# Reinforcement Learning - Lecture 4

**Profa. Dra. Esther Luna Colombini**
esther@ic.unicamp.br

**Prof. Dr. Alexandre Simoes**
alexandre.simoes@unesp.br

**LaRoCS – Laboratory of Robotics and Cognitive Systems**

# Summary



- Model-free Prediction
  - Introduction
  - Monte-Carlo Learning
  - Temporal-Difference Learning
  - TD(λ)

# Disclaimer



- These slides were built upon David Silver's Lecture notes on Reinforcement Learning

- Planning by dynamic programming
  - Solve a known MDP
- **Model-free prediction**
  - Estimate the value function of an unknown MDP
- **Model-free control**
  - Optimize the value function of an unknown MDP

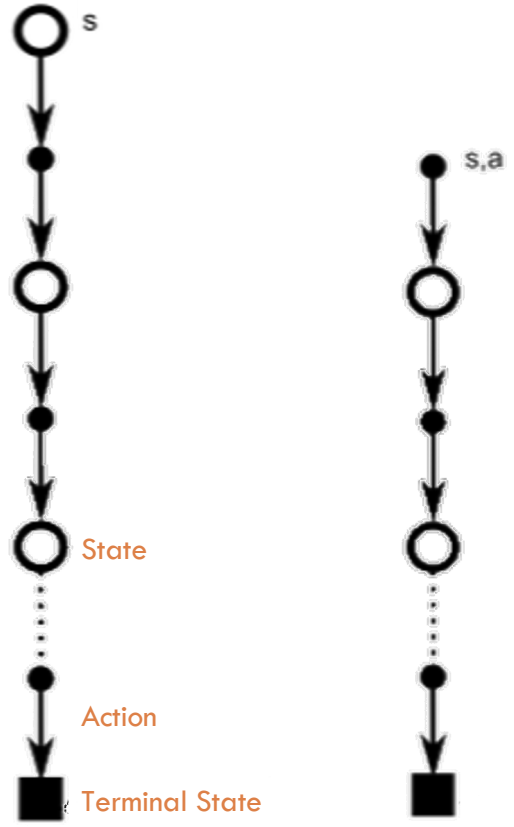
# Model-free prediction

# Tow main methods: MC and TD


Source: https://baijayantaroy.github.io/baijayantaroy.github.io/

# Monte-Carlo Reinforcement Learning

- **Monte Carlo methods**, or **Monte Carlo experiments**, are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results.
- MC methods features:
  - learns directly from episodes of experience
  - model-free
    - no knowledge of MDP transitions/rewards
  - learns from **complete episodes**
    - no bootstrapping
  - uses the simplest possible idea
    - value = mean return
- Caveat
  - Can only apply MC to episodic MDPs
    - All episodes must terminate

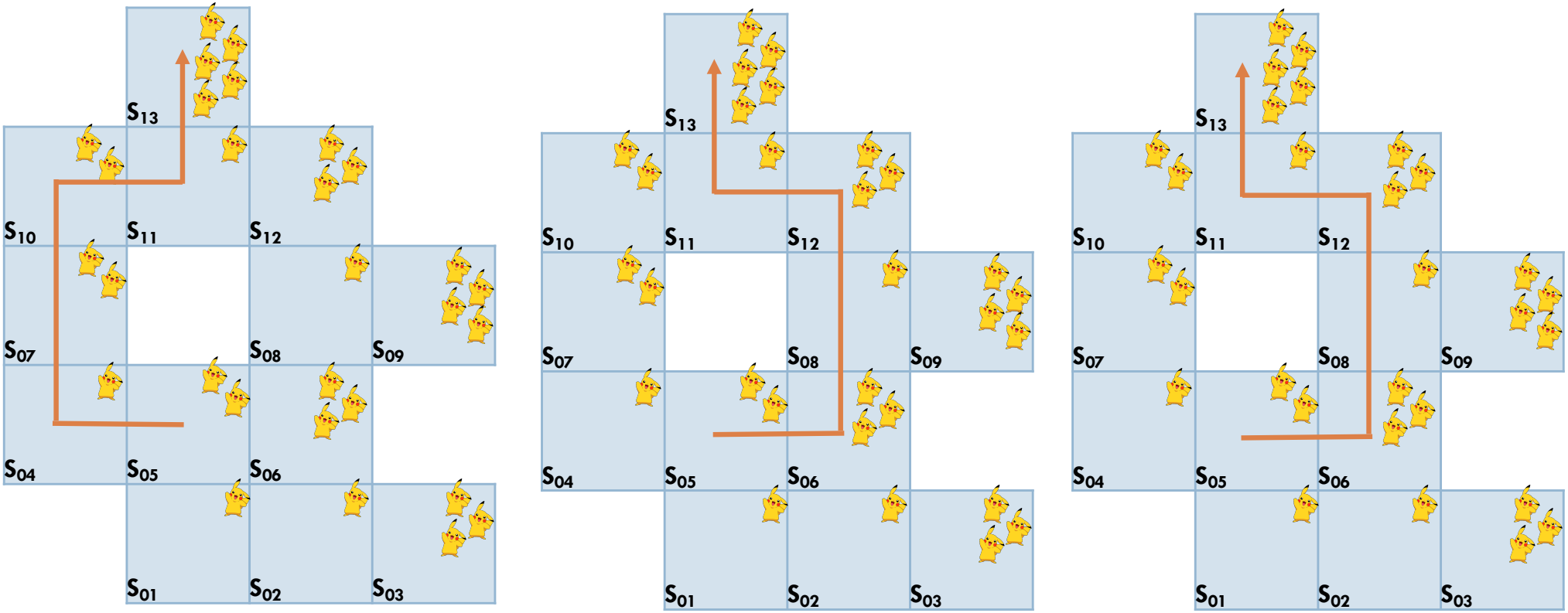



MC State Value $v_\pi(s)$ MC Action Value $q_\pi(s,a)$

# Monte-Carlo Reinforcement Learning



3 Samples from State S05




- Return(Sample 1) = 2 + 1 + 2 + 2 + 1 + 5 = 13 pokemons
- Return(Sample 2) = 2 + 3 + 1 + 3 + 1 + 5 = 15 pokemons
- Return(Sample 3) = 2 + 3 + 1 + 3 + 1 + 5 = 15 pokemons
- Observed mean return (based on 3 samples) = (13 + 15 + 15)/3 = 14.33 pokemons
- Thus, state value as per Monte Carlo Method, $v_\pi\ (S_{05})$ is 14.33 pokemons based on 3 samples following policy $\pi$.

# Monte-Carlo Policy Evaluation

- Goal: learn $v_\pi$ from episodes of experience under policy $\pi$

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

- Recall that the return is the total discounted reward:

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

- Recall that the value function is the expected return:

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

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

# 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 Monte-Carlo Policy Evaluation



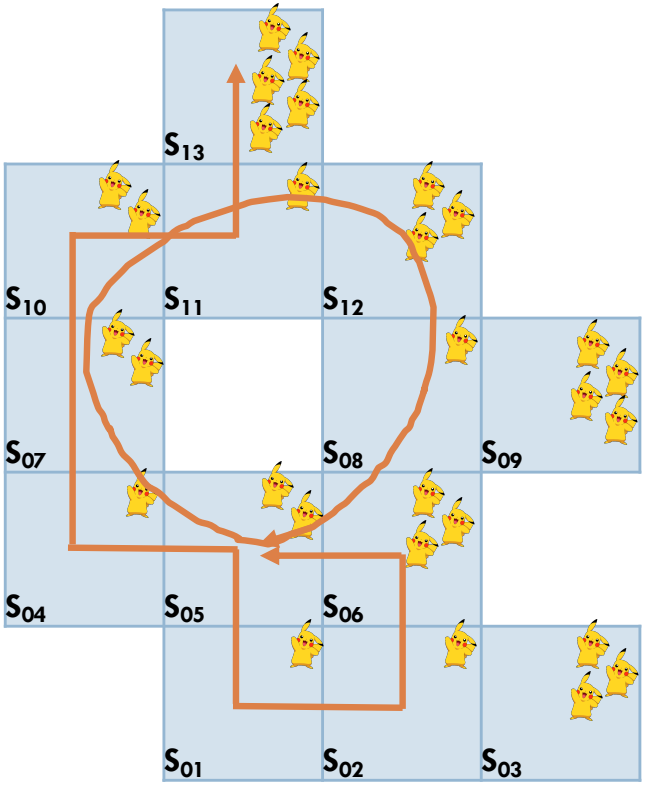
- Initialize
  - $\pi \leftarrow$ policy to be evaluated
  - $V \leftarrow$ an arbitrary state-value function
  - $Returns(s) \leftarrow$an empty list, for all $s \in S$
- Repeat forever:
  - Generate episode using $\pi$
  - For each state $s$ appearing in the episode
    - $R \leftarrow$ return following the first occurrence of $s$
    - Append $R$ to $Returns(s)$
    - $V(s) \leftarrow$average$(Returns(s))$

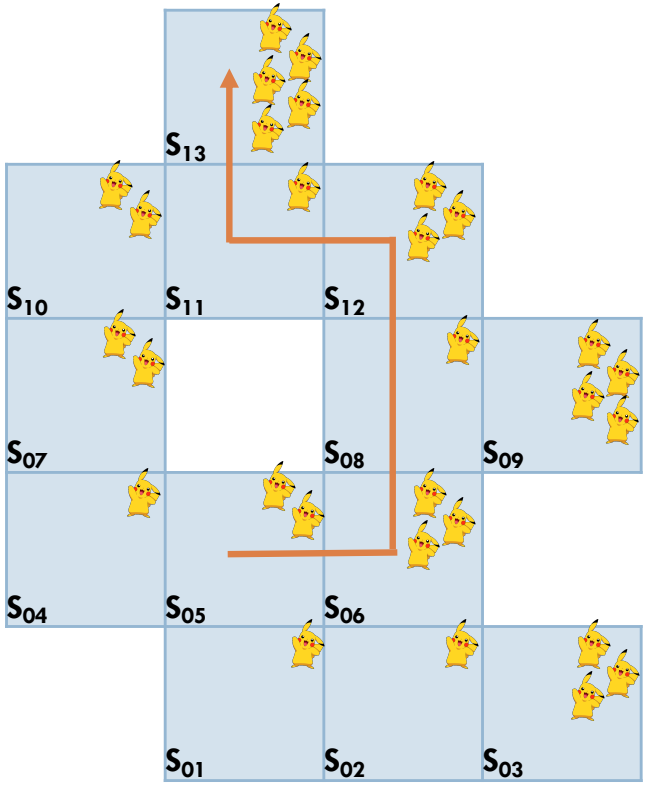
R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction

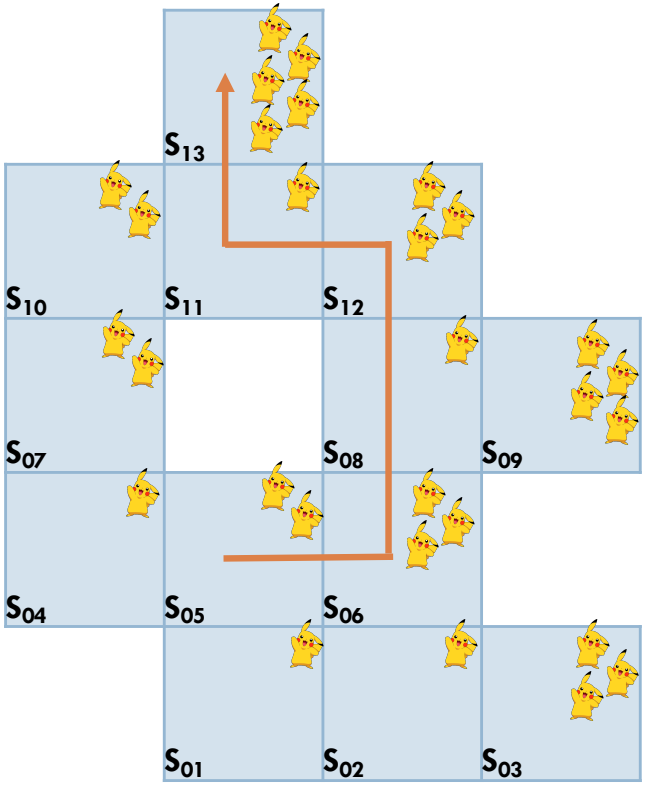
# First-Visit Monte-Carlo Policy Evaluation



