Reinforcement Learning

Lecture 2: Markov Decision Processes (MDPs)

Instructor: Chongjie Zhang

Tsinghua University

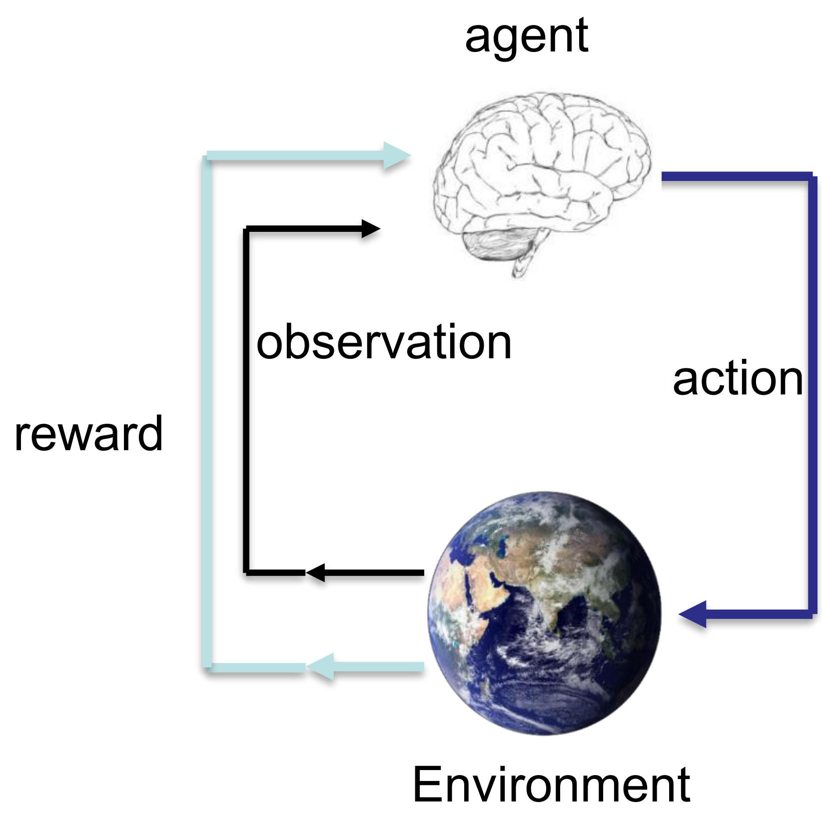
Reinforcement

- Behavior is primarily shaped by reinforcement rather than free-will
  - **Positive reinforcement** is the strengthening of behavior by the occurrence of some event
  - **negative reinforcement** is the strengthening of behavior by the removal or avoidance of some aversive event
- Behaviors that result in praise/pleasure tend to repeat, and behaviors that result in punishment/pain tend to become extinct.

Burrhus F. Skinner
1904-1990
Harvard psychology

Reinforcement Learning (RL)

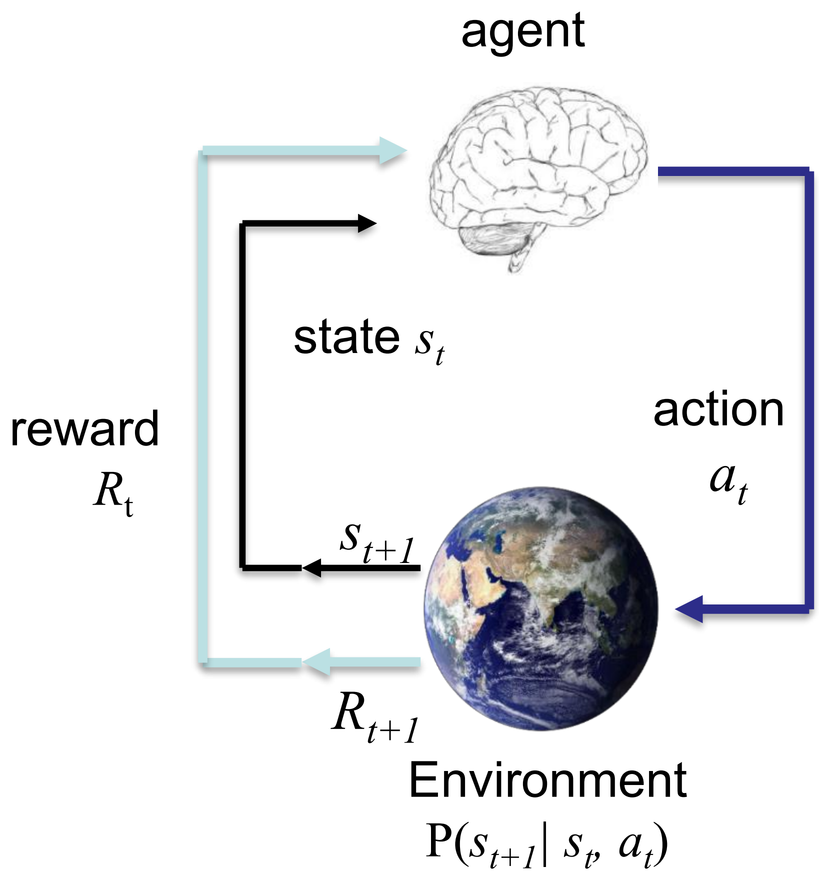
- A computational framework for behavior learning through reinforcement
  - RL is for an agent with the capacity to act
  - Each action influences the agent's future observation
  - Success is measured by a scalar reward signal
  - Goal: find a policy that maximizes expected total rewards



