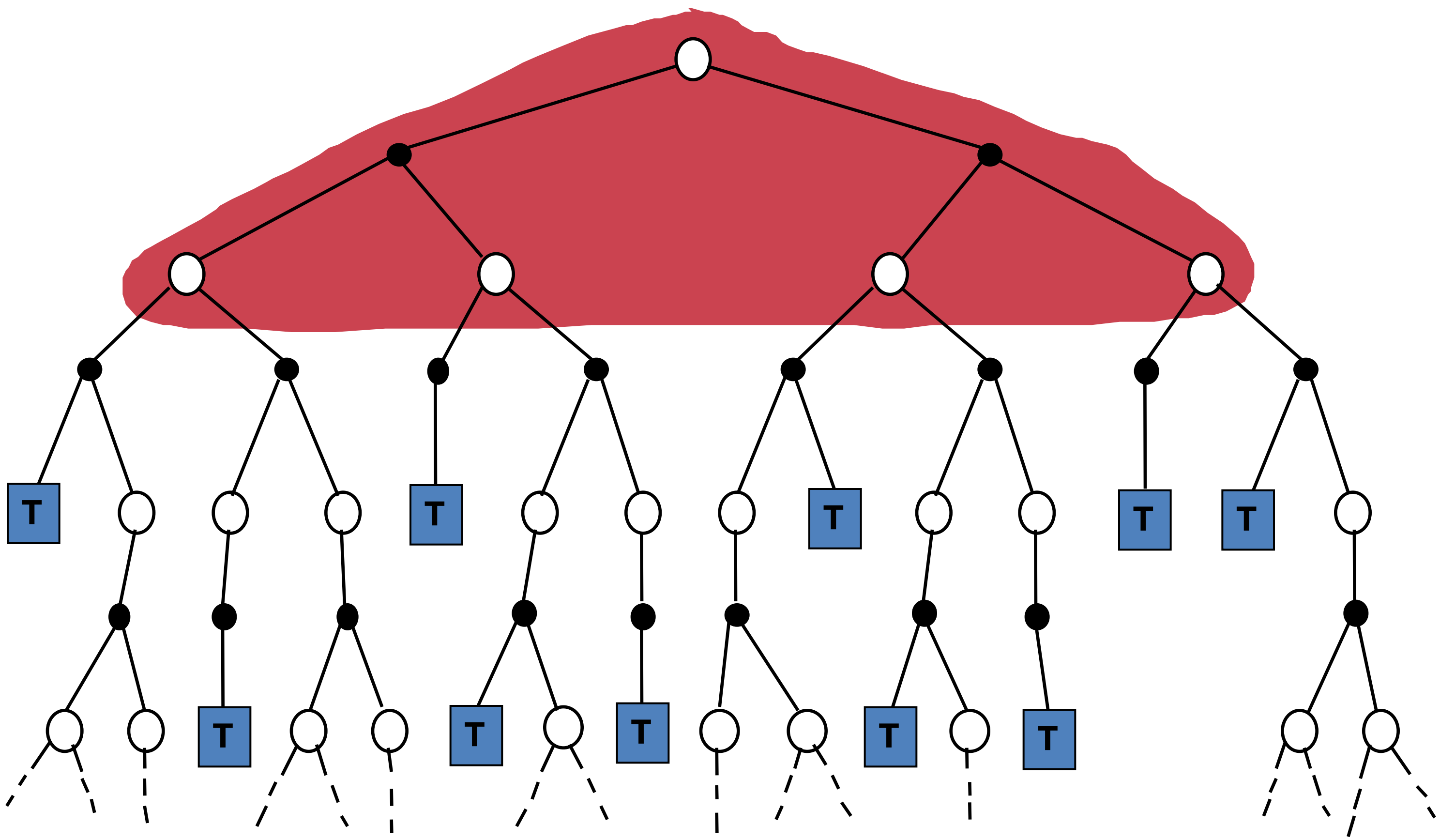
Lecture 5: Temporal Difference Learning and Monte-Carlo Methods

