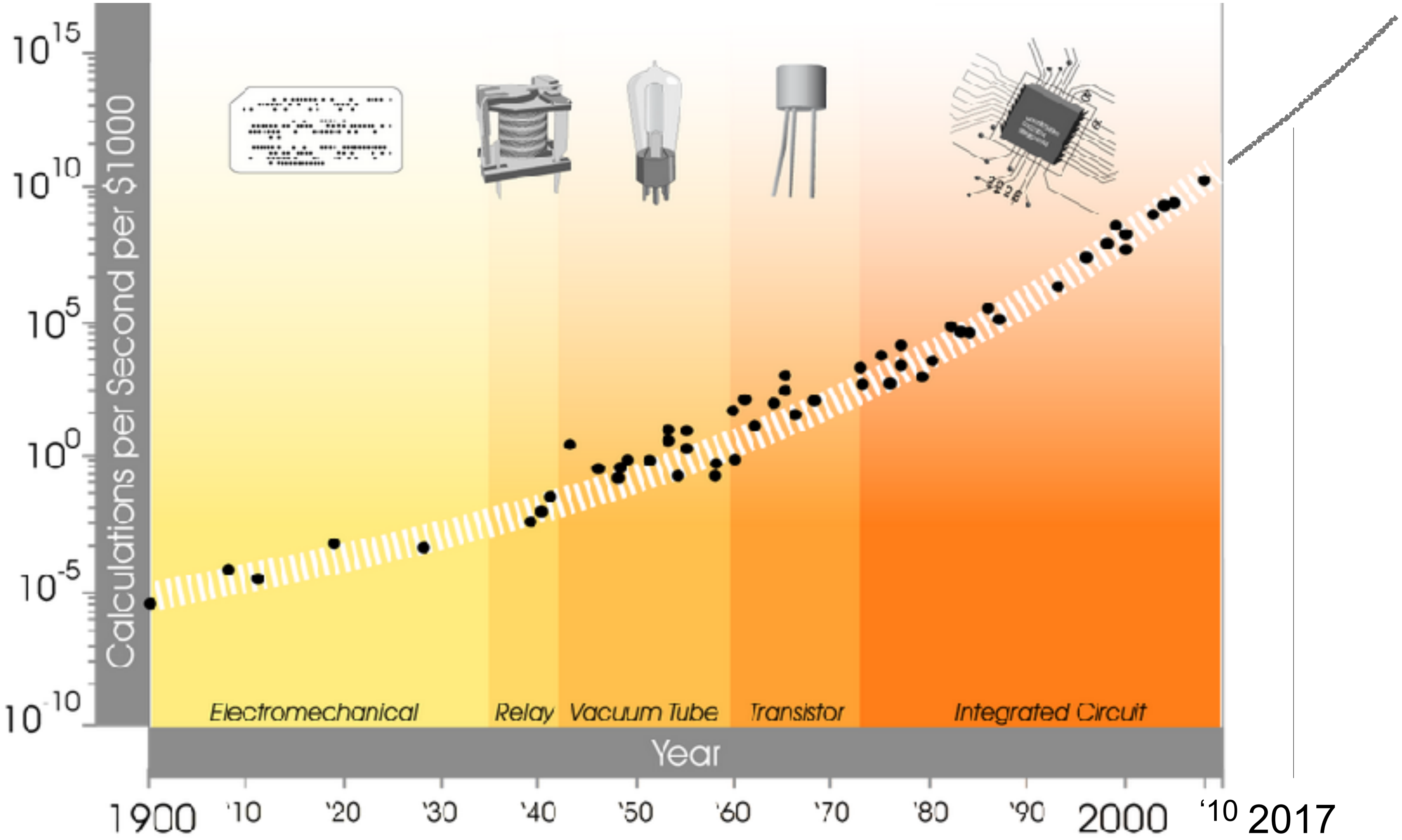
Temporal-Difference Learning

Rich Sutton

Reinforcement Learning & Artificial Intelligence Laboratory
Alberta Machine Intelligence Institute
Dept. of Computing Science, University of Alberta
Canada

We are entering an era of vastly increased computation



from Kurzweil AI

Methods that *scale with computation* are the future of AI

- e.g., learning and search
    - general-purpose methods
- One of the oldest questions in AI has been answered!
    - “weak” general-purpose methods are better than “strong” methods (utilizing human insight)
- Supervised learning and model-free RL methods are only weakly scalable

Prediction learning is scalable

- It’s the unsupervised supervised learning
    - We have a target (just by waiting)
    - Yet no human labeling is needed!
- Prediction learning is the *scalable* model-free learning

Real-life examples of action and prediction learning
Perception, action, and anticipations, as fast as possible

Temporal-difference learning

is a method for learning to predict

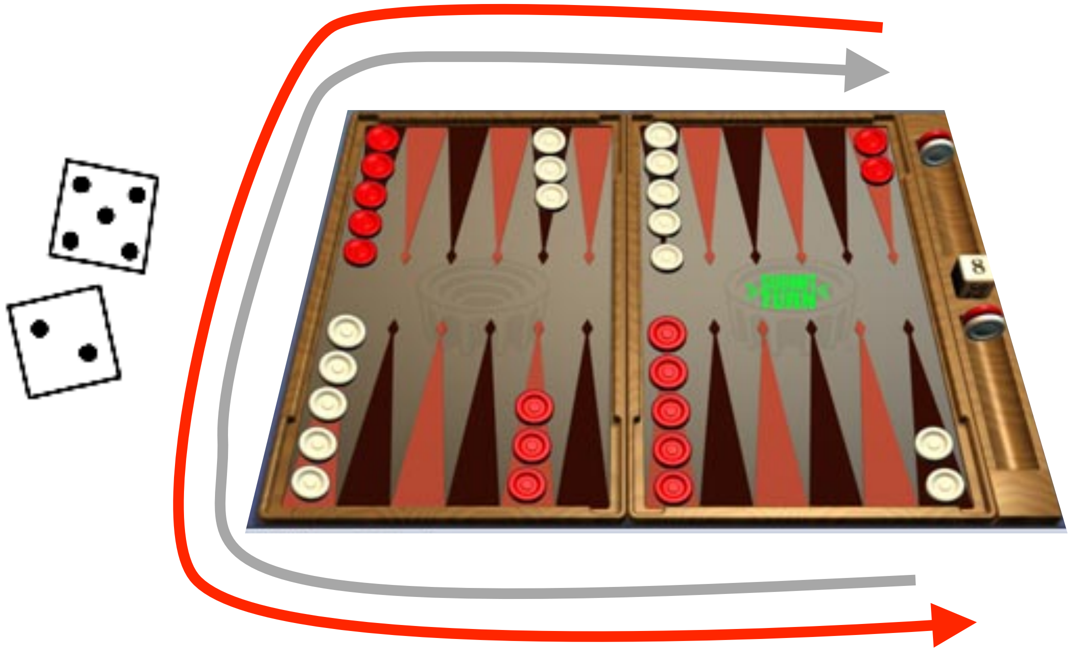
- Widely used in RL to predict future reward (value functions)
- Key to Q-learning, Sarsa, TD(λ), Deep Q network, TD-Gammon, actor-critic methods, Samuel's checker player
    - but not AlphaGo, helicopter autopilots, pure-policy-based methods…
- Appears to be how brain reward systems work
- Can be used to predict any signal, not just reward

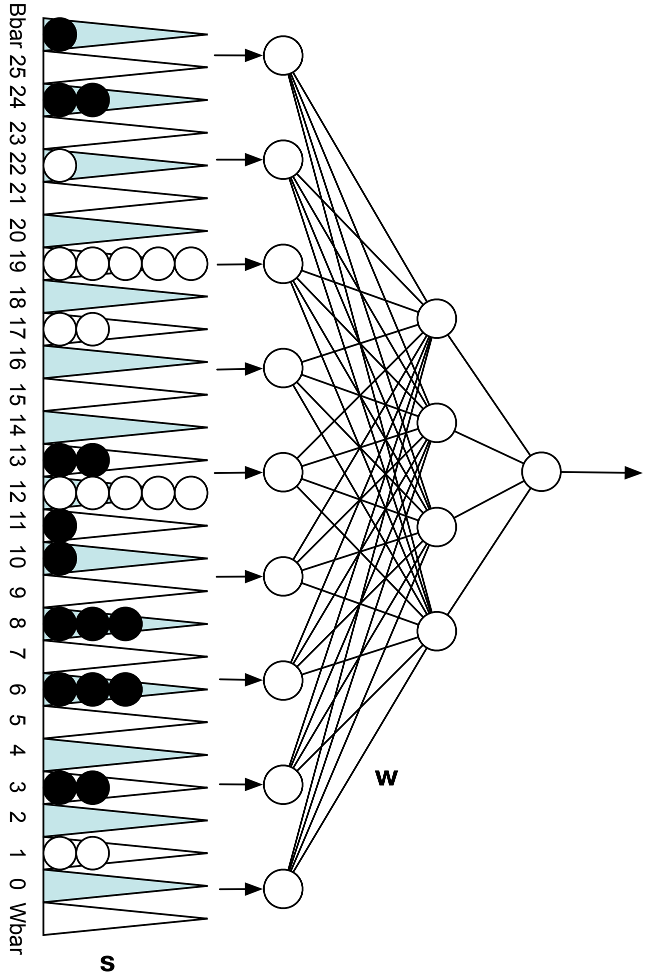
TD learning is learning a prediction **from another, later, learned prediction**

- i.e., *learning a guess from a guess*
- The TD error is the difference between the two predictions, the *temporal difference*
- Otherwise TD learning is the same as supervised learning, backpropagating the error

Example: TD-Gammon

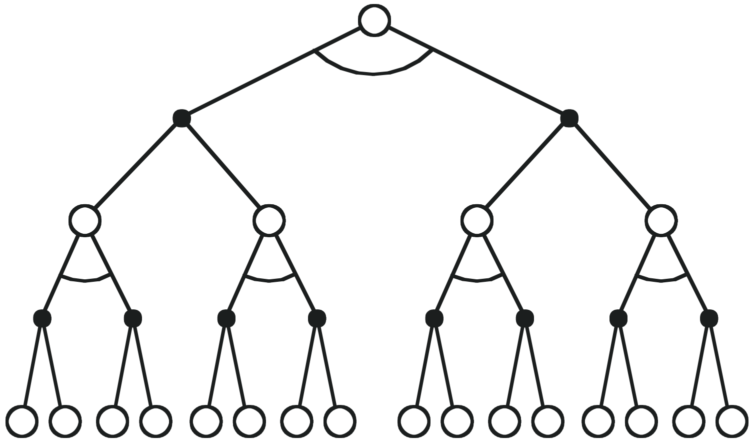
Tesauro, 1992-1995




Bbar 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 Wbar
w
s

estimated state value
(≈ prob of winning)