3 Samples from State S05




- Return(Sample 1) = (2 + 1 + 2 + 2 + 1 + 3 + 1 + 3) + (2 + 1 + 1 + 3) + (2 + 1 + 2 + 2 + 1 + 5) = 35 pokemons (only first visit is counted) $N(S_{05}) = 1$
- Return(Sample 2) = 2 + 3 + 1 + 3 + 1 + 5 = 15 pokemons $N(S_{05}) = 2$
- Return(Sample 3) = 2 + 3 + 1 + 3 + 1 + 5 = 15 pokemons $N(S_{05}) = 3$
- $v\ \ (S_{05})$ is (35+15+15)/3 = 21.67 pokemons

# Every-Visit Monte-Carlo Policy Evaluation

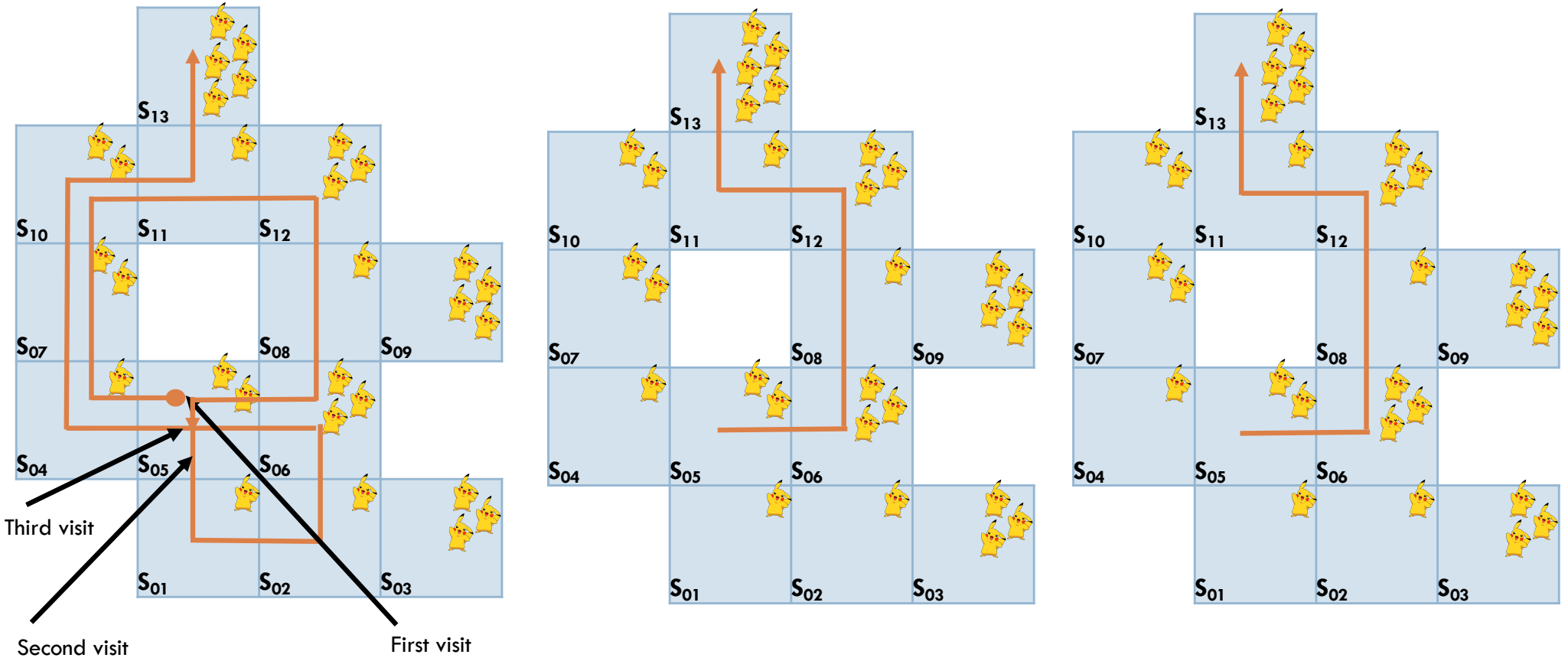


- To evaluate state $s$
- Every time-step $t$ that state $s$ is visited in an episode,
- Increment counter $N(s) \leftarrow N(s)+1$
- Increment total return $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return $V(s) = S(s)/\ N(s)$
- Again, $V(s) \rightarrow v_\pi\ (s) =$ as $N(s) \rightarrow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

# Every-Visit Monte-Carlo Policy Evaluation



3 Samples from State S05




- Return(Sample 1) = 35 pokemons (all visits are counted) $N(S_{05}) = 3$
- Return(Sample 2) = 2 + 3 + 1 + 3 + 1 + 5 = 15 pokemons $N(S_{05}) = 4$
- Return(Sample 3) = 2 + 3 + 1 + 3 + 1 + 5 = 15 pokemons $N(S_{05}) = 5$
- $v\ \ (S_{05})$ is (35+15+15)/5 = 13 pokemons

# Blackjack Example - Rules



- *The game begins with two cards dealt to both dealer and player.*
- *One of the dealer's cards is face up and the other is face down.*
- *If the player has 21 immediately (an ace and a 10-card), it is called a natural. He then wins unless the dealer also has a natural, in which case the game is a draw.*
- *If the player does not have a natural, then he can request additional cards, one by one (hits), until he either stops (sticks) or exceeds 21 (goes bust). If he goes bust, he loses; if he sticks, then it becomes the dealer's turn.* ***The dealer hits or sticks according to a fixed strategy without choice: he sticks on any sum of 17 or greater, and hits otherwise.*** *If the dealer goes bust, then the player wins; otherwise, the outcome — win, lose, or draw — is determined by whose final sum is closer to 21.*
- *If the player holds an ace that he could count as 11 without going bust, then the ace is said to be usable.*

https://towardsdatascience.com/reinforcement-learning-solving-blackjack-5e31a7fb371f

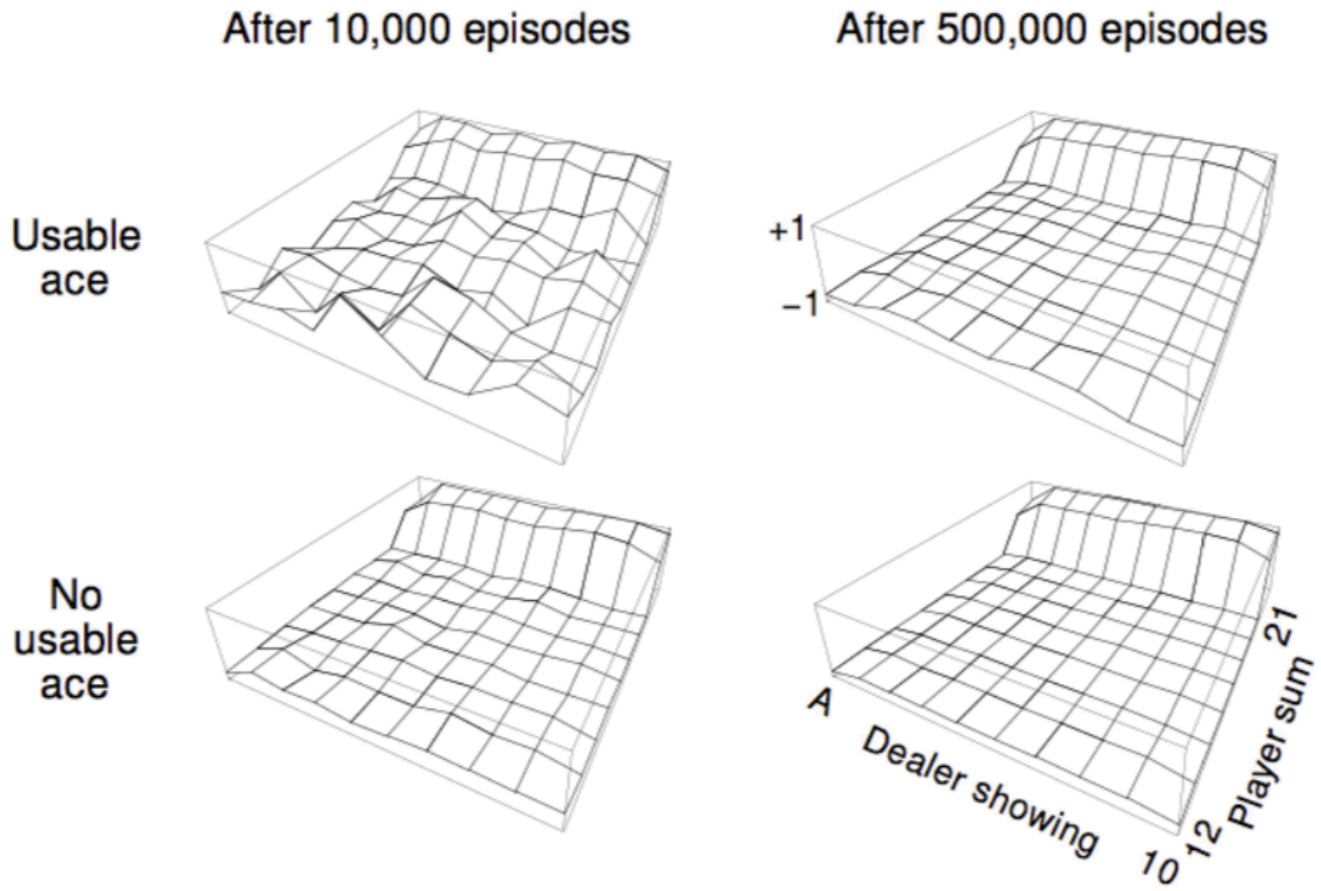
# Blackjack Example



- States (200 of them):
  - Current sum (12-21)
  - Dealer's showing card (ace-10)
  - Do I have a useable (takes value 1 or 11) 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






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

# Incremental Mean

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

$$\mu_k = \frac{1}{k}\sum_{j=1}^{k} x_j$$

$$= \frac{1}{k}\left(x_k + \sum_{j=1}^{k-1} x_j\right)$$

$$= \frac{1}{k}\left(x_k + (\mathrm{k}-1)\mu_{k-1}\right)$$

$$= \mu_{k-1} + \frac{1}{k}\left(x_k - \mu_{k-1}\right)$$

# Incremental Monte-Carlo Updates



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

Value computed by running the episode

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

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

- **Be careful, this is not performed until an episode is over**
- In non-stationary problems, it can be useful to track a running mean, i.e. forget old episodes (moving window)

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

# Monte-Carlo: Summary



- Advantages
  - Zero bias
  - Good convergence properties (even with function approximation)
  - Not very sensitive to initial value
  - Very simple to understand and use
- Limitations
  - MC must wait until end of episode before return is known
  - MC has high variance
  - MC can only learn from complete sequences
  - MC only works for episodic (terminating) environments

