Reinforcement Learning – COMS4061A/7071A

Model-Free Learning

Prof. Benjamin Rosman

Benjamin.Rosman1@wits.ac.za / benjros@gmail.com

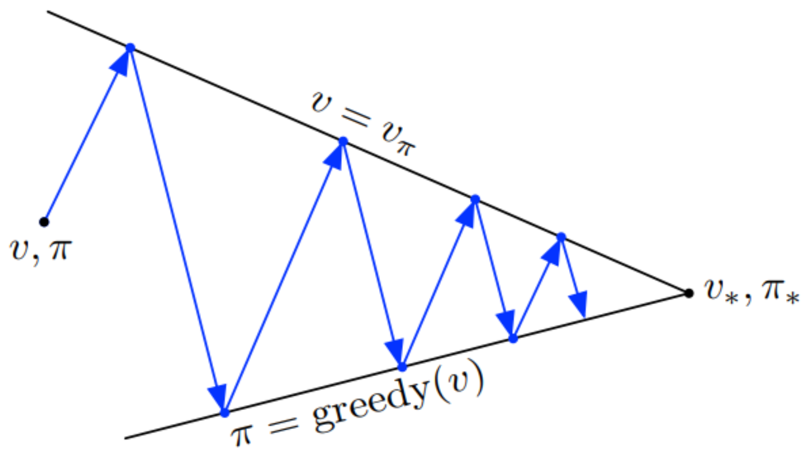
Based heavily on slides by Steve James

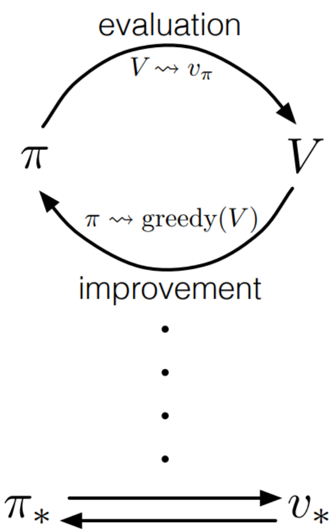
Sutton and Barto [2018], Chapters 5, 6, 7, 12, 15

Last time

- MDP: $\langle S, A, P, R, \gamma \rangle$
  - Known dynamics and reward function: $\langle S, A, P, R, \gamma \rangle$

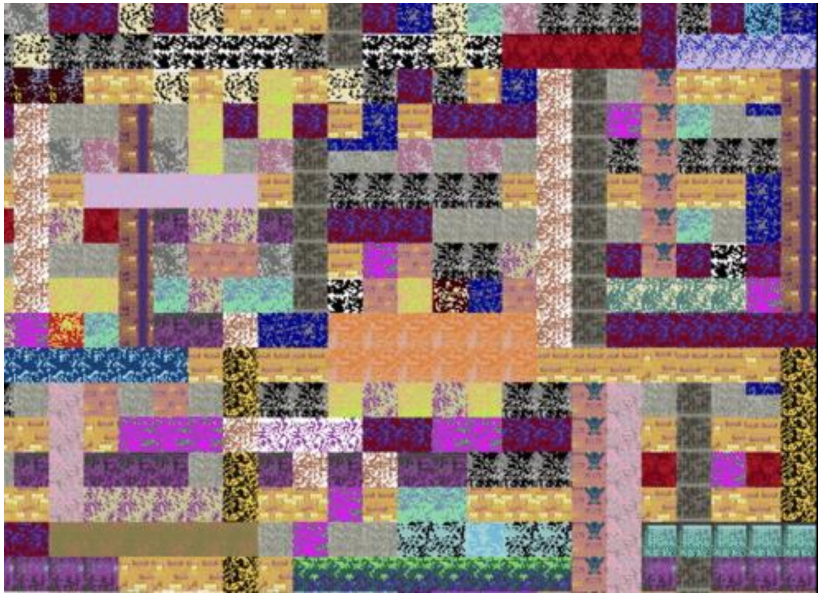
$$v_{k+1}(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_k(s')]$$





This week

- The full RL problem
  - We know nothing!



Challenges?

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   $\quad \Delta \leftarrow 0$
   $\quad$ Loop for each $s \in \mathcal{S}$:
   $\quad\quad v \leftarrow V(s)$
   $\quad\quad V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s))\big[r + \gamma V(s')\big]$
   $\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad$ *old-action* $\leftarrow \pi(s)$
   $\quad \pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
   $\quad$ If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

We can't just query for every possible state!

One-step look-ahead needed!

MONTE CARLO METHODS

Prediction and Control

Dr Strange uses Monte Carlo

14 000 605 “episodes”; 1 win
If $+1$ for winning and 0 otherwise

$$v_\pi(s) = \frac{1}{14000605}$$

Blackjack Example

- Previously, to compute value function, we needed the dynamics
- But computing $p(s', r \mid s, a)$ is non-trivial
  - e.g. I have 13, what's the probability I get to 15 given the cards on the table?
- But MC is easy!
  - Just play out many games (according to policy) from that position and average!

Monte Carlo prediction

- Compute $v_\pi(s)$

- For episodic tasks, the return is:

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

- The value function is the expected return

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

- Start at s, execute episodes according to π and take average
  - Empirical mean

Monte Carlo prediction

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

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

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

Loop forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

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

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

Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:

Append G to $Returns(S_t)$

$V(S_t) \leftarrow \text{average}(Returns(S_t))$

For "every-visit", delete this check

Note: no dependence on |S|

Incremental mean

- 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, we may want to forget old episodes:
$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$

Monte Carlo control

- We can evaluate π
- So we just need policy improvement
  - Then we can do policy iteration!
- But how do we improve the policy with $v_\pi(s)$?
- We instead estimate $q_\pi(s, a)$ and take argmax
- Must ensure all state-action pairs are visited an infinite number of times (exploring starts) for convergence

Monte Carlo Exploring Starts

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 $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Explore all states infinitely by random starts

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 $Returns(S_t, A_t)$

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

$\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$

Exploration!