Action selection
by a shallow search



Start with a random Network

Play millions of games against itself

Learn a value function from this simulated experience

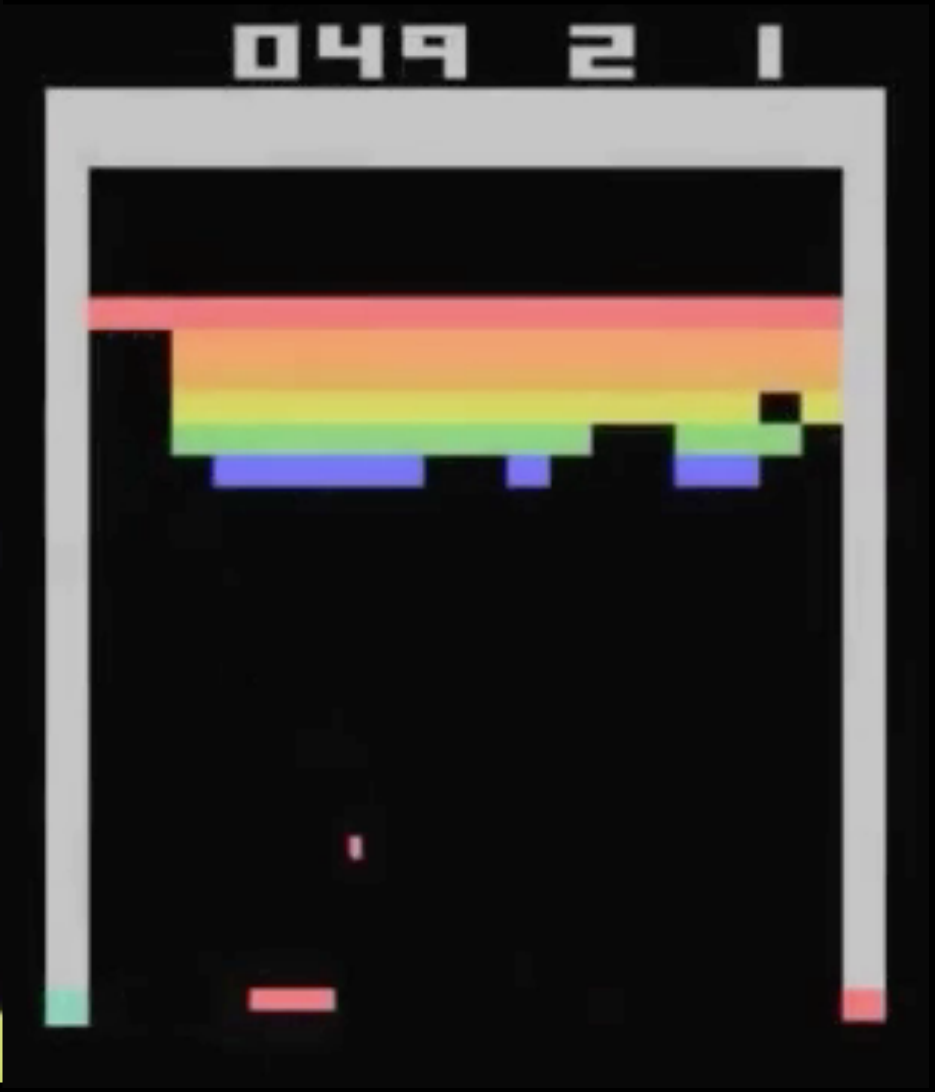
Six weeks later it's the best player of backgammon in the world

Originally used expert handcrafted features, later repeated with raw board positions

But do I need TD learning?
or can I use ordinary supervised learning?

RL + Deep Learning Performance on Atari Games

Space Invaders

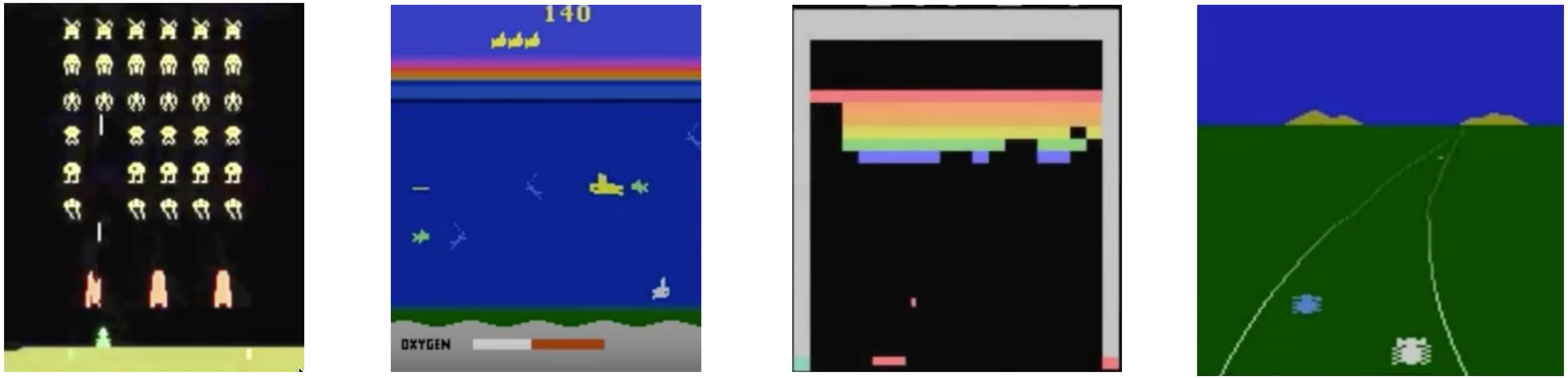


Breakout

Enduro

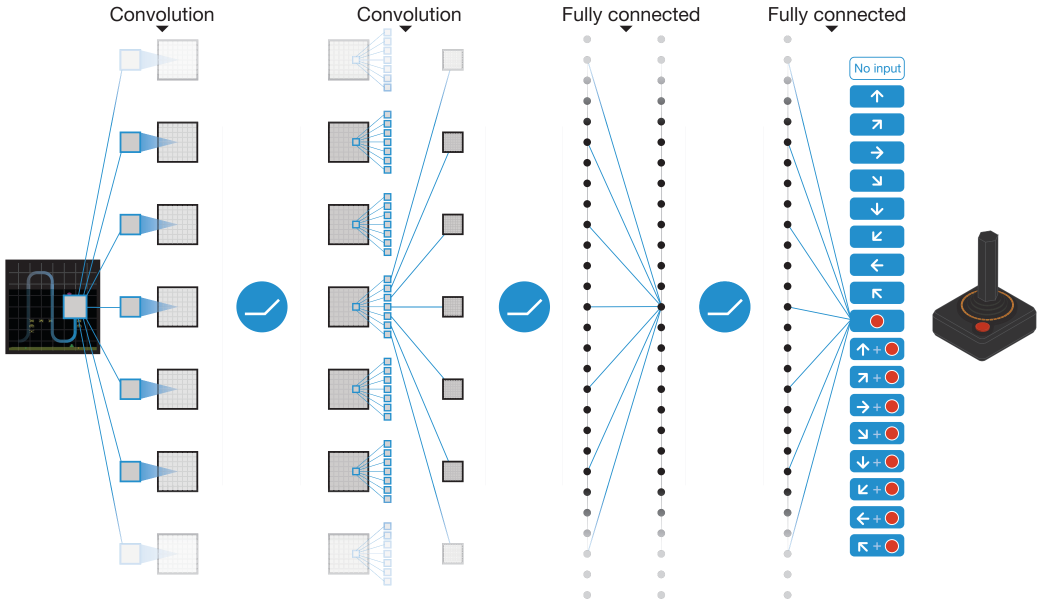
RL + Deep Learning, applied to Classic Atari Games

Google Deepmind 2015, Bowling et al. 2012



- Learned to play 49 games for the Atari 2600 game console, without labels or human input, from self-play and the score alone

mapping raw screen pixels



to predictions of final score for each of 18 joystick actions

- Learned to play better than all previous algorithms and at human level for more than half the games

Same learning algorithm applied to all 49 games! w/o human tuning

TD learning is relevant only on *multi-step* prediction problems

- Only when the thing predicted is multiple steps in the future
  - with information about it possibly revealed on each step
- In other words, everything other than the classical supervised learning setup

Examples of multi-step prediction

- Predicting the outcome of a game, like chess or backgammon
- Predicting what a stock-market index will be at the end of the year, or in six months
- Predicting who will be the next US president
- Predicting who the US will next go to war against
    - or how many US soldiers will be killed during a president's term
- Predicting a sensory observation, in 10 steps, in roughly 10 steps, or when something else happens
- Predicting discounted cumulative reward conditional on behavior

Do we need to think about multi-step predictions?

- Can’t we just think of the multi-step as one big step, and then use one-step methods?
- Can’t we just learn one-step predictions, and then iterate them (compose them) to produce multi-step predictions when needed?
- No, we really can’t (and shouldn’t want to)

The one-step trap:

Thinking that one-step predictions are sufficient

- That is, at each step predict the state and observation *one step later*
- Any long-term prediction can then be made by simulation
- In theory this works, but not in practice
  - Making long-term predictions by simulation is exponentially complex
  - and amplifies even small errors in the one-step predictions
- Falling into this trap is very common: POMDPs, Bayesians, control theory, compression enthusiasts

Can’t we just use our familiar one-step supervised learning methods?

(applied to RL, these are known as Monte Carlo methods)

- Can’t we just wait until the target is known, then use a one-step method? (reduce to input-output pairs)
  - E.g., wait until the end of the game, then regress to the outcome
- No, not really; there are significant computational costs to this
  - memory scales with the span (#steps) of the prediction
  - computation is poorly distributed over time
- These can be avoided with learning methods specialized for multi-step
- Also, sometimes the target is never known (off-policy)
- We should not ignore these things; they are not nuisances, they are <u>clues</u>, hints from nature

New RL notation

- Life: $\underset{\text{State}}{S_0}, \underset{\text{Action}}{A_0}, \underset{\text{Reward}}{R_1}, S_1, A_1, R_2, S_2, \ldots$

- Return:

$$
\begin{aligned}
G_t &\overset{\text{Definition}}{\doteq} R_{t+1} + \overset{\text{Discount rate, e.g., 0.9}}{\gamma} R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\
&= R_{t+1} + \gamma\left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots\right) \\
&= R_{t+1} + \gamma G_{t+1}
\end{aligned}
$$

- state-value function:

$$
\begin{aligned}
\underset{\text{True value of state s under policy }\pi}{v_\pi(s)} &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]
\end{aligned}
$$