B. Ravindran

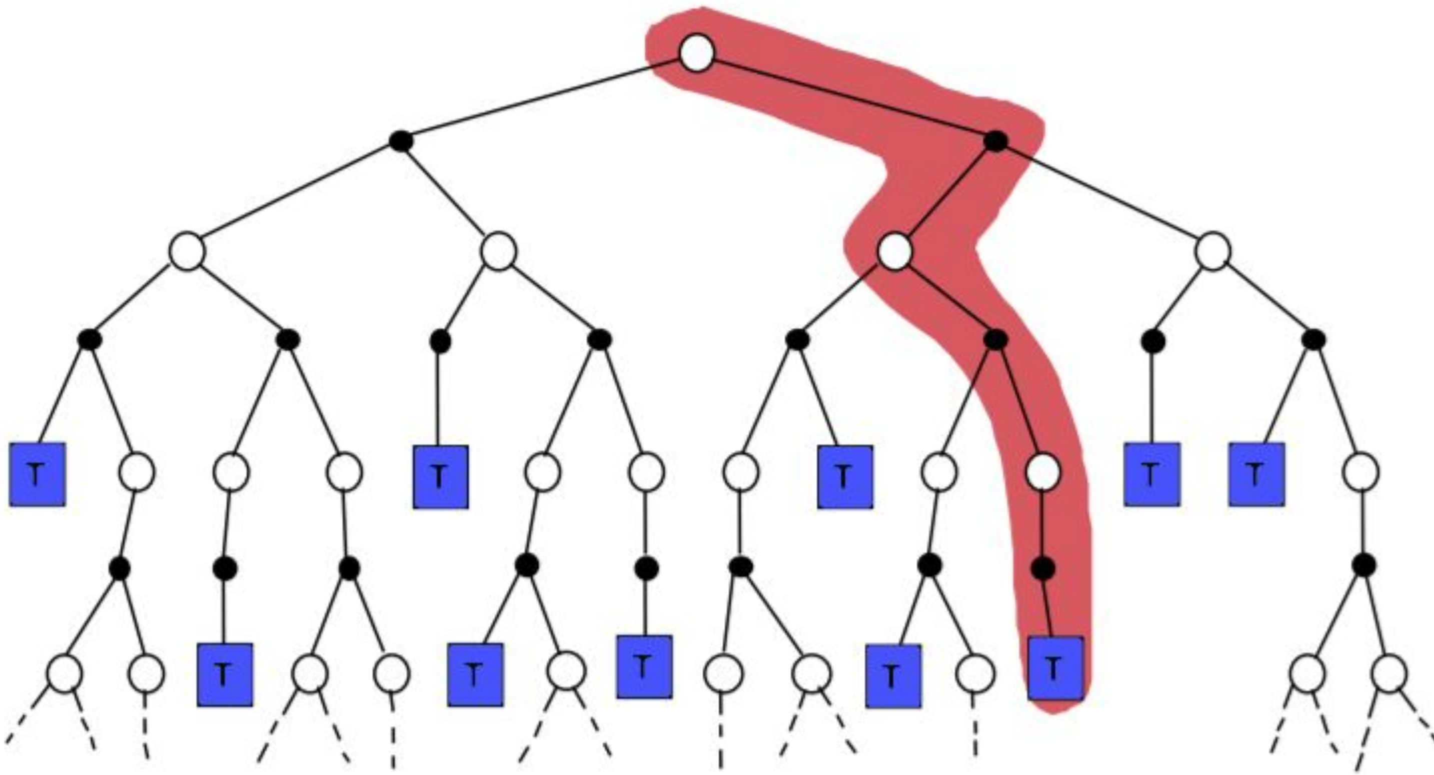
Dynamic Programming

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

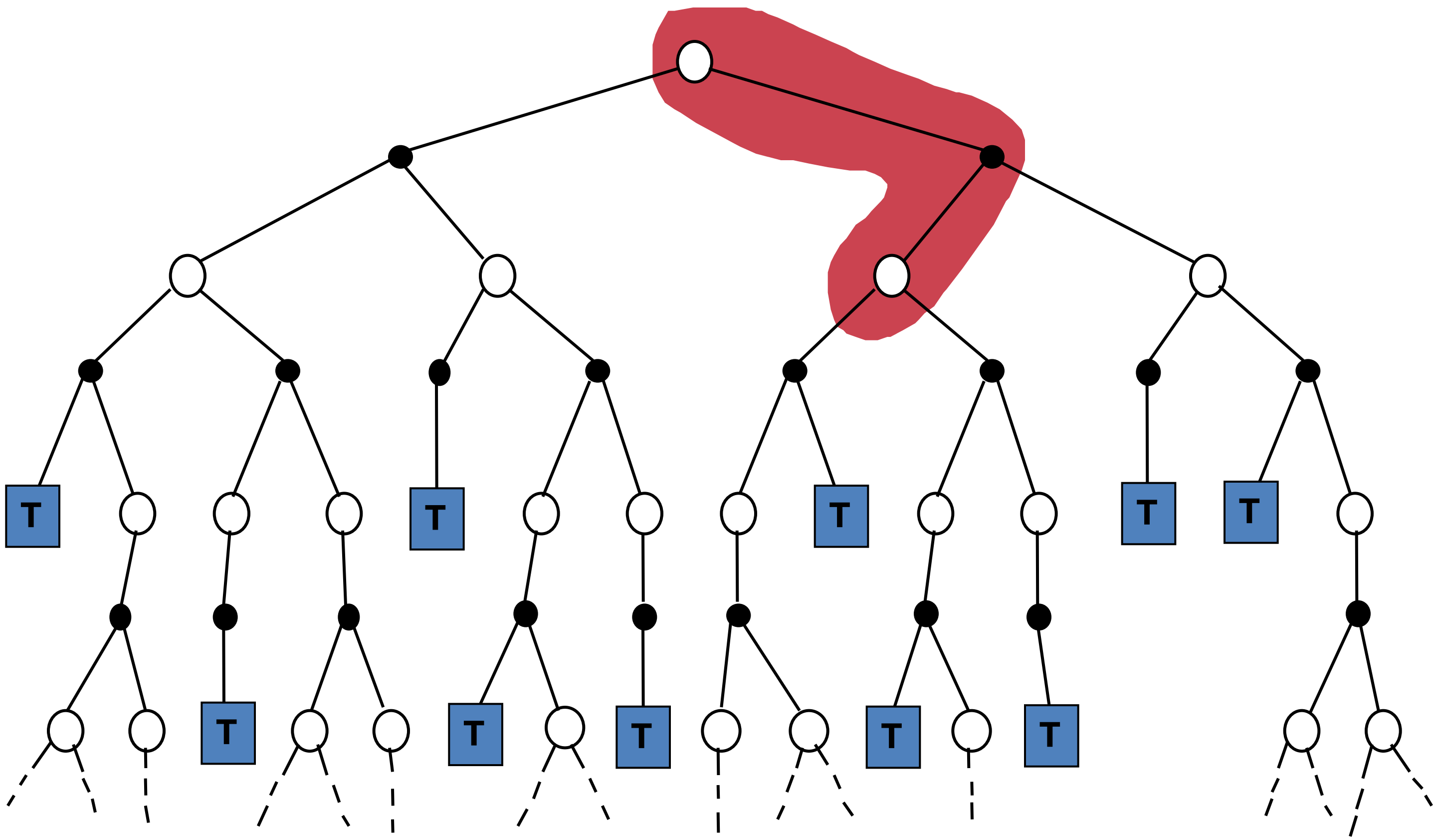


Monte-Carlo Reinforcement Learning

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s], \text{ for all } s \in \mathcal{S},$$

