Reinforcement Learning

Slides by Rich Sutton

Mods by Dan Lizotte

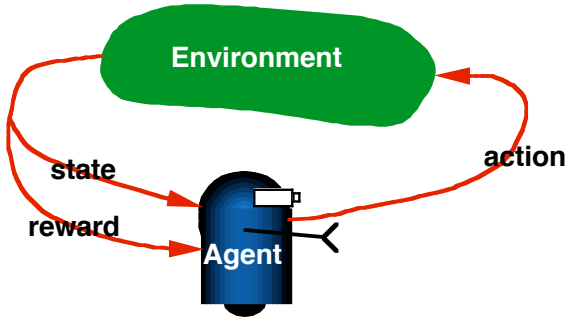
Refer to "Reinforcement Learning: An Introduction" by Sutton and Barto
Alpaydin Chapter 16

Up until now we have been…

- Supervised Learning
  - Classifying, mostly
  - Also saw some regression
  - Also doing some probabilistic analysis
- In comes data
  - Then we think for a while
- Out come predictions
- Reinforcement learning is in some ways similar, in some ways very different. (Like this font!)

Complete Agent

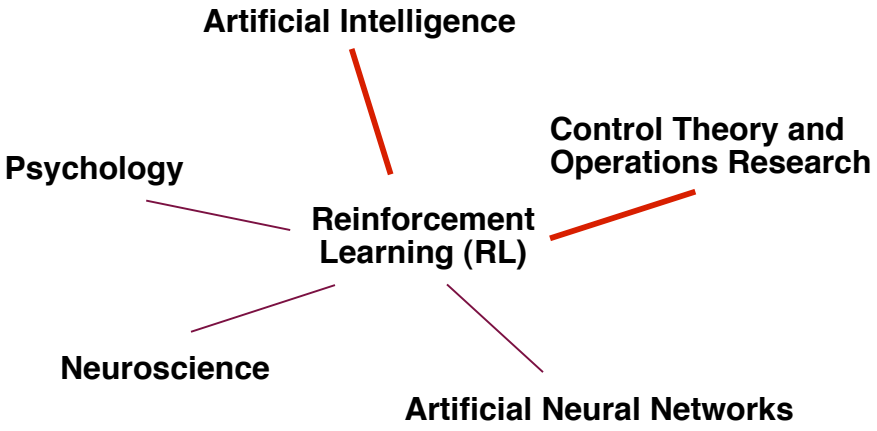
- Temporally situated
- Continual learning and planning
- Objective is to ***affect*** the environment
- Environment is stochastic and uncertain



What is Reinforcement Learning?

- An approach to Artificial Intelligence
- Learning from interaction
- Goal-oriented learning
- Learning about, from, and while interacting with an external environment
- Learning what to do—how to map situations to actions—so as to maximize a numerical reward signal

Chapter 1: Introduction



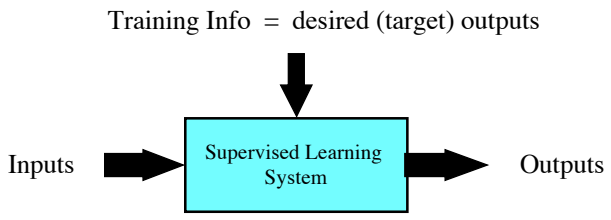
Key Features of RL

- Learner is not told which actions to take
- Trial-and-Error search
- Possibility of delayed reward
  - Sacrifice short-term gains for greater long-term gains
- The need to ***explore*** and ***exploit***
- Considers the whole problem of a goal-directed agent interacting with an uncertain environment

Examples of Reinforcement Learning

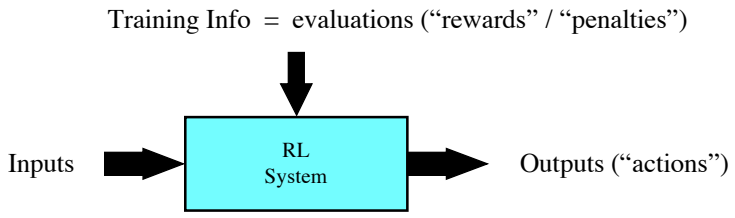
- Robocup Soccer Teams Stone & Veloso, Reidmiller et al.
  - World's best player of simulated soccer, 1999; Runner-up 2000
- Inventory Management Van Roy, Bertsekas, Lee & Tsitsiklis
  - 10-15% improvement over industry standard methods
- Dynamic Channel Assignment Singh & Bertsekas, Nie & Haykin
  - World's best assigner of radio channels to mobile telephone calls
- Elevator Control Crites & Barto
  - (Probably) world's best down-peak elevator controller
- Many Robots
  - navigation, bi-pedal walking, grasping, switching between skills...
- TD-Gammon and Jellyfish Tesauro, Dahl
  - World's best backgammon player