Different from argmax!

- ε-soft policies: any policy where $\pi(a \mid s) > 0$ for all states, actions
- With probability ε, pick suboptimal action at random; otherwise, greedy action

$$\pi(a \mid s) = \begin{cases} 1 - \varepsilon + \dfrac{\varepsilon}{|A|}, \text{if } a = \text{argmaxQ}(s,\cdot) \\ \dfrac{\varepsilon}{|A|}, \text{otherwise} \end{cases}$$

Shortcomings of Monte Carlo

- Need episodes!
  - i.e. works in the episodic case only
  - Must have policies that can actually terminate the episode
  - Must wait until the end of the episode until we can update

- High variance
  - The return depends on many random actions, transitions and rewards

If one had to identify one idea as central and novel to reinforcement learning, it would undoubtedly be temporal-difference (TD) learning. TD learning is a combination of Monte Carlo ideas and dynamic programming (DP) ideas.

– Sutton and Barto [2018]

TEMPORAL DIFFERENCE

Prediction

Temporal difference learning

- TD learns from experience
- Model-free
- Learns from incomplete episodes
  - Can learn without the final outcome
  - And also in the continuing case
- Learns by bootstrapping
  - Similar to DP

TD Update

- MC update:

Update toward full episode return

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

- TD(0) update:

Update toward <u>estimated</u> return

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

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

Policy evaluation:

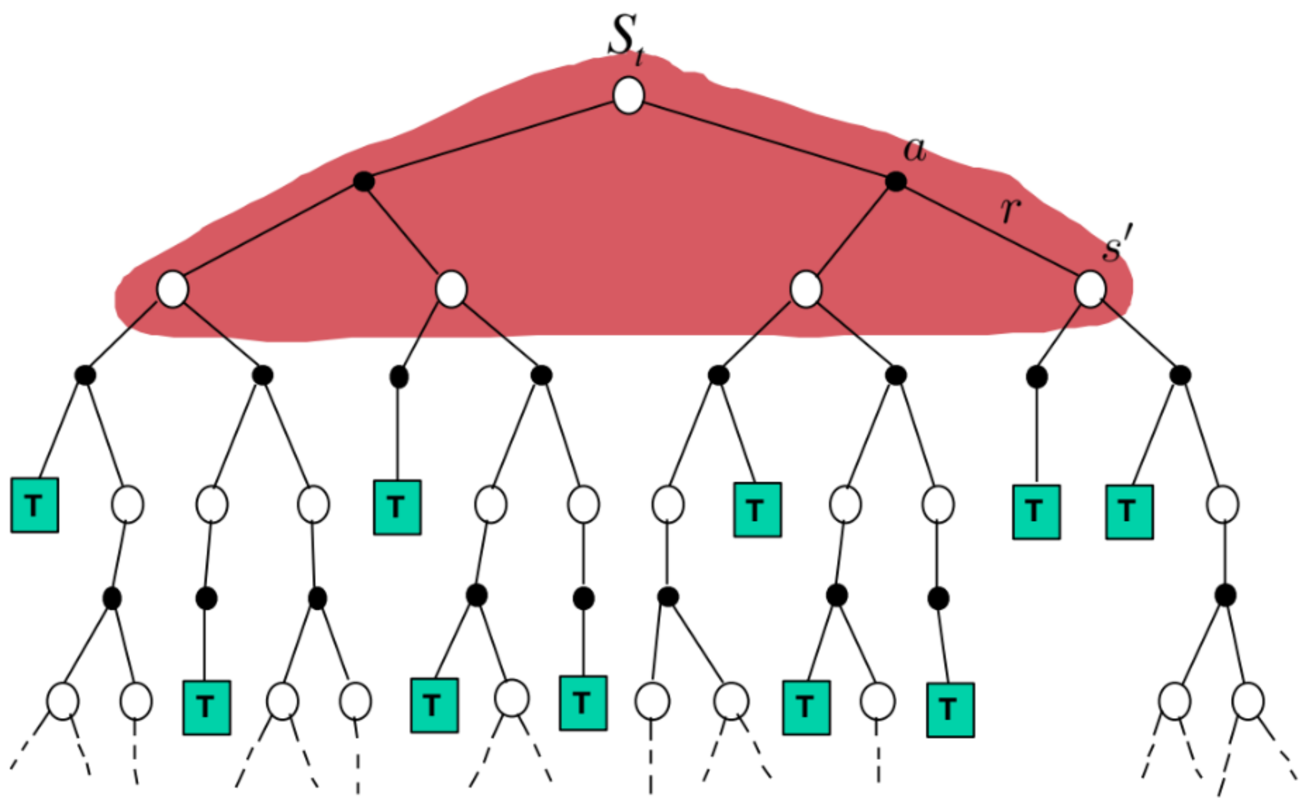
Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\textit{terminal}) = 0$

Loop for each episode:
  Initialize S
  Loop 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

