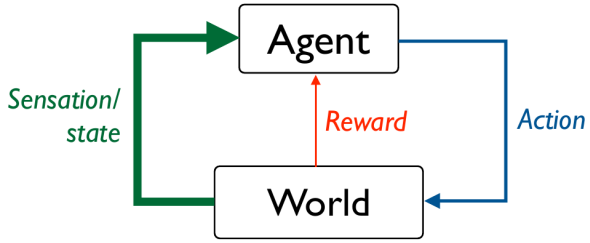


- Learning that requires less input from people
- AI that can learn for itself, during its normal operation

Reinforcement Learning
Tabular Methods

Slides from or based on the 2nd edition of the book by Sutton and Barto

http://www.incompleteideas.net/book/the-book.html

See also Chapter 23 of the AIMA textbook

Last modified on 2022/05/14
by f.maire@qut.edu.au

Key concepts

- Agent-environment interaction
  - States
  - Actions
  - Rewards
- Policy: stochastic rule for selecting actions
- Return: the function of future rewards the agent tries to maximize
- Episodic and continuing tasks
- Markov Decision Process
  - Transition probabilities
  - Expected rewards
- Value functions
  - State-value function for a policy
  - Action-value function for a policy
  - Optimal state-value function
  - Optimal action-value function
- Optimal value functions
- Optimal policies
- Bellman Equations
- The need for approximation
- Learning algorithms
  - SARSA
  - Q-learning

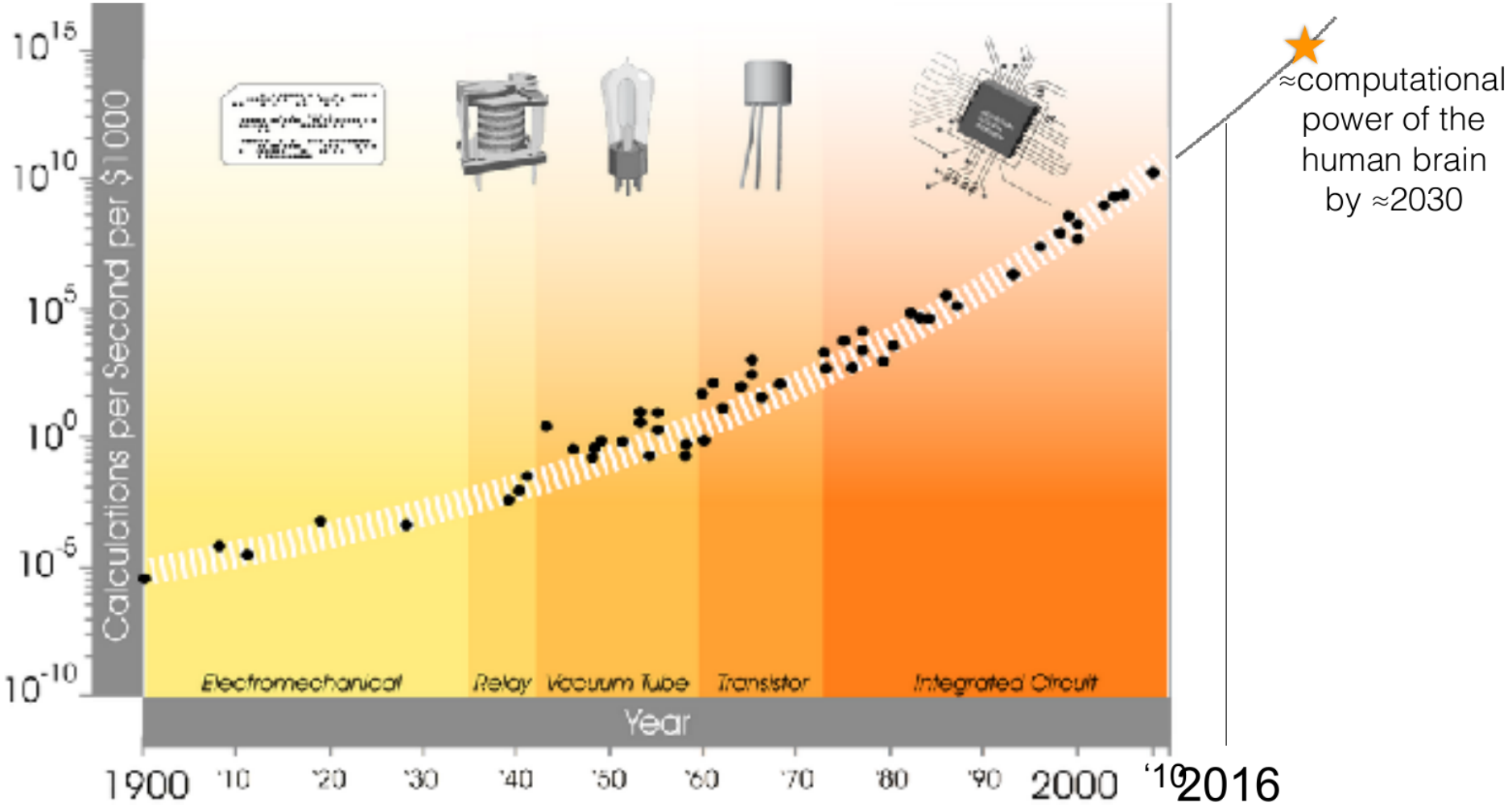
Outline

- Recent breakthroughs in AI/RL
- Multi-arm bandits
- The RL interface
- Temporal Difference (TD) error
- State-Value, Action-Value functions
- Bellman Equations
- Learning algorithms
  - Policy evaluation, policy improvement, policy iteration
  - Value iteration
  - Dynamic Programming, Monte Carlo, TD method
  - SARSA, Q-learning

The 2nd industrial revolution

- The 1st industrial revolution was the *physical power* of machines substituting for that of people
- The 2nd industrial revolution is the *computational power* of machines substituting for that of people
  - Computation for perception, motor control, prediction, decision making, optimization, search
  - Until now, people have been our cheapest source of computation
  - But now our machines are starting to provide greater, cheaper computation

The computational revolution



Kurzweil AI

About *Intelligence*

1. *Intelligence is the computational part of the ability to achieve goals*
   - looking deeper: 1) its a continuum, 2) its an appearance, 3) it varies with observer and purpose

2. We will (probably) figure out how to make intelligent systems in our lifetimes; it will change everything

3. But prior to that it will probably change our careers
   - as companies gear up to take advantage of the economic opportunities

Advances in AI abilities are coming faster

- IBM's Watson beats the best human players of *Jeopardy!* (2011)
- Deep neural networks greatly improve the state of the art in speech recognition and computer vision (2012–)
- Google's self-driving car becomes a plausible reality (≈2013)
- Deepmind's DQN learns to play Atari games at the human level, from pixels, with no game-specific knowledge (≈2014, *Nature*)
- University of Alberta program solves Limit Poker (2015, Science), and then defeats professional players at No-limit Poker (2017, Science)
- Google Deepmind's AlphaGo defeats legendary Go player Lee Sedol (2016, Nature), and world champion Ke Jie (2017), vastly improving over all previous programs

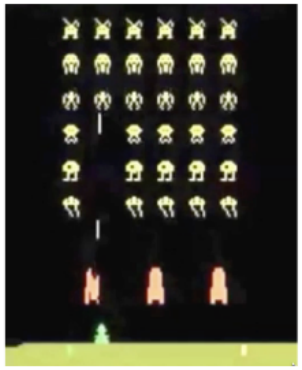
RL + Deep Learing Performance on Atari Games

Space Invaders

Breakout

Enduro

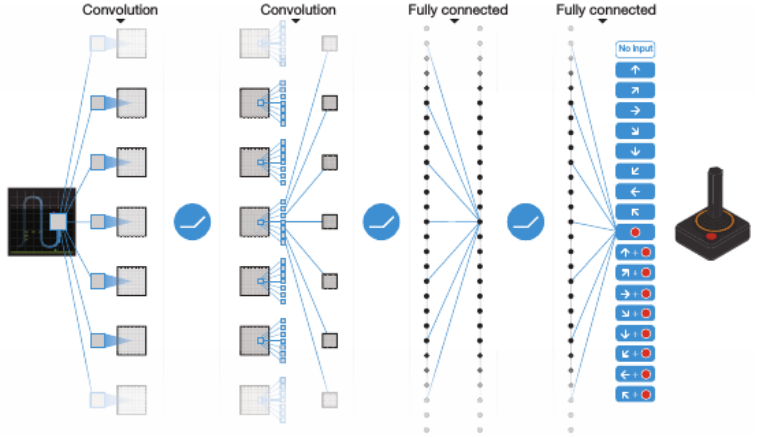
RL + Deep Learning, applied to Classic Atari Games





- 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

Cheap computation power drives progress in AI

- Deep learning algorithms are essentially the same as what was used in ‘80s
  - only now with larger computers (GPUs) and larger data sets
  - enabling today’s vastly improved speech recognition
- Similar impacts of computer power can be seen in recent years, and throughout AI’s history, in natural language processing, computer vision, and computer chess, Go, and other games

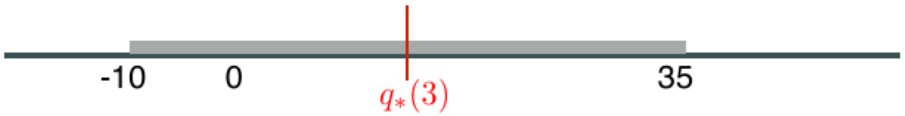
Multi-arm Bandits

Sutton and Barto, Chapter 2

The simplest reinforcement learning problem

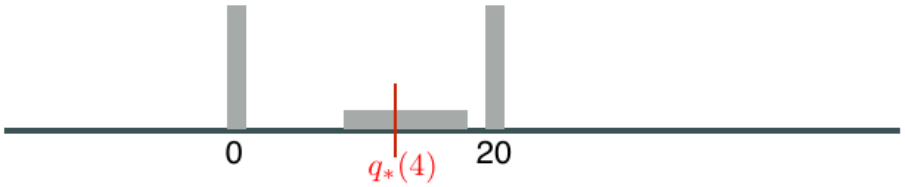
You are the algorithm!

