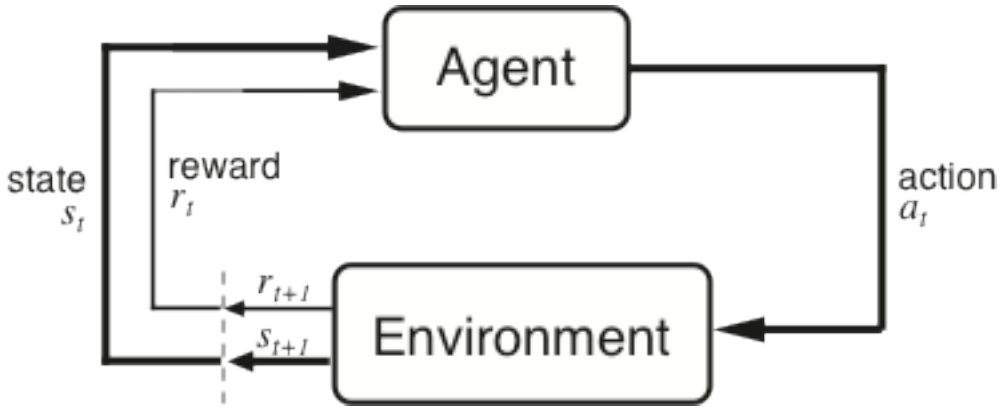
Reinforcement Learning

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

http://www.cs.ualberta.ca/~sutton/book/the-book.html
http://rlai.cs.ualberta.ca/RLAI/RLAIcourse/RLAIcourse.html

The Agent-Environment Interface



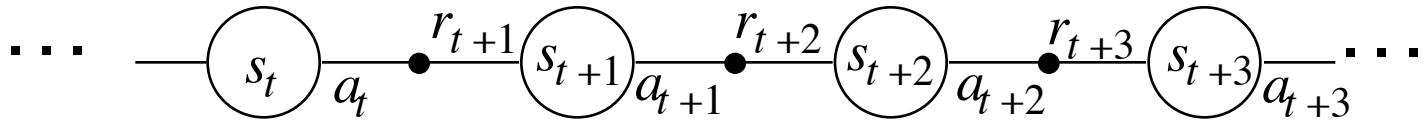
Agent and environment interact at discrete time steps: $t = 0, 1, 2, \ldots$

Agent observes state at step t: $s_t \in S$

produces action at step t: $a_t \in A(s_t)$

gets resulting reward: $r_{t+1} \in \mathbb{R}$

and resulting next state: s_{t+1}



The Agent Learns a Policy

Policy at step t, π_t :

a mapping from states to action probabilities

$\pi_t(s, a)$ = probability that $a_t = a$ when $s_t = s$

- Reinforcement learning methods specify how the agent changes its policy as a result of experience.
- Roughly, the agent's goal is to get as much reward as it can over the long run.

Getting the Degree of Abstraction Right

- Time steps need not refer to fixed intervals of real time.
- Actions can be low level (e.g., voltages to motors), or high level (e.g., accept a job offer), "mental" (e.g., shift in focus of attention), etc.
- States can be low-level "sensations", or they can be abstract, symbolic, based on memory, or subjective (e.g., the state of being "surprised" or "lost").
- An RL agent is not like a whole animal or robot, which consist of many RL agents as well as other components.
- The environment is not necessarily unknown to the agent, only incompletely controllable.
- Reward computation is in the agent's environment because the agent cannot change it arbitrarily.

Goals and Rewards

- Is a scalar reward signal an adequate notion of a goal?—maybe not, but it is surprisingly flexible.
- A goal should specify **what** we want to achieve, not **how** we want to achieve it.
- A goal must be outside the agent's direct control—thus outside the agent.
- The agent must be able to measure success:
  - explicitly;
  - frequently during its lifespan.

Returns

Suppose the sequence of rewards after step t is :

$$r_{t+1}, r_{t+2}, r_{t+3}, \ldots$$

What do we want to maximize?

In general,

we want to maximize the **expected return**, $E\{R_t\}$, for each step t.

Episodic tasks: interaction breaks naturally into episodes, e.g., plays of a game, trips through a maze.

$$R_t = r_{t+1} + r_{t+2} + \cdots + r_T,$$

where T is a final time step at which a **terminal state** is reached, ending an episode.

Returns for Continuing Tasks

Continuing tasks: interaction does not have natural episodes.

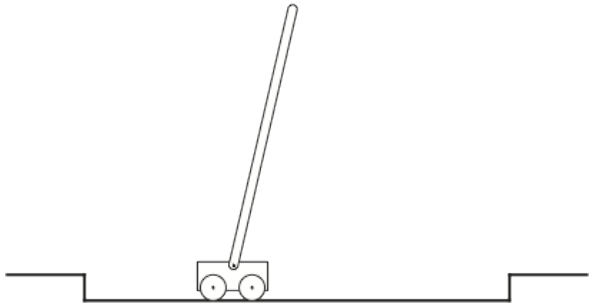
Discounted return:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1},$$

where γ, $0 \le \gamma \le 1$, is the **discount rate**

shortsighted $0 \leftarrow \gamma \rightarrow 1$ farsighted

An Example



Avoid **failure:** the pole falling beyond a critical angle or the cart hitting end of track.

