Reinforcement Learning
or,
Learning and Planning with Markov Decision Processes

295 Seminar, Fall 2019
Rina Dechter

Slides will follow David Silver's, and Sutton's book

Goals: To learn together the basics of RL.
Some lectures and classic and recent papers from the literature

Students will be active learners and teachers

Class Page

Demo

Detailed demo

My example: Driving to campus

- On Wednesday I drove to campus from LA after a year in a rented car.
- Mirrors were not adjusted, air-condition was not on, I am shortsighted, no glasses?
- Goal, to drive safely…
- Where is the AC control? How to control it?
- Where is the mirror control? How to control it?

- I have to sense, I do this while driving, dangerous.

Example of Learning from experience!

- Model is partial
- Partially observable due to short-sightedness
- Also there are changes from the past: more traffic due to some constructions, more traffic since I cannot drive on the carpool lane
- So, need to re-learn from new experience and use the previous model as input.

Topics

1. Introduction and Markov Decision Processes: Basic concepts. S&B chapters 1, 3. (myslides 2)

2. Planning Dynamic Programming – Policy Iteration, Value Iteration, S&B chapter 4, (myslides 3)

3. Monte-Carlo(MC) and Temporal Differences (TD): S&B chapters 5 and 6, (myslides 4, myslides 5)

4. Multi-step bootstrapping: S&B chapter 7, (myslides 4, last part, slides 6 Sutton)

5. Bandit algorithms: S&B chapter 2, (myslides 7 , sutton-based)

6. Exploration exploitation. (Slides: silver 9, Brunskill)

7. Planning and learning MCTS: S&B chapter 8, (slides Brunskill)

8. Function approximations S&B chapter 9,10,11, (slides: silver 6,7, Sutton 9,10,11)

9. Policy gradient methods: S&B chapter 13, (slides: silver 7, Sutton 13)

10. Causal Driven RL ???

Resources

- Book: Reinforcement Learning: An Introduction
  Richard S. Sutton and Andrew G. Barto
- UCL Course on Reinforcement Learning
  David Silver
- RealLife Reinforcement Learning
  Emma Brunskill
- Udacity course on Reinforcement Learning:
  Isbell, Littman and Pryby

References

Bertsekas, D. P. (2007a). *Dynamic Programming and Optimal Control*, volume 1. Athena Scientific, Belmont, MA, 3 edition.

Bertsekas, D. P. (2007b). *Dynamic Programming and Optimal Control*, volume 2. Athena Scientific, Belmont, MA, 3 edition.

Bertsekas, D. P. and Shreve, S. (1978). *Stochastic Optimal Control (The Discrete Time Case)*. Academic Press, New York.

Puterman, M. (1994). *Markov Decision Processes — Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY.

Singh, S. P. and Yee, R. C. (1994). An upper bound on the loss from approximate optimal-value functions. *Machine Learning*, 16(3):227–233.

Course Outline, Silver

- Part I: Elementary Reinforcement Learning
  1. Introduction to RL
  2. Markov Decision Processes
  3. Planning by Dynamic Programming
  4. Model-Free Prediction
  5. Model-Free Control
- Part II: Reinforcement Learning in Practice
  1. Value Function Approximation
  2. Policy Gradient Methods
  3. Integrating Learning and Planning
  4. Exploration and Exploitation
  5. Case study - RL in games

Introduction to Reinforcement Learnintg

Chapter 1 S&B

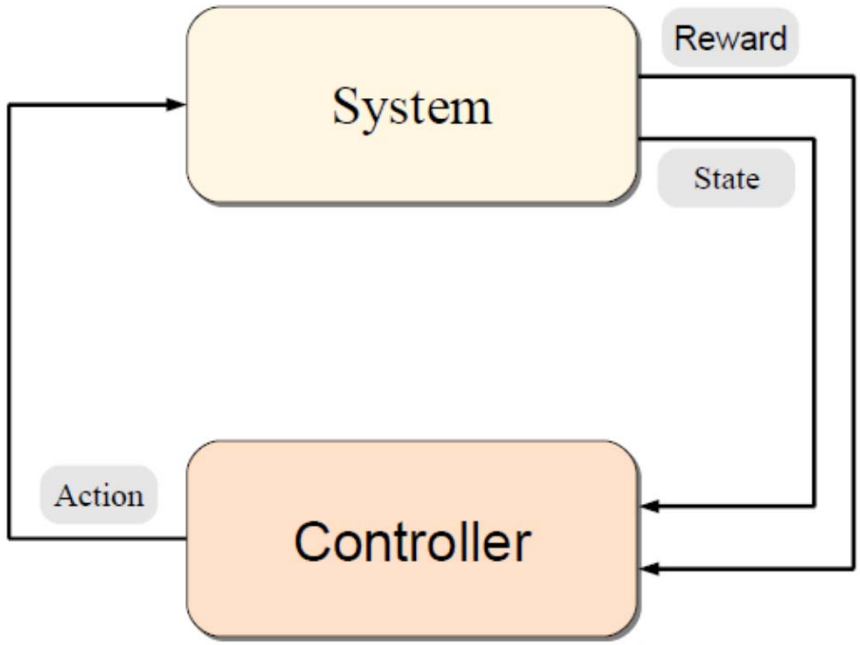
Reinforcement Learning

Learn a behavior strategy (policy) that maximizes the long term
Sum of rewards **in an unknown and stochastic environment (**Emma Brunskill:)