Supervised Learning



Error = (target output – actual output)

Reinforcement Learning

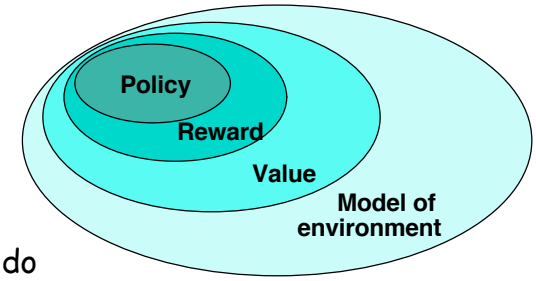


Objective: get as much reward as possible

Today

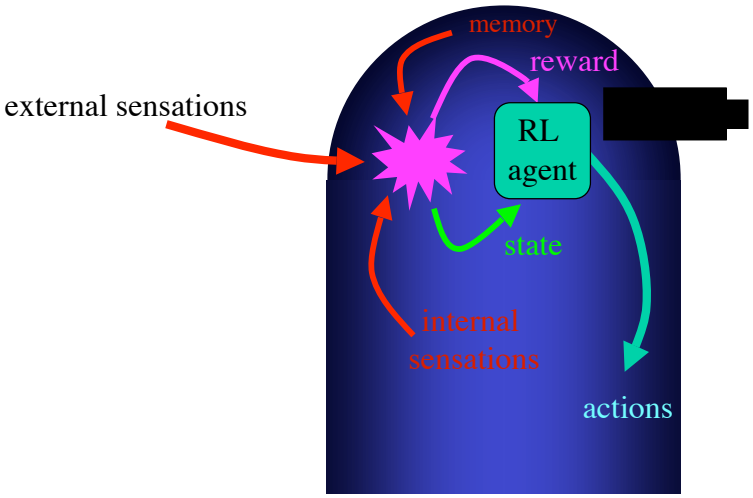
- Give an overview of the whole RL problem…
  - Before we break it up into parts to study individually
- Introduce the cast of characters
  - Experience (reward)
  - Policies
  - Value functions
  - Models of the environment
- Tic-Tac-Toe example

Elements of RL

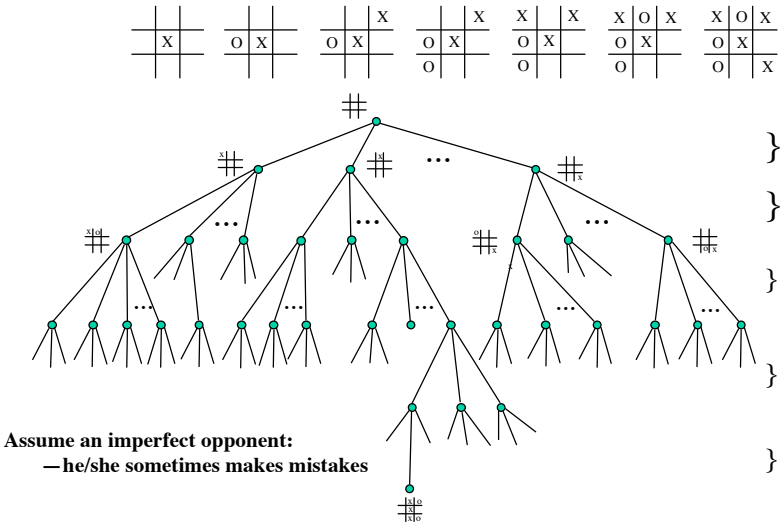


- **Policy**: what to do
- **Reward**: what is good
- **Value**: what is good because it *predicts* reward
- **Model**: what follows what

