07 Model-Free Prediction (MC & TD Learning)

Table of contents

Model-Free Reinforcement Learning

Last Module (5):

- Integrating learning and planning
- Use planning to construct a value function or policy

This Module (6):

- Model-free prediction
- Prediction: *Optimise* the value function of an unknown MDP

Monte-Carlo Learning

Monte-Carlo Reinforcement Learning

MC methods learn directly from episodes of experience

- MC is model-free: no knowledge of MDP transitions / rewards

MC learns from complete episodes

- No bootstrapping, as we will see later

MC uses the simplest possible idea of looking at sample returns: *value = mean return*

- Caveat: can only apply to *episodic* MDPs, i.e. all episodes must terminate

Monte-Carlo Policy Evaluation

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

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

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

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

- Recall 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
- Computes empirical mean from the point t onwards using as many samples as we can
- Will be different for every time step

First-Visit Monte-Carlo Policy Evaluation

To evaluate state, s:

- On the first time-step, t that state, s, is visited in an episode:
  - Increment counter $N(s) \leftarrow N(s) + 1$
  - Increment total return $S(s) \leftarrow S(s) + G_t$
- Estimate:

$$V(s) = \frac{S(s)}{N(s)}$$

By Law of Large Numbers, $V(s) \to v_\pi(s)$ as $N(s) \to \infty$

- Only requirement is we somehow visit all of these states

The **Central Limit Theorem** tells us how quickly it approaches the mean

- The variance (mean squared error) of estimator reduces with $\frac{1}{N}$
- i.e. rate is independent of size of state space, $|s|$
- speed depends on how many episodes/visits reach s (coverage probabilities).

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 $S(s) \leftarrow S(s) + G_t$
- Estimate:

$$V(s) = \frac{S(s)}{N(s)}$$

Again, $V(s) \to v_\pi(s)$ as $N(s) \to \infty$.

First-Visit versus Every-Visit Monte-Carlo Policy Evaluation?

Every-Visit Advantages:

- Especially good when episodes are short or when states are *rarely* visited — no sample gets "wasted."
- i.e. uses more of the data collected, often faster convergence in practice.

First-visit Advantages:

- Useful when episodes are long and states repeated many times.
- i.e. avoids dependence between multiple visits to the same state in one episode.

Blackjack Example

States (~ 200):

- Current sum of cards ($12 - 21$)
- Dealer's showing card (Ace-10)
- Whether you have a *usable* ace - (can be counted as 1 *or* 11 without *busting* > 21) (yes/no)

Actions:

- Stand: stop receiving cards (and terminate)
- Hit: take another card (no replacement)

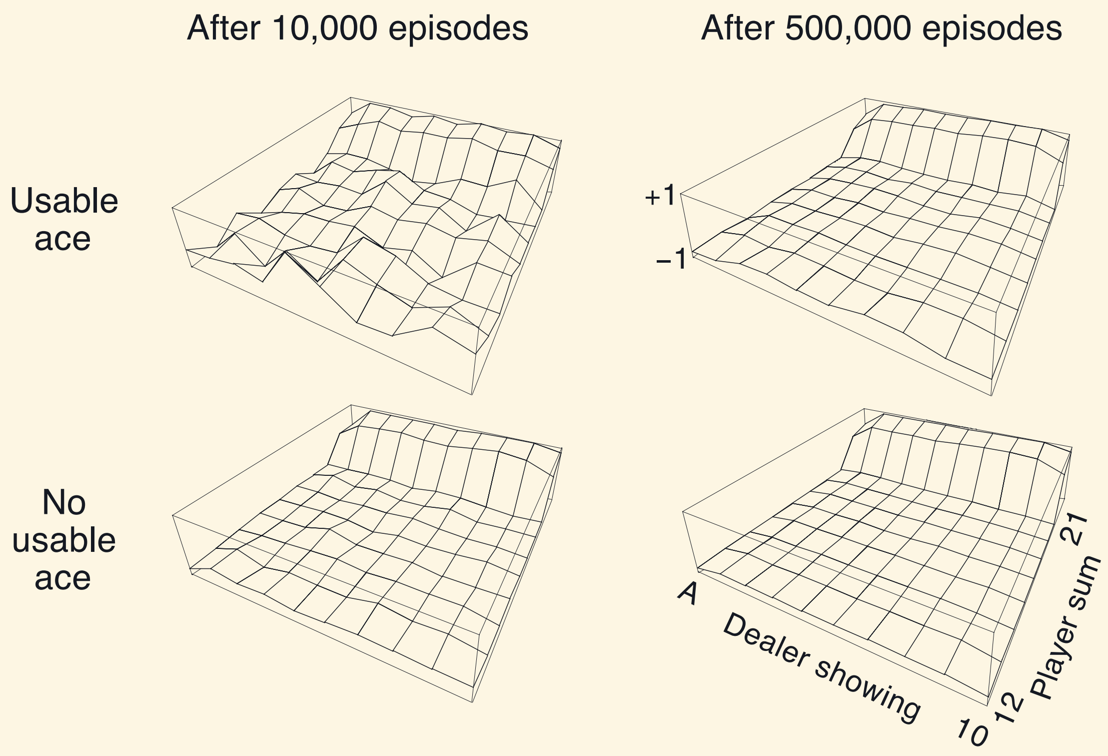
Transitions:

- You are automatically hit if your sum < 12

Rewards:

- For stand: $+1$ if your sum $>$ dealer's; 0 if equal; -1 if less
- For hit: -1 if your sum > 21 (and terminate); 0 otherwise

Blackjack Value Function after Monte-Carlo Learning



Policy: *stand* if sum of cards ≥ 20, otherwise *hit*

- Learning value function directly from experience
- Usable ace value is noisier because the state is *rarer*

Key point: once we have learned the value function from experience,

- we can *evaluate* actions for making the best decision for optimising a *policy* as we will see later

Incremental Mean (Refresher)

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

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

- μ_{k-1} is the previous mean: *predicts* what think value will be
- x_k is the new value
- Incrementally *corrects* mean $\frac{1}{k}$ in direction of error $x_k - \mu_{k-1}$

Incremental Monte-Carlo Updates (Same idea)

Update $V(s)$ incrementally after each episode $S_1, A_1, R_2, \ldots, S_T$:

- For each state S_t with return G_t:

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

In non-stationary problems, it can be useful to track a *running mean* by forgetting old episodes using a *constant* step size α:

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

A constant step size turns the MC estimate into an exponentially weighted moving average of past returns

- Decays *geometrically* with visits - the return from k visits ago depends on $\alpha(1-\alpha)^k$), instead of *arithmetically* according to N returns having an equal weight of $\frac{1}{N}$

In general, we *prefer* non-stationary estimators because our policy we will be evaluating is continuously improving

- Essentially we are always in a non-stationary setting in RL as we improve our policy through experience

In summary, in Monte-Carlo learning we

1. Run out episodes,
2. look at the complete returns, and
3. update estimates of the mean value of return at each state of the return.

Temporal-Difference Learning

Temporal-Difference (TD) Learning

TD methods learn directly from episodes of experience

- TD is *model-free*: no knowledge of transitions/rewards (as in MC)

TD learns from *incomplete* episodes, by bootstrapping

- It substitutes, or bootstraps, reminder of the trajectory with the *estimate* of what will happen, instead of waiting for full returns
- i.e. TD updates one guess with a subsequent guess

MC versus 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

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

Simplest temporal-difference learning algorithm: TD(0):

Update value $V(S_t)$ toward *estimated* return $R_{t+1} + \gamma V(S_{t+1})$ (*like Bellman equation*)