# Temporal-Difference Learning

- TD methods features:
  - learn directly from episodes of experience
  - model-free:
    - no knowledge of MDP transitions/rewards
  - learns from incomplete episodes
    - by **bootstrapping**
      - substitute part of our trajectory for a guess
  - updates a guess towards a guess

- Goal: learn $v_\pi$ online from experience under policy $\pi$
- Incremental every-visit Monte-Carlo
  - Update value $V(S_t)$ toward actual return $\boldsymbol{G_t}$

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

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

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

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

# Driving Home Example

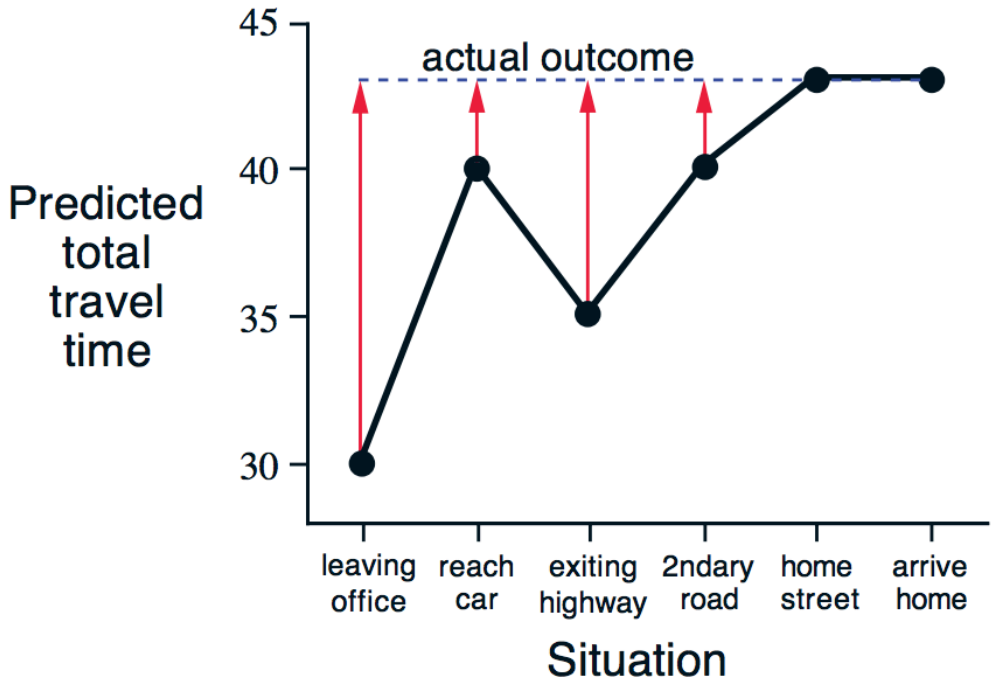


| State | Elapsed Time (minutes) | Predicted Time to Go | Predicted Total Time |
|---|---|---|---|
| leaving office | 0 | 30 | 30 |
| reach car, raining | 5 | 35 | 40 |
| exit highway | 20 | 15 | 35 |
| behind truck | 30 | 10 | 40 |
| home street | 40 | 3 | 43 |
| arrive home | 43 | 0 | 43 |

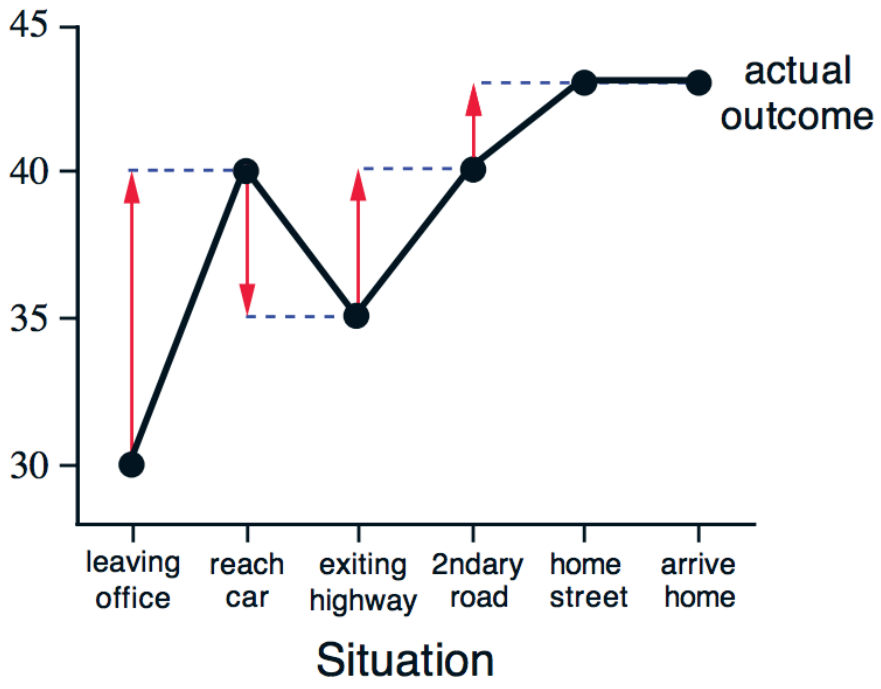
# Driving Home Example: MC vs. TD



Changes recommended by MC methods ($\alpha = 1$)

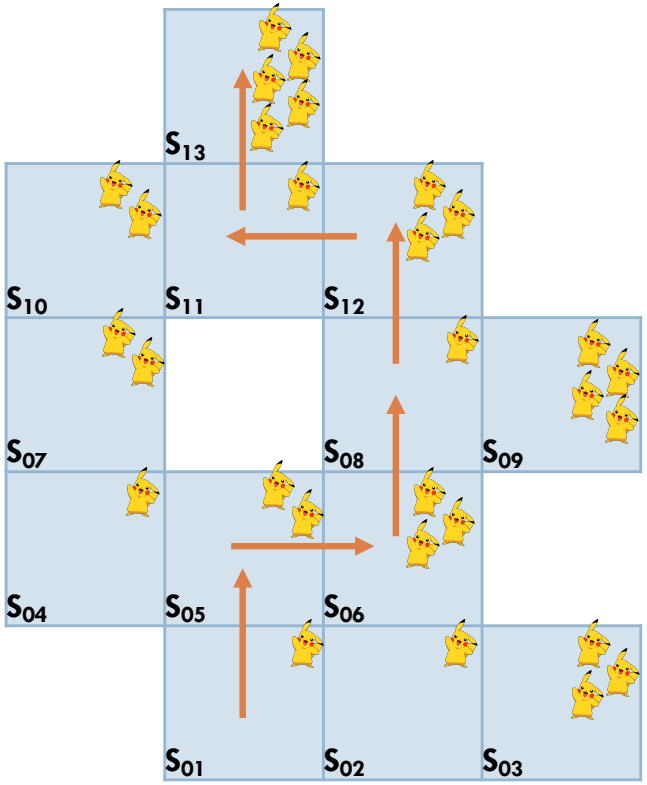


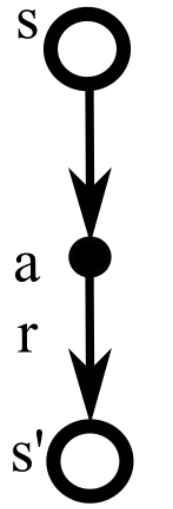

Changes recommended by TD methods ($\alpha = 1$)



# TD(0) in Pokemon






Backup

# Tabular TD(0) for estimating $v_\pi$



**Algorithm: Tabular TD(0) for estimating $v_\pi$**

Input: the policy $\pi$ to be evaluated

Algorithm parameter: step size $\alpha \in (0, 1]$

*Initialize $V(s)$*, for all $s \in S^+$, arbitrarily, except that $V(terminal)=0$

***repeat (for each episode)***

    Initialize $s$

    **repeat** (for each step of episode)

        $A \leftarrow$ action given by $\pi$ for $S$

        Take action $A$, observe $R$, $S'$

        $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$

        $S \leftarrow S'$

    **until** $s$ is terminal

# 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

# Bias/Variance Trade-Off



- Return $G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$ is unbiased estimate of $v_\pi(S_t)$
- **True** (given by an oracle) TD target $R_{t+1} + \gamma v_\pi(S_{t+1})$ is unbiased estimate of $v_\pi(S_t)$