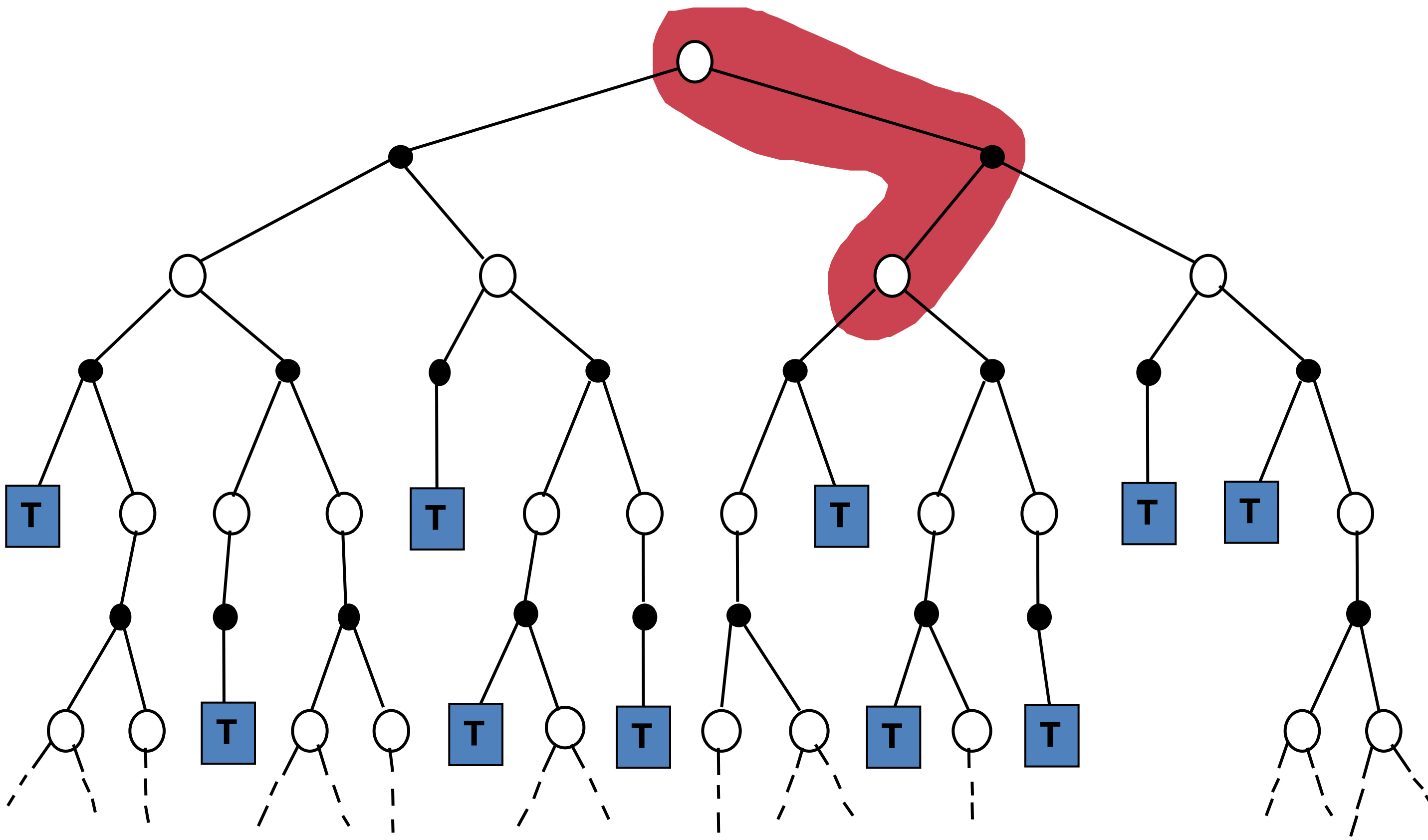


Simplest "RL" Method



Simplest "RL" Method

$$V(s_t) \leftarrow V(s_t) + \alpha\left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right]$$



Temporal Difference

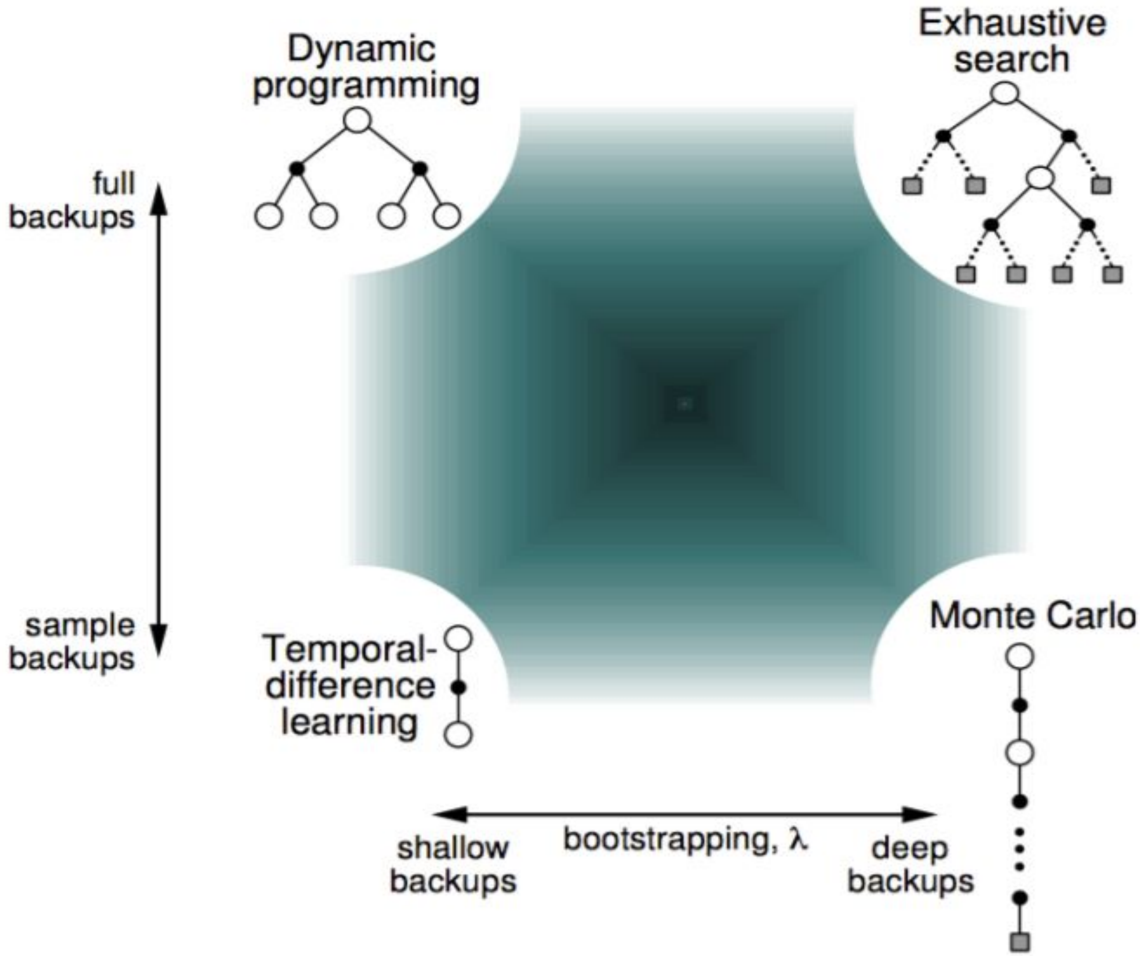
- Simple rule to explain complex behaviors
- Intuition: Prediction of outcome at time $t+1$ is better than the prediction at time t. Hence use the later prediction to adjust the earlier prediction.
- Has had profound impact in behavioral psychology and neuroscience!