- Mathematical Model: Markov Decision Processes (MDP)

Markov Decision Process (MDP)

- An Finite MDP is defined by:
  - A finite set of states s $\in$ S
  - A finite set of actions a $\in$ A
  - A transition function T(s, a, s')
    - Probability that a from s leads to s', i.e., P(s'| s, a)
    - Also called the model or the dynamics
  - A reward function R(s) (Sometimes R(s, a) or R(s, a, s'))
  - A start state
  - Maybe a terminal state
- A model for sequential decision making problem under uncertainty



Assumptions

- **First-Order Markovian dynamics** (history independence)
  - Next state only depends on current state and current action

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \ldots S_0 = s_0)$$
$$= \; P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

- **State-Dependent Reward**
  - Reward is a deterministic function of current state
- **Stationary dynamics:** do not depend on time
  - $P(S_{t+1} \mid A_t, S_t) = P(S_{k+1} \mid A_k, S_k)$ for all t, k
- **Full observability**
  - Though we can't predict exactly which state we will reach when we execute an action, after the action is executed, we know the new state

States

- Experience is a sequence of observations, actions, rewards
  - $o_1, r_1, a_1, \ldots, a_{t-1}, o_t, r_t$
- Observations : the (raw) input of the agent's sensors, e.g., images, tactile signals, waveforms, etc.
- The state is a summary of experience
  - $s_t = f(o_1, r_1, a_1, \ldots, a_{t-1}, o_t, r_t)$
- The state can include immediate "observations," highly processed observations, and structures built up over time from sequences of observations, memories etc.
- In a fully observed environment, $s_t = f(o_t)$

Actions

- They are used by the agent to interact with the world.
- They can have many different temporal granularities and abstractions.
- Actions can defined to be
  - The instantaneous torques on the gripper
  - The instantaneous gripper translation, rotation, opening
  - Instantaneous forces applied to the objects
  - Short sequences of the above

Rewards

- They are scalar values provided by the environment to the agent that indicate whether goals have been achieved,
  - e.g., 1 if goal is achieved, 0 otherwise, or -1 for overtime step the goal is not achieved
- Rewards specify what the agent needs to achieve, not how to achieve it.
- The simplest and cheapest form of supervision, and surprisingly general
- Dense rewards are always preferred if available
  - e.g., distance changes to a goal.

Dynamics or The Environment Model

- How the state change given the current state and action

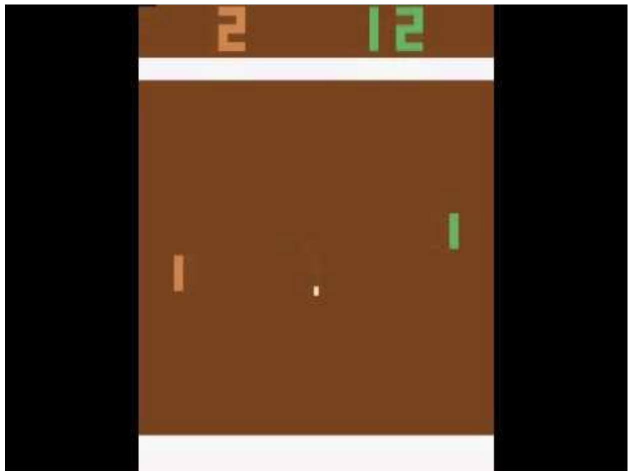
$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

- Modeling the uncertainty

- Two problems:
  - Planning: the dynamics model is known
  - Reinforcement learning: the dynamics model is unknown

Example: Atari games

- States: raw image frames
- Actions: playing joysticks (18 actions)
- Reward: score changes



Example: Go

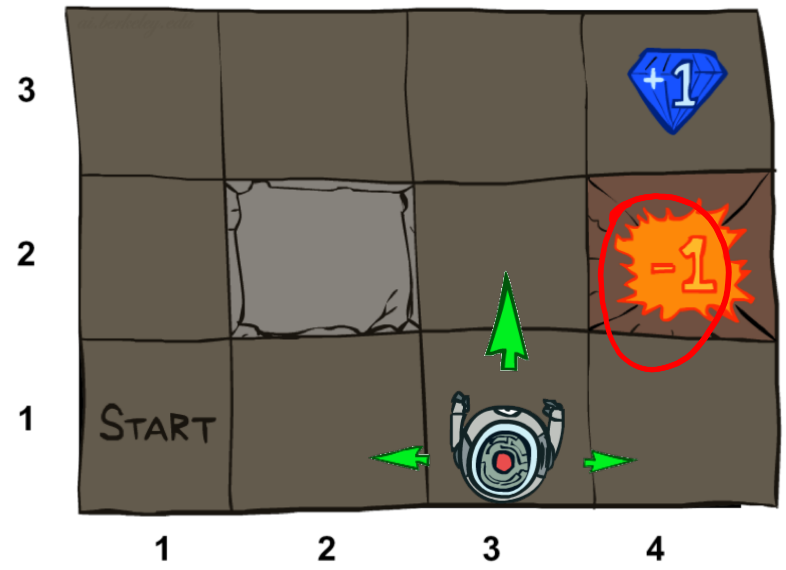
- States: features of the game board
- Actions: place a stone or resign
- Rewards: win +1, lose -1, otherwise 0