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

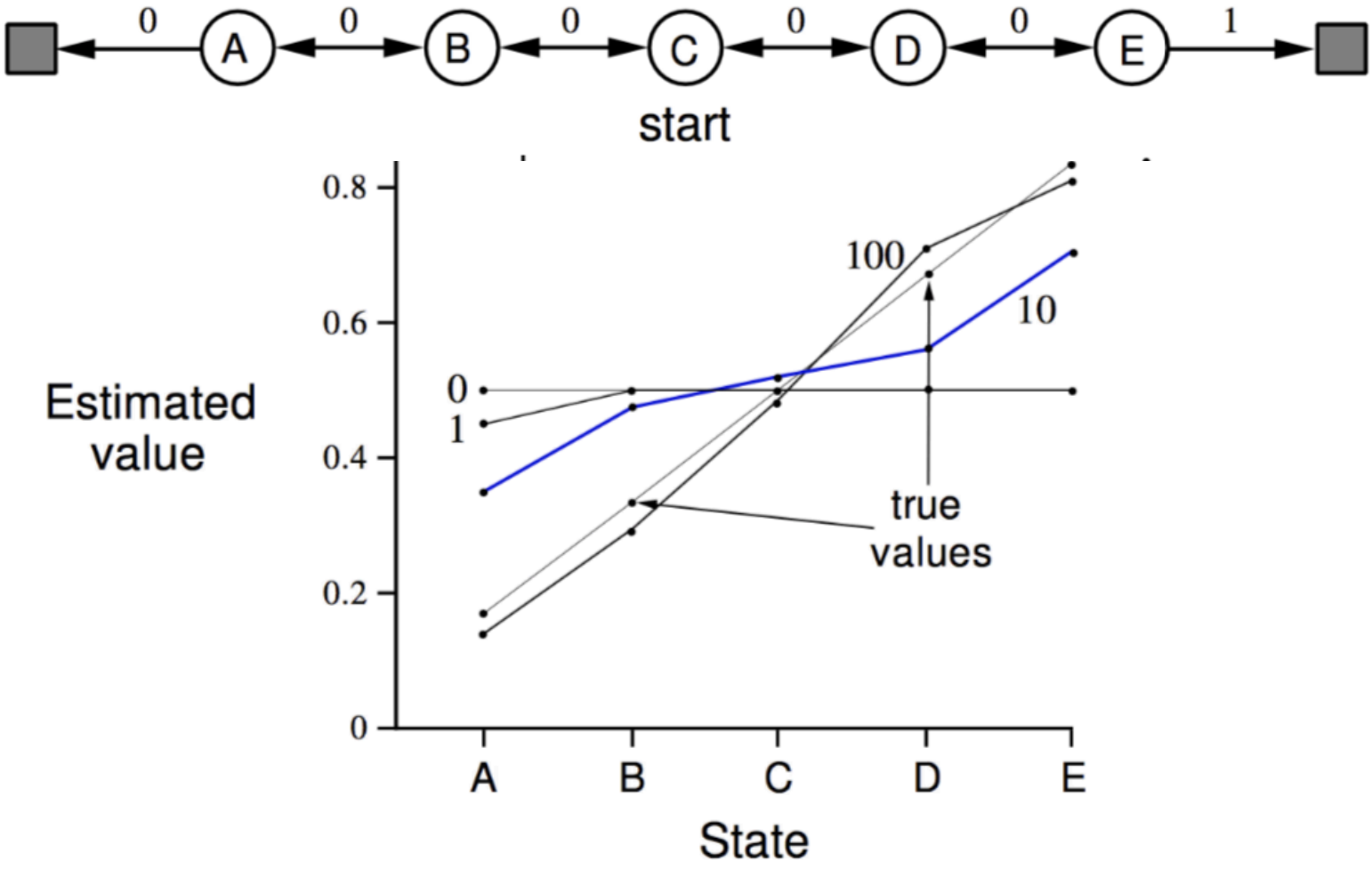
# Bias/Variance Trade-Off



- 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

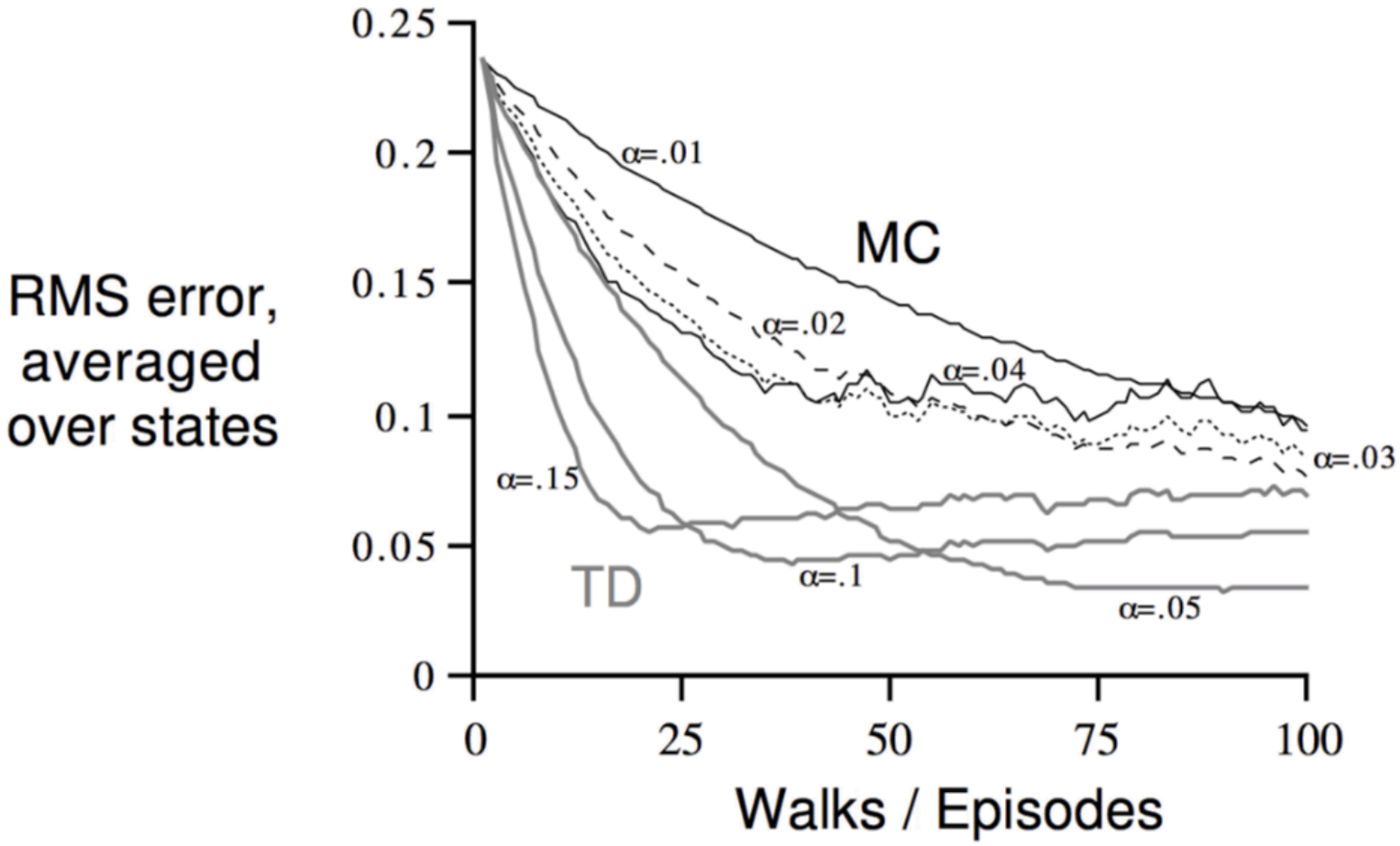
# Random Walk Example

- **Actions:** left and right with same probability



# Random Walk: MC vs. TD

- Step size do matter



# Batch MC and TD

- MC and TD converge: $V(s) \leftarrow v_\pi(s)$ as experience $\rightarrow \infty$

- 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 $k$

# AB Example

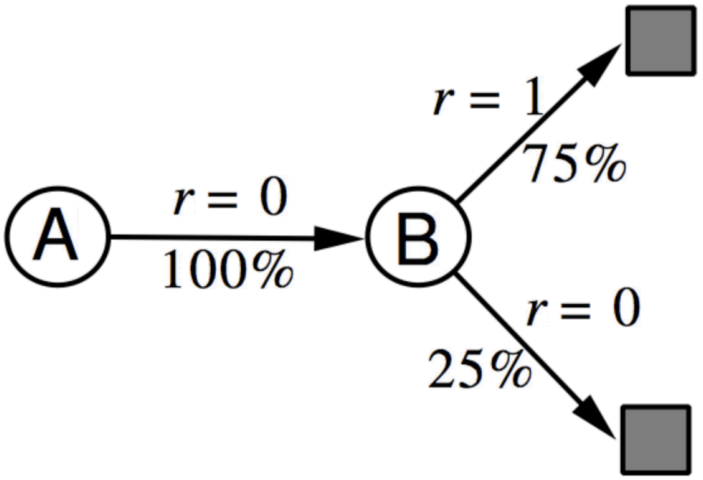


- Two states A,B with $\gamma = 1$
- Given 8 episodes of experience:
  - A, 0, B, 0
  - B, 1 (observed 6 times)
  - B, 0
- Imagine run TD updates over data infinite number of times
- $V(A)$ and $V(B)$? It depends if it is MC our TD
  - $V(B) = 0.75$ by TD or MC (first visit or every visit)
  - What about $V(A)$?

# AB Example



- Two states A,B with $\gamma = 1$
- Given 8 episodes of experience:
  - A, 0, B, 0
  - B, 1 (observed 6 times)
  - B, 0




- Imagine run TD updates over data infinite number of times
- $V(A)$ and $V(B)$? It depends if it is MC our TD
  - $V(B) = 0.75$ by TD or MC (first visit or every visit)
  - What about $V(A)$?
    - $V^{MC}(A) = 0$
    - $V^{TD}(A) = 0.75$
    - Let's check why

# Certainty Equivalence

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

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

  - In the AB example, $V^{MC}(A) = 0$

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

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

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

  - In the AB example, $V^{TD}(A) = 0.75$

# Advantages and Disadvantages of MC vs. TD



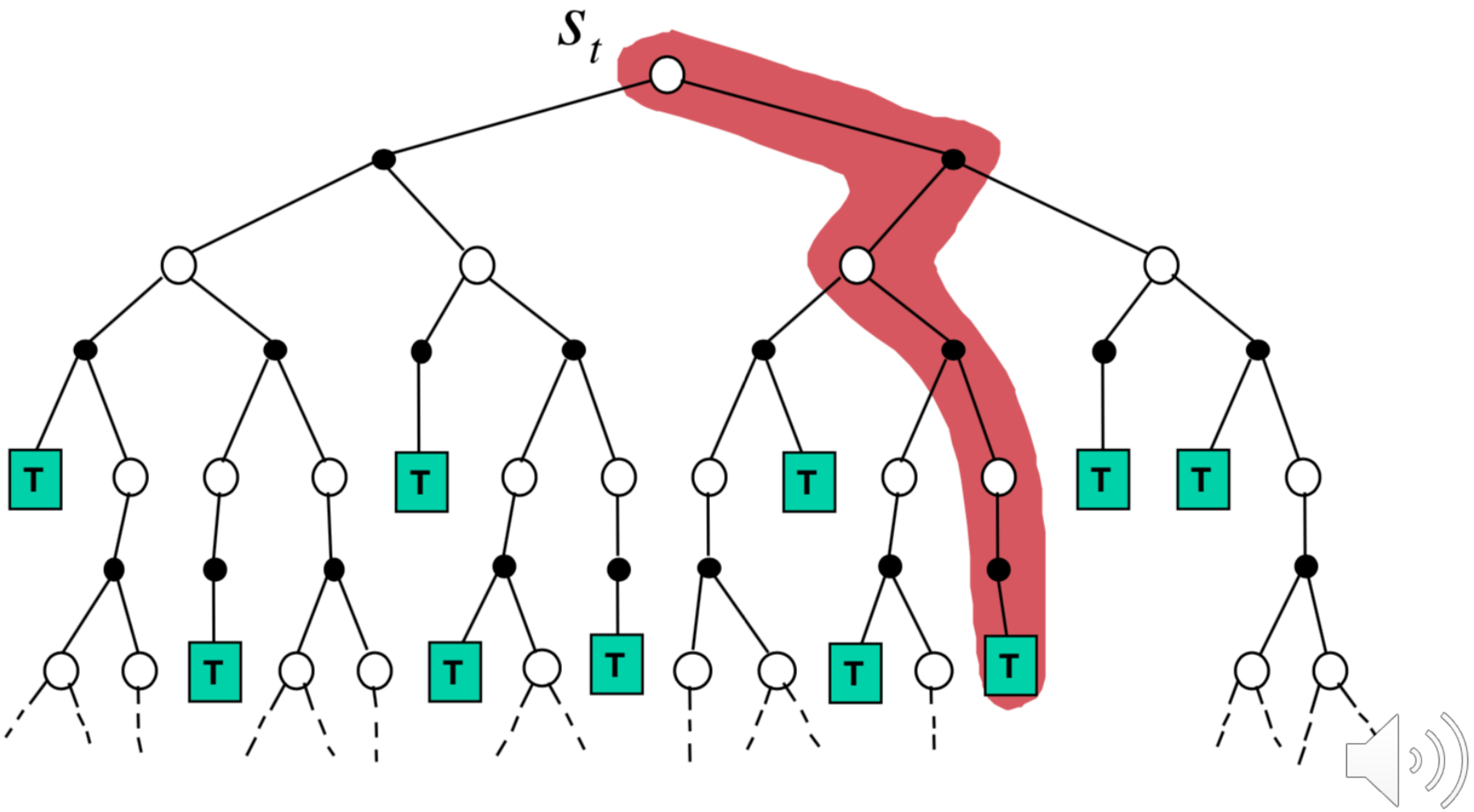
- TD exploits Markov property
  - Usually more efficient in Markov environments

- MC does not exploit Markov property
  - Usually more effective in non-Markov environments