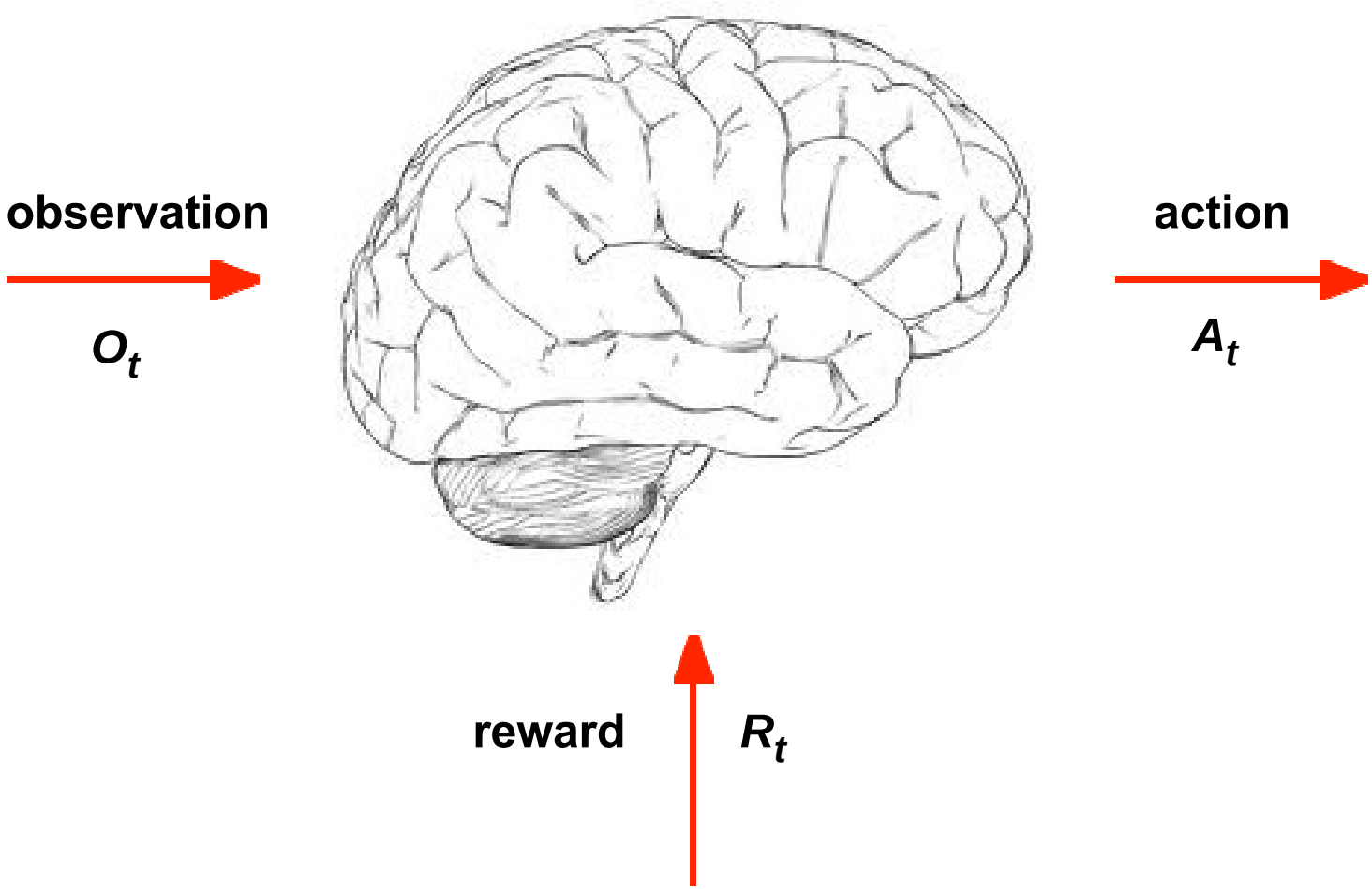
Planning under Uncertainty

Learn a behavior strategy (policy) that maximizes the long term
Sum of rewards **in a known stochastic environment (**Emma Brunskill:)

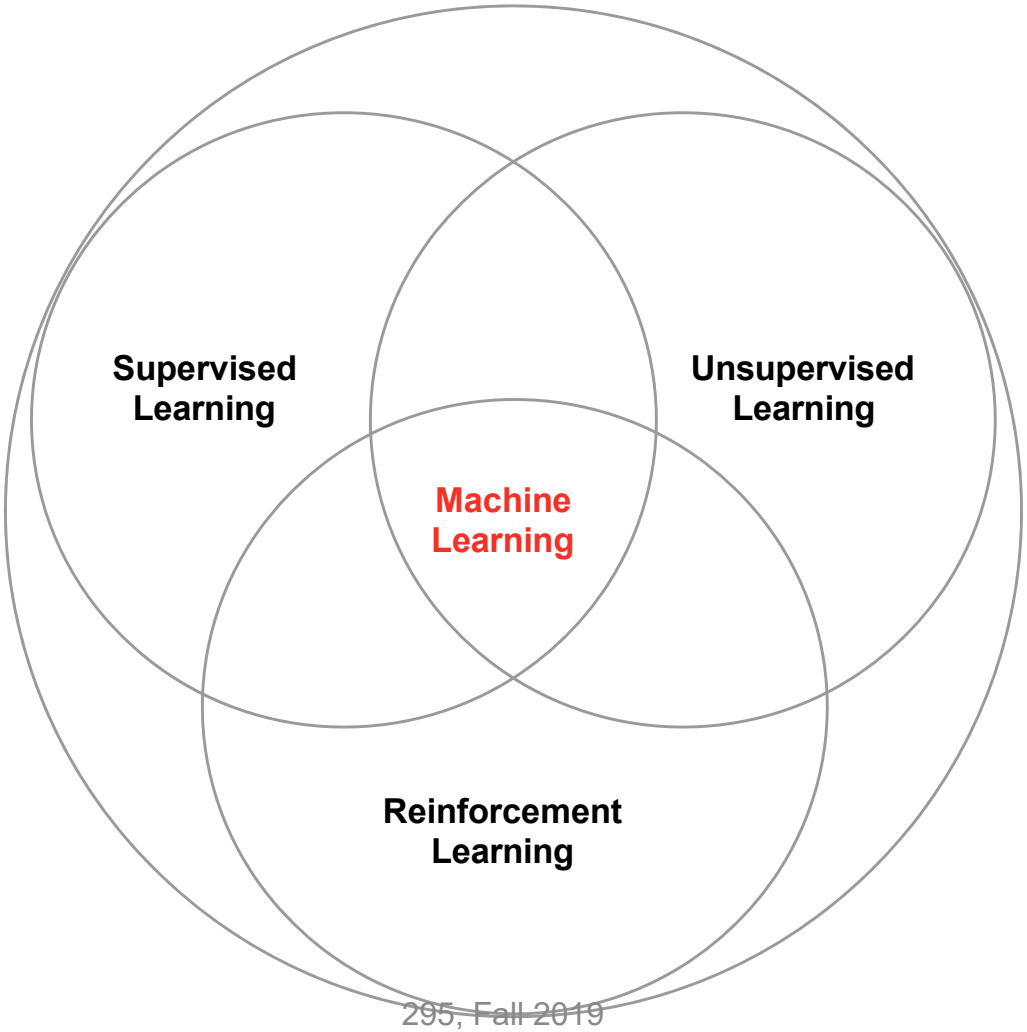
Reinforcement Learning



Agent and Environment



Branches of Machine Learning



Sequential Decision Making

Reward

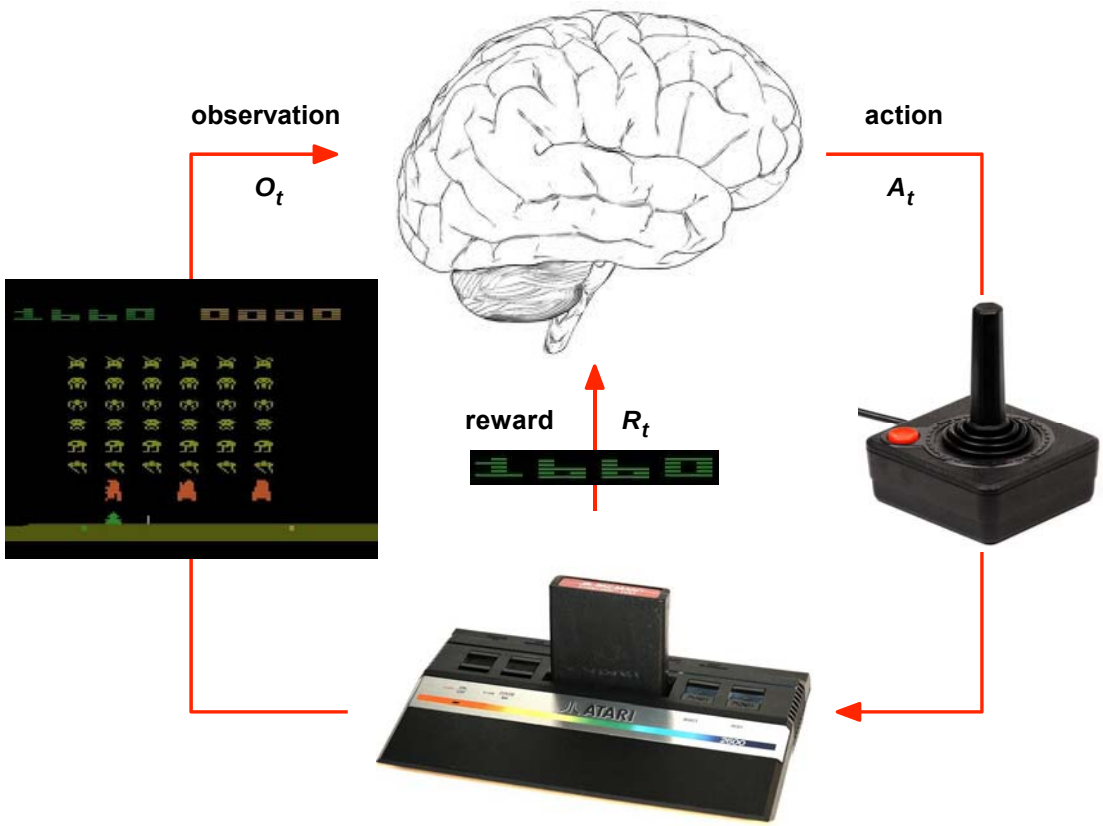
- Goal: *select actions to maximise total future reward*
- Actions may have long term consequences
- Reward may be delayed
- It may be better to sacrifice immediate reward to gain more long-term reward
- Examples:
  - A financial investment (may take months to mature)
  - Refueling a helicopter (might prevent a crash in several hours)
  - Blocking opponent moves (might help winning chances many moves from now)