As an **episodic task** where episode ends upon failure:

$$\text{reward} = +1 \text{ for each step before failure}$$

$$\Rightarrow \text{return} = \text{number of steps before failure}$$

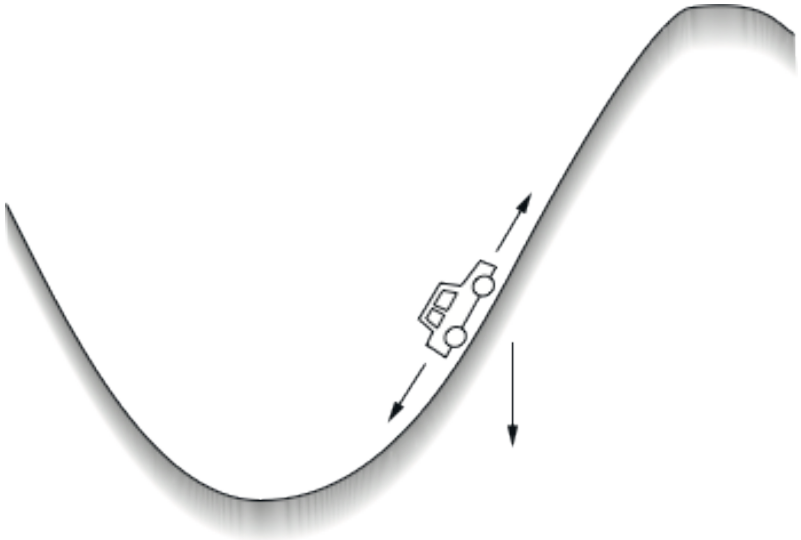
As a **continuing task** with discounted return:

$$\text{reward} = -1 \text{ upon failure; } 0 \text{ otherwise}$$

$$\Rightarrow \text{return} = -\gamma^k, \text{ for } k \text{ steps before failure}$$

In either case, return is maximized by avoiding failure for as long as possible.

Another Example



Get to the top of the hill
as quickly as possible.

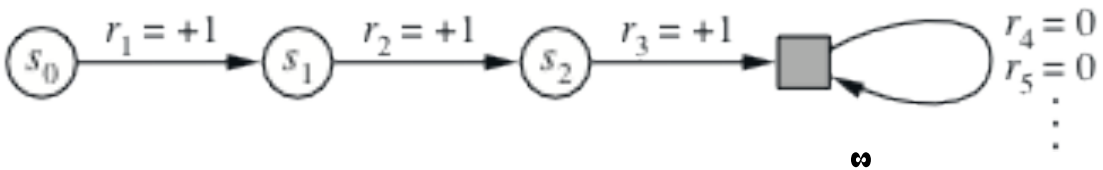
reward = −1 for each step where **not** at top of hill

$\Rightarrow$ return = − number of steps before reaching top of hill

Return is maximized by minimizing
number of steps reach the top of the hill.

A Unified Notation

- In episodic tasks, we number the time steps of each episode starting from zero.
- We usually do not have distinguish between episodes, so we write s_t instead of $s_{t,j}$ for the state at step t of episode j.
- Think of each episode as ending in an absorbing state that always produces reward of zero:



- We can cover all cases by writing $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$,

where γ can be 1 only if a zero reward absorbing state is always reached.

The Markov Property

- By "the state" at step t, the book means whatever information is available to the agent at step t about its environment.
- The state can include immediate "sensations," highly processed sensations, and structures built up over time from sequences of sensations.
- Ideally, a state should summarize past sensations so as to retain all "essential" information, i.e., it should have the **Markov Property**:

$$\Pr\left\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0\right\} =$$
$$\Pr\left\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t\right\}$$

for all s', r, and histories $s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0$.

Markov Decision Processes

- If a reinforcement learning task has the Markov Property, it is basically a **Markov Decision Process (MDP)**.
- If state and action sets are finite, it is a **finite MDP**.
- To define a finite MDP, you need to give:
  - **state and action sets**
  - one-step "dynamics" defined by **transition probabilities**:

$$P^a_{ss'} = \Pr\left\{s_{t+1} = s' \mid s_t = s, a_t = a\right\} \quad \text{for all } s, s' \in S,\ a \in A(s).$$

  - **reward probabilities**:

$$R^a_{ss'} = E\left\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\right\} \quad \text{for all } s, s' \in S,\ a \in A(s).$$

An Example Finite MDP

Recycling Robot

- At each step, robot has to decide whether it should (1) actively search for a can, (2) wait for someone to bring it a can, or (3) go to home base and recharge.
- Searching is better but runs down the battery; if runs out of power while searching, has to be rescued (which is bad).
- Decisions made on basis of current energy level: `high`, `low`.
- Reward = number of cans collected

Value Functions

- The **value of a state** is the expected return starting from that state; depends on the agent's policy:

State-value function for policy π:

$$V^{\pi}(s) = E_{\pi}\left\{ R_t \mid s_t = s \right\} = E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s \right]$$

- The **value of taking an action in a state under policy π** is the expected return starting from that state, taking that action, and thereafter following π:

Action-value function for policy π:

$$Q^{\pi}(s, a) = E_{\pi}\left\{ R_t \mid s_t = s, a_t = a \right\} = E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a \right]$$

Bellman Equation for a Policy π

The basic idea:

$$\begin{aligned} R_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \cdots \\ &= r_{t+1} + \gamma \left(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} \cdots \right) \\ &= r_{t+1} + \gamma R_{t+1} \end{aligned}$$