Bootstrapping and Sampling

- Bootstrapping: Update using an estimate
  - DP and TD bootstrap
  - Monte Carlo does not bootstrap.

- Sampling: Update calculated using samples without model
  - TD and Monte Carlo sample.
  - DP (typically) does not sample

Bootstrapping and Sampling



Monte-Carlo Reinforcement Learning

- Learning directly from sample episodes of experience
- Does not use a known model and is model-free
- MC does not use bootstrapping
- Value functions are calculated as mean of discounted returns (G_t)

Monte-Carlo Prediction

- First-visit MC method estimates $V(s)$ as the average of the returns following first visits to s.
- Every-visit MC method averages returns following all visits to s.

Monte-Carlo Prediction : First Visit MC

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

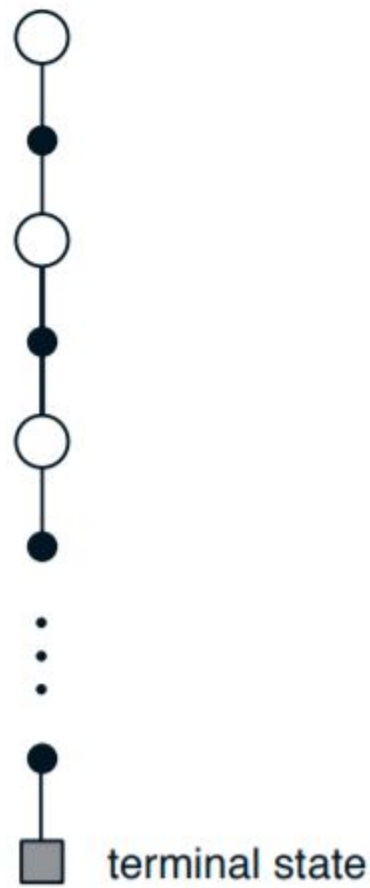
For each state s appearing in the episode:

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

Append G to $Returns(s)$

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

Monte-Carlo Prediction : First Visit MC



Monte-Carlo Control

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:
        $G \leftarrow$ 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:
        $\pi(s) \leftarrow \arg\max_a Q(s,a)$

Bootstrapping and Sampling

- Bootstrapping: Update using an estimate
  - DP and TD bootstrap
  - Monte Carlo does not bootstrap.

- Sampling: Update calculated using samples without model
  - TD and Monte Carlo sample.
  - DP does not sample

Advantages of TD

- TD methods (like MC) do not require a model of the environment, only experience (sampling)

- TD methods can be fully incremental (bootstrapping)
  - You can learn before knowing the final outcome
    - Less memory
    - Less peak computation
  - You can learn without the final outcome
    - From incomplete sequences

- TD methods thus combine individual advantages of DP and MC.

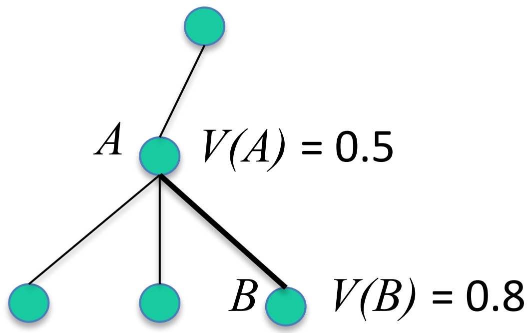
TD Prediction

- Policy Evaluation (the prediction problem): for a given policy, compute the state-value function.
- No knowledge of p and r, but access to the real system, or a "sample" model assumed.
- Uses "bootstrapping" and sampling

The simplest TD method, TD(0):

$$V(s_t) \leftarrow V(s_t) + \alpha\left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right]$$

TD Update Example

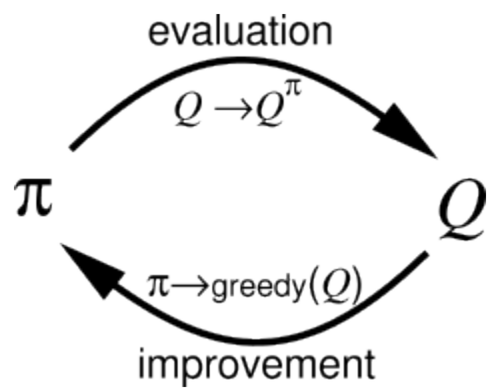


$$\text{Assuming :}$$
$$\text{reward, } r, \text{ A} \rightarrow \text{B} : 0$$
$$\alpha : 0.2$$
$$\gamma : 0.9$$

$$V(A) = V(A) + \alpha[r + \gamma V(B) - V(A)]$$
$$V(A) = 0.5 + 0.2[0 + 0.9 * 0.8 - 0.5] = 0.544$$

TD Control

- The control problem: approximate optimal policies.
- Recall the idea of GPI:



- Policy evaluation: use TD(0) to evaluate value function.
- Policy improvement: make policy **greedy** wrt current value function.
- Note that we estimate action values rather than state values in the absence of a model.

ε-Greedy Policies

$$
\begin{aligned}
&a^* \leftarrow \arg\max_a Q(s,a) \\
&\text{For all } a \in \mathcal{A}(s)\text{:} \\
&\qquad \pi(a|s) \leftarrow \begin{cases} 1-\varepsilon+\varepsilon/|\mathcal{A}(s)| & \text{if } a = a^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq a^* \end{cases}
\end{aligned}
$$

- Any ε-greedy policy with respect to Q following π is an improvement over any ε-soft policy π is assured by the policy improvement theorem

Sarsa: On-Policy TD Control

- In on-policy control, we try improving the policy used for making decisions.

SARSA

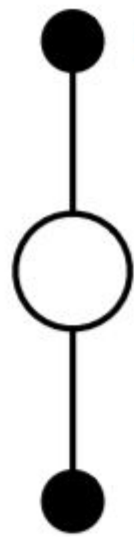
After every transition from a nonterminal state s_t, do this:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \, Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

If s_{t+1} is terminal, then $Q(s_{t+1}, a_{t+1}) = 0$.