Example: Learning to Drive

- States: speed, direction, traffic, weather,…
- Actions: steer, brake, throttle
- Rewards:
  - +1: reaching goal
  - -1: honking from surrounding driver
  - -100: collision

Example: Grid World

- A maze-like problem
  - The agent lives in a grid
- States: the position of the agent
- Noisy actions: east, south, west, north
- Dynamics: actions not always go as planned
  - 80% of the time, the action North takes the agent North (if there is a wall, it stays)
  - 10% of the time, North takes the agent West; 10% East
- Rewards the agent receives each time step
  - Small "living" reward each step (can be negative)
  - Big rewards come at the end (good or bad)



What is a solution to an MDP?

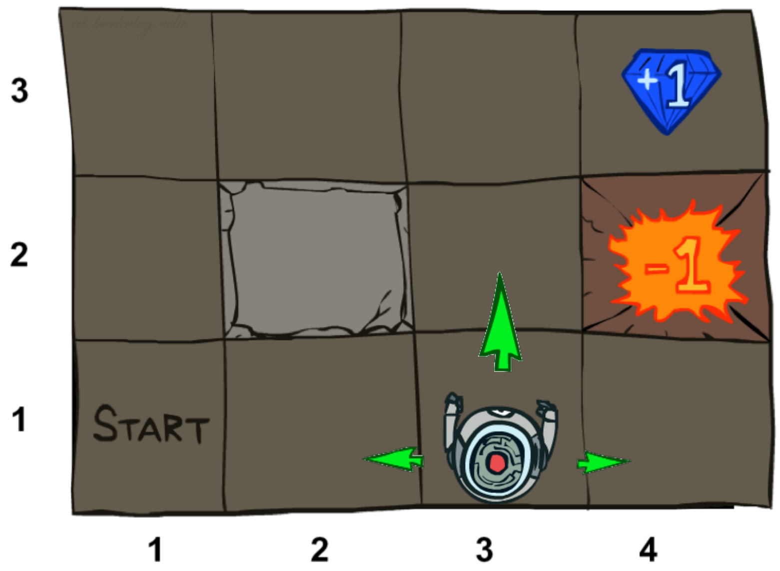
MDP Planning Problem:

Input: an MDP (S,A,R,T)
Output: ????

- Should the solution to an MDP from an initial state be just a sequence of actions such as (a_1, a_2, a_3, ...) ?

Example: Grid World

- Action sequence: north, north, east



What is a solution to an MDP?

MDP Planning Problem:

Input: an MDP (S,A,R,T)
Output: ????

- Should the solution to an MDP from an initial state be just a sequence of actions such as (a_1, a_2, a_3, ...) ?
- No! In general an action sequence is not sufficient
  - Actions have stochastic effects, so the state we end up in is uncertain
  - This means that we might end up in states where the remainder of the action sequence doesn't apply or is a bad choice
  - A solution should tell us what the best action is for any possible situation/state that might arise

MDP: Non-Stationary Policy

- A solution to an MDP is a policy
  - Two types of policies: **nonstationary** and **stationary**
- Nonstationary policies are used when we are given a finite planning horizon H
  - i.e. we are told how many actions we will be allowed to take
- Nonstationary policies are functions from states and times to actions
  - π: S x T → A, where T is the non-negative integers
  - π(s, t) tells us what action to take at state s when there are t stages-to-go (note that we are using the convention that t represents stages/decisions to go, rather than the time step)

MDP: Stationary Policy

- What if we want to continue taking actions indefinately?
  - Use stationary policies
- A Stationary policy is a mapping from states to actions
  - $\pi: S \rightarrow A$
  - $\pi(s)$ is action to do at state s (regardless of time)
  - specifies a continuously reactive controller
- Note that both nonstationary and stationary policies assume or have these properties:
  - full observability of the state
  - history-independence
  - deterministic action choice

What is a solution to an MDP?

MDP Planning Problem:

Input: an MDP (S,A,R,T)
Output: a policy such that ????

- We don't want to output just any policy
- We want to output a "good" policy
- One that accumulates a lot of reward

Value of a Policy

- How good is a policy π?
  - How do we measure reward "accumulated" by π?
- **Value function** $V: S \rightarrow$ R associates value with each state (or each state and time for non-stationary π)
- $V_{\pi}(s)$ denotes value of policy π at state s
  - Depends on immediate reward, but also what you achieve subsequently by following π
  - An **optimal policy** is one that is no worse than any other policy at any state
- The goal for a MDP is to compute or learn an optimal policy

What is a solution to an MDP?

MDP Planning Problem:

Input: an MDP (S,A,R,T)
Output: a policy that achieves an "optimal value"

- This depends on how we define the value of a policy
- There are several choices and the solution algorithms depend on the choice
- We will consider two common choices
  - Finite-Horizon Value
  - Infinite Horizon Discounted Value

Finite-Horizon Value Functions

- We first consider maximizing expected total reward over a finite horizon
- Assumes the agent has H time steps to live (that is, it gets to take H actions)
- To act optimally, should the agent use a stationary or non-stationary policy?
  - i.e. Should the action it takes depend on absolute time?
- Put another way:
  - If you had only one week to live would you act the same way as if you had fifty years to live?