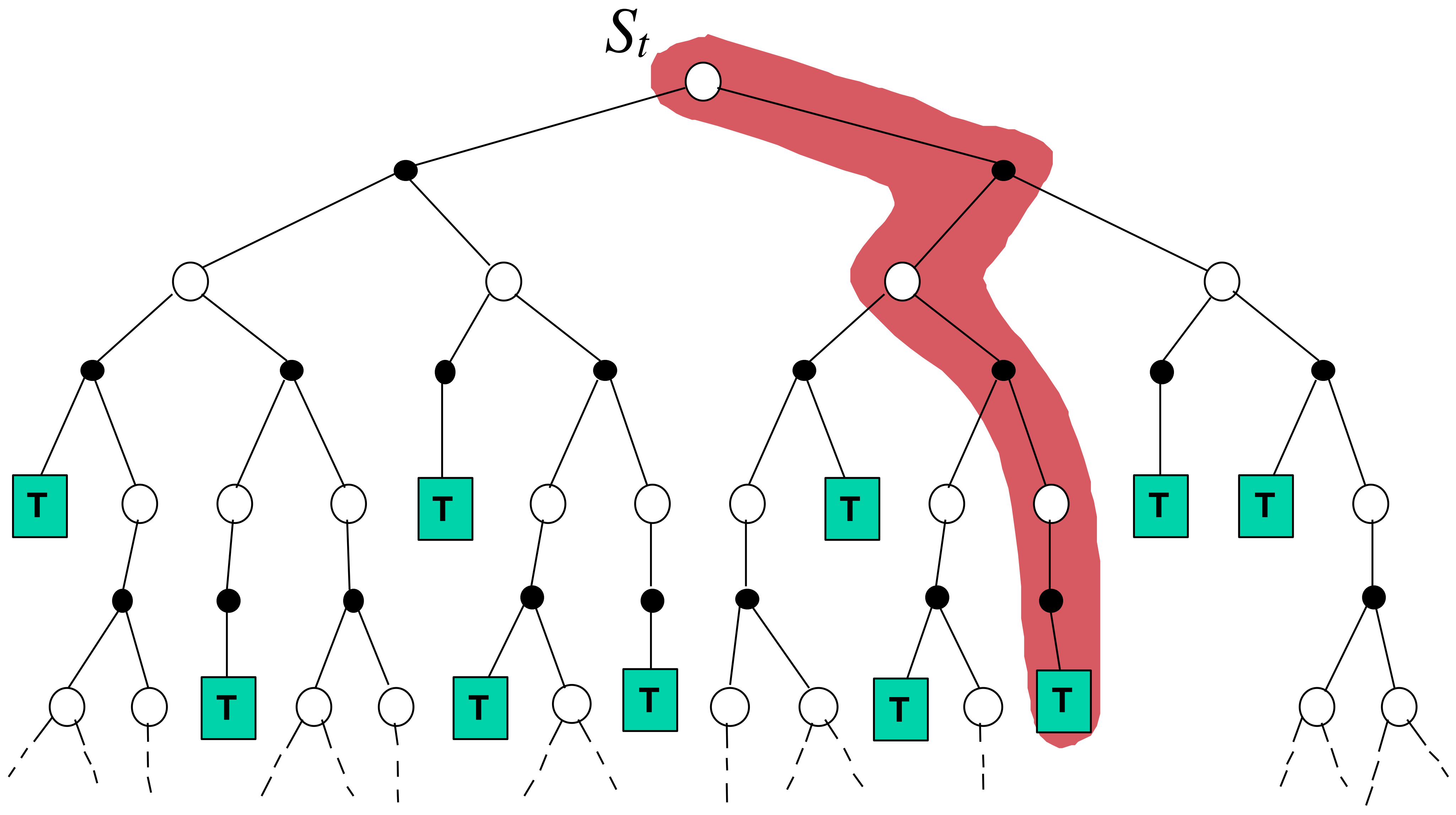
- TD error:

Estimated value function

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

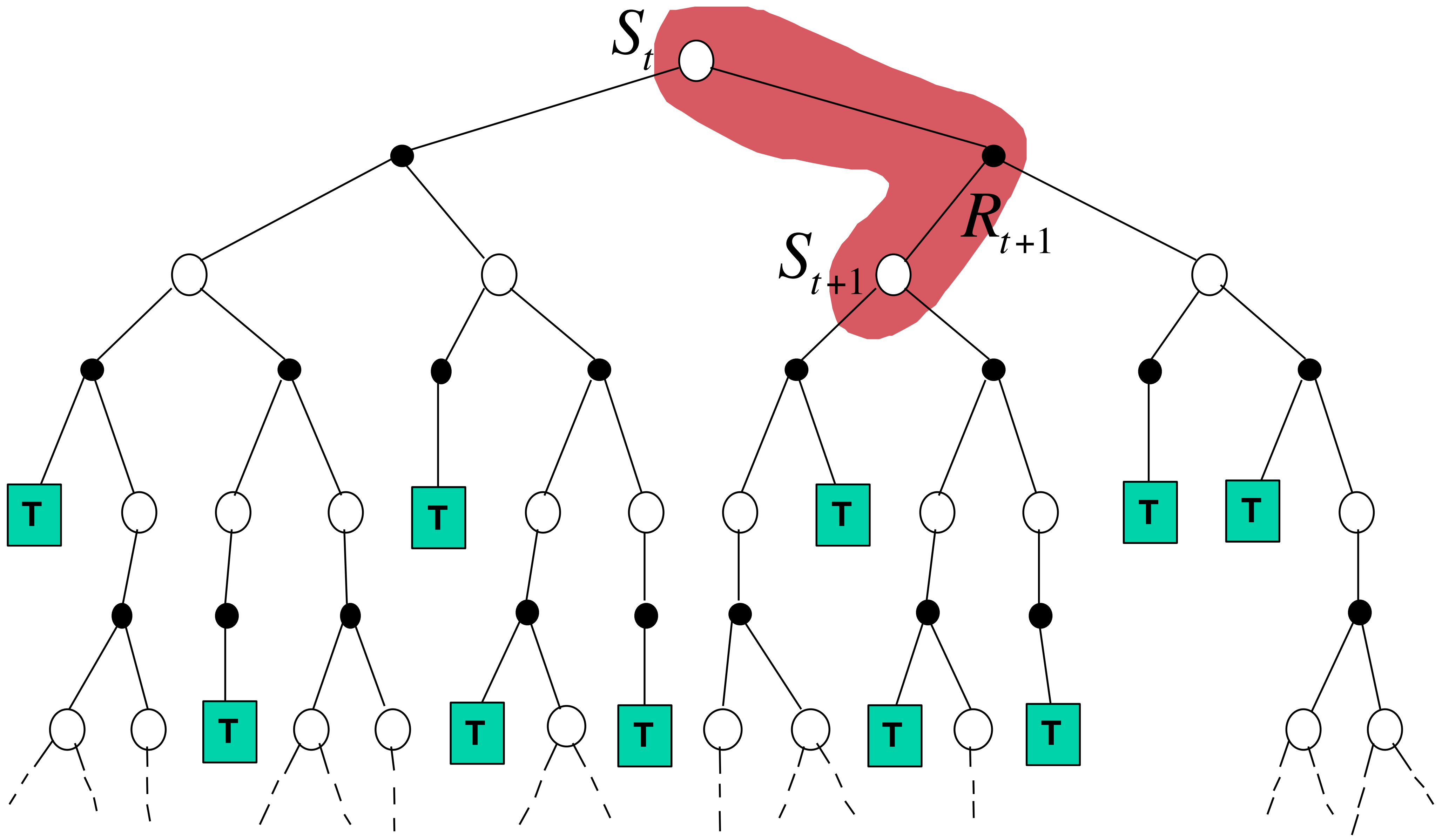
Monte Carlo (Supervised Learning) (MC)

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



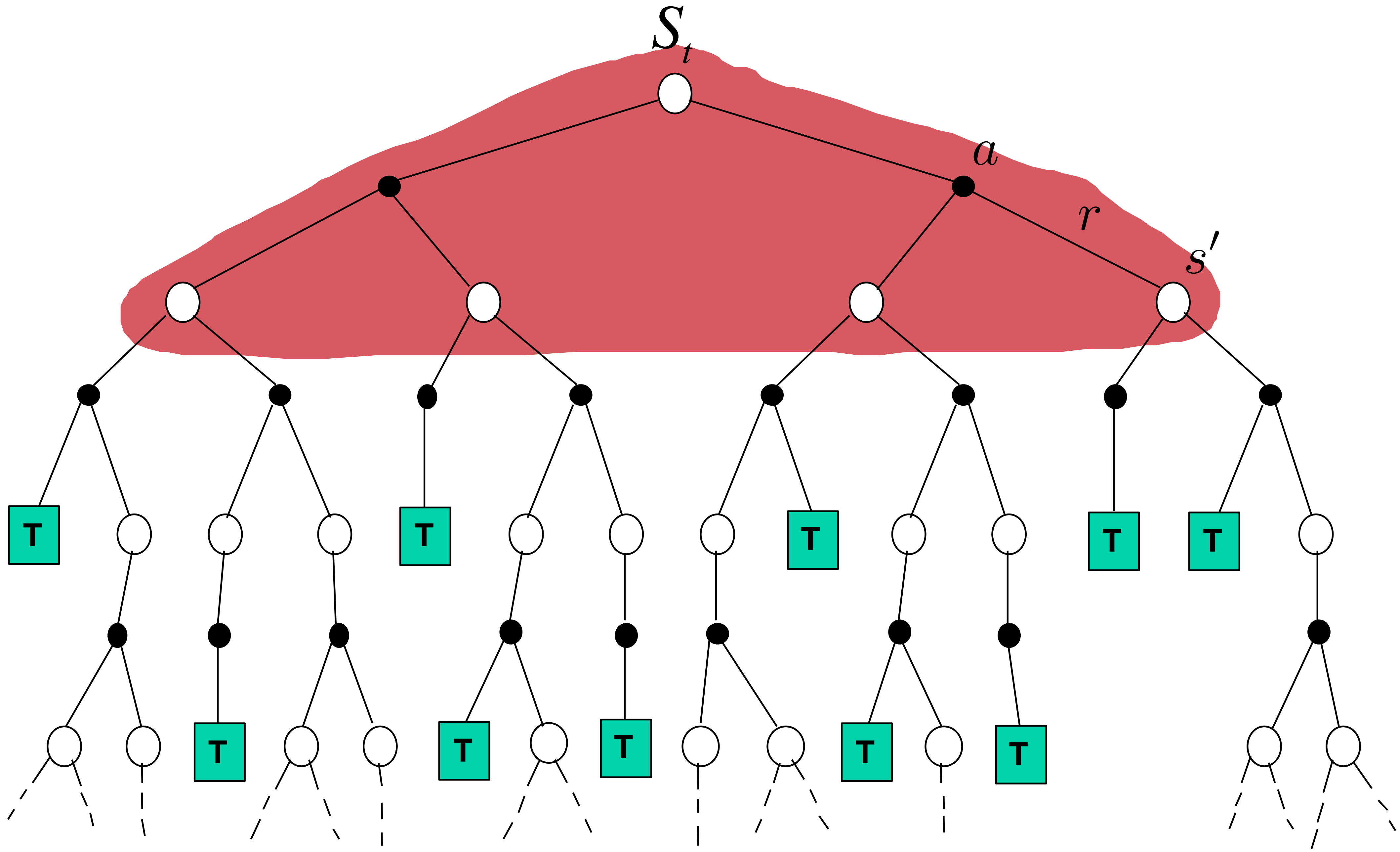
Simplest TD Method

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



cf. Dynamic Programming

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



TD methods bootstrap and sample

- Bootstrapping: update involves an *estimate*
  - MC does not bootstrap
  - Dynamic Programming bootstraps
  - TD bootstraps
- Sampling: update does not involve an *expectation*
  - MC samples
  - Dynamic Programming does not sample
  - TD samples

TD Prediction

Policy Evaluation (the prediction problem):

for a given policy π, compute the state-value function v_π

Recall: Simple every-visit Monte Carlo method:

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

Step-size parameter

target: the actual return after time *t*

The simplest temporal-difference method TD(0):