# Monte-Carlo Backup

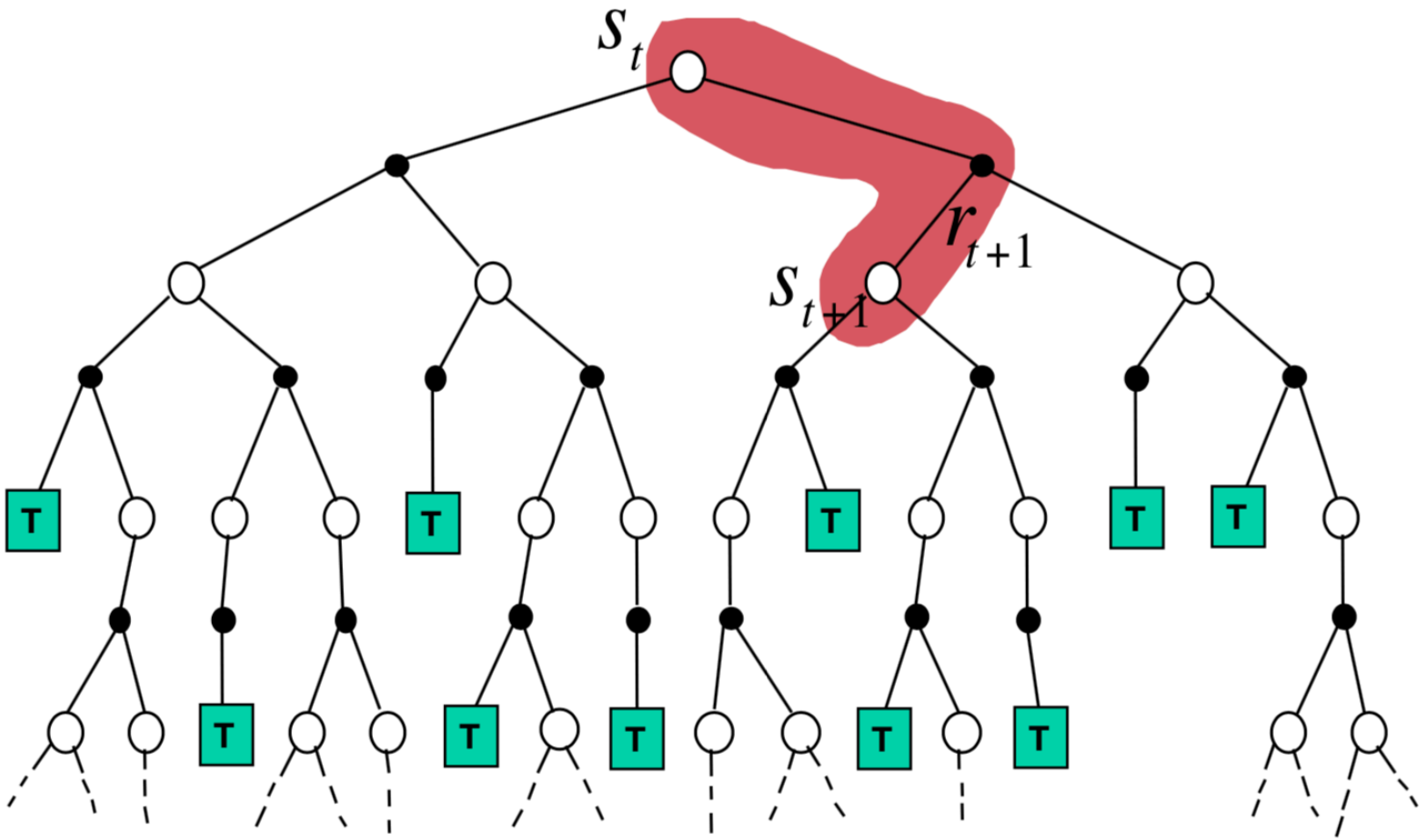


$$V(S_t) \leftarrow V(S_t) + \alpha\,(G_t - V(S_t))$$



# TD Backup

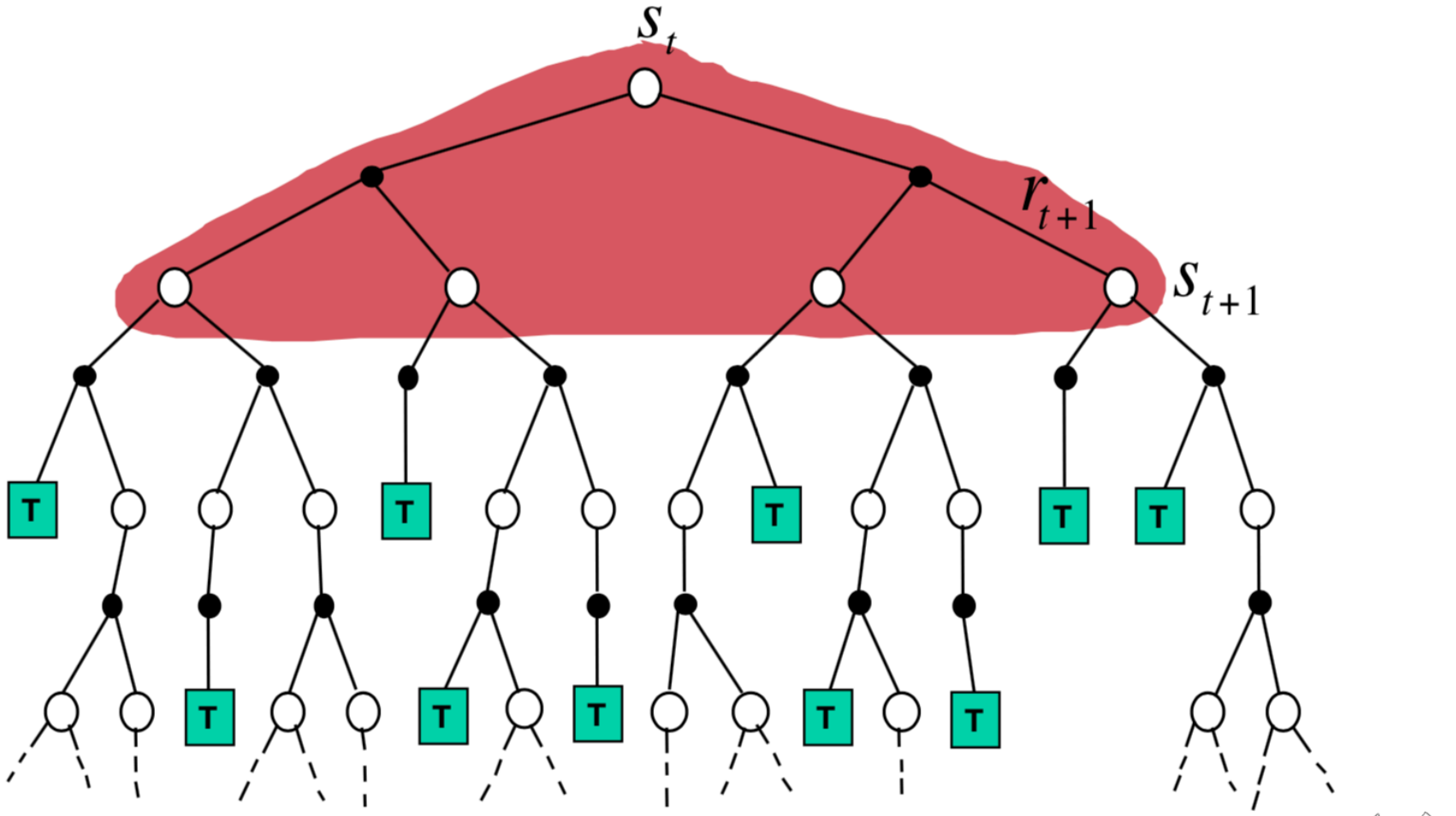
$$V(S_t) \leftarrow V(S_t) + \alpha \left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right)$$



# Dynamic Programming Backup



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



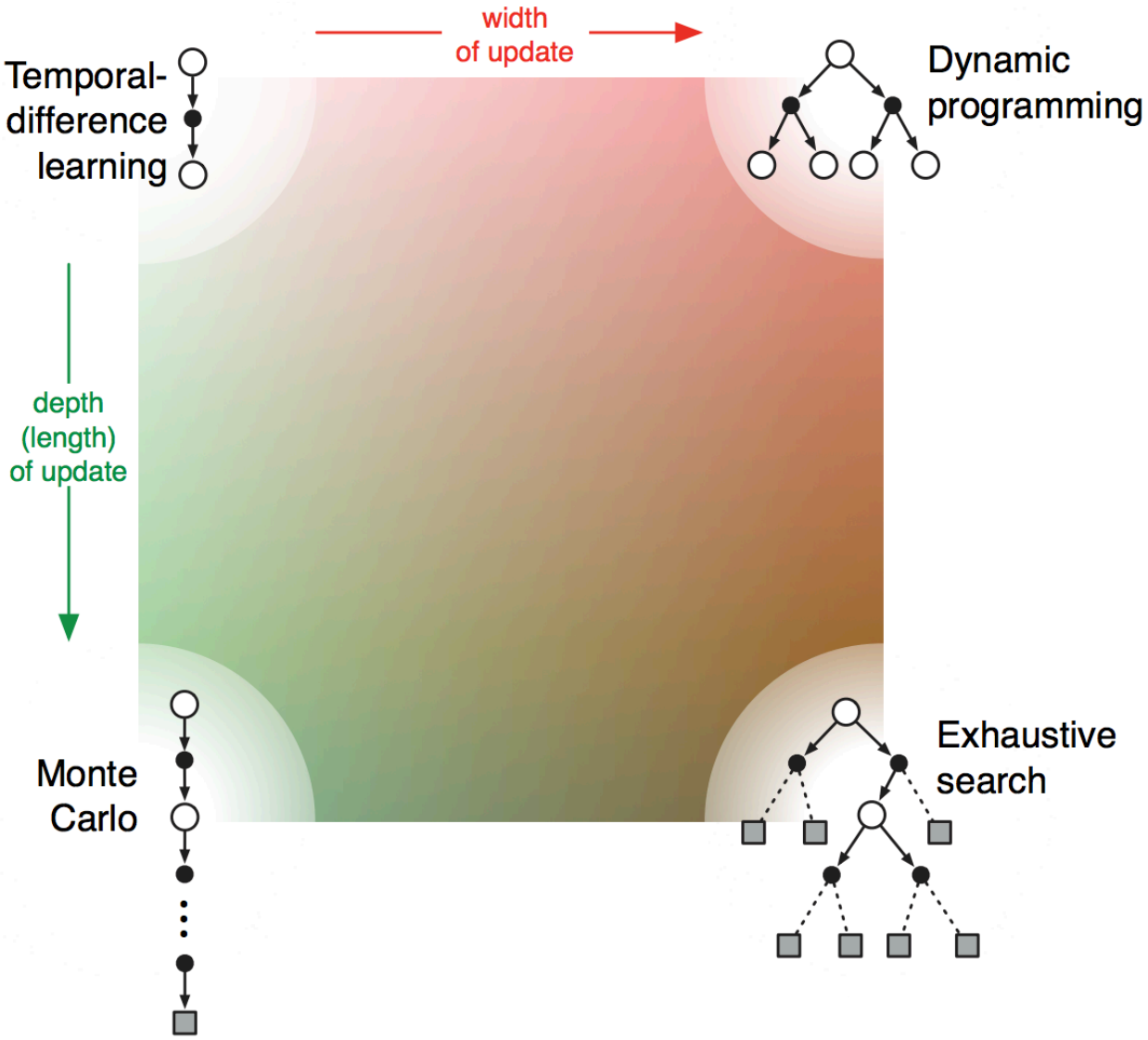
# Bootstrapping and Sampling



- Bootstrapping: update involves an estimate
  - MC does not bootstrap
  - DP bootstraps
  - TD bootstraps
- Sampling: update samples an expectation (not full-width)
  - MC samples
  - DP does not sample
  - TD samples