- Action 1 — Reward is always 8
  - value of action 1 is $q_*(1) =$
- Action 2 — 88% chance of 0, 12% chance of 100!
  - value of action 2 is $q_*(2) = .88 \times 0 + .12 \times 100 =$
- Action 3 — Randomly between -10 and 35, equiprobable



$q_*(3) =$

- Action 4 — a third 0, a third 20, and a third from {8,9,…, 18}



$q_*(4) =$

The k-armed Bandit Problem

- On each of an infinite sequence of *time steps*, $t=1, 2, 3, \ldots$, you choose an action A_t from k possibilities, and receive a real-valued *reward* R_t

- The reward depends only on the action taken; it is indentically, independently distributed (i.i.d.):

$$q_*(a) \doteq \mathbb{E}[R_t | A_t = a], \quad \forall a \in \{1, \ldots, k\}$$ *true values*

- These true values are *unknown*. The distribution is unknown

- Nevertheless, you must maximize your total reward

- You must both try actions to learn their values (explore), and prefer those that appear best (exploit)

The Exploration/Exploitation Dilemma

- Suppose you form estimates

$$Q_t(a) \approx q_*(a), \quad \forall a$$ *action-value estimates*

- Define the *greedy action* at time *t* as

$$A_t^* \doteq \arg\max_a Q_t(a)$$

- If $A_t = A_t^*$ then you are *exploiting*
  If $A_t \neq A_t^*$ then you are *exploring*

- You can't do both, but you need to do both

- You can never stop exploring, but maybe you should explore less with time. Or maybe not.

Action-Value Methods

- Methods that learn action-value estimates and nothing else
- For example, estimate action values as *sample averages*:

$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbf{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbf{1}_{A_i=a}}$$

- The sample-average estimates converge to the true values
  If the action is taken an infinite number of times

$$\lim_{N_t(a)\to\infty} Q_t(a) = q_*(a)$$

The number of times action a has been taken by time t

ε-Greedy Action Selection

- In greedy action selection, you always exploit
- In ε-greedy, you are usually greedy, but with probability ε you instead pick an action at random (possibly the greedy action again)
- This is perhaps the simplest way to balance exploration and exploitation

A simple bandit algorithm

Initialize, for $a = 1$ to k:

$Q(a) \leftarrow 0$

$N(a) \leftarrow 0$

Repeat forever:

$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

$R \leftarrow bandit(A)$

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

$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\big[R - Q(A)\big]$

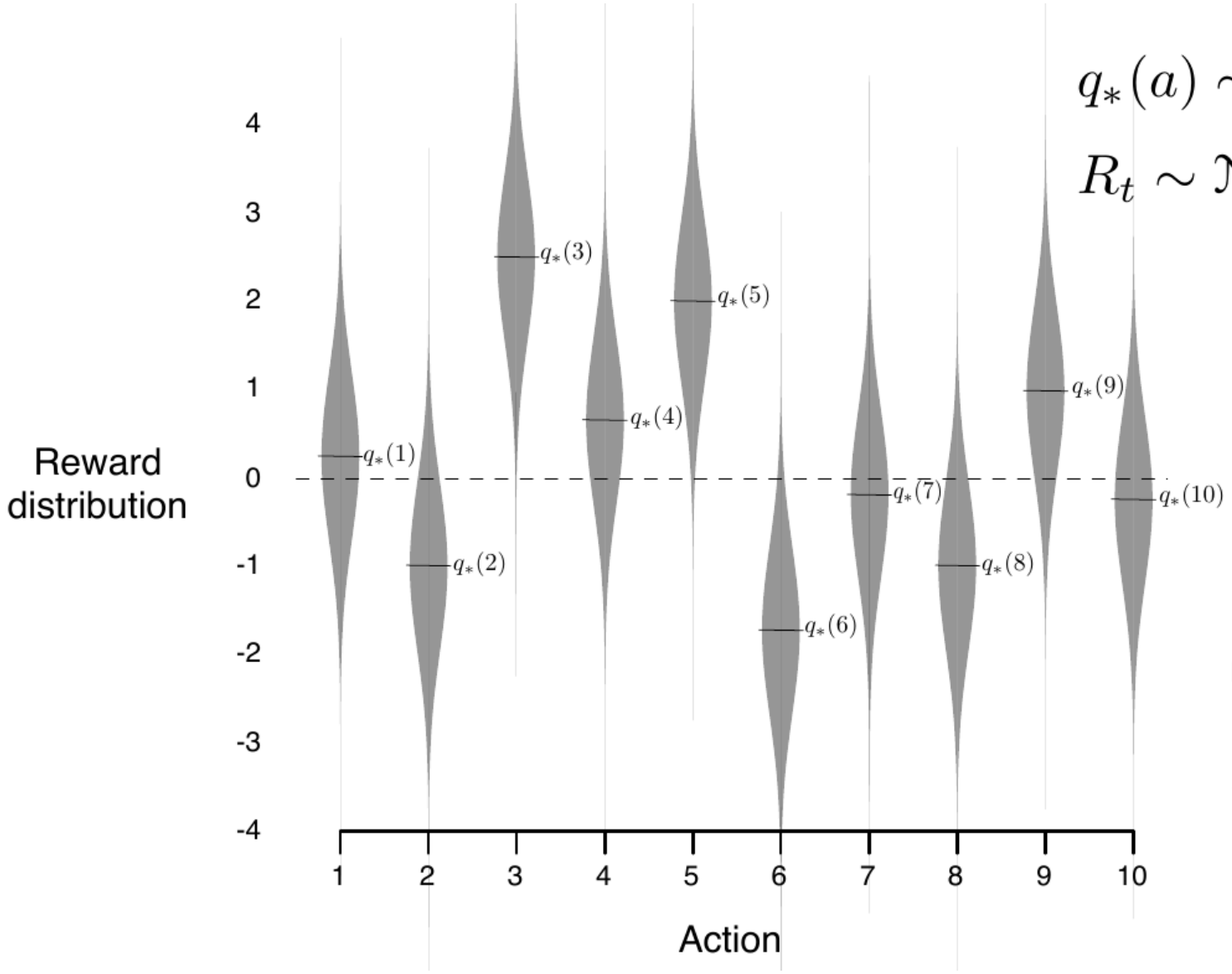
Bandits: What we learned so far

1. *Multi-armed bandits* are a simplification of the real problem
    1. they have action and reward (a goal), but no input or sequentiality
2. A fundamental *exploitation-exploration tradeoff* arises in bandits
    1. ε-*greedy action selection* is the simplest way of trading off
3. *Learning action values* is a key part of solution methods
4. *The 10-armed testbed* illustrates all

The 10-armed Testbed

$$q_*(a) \sim \mathcal{N}(0, 1)$$

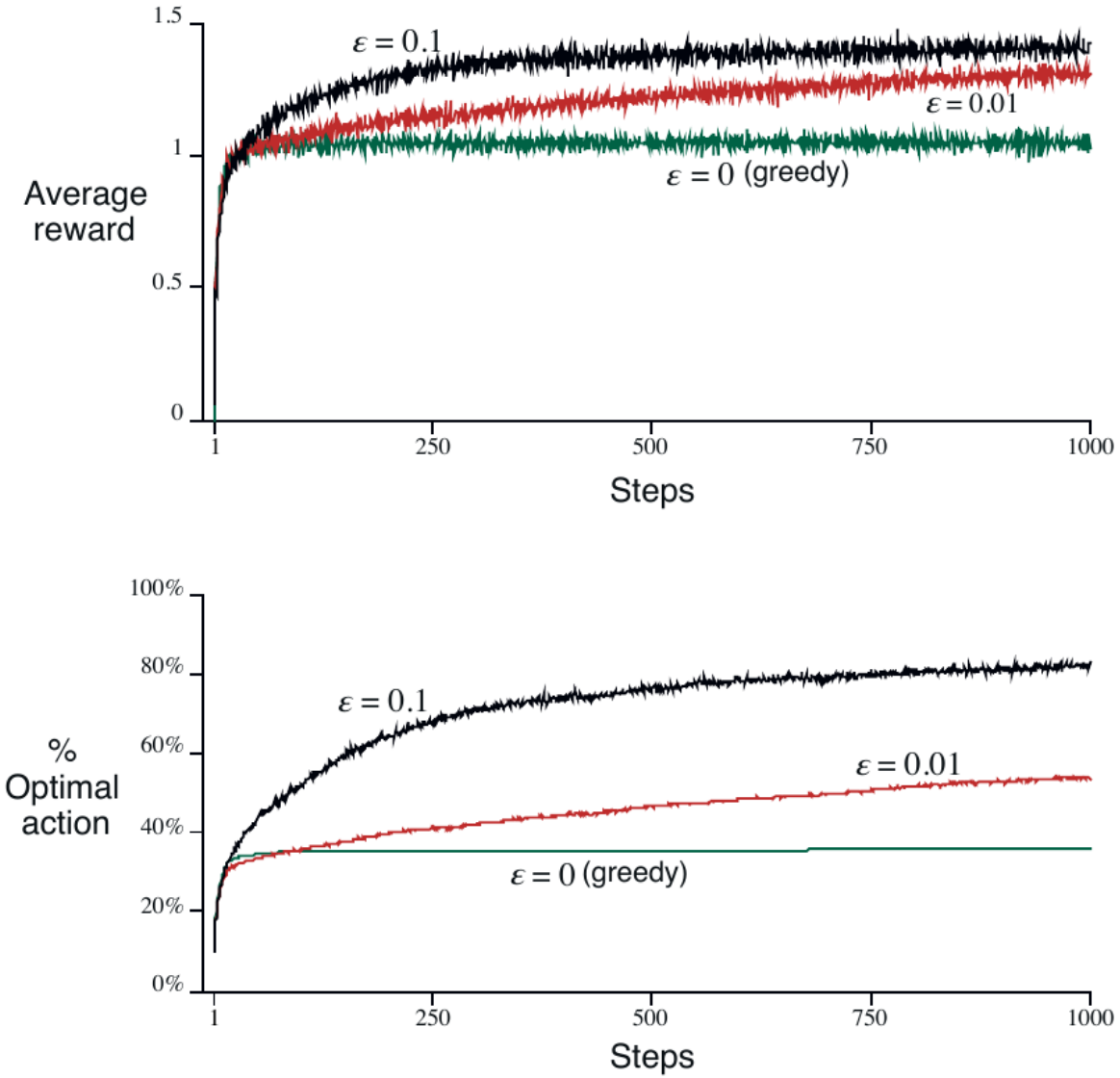
$$R_t \sim \mathcal{N}(q_*(a), 1)$$



Run for 1000 steps

Repeat the whole thing 2000 times with different bandit tasks

ε-Greedy Methods on the 10-Armed Testbed



Averaging → learning rule

- To simplify notation, let us focus on one action
  - We consider only its rewards, and its estimate after $n+1$ rewards:

$$Q_n \doteq \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}$$

- How can we do this incrementally (without storing all the rewards)?
- Could store a running sum and count (and divide), or equivalently:

$$Q_{n+1} = Q_n + \frac{1}{n}\Big[R_n - Q_n\Big]$$

- This is a standard form for learning/update rules:

$$NewEstimate \leftarrow OldEstimate \; + \; StepSize \; \Big[Target - OldEstimate\Big]$$