A Somewhat Less Misleading View…



An Extended Example: Tic-Tac-Toe



} x's move
} o's move
} x's move
} o's move
} x's move

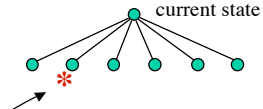
Assume an imperfect opponent:
—he/she sometimes makes mistakes

An RL Approach to Tic-Tac-Toe

1. Make a table with one entry per state:

State	$V(s)$ – estimated probability of winning	
	.5	?
	.5	?
⋮	⋮	
	1	win
⋮	⋮	
	0	loss
⋮	⋮	
	0	draw

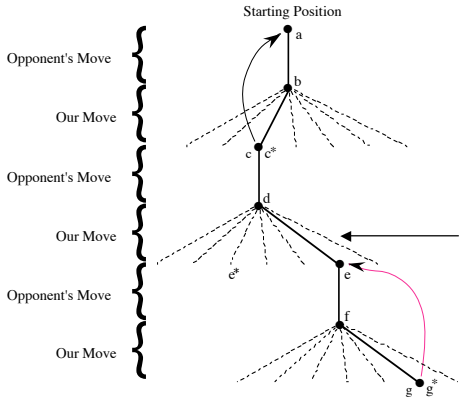
2. Now play lots of games.
To pick our moves,
look ahead one step:



Just pick the next state with the highest estimated prob. of winning — the largest $V(s)$; a ***greedy*** move.

But 10% of the time pick a move at random; an ***exploratory move***.

RL Learning Rule for Tic-Tac-Toe



s – the state before our greedy move

s' – the state after our greedy move

We increment each $V(s)$ toward $V(s')$ – a ***backup***:

$$V(s) \leftarrow V(s) + \alpha\left[V(s') - V(s)\right]$$

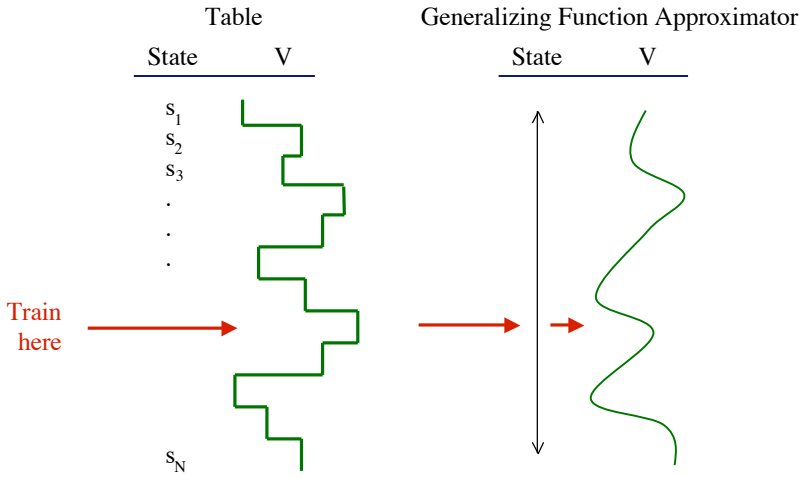
a small positive fraction, e.g., $\alpha = .1$

the ***step - size parameter***

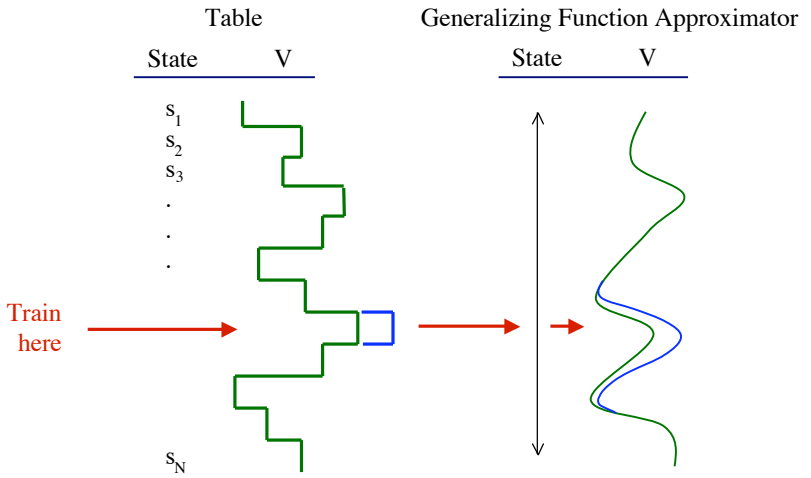
How can we improve this T.T.T. player?

