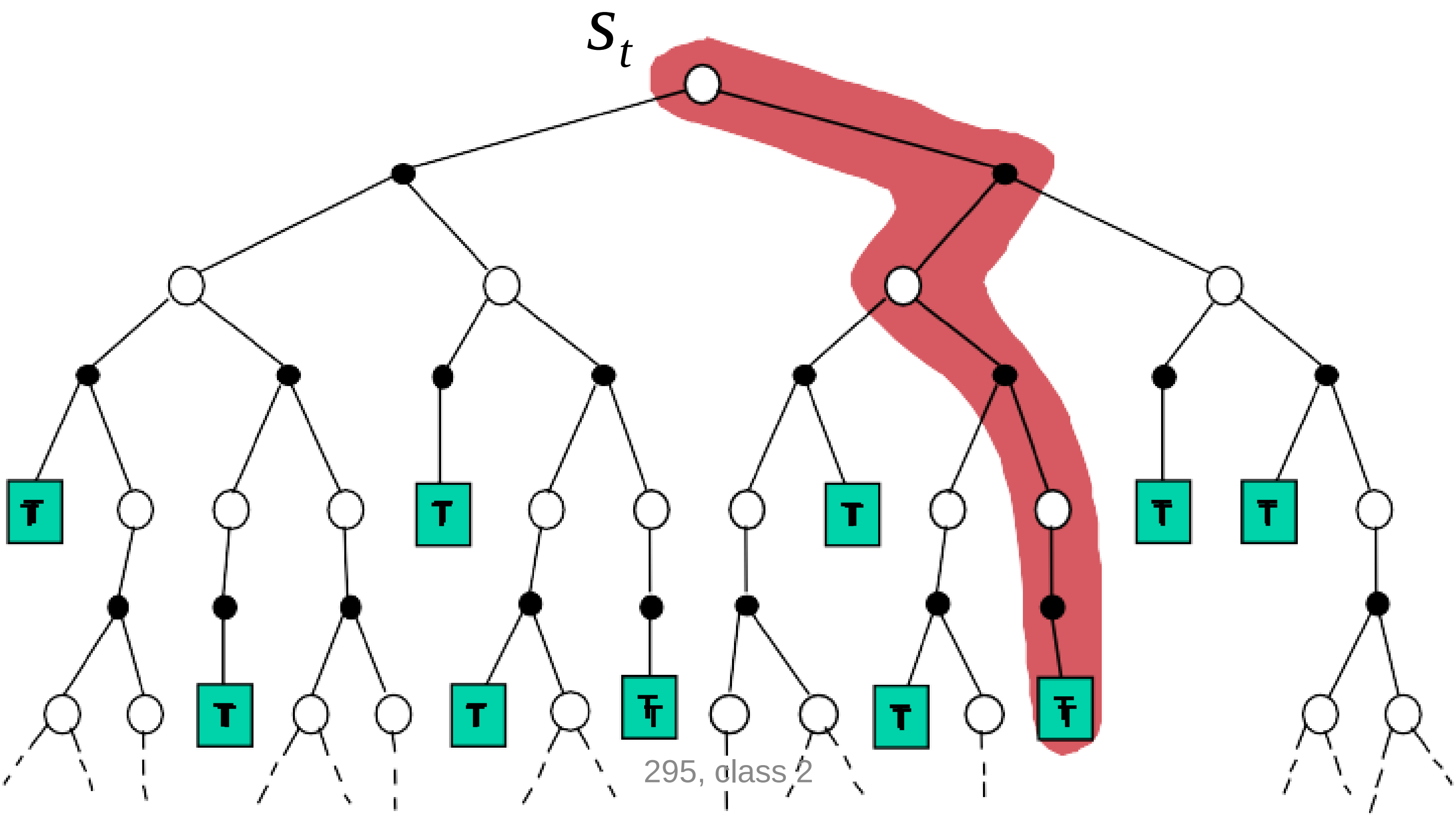
$$\vdots$$

$$s_1^K, a_1^K, r_2^K, \ldots, s_{T_K}^K$$

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

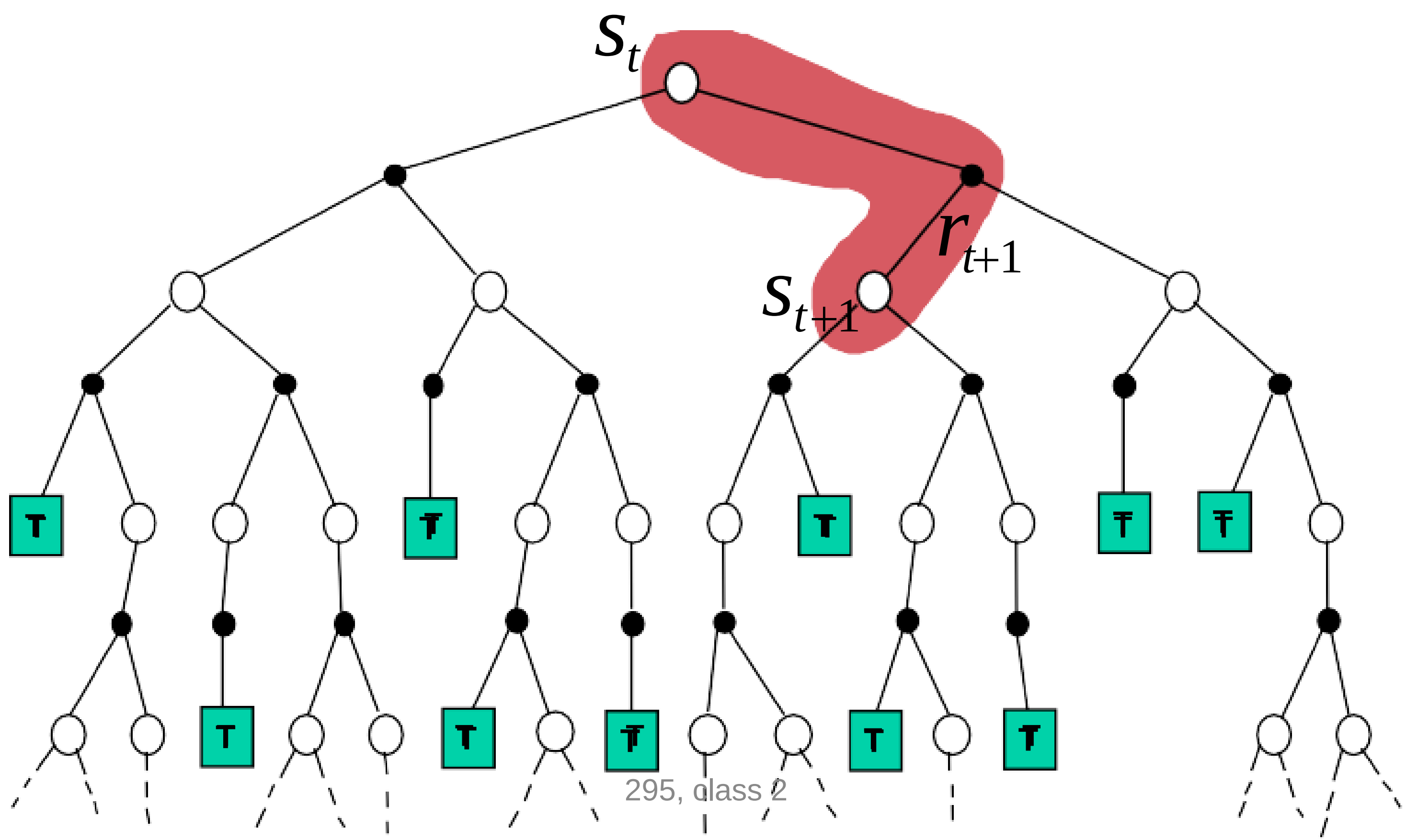
Monte-Carlo Backup

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



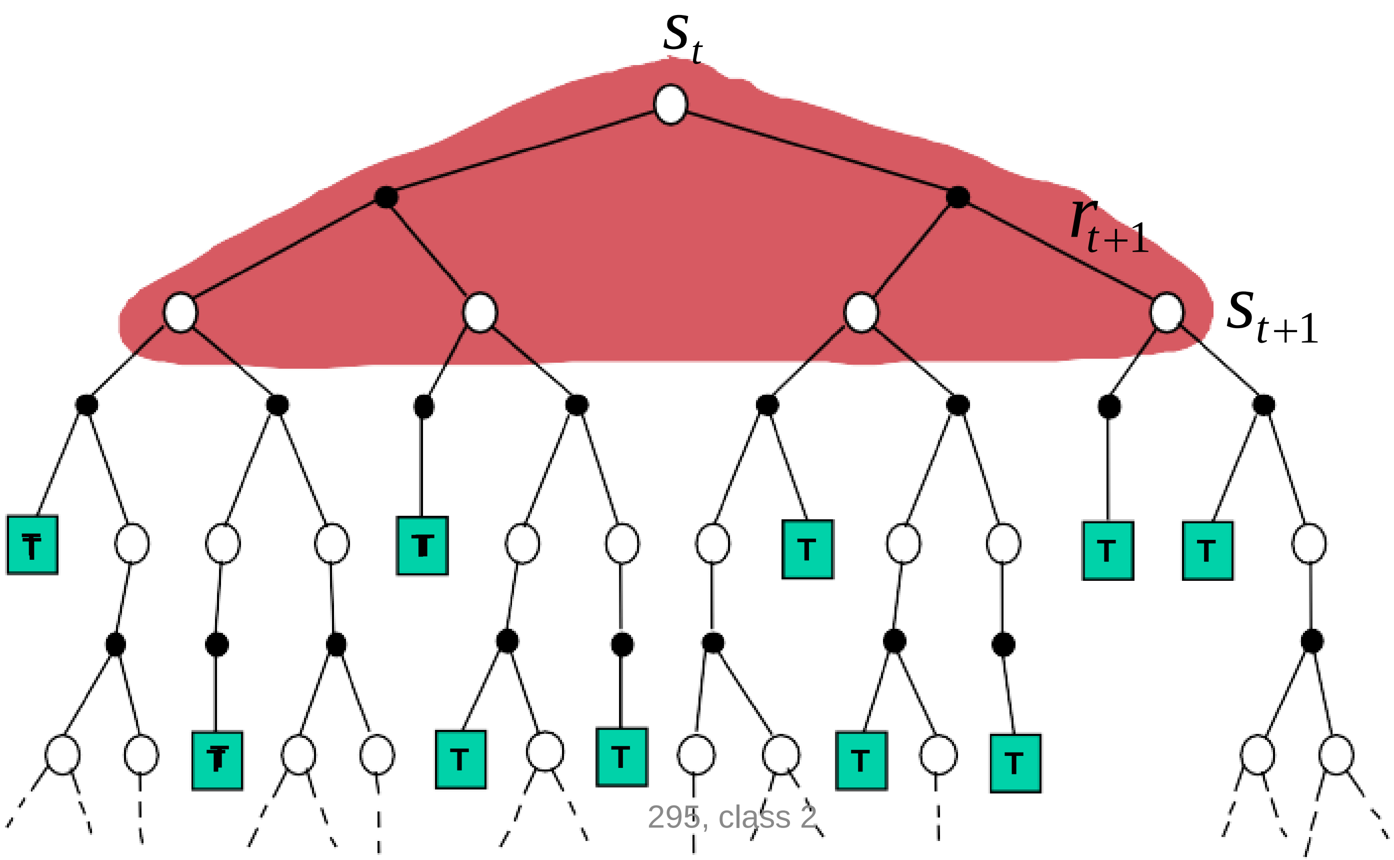
Temporal-Difference Backup

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



Dynamic Programming Backup

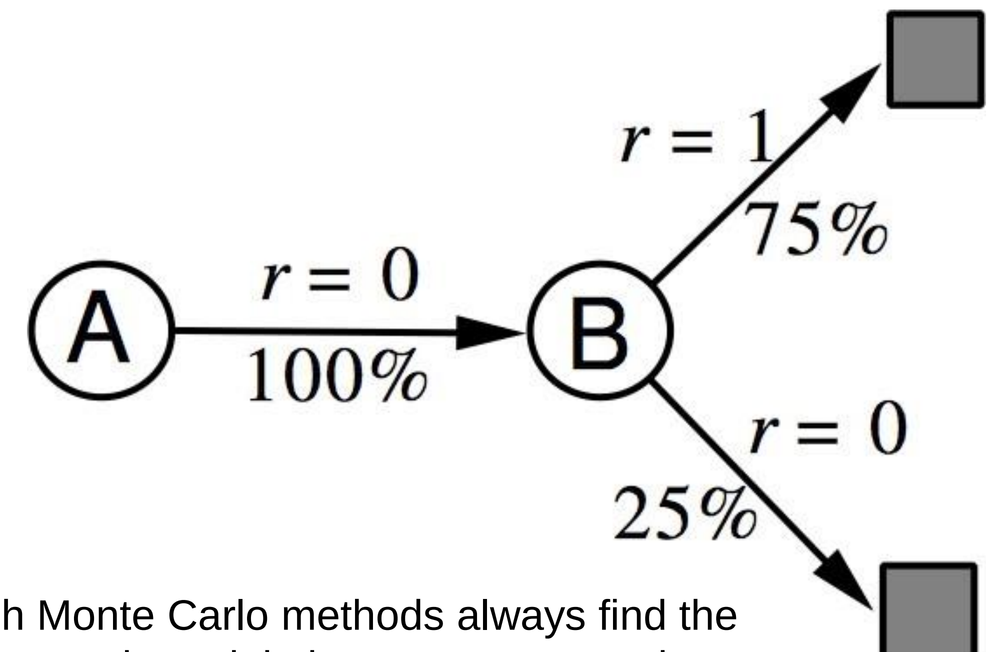
$$V(S_t) \leftarrow E_\pi[R_{t+1} + \gamma V(S_{t+1})]$$



AB Example

Two states *A*, *B*; no discounting; 8 episodes of experience

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



What is $V(A)$, $V(B)$?

Batch Monte Carlo methods always find the estimates that minimize means squared error on the training set, whereas batch TD(0) always finds the estimates that would be exactly correct for the maximum-likelihood model of the Markov process

Certainty Equivalence

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

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

  - In the AB example, $V(A) = 0$
- TD(0) converges to solution of max likelihood Markov model
  - Solution to the MDP $\langle \mathcal{S}, \mathcal{A}, \hat{\mathcal{P}}, \hat{\mathcal{R}}, \gamma \rangle$ that best fits the data

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

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

  - In the AB example, $V(A) = 0.75$

Sarsa Algorithm for On-Policy Control

Same as TD for value prediction just for (state,action) pairs

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

The convergence properties of the Sarsa algorithm depend on the nature of the policy's dependence on Q. For example, one could use ε-greedy or ε-soft policies. Sarsa converges with probability 1 to an optimal policy and action-value function as long as all state–action pairs are visited an infinite number of times and the policy converges in the limit to the greedy policy (which can be arranged, for example, with ε-greedy policies by setting $\varepsilon = 1/t$).