StepSize aka ***learning rate***

Derivation of incremental update

$$Q_n \doteq \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}$$

$$\begin{aligned}
Q_{n+1} &= \frac{1}{n}\sum_{i=1}^{n} R_i \\
&= \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\left(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\Big(R_n + (n-1)Q_n\Big) \\
&= \frac{1}{n}\Big(R_n + nQ_n - Q_n\Big) \\
&= Q_n + \frac{1}{n}\Big[R_n - Q_n\Big],
\end{aligned}$$

Tracking a Non-stationary Problem

- Suppose the true action values change slowly over time
  - then we say that the problem is *nonstationary*
- In this case, sample averages are not a good idea (Why?)
- Better is an "exponential, recency-weighted average":

$$Q_{n+1} \doteq Q_n + \alpha\Big[R_n - Q_n\Big]$$
$$= (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i,$$

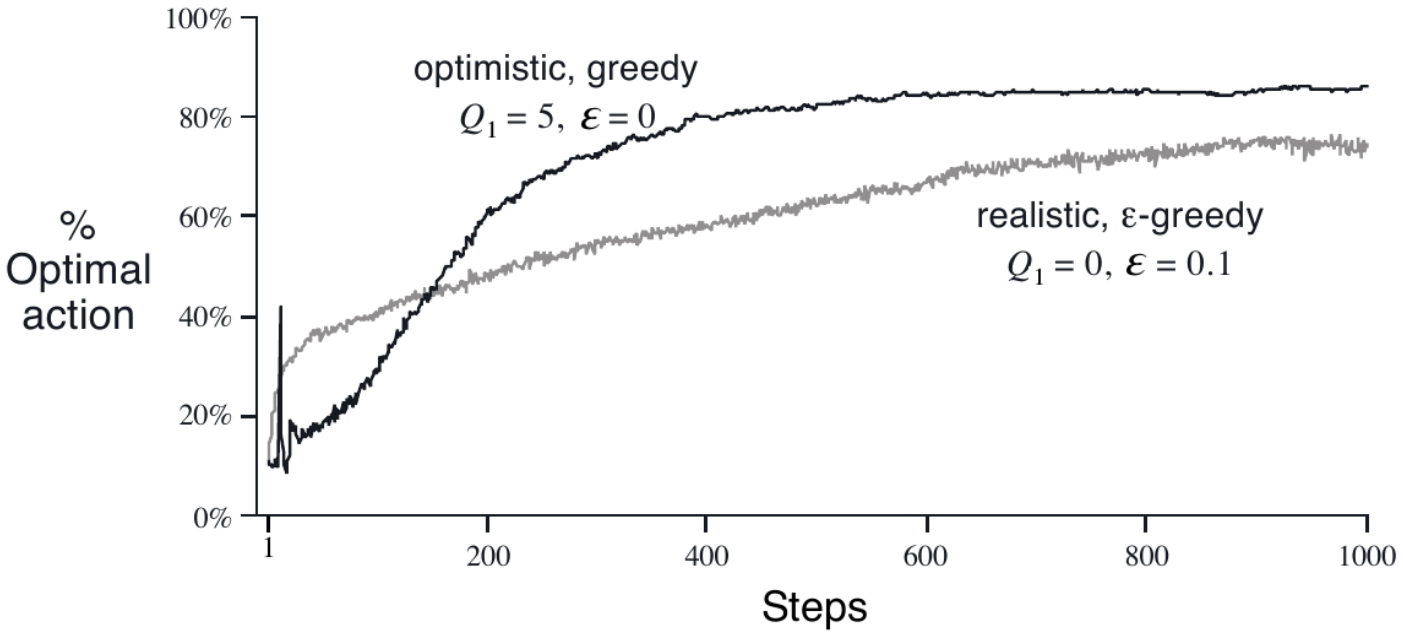
where α is a constant *step-size parameter*, $\alpha \in (0, 1]$

- There is bias due to Q_1 that becomes smaller over time

Optimistic Initial Values

- All methods so far depend on $Q_1(a)$, i.e., they are biased.
  So far we have used $Q_1(a) = 0$

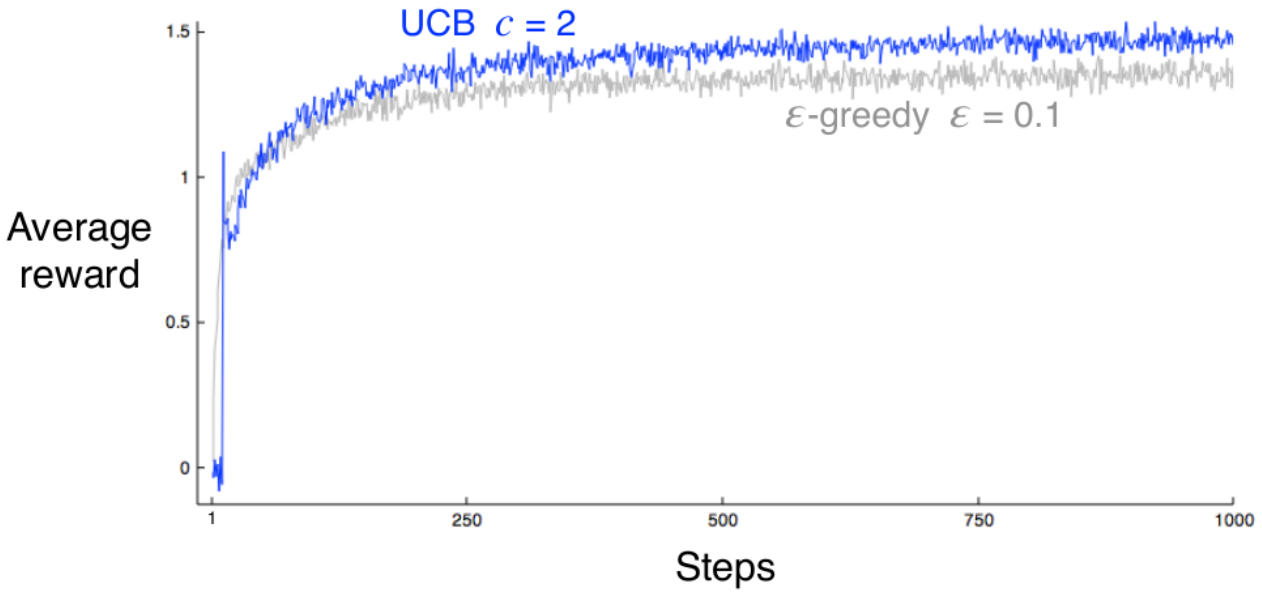
- Suppose we initialize the action values *optimistically* ($Q_1(a) = 5$),
  e.g., on the 10-armed testbed (with $\alpha = 0.1$)



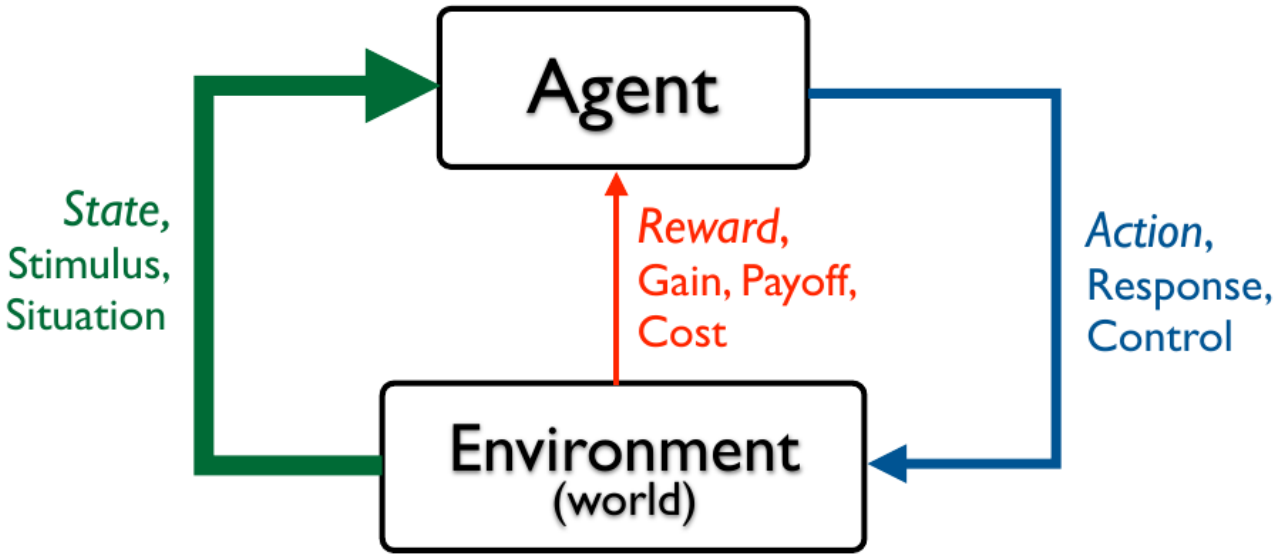
Upper Confidence Bound (UCB) action selection

- A clever way of reducing exploration over time
- Estimate an upper bound on the true action values
- Select the action with the largest (estimated) upper bound

$$A_t \doteq \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\log t}{N_t(a)}} \right]$$



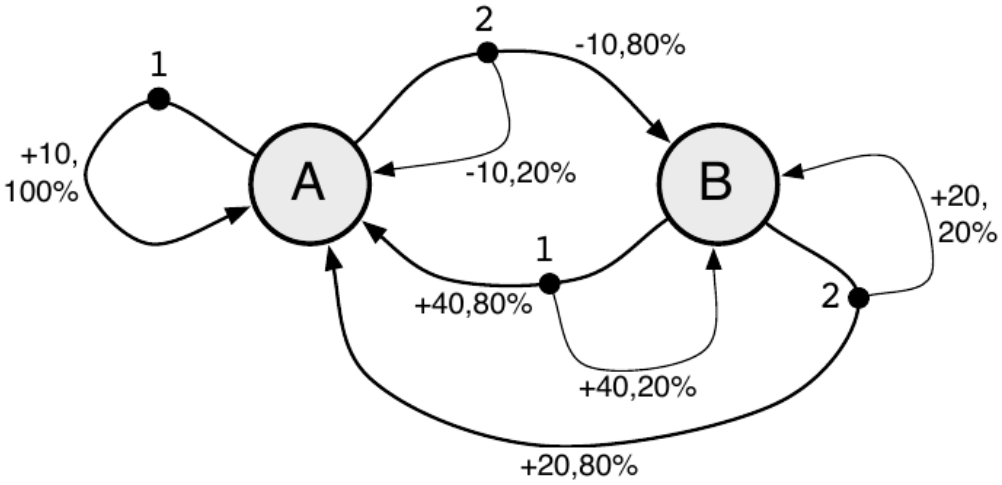
The RL Interface



- Environment may be unknown, nonlinear, stochastic and complex
- Agent learns a policy mapping states to actions
  - Seeking to maximize its cumulative reward in the long run