My pet projects:

- **The academic commitment problem**. Given outside requests (committees, reviews, talks, teach…) what to accept and what to reject today?
- How to formulate as an MDP?
- **Baccarat:** See here

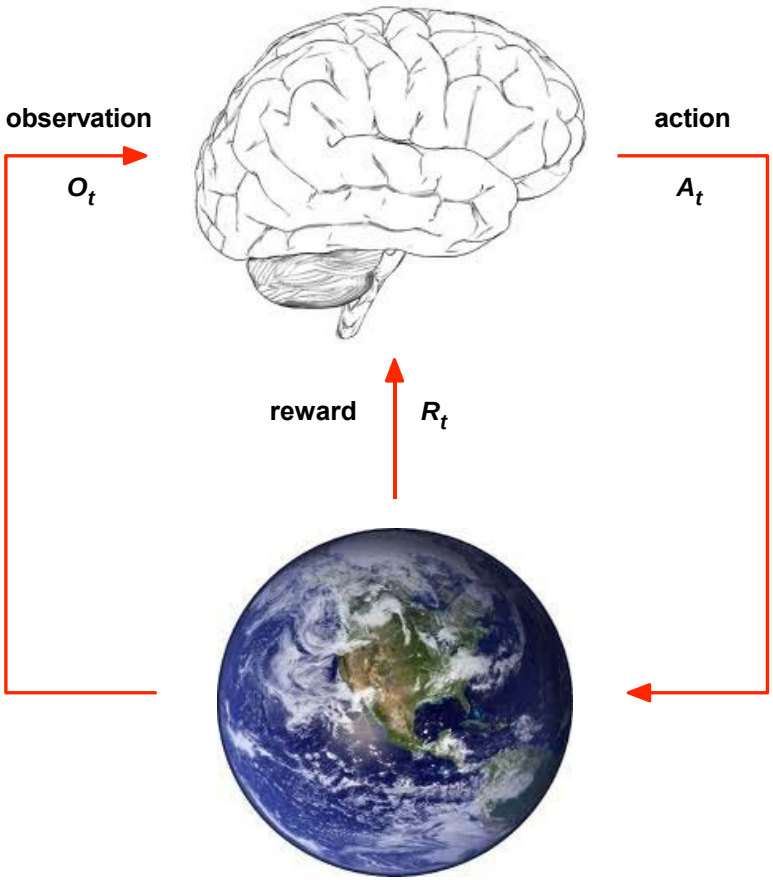
Examples: Robotics

Atari Example: Reinforcement Learning



- Rules of the game are unknown
- Learn directly from interactive game-play
- Pick actions on joystick, see pixels and scores

Agent and Environment



- At each step t the agent:
  - Executes action A_t
  - Receives observation O_t
  - Receives scalar reward R_t
- The environment:
  - Receives action A_t
  - Emits observation O_{t+1}
  - Emits scalar reward R_{t+1}
- t increments at env. step

Markov Decision Processes

In a nutshell (from Emma Brunskill):

MDP is a tuple (S,A,P,R,γ)

- Set of states S
- Start state s_0
- Set of actions A
- Transitions P(s'|s,a) (or T(s,a,s'))
- Rewards R(s,a,s') (or R(s) or R(s,a)
- Discount γ
- Policy = Choice of action for each state
- Utility / Value = sum of (discounted) rewards

Policy: $\pi(s) \rightarrow a$

Markov Decision Processes

Most of the story in a nutshell:

- States: S
- Model: T(s,a,s’) = P(s’|s,a)
- Actions: A(s), A
- Reward: R(s), R(s,a), R(s,a,s’)
- Discount: γ
- Policy: $\pi(s) \rightarrow a$
- Utility/Value: sum of discounted rewards.
- We seek optimal policy that maximizes the expected total (discounted) reward

Value and Q Functions

Most of the story in a nutshell:

- Value of a Policy
- $$V^{\pi}(s) = \sum_{s' \in S} p(s' \mid s, \pi(s)) \left[R(s, \pi(s), s') + \gamma V^{\pi}(s') \right]$$

$$Q^{\pi}(s,a) = \sum_{s' \in S} p(s' \mid s, a) \left[R(s, a, s') + \gamma V^{\pi}(s') \right]$$

- Optimal Value & Optimal Policy

$$V^*(s_i) = \max_a \left(\sum_{s_j \in S} p(s_j \mid s_i, a) \left[R(s, \pi(s), s') + \gamma V^*(s_j) \right] \right)$$

$$= \max_a Q^*(s,a)$$

$$\pi^*(s) = \operatorname{argmax}_a Q^*(s,a)$$

Most of the story in a nutshell:

Bellman Equation

$$V^*(s_i) = \max_a \left(\sum_{s_j \in S} p(s_j \mid s_i, a) \left[R(s, \pi(s), s') + \gamma V^*(s_j) \right] \right)$$

- Holds for V*
- Inspires an update rule

Most of the story in a nutshell:

Value Iteration

1. Initialize $V_1(s_i)$ for all states s_i
2. k=2
3. While k < desired horizon or (if infinite horizon) values have converged
   - For all s,

$$V_k(s_i) = \max_a \left(\sum_{s_j \in S} p(s_j \mid s_i, a) \left[R(s, \pi(s), s') + \gamma V_{k-1}(s_j) \right] \right)$$

$$\pi_k(s_i) = \operatorname*{argmax}_a \left(\sum_{s_j \in S} p(s_j \mid s_i, a) \left[R(s, \pi(s), s') + \gamma V_{k-1}(s_j) \right] \right)$$

Most of the story in a nutshell:

Will Value Iteration Converge?