# Unified View of Reinforcement Learning

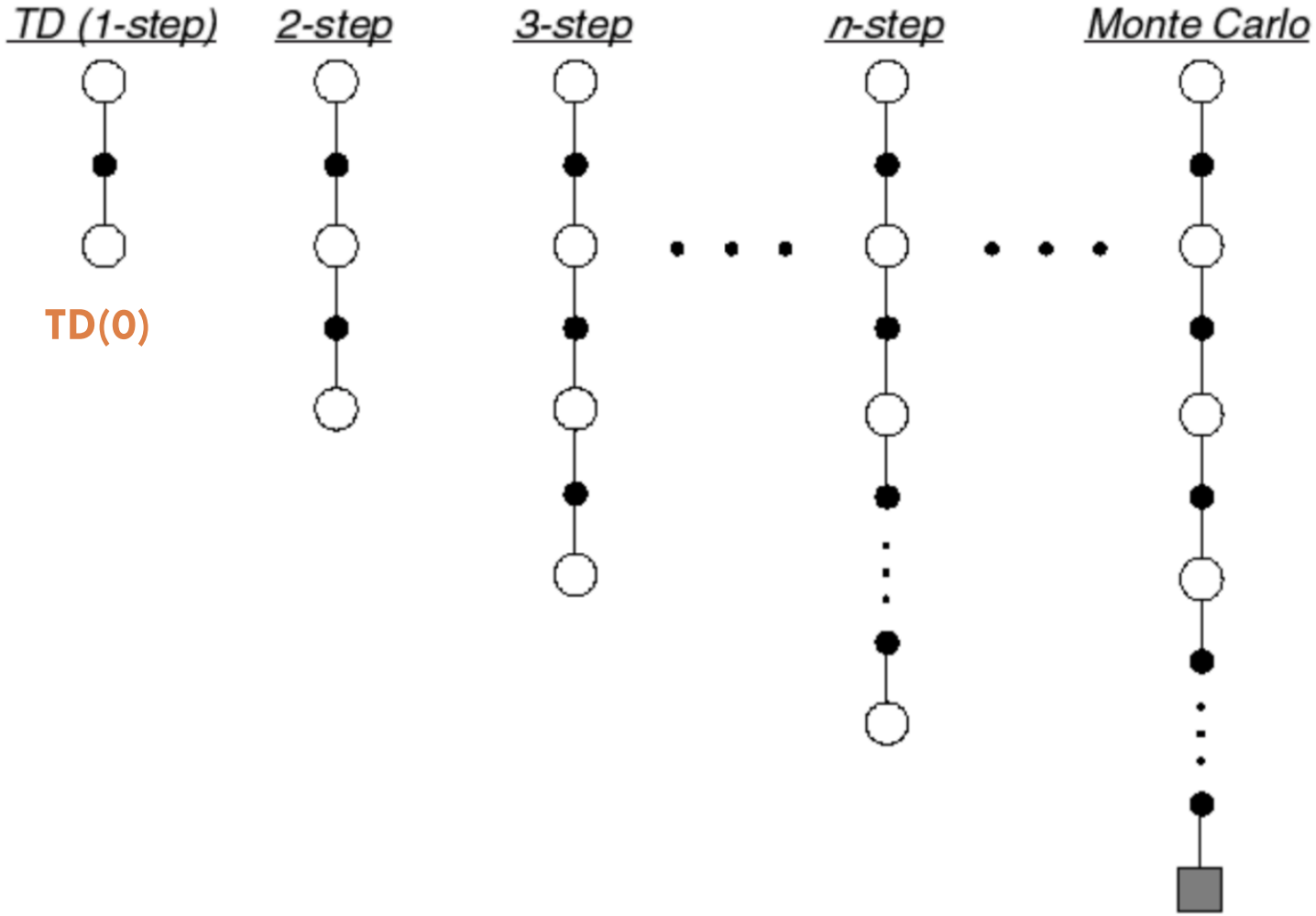




# n-Step Prediction



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



# n-Step Return

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

$n = 1$ (TD) $\quad G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1})$

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

$\vdots$

$n = \infty$ (MC) $\quad G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} \ldots + \gamma^{T-1} V(S_T)$

- Define the n-step return

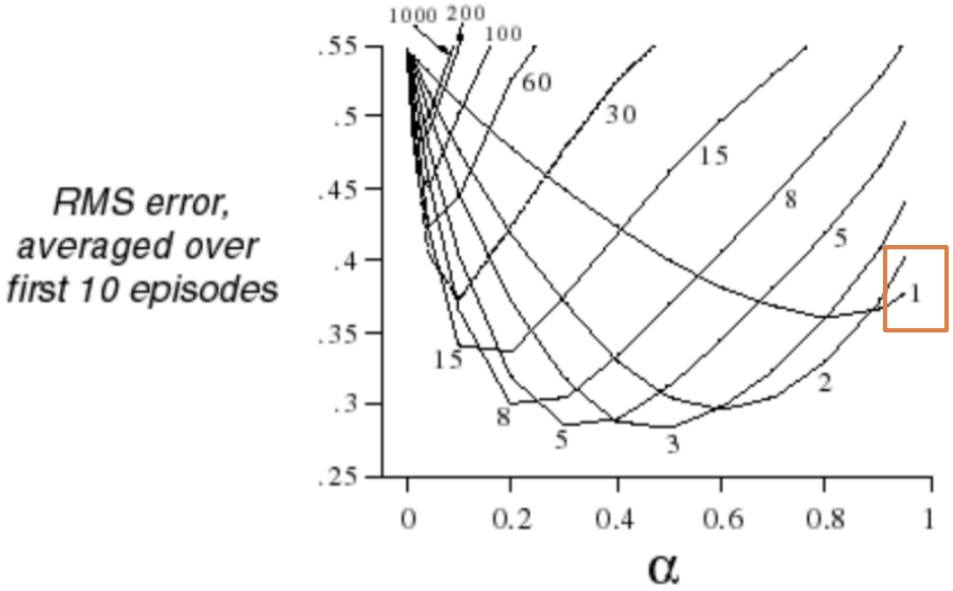
$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} \ldots + \gamma^{n-1} V(S_{t+n}) + \gamma^n V(S_{t+n})$$

- n-step temporal-difference learning

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

# n-step TD: Large Random Walk Example

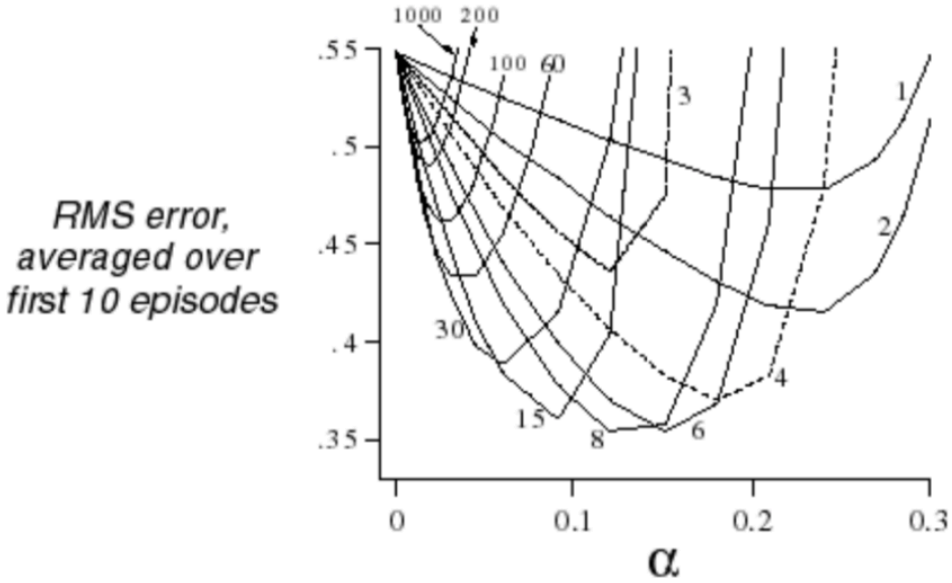





ON-LINE n-STEP TD

**Updates the value function everytime**

Each n influences in a different way



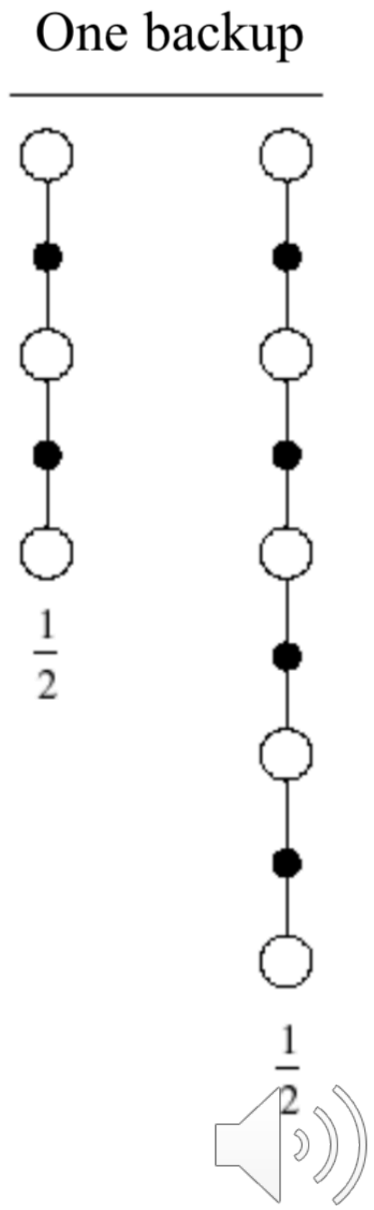

OFF-LINE n-STEP TD

# Averaging n-Step Returns



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

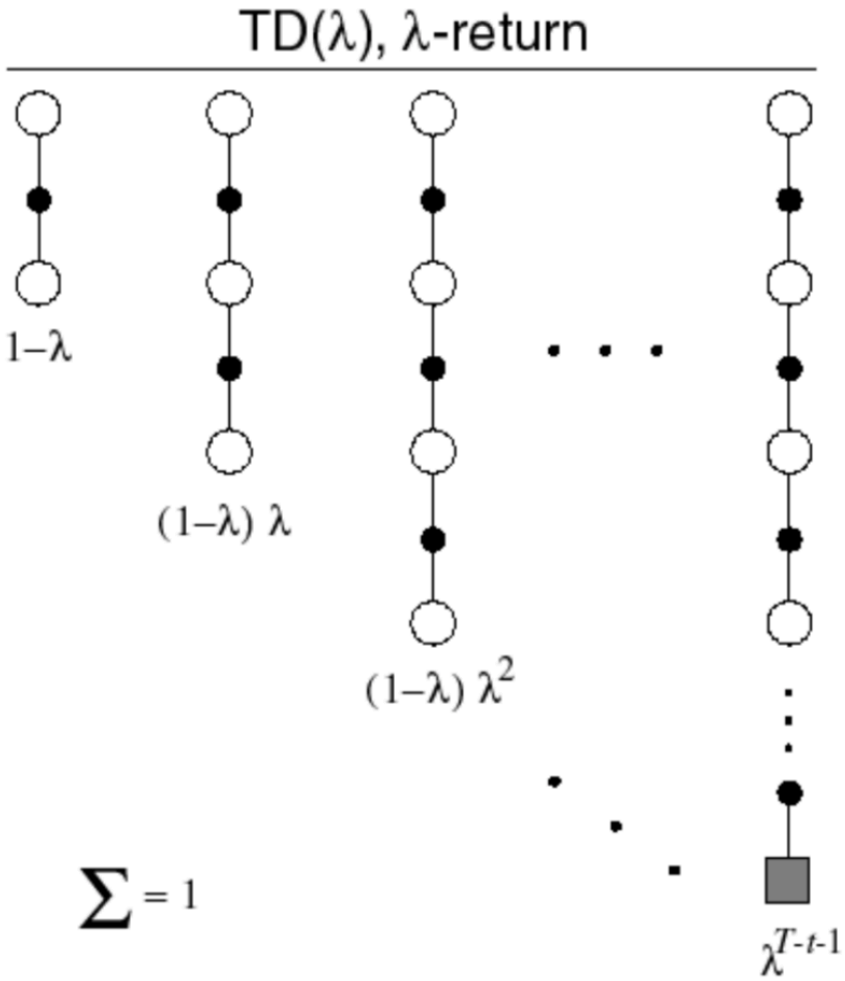
$$1/2G^{(2)}+1/2G^{(4)}$$




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

# λ-return






- The $\lambda$-return $G_t^{(\lambda)}$ combines all n-step returns $G_t^{(n)}$
- With a geometric weighted average (G.A are memoryless - cheap)
- Using weight $(1-\lambda)\lambda^{n-1}$