The Environment: A Finite Markov

- Discrete time $t = 1, 2, 3, \ldots$
- A finite set of states
- A finite set of actions
- A finite set of rewards
- Life is a trajectory:



$$\ldots S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, S_{t+2}, \ldots$$

- With arbitrary Markov (stochastic, state-dependent) dynamics:

$$p(r, s'|s, a) = Prob\Big[R_{t+1} = r, S_{t+1} = s' \;\Big|\; S_t = s, A_t = a\Big]$$

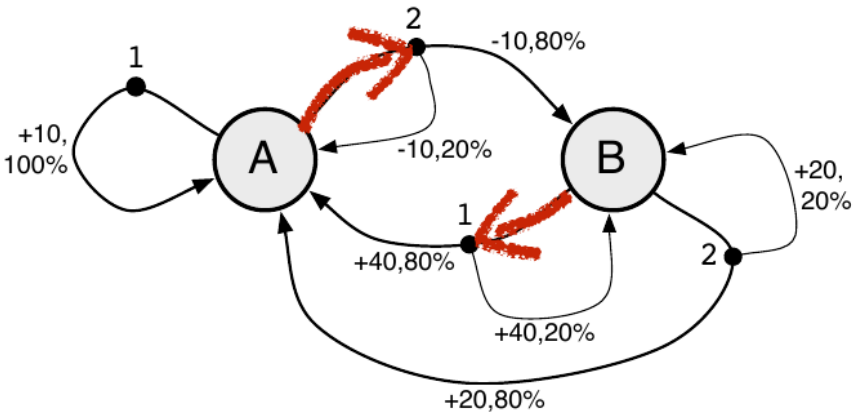
Action-value functions

- An action-value function says how good it is to be in a state, take an action, and thereafter follow a policy:

$$q_\pi(s,a) = \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \;\middle|\; S_t = s, A_t = a, A_{t+1:\infty} \sim \pi\right]$$

Action-value function
for the optimal policy and γ=0.9

State	Action	Value
A	1	130.39
A	2	133.77
B	1	166.23
B	2	146.23



Optimal policies

- A policy π_* is optimal iff it maximizes the action-value function:

$$q_{\pi_*}(s,a) = \max_{\pi} q_\pi(s,a) = q_*(s,a)$$

- Thus all optimal policies share the same optimal value function

- Given the optimal value function, it is easy to act optimally:

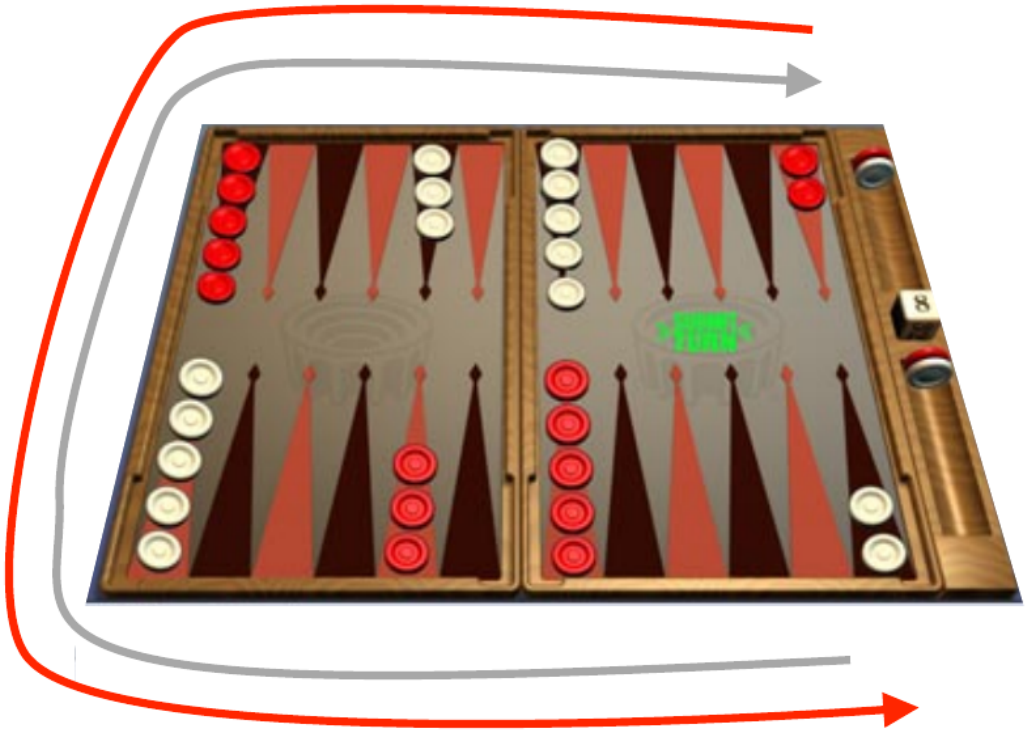
$$\pi_*(s) = \arg\max_{a} q_*(s,a)$$

"greedification"

- We say that the optimal policy is greedy with respect to the optimal value function

- There is always at least one deterministic optimal policy

Backgammon



STATES: configurations of the playing board ($\approx 10^{20}$)

ACTIONS: moves

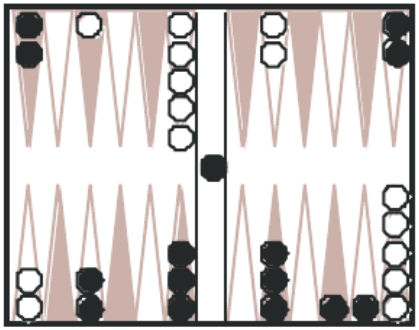
REWARDS: win: +1

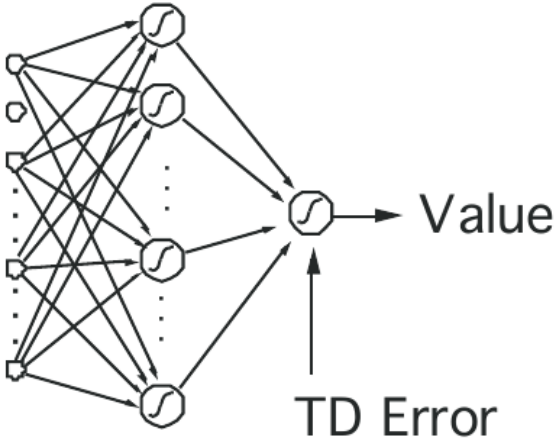
lose: −1

else: 0

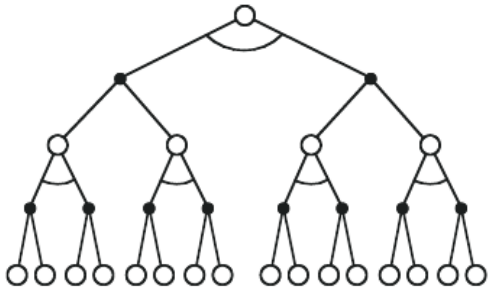
a "big" game

TD-Gammon [Tesauro, 1992-1995]





Action selection
by 2-3 ply search



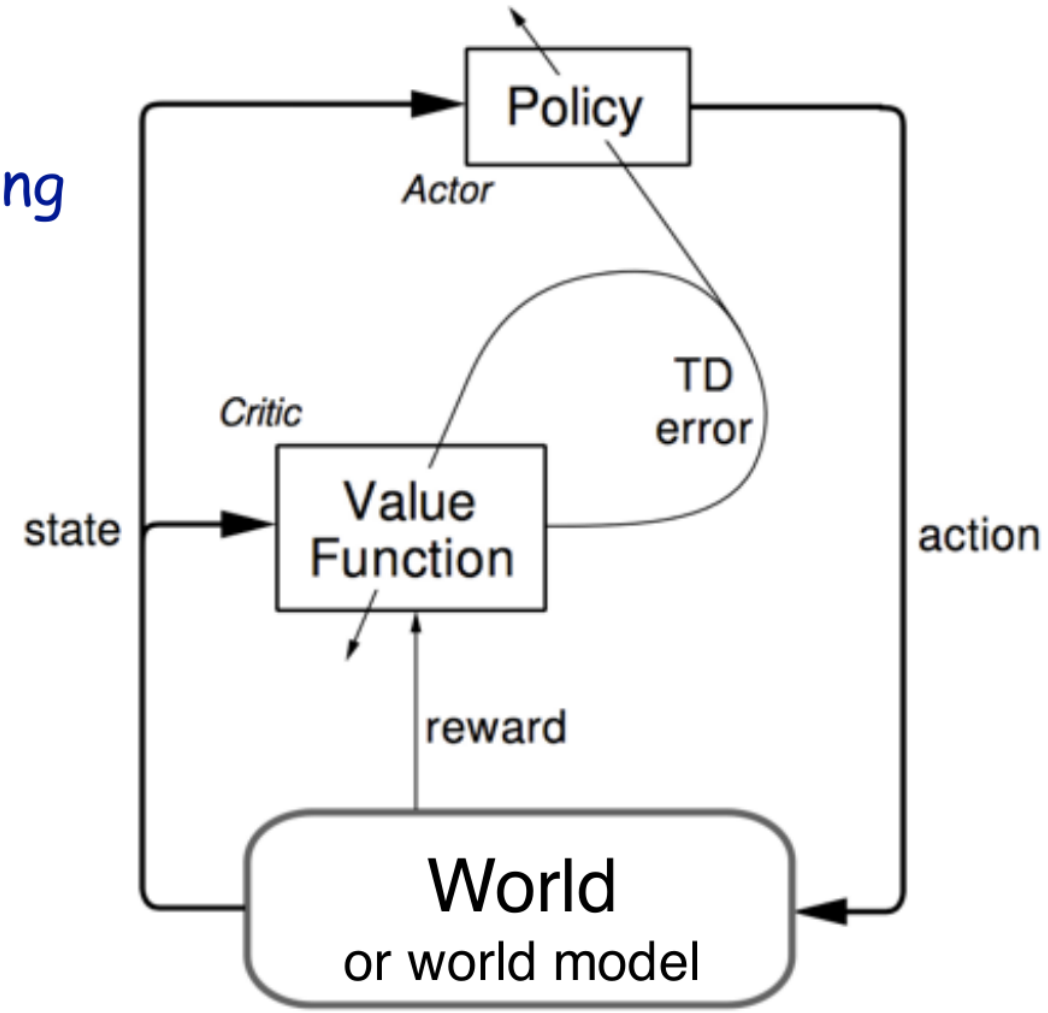
Start with a random Network

Play millions of games against itself

Learn a value function from this simulated experience

Temporal-difference (TD) error

the actor-critic reinforcement learning architecture



Return

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?