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

Convergence of Sarsa

Theorem

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

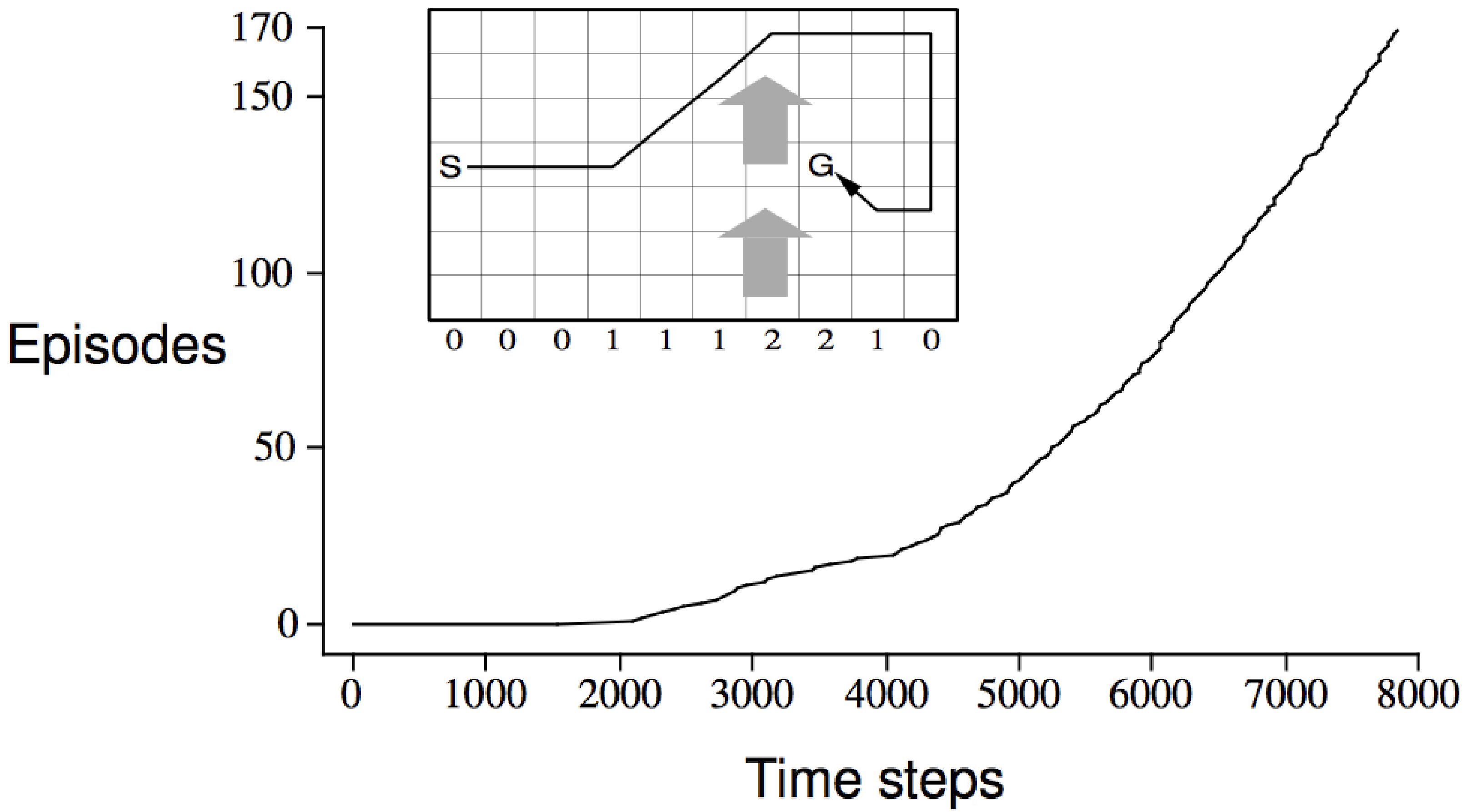
- *GLIE sequence of policies $\pi_t(a|s)$*
- *Robbins-Monro sequence of step-sizes α_t*

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

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

Sarsa on the Windy Gridworld

The results of applying “e-greedy Sarsa to this task, with " e= 0:1,
a= 0:5, and the initial values Q(s; a) = 0 for all s; a. The increasing slope of the graph s
that the goal is reached more and more quickly over time