- Convergence is guaranteed as long as
  - all state-action pairs are visited an infinite number of times
  - the policy converges in the limit to the greedy policy

Sarsa: On-Policy TD Control



$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma\, Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right]$$

Sarsa Algorithm

Initialize $Q(s,a)$ arbitrarily

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'

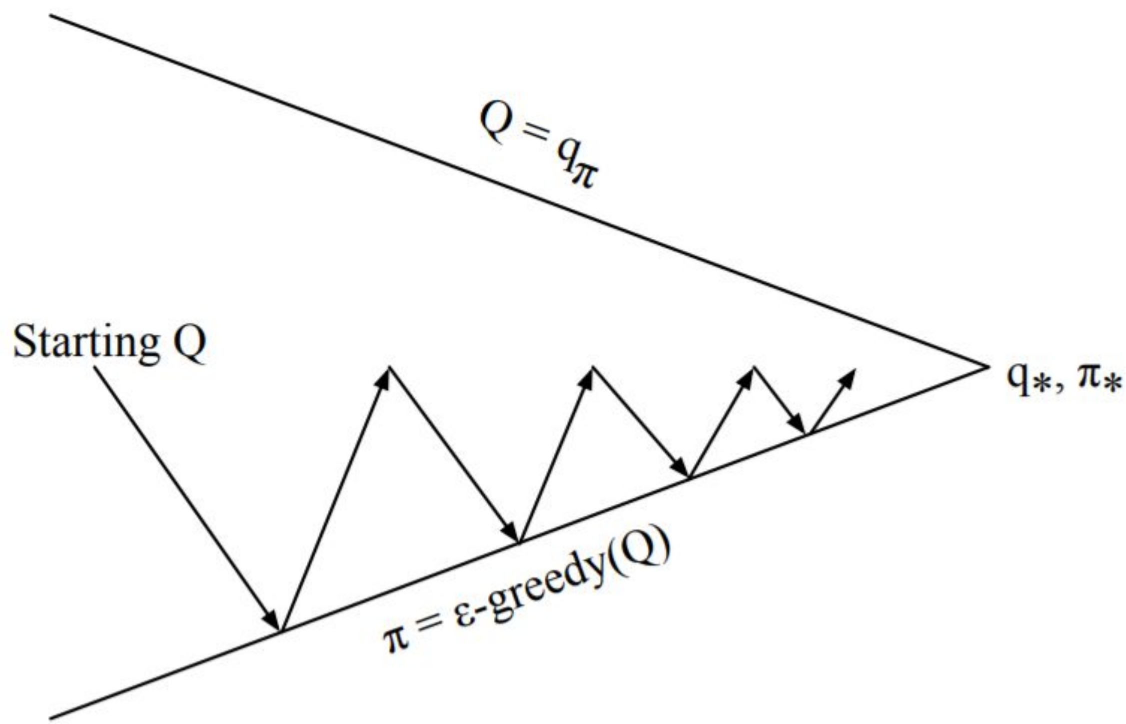
    Choose a' from s' using policy derived from Q (e.g., ε - greedy)

    $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma Q(s',a') - Q(s,a)\right]$

    $s \leftarrow s'; a \leftarrow a';$

  until s is terminal

Sarsa Algorithm



Every time-step:

Policy evaluation Sarsa, $Q \approx q_\pi$

Policy improvement ϵ-greedy policy improvement

Q-Learning: Off-Policy TD Control

- In off-policy control, we have two policies:
  - the behavior policy – used to generate behavior
  - estimation policy – the policy that is being evaluated and improved.

Q-learning

One-step Q-learning:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

Q-learning Algorithm

Initialize $Q(s,a)$ arbitrarily

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)

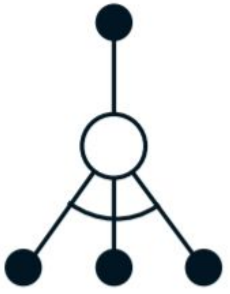
    Take action a, observev r, s'

    $Q(s,a) \leftarrow Q(s,a) \ + \ \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$

    $s \leftarrow s'$;

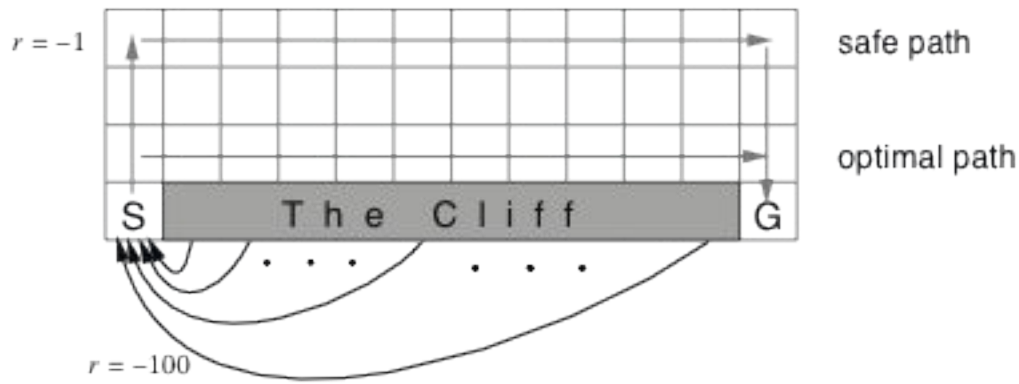
until s is terminal

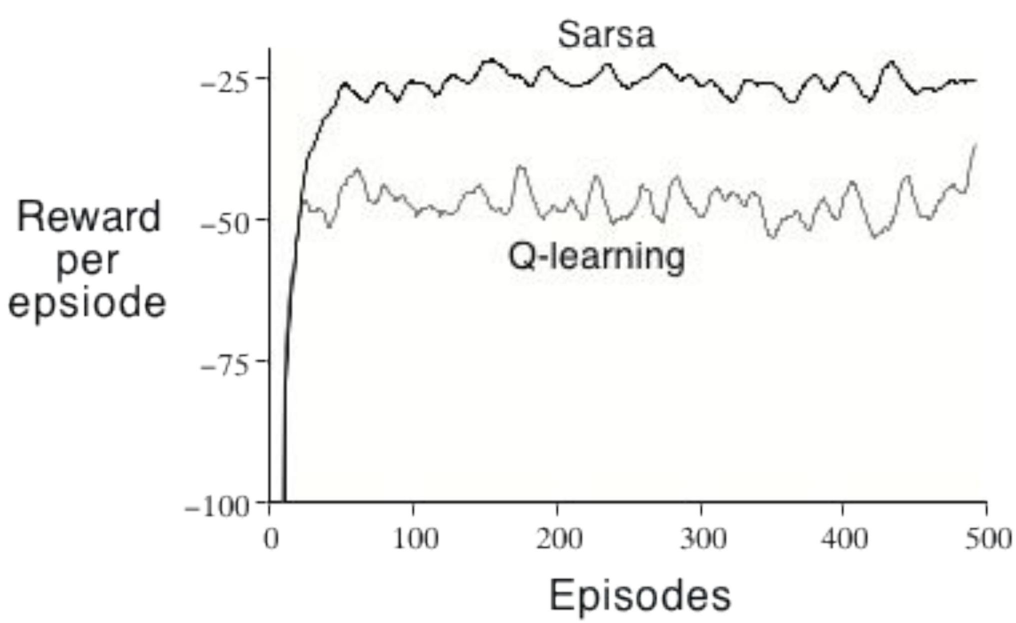
Q-learning Algorithm



$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

Cliff Walking: SARSA vs Q-learning





n-Step TD Prediction

Consider TD(0)

$$V(s_t) \leftarrow V(s_t) + \alpha\left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right]$$