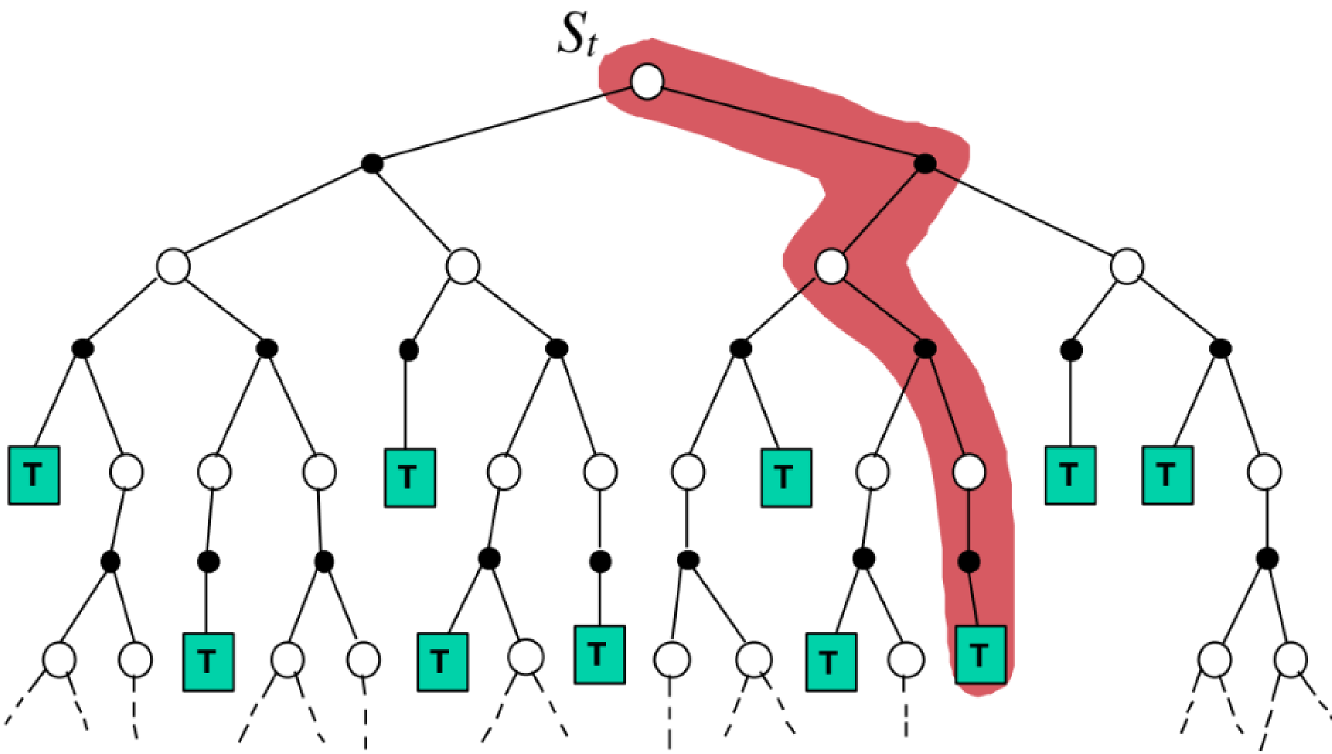
Dynamic Programming

$$V(S_t) \leftarrow \mathbb{E}_\pi[R_{t+1} + \gamma V(S_{t+1}) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid S_t, a)[r + \gamma V(s')]$$



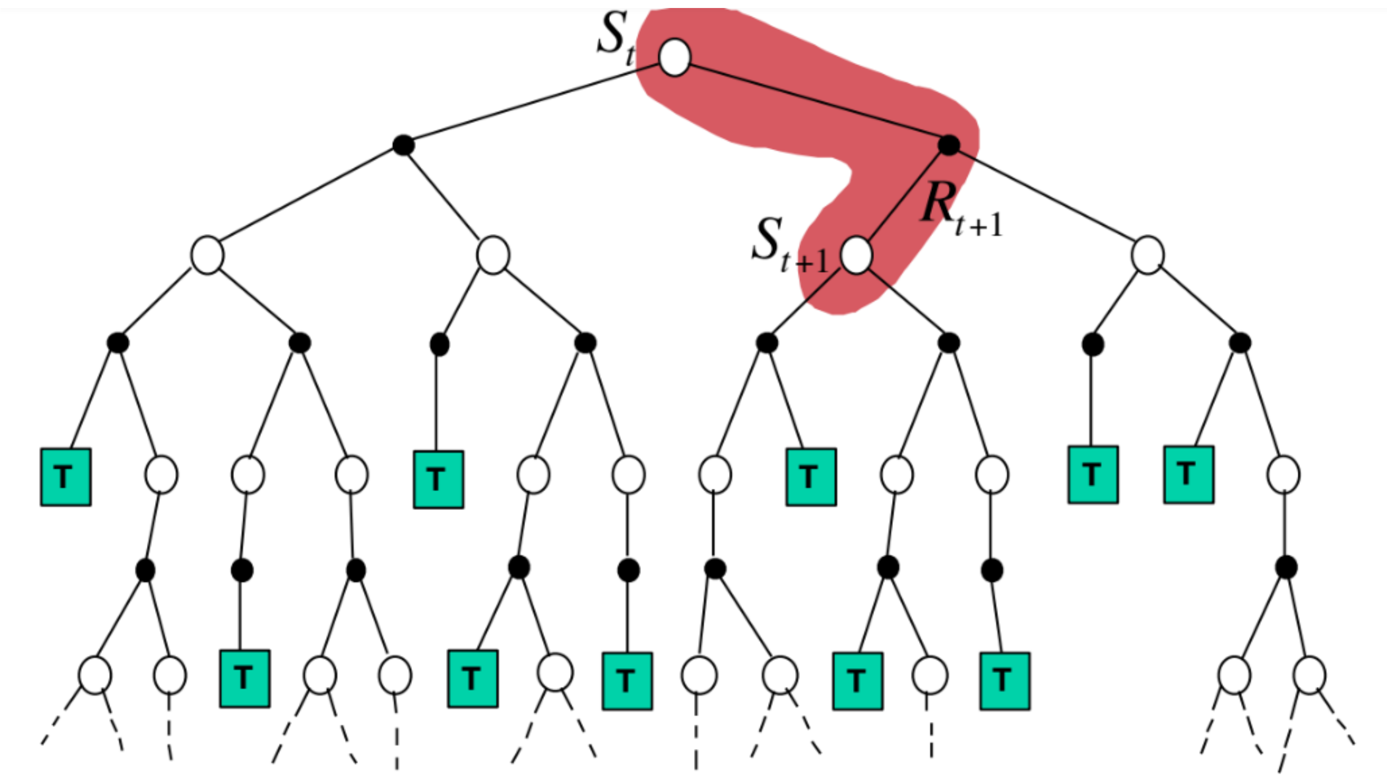
Monte Carlo

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



TD(0)