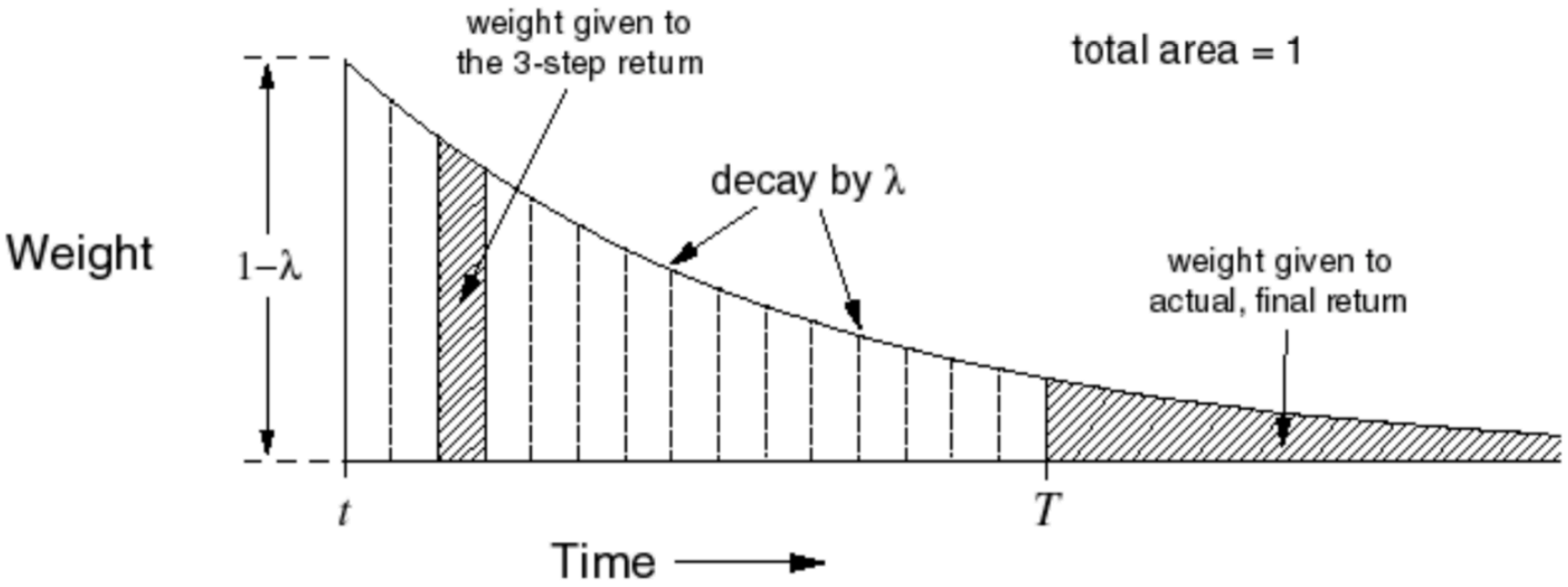
$$G_t^{(\lambda)} = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1}\, G_t^{(n)}$$

- Forward-view TD($\lambda$)

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

# TD(λ) Weighting Function

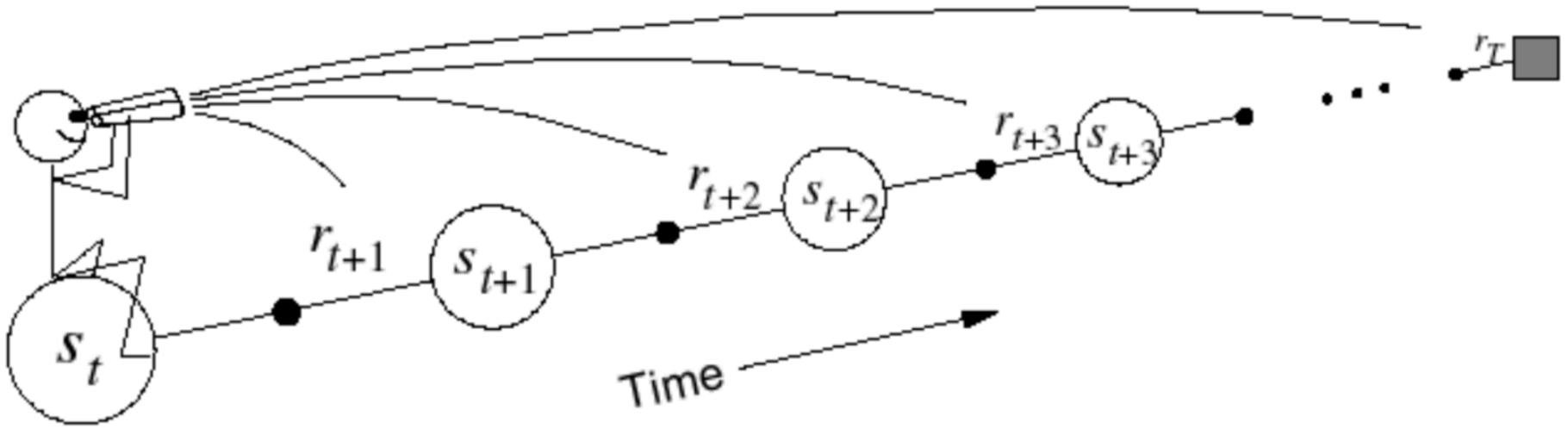





$$G_t^{(\lambda)} = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1}\, G_t^{(n)}$$

# Forward view TD(λ)



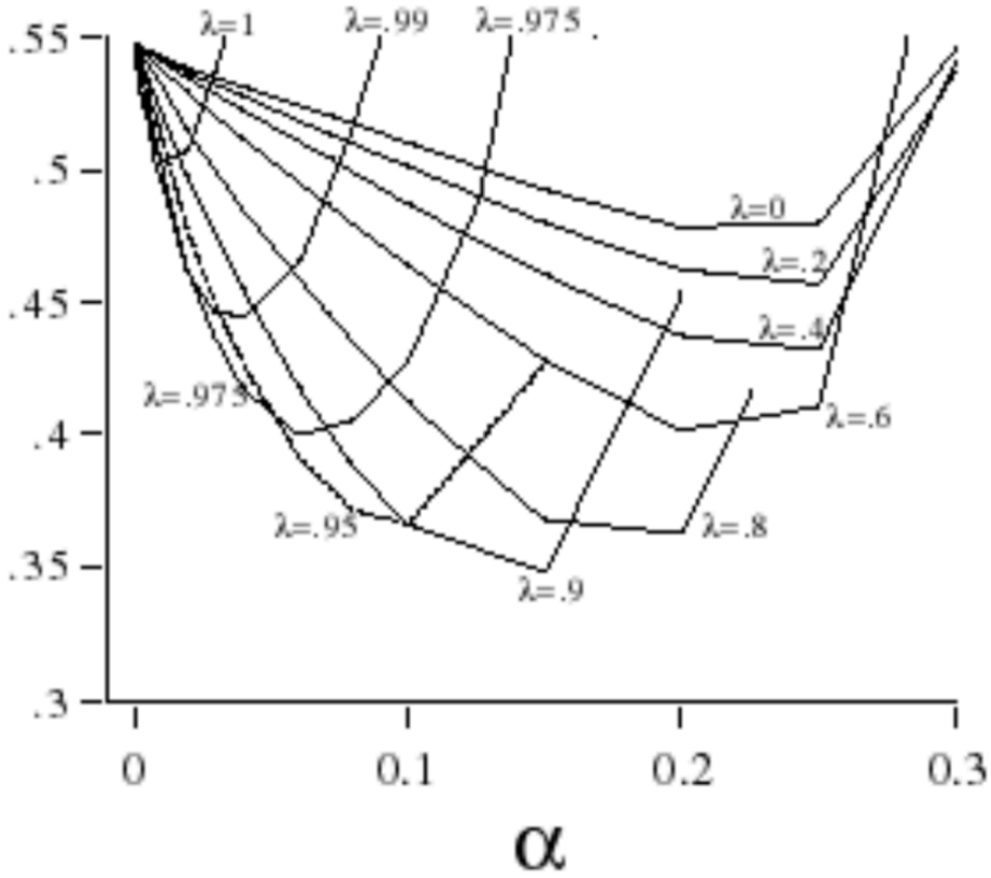



- Update value function towards the $\lambda$ -return
- Forward-view looks into the future to compute $G_t^{(\lambda)}$
- Like MC, can only be computed from complete episodes

# Forward view TD(λ) on Large Random Walk



RMS error, averaged over first 10 episodes




OFF-LINE
λ-RETURN

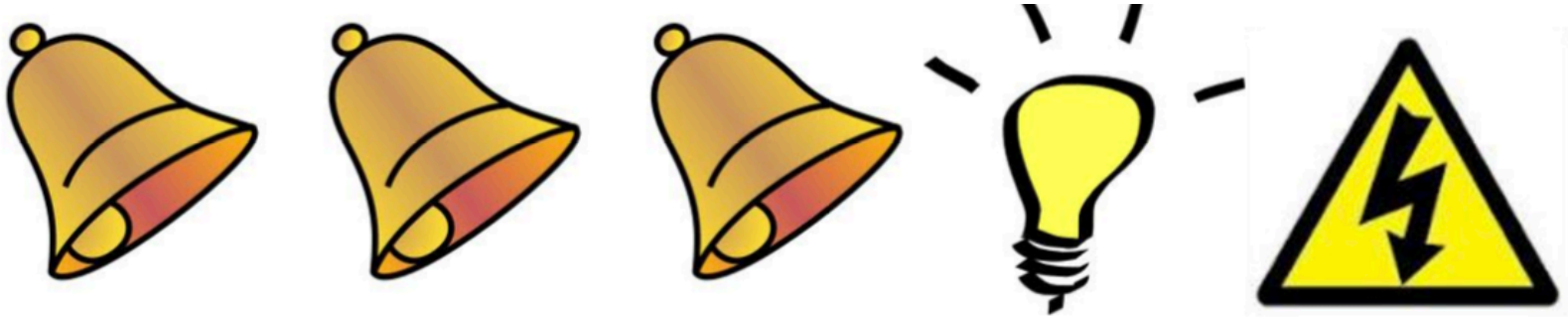
# Backward view TD(λ)