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\text{target: an estimate of the return}} - V(S_t)\Big]$$

target: an estimate of the return

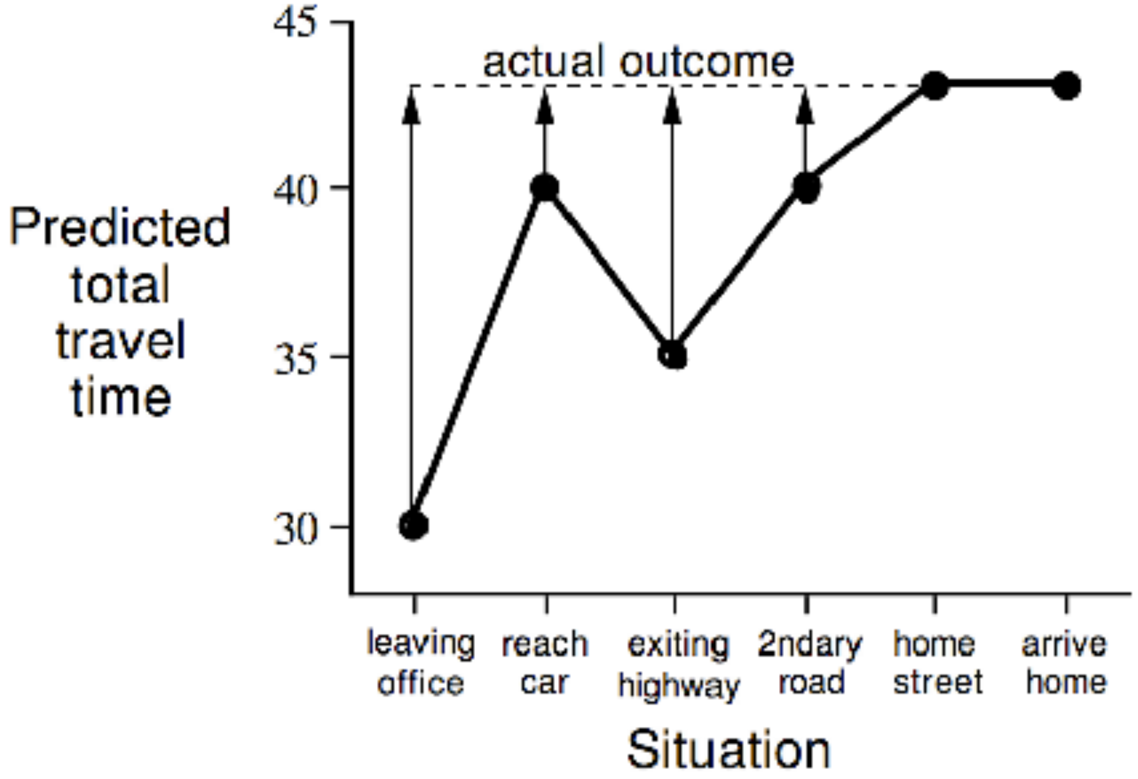
Example: Driving Home

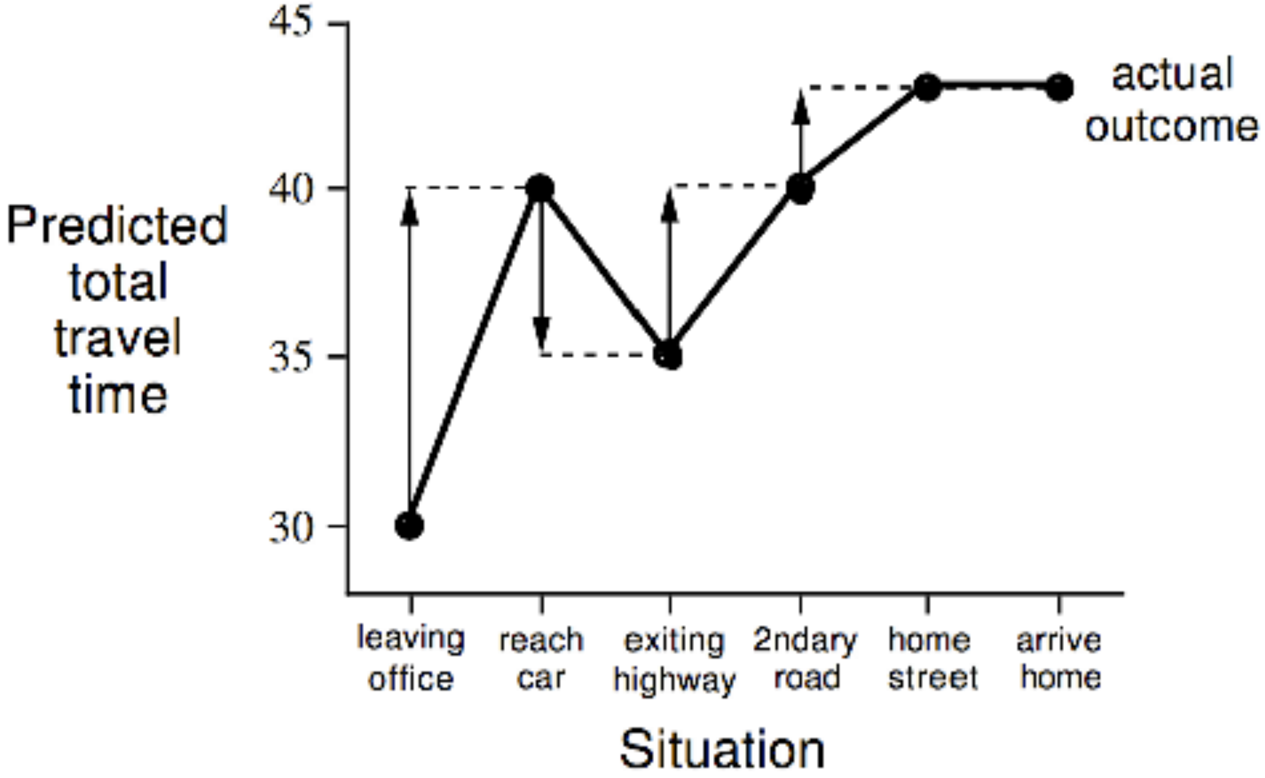
	Elapsed Time	*Predicted*	*Predicted*
State	*(minutes)*	*Time to Go*	*Total Time*
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43

Driving Home

Changes recommended by Monte Carlo methods (α=1)

Changes recommended by TD methods (α=1)

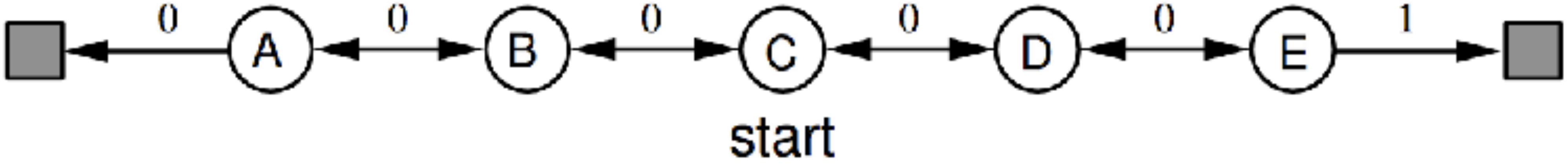


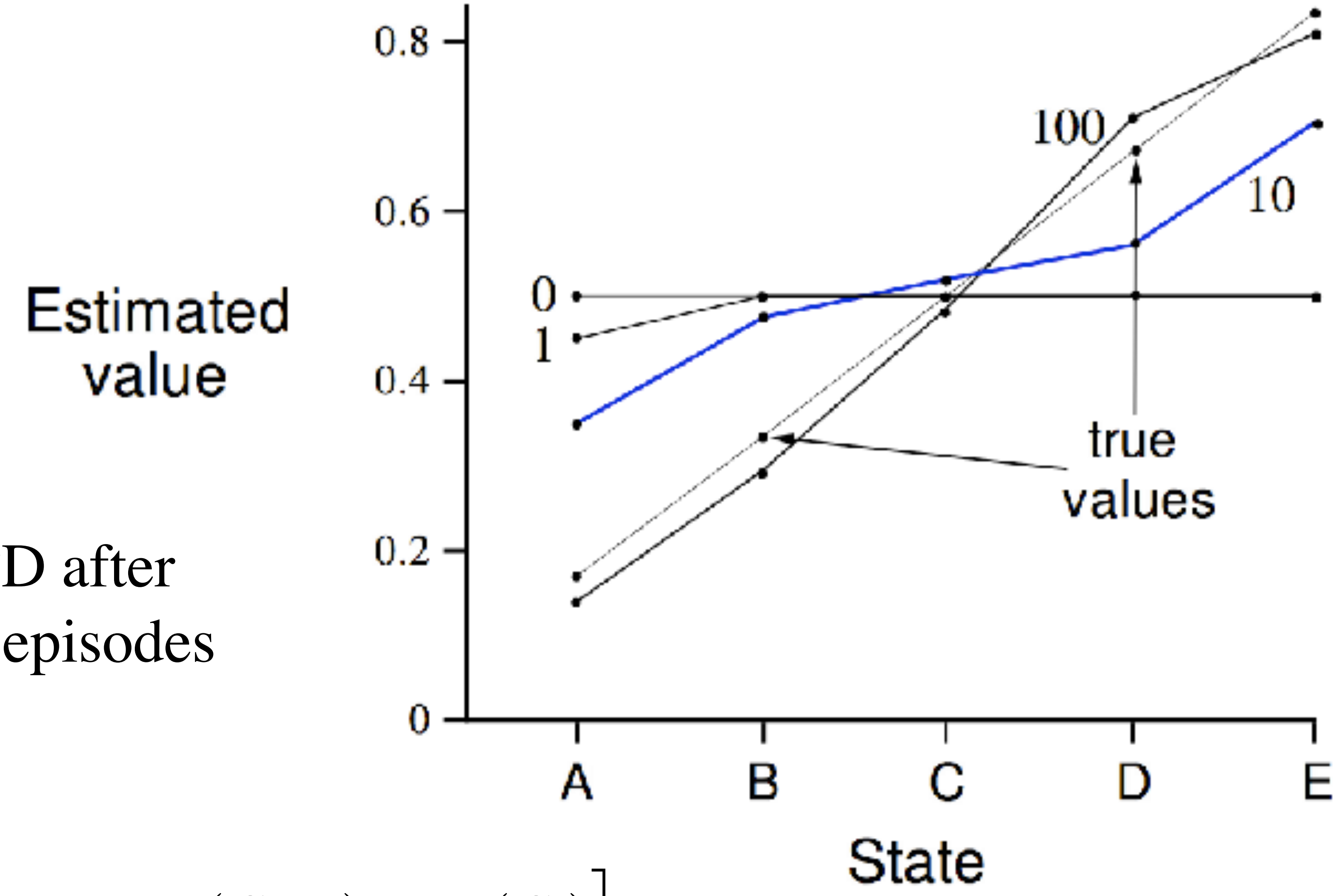


Advantages of TD Learning

- TD, but not MC, methods can be fully incremental
  - You can learn before knowing the final outcome
    - Less memory
    - Less peak computation
  - You can learn without the final outcome
    - From incomplete sequences
- Both MC and TD converge (under certain assumptions to be detailed later), but which is faster?