Finite Horizon Problems

- Value (utility) depends on stage-to-go
  - Hence use a nonstationary policy!
- $V_\pi^k(s)$ is k-stage-to-go value function for non-stationary π
  - expected total reward for executing π starting in s for k time steps

$$V_\pi^k(s) = E\,[\sum_{t=0}^{k} R^t \mid \pi, s\,]$$

$$= E\,[\sum_{t=0}^{k} R(s^t) \mid a^t = \pi(s^t, k-t), s^0 = s\,]$$

$R^t = R(s^t)$

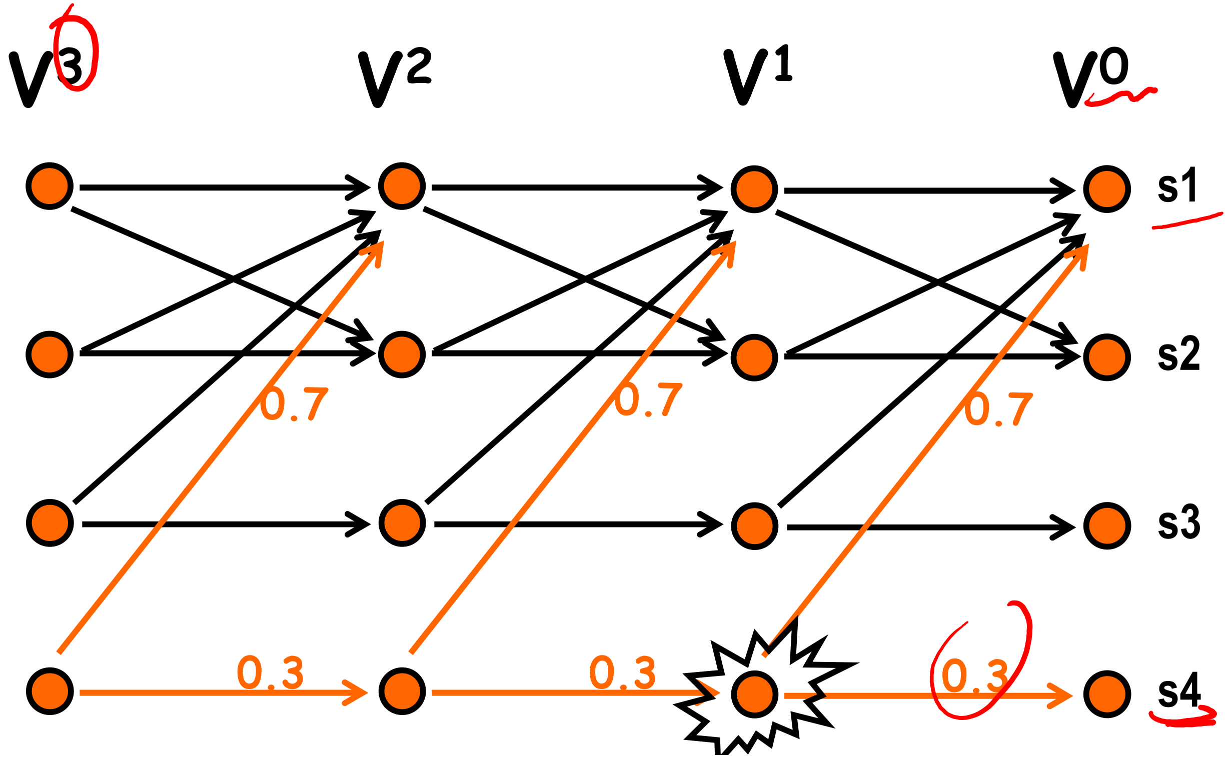
$s^{t+1} \sim p(s^{t+1} \mid s^t, a^t)$

- Here R_t and s_t are random variables denoting the reward received and state at time-step t when starting in s
  - These are random variables since the world is stochastic

Computational Problems

- There are two problems that we will be interested in solving
- Policy evaluation:
  - Given an MDP, a nonstationary policy π and a horizon H
  - Compute finite-horizon value function $V_\pi^k(s)$ for any k <= H
- Policy optimization:
  - Given an MDP and a horizon H
  - Compute the optimal finite-horizon policy
  - We will see this is equivalent to computing the optimal value function

Finite-Horizon Policy Evaluation



$V^0(s) = R(s)$

$V^1(s4) = R(s4) + 0.7\, V^0\,(s1) + 0.3\, V^0\,(s4)$

Finite-Horizon Policy Evaluation

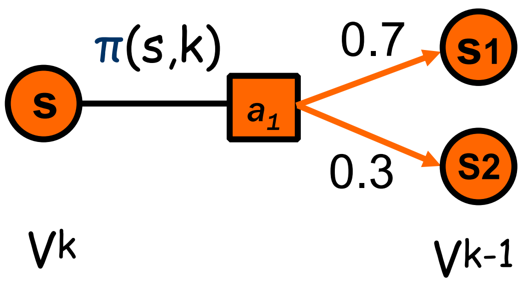
- Can use dynamic programming to compute $V_\pi^k(s)$
  - Markov property is critical for this