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

# Eligibility Traces

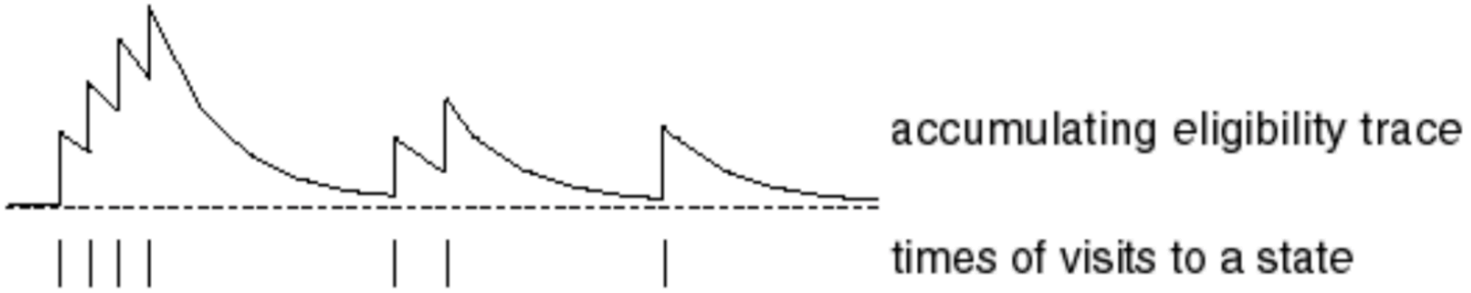




- Credit assignment problem: did bell or light cause shock?
- Frequency heuristic: assign credit to most frequent states
- Recency heuristic: assign credit to most recent states
- Eligibility traces combine both heuristics

$$E_0(s) = 0$$

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



# Eligibility Traces



- Eligibility traces are one of the basic mechanisms of reinforcement learning
- Almost any temporal-difference method can be combined with eligibility traces to obtain a more general method that may learn more efficiently
- They unify and generalize TD and Monte Carlo methods
  - producing a family of methods spanning a spectrum that has Monte Carlo methods at one end ($\lambda = 1$) and one-step TD methods at the other ($\lambda = 0$)
    - In between are intermediate methods that are often better than either extreme method

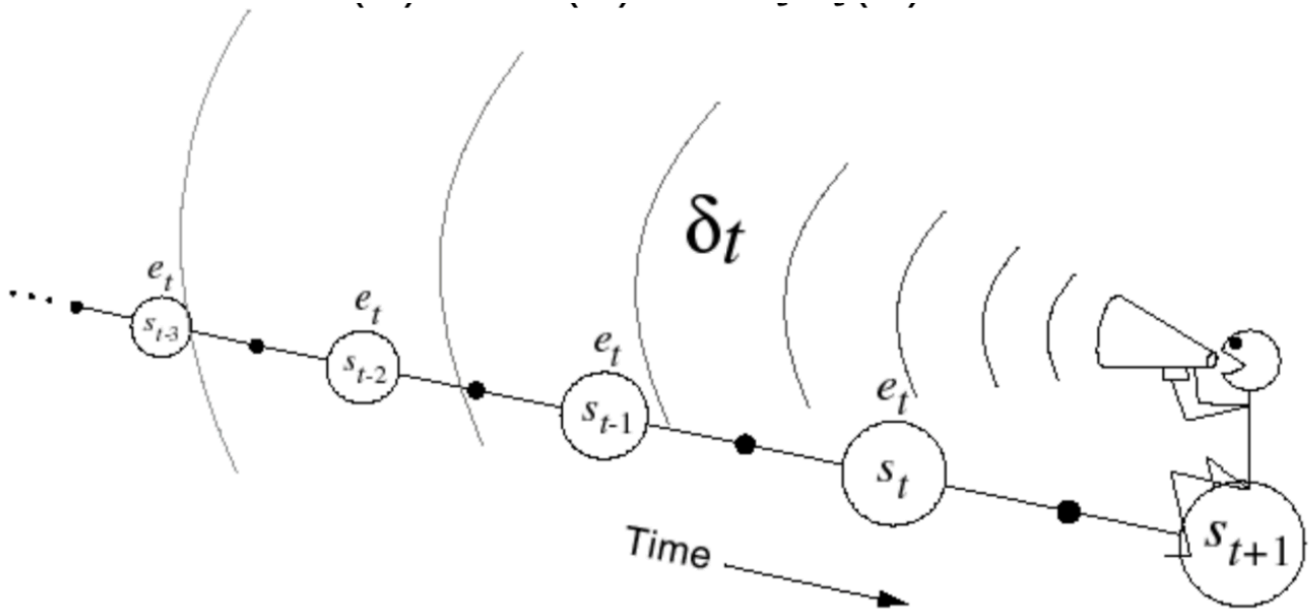
# Backward View TD(λ)



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

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

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



# TD(λ) and TD(0)



- When $\lambda = 0$, only current state is updated
  - $E_t(s) = 1(S_t = s)$
  - $V(S_t) \leftarrow V(S_t) + \alpha\delta_t E_t(s)$
- This is exactly equivalent to TD(0) update
  - $V(S_t) \leftarrow V(S_t) + \alpha\delta_t$

- 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
- Recent results show that is true also to online melthods

- **Theorem:** The sum of offline updates is identical for forward-view and backward-view TD($\lambda$)

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

# Offline/Online Equivalence of Forward and Backward TD



- Offline updates
  - Updates are accumulated within episode
  - but applied in batch at the end of episode
- Online updates
  - TD(λ) updates are applied online at each step within episode
  - Forward and backward-view TD(λ) are slightly different
  - Exact online TD(λ) achieves perfect equivalence
    - By using a slightly different form of eligibility trace
    - Sutton and von Seijen, ICML 2014

# Summary of Forward and Backward TD(λ)



| Offline updates | $\lambda = 0$ | $\lambda \in (0,1)$ | $\lambda = 1$ |
|---|---|---|---|
| Backward view | TD(0) | TD($\lambda$) | TD(1) |
| | $\Vert$ | $\Vert$ | $\Vert$ |
| Forward view | TD(0) | Forward TD($\lambda$) | MC |
| **Online updates** | $\lambda = 0$ | $\lambda \in (0,1)$ | $\lambda = 1$ |
| Backward view | TD(0) | TD($\lambda$) | TD(1) |
| | $\Vert$ | $\nparallel$ | $\nparallel$ |
| Forward view | TD(0) | Forward TD($\lambda$) | MC |
| | $\Vert$ | $\Vert$ | $\Vert$ |
| Exact Online | TD(0) | Exact Online TD($\lambda$) | Exact Online TD(1) |

= here indicates equivalence in total update at end of episode.

- Consider an episode where $s$ is visited once at time-step $k$,
- TD(1) eligibility trace discounts time since visit,

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

$$= \begin{cases} 0, & t < k \\ \gamma^{t-k}, & t \geq k \end{cases}$$

- TD(1) updates accumulate error online

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

- By end of episode it accumulates total error

$$\delta_k + \gamma \delta_{k+1} + \gamma^2 \delta_{k+2} + \cdots + \gamma^{T-1-k} \delta_{T-1}$$

# Telescoping in TD(1)

- When λ = 1, sum of TD errors telescopes into MC error,

$$
\begin{aligned}
&\delta_t + \gamma\delta_{t+1} + \gamma^2\,\delta_{t+2} + \cdots + \gamma^{T-1-t}\delta_{T-1} \\
&= R_{t+1} + \gamma V(S_{t+1}) - \gamma V(S_t) \\
&+ \gamma R_{t+2} + \gamma^2\, V(S_{t+2}) - \gamma V(S_{t+1}) \\
&+ \gamma^2\, R_{t+3} + \gamma^3\, V(S_{t+3}) - \gamma^2\, V(S_{t+2}) \\
&\vdots \\
&+ \gamma^{T-1-t} R_T + \gamma^{T-t} V(S_T) - \gamma^{T-1-t} V(S_{T-1}) \\
&= R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1-t} R_T - V(S_t) \quad ) \\
&= G_t - V(S_t)
\end{aligned}
$$

# TD(λ) and TD(1)



- TD(1) is roughly equivalent to every-visit Monte-Carlo
- Error is accumulated online, step-by-step
- If value function is only updated offline at end of episode
- Then total update is exactly the same as MC

# Telescoping in TD(λ)



- For general λ, TD errors also telescopes to λ-error, $G_t^\lambda - V(S_t)$

$$G_t^\lambda - V(S_t) = -V(S_t) + (1-\lambda)\lambda^0(R_{t+1}+\gamma V(S_{t+1}))$$
$$+(1-\lambda)\lambda^1((R_{t+1}+\gamma R_{t+2}+\gamma^2 V(S_{t+2}))$$
$$+(1-\lambda)\lambda^2((R_{t+1}+\gamma R_{t+2}+\gamma^2 R_{t+3}+\gamma^3 V(S_{t+3})) + \ldots$$

$$= -V(S_t) + (\gamma\lambda)^0(R_{t+1} + \gamma V(S_{t+1}) - \gamma\lambda V(S_{t+1}))$$
$$+(\gamma\lambda)^1(R_{t+2} + \gamma V(S_{t+2}) - \gamma\lambda V(S_{t+2}))$$
$$+(\gamma\lambda)^2(R_{t+3} + \gamma V(S_{t+3}) - \gamma\lambda V(S_{t+3})) + \ldots$$

$$= (\gamma\lambda)^0(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$
$$+(\gamma\lambda)^1(R_{t+2} + \gamma V(S_{t+2}) - V(S_{t+1}))$$
$$+ (\gamma\lambda^2)(R_{t+3}+\gamma V(S_{t+3}) - V(S_{t+2})) + \ldots$$

$$G_t^\lambda - V(S_t) = \delta_t + \gamma\lambda\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + \ldots$$

# 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,

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

$$= \begin{cases} 0, & t < k \\ (\gamma\lambda)^{t-k}, & t \geq k \end{cases}$$

- 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

# A final Comparison



| | MC | TD(0) | TD(0) | TD(λ) |
|---|---|---|---|---|
| **Pros** | • don't bootstrap<br>  • have advantages in partially non-Markov tasks<br>  • can focus on a small subset of states<br>• model-free (over DP): learn optimal policy directly from interaction with the environment or sample episodes, with no model of the dynamics which is required by DP | • fast because of (over MC) update at each step without delay and (over other) simple equation with minimum amount of computation: suitable for offline application in which data can be generated cheaply from an inexpensive simulation and the objective is simply to process as much data as possible as quickly as possible | • typically performs better (faster) at an intermediate n than either extreme (i.e., TD(0) and MC) because it allows bootstrapping over multiple time steps<br>• conceptually simple/clear (over TD(λ)) | • faster learning particularly when rewards are delayed by many steps because the update is made for every action value in the episode up to the beginning, to different degrees. It makes sense to use TD(λ) in online application where data is scarce and cannot be repeatedly processed<br>• efficient (over n-step method): only need to store a trace vector |
| **Con** | • update delayed until the termination, not applicable for continuing task<br>• high variance and slow convergence | | • update delayed n time steps<br>• require more memory to record states, actions and rewards | • require more computation (over one-step method) |

Lecture 4

- Reading:
  - RUSSELL, S. NORVIG, P. Artificial Intelligence. 3a edição. Chapter 21.
  - BARTO, A., SUTTON, R. Reinforcement Learning: An Introduction. Second Edition. Freely Available at: https://drive.google.com/file/d/1xeUDVGWGUUv1-ccUMAZHJLej2C7aAFWY/view?usp=sharing

# References



Lecture 4


- BARTO, A., SUTTON, R. Reinforcement Learning: An Introduction. Second Edition.
- MURPHY, R. R. Introduction to AI robotics. MIT Press, 2002.
- Lex Fridman, MIT Deep Learning Course, MIT, 2019.
- DUDEK, G.; JENKIN, M. Computational Principles of mobile robotics. Cambridge Press, 2000.
- ROMERO, R. A. F.; PRESTES, E.; OSÓRIO, F.; WOLF, D. (Orgs) Robótica móvel. LTC, 2014.
- BROOKS, R. Intelligence without representation. Artificial Intelligence, 47:139-159, 1991.
- RUSSEL, S. NORVIG, P. Artificial Intelligence: a modern approach. Prentice Hall, 2002.
- BRATKO, I. PROLOG: programming for artificial intelligence. Addison Wesley, 2nd edition, 1990.

# This material is part of the Machine Learning Course

## By Esther Colombini and Alexandre Simões