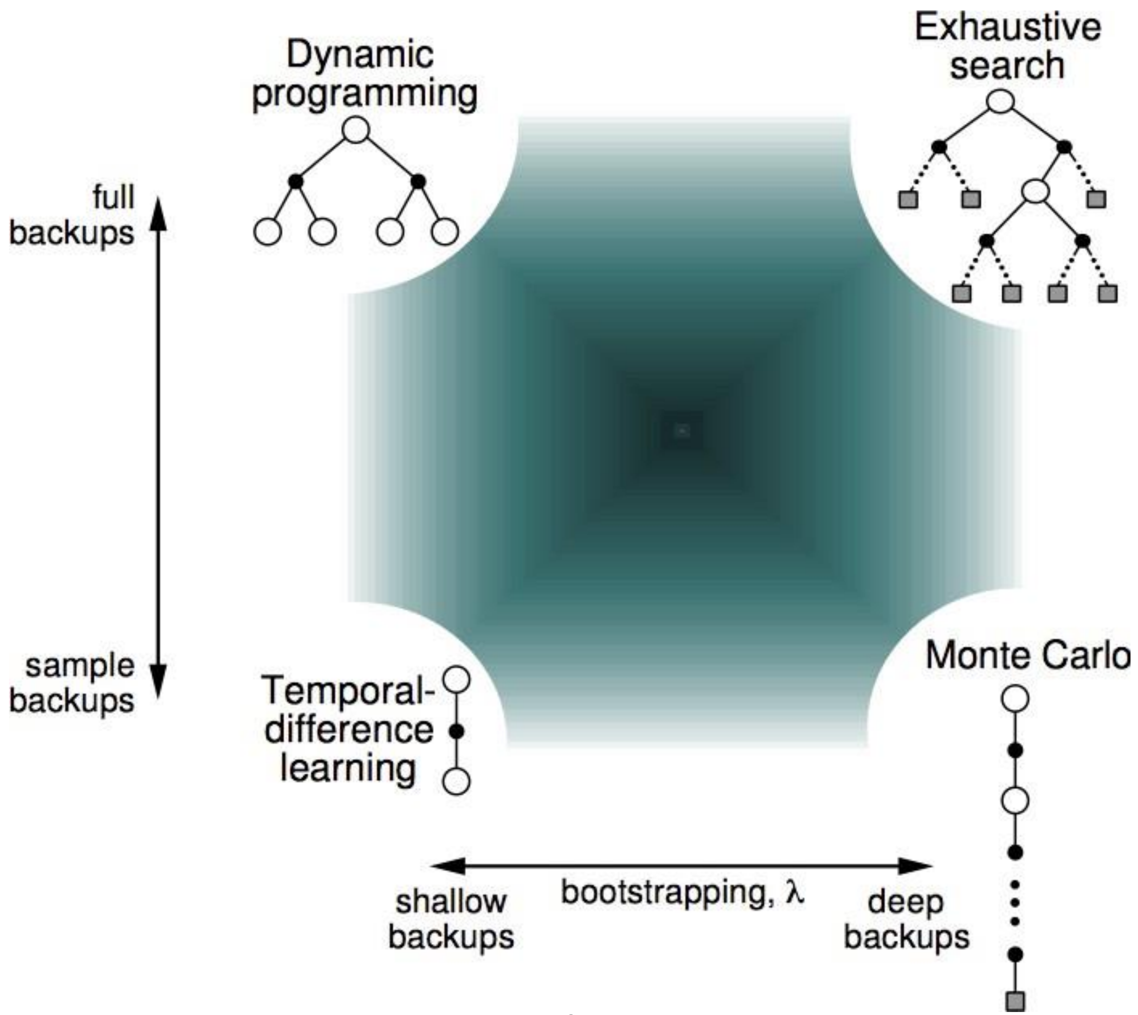


Bootstrapping and Sampling

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

Outline

Unified View of Reinforcement Learning



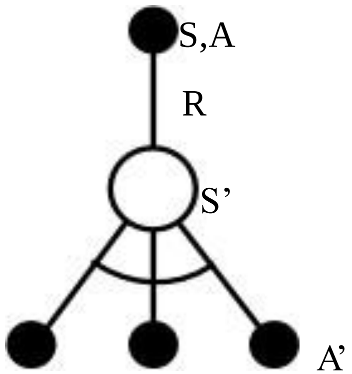
Q-learning: Off-policy Control

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

- Here the learned action-value function, Q, directly approximates q, the optimal action-value function, **independent of the policy being followed.**
- simplifies the analysis of the algorithm and enabled early convergence proofs.
- The policy impacts which state-action pairs are visited and updated.
- For correct convergence all pairs continue to be updated
- Under this assumption and a variant of the usual stochastic approximation conditions on the sequence of step-size parameters, Q has been shown to converge with probability 1 to q.

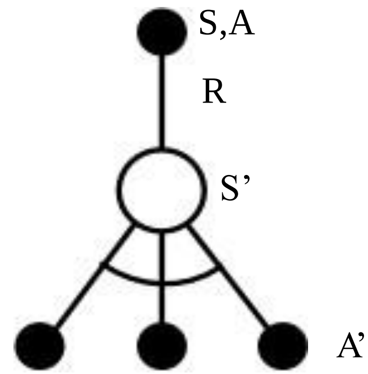
Q-learning for off-policy Control

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \Big]. \quad (6.8$$



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

Q-Learning Control Algorithm



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

Theorem

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

Example of SARSA (On policy control) vs Q-learning (Off-policy)

Q learning leans the values of
The optimal solution and optimal policy
But will fall of the cliff ocasioanlly due to
Exploration.
Its online performance is worse
Than SARSA .

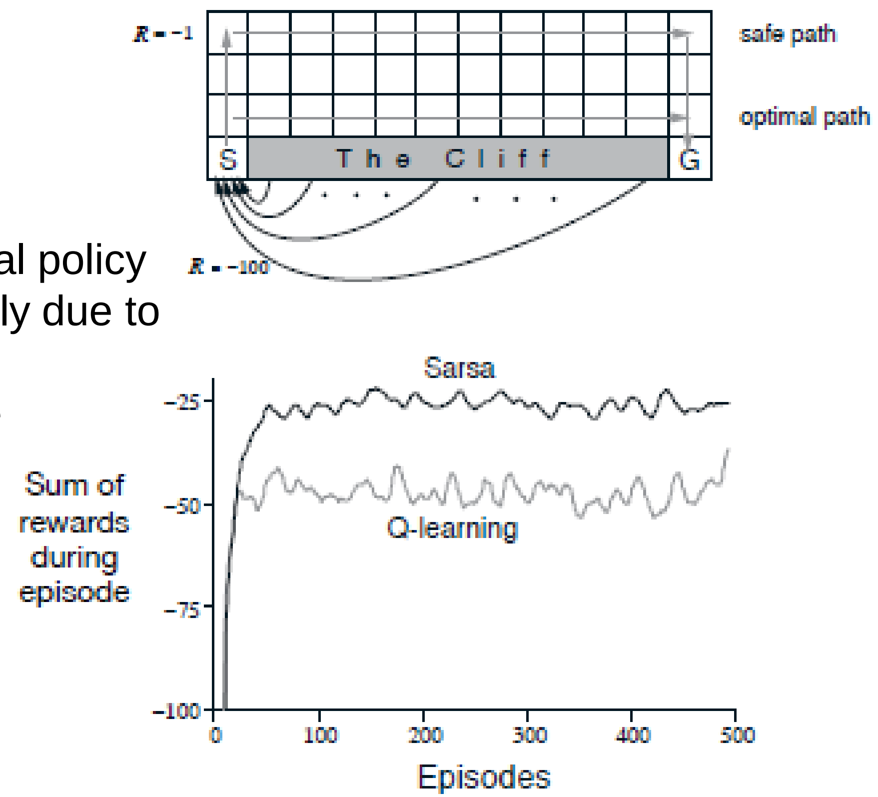


Figure 6.4: The cliff-walking task. The results are from a single run, but smoothed by averaging the reward sums from 10 successive episodes. ■

Summary chapters 5,6

In this chapter we introduced a new kind of learning method, temporal-difference (TD) learning, and showed how it can be applied to the reinforcement learning problem. As usual, we divided the overall problem into a prediction problem and a control problem. TD methods are alternatives to Monte Carlo methods for solving the prediction problem. In both cases, the extension to the control problem is via the idea of generalized policy iteration (GPI) that we abstracted from dynamic programming. This is the idea that approximate policy and value functions should interact in such a way that they both move toward their optimal values.

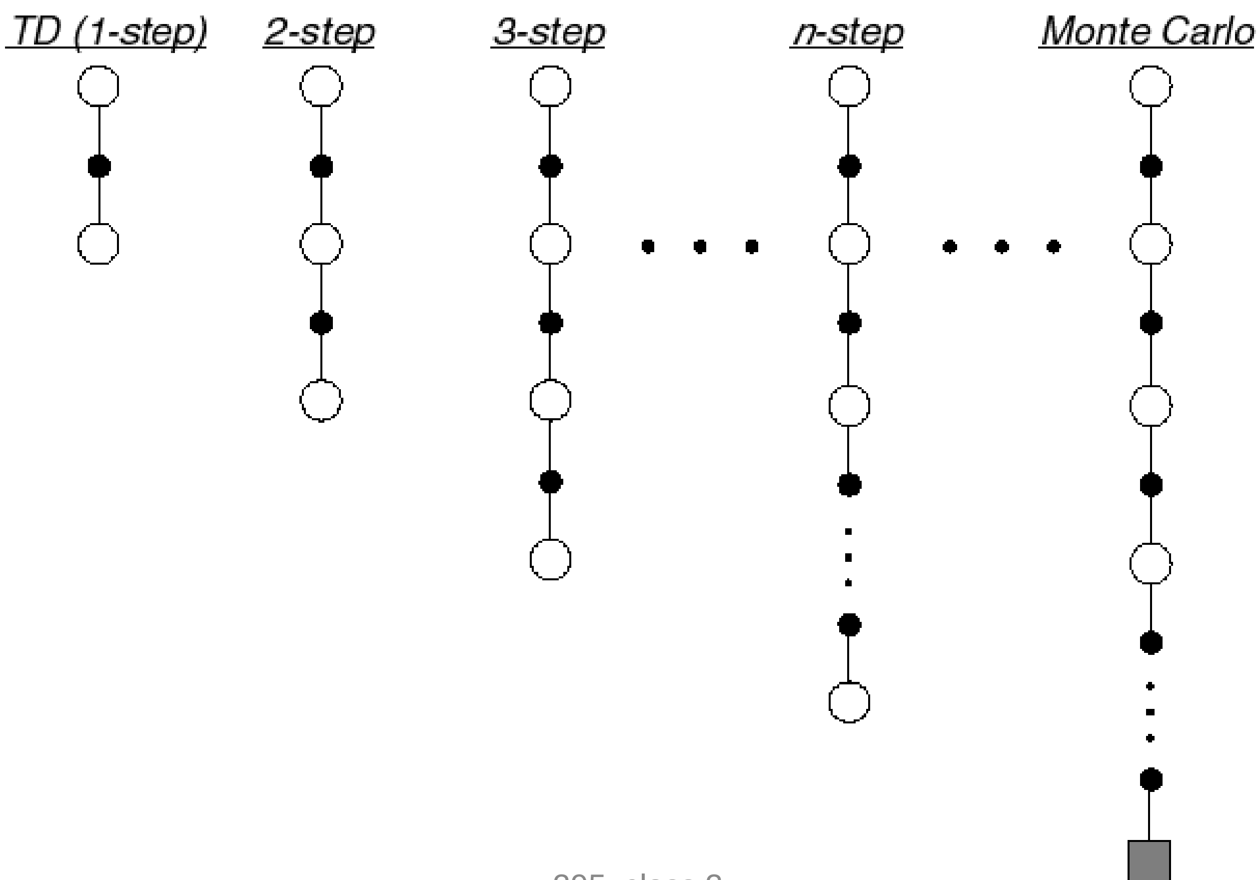
One of the two processes making up GPI drives the value function to accurately predict returns for the current policy; this is the prediction problem. The other process drives the policy to improve locally (e.g., to be ε-greedy) with respect to the current value function. When the first process is based on experience, a complication arises concerning maintaining sufficient exploration. We can classify TD control methods according to whether they deal with this complication by using an on-policy or off-policy approach. Sarsa is an on-policy method, and Q-learning is an off-policy method. Expected Sarsa is also an off-policy method as we present it here. There is a third way in which TD methods can be extended to control which we did not include in this chapter, called actor–critic methods. These methods are covered in full in Chapter 13.