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



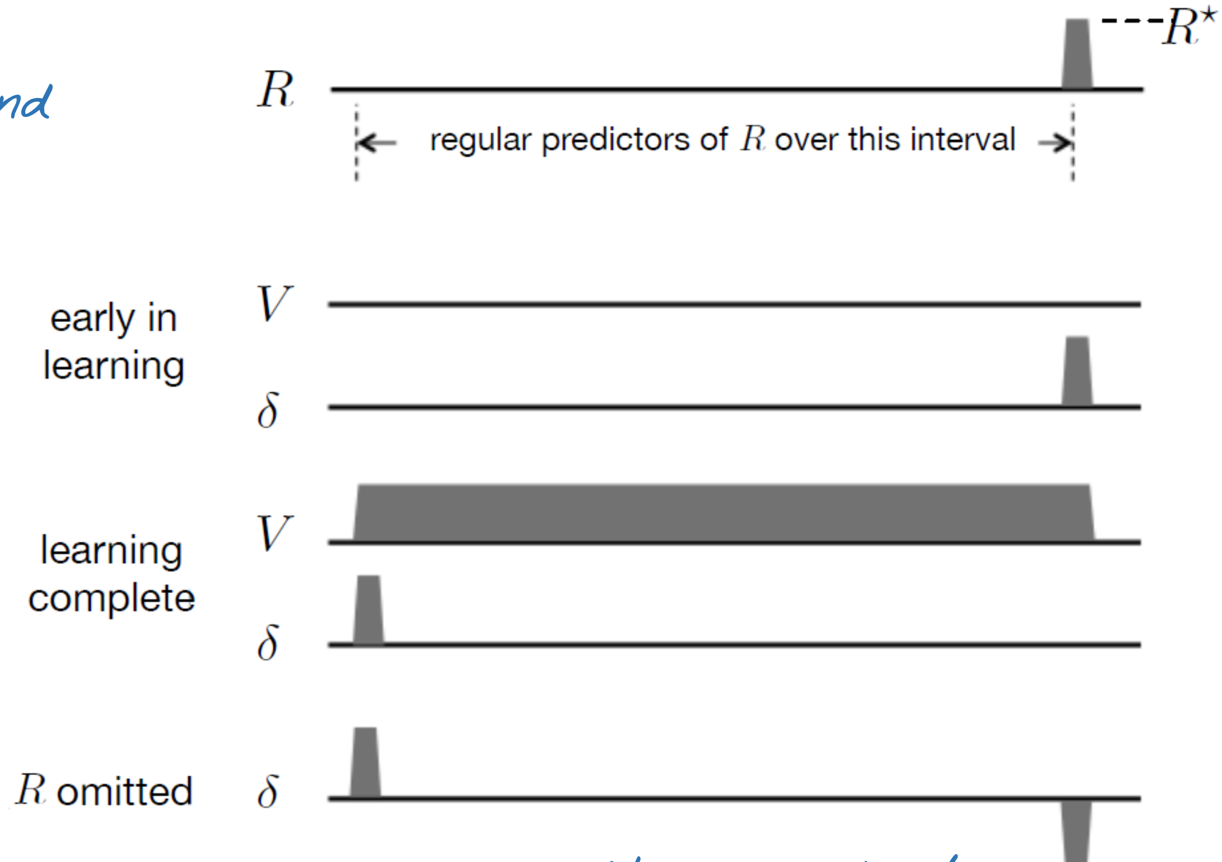
Aside

- $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is the learning signal
- $V(S_t) \leftarrow V(S_t) + \alpha\delta_t$
- The signal is high/low when something unexpectedly positive/negative happens

No discounting
Single reward at end

V is zero initially.
Then we see a reward!

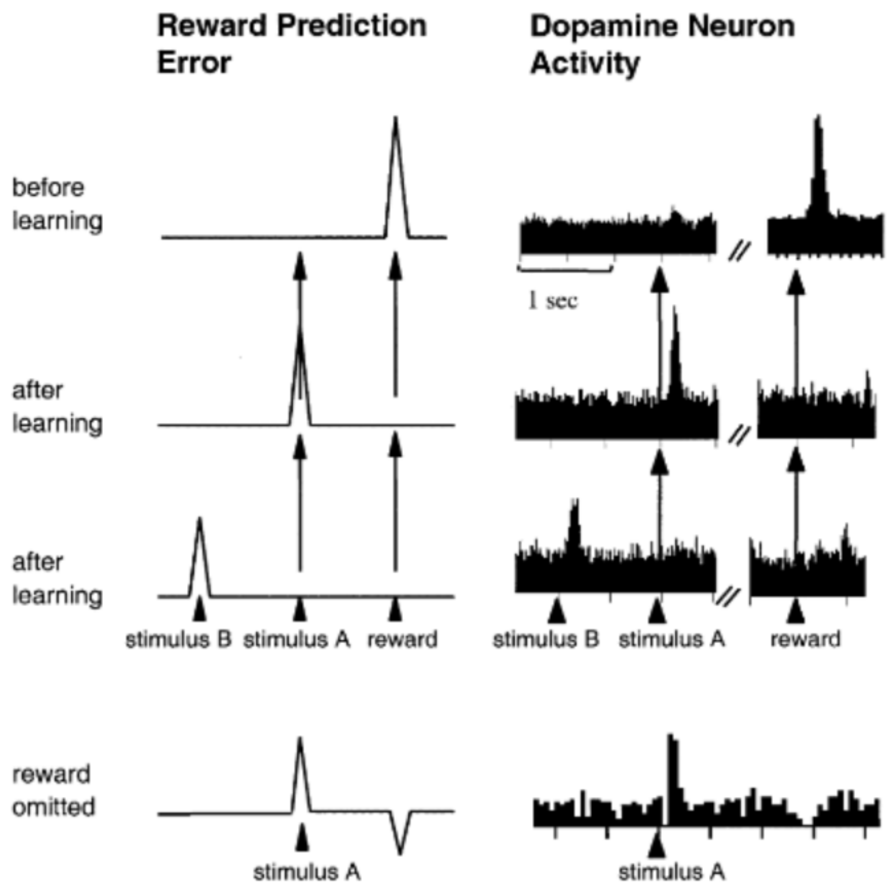
For any state that enters the predictive states



No reward when it was expected!
Negative error

Sutton and Barto [2018], Chapter 15

Dopamine



Schultz, W., Romo, R., Ljungberg, T., Mirenowicz, J., Hollerman, J. R., Dickinson, A. (1995). Reward-related signals carried by dopamine neurons. In J. C. Houk, J. L. Davis, and D. G. Beiser (Eds.), Models of Information Processing in the Basal Ganglia, pp. 233–248. MIT Press, Cambridge, MA.