At least three cases, but in all of them,

we seek to maximize the **expected return**, $E\{G_t\}$, on each step t.

- Total reward, G_t = sum of all future reward in the episode
- Discounted reward, G_t = sum of all future *discounted* reward
- Average reward, G_t = average reward per time step

Episodic Tasks

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

In episodic tasks, we almost always use simple *total reward*:

$$G_t = R_{t+1} + R_{t+2} + \cdots + R_T ,$$

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

Continuing Tasks

Continuing tasks: interaction does not have natural episodes, but just goes on and on...

In this class, for continuing tasks we will always use *discounted return*:

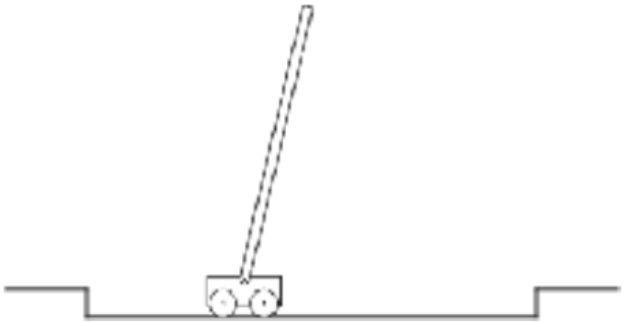
$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

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

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

Typically, $\gamma = 0.9$

An Example: Pole Balancing



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:

reward = +1 for each step before failure

$\Rightarrow$ return = number of steps before failure

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

reward = −1 upon failure; 0 otherwise

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

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

Values are *expected* returns

- The value of a state, given a policy:

$$v_\pi(s) = \mathbb{E}\{G_t \mid S_t = s, A_{t:\infty} \sim \pi\} \qquad v_\pi : \mathcal{S} \to \Re$$

- The value of a state-action pair, given a policy:

$$q_\pi(s,a) = \mathbb{E}\{G_t \mid S_t = s, A_t = a, A_{t+1:\infty} \sim \pi\} \qquad q_\pi : \mathcal{S} \times \mathcal{A} \to \Re$$

- The optimal value of a state:

$$v_*(s) = \max_\pi v_\pi(s) \qquad v_* : \mathcal{S} \to \Re$$

- The optimal value of a state-action pair:

$$q_*(s,a) = \max_\pi q_\pi(s,a) \qquad q_* : \mathcal{S} \times \mathcal{A} \to \Re$$

- Optimal policy: π_* is an optimal policy if and only if

$$\pi_*(a|s) > 0 \text{ only where } q_*(s,a) = \max_b q_*(s,b) \qquad \forall s \in \mathcal{S}$$

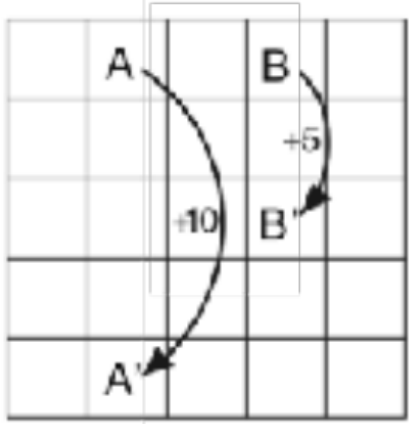
  - in other words, π_* is optimal iff it is *greedy* wrt q_*

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.



(a)

Actions

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

(b)

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

Optimal Value Functions

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

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

There are always one or more policies that are better than or equal to all the others. These are the **optimal policies**. We denote them all π_*.

Optimal policies share the same **optimal state-value function**:

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

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

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

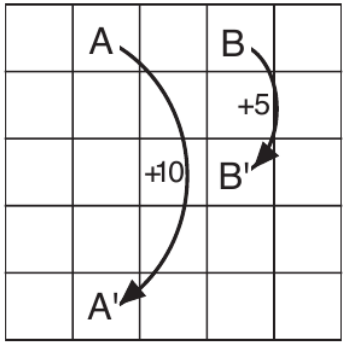
This is the expected return for taking action a in state s and thereafter following an optimal policy.

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.

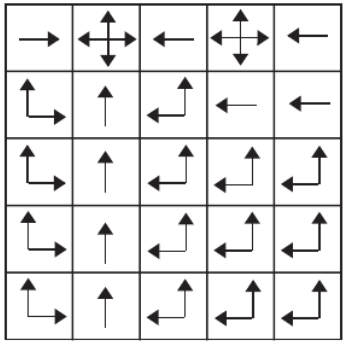
E.g., back to the gridworld:



a) gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

b) v_*



c) π_*

Bellman Equation for a Policy π

The basic idea:

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

So:

$$\begin{aligned} v_\pi(s) &= E_\pi \left\{ G_t \,\middle|\, S_t = s \right\} \\ &= E_\pi \left\{ R_{t+1} + \gamma v_\pi \left(S_{t+1} \right) \middle|\, S_t = s \right\} \end{aligned}$$

Or, without the expectation operator:

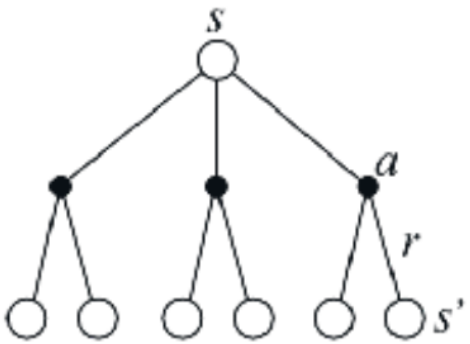
$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_\pi(s') \right]$$

More on the Bellman Equation

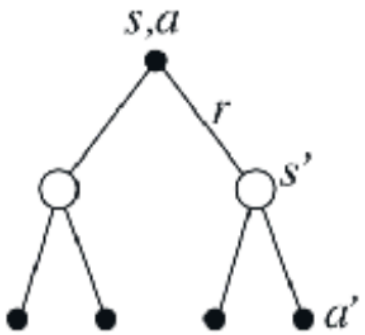
$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_\pi(s')\Big]$$

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



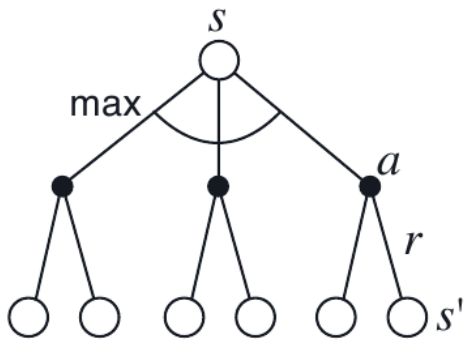
for q_π

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:

$$
\begin{aligned}
v_*(s) &= \max_a q_{\pi_*}(s,a) \\
&= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma v_*(s')\big].
\end{aligned}
$$

The relevant backup diagram:

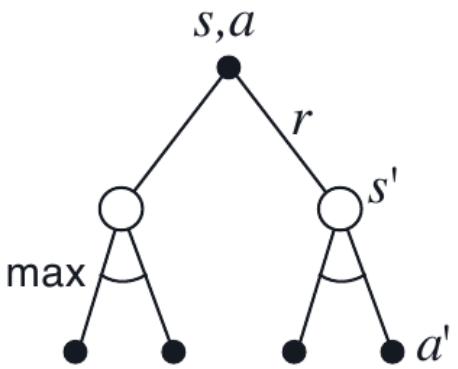


v_* is the unique solution of this system of nonlinear equations.

Bellman Optimality Equation for q*

$$
\begin{aligned}
q_*(s,a) &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \;\middle|\; S_t = s, A_t = a\right] \\
&= \sum_{s',r} p(s', r|s, a)\left[r + \gamma \max_{a'} q_*(s', a')\right].
\end{aligned}
$$

The relevant backup diagram:



q_* is the unique solution of this system of nonlinear equations.

Policy Evaluation (Prediction)

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

Recall: **State-value function for policy π**

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right]$$

Recall: **Bellman equation for v_π**

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

—a system of $|\mathcal{S}|$ simultaneous equations

Iterative Policy Evaluation

$$v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_k \rightarrow v_{k+1} \rightarrow \cdots \rightarrow v_\pi$$

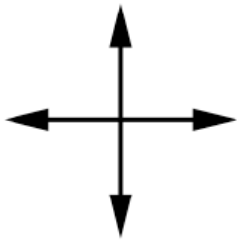
a "sweep"

A sweep consists of applying a **backup operation** to each state.