Random Walk Example

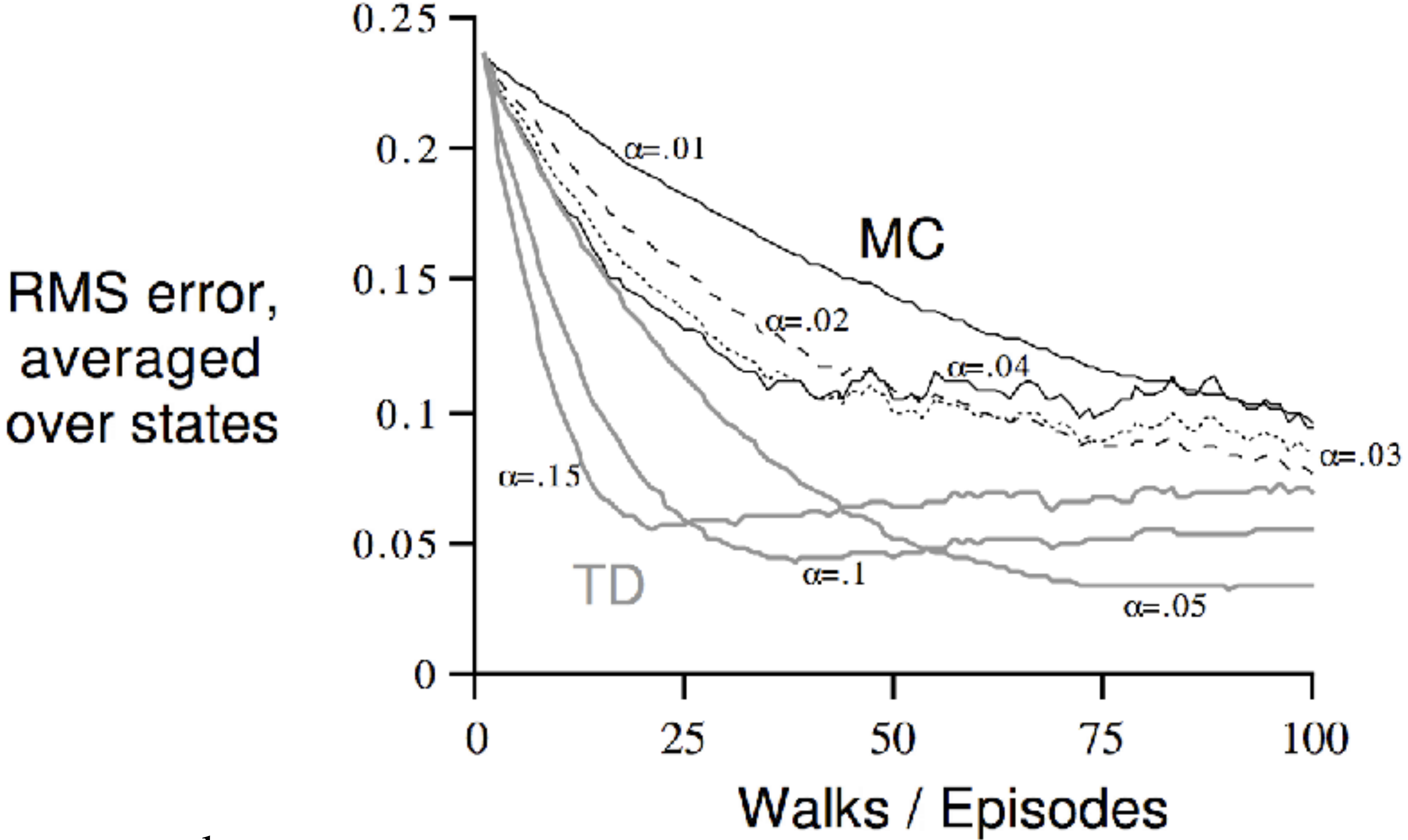




Values learned by TD after various numbers of episodes

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

TD and MC on the Random Walk



Data averaged over
100 sequences of episodes

Batch Updating in TD and MC methods

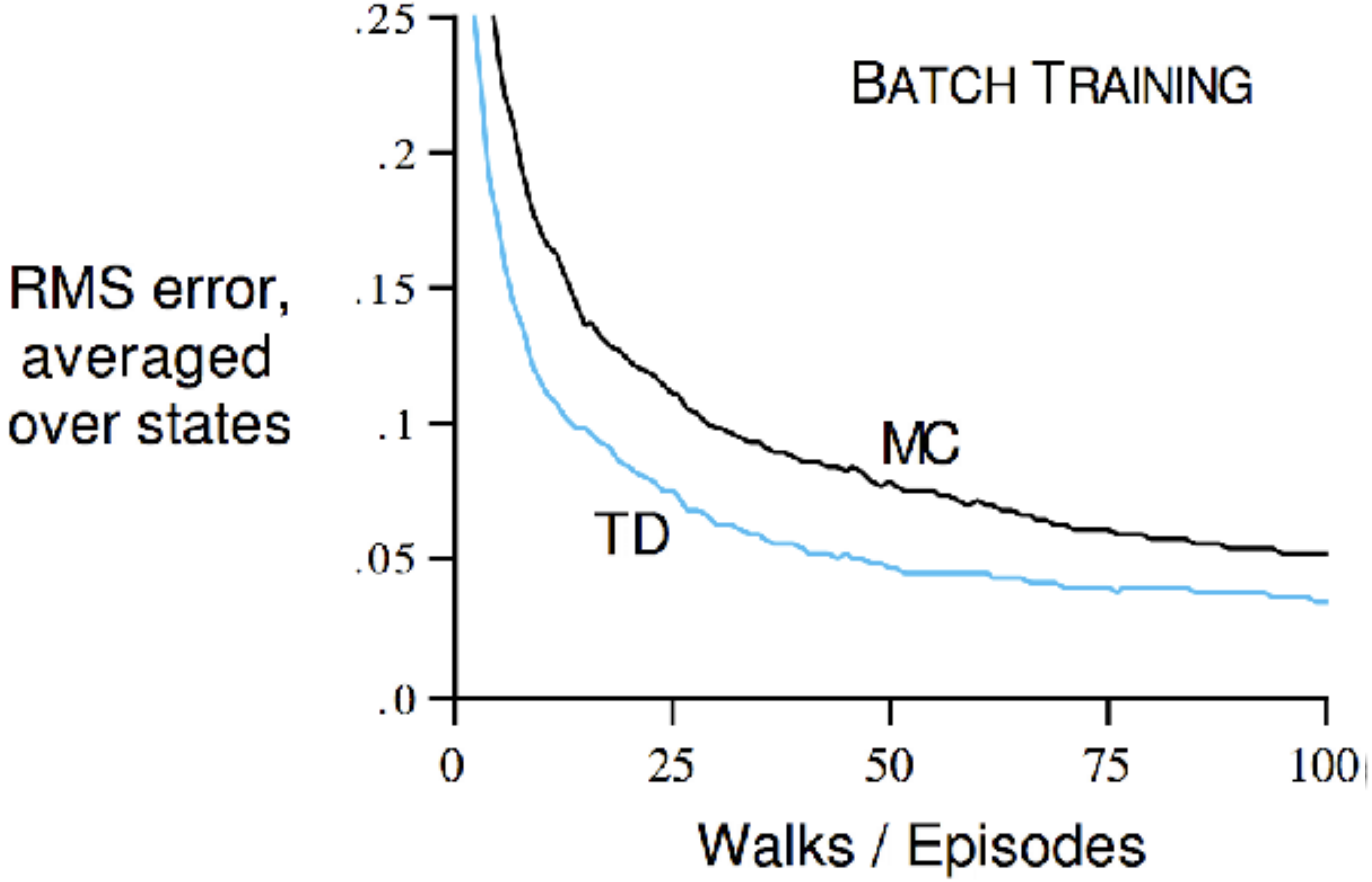
Batch Updating: train completely on a finite amount of data, e.g., train repeatedly on 10 episodes until convergence.

Compute updates according to TD or MC, but only update estimates after each complete pass through the data.

For any finite Markov prediction task, under batch updating, TD converges for sufficiently small α.

Constant-α MC also converges under these conditions, but to a difference answer!

Random Walk under Batch Updating



After each new episode, all previous episodes were treated as a batch, and algorithm was trained until convergence. All repeated 100 times.

You are the Predictor

Suppose you observe the following 8 episodes:

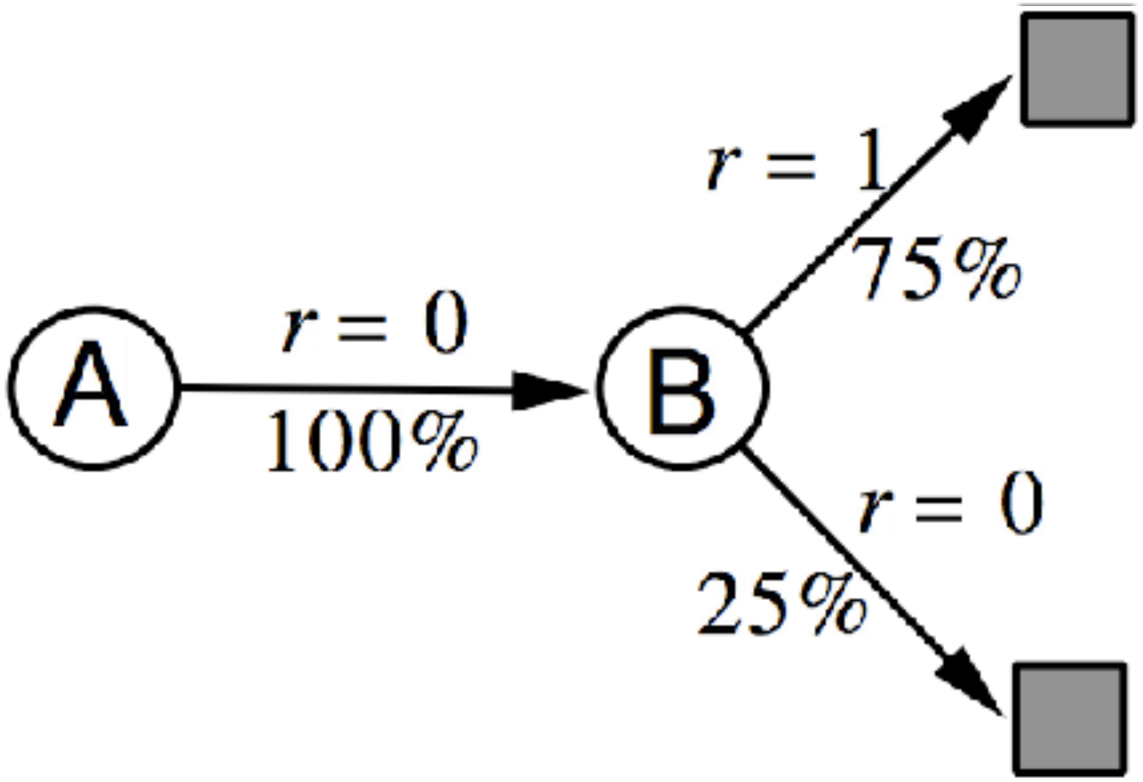
A, 0, B, 0
B, 1
B, 1
B, 1
B, 1
B, 1
B, 1
B, 0

$V(\mathrm{B})$? 0.75

$V(\mathrm{A})$? 0?