TEMPORAL DIFFERENCE

Control

TD Control

- We have no model
  - Learning V is problematic. Why?

- We will use TD learning to update the Q-value function.

- Previously, we used $(S_t, A_t, R_{t+1}, S_{t+1})$

- Now we use $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$

TD Control

- 2 methods:
  - SARSA
  - Q-learning
- Difference is how they select A_{t+1}
- SARSA: select A_{t+1} according to policy that selected A_t (e.g. with ϵ-greedy exploration)

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

- Q-learning: $A_{t+1} = \mathrm{argmax}_a Q(S_{t+1}, a)$

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

Note on convergence

- Greedy in the limit with infinite exploration (GLIE)
  - All state-action pairs visited infinitely many times
  - Policy converges to greedy policy e.g. ε-greedy with $\varepsilon_k = \frac{1}{k}$
- Robbins-Monro sequence of learning rates α_t:

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

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

SARSA

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

Same policy

Q-learning

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

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 $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop 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

not ε-greedy

On-policy vs off-policy

- On-policy methods learn by following the current policy
- Off-policy methods learn from a different behaviour policy
- SARSA is on-policy. Q-learning is off-policy
  - Why?
- Why is this important?
  - Learn from observing humans or other agents
  - Re-use experience generated from old policies (e.g. replay buffers)
  - Learn about optimal policy while following exploratory policy

N-STEP RETURNS

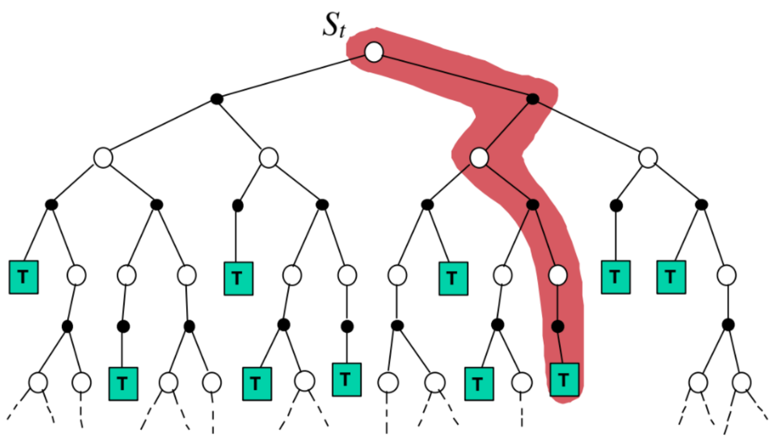
SARSA(λ)

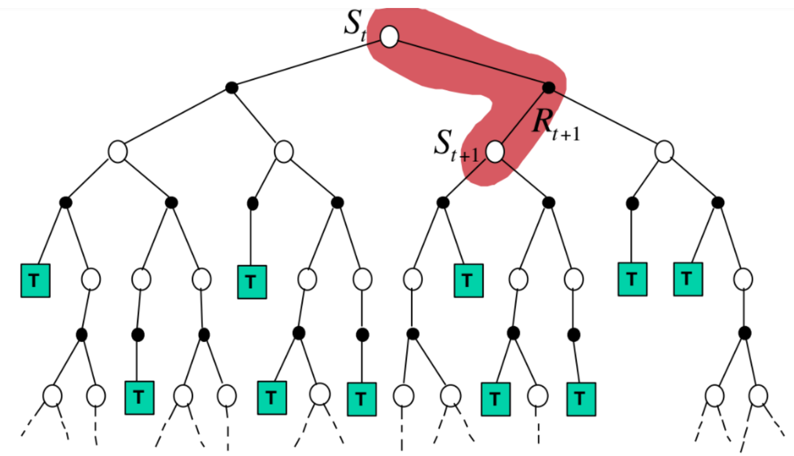
Between TD and MC

MC Methods (∞-step lookahead)

TD(0) Methods (one-step lookahead)

WHAT ABOUT 2-steps ahead? 3? 4? ...





n-step returns

- $n = 1$ (TD) $G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1})$
- $n = 2$ $G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2})$
- …
- $n = \infty$ (MC) $G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \ \ldots + \gamma^n R_T$

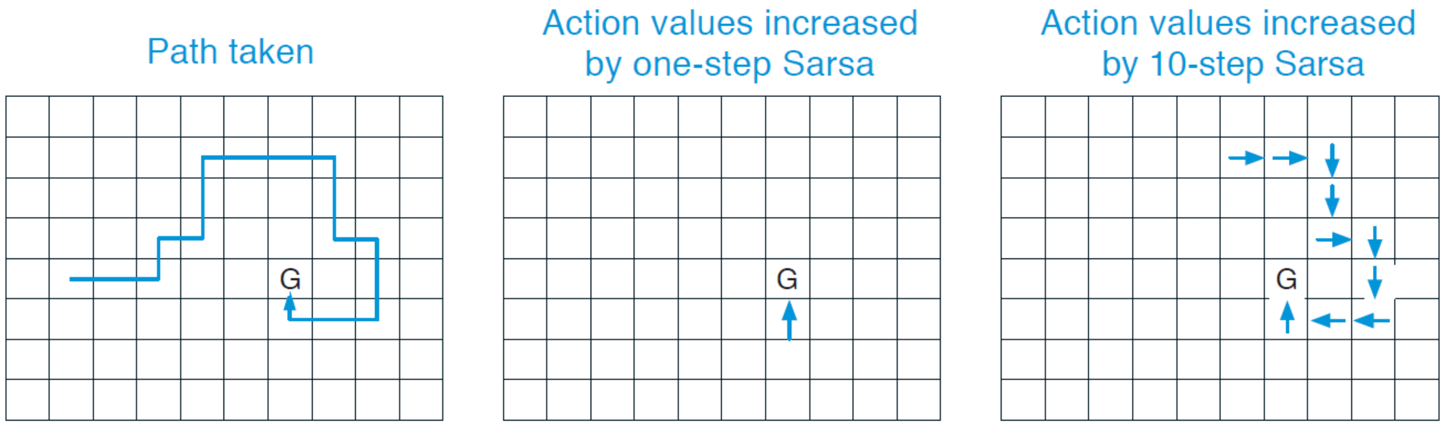
- n-step TD learning:

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

n-step returns

- More computation required

- But better estimates:
  - n-step return better than $(n-1)$-step return for approximating v_π



λ-returns

- How about averaging n-step returns over different n?

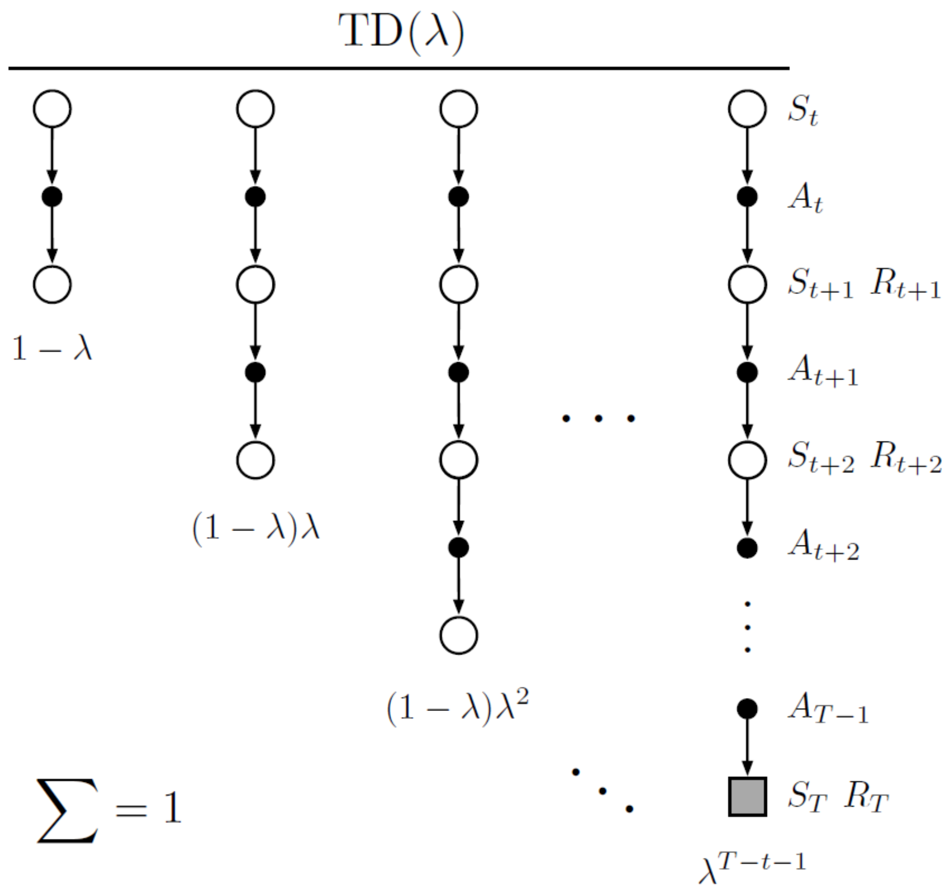
$$\frac{1}{2}G_t^{(1)} + \frac{1}{2}G_t^{(2)}$$

- How about averaging over all n?
- The λ-return combines all n-step returns with weights $(1 - \lambda)\lambda^{n-1}$