A **full policy-evaluation backup**:

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \Big[r + \gamma v_k(s') \Big] \qquad \forall s \in \mathcal{S}$$

A Small Gridworld Example



actions

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

$R = -1$
on all transitions

$\gamma = 1$

An undiscounted episodic task

Nonterminal states: $1, 2, \ldots, 14$;

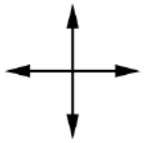
One terminal state (shown twice as shaded squares)

Actions that would take agent off the grid leave state unchanged

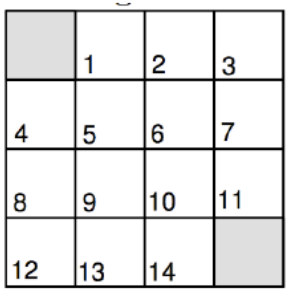
Reward is -1 until the terminal state is reached

Iterative Policy Eval for the Small Gridworld

$\pi =$ equiprobable random action choices



actions



$R = -1$
on all transitions

$\gamma = 1$

- An undiscounted episodic task
- Nonterminal states: 1, 2, . . ., 14;
- One terminal state (shown twice as shaded squares)
- Actions that would take agent off the grid leave state unchanged
- Reward is –1 until the terminal state is reached

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_k(s')\Big] \qquad \forall s \in \mathcal{S}$$

V_k for the Random Policy

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

Iterative Policy Evaluation – One array version

Input π, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}$
Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

Policy improvement

- Given the value function for *any policy* π:

$$q_\pi(s, a) \qquad \text{for all } s, a$$

- It can always be greedified to obtain a *better policy*:

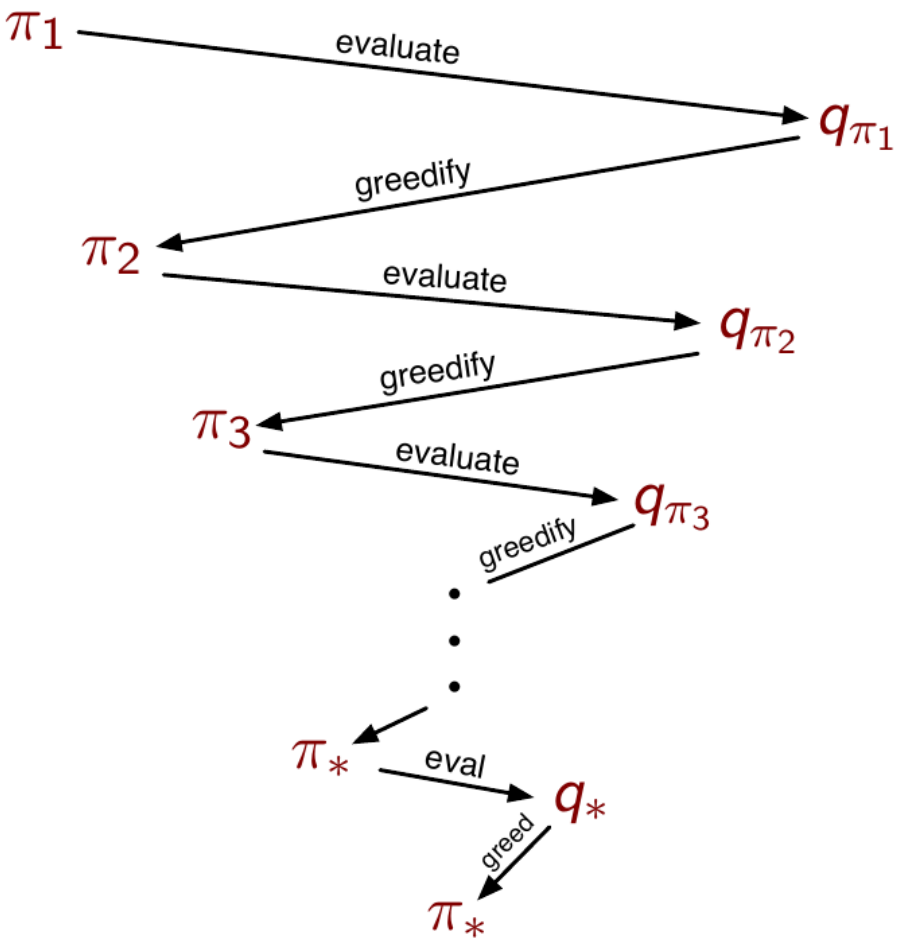
$$\pi'(s) = \arg\max_a q_\pi(s, a) \qquad (\pi' \text{ is not unique})$$

- where better means:

$$q_{\pi'}(s, a) \geq q_\pi(s, a) \qquad \text{for all } s, a$$

- with equality only if both policies are optimal

The dance of policy and value (Policy Iteration)



Any policy evaluates to a unique value function (soon we will see how to learn it)

which can be greedified to produce a better policy

That in turn evaluates to a value function

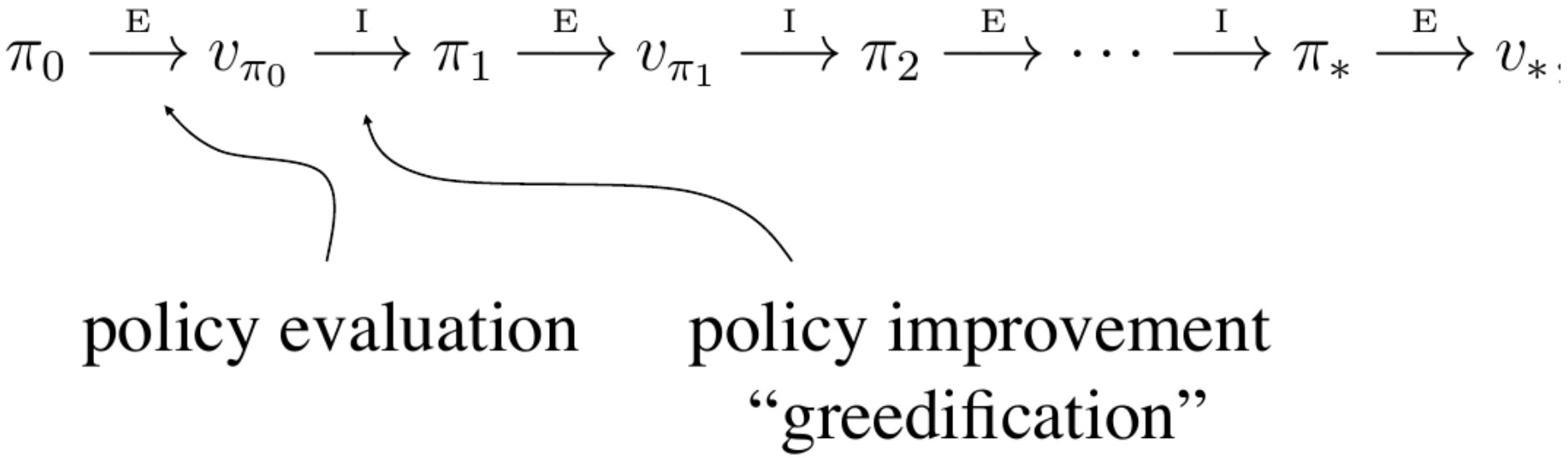
which can in turn be greedified…

Each policy is *strictly better* than the previous, until *eventually both are optimal*

There are *no local optima*

The dance converges in a finite number of steps, usually very few

Policy Iteration



Policy Iteration – One array version

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
   $\Delta \leftarrow 0$
   For each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\big[r + \gamma V(s')\big]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $a \leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
   If $a \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return V and π; else go to 2

Value Iteration – One array version

Initialize array V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

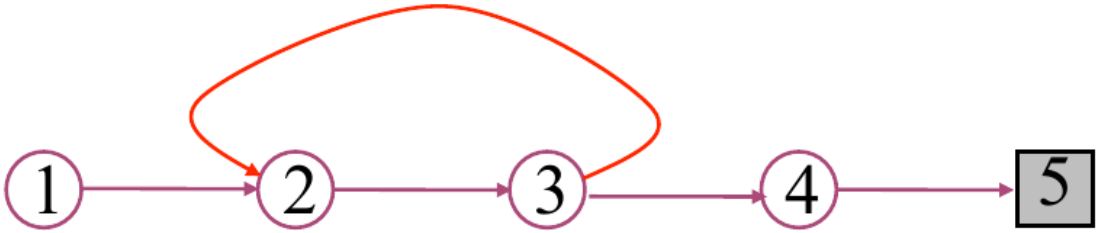
Output a deterministic policy, π, such that
  $\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$

Monte Carlo Policy Evaluation

Goal: learn $v_\pi(s)$

Given: some number of episodes under π which contain *s*

Idea: Average returns observed after visits to s



Every-Visit MC: average returns for *every* time *s* is visited in an episode

First-visit MC: average returns only for *first* time *s* is visited in an episode

Both converge asymptotically

TD methods bootstrap and sample

Bootstrapping: update involves an estimate of the value function

- TD and DP methods bootstrap
- MC methods **do not** bootstrap

Sampling: update **does not** involve an expected value

- TD and MC method sample
- Classical DP **does not** sample

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:

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

target: the actual return after time t

The simplest temporal-difference method TD(0):

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

target: an estimate of the return

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0$, for all $s \in \mathcal{S}^+$)
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        $A \leftarrow$ action given by π for S
        Take action A, observe R, S'
        $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
        $S \leftarrow S'$
    until S is terminal