- Yes, if discount factor is < 1 or end up in a terminal state with probability 1

- Bellman equation is a contraction
- If apply it to two different value functions, distance between value functions shrinks after apply Bellman equation to each

Most of the story in a nutshell:

Bellman Operator is a Contraction

|| V-V'|| = Infinity norm
(find max diff
Over all states)

$$\|BV - BV'\| = \left\| \begin{array}{l} \max_a \left[R(s,a) + \gamma \sum_{s_j \in S} p(s_j \mid s_i, a) V(s_j) \right] \\ - \max_{a'} \left[R(s,a') - \gamma \sum_{s_j \in S} p(s_j \mid s_i, a') V'(s_j) \right] \end{array} \right\|$$

$$\leq \left\| \max_a \left[R(s,a) + \gamma \sum_{s_j \in S} p(s_j \mid s_i, a) V(s_j) - R(s,a) + \gamma \sum_{s_j \in S} p(s_j \mid s_i, a) V'(s_j) \right] \right\|$$

$$\leq \gamma \left\| \max_a \left[\sum_{s_j \in S} p(s_j \mid s_i, a) V(s_j) - \sum_{s_j \in S} p(s_j \mid s_i, a) V'(s_j) \right] \right\|$$

$$= \gamma \max_a \left\| \left[\sum_{s_j \in S} p(s_j \mid s_i, a)(V(s_j) - V'(s_j)) \right] \right\|$$

$$\leq \gamma \max_{a, s_i} \sum_{s_j \in S} p(s_j \mid s_i, a) \left| V(s_j) - V'(s_j) \right|$$

$$\leq \gamma \max_{a, s_i} \sum_{s_j \in S} p(s_j \mid s_i, a) \|V - V'\|$$

$$= \gamma \|V - V'\|$$

Most of the story in a nutshell:

Properties of Contraction

- Only has 1 fixed point
  - If had two, then would not get closer when apply contraction function, violating definition of contraction
- When apply contraction function to any argument, value must get closer to fixed point
  - Fixed point doesn't move
  - Repeated function applications yield fixed point

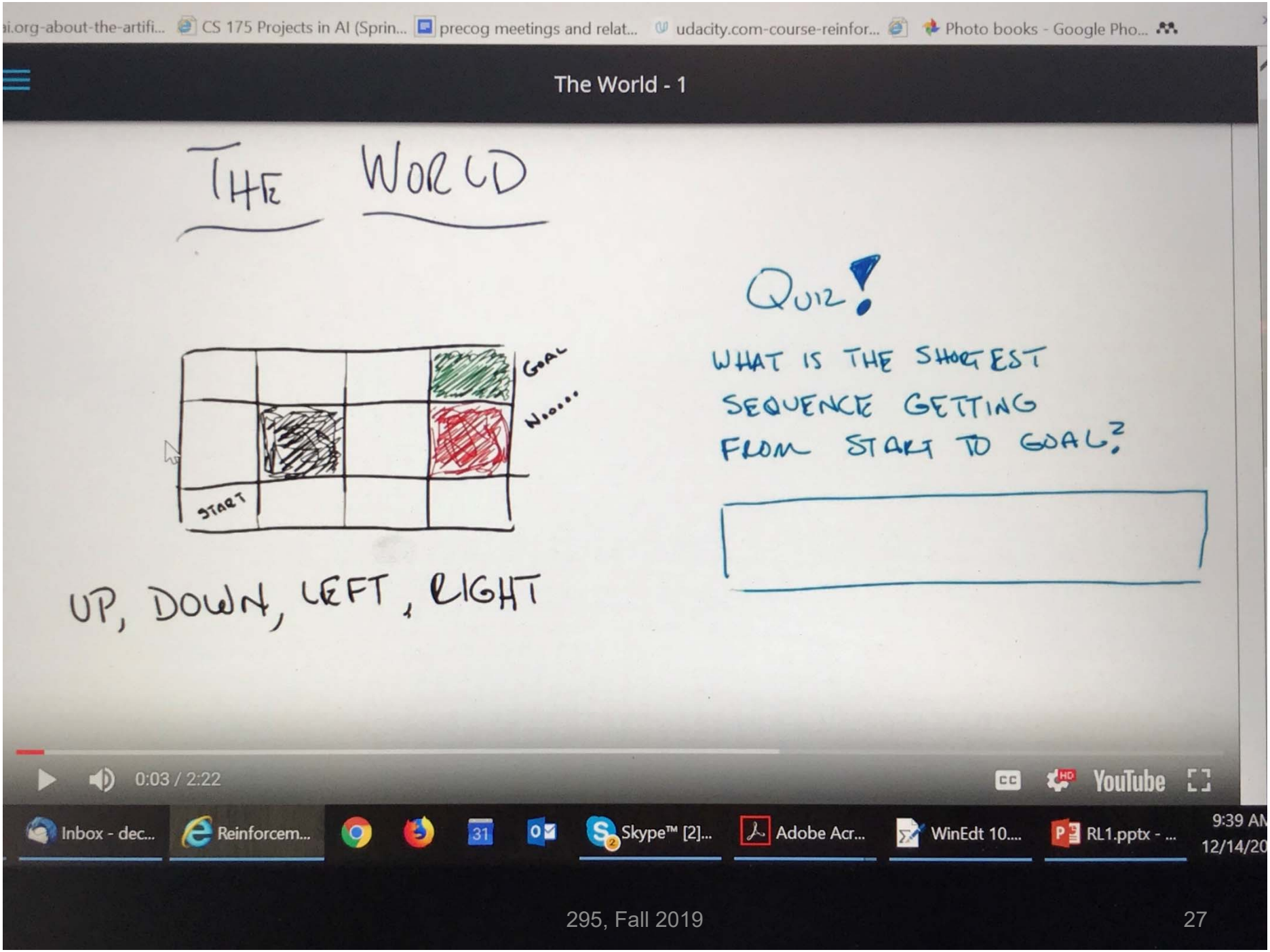
Most of the story in a nutshell:

Value Iteration Converges

- If discount factor < 1
- Bellman is a contraction
- Value iteration converges to unique solution which is optimal value function

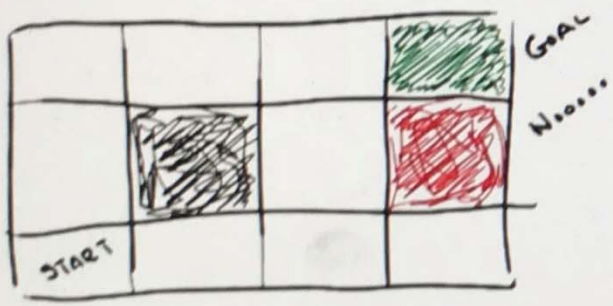
RL: Designer Choices

- Representation (how represent the world and the space of actions/interventions, and feedback signal/ reward)
- Algorithm for learning
- Objective function
- Evaluation


The World - 1
THE WORLD
GOAL
START
UP, DOWN, LEFT, RIGHT
QUIZ!
WHAT IS THE SHORTEST SEQUENCE GETTING FROM START TO GOAL?
0:03 / 2:22
YouTube

The World - 2

THE WORLD



UP, DOWN, LEFT, RIGHT

- ACTIONS EXECUTES .8
- MOVE AT RIGHT ANGLE .1 + .1

QUIZ!

WHAT IS THE RELIABILITY OF OUR SEQUENCE

UP UP RIGHT RIGHT RIGHT ?

History and State

- The history is the sequence of observations, actions, rewards

$$H_t = O_1, R_1, A_1, \ldots, A_{t-1}, O_t, R_t$$

- i.e. all observable variables up to time t
- i.e. the sensorimotor stream of a robot or embodied agent
- What happens next depends on the history:
  - The agent selects actions
  - The environment selects observations/rewards
- State is the information used to determine what happens next
- Formally, state is a function of the history:

$$S_t = f(H_t)$$

Information State

An information state (a.k.a. Markov state) contains all useful information from the history.