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

TD target

λ-returns



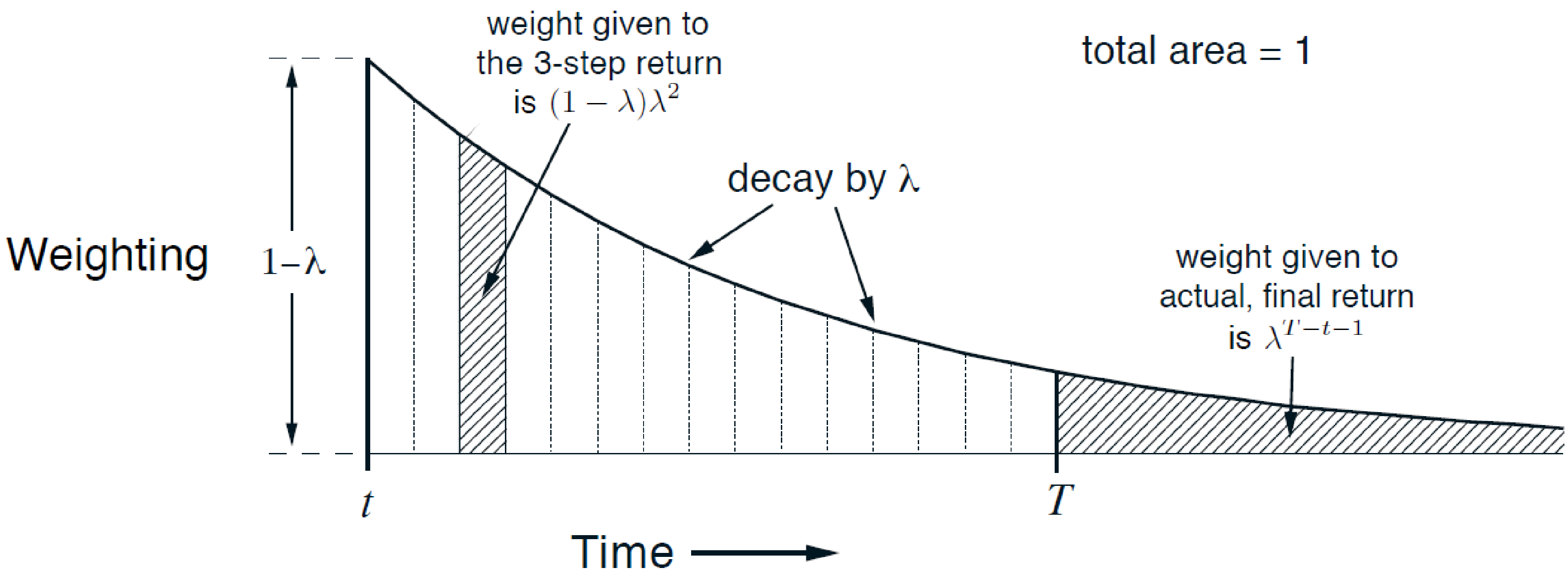
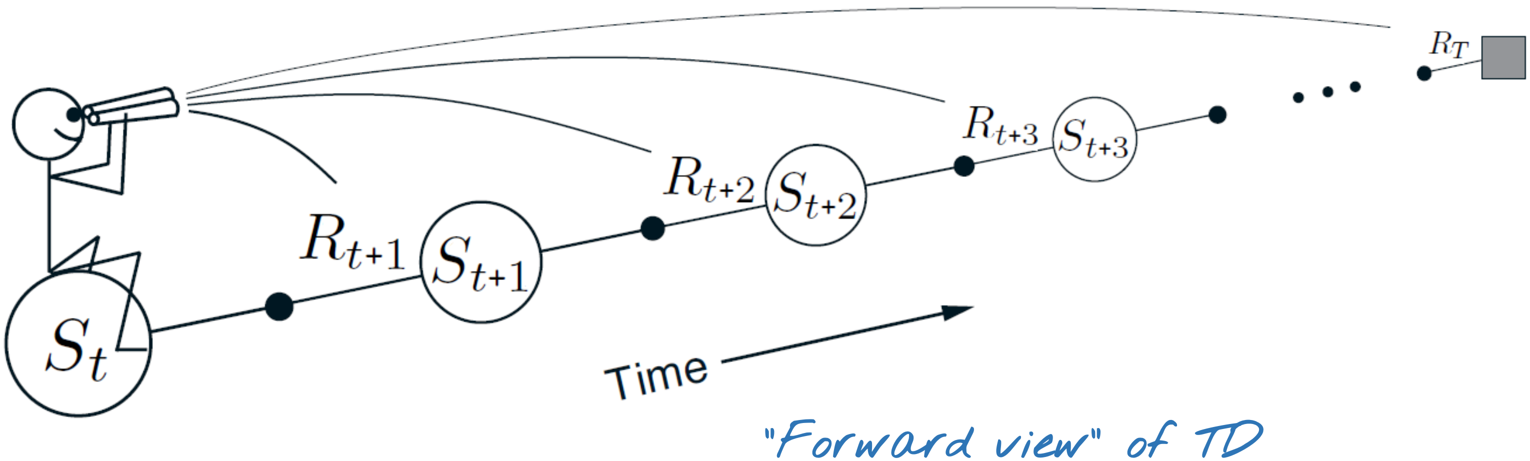


Figure 12.2: Weighting given in the λ-return to each of the n-step returns.

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

- What happens if $\lambda = 0$?
  - And for $\lambda = 1$?

n-step and λ-returns

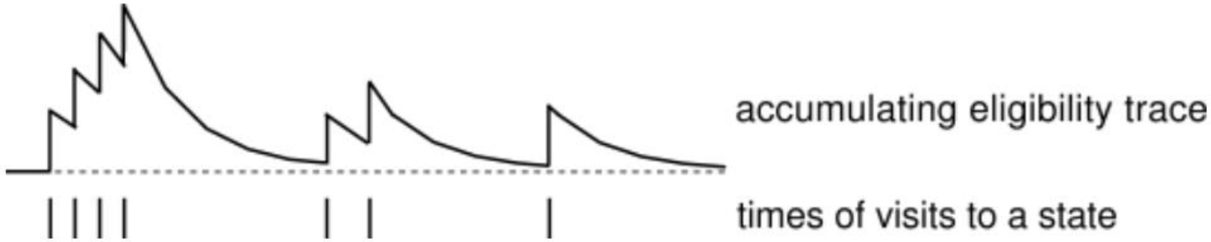


- Need to have knowledge of future rewards
- But we only know the present
- So we must wait until the end of the episode ☹

Eligibility traces

- A way of assigning credit backward in time
- Frequency heuristic: assign credit to most frequent states
- Recency heuristic: assign credit to most recent states
- Eligibility traces combine both heuristics

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



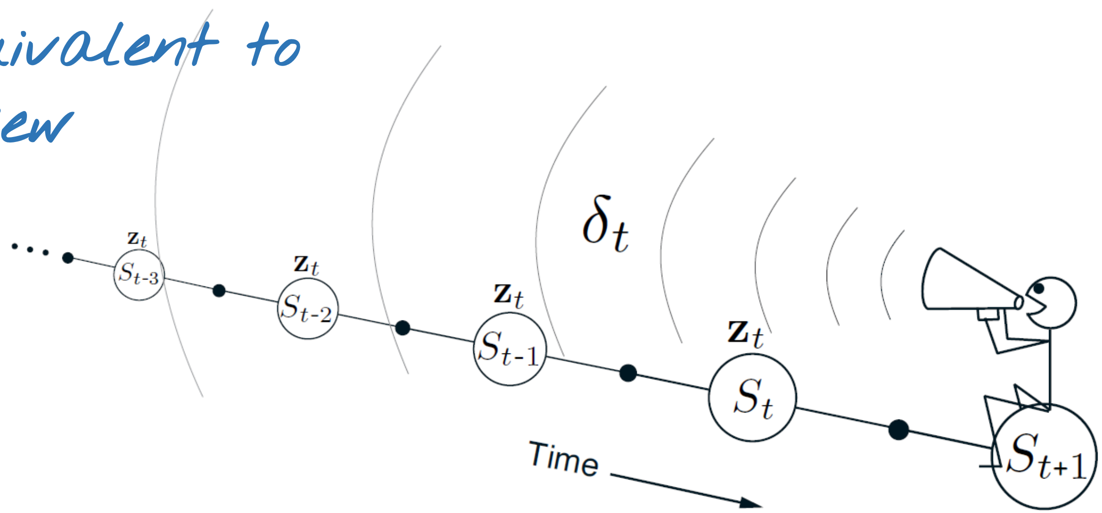
TD(λ)