- Take advantage of symmetries
  - representation/generalization
  - How might this backfire?
- Do we need "random" moves? Why?
  - Do we need the full 10%?
- Can we learn from "random" moves?
- Can we learn offline?
  - Pre-training from self play?
  - Using learned models of opponent?
- . . .

e.g. Generalization



e.g. Generalization



How is Tic-Tac-Toe Too Easy?

- Finite, small number of states
- One-step look-ahead is always possible
- State completely observable
- . . .

Chapter 2: Evaluative Feedback

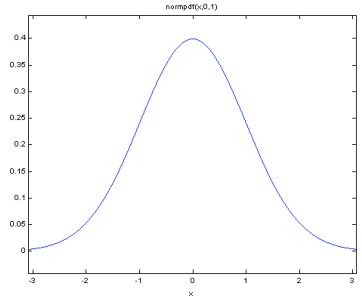
- **Evaluating** actions vs. **instructing** by giving correct actions
- Pure evaluative feedback depends totally on the action taken. Pure instructive feedback depends not at all on the action taken.
- Supervised learning is instructive; optimization is evaluative
- ***Associative*** vs. **Nonassociative**:
  - Associative: inputs mapped to outputs;
    - learn the best output **for each** input
  - Nonassociative: "learn" (find) one best output
    - ignoring inputs
- A simpler example: *n*-armed bandit (at least how we treat it) is:
  - Nonassociative
  - Evaluative feedback

= Pause for Stats =

- Suppose X is a real-valued random variable
- Expectation ("Mean")

$$E\{X\} = \lim_{n \to \infty} \frac{x_1 + x_2 + x_3 + \ldots + x_n}{n}$$

- Normal Distribution
  - Mean μ
  - Standard Deviation σ
  - Almost all values will be
  - $-3\sigma < x < 3\sigma$



The *n*-Armed Bandit Problem

- Choose repeatedly from one of *n* actions; each choice is called a **play**
- After each play a_t , you get a reward r_t , where

$$E\{r_t \mid a_t\} = Q^*(a_t)$$

These are unknown ***action values***
Distribution of r_t depends only on a_t

- Objective is to maximize the reward in the long term, e.g., over 1000 plays

To solve the *n*-armed bandit problem,
you must **explore** a variety of actions
and **exploit** the best of them

The Exploration/Exploitation Dilemma

- Suppose you form estimates

$$Q_t(a) \approx Q^*(a)$$ action value estimates

- The **greedy** action at t is a_t

$$a_t^* = \arg\max_a Q_t(a)$$

$$a_t = a_t^* \Rightarrow \text{exploitation}$$

$$a_t \neq a_t^* \Rightarrow \text{exploration}$$

- If you need to learn, you can't exploit all the time; if you need to do well, you can't explore all the time
- You can never stop exploring; but you should always reduce exploring. Maybe.

Action-Value Methods

- Methods that adapt action-value estimates and nothing else, e.g.: suppose by the t-th play, action a had been chosen k_a times, producing rewards $r_1, r_2, \ldots, r_{k_a}$, then

$$Q_t(a) = \frac{r_1 + r_2 + \cdots + r_{k_a}}{k_a}$$

"sample average"

- $\lim_{k_a \to \infty} Q_t(a) = Q^*(a)$

ε-Greedy Action Selection

- Greedy action selection:

$$a_t = a_t^* = \arg\max_a Q_t(a)$$

- ε-Greedy:

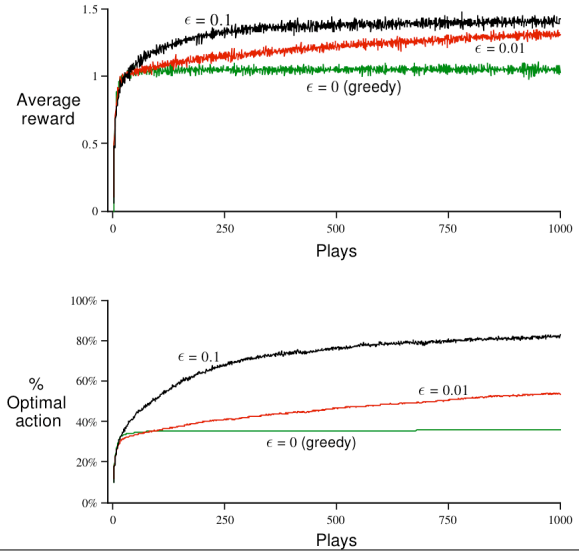
$$a_t = \begin{cases} a_t^* & \text{with probability } 1 - \varepsilon \\ \text{random action with probability} & \varepsilon \end{cases}$$