Assume Markov states, no discounting ($\gamma = 1$)

You are the Predictor



$V(\text{A})$? 0.75

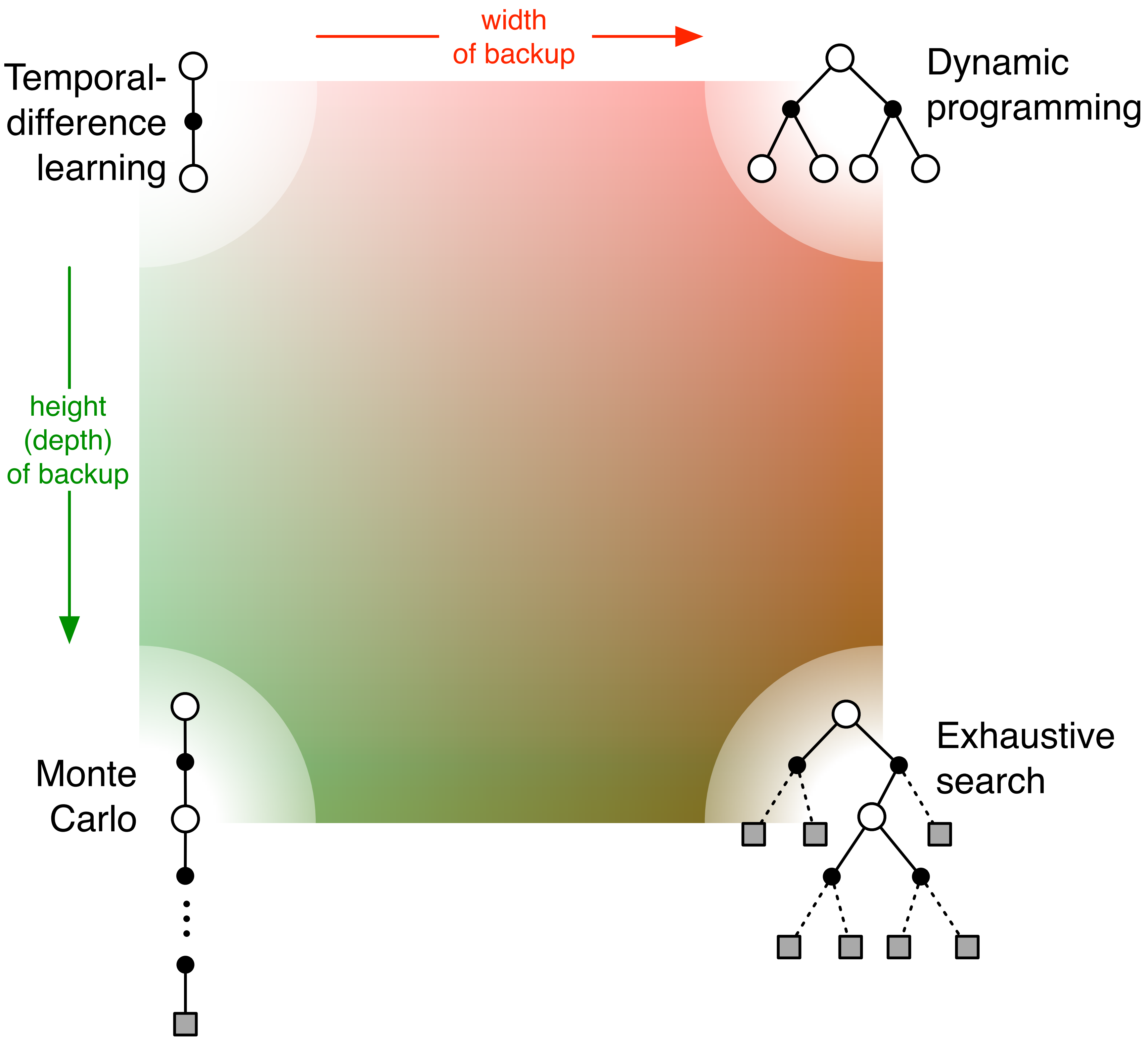
You are the Predictor

- The prediction that best matches the training data is V(A)=0
  - This minimizes the mean-square-error on the training set
  - This is what a batch Monte Carlo method gets
- If we consider the sequentiality of the problem, then we would set V(A)=.75
  - This is correct for the maximum likelihood estimate of a Markov model generating the data
  - i.e, if we do a best fit Markov model, and assume it is exactly correct, and then compute what it predicts (how?)
  - This is called the certainty-equivalence estimate
  - This is what TD gets

Summary so far

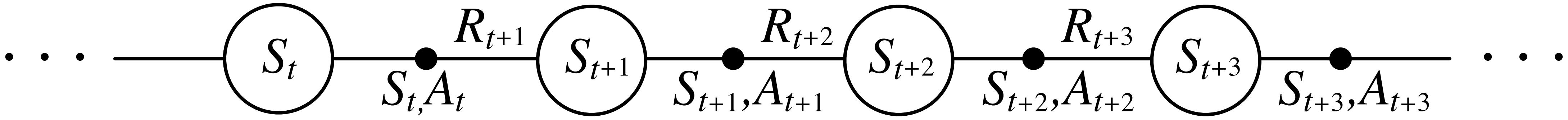
- Introduced *one-step tabular model-free TD methods*
- These methods bootstrap and sample, combining aspects of Dynamic Programming and MC methods
- TD methods are *computationally congenial*
- If the world is truly Markov, then TD methods will learn faster than MC methods
- MC methods have lower error on past data, but higher error on future data

Unified View



Learning An Action-Value Function

Estimate q_π for the current policy π



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

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

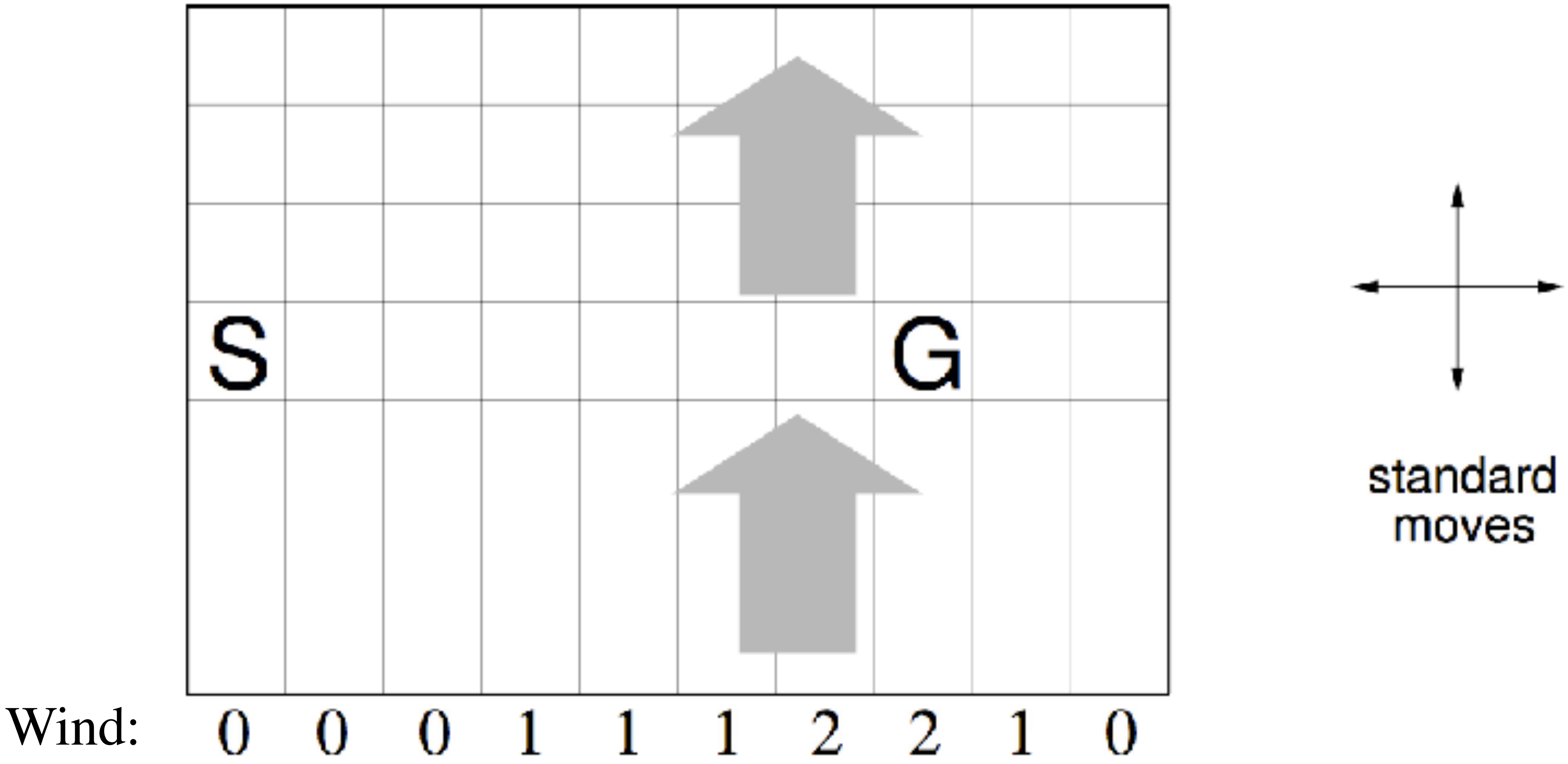
If S_{t+1} is terminal, then define $Q(S_{t+1}, A_{t+1}) = 0$

Sarsa: On-Policy TD Control

Turn this into a control method by always updating the policy to be greedy with respect to the current estimate:

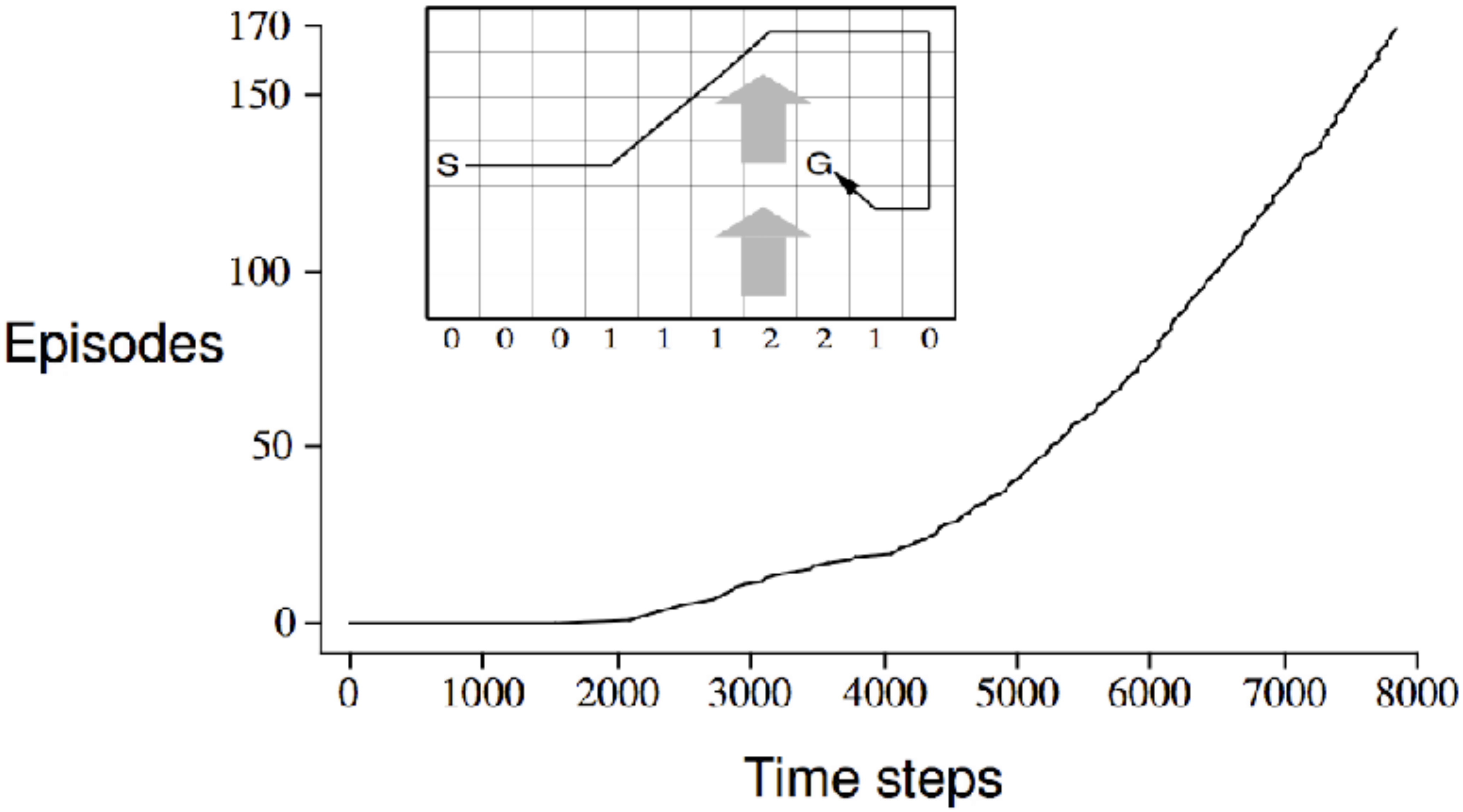
```
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Repeat (for each step of episode):
        Take action $A$, observe $R$, $S'$
        Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until $S$ is terminal
```

Windy Gridworld



undiscounted, episodic, reward = –1 until goal

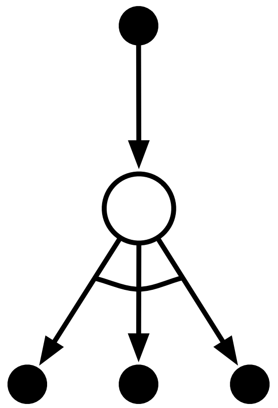
Results of Sarsa on the Windy Gridworld



Q-Learning: Off-Policy TD Control

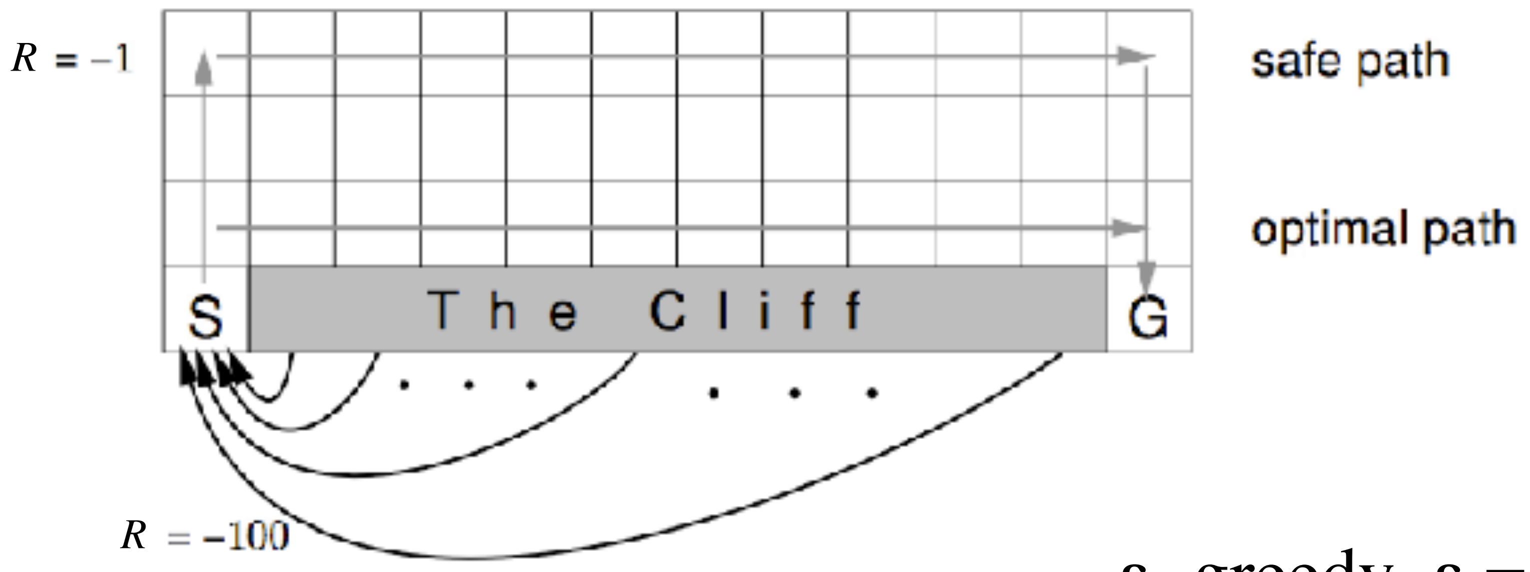
One-step Q-learning:

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

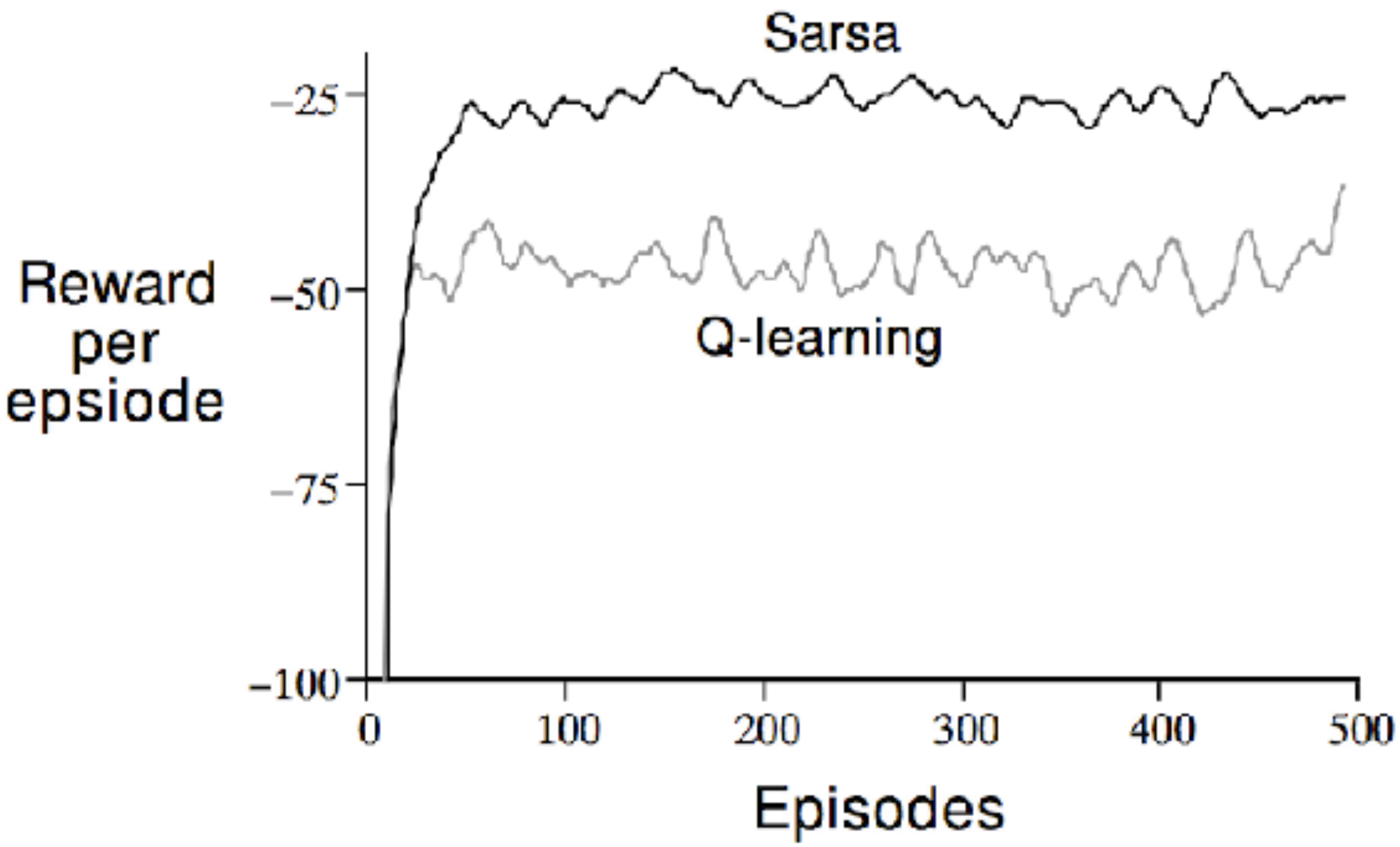


```
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize $S$
    Repeat (for each step of episode):
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        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
```

Cliffwalking



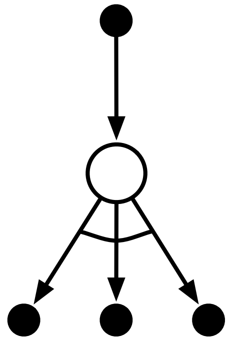
ε−greedy, ε = 0.1



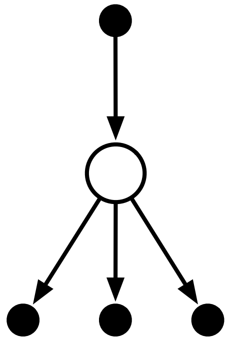
Expected Sarsa

- Instead of the *sample* value-of-next-state, use the expectation!

$$\begin{aligned} Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \mathbb{E}[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t)\Big] \\ &\leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)\Big] \end{aligned}$$