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

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

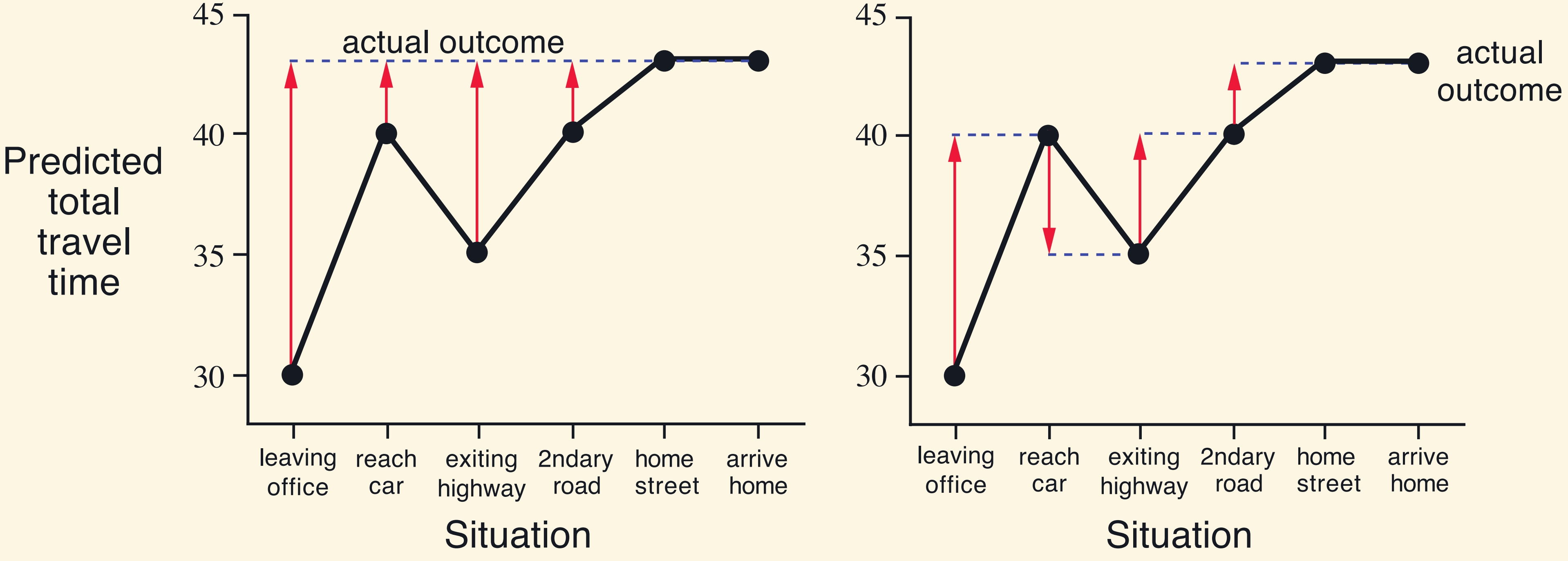
Driving Home Example

State	Elapsed Time (min)	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 versus TD

Changes recommended by Monte Carlo methods ($\alpha = 1$):

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



Red arrow represent recommended updates by MC and TD respectively

Advantages & Disadvantages of MC versus 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

Bias/Variance Trade-Off

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

True TD target $R_{t+1} + \gamma\, v_\pi(S_{t+1})$ is an *unbiased* estimate of $v_\pi(S_t)$.

- TD target $R_{t+1} + \gamma\, V(S_{t+1})$ is a *biased* estimate of $v_\pi(S_t)$.
- TD target has much lower variance than the return, since
  - Return depends on *many* random actions, transitions, rewards.
  - TD target depends on *one* random action, transition, reward.

Advantages & Disadvantages of MC versus TD (2)

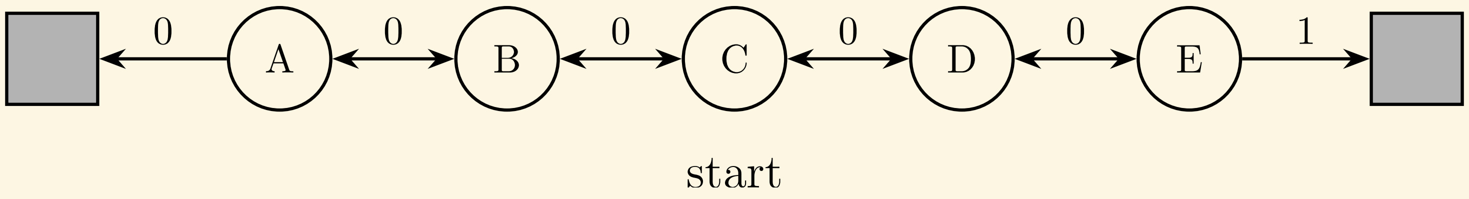
MC has high variance, zero bias

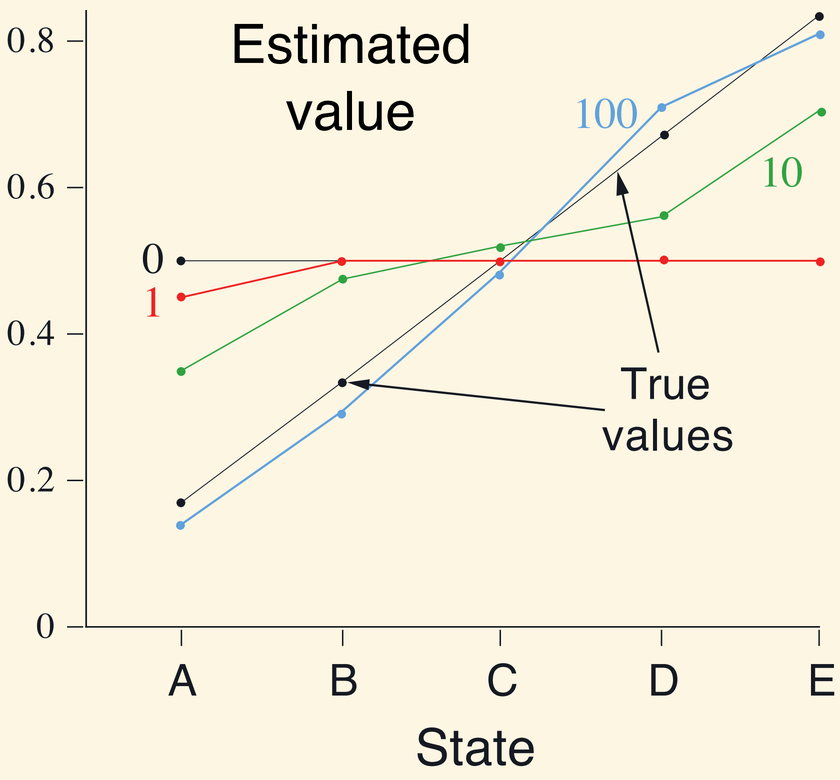
- Good convergence properties (even with function approximation)
- Not very sensitive to initial value
- Very simple to understand and use

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

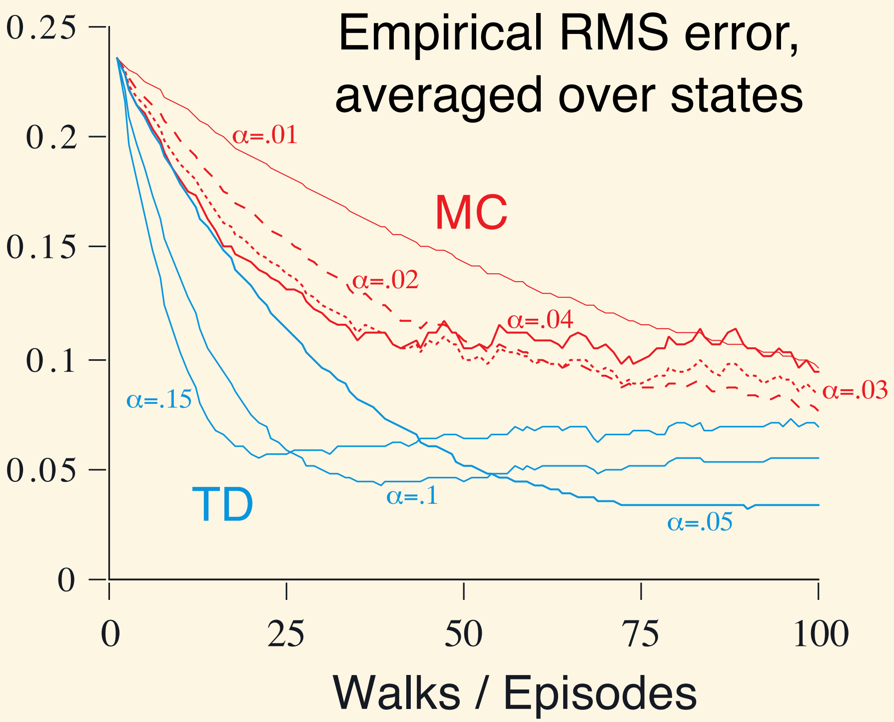
Random Walk Example





Random Walk example based on uniform random policy: left 0.5 and right 0.5

Random Walk: MC versus TD



- This demonstrates the benefit of bootstrapping

Batch MC and TD

MC and TD both converge in the limit

- $V(s) \to v_\pi(s)$ as experience $\to \infty$

What about a *batch* solution for finite experience, k finite episodes?