Definition

A state S_t is Markov if and only if

$$P[S_{t+1} \mid S_t] = P[S_{t+1} \mid S_1, \ldots, S_t]$$

- "The future is independent of the past given the present"

$$H_{1:t} \rightarrow S_t \rightarrow H_{t+1:\infty}$$

- Once the state is known, the history may be thrown away
- i.e. The state is a sufficient statistic of the future
- The environment state S_t is Markov
- The history H_t is Markov

Major Components of an RL Agent

- An RL agent may include one or more of these components:
  - Policy: agent's behaviour function
  - Value function: how good is each state and/or action
  - Model: agent's representation of the environment

Policy

- A policy is the agent's behaviour
- It is a map from state to action, e.g.
- Deterministic policy: $a = \pi(s)$
- Stochastic policy: $\pi(a|s) = P[A_t = a|S_t = s]$

Value Function

- Value function is a prediction of future reward
- Used to evaluate the goodness/badness of states
- And therefore to select between actions, e.g.

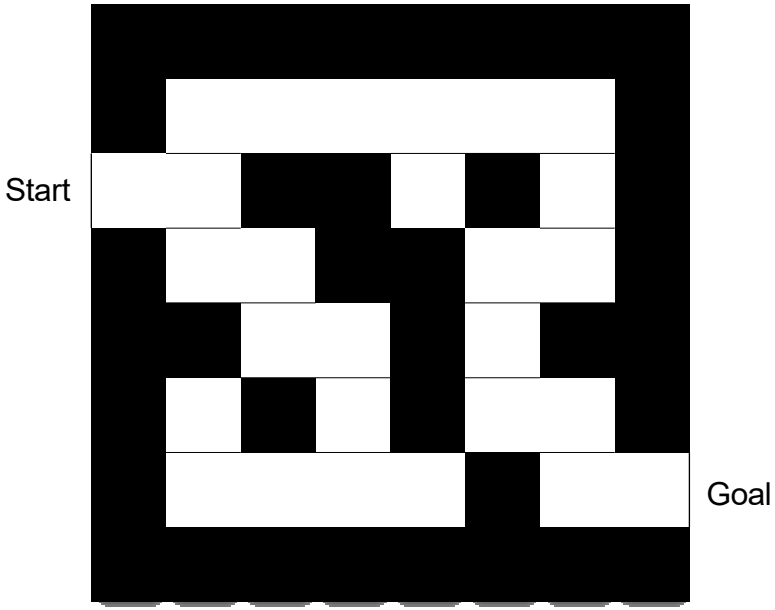
$$v_\pi(s) = E_\pi R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s$$

Model

- A model predicts what the environment will do next
- $\mathcal{P}$ predicts the next state
- $\mathcal{R}$ predicts the next (immediate) reward, e.g.

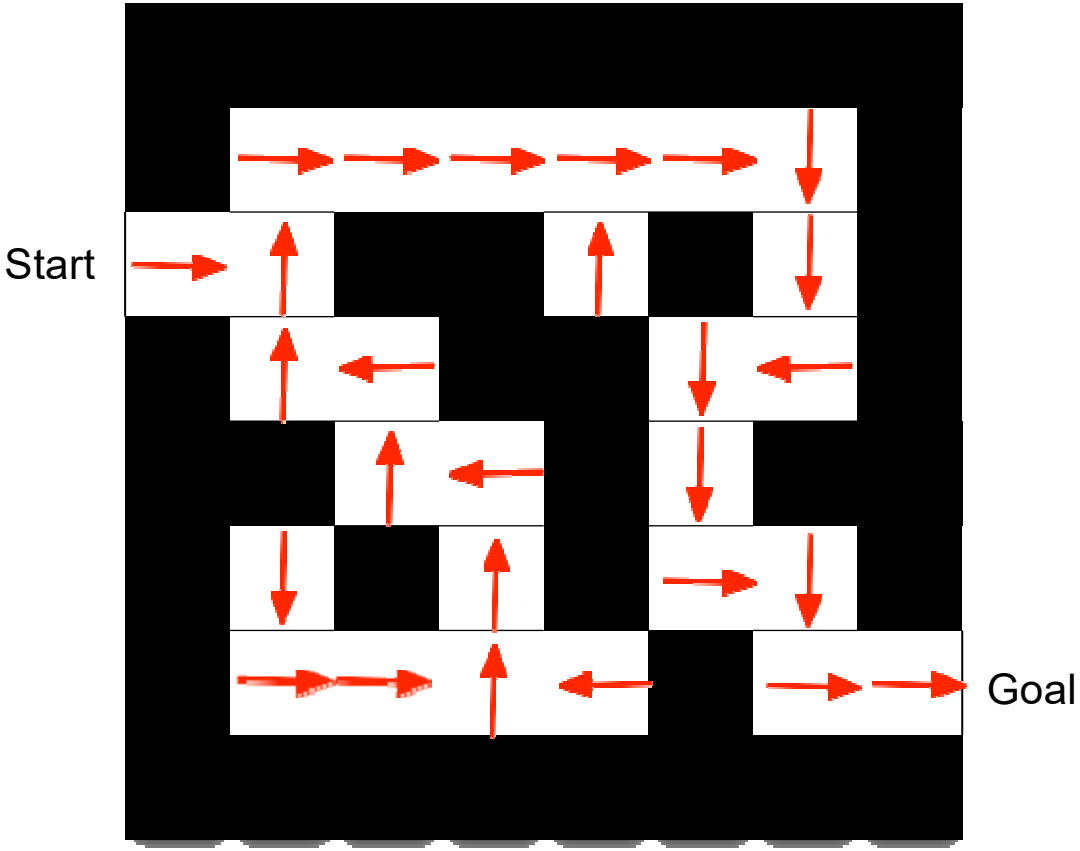
$$\mathcal{P}_{ss'}^{a} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$
$$\mathcal{R}_{s}^{a} = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$$

Maze Example



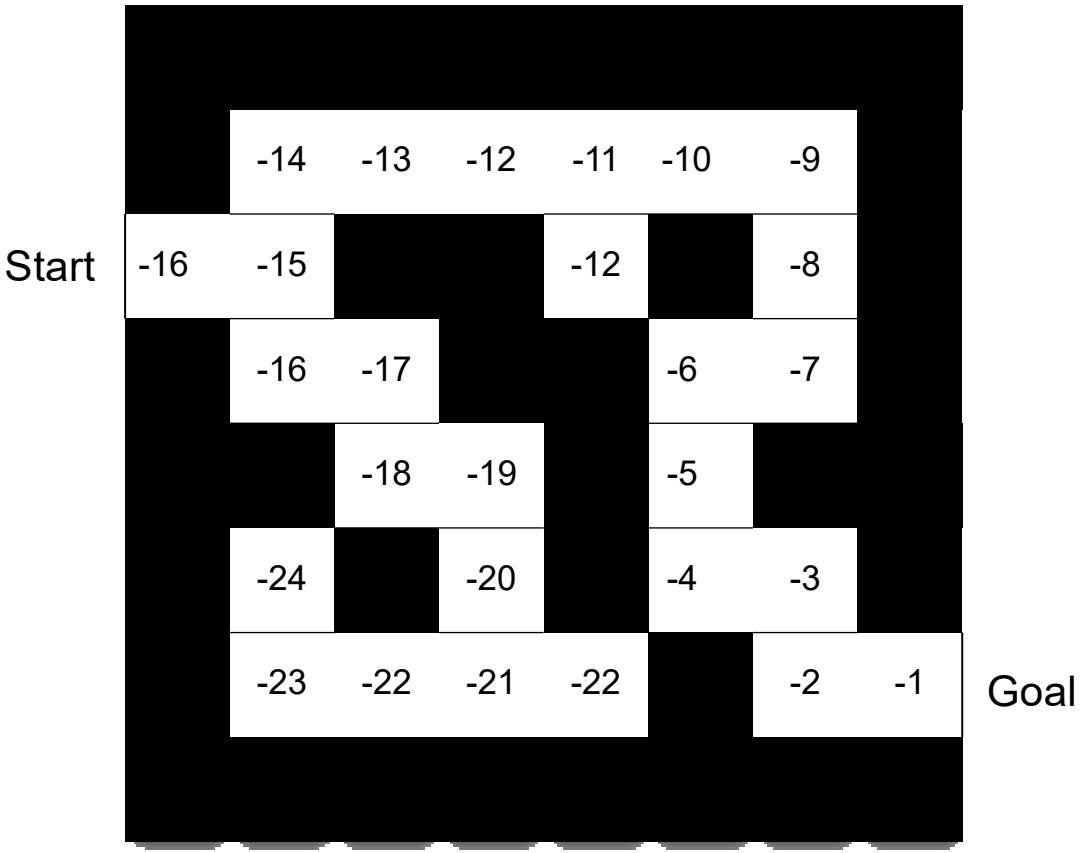
- Rewards: -1 per time-step
- Actions: N, E, S, W
- States: Agent's location