(k=0) $V_\pi^0(s) = R(s), \quad \forall s$

$n = |S|$

(k>0) $V_\pi^k(s) = \underbrace{R(s)}_{\text{immediate reward}} + \underbrace{\sum_{s'} T(s,\pi(s,k),s') \cdot V_\pi^{k-1}(s')}_{\text{expected future payoff with } k\text{-1 stages to go}}, \quad \forall s$

$O(H n^2)$

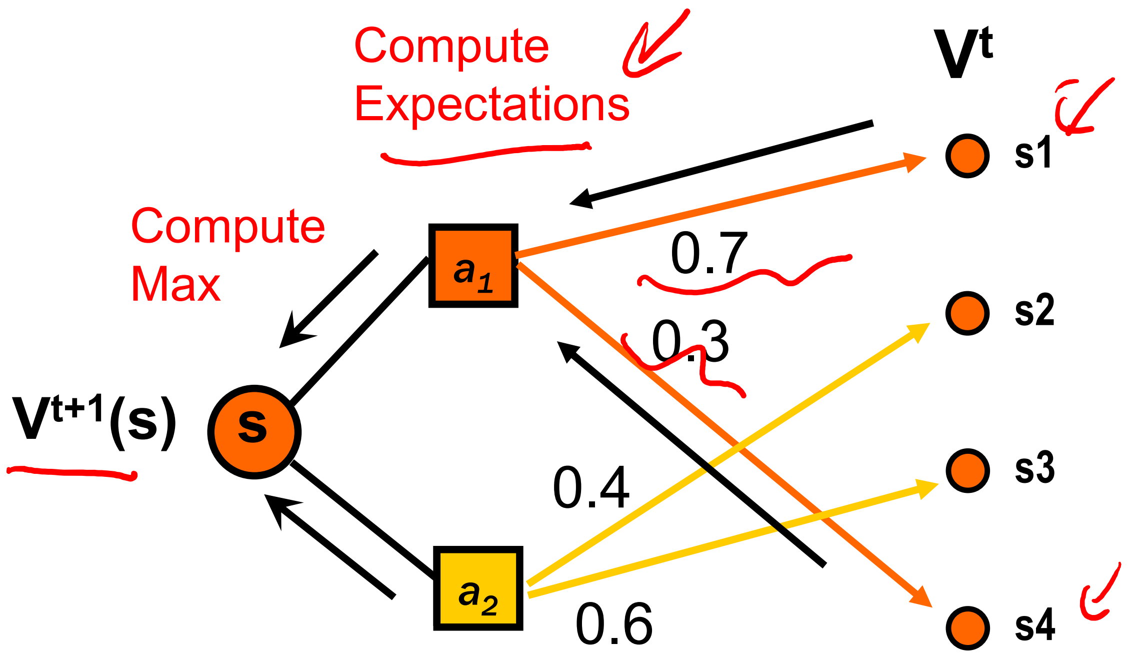


Computational Problems

- There are two problems that we will be interested in solving
- Policy evaluation:
  - Given an MDP, a nonstationary policy π, and a horizon H
  - Compute finite-horizon value function $V_{\pi}^{k}(s)$ for any k <= H
- **Policy optimization**:
  - Given an MDP and a horizon H
  - Compute the optimal finite-horizon policy
  - We will see this is equivalent to computing optimal value function
- How many finite horizon policies are there? |A| , |S| , H
  - $|A|^{Hn}$
  - So can't just enumerate policies for efficient optimization