Chapter 7: n-step Bootstrapping

n-Step Prediction

Vary the size of a look-ahead before updating.
TD(0) is one-step look-ahead. MC is full-episode look-ahead.

- Let TD target look *n* steps into the future



n-Step Return

All n-step returns can be considered approximations to the
full return, truncated after n steps and then corrected for the remaining missing terms by
V_(t+n-1)(S_(t+n)).

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

$$
\begin{array}{lll}
n = 1 & (TD) & G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1}) \\
n = 2 & & G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2}) \\
\vdots & & \vdots \\
n = \infty & (MC) & G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T
\end{array}
$$

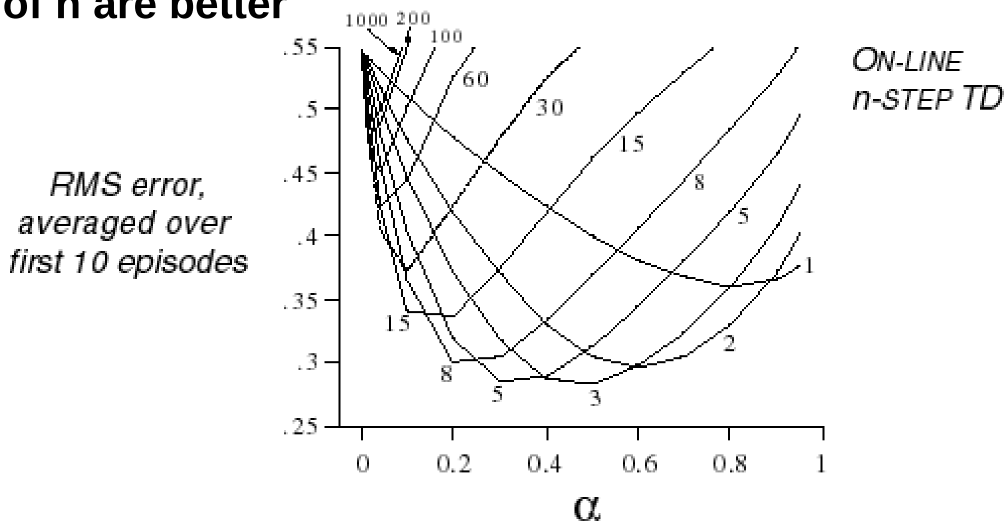
- Define the n-step return

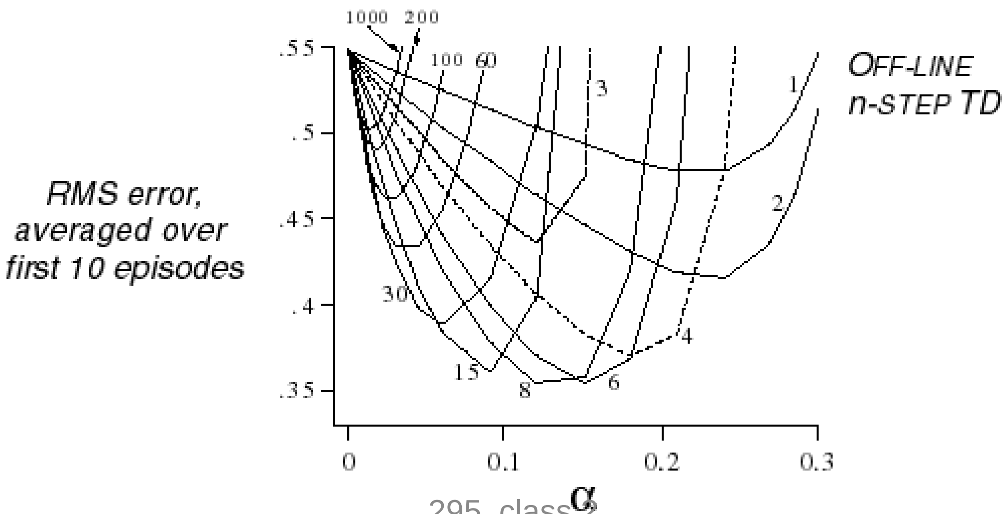
$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

- n-step temporal-difference learning

Large Random Walk Example

Intermediate values of n are better





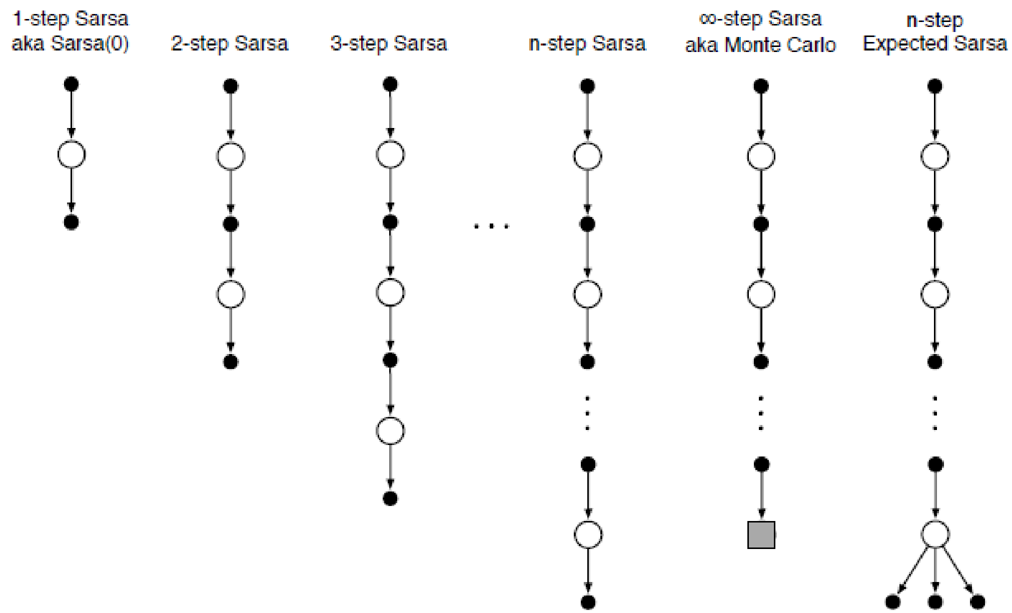
n-step TD for value of a policy

n-step TD for estimating $V \approx v_\pi$

Initialize $V(s)$ arbitrarily, $s \in \mathcal{S}$
Parameters: step size $\alpha \in (0, 1]$, a positive integer n
All store and access operations (for S_t and R_t) can take their index mod n