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

$$\vdots$$

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

In *batch* mode you repeatedly sample episode $k \in [1, K]$ and apply MC or TD(0) to episode k

AB Example

Two states A, B; no discounting; 8 episodes:

$A, 0, B, 0$

$B, 1$

$B, 1$

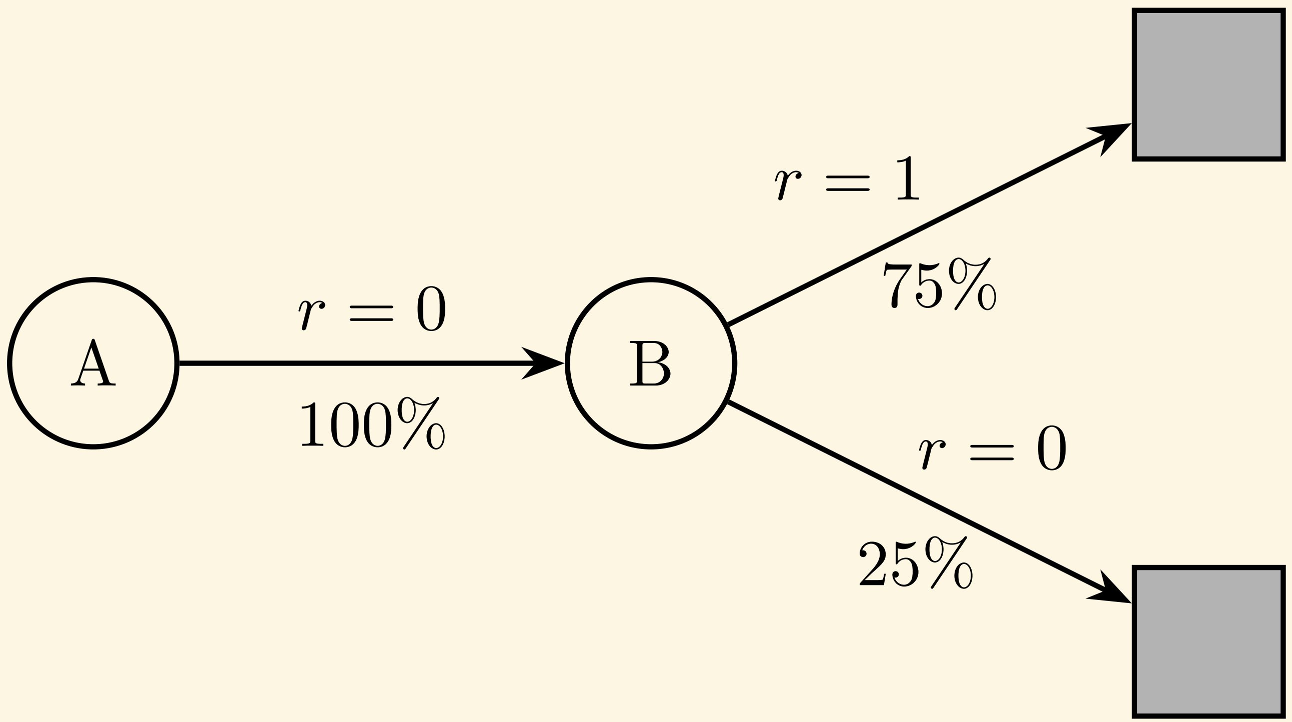
$B, 1$

$B, 1$

$B, 1$

$B, 1$

$B, 0$



Clearly, $V(B) = \frac{6}{8} = 0.75$, but what about $V(A)$?

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 that best explains the data

- Solution to the MDP $\langle \mathcal{S}, \mathcal{A}, \hat{\mathcal{P}}, \hat{\mathcal{R}}, \gamma \rangle$ that best fits the data ($\hat{\mathcal{P}}$ counts the transitions, and $\hat{\mathcal{R}}$ the rewards)

$$\hat{\mathcal{P}}^a_{s,s'} = \frac{1}{N(s,a)} \sum_{k=1}^{K} \sum_{t=1}^{T_k} \mathbf{1}(s^k_t, a^k_t, s^k_{t+1} = s, a, s')$$

$$\hat{\mathcal{R}}^a_s = \frac{1}{N(s,a)} \sum_{k=1}^{K} \sum_{t=1}^{T_k} \mathbf{1}(s^k_t, a^k_t = s, a) r^k_t$$

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

Advantages and Disadvantages of MC versus TD (3)

TD exploits Markov property

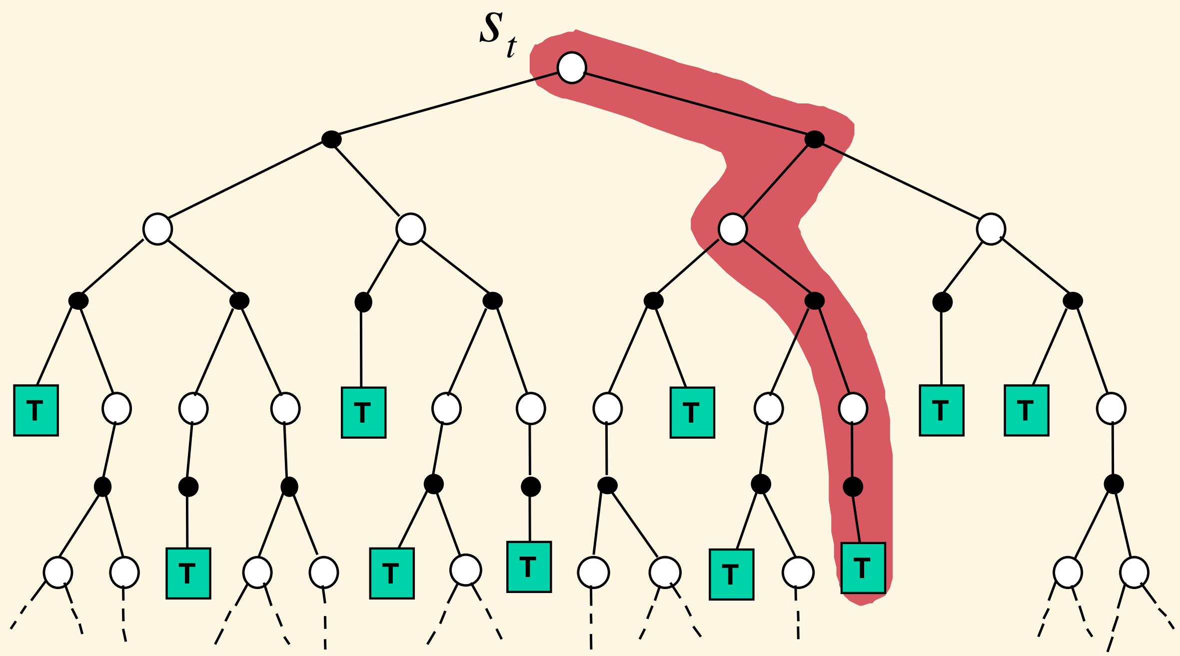
- Usually more efficient in Markov environments

MC does not exploit Markov property

- Usually more effective in non-Markov environments
- Note that *partial observability* and *non-stationarity* are reasons an environment can be non-Markov

Monte-Carlo Backup

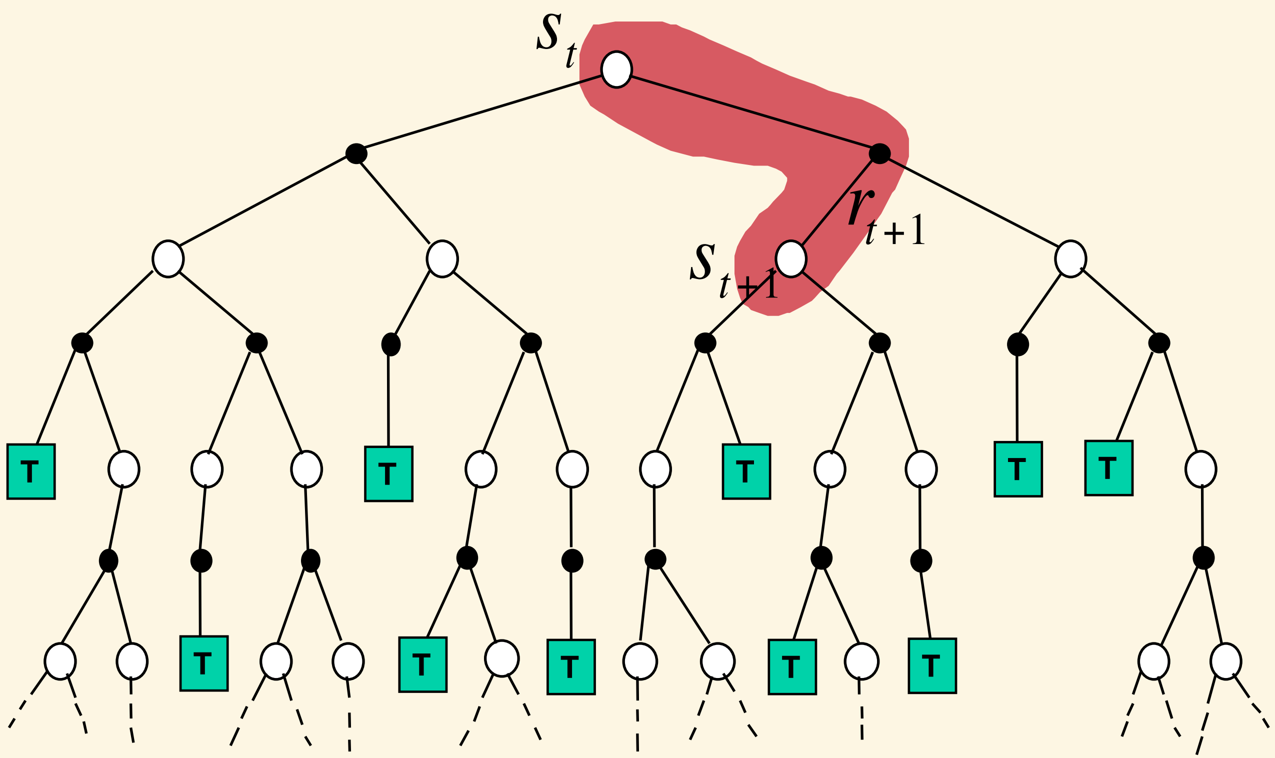
$$V(S_t) \leftarrow V(S_t) + \alpha\big(G_t - V(S_t)\big)$$