Maze Example: Policy



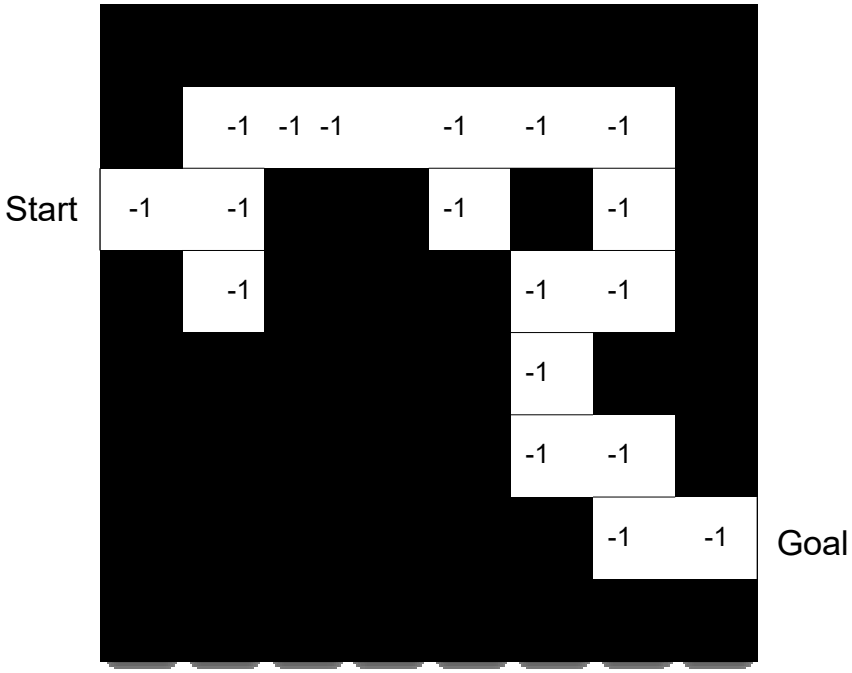
- Arrows represent policy $\pi(s)$ for each state s

Maze Example: Value Function



- Numbers represent value $v_\pi(s)$ of each state s

Maze Example: Model



- Agent may have an internal model of the environment
- Dynamics: how actions change the state
- Rewards: how much reward from each state
- The model may be imperfect

- Grid layout represents transition model $P^a_{ss'}$
- Numbers represent immediate reward R^a_s from each state s (same for all a)

Learning and Planning

Two fundamental problems in sequential decision making

- Reinforcement Learning:
  - The environment is initially unknown
  - The agent interacts with the environment
  - The agent improves its policy
- Planning:
  - A model of the environment is known
  - The agent performs computations with its model (without any external interaction)
  - The agent improves its policy
  - a.k.a. deliberation, reasoning, introspection, pondering, thought, search

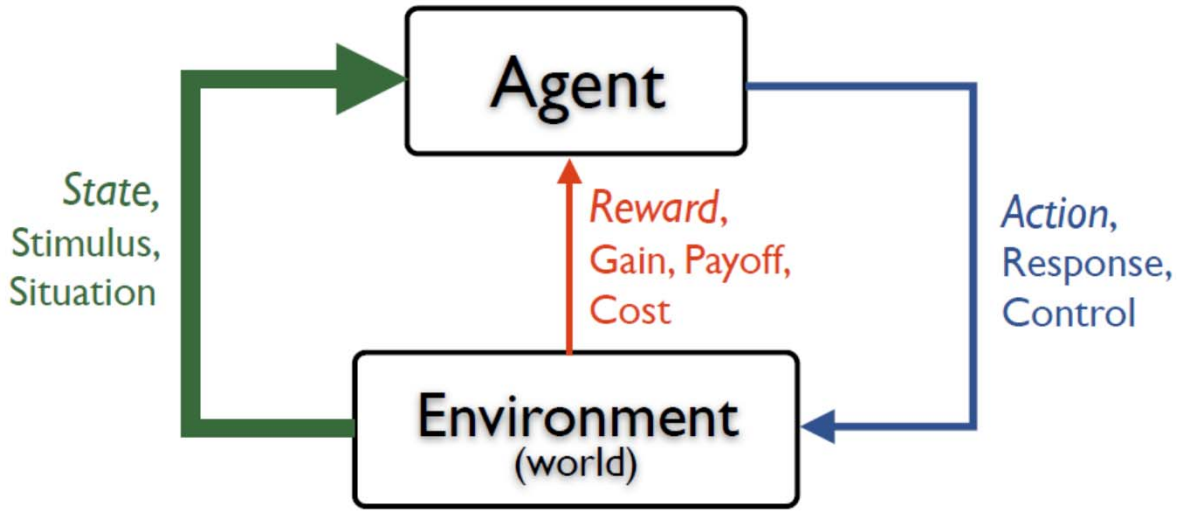
Prediction and Control

- Prediction: evaluate the future
  - Given a policy
- Control: optimise the future
  - Find the best policy

Markov Decision Processes

Chapter 3 S&B

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

MDPs

- The world is an MDP (combining the agent and the world): give rise to a trajectory

  S0,A0,R1,S1,A1,R2,S2,A3,R3,S3,…
- The process is governed by a transition function

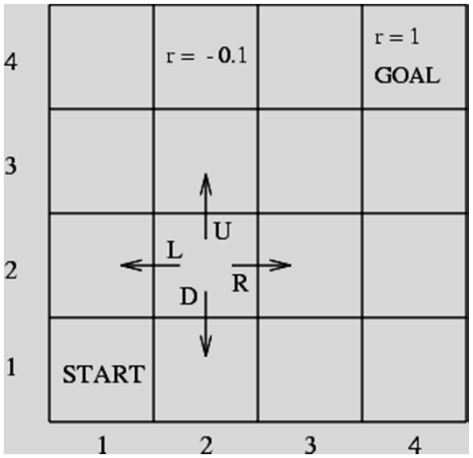
$$p(s', r \mid s, a) \doteq \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\},$$

- Markov Process (MP)
- Markov Reward Process (MRP)
- Markov Decision Process (MDP)

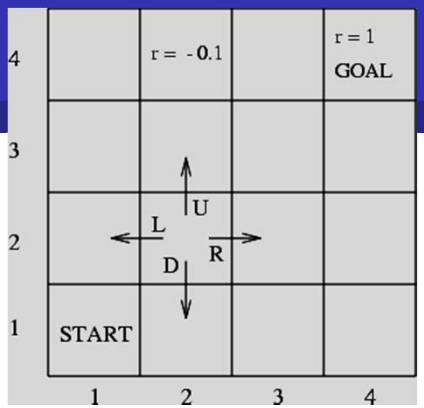
Grid world examples

Example: robot navigation

- State = {X, Y, Battery_Level}
- Actions = {Go_North, Go_South, Go_West, Go_East}
- Probability of success = P
- **Task:** reach the goal location ASAP



Markov Property



"The future is independent of the past given the present"

Definition

A state S_t is *Markov* if and only if

$$P[S_{t+1} \mid S_t] = P[S_{t+1} \mid S_1, \ldots, S_t]$$

- The state captures all relevant information from the history
- Once the state is known, the history may be thrown away
- i.e. The state is a sufficient statistic of the future

State Transition Matrix

For a Markov state s and successor state s', the *state transition probability* is defined by

$$\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$$

State transition matrix $\mathcal{P}$ defines transition probabilities from all states s to all successor states s',

$$\mathcal{P} = \begin{array}{cc} & \textit{to} \\ \textit{from} & \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix} \end{array}$$

where each row of the matrix sums to 1.