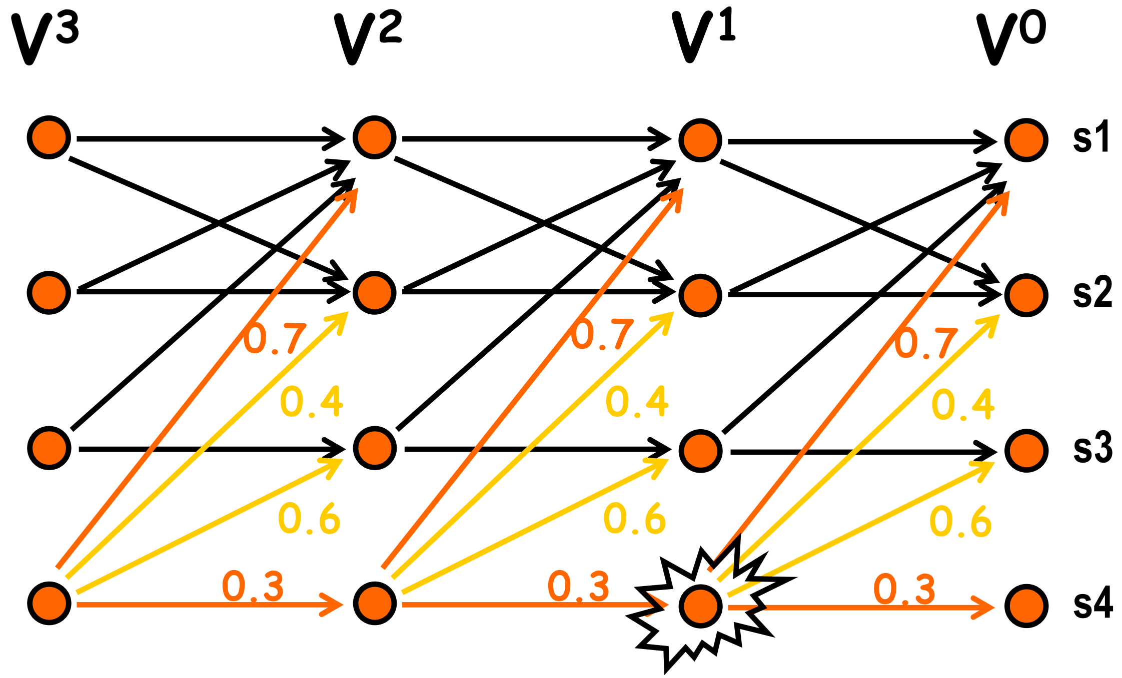
Policy Optimization: Bellman Backups

How can we compute the optimal $\mathbf{V^{t+1}(s)}$ given optimal $\mathbf{V^t}$?



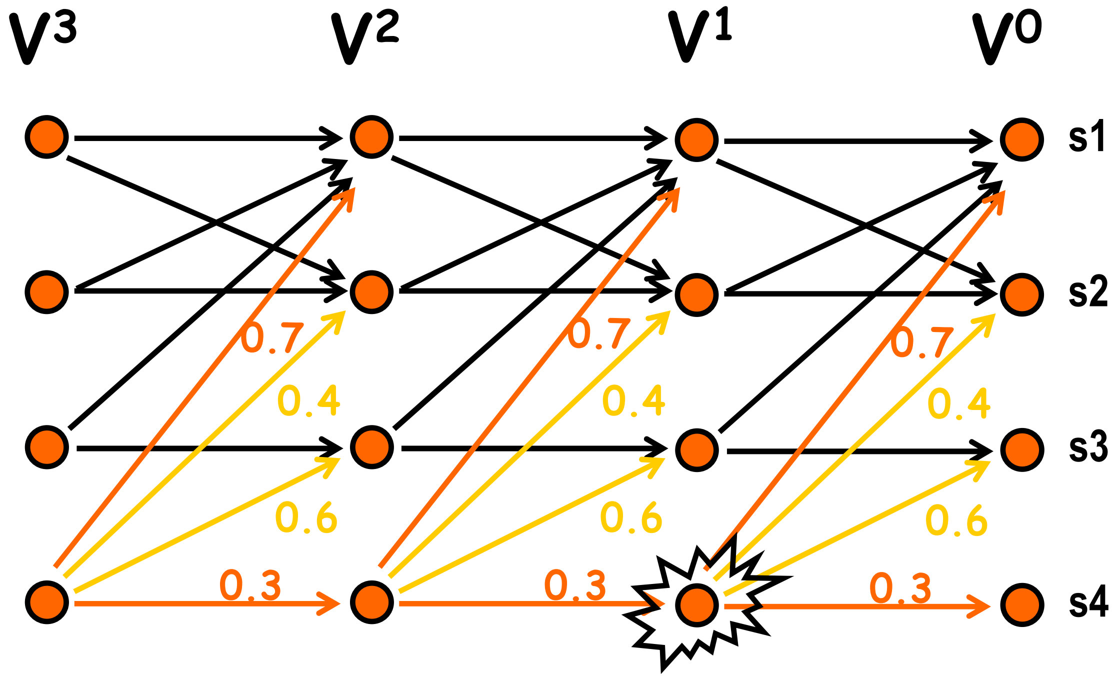
$$V^{t+1}(s) = R(s) + \max \left\{ \begin{array}{l} 0.7\, V^t(s1) + 0.3\, V^t(s4) \\ 0.4\, V^t(s2) + 0.6\, V^t(s3) \end{array} \right\}$$

Value Iteration



$$V^1(s4) = R(s4) + \max \{ 0.7\, V^0(s1) + 0.3\, V^0(s4), \quad 0.4\, V^0(s2) + 0.6\, V^0(s3) \}$$

Value Iteration



$P^*(s4,t) = \max \{\square\ \square\}$

Value Iteration: Finite Horizon Case

- Markov property allows exploitation of DP principle for optimal policy construction
  - no need to enumerate $|A|^{Hn}$ possible policies
- Value Iteration

Bellman backup

$$V^0(s) = R(s), \quad \forall s$$

$$\boxed{V^k(s) = R(s) + \max_a \sum_{s'} T(s,a,s') \cdot V^{k-1}(s')}$$

$n \times n \times m \times H$

$$\pi^*(s,k) = \arg\max_a \sum_{s'} T(s,a,s') \cdot V^{k-1}(s')$$

V^k is optimal k-stage-to-go value function

Π*(s,k) is optimal k-stage-to-go policy

Value Iteration: Complexity

- Note how DP is used
  - optimal solution to k-1 stage problem can be used without modification as part of optimal solution to k-stage problem
- What is the computational complexity?
  - H iterations
  - At each iteration, each of n states, computes expectation for m actions
  - Each expectation takes O(n) time
- Total time complexity: $O(Hmn^2)$
  - Polynomial in number of states. Is this good?

Summary: Finite Horizon

- Resulting policy is optimal

$$V_{\pi^*}^k(s) \geq V_\pi^k(s), \quad \forall \pi, s, k$$

  - convince yourself of this (use induction on k)

- Note: optimal value function is unique.