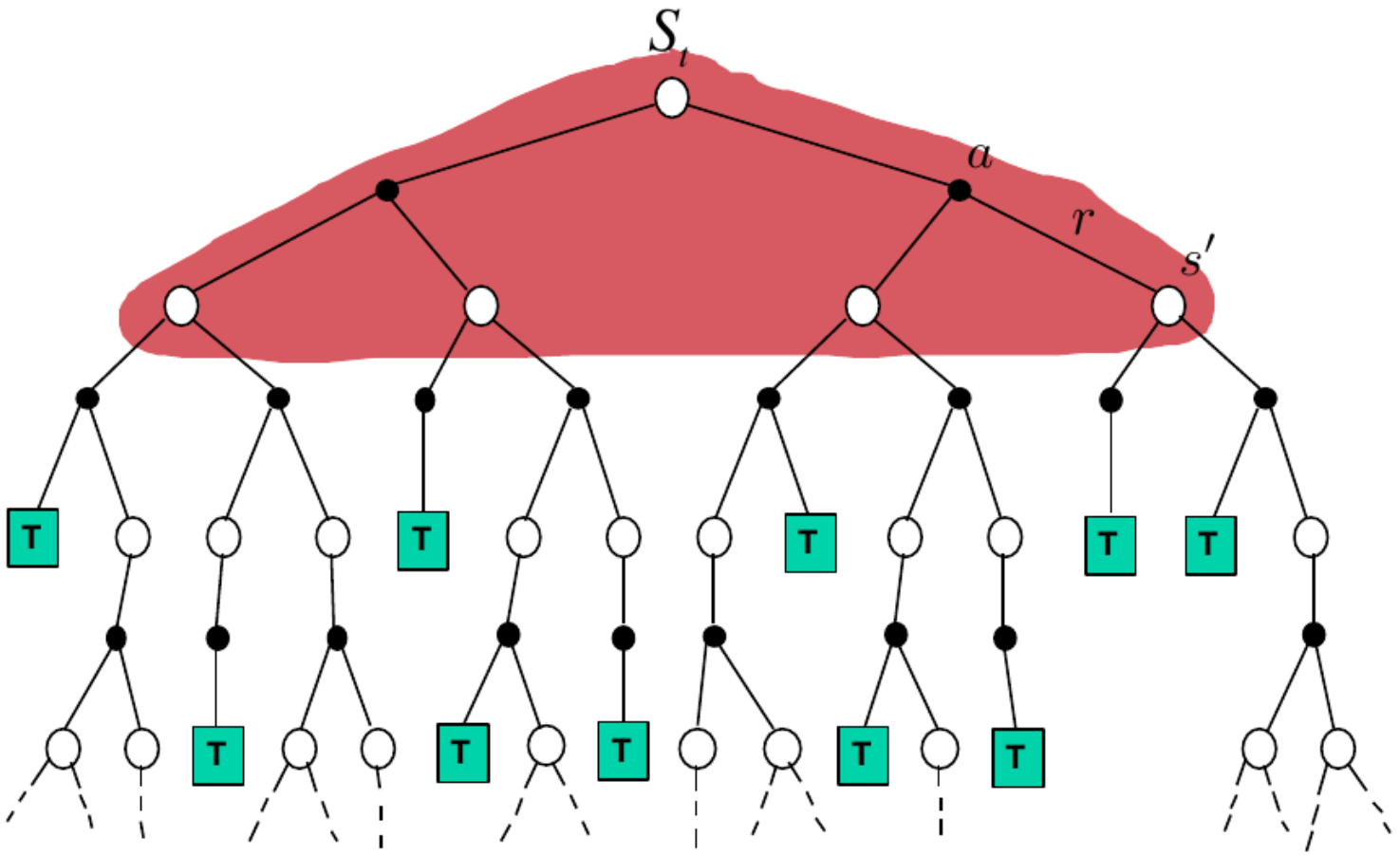
Agent program

Environment program

Experiment program

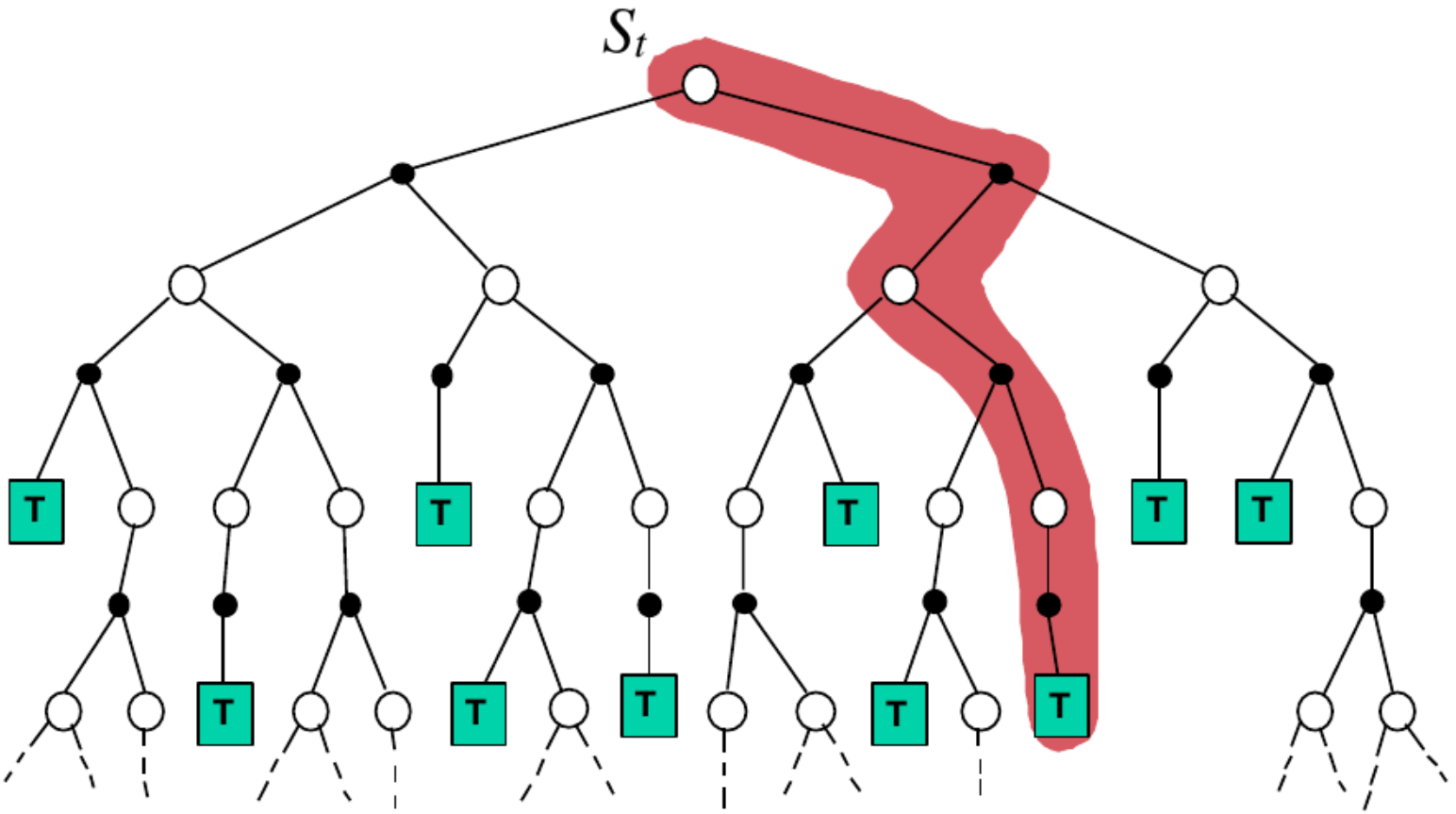
Dynamic Programming

$$V(S_t) \leftarrow E_\pi\left[R_{t+1} + \gamma V(S_{t+1})\right] = \sum_a \pi(a|S_t) \sum_{s',r} p(s', r|S_t, a)[r + \gamma V(s')]$$



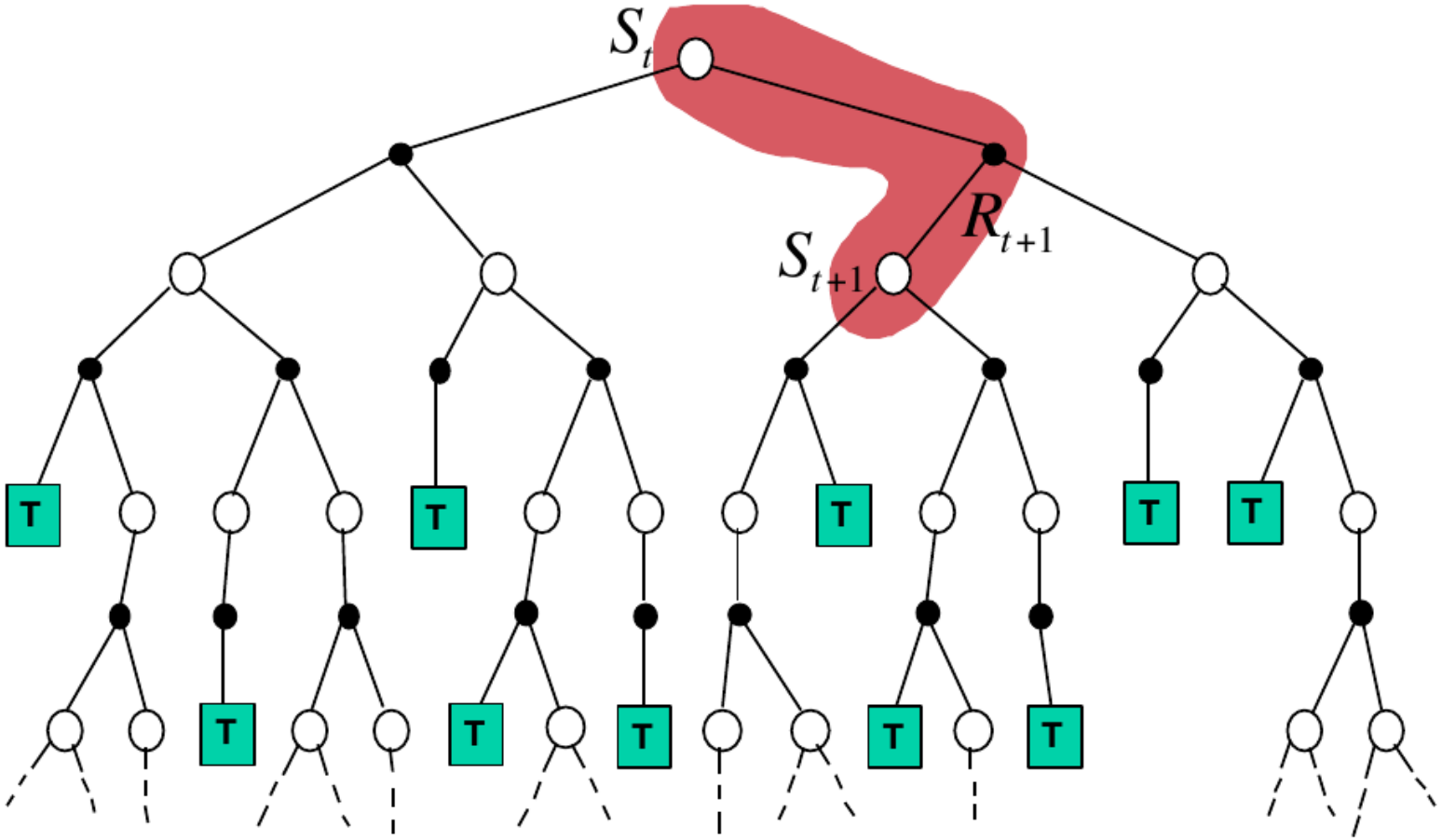
Simple Monte Carlo

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



Simplest TD method

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



Advantages of TD learning

TD methods do not require a model of the environment, only experience

TD methods can be fully incremental

- Make updates **before** knowing the final outcome
- Requires less memory
- Requires less peak computation

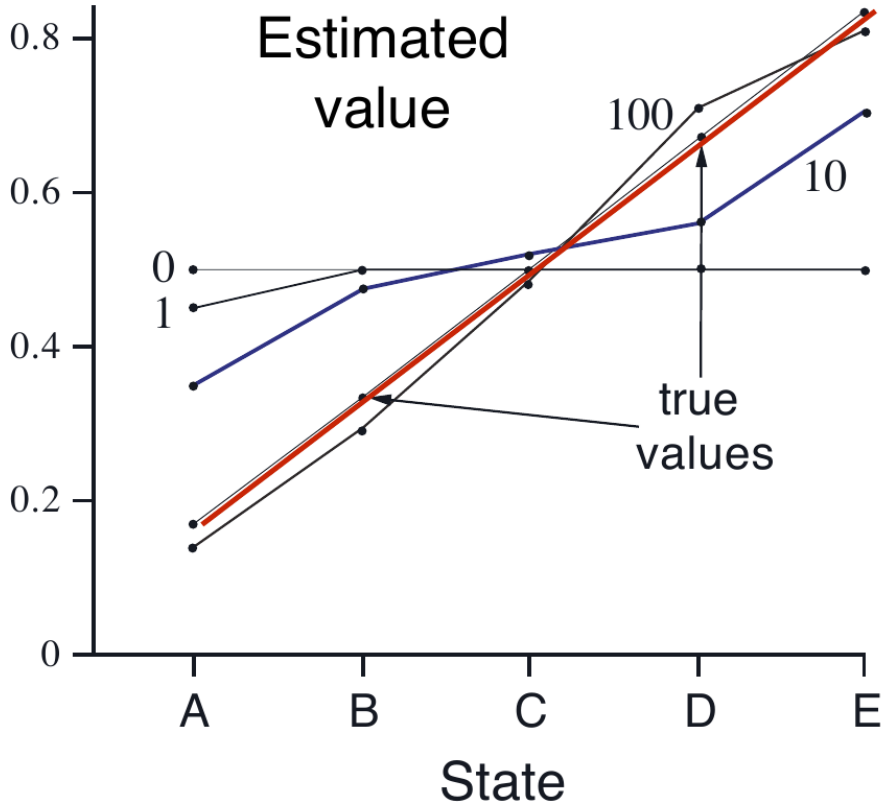
You can learn **without** the final outcome, from incomplete sequences

Both MC and TD converge (under certain assumptions to be detailed later), but which is faster?