Starting at one state, sample one complete trajectory to update the value function

Temporal-Difference Backup

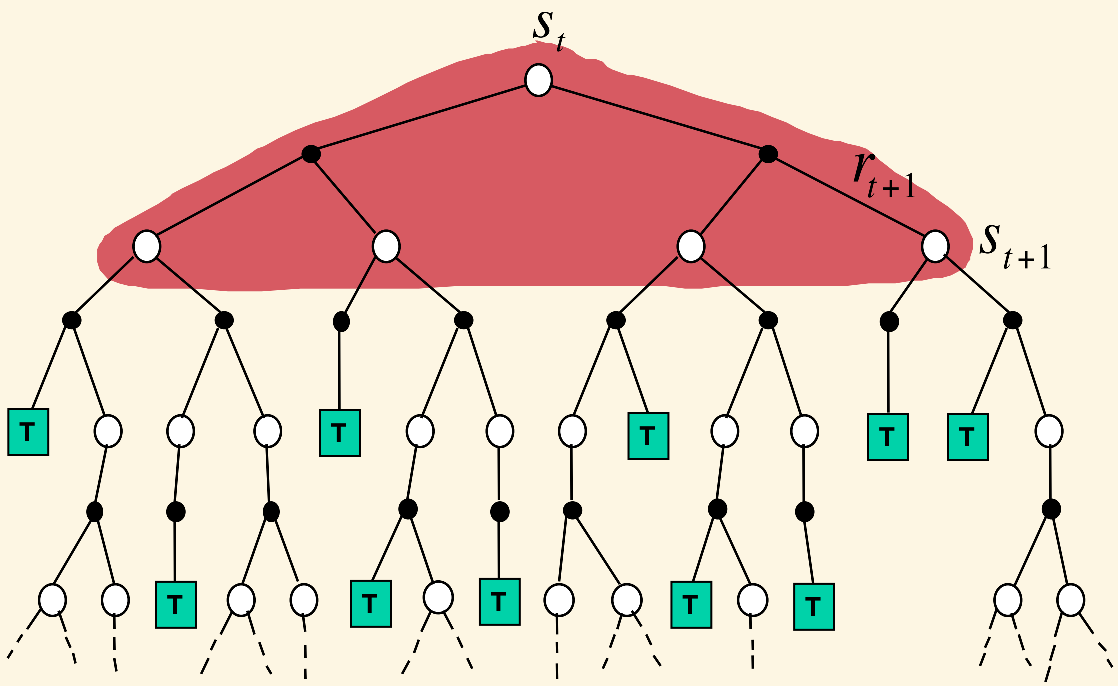
$$V(S_t) \leftarrow V(S_t) + \alpha\big(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\big)$$



In TD backup is just over one step

Tree Search/Dynamic Programming Backup

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



If we know the dynamics of the environment, we can do search and a complete backup over the tree

Bootstrapping and Sampling

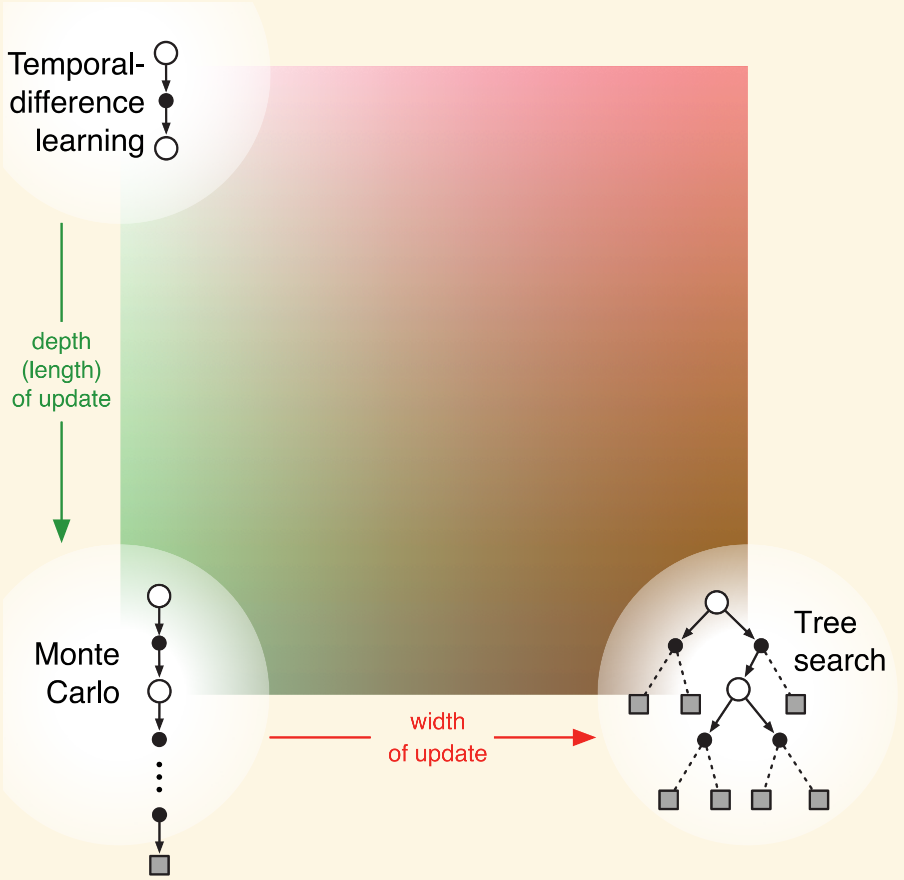
Bootstrapping: update involves an *estimate*

- MC does not bootstrap
- Tree Search (with heuristic search) or dynamic programming bootstraps
- TD bootstraps

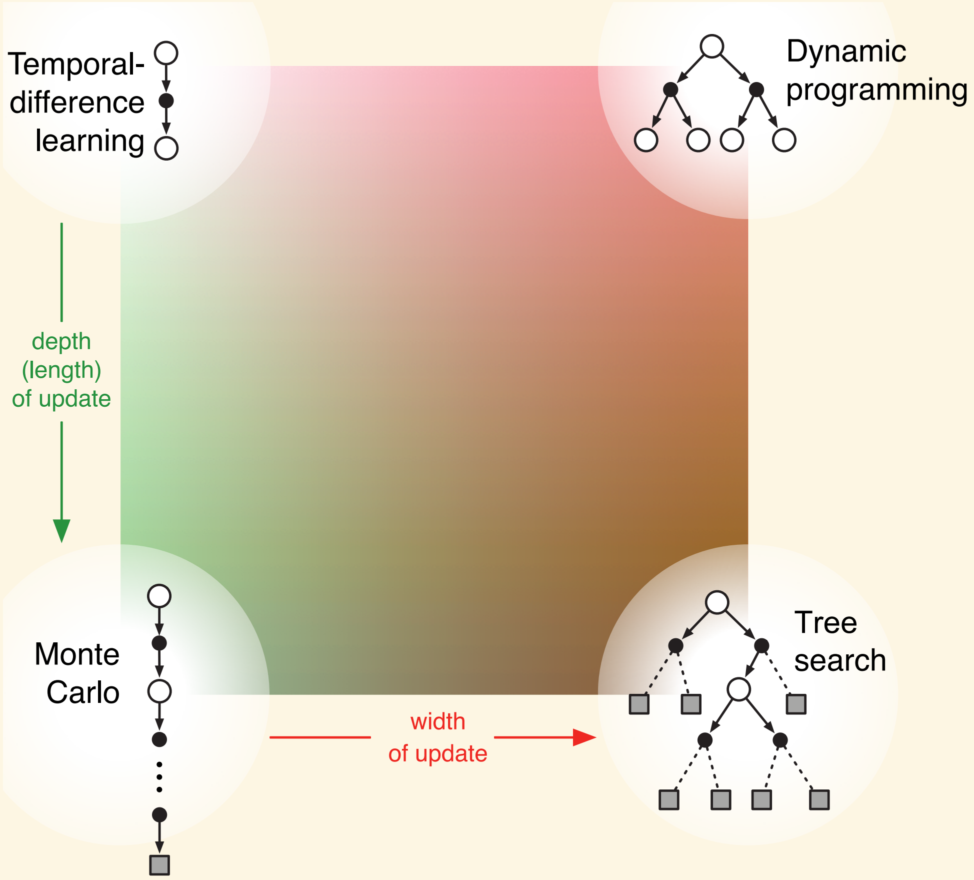
Sampling: update samples an *expectation*