So:

$$\begin{aligned} V^\pi(s) &= E_\pi \left\{ R_t \mid s_t = s \right\} \\ &= E_\pi \left\{ r_{t+1} + \gamma V(s_{t+1}) \mid s_t = s \right\} \end{aligned}$$

Or, without the expectation operator:

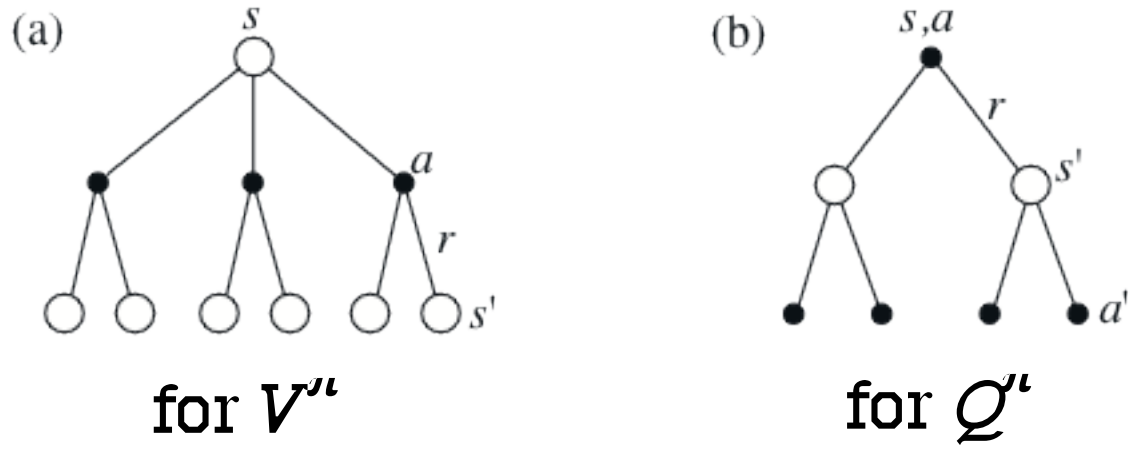
$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^\pi(s') \right]$$

More on the Bellman Equation

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'} \left[R^{a}_{ss'} + \gamma V^{\pi}(s') \right]$$

This is a set of equations (in fact, linear), one for each state.
The value function for π is its unique solution.

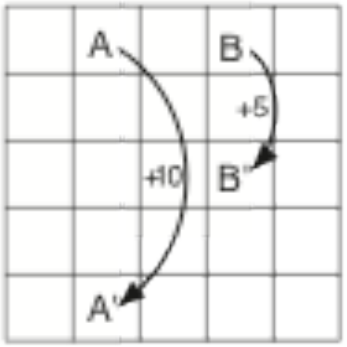
Backup diagrams:



for V^{π} for Q^{π}

Gridworld

- Actions: `north`, `south`, `east`, `west`; deterministic.
- If would take agent off the grid: no move but reward = –1
- Other actions produce reward = 0, except actions that move agent out of special states A and B as shown.



3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

State-value function for equiprobable random policy; $\gamma = 0.9$

Optimal Value Functions

- For finite MDPs, policies can be **partially ordered**:

$$\pi \geq \pi' \quad \text{if and only if} \quad V^{\pi}(s) \geq V^{\pi'}(s) \quad \text{for all } s \in S$$

- There is always at least one (and possibly many) policies that is better than or equal to all the others. This is an **optimal policy**. We denote them all π^*.
- Optimal policies share the same **optimal state-value function**:

$$V^*(s) = \max_{\pi} V^{\pi}(s) \quad \text{for all } s \in S$$

- Optimal policies also share the same **optimal action-value function**:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \quad \text{for all } s \in S \text{ and } a \in A(s)$$

This is the expected return for taking action a in state s and thereafter following an optimal policy.

Bellman Optimality Equation for V^*

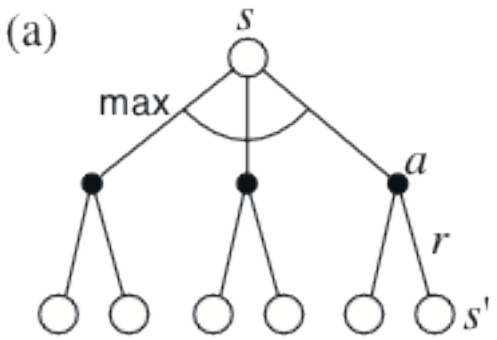
The value of a state under an optimal policy must equal the expected return for the best action from that state:

$$V^*(s) = \max_{a \in A(s)} Q^{\pi^*}(s,a)$$

$$= \max_{a \in A(s)} E\left\{ r_{t+1} + \gamma V^*(s_{t+1}) \,\middle|\, s_t = s, a_t = a \right\}$$

$$= \max_{a \in A(s)} \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^*(s') \right]$$

The relevant backup diagram:



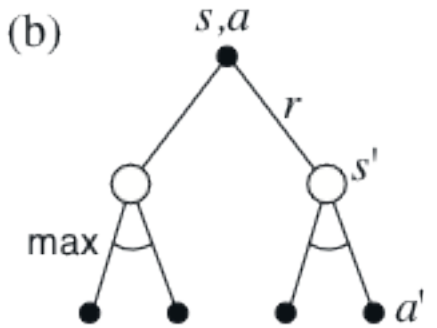
V^* is the unique solution of this system of nonlinear equations.