Here, the target (for estimating TD error) contains only the next step reward:

$$R_t^{(1)} = r_{t+1} + \gamma V_t(s_{t+1})$$

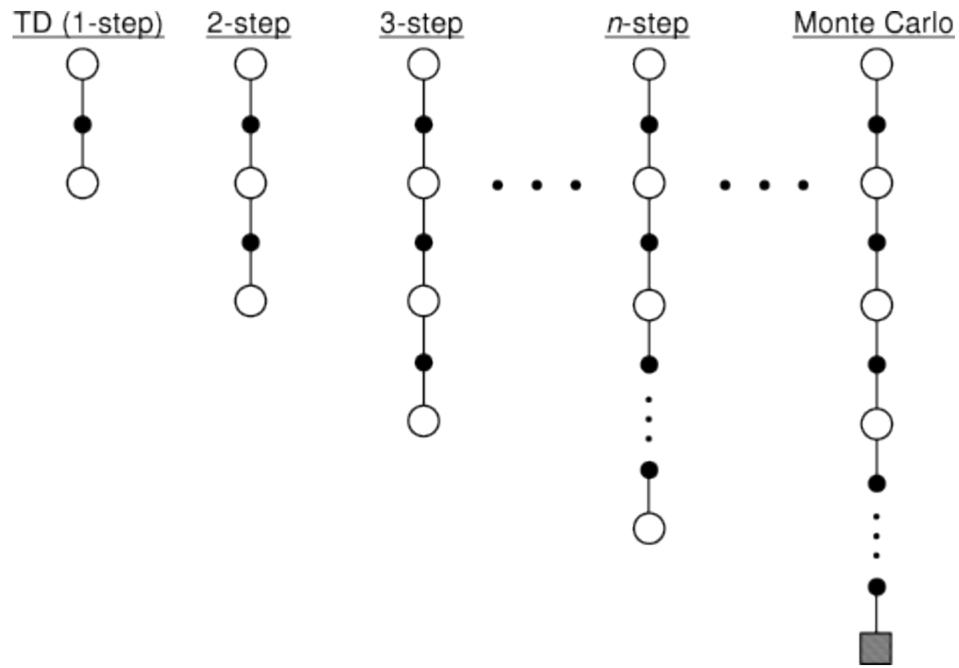
Alternatively, we can consider the rewards received in the next ***n*** steps:

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{n-1} r_{t+n} + \gamma^n V(s_{t+n}) - V(s_t)]$$

$$R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots + \gamma^{n-1} r_{t+n} + \gamma^n V_t(s_{t+n})$$

The extreme would be to consider rewards till the end of the episode (Monte Carlo).

n-Step TD Prediction Cont.



The spectrum of back-ups from one-step TD to up-until-termination MC

TD(λ)

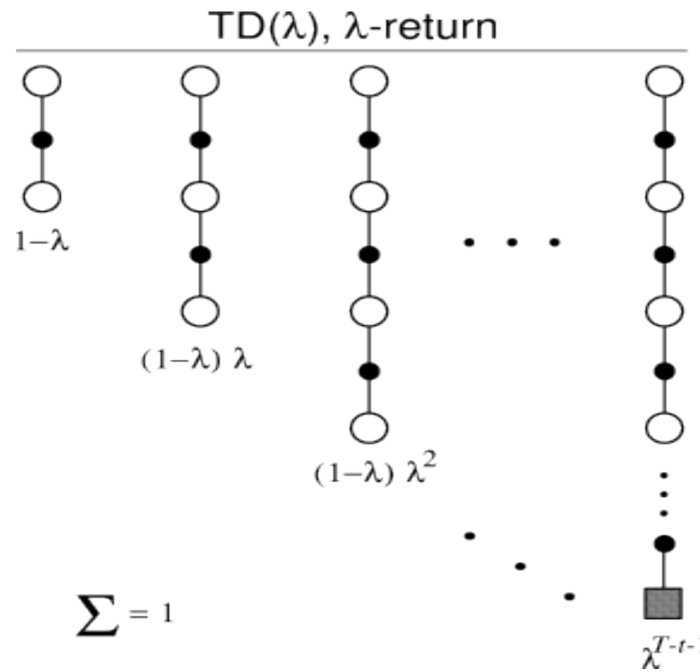
Instead of using n-step backups, we can consider an average of n-step returns.

Example: $R_t^{avg} = 1/2 R_t^{(2)} + 1/2 R_t^{(4)}$

In TD(λ), the average contains all the n-step backups each weighted proportional to λ^{n-1}, where $0 \leq \lambda \leq 1$.

λ-return:

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

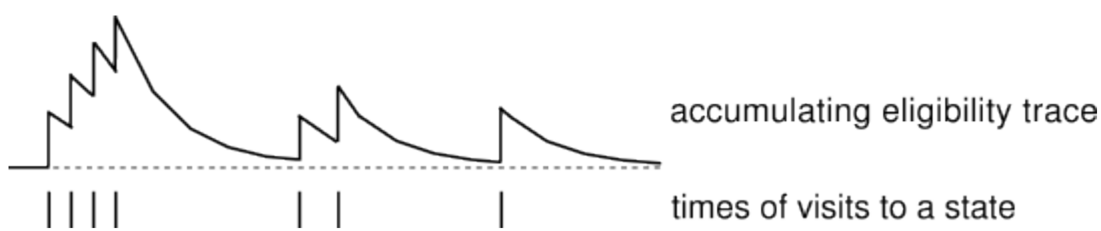


Eligibility Traces

- To implement TD(λ) we use the concept of eligibility traces.
- These are variables associated with each state denoted by $e_t(s)$.
- They indicate the degree to which each state is eligible for undergoing learning changes.
- On each step:

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{if } \mathrm{s} \neq \mathrm{s_t}; \\ \gamma\lambda e_{t-1}(s) + 1 & \text{if } \mathrm{s} = \mathrm{s_t} \end{cases}$$

for all $s \in S$, where γ is the discount rate.