- MC samples
- Tree Search does not sample
- TD samples

Unified View of Reinforcement Learning (1)



Unified View of Reinforcement Learning (2)

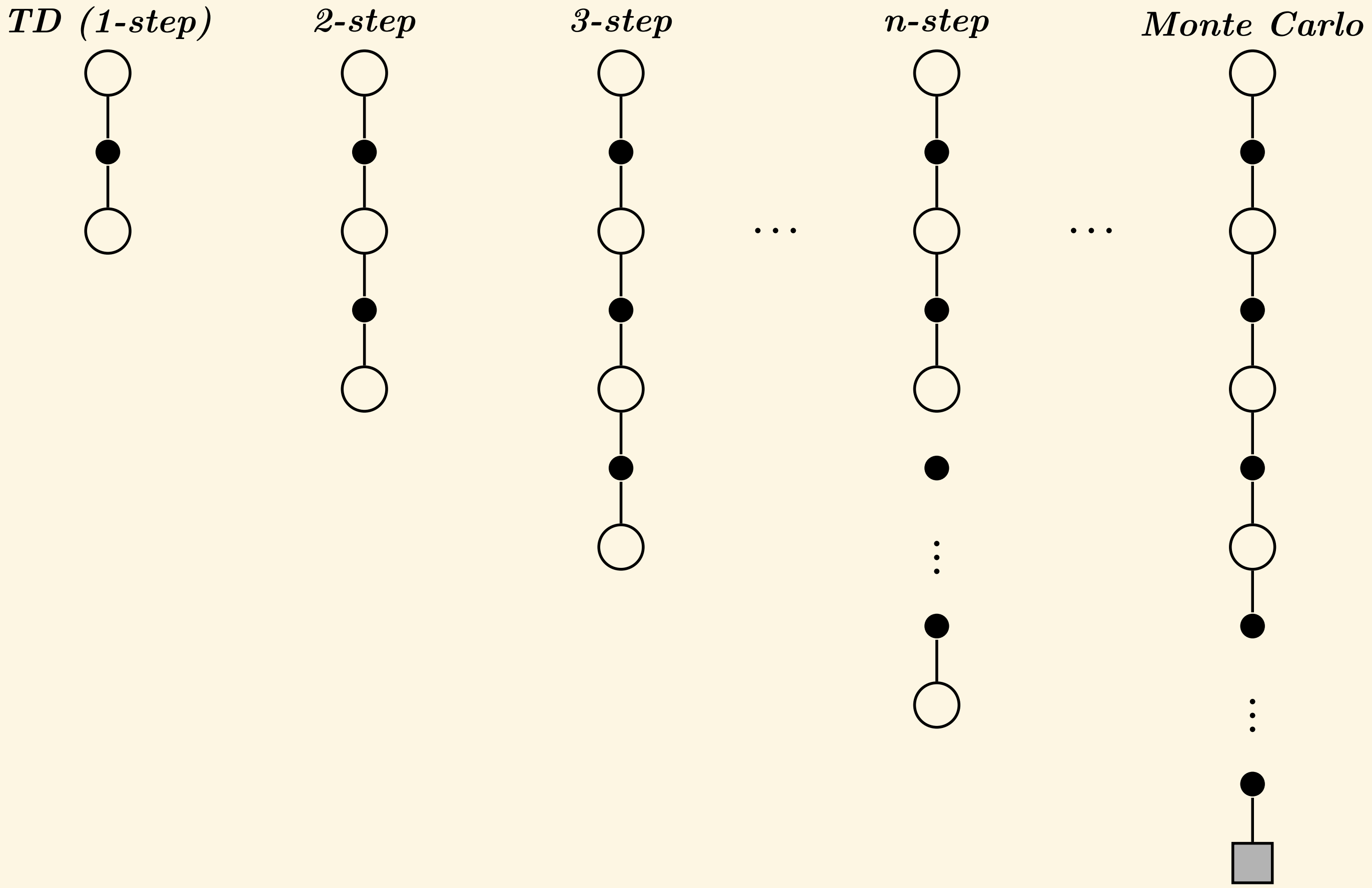


- Dynamic programming only explores one level, in it’s most simple form.
- In practice, dynamic programming is used during tree search, similar to classical planning.

TD(λ)

n-Step Prediction

Let TD target look n steps into the future



n-Step Return

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

$$
\begin{aligned}
n = 1 \text{ (TD(0))} \qquad & G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1}) \\
n = 2 \qquad & G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2}) \\
\vdots \qquad & \qquad \vdots \\
n = \infty \text{ (MC)} \qquad & G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T
\end{aligned}
$$

Define the n-step Return (real reward + estimated reward, $\gamma^n V(S_{t+n})$):

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

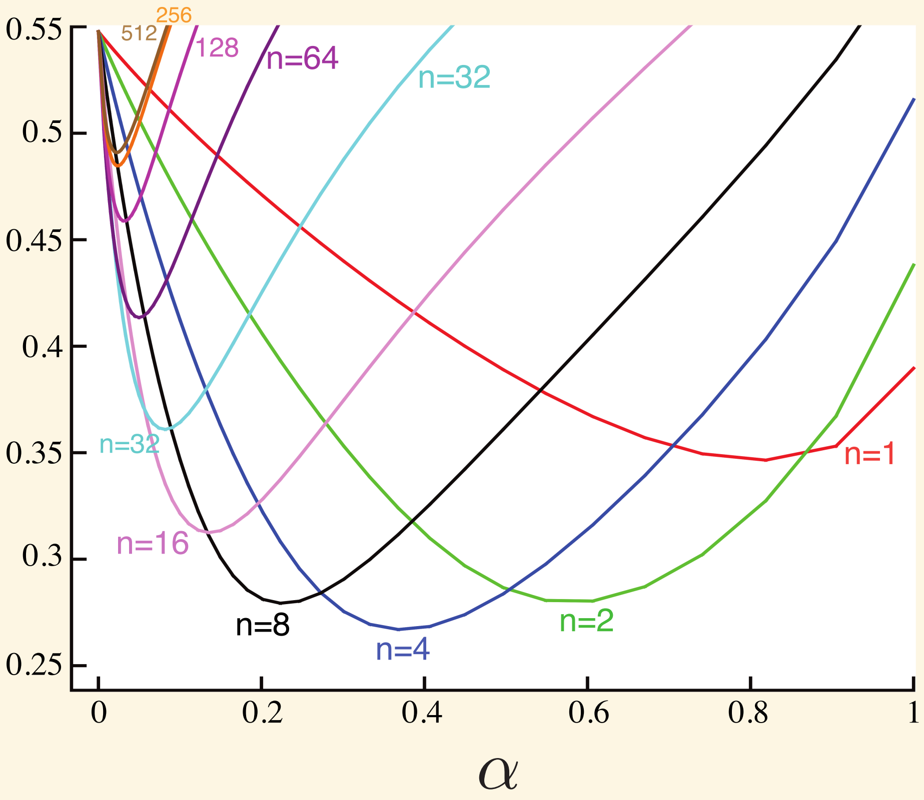
n-step temporal-difference learning (updated in direction of error)

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

- MC looks at the real reward
- So which n is the best?