. . . the simplest way to balance exploration and exploitation

10-Armed Testbed

- *n* = 10 possible actions
- Each $Q^*(a)$ is chosen randomly from a normal distribution with mean 0 and variance 1
- each r_t is also normal, with mean $Q^*(a_t)$ and variance 1
- 1000 plays
- repeat the whole thing 2000 times and average the results
- Use sample average to estimate Q

ε-Greedy Methods on the 10-Armed Testbed



Softmax Action Selection

- Softmax action selection methods grade action probs. by estimated values.
- The most common softmax uses a Gibbs, or Boltzmann, distribution:

Choose action a on play t with probability

$$\frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{n} e^{Q_t(b)/\tau}}$$

where τ is the "computational temperature"

- Actions with greater value are more likely to be selected

Softmax and 'Temperature'

Choose action a on play t with probability

$$\frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{n} e^{Q_t(b)/\tau}}$$

where τ is the "computational temperature"

Probability ↘	$Q(a_1) = 1.0$	$Q(a_2) = 2.0$	$Q(a_3) = -3.0$
$\tau = 0.25$	0.0180	0.9820	< 0.0001
$\tau = 0.5$	0.1192	0.8808	< 0.0001
$\tau = 1.0$	0.2676	0.7275	0.0049
$\tau = 10.0$	0.3603	0.3982	0.2415
$\tau = 100.0$	0.3366	0.3400	0.3234

Small τ is like 'max.' Big τ is like 'uniform.'

Incremental Implementation

Recall the sample average estimation method:

The average of the first k rewards is (dropping the dependence on a):

$$Q_k = \frac{r_1 + r_2 + \cdots r_k}{k}$$

Can we do this incrementally (without storing all the rewards)?

We could keep a running sum and count, or, equivalently:

$$Q_{k+1} = Q_k + \frac{1}{k+1}\left[r_{k+1} - Q_k\right]$$

This is a common form for update rules:

NewEstimate = *OldEstimate* + *StepSize*[*Target* – *OldEstimate*]

Tracking a Nonstationary Problem

Choosing Q_k to be a sample average is appropriate in a stationary problem,

i.e., when none of the $Q^*(a)$ change over time,

But not in a nonstationary problem.

Better in the nonstationary case is:

$$Q_{k+1} = Q_k + \alpha\,[r_{k+1} - Q_k]$$

for *constant* α, $0 < \alpha \leq 1$

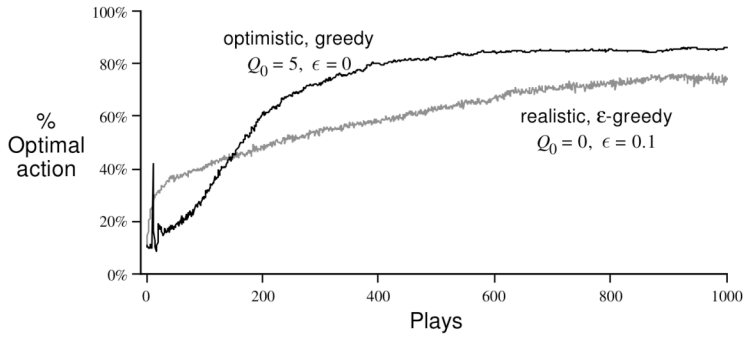
$$= (1-\alpha)^k Q_0 + \sum_{i=1}^{k} \alpha(1-\alpha)^{k-i} r_i$$

exponential, recency-weighted average

Optimistic Initial Values

- All methods so far depend on $Q_0(a)$, i.e., they are **biased**.
- Suppose instead we initialize the action values optimistically,

  i.e., on the 10-armed testbed, use $Q_0(a) = 5$ for all a



Conclusions

- These are all very simple methods
  - but they are complicated enough—we will build on them
  - we should understand them *completely*