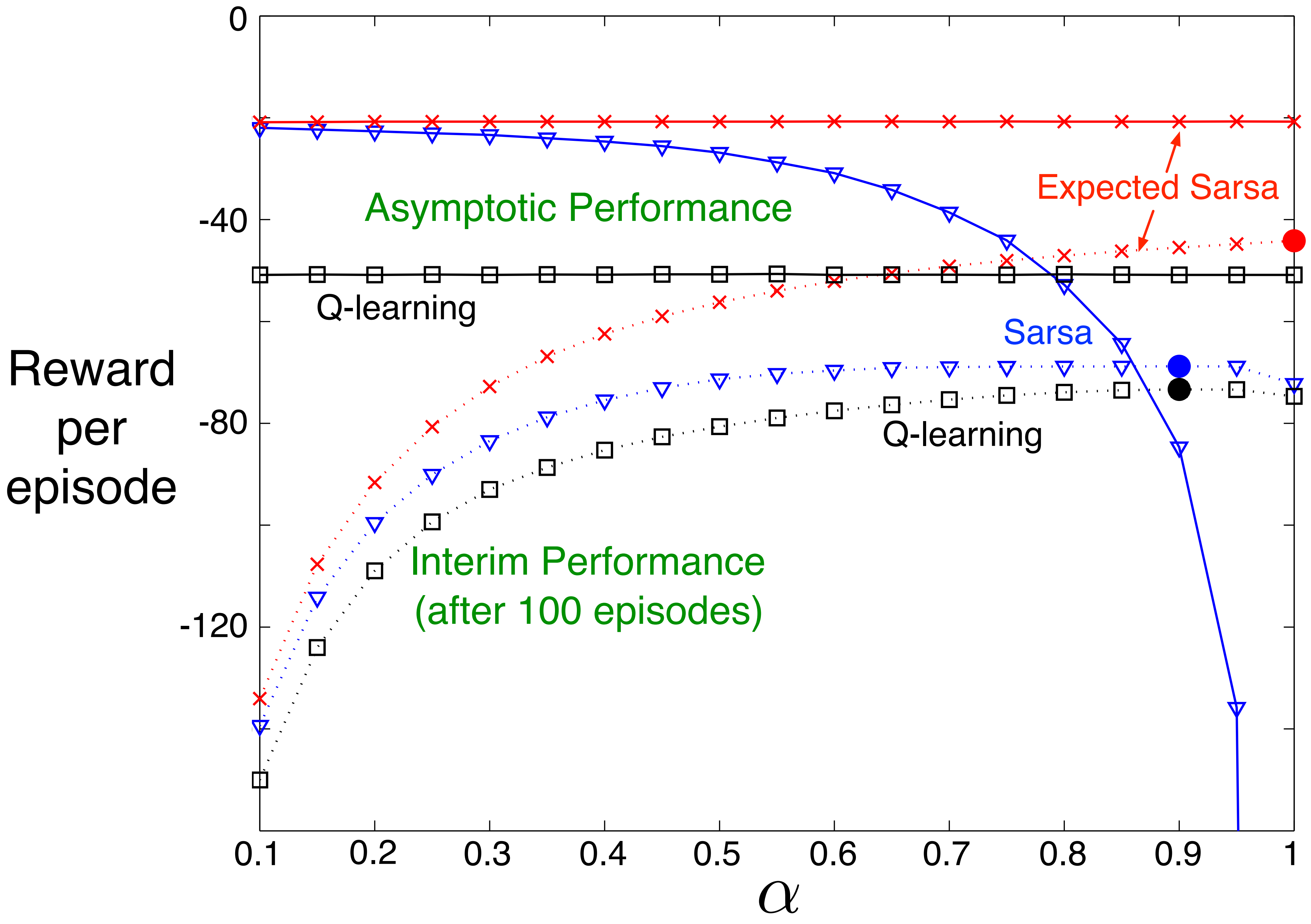
Q-learning



Expected Sarsa

- Expected Sarsa’s performs better than Sarsa (but costs more) van Seijen, van Hasselt, Whiteson, & Wiering 2009

Performance on the Cliff-walking Task

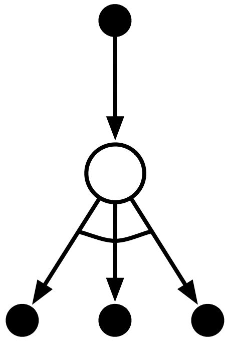


Off-policy Expected Sarsa

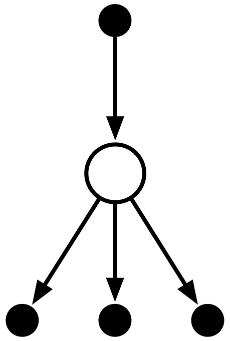
- Expected Sarsa generalizes to arbitrary behavior policies μ
  - in which case it includes Q-learning as the special case in which π is the greedy policy

$$
\begin{aligned}
Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \mathbb{E}[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t) \Big] \\
&\leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \Big]
\end{aligned}
$$

Nothing changes here



Q-learning



Expected Sarsa

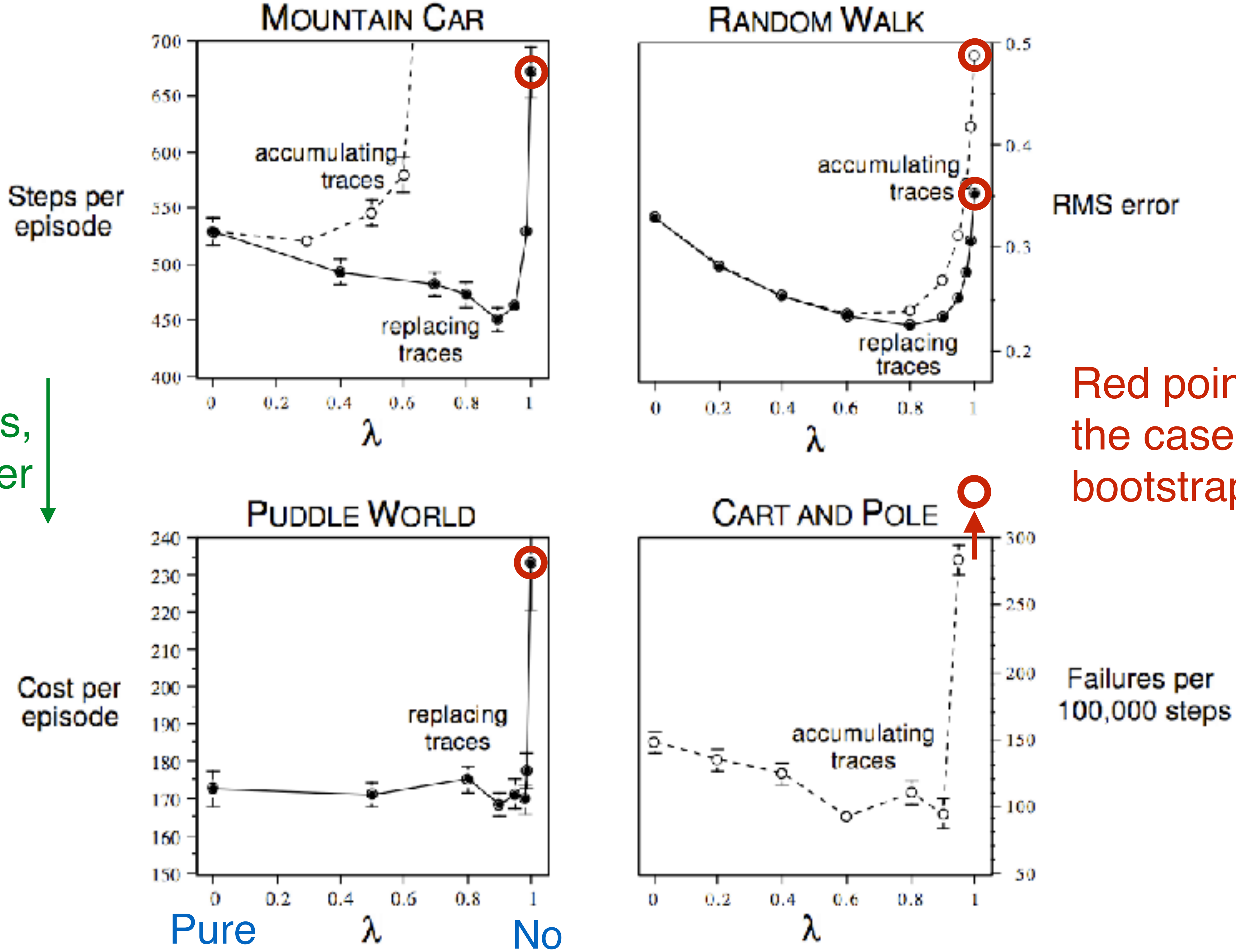
- This idea seems to be new

Summary

- Introduced *one-step tabular model-free TD methods*
- These methods bootstrap and sample, combining aspects of Dynamic Programming and MC methods
- TD methods are *computationally congenial*
- If the world is truly Markov, then TD methods will learn faster than MC methods
- MC methods have lower error on past data, but higher error on future data
- Extending prediction to control
  - On-policy control: Sarsa, Expected Sarsa
  - Off-policy control: Q-learning, Expected Sarsa
- Avoiding maximization bias with Double Q-learning

4 examples of the effect of bootstrapping

suggest that λ=1 (no bootstrapping) is a very poor choice
(i.e., Monte Carlo has high variance)



In all cases, lower is better

Red points are the cases of no bootstrapping

Pure bootstrapping

No bootstrapping

With linear function approximation, TD converges to the TD fixedpoint, $\boldsymbol{\theta}_{TD}$, a biased but interesting answer

parameter vector (weights)

transpose

feature vector for S_t

TD(0) update:

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha\Big(R_{t+1} + \gamma\boldsymbol{\theta}_t^\top\boldsymbol{\phi}_{t+1} - \boldsymbol{\theta}_t^\top\boldsymbol{\phi}_t\Big)\boldsymbol{\phi}_t$$
$$= \boldsymbol{\theta}_t + \alpha\Big(R_{t+1}\boldsymbol{\phi}_t - \boldsymbol{\phi}_t\big(\boldsymbol{\phi}_t - \gamma\boldsymbol{\phi}_{t+1}\big)^\top\boldsymbol{\theta}_t\Big)$$

Fixed-point analysis:

$$\mathbf{b} - \mathbf{A}\boldsymbol{\theta}_{TD} = \mathbf{0}$$
$$\Rightarrow \qquad \mathbf{b} = \mathbf{A}\boldsymbol{\theta}_{TD}$$
$$\Rightarrow \qquad \boldsymbol{\theta}_{TD} \doteq \mathbf{A}^{-1}\mathbf{b}$$

Guarantee:

$$\mathrm{MSVE}(\boldsymbol{\theta}_{TD}) \leq \frac{1}{1-\gamma}\min_{\boldsymbol{\theta}}\mathrm{MSVE}(\boldsymbol{\theta})$$

In expectation:

$$\mathbb{E}[\boldsymbol{\theta}_{t+1}|\boldsymbol{\theta}_t] = \boldsymbol{\theta}_t + \alpha(\mathbf{b} - \mathbf{A}\boldsymbol{\theta}_t),$$

where

$$\mathbf{b} \doteq \mathbb{E}[R_{t+1}\boldsymbol{\phi}_t] \in \mathbb{R}^n \quad \text{and} \quad \mathbf{A} \doteq \mathbb{E}\Big[\boldsymbol{\phi}_t\big(\boldsymbol{\phi}_t - \gamma\boldsymbol{\phi}_{t+1}\big)^\top\Big] \in \mathbb{R}^n \times \mathbb{R}^n$$

Frontiers of TD learning

- Off-policy prediction with linear function approx
- Non-linear function approximation
- Convergence theory for TD control methods
- Finite-time theory (beyond convergence)
- Combining with deep learning
    - e.g., is a replay buffer really necessary?
- Predicting myriad signals other than reward, as in Horde, Unreal, and option models

TD learning is a uniquely important kind of learning, maybe ubiquitous

- It is learning to predict, perhaps the only scalable kind of learning
- It is learning specialized for general, multi-step prediction, which may be key to perception, meaning, and modeling the world
- It takes advantage of the state property
  - which makes it fast, data efficient
  - which also makes it asymptotically biased
- It is computationally congenial
- We have just begun to use it for things other than reward