Random walk

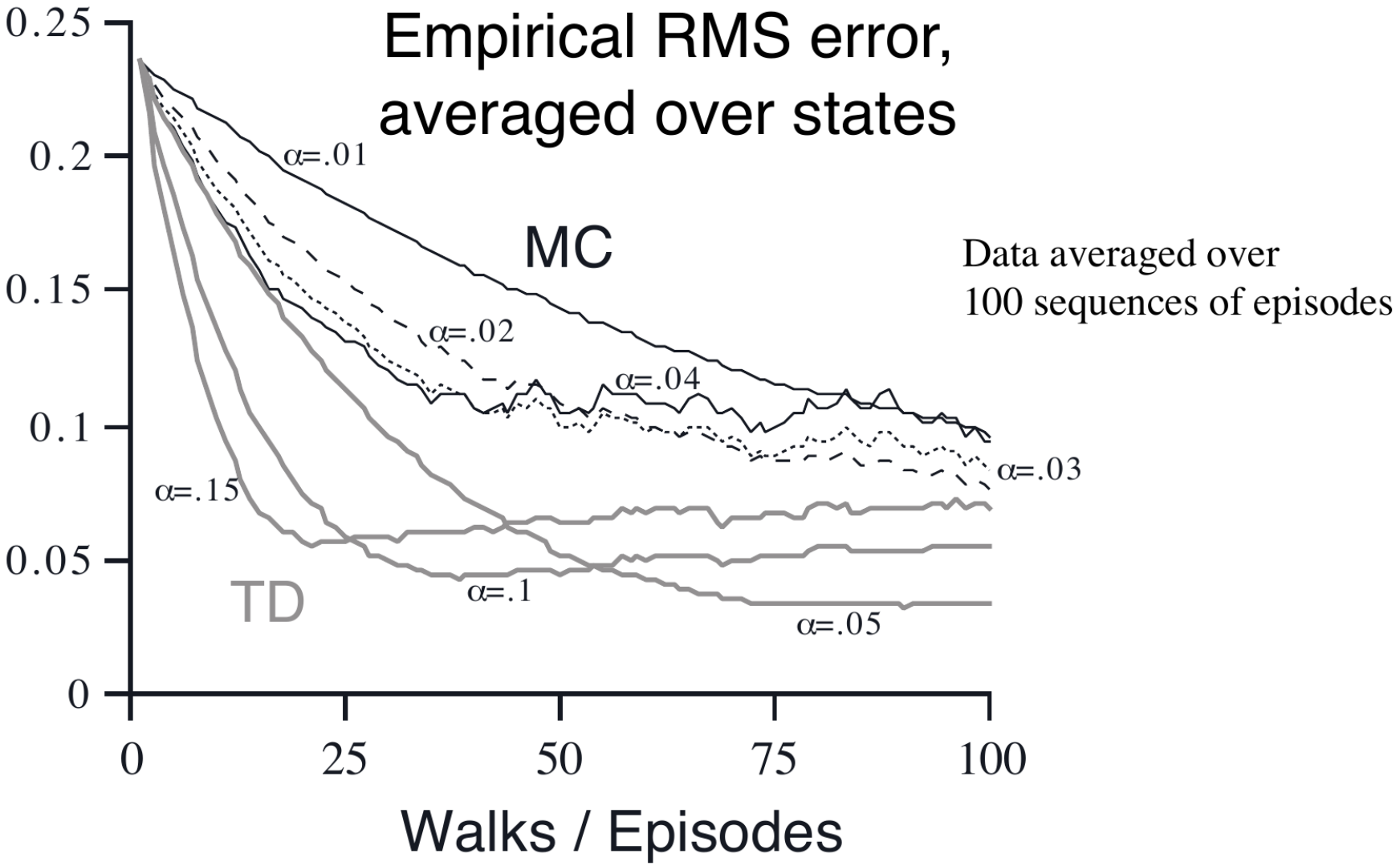
- C is start state, episodic, undiscounted γ = 1
- π is left or right with equal probability in all states
- termination at either end
- rewards +1 on **right** termination, 0 otherwise
- what does $v_\pi(s)$ tell us?
  - probability of termination on right side from each state, under random policy
  - what is v_π =[A B C D E]?
    - v_π = [1/6 2/6 3/6 4/6 5/6]
- Initialize V(s) = 0.5 $\forall\ s\ \in\ \mathcal{S}$

Values learned by TD after various numbers of episodes

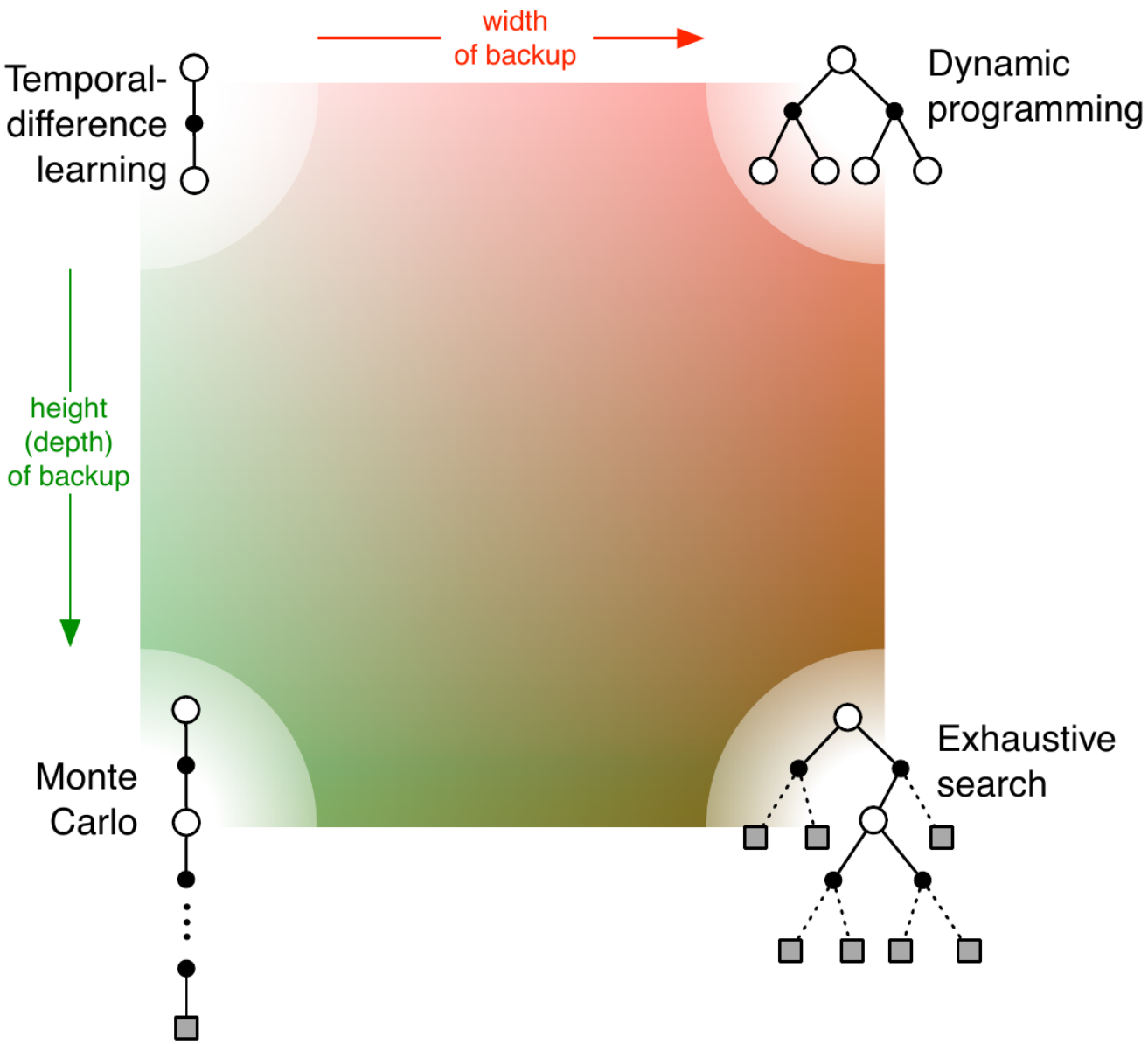


$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$

TD and MC on the Random Walk

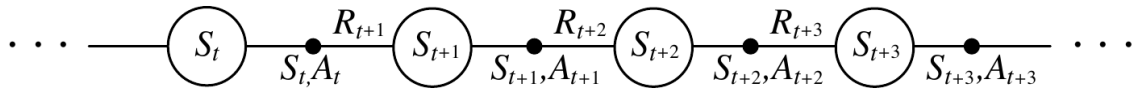


Unified View



Let’s conclude with two practical RL algorithms

SARSA



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 $Q(S_{t+1}, A_{t+1}) = 0$

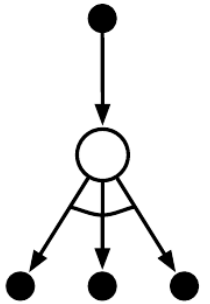
Turn this into a control method by always updating the policy to be greedy with respect to the current estimate:

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., 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)
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Q-Learning

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., ε-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