Bellman Optimality Equation for Q^*

$$Q^*(s,a) = E\left\{ r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \,\middle|\, s_t = s, a_t = a \right\}$$

$$= \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma \max_{a'} Q^*(s', a') \right]$$

The relevant backup diagram:



Q^* is the unique solution of this system of nonlinear equations.

Why Optimal State-Value Functions are Useful

Any policy that is greedy with respect to V^* is an optimal policy.

Therefore, given V^*, one-step-ahead search produces the long-term optimal actions.

E.g., back to the gridworld:

a) gridworld b) V^* c) π^*

What About Optimal Action-Value Functions?

Given Q^*, the agent does not even have to do a one-step-ahead search:

$$\pi^*(s) = \arg\max_{a \in A(s)} Q^*(s, a)$$

Solving the Bellman Optimality Equation

- Finding an optimal policy by solving the Bellman Optimality Equation requires the following:
  - accurate knowledge of environment dynamics;
  - we have enough space an time to do the computation;
  - the Markov Property.
- How much space and time do we need?
  - polynomial in number of states (via dynamic programming methods; Chapter 4),
  - BUT, number of states is often huge (e.g., backgammon has about 10**20 states).
- We usually have to settle for approximations.
- Many RL methods can be understood as approximately solving the Bellman Optimality Equation.

TD Prediction

Policy Evaluation (the prediction problem):
for a given policy π, compute the state-value function V^{π}

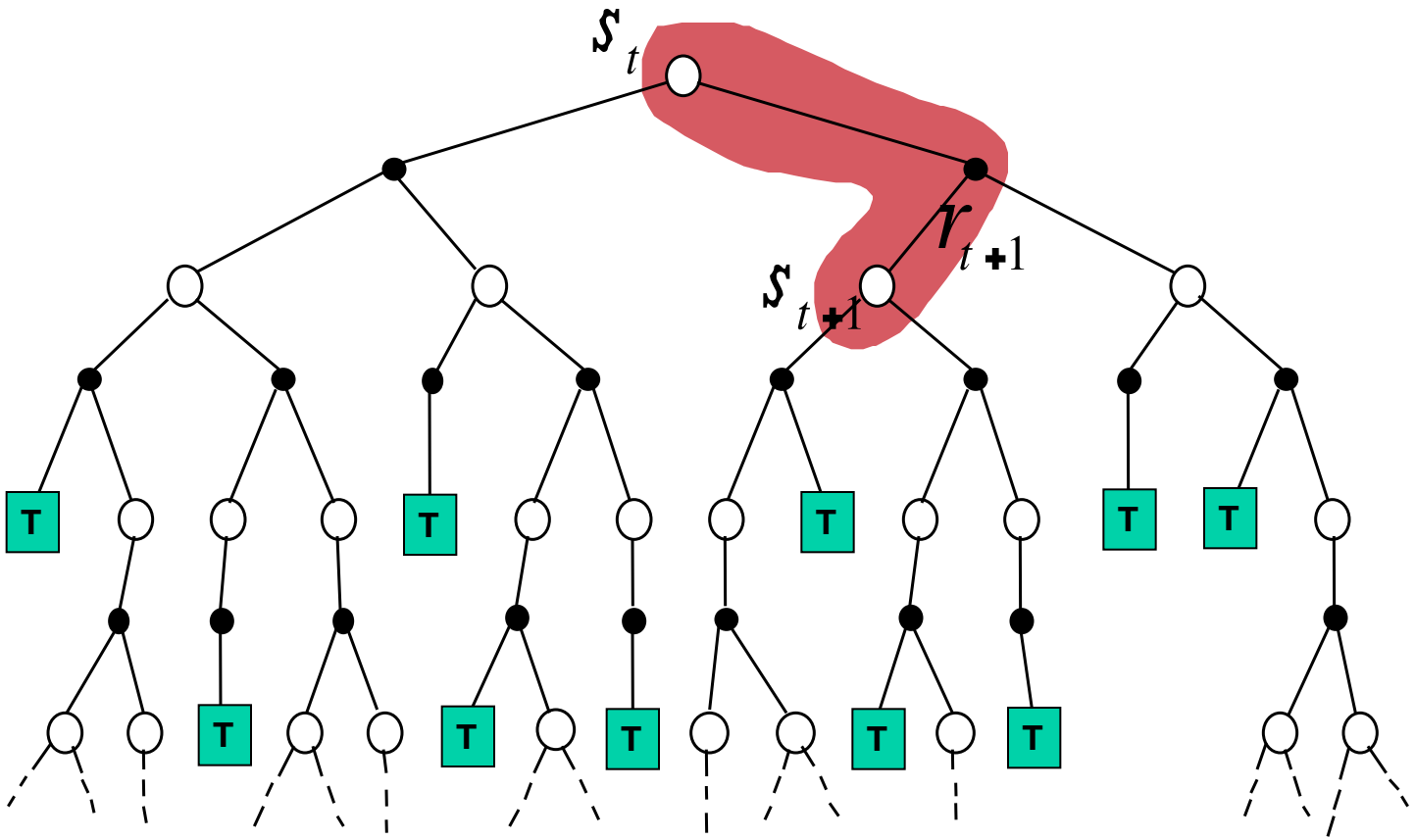
The simplest TD method, TD(0) :