- Is the optimal policy unique?
  - No. Many policies can have same value (there can be ties among actions during Bellman backups).

Horizon Matters

Live everyday as if it is the last day.   -- Steve Jobs

Infinite Horizon MDPs

Infinite Horizon MDPs

- Defining value as total reward is problematic with infinite horizons
  - many or all policies have infinite expected reward
  - some MDPs are ok (e.g., zero-cost absorbing states)
- "Trick": introduce discount factor $0 \leq \gamma < 1$
  - future rewards discounted by γ per time step

$$V_\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R^t \mid \pi, s\right]$$

Bounded Value

- Note:

$$V_\pi(s) \leq E\left[\sum_{t=0}^{\infty} \gamma^t R^{\max}\right] = \frac{1}{1-\gamma} R^{\max}$$

Notes: Discounted Infinite Horizon

Banach fixed-point theorem

- Optimal policies guaranteed to exist (Howard, 1960)
  - i.e. there is a policy that maximizes value at each state
- Furthermore there is always an optimal stationary policy
  - Intuition: why would we change action at s at a new time when there is always forever ahead
  - We define $V^*(s) = V_\pi(s)$ for some optimal stationary π

Policy Evaluation

- Value equation for a fixed policy
  - Immediate reward + Expected discounted future reward

$$\boxed{V_\pi(s)} = R(s) + \gamma \sum_{s'} T(s,\pi(s),s') \cdot V_\pi(s')$$

Bellman equation

derive this from original definition

$$V_\pi(s) = E\left(\sum_{t=0} \lambda^t R_t \mid \pi, s\right)$$

- How can we compute the value function for a policy?
  - we are given R and T
  - linear system with n variables and n constraints
    - Variables are values of states: V(s1),…,V(sn)
    - Constraints: one value equation (above) per state
  - Use linear algebra to solve for V (e.g. matrix inverse)

Policy Evaluation via Matrix Inverse

V$_\pi$ and ***R*** are n-dimensional column vector (one element for each state)

T is an n x n matrix s.t. $\mathrm{T}(\mathrm{i}, \mathrm{j}) = \mathrm{T}(\mathrm{s}_\mathrm{i}, \pi(s_i), s_j)$

$$V_\pi = R + \gamma \mathrm{T} V_\pi$$

$$\Downarrow$$

$$(I - \gamma \mathrm{T}) V_\pi = R$$

$$\Downarrow$$

$$V_\pi = (I - \gamma \mathrm{T})^{-1} R$$

Computing an Optimal Value Function

- **Bellman equation** for the optimal value function

$P(s' \mid s, a)$

$$V^*(s) = R(s) + \gamma \max_a \sum_{s'} \boxed{T(s,a,s')} \cdot V^*(s')$$

  - Bellman proved this is always true for an optimal value function
- How can we compute the optimal value function? $\pi^*(s)$
  - The MAX operator makes the system non-linear, so the problem is more difficult than policy evaluation
- Notice that the optimal value function is a **fixed-point** of the Bellman Backup operator B (i.e. B[V*]=V*)

$V^{k+1} = B V^k$

  - B takes a value function as input and returns a new value function

$$B[V](s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s') \cdot V(s')$$

Computing the optimal policy

How do we compute the optimal policy? (or equivalently, the optimal value function?)

Approach #1: **value iteration**: repeatedly update an estimate of the optimal value function according to Bellman optimality equation

1. Initialize an estimate for the value function arbitrarily

$$\hat{V}(s) \leftarrow 0, \;\; \forall s \in \mathcal{S}$$

$\hat{V}(s) \leftarrow R(s)$

2. Repeat, update:

$T(s,a,s')$

$T(s'|s,a)$

$$\hat{V}(s) \leftarrow R(s) + \gamma \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a)\,\hat{V}(s'), \;\; \forall s \in \mathcal{S}$$