Repeat (for each episode):
  Initialize and store $S_0 \neq$ terminal
  $T \leftarrow \infty$
  For $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take an action according to $\pi(\cdot|S_t)$
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, then $T \leftarrow t + 1$
  | $\tau \leftarrow t - n + 1$ (τ is the time whose state's estimate is being updated)
  | If $\tau \geq 0$:
  | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
  | If $\tau + n < T$, then: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $V(S_\tau) \leftarrow V(S_\tau) + \alpha \left[G - V(S_\tau)\right]$
  Until $\tau = T - 1$

n-step Sarsa (on-line control)



The main idea is to simply switch states for actions (state–action pairs) and then use an ε-greedy policy. The backup diagrams for n-step Sarsa (shown in Figure 7.3), like those of n-step TD (Figure 7.1), are strings of alternating states and actions, except that the Sarsa ones all start and end with an action rather a state. We redefine n-step returns (update targets) in terms of estimated action values:

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}), \quad n \geq 1, 0 \leq t < T - n, \quad (7.4)$$

with $G_{t:t+n} \doteq G_t$ if $t + n \geq T$. The natural algorithm is then

[redacted] (7.5)

n-step Sarsa estimating Q

n-step Sarsa for estimating $Q \approx q_*$, or $Q \approx q_\pi$ for a given π

Initialize $Q(s, a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be ε-greedy with respect to Q, or to a fixed given policy
Parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$, a positive integer n
All store and access operations (for S_t, A_t, and R_t) can take their index mod n

Repeat (for each episode):
  Initialize and store $S_0 \neq$ terminal
  Select and store an action $A_0 \sim \pi(\cdot|S_0)$
  $T \leftarrow \infty$
  For $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take action A_t
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, then:
  | $T \leftarrow t + 1$
  | else:
  | Select and store an action $A_{t+1} \sim \pi(\cdot|S_{t+1})$
  | $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)
  | If $\tau \geq 0$:
  | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n, T)} \gamma^{i-\tau-1} R_i$
  | If $\tau + n < T$, then $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha \left[G - Q(S_\tau, A_\tau)\right]$
  | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is ε-greedy wrt Q
  Until $\tau = T - 1$

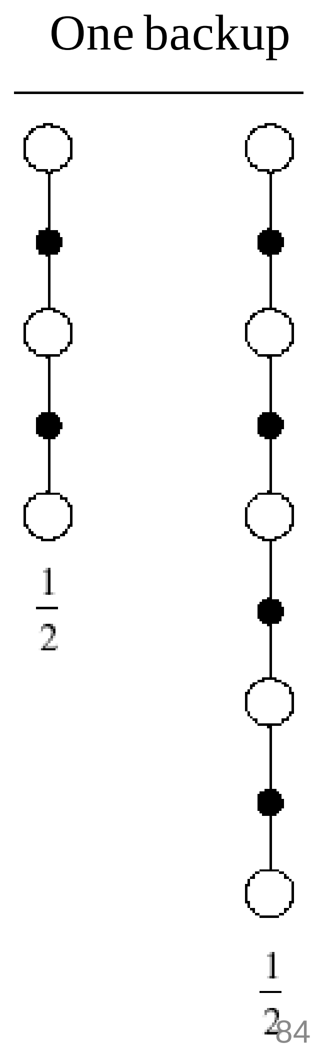
Chapter 12:
Eligibility Traces

Averaging n-Step Returns

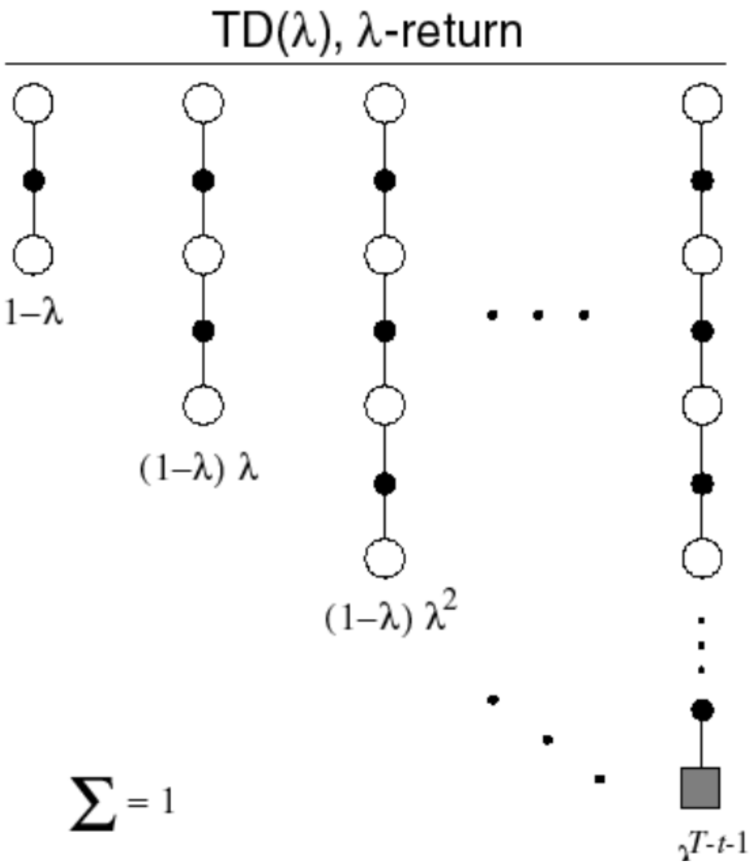
- We can average n-step returns over different n
- e.g. average the 2-step and 4-step returns