TD(λ) Algorithm

Initialize V(s) arbitrarily and e(s) = 0, for all $s \in S$

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 state, s'

$\underline{\delta \leftarrow r + \gamma V(s')\text{-}V(s)}$

$e(s) \leftarrow e(s) + 1$

For all s :

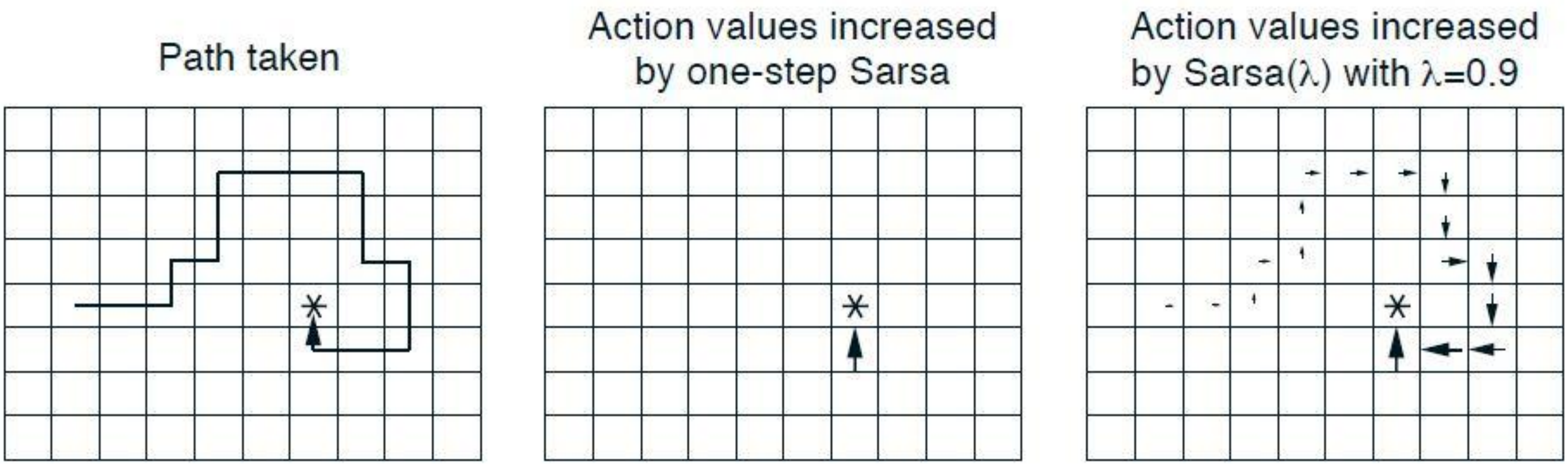
$\underline{V(s) \leftarrow V(s) + \alpha\delta e(s)}$

$e(s) \leftarrow \gamma\lambda e(s)$

$s \leftarrow s'$

until s is terminal

Speedup in Policy Learning



TD(0) vs TD(λ)

Example grid world domain

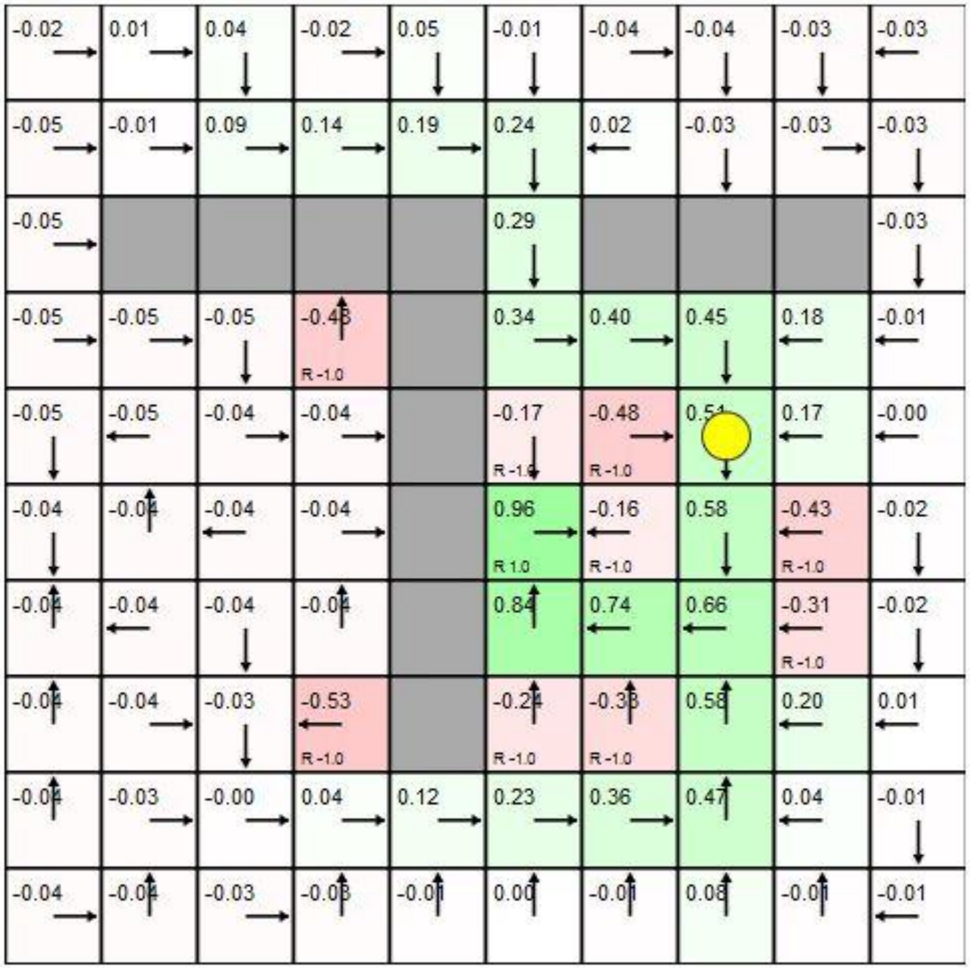
Grey cells = obstacles

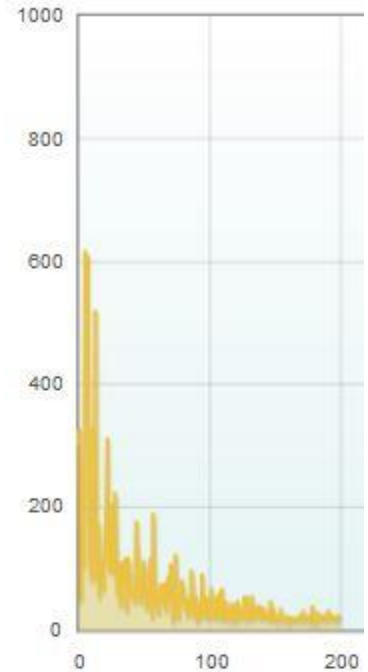
Notice R = 1.0 for centre cell with most surrounding cells having R = -1.0