Illustration of value iteration

Running value iteration with $\gamma = 0.9$

0	0	0	1
0		0	-100
0	0	0	0

Original reward function

$1 + \max\{ \quad \}$

$1 + (0.8 \times 1 + 0.1 \times 1) \times 0.9$
$= 1.81$

$0 + (0.8 \times 1) \times 0.9$
$= 0.72$

Illustration of value iteration

Running value iteration with $\gamma = 0.9$

0	0	0.72	1.81
0		0	-99.91
0	0	0	0

$\hat{V}$ at one iteration

Illustration of value iteration

Running value iteration with $\gamma = 0.9$

0.809	1.598	2.475	3.745
0.268		0.302	-99.59
0	0.034	0.122	0.004

$\hat{V}$ at five iterations

Illustration of value iteration

Running value iteration with $\gamma = 0.9$

2.686	3.527	4.402	5.812
2.021		1.095	-98.82
1.390	0.903	0.738	0.123

$\hat{V}$ at 10 iterations

Illustration of value iteration

Running value iteration with $\gamma = 0.9$

$V(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s,a) V(s')$

$V(s)$

$V(s) \rightarrow V^*(s)$

$V^*(s) = V^{\pi^*}(s)$

$\pi(s) = \max_a \sum_{s'} p(s'|s,a) \, v(s')$

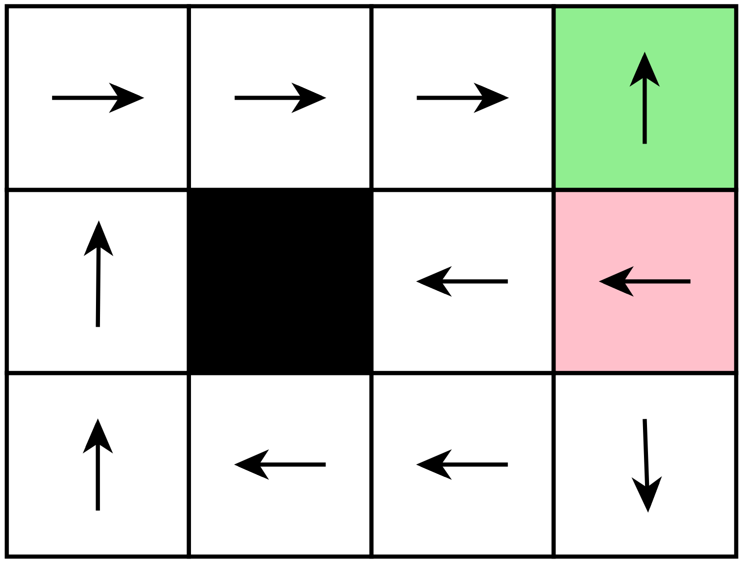
5.470	6.313	7.190	8.669
4.802		3.347	-96.67
4.161	3.654	3.222	1.526

$\hat{V}$ at 1000 iterations

Illustration of value iteration

$R(s) = 0$

Running value iteration with $\gamma = 0.9$

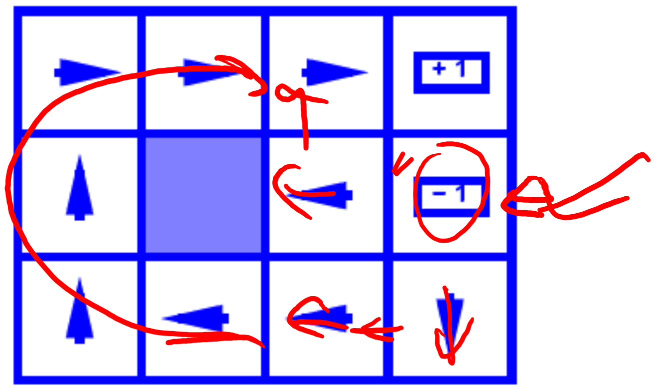


Resulting policy after 1000 iterations

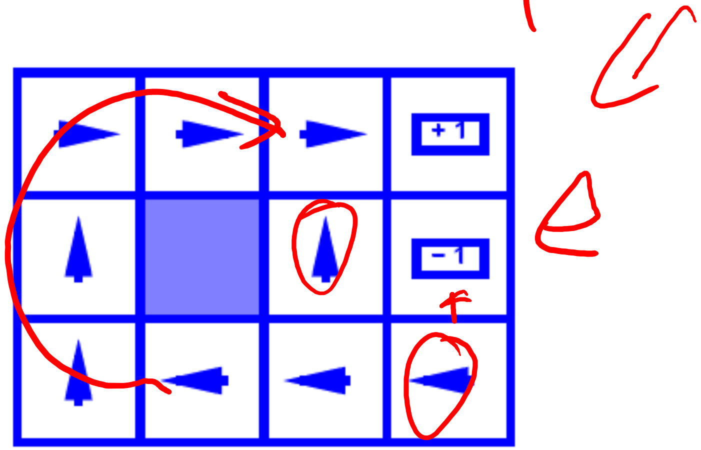
Optimal Policies

MDP1

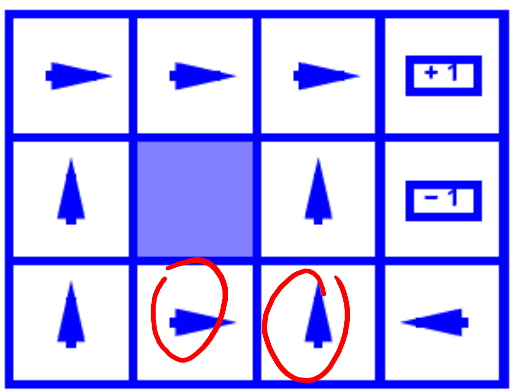
MDP2



R(s) = -0.01

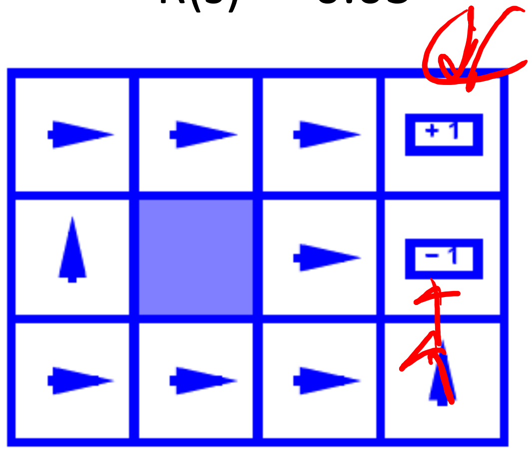


R(s) = -0.03



R(s) = -0.4

R(S) = -2



R(s) = -2.0