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

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



λ-return



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

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

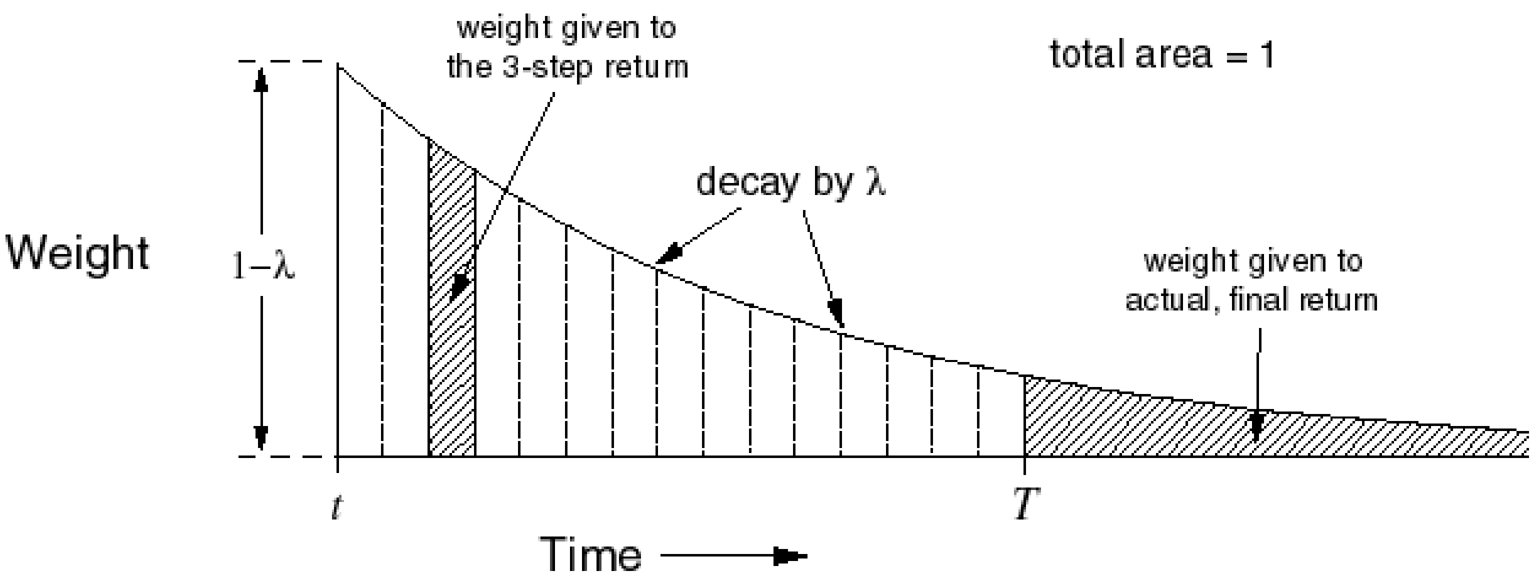
- Forward-view TD(λ)

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

The TD(λ) algorithm can be understood as one particular way of averaging n-step updates. This average contains all the n-step updates, each weighted proportional to λ^{n-1}, where $\lambda \in [0,1]$, and is normalized by a factor of $1-\lambda$ to ensure that the weights sum to 1 (see Figure 12.1). The resulting update is toward a return, called the λ-*return*, defined in its state-based form by

$$G_t^\lambda \doteq (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_{t:t+n}. \qquad (12.2)$$

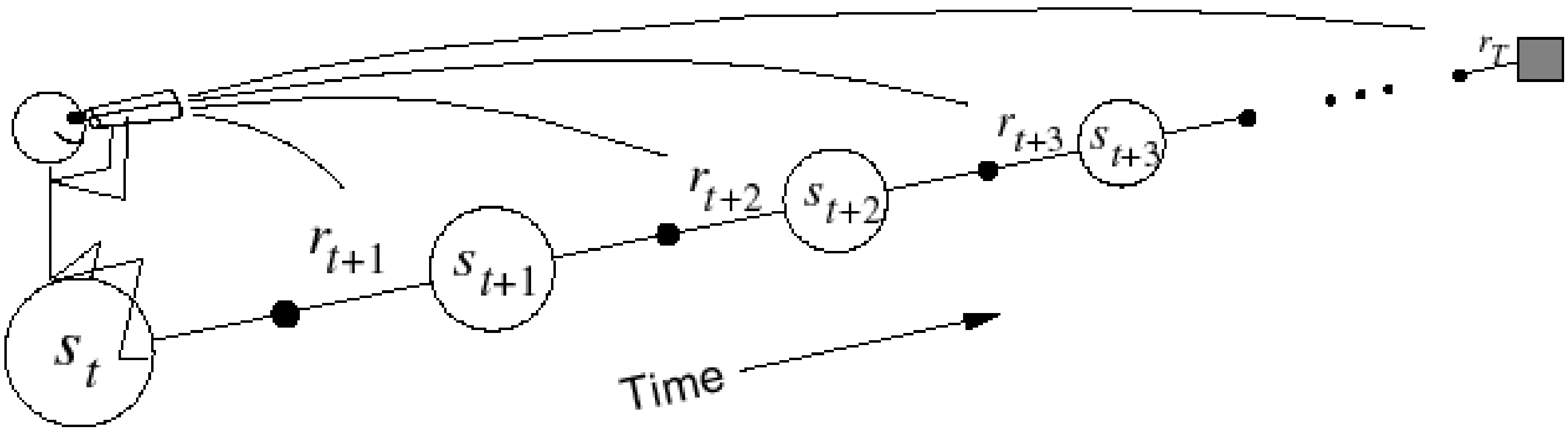
TD(λ) Weighting Function



$$G_t^{\lambda} \doteq (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n}.$$

Forward-view TD(λ)

Forward view of TD(λ)



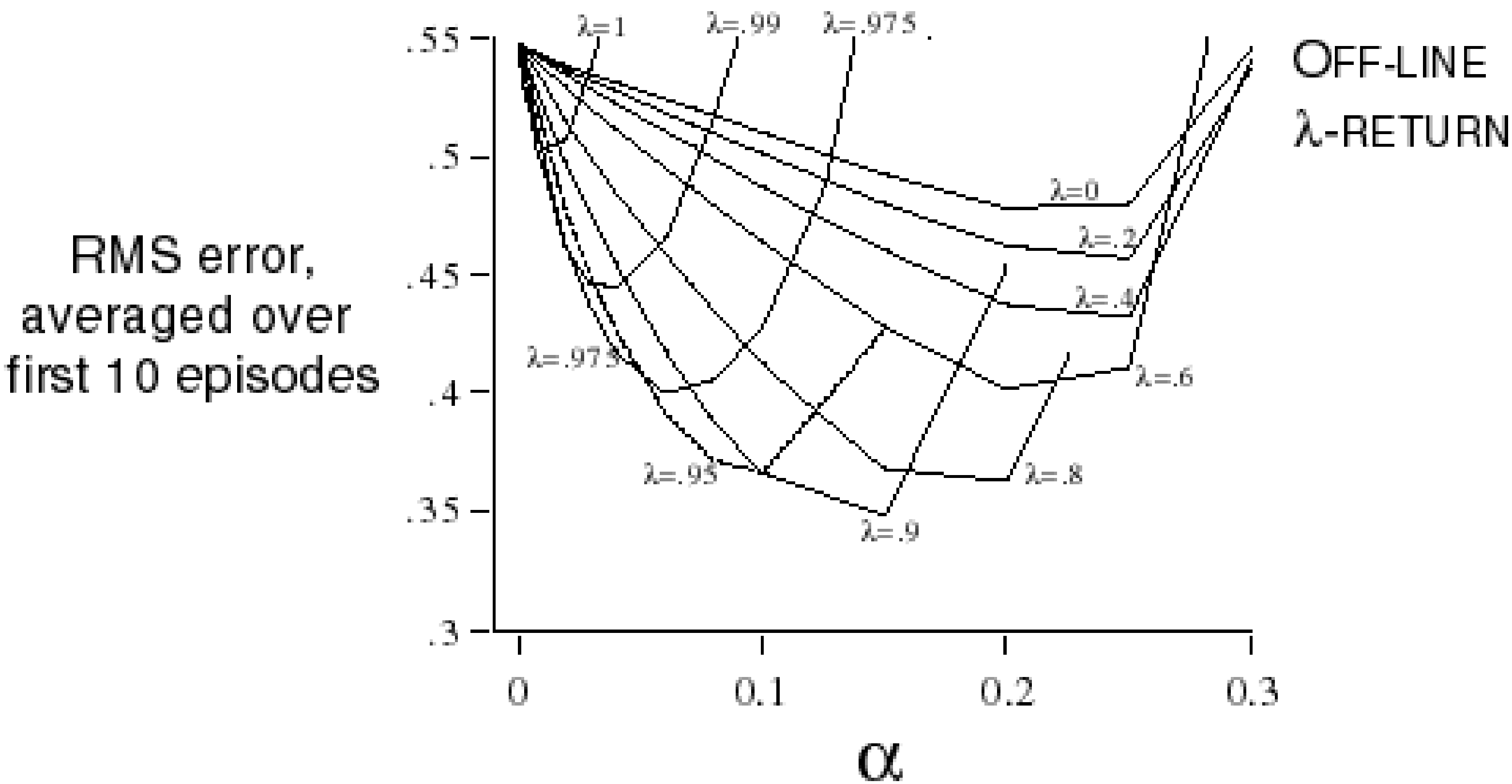
Update value function towards the λ-return