Large Random Walk Example

Average
RMS error
over 19 states
and first 10
episodes



- RMS errors vary according to step size α, with the *optimum* dependent on n

Note that RMS errors also vary according to whether learning is *on-line* or *off-line* updates (not shown here)

- i.e. whether *immediately* update value function or *defer* updates until episode ends

Averaging n-Step Returns

We can form *mixtures* of different n:

e.g. average of 2-step and 4-step returns:

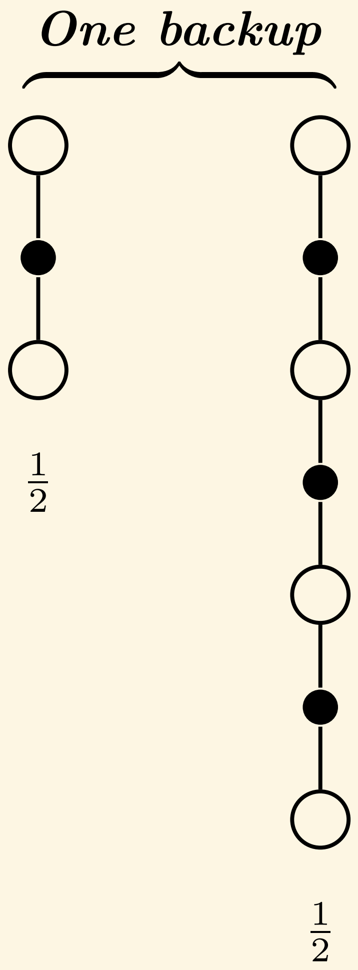
$$\frac{1}{2}G_t^{(2)} + \frac{1}{2}G_t^{(4)}$$

We can average n-step returns over different n

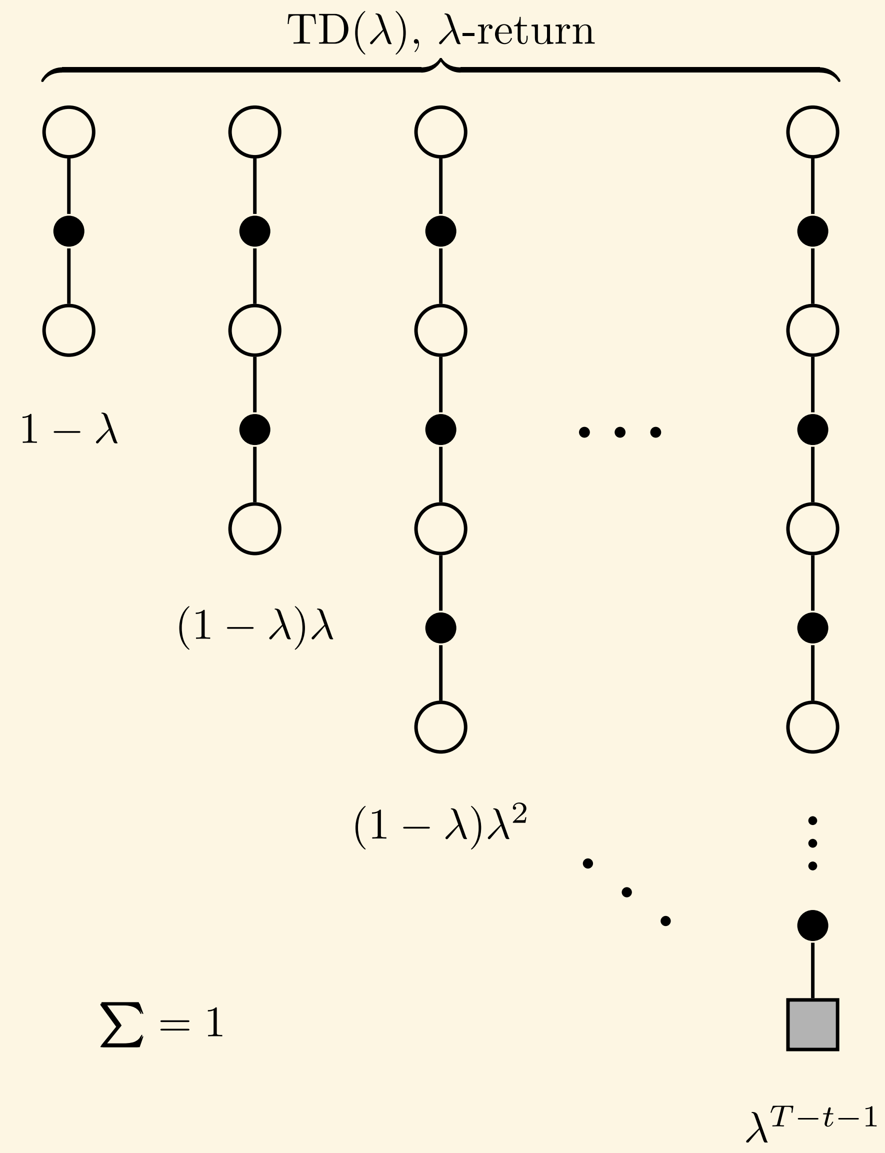
- e.g. average the 2-step and 4-step returns

Combines information from two different time-steps

Can we efficiently combine information from *all* time-steps to be more robust?



λ-return



The λ-return G_t^{λ} combines *all* n-step returns $G_t^{(n)}$

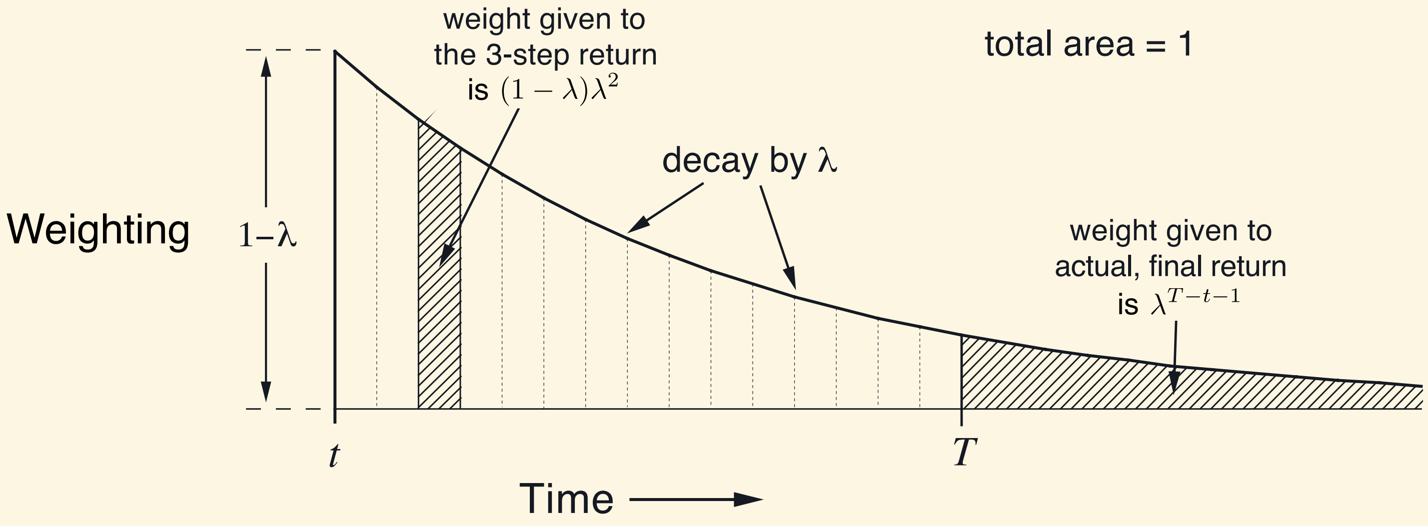
- Using *weight* $(1-\lambda)\lambda^{n-1}$

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

Forward-view TD(λ)

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

TD(λ) Weighting Function



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

- λ-return is a geometrically weighted return for every n-step return

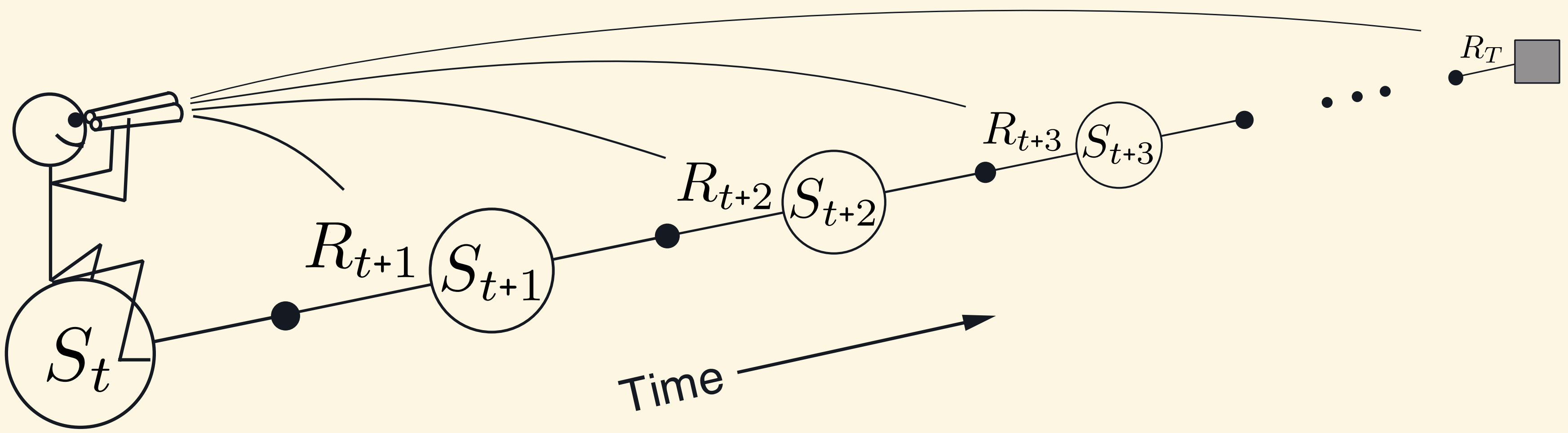
Note that geometric weightings are *memoryless*

- i.e. we can compute $TD(\lambda)$ with with no greater complexity than $TD(0)$

However, the formulation presented so far is a *forward view*

- we can't look into the future!