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

target: an estimate of the return

Simplest TD Method

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

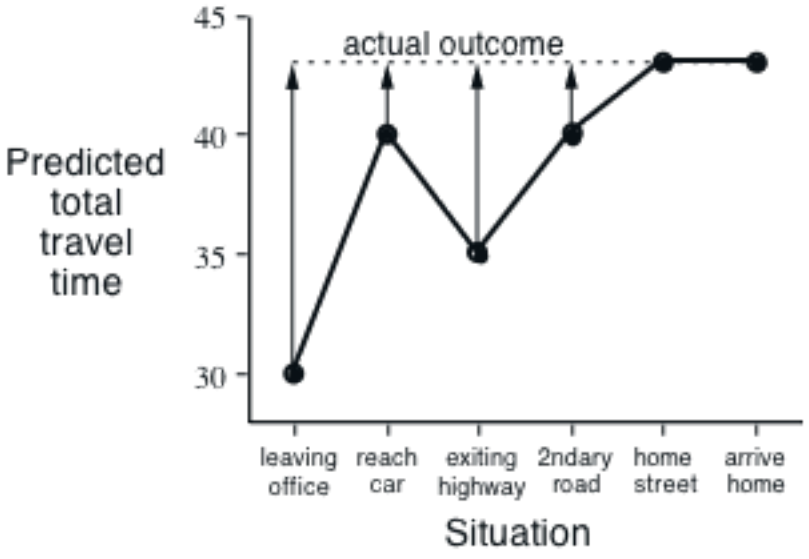


Example: Driving Home

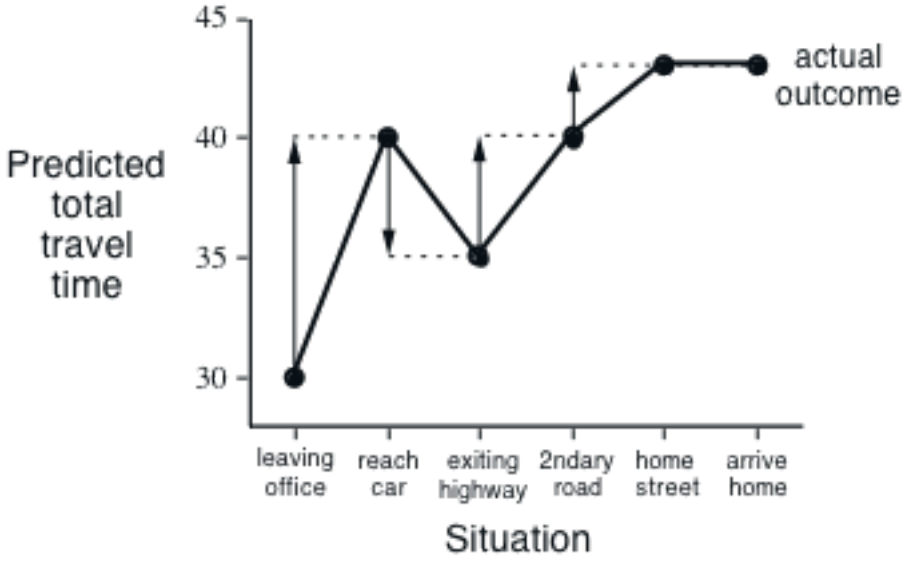
State	Elapsed Time (minutes)	Predicted Time to Go	Predicted Total Time
leaving office	0	30	30
reach car, raining	5	35	40
exit highway	20	15	35
behind truck	30	10	40
home street	40	3	43
arrive home	43	0	43

Driving Home

Changes recommended by Monte Carlo methods (α=1)



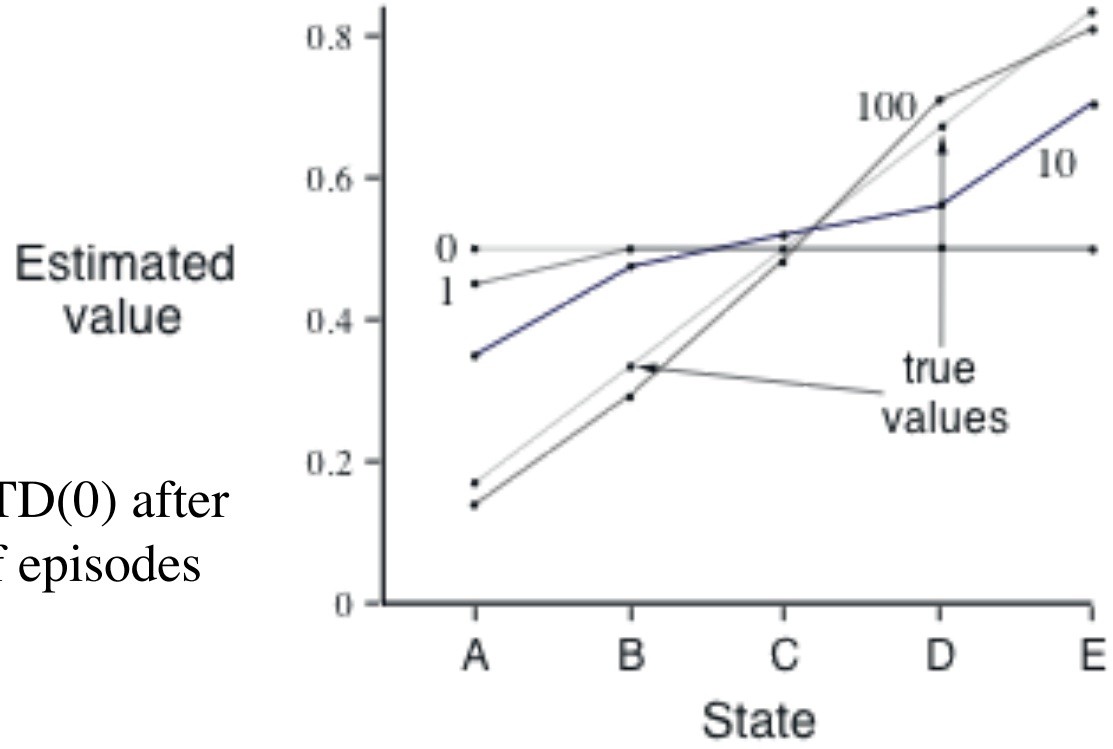
Changes recommended by TD methods (α=1)