Forward-view looks into the future to compute G^λ_t

Like MC, can only be computed from complete episodes

The approach that we have been taking so far is what we call the theoretical, or *forward*, view of a learning algorithm. For each state visited, we look forward in time to all the future rewards and decide how best to combine them. We might imagine ourselves riding the stream of states, looking forward from each state to determine its update, as suggested by Figure 12.4. After looking forward from and updating one state, we move on to the next and never have to work with the preceding state again.

Forward-View TD(λ) on Large Random Walk

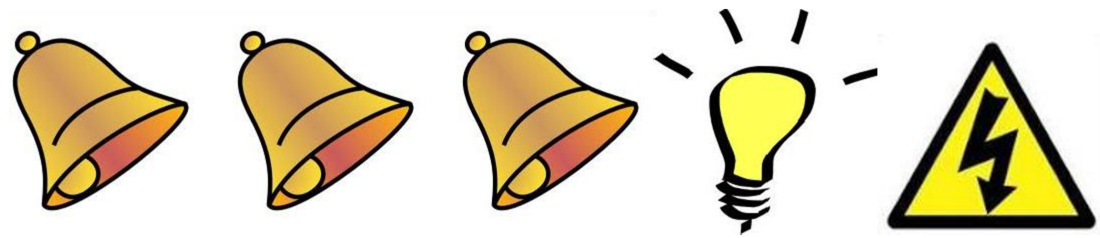


Backward View TD(λ)

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

Forward view is analogous to forward checking
Backword view is analogous to backup methods, that are backward looking
In heuristic search and in csp

Eligibility Traces



- Credit assignment problem: did bell or light cause shock?
- Frequency heuristic: assign credit to most frequent states
- Recency heuristic: assign credit to most recent states
- *Eligibility traces* combine both heuristics

$$E_0(s) = 0$$

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

Eligibility Traces

In the backword view of TD(lambda), there is an additional memory variable associated with each state called "eligibility trace" fo state s at time t.
On each step, the eligibility trace of all states decay by $\gamma\lambda$*,*
and the eligibility state of one state is incremented by 1:

$$E_0(s) = 0$$

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

At any time, eligibility traces record which states have recently been visited where Recency is defined in terms of $\gamma\lambda$

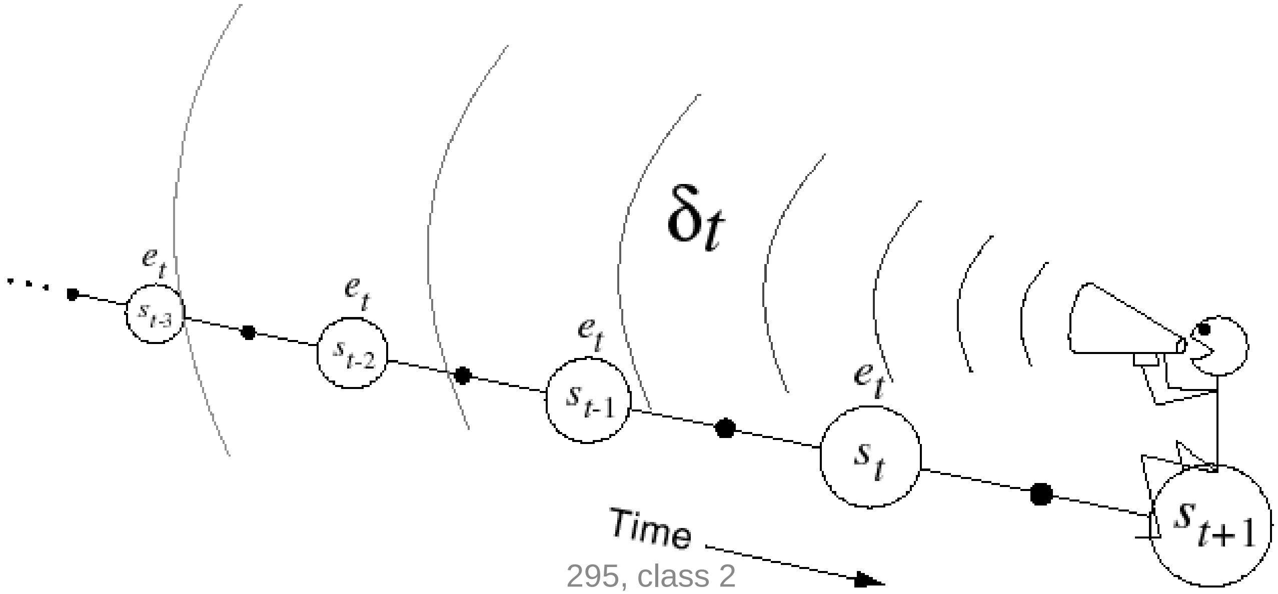
Backward View TD(λ)

Backward View of 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 $E_t(s)$

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

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



TD(λ) and TD(0)

Relationship Between Forward and Backward TD

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

$$E_t(s) = \mathbf{1}(S_t = s)$$
$$V(s) \leftarrow V(s) + \alpha\delta_t E_t(s)$$

- This is exactly equivalent to TD(0) update

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

On-line Tabular TD(lambda)

Initialize $V(s)$ arbitrarily and $e(s) = 0$, for all $s \in \mathcal{S}$
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    $a \leftarrow$ action given by π for s
    Take action a, observe reward, r, and next stat S’
    $\delta \leftarrow r + \gamma V(s') - V(s)$
    $e(s) \leftarrow e(s) + 1$
    For all s:
      $V(s) \leftarrow V(s) + \alpha \delta e(s)$
      $e(s) \leftarrow \gamma \lambda e(s)$
    $s \leftarrow s'$
  until s is terminal

Figure 7.7 On-line tabular TD(λ).

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)

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

Forwards and Backwards TD(λ)

- Consider an episode where s is visited once at time-step k
- TD(λ) eligibility trace discounts time since visit,

$$\begin{aligned} E_t(s) &= \gamma\lambda E_{t-1}(s) + \mathbf{1}(S_t = s) \\ &= \begin{cases} 0 & \text{if } t < k \\ (\gamma\lambda)^{t-k} & \text{if } t \geq k \end{cases} \end{aligned}$$

- Backward TD(λ) updates accumulate error *online*

$$\sum_{t=1}^{T} \alpha\delta_t E_t(s) = \alpha \sum_{t=k}^{T} (\gamma\lambda)^{t-k}\delta_t = \alpha\left(G_k^\lambda - V(S_k)\right)$$

- By end of episode it accumulates total error for λ-return
- For multiple visits to s, $E_t(s)$ accumulates many errors

Offline Equivalence of Forward and Backward TD

Offline updates

- Updates are accumulated within episode
- but applied in batch at the end of episode

End of class 2