Cell values = value function estimates

Interactive demo at:
cs.stanford.edu/people/karpathy/reinforcejs/gridworld_td.html

TD(0) vs TD(λ) Contd.





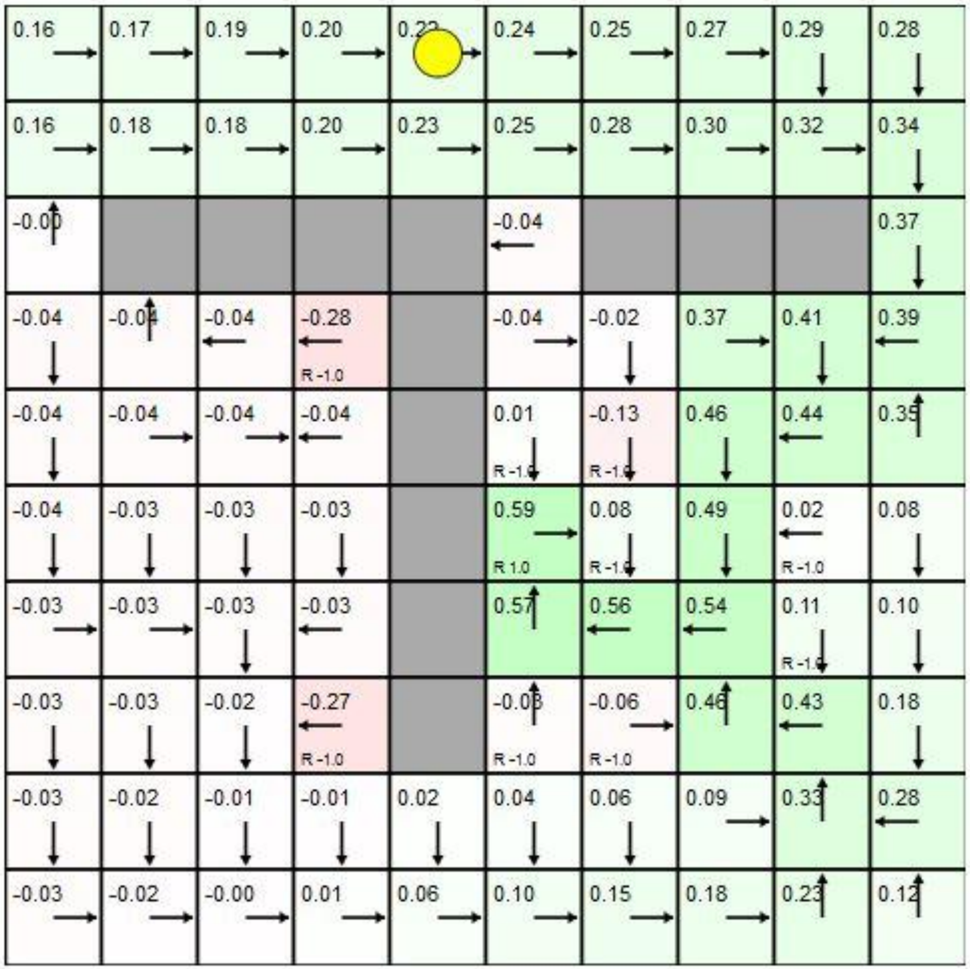
Number of actions before reaching the goal state

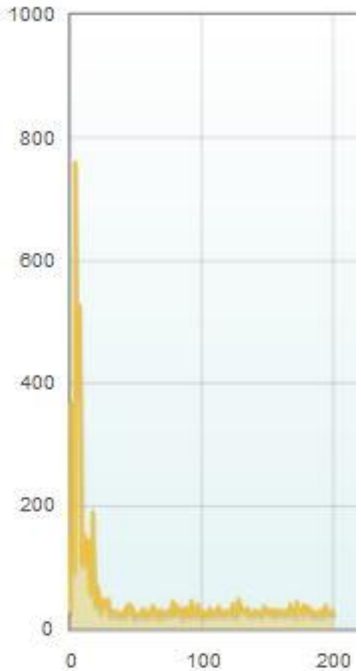
$$\text{Algo}: \text{Q-learning}$$

$$\gamma = 0.9, \varepsilon = 0.2, \alpha = 0.1, \underline{\lambda = 0}$$

Interactive demo at:
cs.stanford.edu/people/karpathy/reinforcejs/gridworld_td.html

TD(0) vs TD(λ) Contd.





Number of actions before reaching the goal state

$$\text{Algo}:\text{Q-learning}$$

$$\gamma = 0.9, \varepsilon = 0.2, \alpha = 0.1, \lambda = 0.8$$

Interactive demo at:
cs.stanford.edu/people/karpathy/reinforcejs/gridworld_td.html

More on Off-Policy Learning

- Evaluate target policy $\pi(a|s)$ to compute $v_\pi(s)$ or $q_\pi(s, a)$
- Follow behaviour policy $\mu(a|s)$
- Assumption of coverage:
  - Every action taken under π is also taken, at least occasionally, under μ
  - $\pi(a|s) > 0$ implies $\mu(a|s) > 0$

- e.g., π can be greedy while μ can be ε-greedy

Importance Sampling

- A general technique for estimating expected values under one distribution given samples from another.

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

Off-Policy MC with Weighted Importance Sampling

Importance-sampling ratio:

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

Weighted average return for V:

$$V(s) = \frac{\sum_{t \in \mathfrak{T}(s)} \rho_t^{T(t)} G_t}{\sum_{t \in \mathfrak{T}(s)} \rho_t^{T(t)}},$$