Advantages of TD Learning

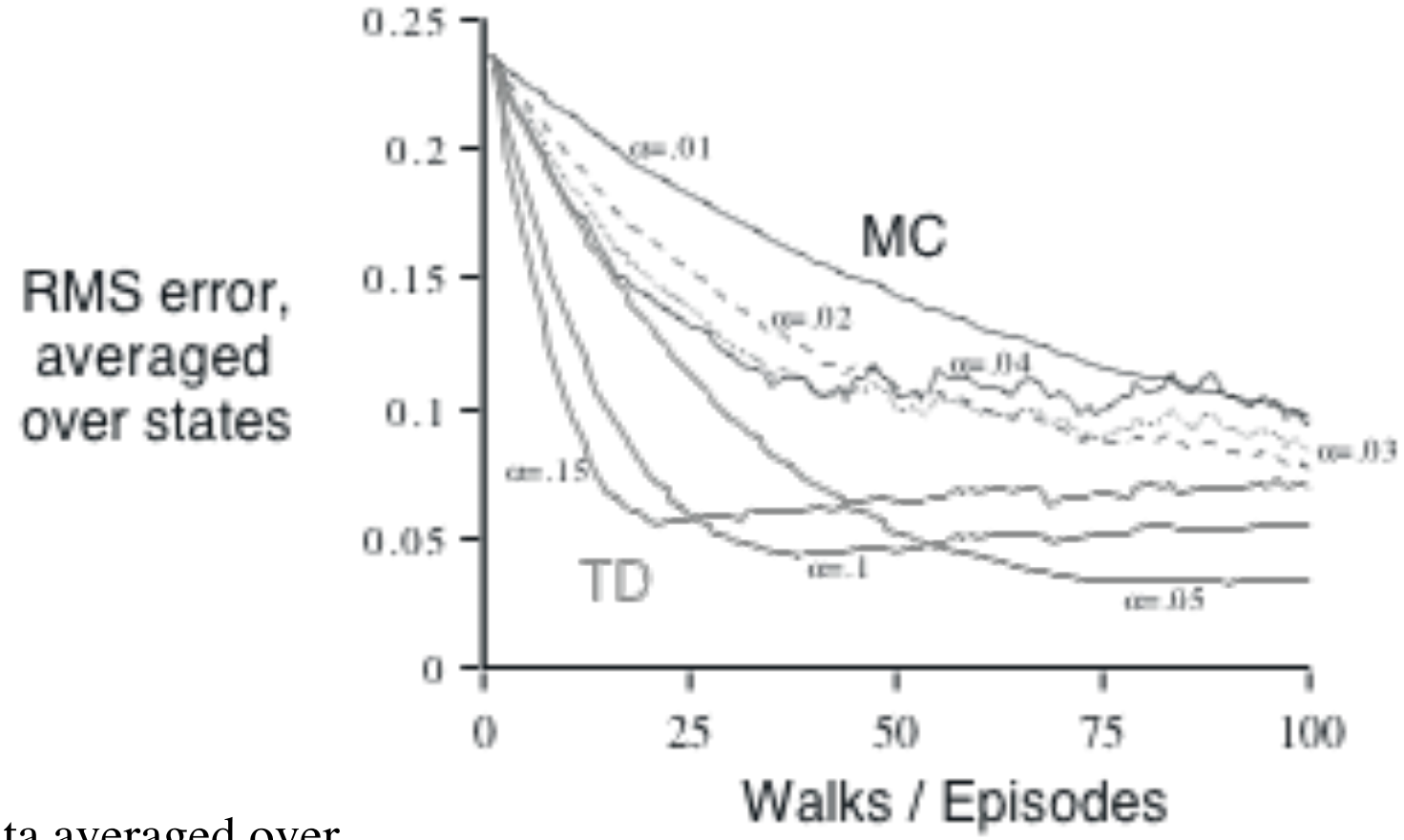
- TD methods do not require a model of the environment, only experience
- TD methods can be fully incremental
  - You can learn before knowing the final outcome
    - Less memory
    - Less peak computation
  - You can learn without the final outcome
    - From incomplete sequences

Random Walk Example



Values learned by TD(0) after various numbers of episodes

TD and MC on the Random Walk



Data averaged over
100 sequences of episodes

Optimality of TD(0)