Conclusions

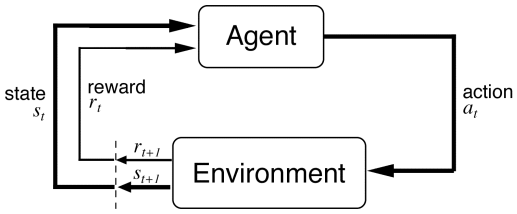
- These are all very simple methods
  - but they are complicated enough—we will build on them
  - we should understand them *completely*
- Ideas for improvements:
  - estimating uncertainties . . . interval estimation
  - new "action elimination" methods (see ICML'03)
  - approximating Bayes optimal solutions
  - Gittens indices
- The full RL problem offers some ideas for solution . . .
  - see work of Duff, e.g., at ICML'03, or Tao Wang

Chapter 3: The Reinforcement Learning Problem

Objectives of this chapter:

- describe the RL problem we will be studying for the remainder of the course
- present idealized form of the RL problem for which we have precise theoretical results;
- introduce key components of the mathematics: value functions and Bellman equations;
- describe trade-offs between applicability and mathematical tractability.

The Agent-Environment Interface



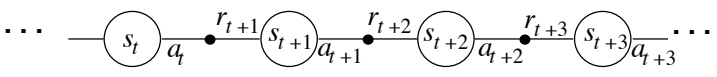
Agent and environment interact at discrete time steps : $t = 0, 1, 2, \ldots$

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

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

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

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 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.
- Reward computation is in the agent's environment because the agent cannot change it arbitrarily.
- The environment is not necessarily unknown to the agent, only incompletely controllable.

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.

The reward hypothesis

- That all of what we mean by goals and purposes can be well thought of as the maximization of the cumulative sum of a received scalar signal (reward)
- A sort of *null hypothesis*.
  - Probably ultimately wrong, but so simple we have to disprove it before considering anything more complicated

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.

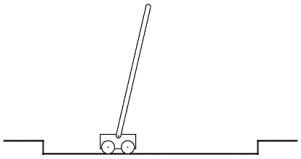
Discounted return:

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

where $\gamma, 0 \le \gamma \le 1$, 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:

reward = +1 for each step before failure

$\Rightarrow$ return = number of steps before failure

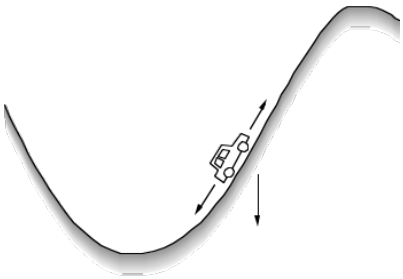
As a **continuing task** with discounted return:

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.

Another Example



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

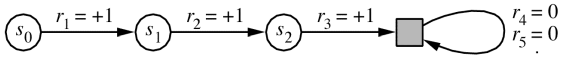
reward = −1 for each step where **not** at top of hill

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

Return is maximized by minimizing number of steps to 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 to 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 <u>all</u> 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**:

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

  - **reward probabilities**:

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

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

Recycling Robot MDP

$S = \{\texttt{high}, \texttt{low}\}$

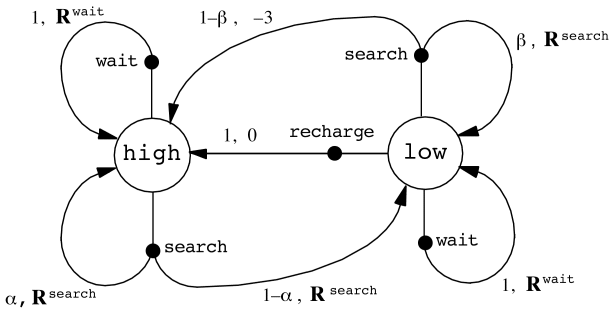
$A(\texttt{high}) = \{\texttt{search}, \texttt{wait}\}$

$A(\texttt{low}) = \{\texttt{search}, \texttt{wait}, \texttt{recharge}\}$

$\mathbf{R}^{\texttt{search}}$ = expected no. of cans while searching

$\mathbf{R}^{\texttt{wait}}$ = expected no. of cans while waiting

$\mathbf{R}^{\texttt{search}} > \mathbf{R}^{\texttt{wait}}$



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:

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

Or, without the expectation operator:

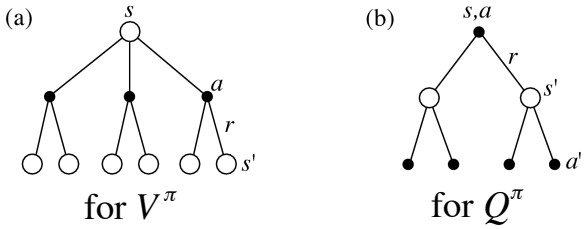
$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} \mathbf{P}^a_{ss'} \left[\mathbf{R}^a_{ss'} + \gamma V^\pi(s') \right]$$

More on the Bellman Equation

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

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

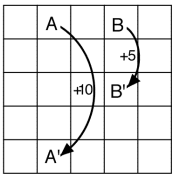
Backup diagrams:



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.



(a)

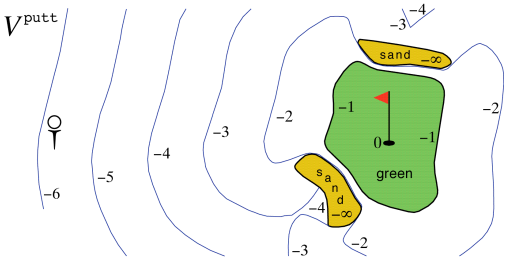
3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

(b)

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

Golf

- State is ball location
- Reward of –1 for each stroke until the ball is in the hole
- Value of a state?
- Actions:
  - `putt` (use putter)
  - `driver` (use driver)
- `putt` succeeds anywhere on the green

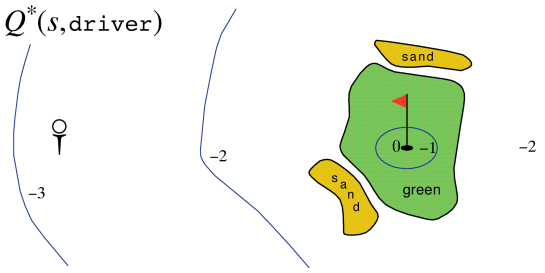


Optimal Value Functions

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

Optimal Value Function for Golf

- We can hit the ball farther with `driver` than with `putter`, but with less accuracy
- $Q^*(s,$`driver`$)$ gives the value or using `driver` first, then using whichever actions are best

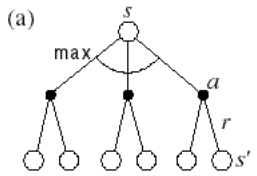


Bellman Optimality Equation for V^*

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

$$\begin{aligned} V^*(s) &= \max_{a \in A(s)} Q^{\pi^*}(s,a) \\ &= \max_{a \in A(s)} E\left\{ r_{t+1} + \gamma V^*(s_{t+1}) \,\middle|\, s_t = s, a_t = a \right\} \\ &= \max_{a \in A(s)} \sum_{s'} \mathbf{P}^a_{ss'} \left[\mathbf{R}^a_{ss'} + \gamma V^*(s') \right] \end{aligned}$$

The relevant backup diagram:

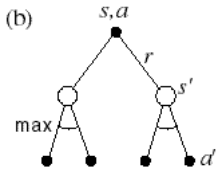


V^* is the unique solution of this system of nonlinear equations.

Bellman Optimality Equation for Q^*

$$\begin{aligned} Q^*(s,a) &= E\left\{ r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \,\middle|\, s_t = s, a_t = a \right\} \\ &= \sum_{s'} \mathbf{P}^a_{ss'} \left[\mathbf{R}^a_{ss'} + \gamma \max_{a'} Q^*(s', a') \right] \end{aligned}$$

The relevant backup diagram:



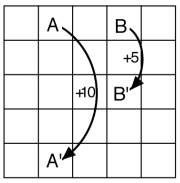
Q^* is the unique solution of this system of nonlinear equations.

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.

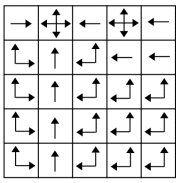
E.g., back to the gridworld:



a) gridworld

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^*

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 and 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.

Summary

- Agent-environment interaction
  - States
  - Actions
  - Rewards
- Policy: stochastic rule for selecting actions
- Return: the function of future rewards agent tries to maximize
- Episodic and continuing tasks
- Markov Property
- 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