- Keep an eligibility trace for every state s
- Update value in proportion to TD error and eligibility trace

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

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

Roughly equivalent to forward view



TD(λ) for policy evaluation

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop for each episode:
    Initialize S
    $\boldsymbol{e} \leftarrow \boldsymbol{0}$ (a vector the size of S)
    Loop for each step of episode:
        $A \leftarrow$ action given by π for S
        Take action A, observe R, S'
        $\delta = R + \gamma V(S') - V(S)$
        $e(S) \leftarrow e(S) + 1$
        For every state $t \in \mathcal{S}$:
            $V(t) \leftarrow V(t) + \alpha \delta e(t)$
            $e(t) \leftarrow \gamma \lambda e(s)$ (decay the eligibility trace of t)
        $S \leftarrow S'$
    until S is terminal

Note when $\boldsymbol{\lambda} = \boldsymbol{0}$ we get TD(0)

Summary

- Methods that learn from experience without a model

- MC: Update toward the full return:

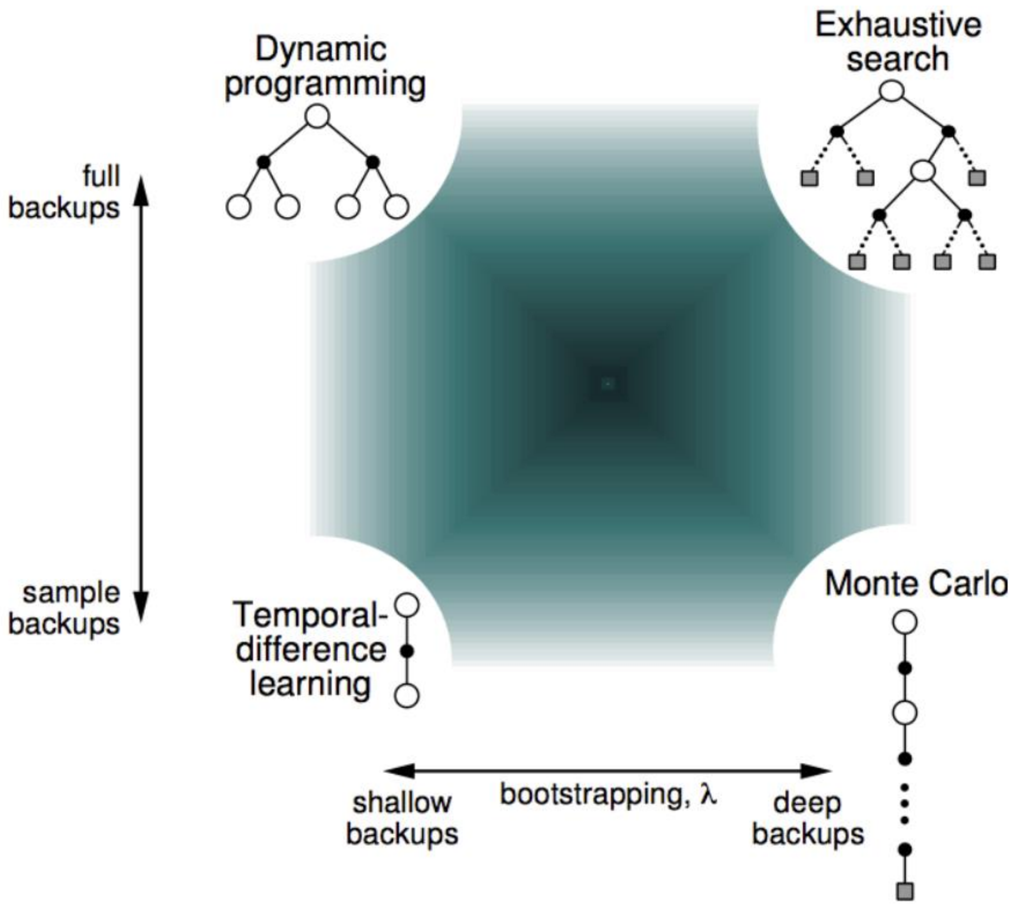
$$\delta_t^{MC} = G_t - V(S_t)$$

- TD(0): Update toward one-step difference:

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

- SARSA (on-policy TD) vs Q-learning (off policy TD)
- TD(λ): interpolate between TD and MC methods

Unified View of RL



Homework

- Use the CliffWalking domain from OpenAI gym
  - See Example 6.6, pg 132 in Sutton and Barto [2018]
- Modify the TD(λ) algorithm presented to implement SARSA(λ)
  - The only difference here is that there is an eligibility trace for each state-action pair!
  - Use ε-greedy policies with $\varepsilon = 0.1$ and a learning rate of $\alpha = 0.5$
  - Run SARSA(λ) on the domain for $\lambda = \{0, 0.3, 0.5\}$ for 200 episodes
    - Record the return for each episode
    - Average your returns over 100 runs

By next week's lecture, submit on Moodle:

1. Perform a single run of the algorithm. After each episode plot the value function (take $\max_a Q(s,a)$) learned so far as a heatmap for each λ side by side. This should result in 200 separate plots/images. Turn these images into an animation/video and submit it.
2. A combined plot of average return over time for the different values of λ. Include error bars/shading indicating variance in your results
3. Your code