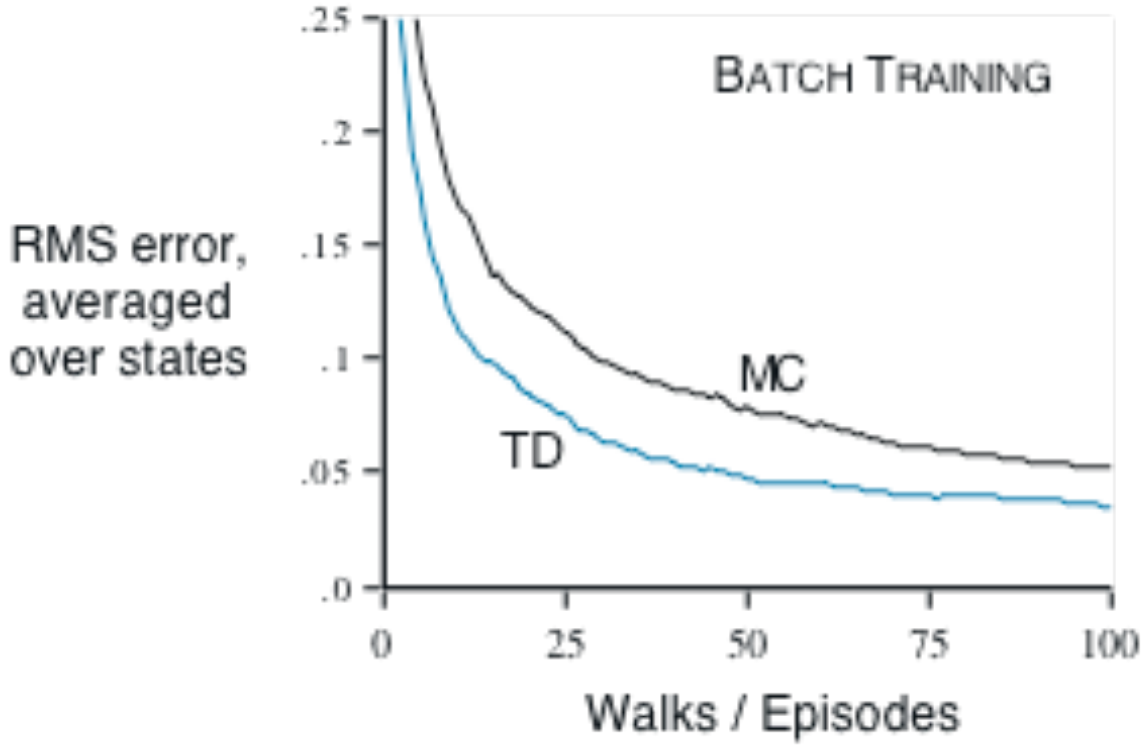
Batch Updating: train completely on a finite amount of data,
e.g., train repeatedly on 10 episodes until convergence.

Compute updates according to TD(0), but only update estimates after each complete pass through the data.

For any finite Markov prediction task, under batch updating, TD(0) converges for sufficiently small α.

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

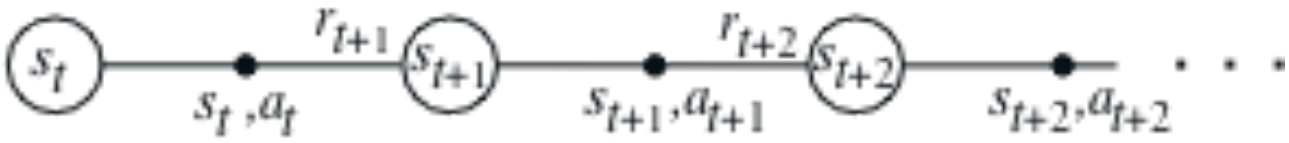
Random Walk under Batch Updating



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

Learning An Action-Value Function: Q-Learning

Estimate Q^{π} for the current behavior policy π.



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

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

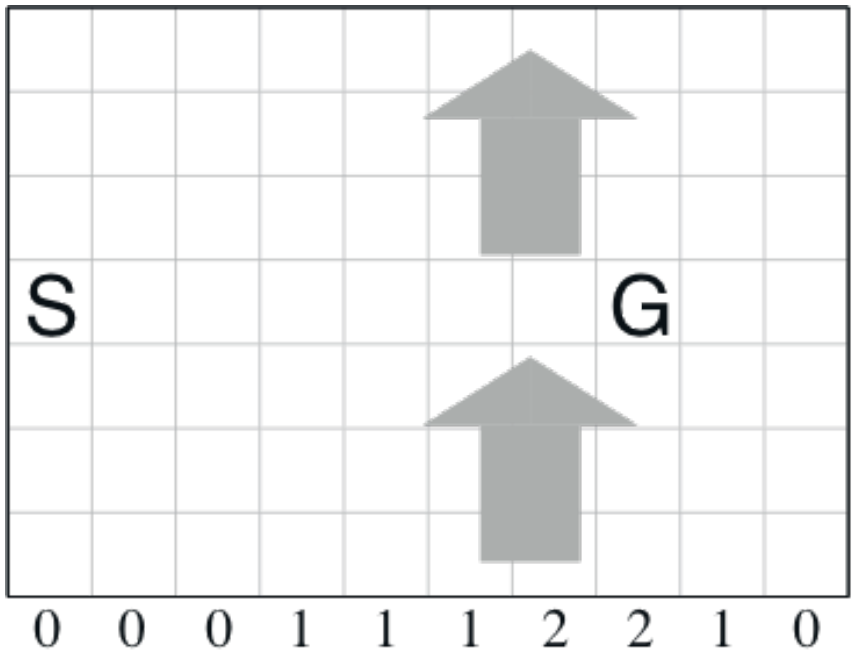
If s_{t+1} is terminal, then $Q(s_{t+1}, a_{t+1}) = 0$.