Forward View of TD(λ)

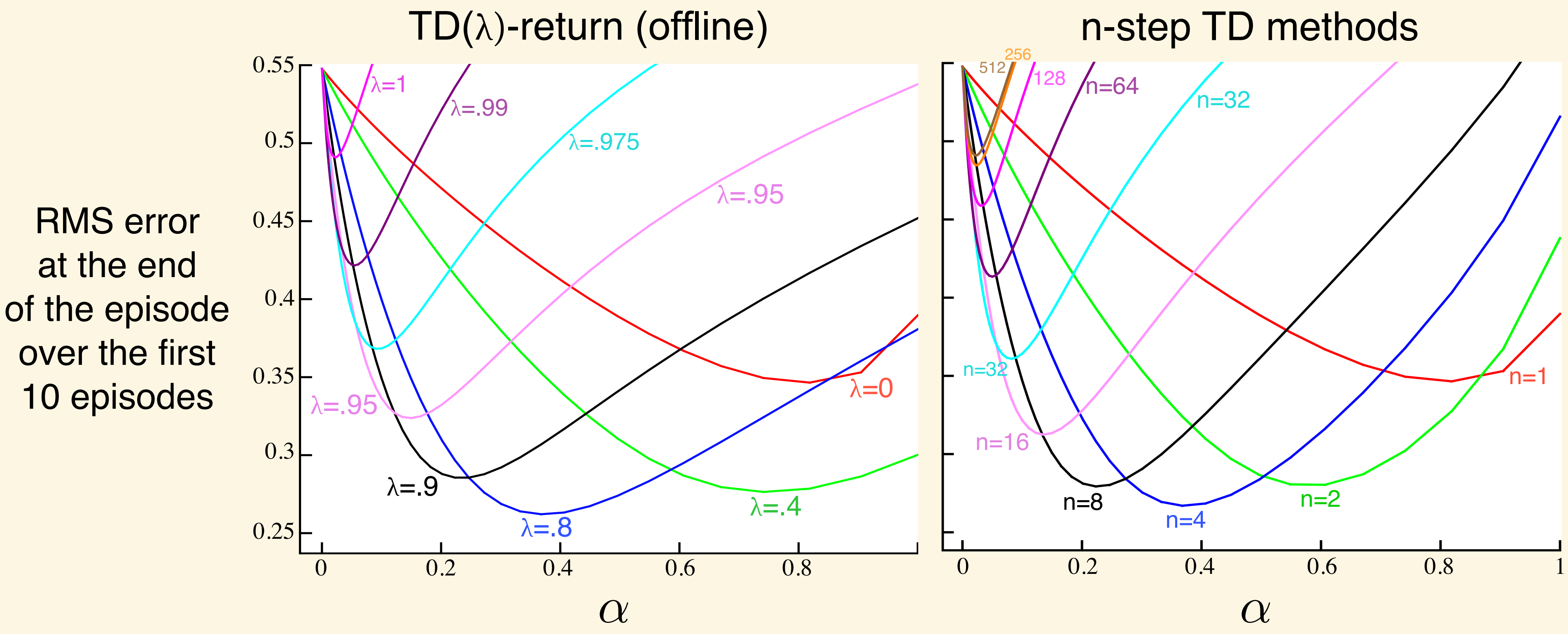


Update value function towards the λ-return

- Forward-view looks into the future to compute G_t^λ
- Like MC, can only be computed from complete episodes

We will see shortly how an *iterative* algorithm achieves the forward view without having to wait until the future

Forward View of TD(λ) on Large Random Walk



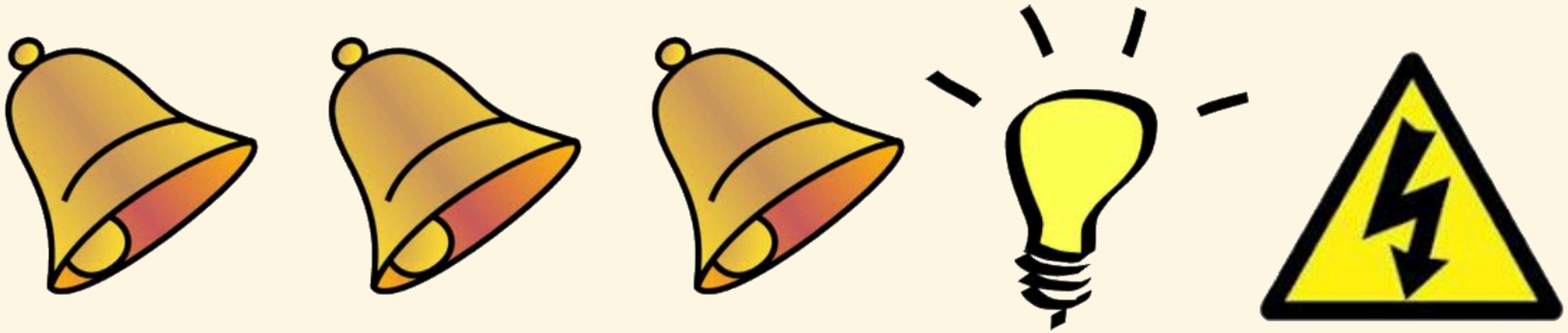
We can see using TD(λ), and choosing λ value (left hand side), is more robust than choosing a *unique* n-step value (right hand side)

- $\lambda = 1$ is MC and $\lambda = 0$ is TD(0)

Backward View of TD(λ)

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

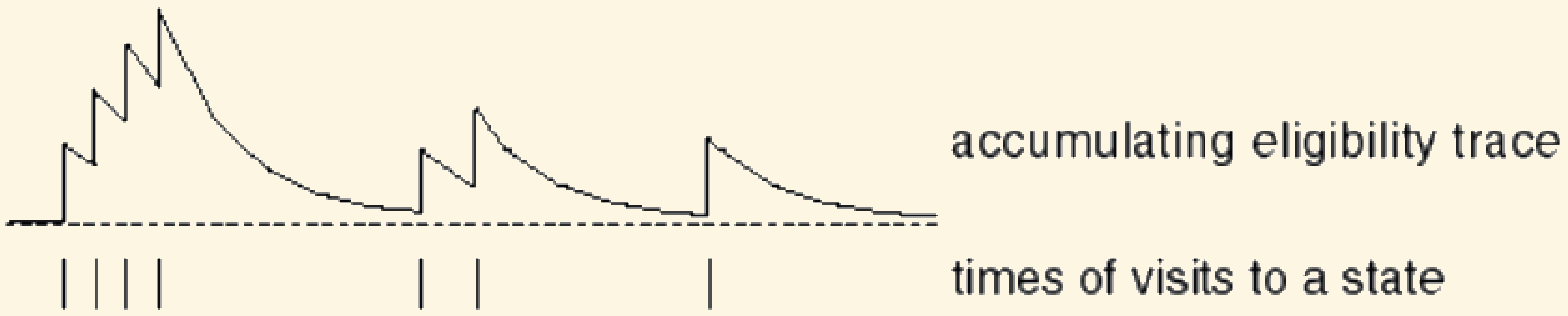
Eligibility Traces



Credit assignment problem: *did bell or light cause shock?*

- Frequency heuristic: assign credit to most frequent states
- Regency heuristic: assign credit to most recent states
- Eligibility traces *combine* both heuristics