Markov Process

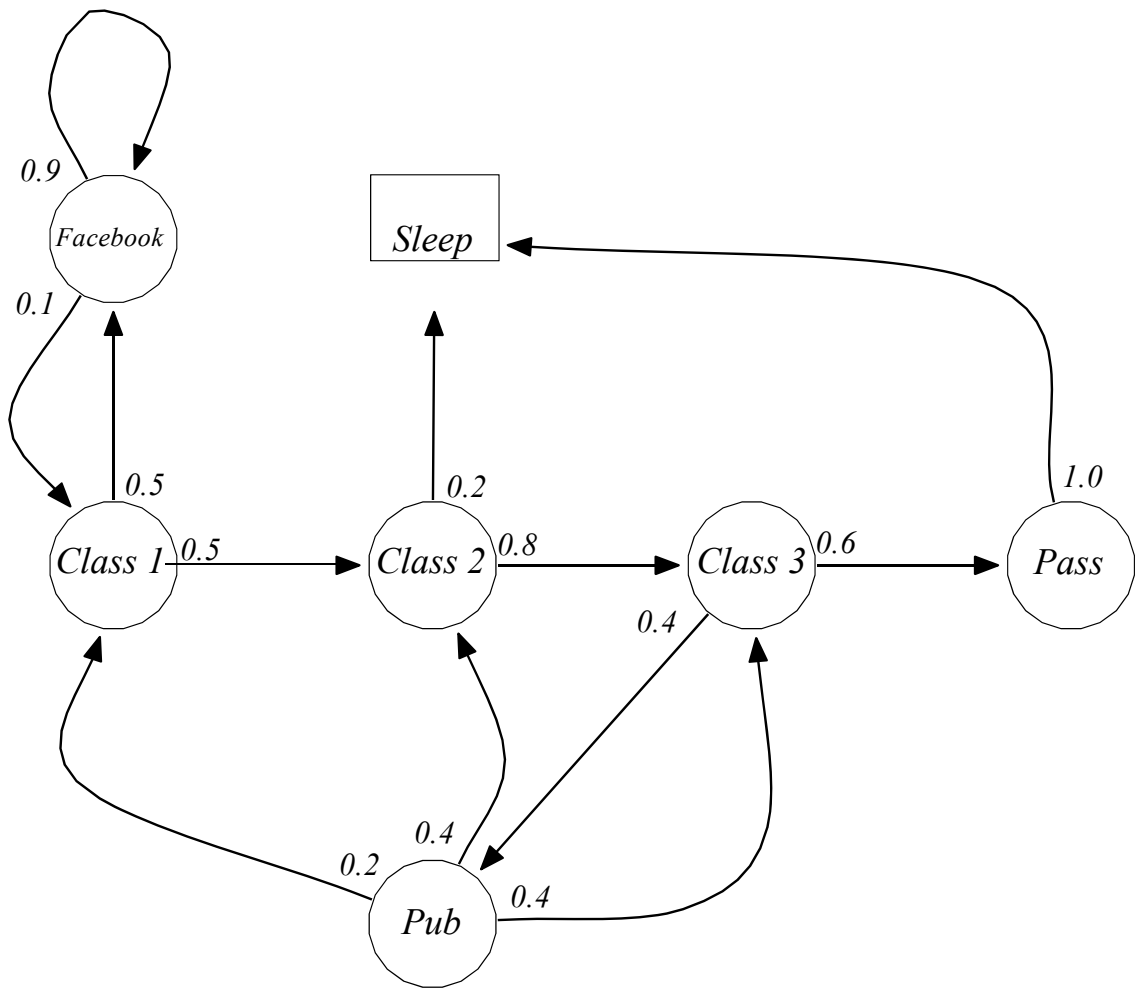
A Markov process is a memoryless random process, i.e. a sequence of random states S_1, S_2, ... with the Markov property.

Definition

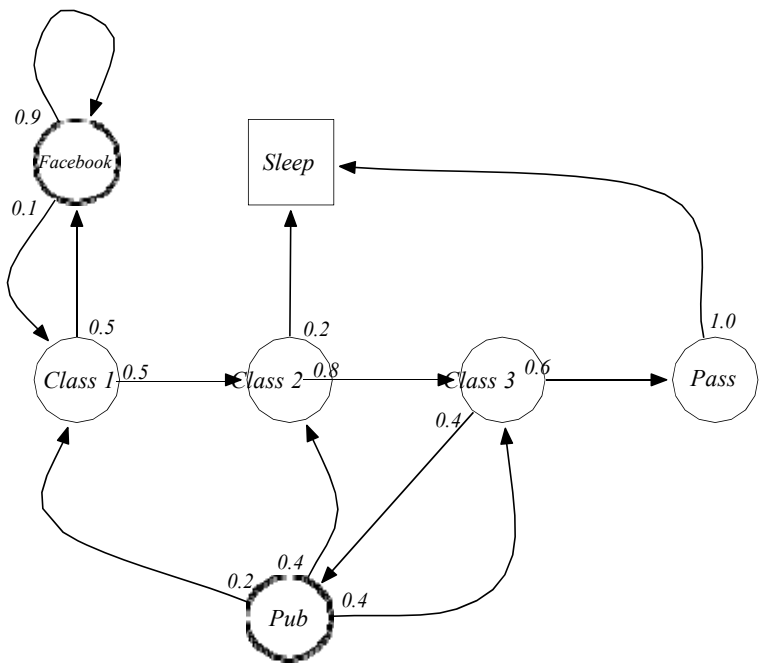
A *Markov Process* (or *Markov Chain*) is a tuple (S, P)

- S is a (finite) set of states
- P is a state transition probability matrix,
  $P_{ss'} = P[S_{t+1} = s' \mid S_t = s]$