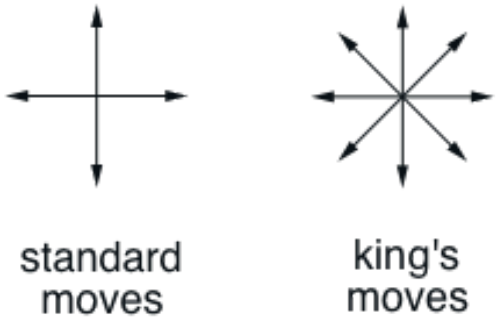
Sarsa: On-Policy TD Control

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

Initialize $Q(s, a)$ 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

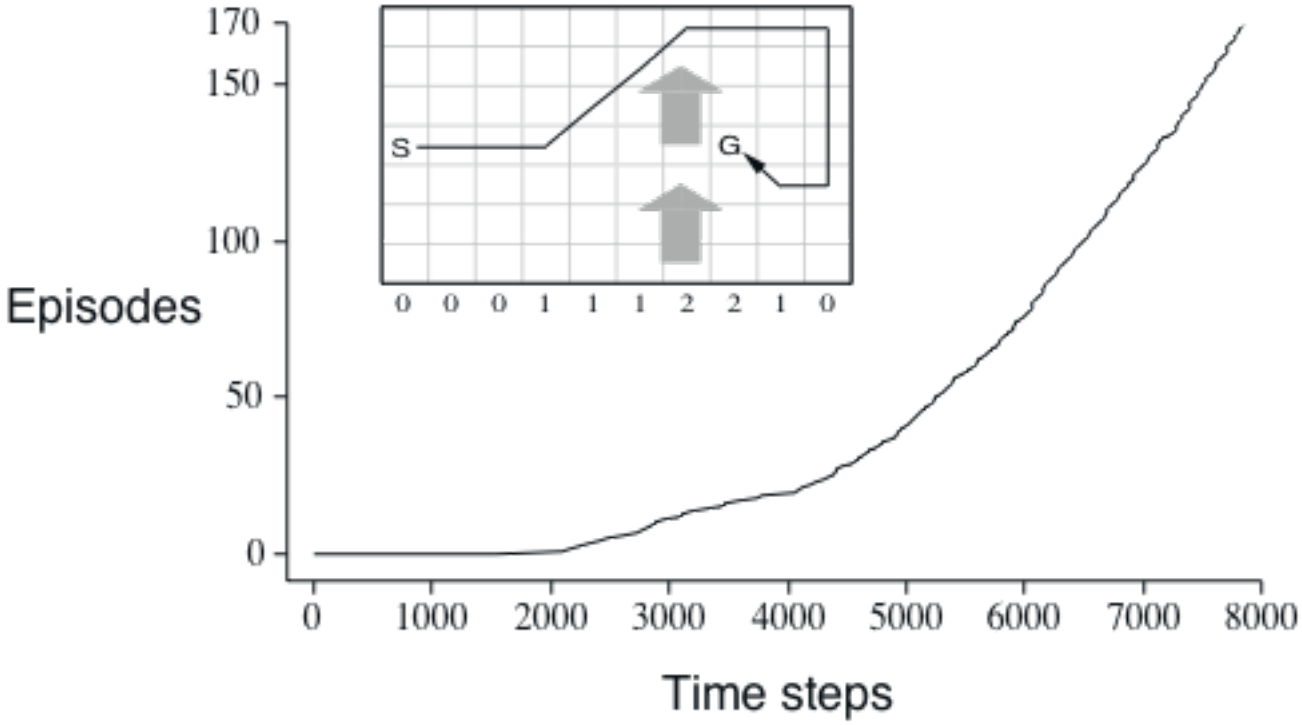
Windy Gridworld



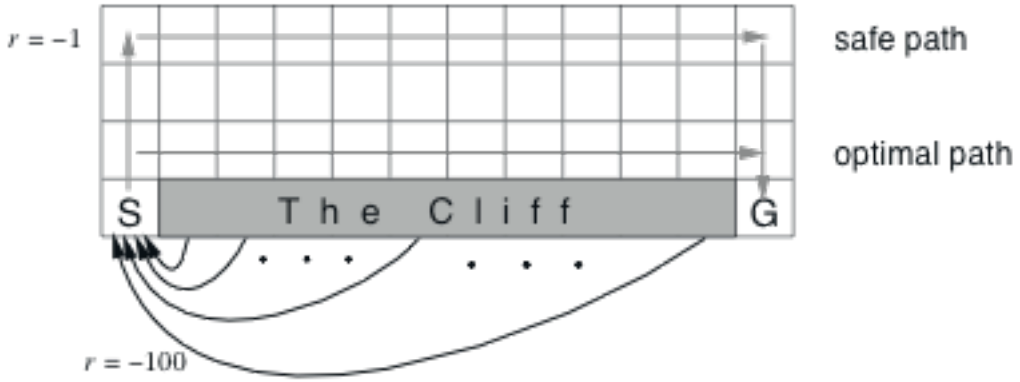


undiscounted, episodic, reward = –1 until goal

Results of Sarsa on the Windy Gridworld



Cliffwalking



ε−greedy, ε = 0.1