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

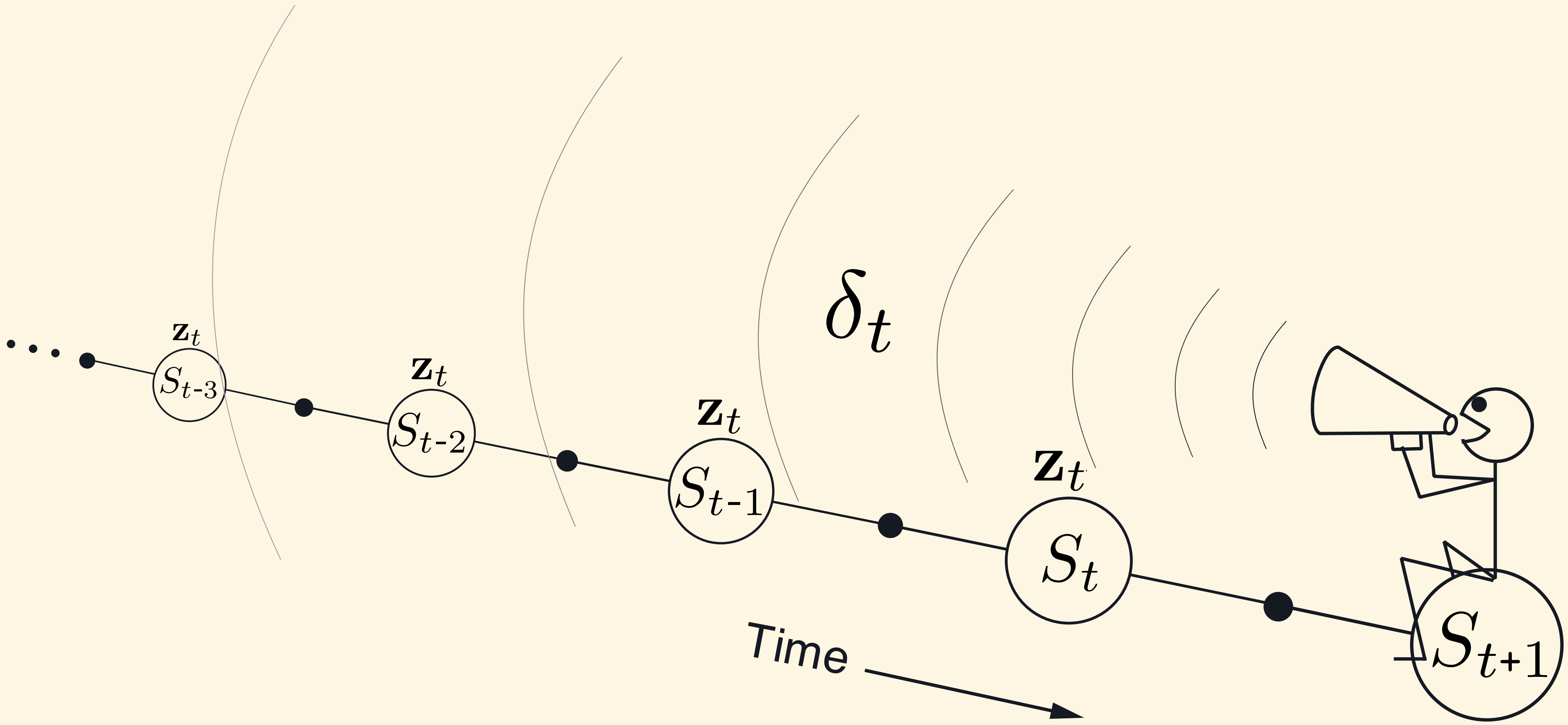


Backward View TD(λ)

- Keep an eligibility trace for every state s
- Update value $V(s)$ for every state s
- In proportion to TD-error δ_t and eligibility trace $E_t(s)$

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

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



δ_t
$\mathbf{z}_t$
$\mathbf{z}_t$
$\mathbf{z}_t$
$\mathbf{z}_t$
$S_{t\text{-}3}$
$S_{t\text{-}2}$
$S_{t\text{-}1}$
S_t
S_{t+1}
Time

TD(λ) and TD(0)

When $\lambda = 0$, only the 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 the TD(0) update.

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

TD(λ) and MC

When $\lambda = 1$, credit is deferred until the end of the episode.

- Consider episodic environments with offline updates.
- Over the course of an episode, the total update for TD(λ) is the same as the total update for MC.

Theorem

The sum of offline updates is identical for forward-view and backward-view TD(λ):

$$\sum_{t=1}^{T} \alpha\, \delta_t\, E_t(s) \;=\; \sum_{t=1}^{T} \alpha\big(G_t^{\lambda} - V(S_t)\big)\, \mathbf{1}(S_t = s).$$

Example: Temporal-Difference Search for MCTS

Example: Temporal-Difference Search for MCTS

Simulation-based search

. . . using TD instead of MC (bootstrapping)

- MC tree search applies MC control to sub-MDP from now
- TD search applies Sarsa to sub-MDP from now

MC versus TD search

For model-free reinforcement learning, bootstrapping is helpful

- TD learning reduces variance but increases bias
- TD learning is usually more efficient than MC
- TD(λ) can be much more efficient than MC

For simulation-based search, bootstrapping is also helpful

- TD search reduces variance but increases bias
- TD search is usually more efficient than MC search
- TD(λ) search can be much more efficient than MC search

TD search

Simulate episodes from the current (real) state s_t

Estimate action-value function $Q(s, a)$

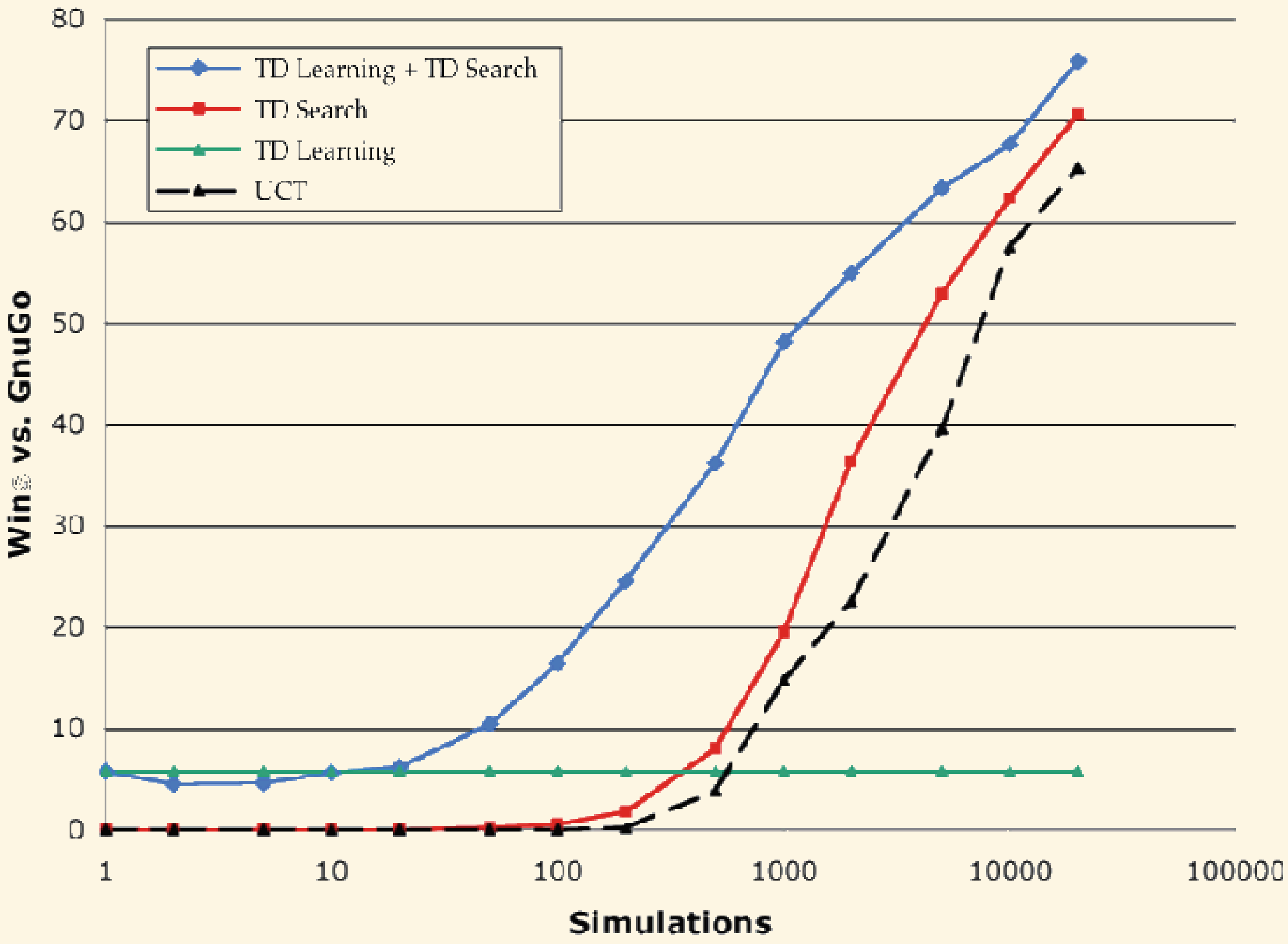
- For each step of simulation, update action-values by Sarsa

$$\Delta Q(S, A) = \alpha\big(R + \gamma Q(S', A') - Q(S, A)\big)$$

- Select actions based on action-values $Q(s, a)$
  - e.g. ϵ-greedy

May also use function approximation for Q, if needed

Results of TD search in Go



- Black dashed line is MCTS
- Blu line is Dyna-Q (not covered in this module)

Learning from simulation is an effective method in search