Example: Student Markov Chain, a transition graph



Example: Student Markov Chain Episodes

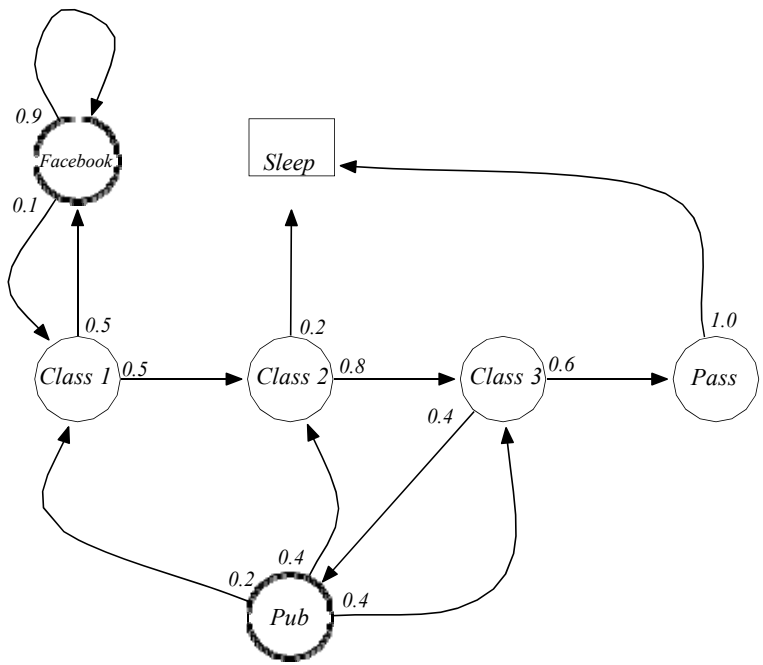


Sample episodes for Student Markov Chain starting from S_1 = C1

$$S_1, S_2, \ldots, S_T$$

- C1 C2 C3 Pass Sleep
- C1 FB FB C1 C2 Sleep
- C1 C2 C3 Pub C2 C3 Pass Sleep
- C1 FB FB C1 C2 C3 Pub C1 FB FB FB C1 C2 C3 Pub C2 Sleep

Example: Student Markov Chain Transition Matrix

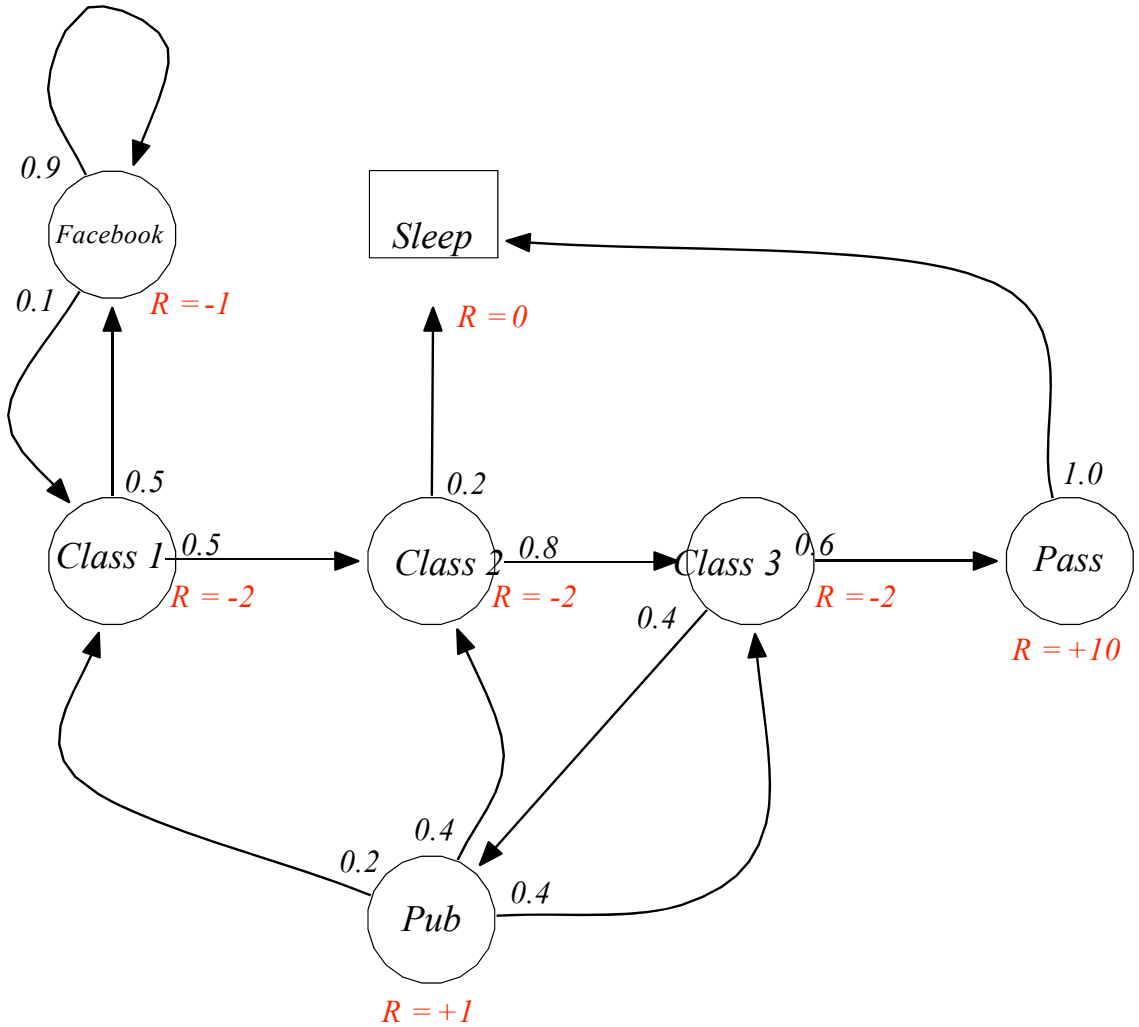


$$\mathcal{P} = \begin{array}{c} \\ C1 \\ C2 \\ C3 \\ Pass \\ Pub \\ FB \\ Sleep \end{array} \begin{array}{c} \begin{array}{ccccccc} C1 & C2 & C3 & Pass & Pub & FB & Sleep \end{array} \\ \begin{bmatrix} & 0.5 & & & & 0.5 & \\ & & 0.8 & & & & 0.2 \\ & & & 0.6 & 0.4 & & \\ & & & & & & 1.0 \\ 0.2 & 0.4 & 0.4 & & & & \\ 0.1 & & & & & 0.9 & \\ & & & & & & 1 \end{bmatrix} \end{array}$$

Markov Decision Processes

- States: S
- Model: T(s,a,s’) = P(s’|s,a)
- Actions: A(s), A
- Reward: R(s), R(s,a), R(s,a,s’)
- Discount: γ
- Policy: $\pi(s) \rightarrow a$
- Utility/Value: sum of discounted rewards.
- We seek optimal policy that maximizes the expected total (discounted) reward

Example: Student MRP



Goals, Returns and Rewards

- The agent's goal is to maximize the total amount of rewards it gets (not immediate ones), relative to the long run.
- Reward is -1 typically in mazes for every time step
- Deciding how to associate rewards with states is part of the problem modelling. If T is the final step then the return is:

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T,$$

Return

Definition

The *return* G_t is the total discounted reward from time-step t.

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

- The *discount* $\gamma \in [0, 1]$ is the present value of future rewards
- The value of receiving reward R after $k + 1$ time-steps is $\gamma^k R$.
- This values immediate reward above delayed reward.
  - γ close to 0 leads to "myopic" evaluation
  - γ close to 1 leads to "far-sighted" evaluation

Why discount?

Most Markov reward and decision processes are discounted. Why?

- Mathematically convenient to discount rewards
- Avoids infinite returns in cyclic Markov processes
- Uncertainty about the future may not be fully represented
- If the reward is financial, immediate rewards may earn more interest than delayed rewards
- Animal/human behaviour shows preference for immediate reward
- It is sometimes possible to use *undiscounted* Markov reward processes (i.e. $\gamma = 1$), e.g. if all sequences terminate.

Value Function

The value function $v(s)$ gives the long-term value of state s

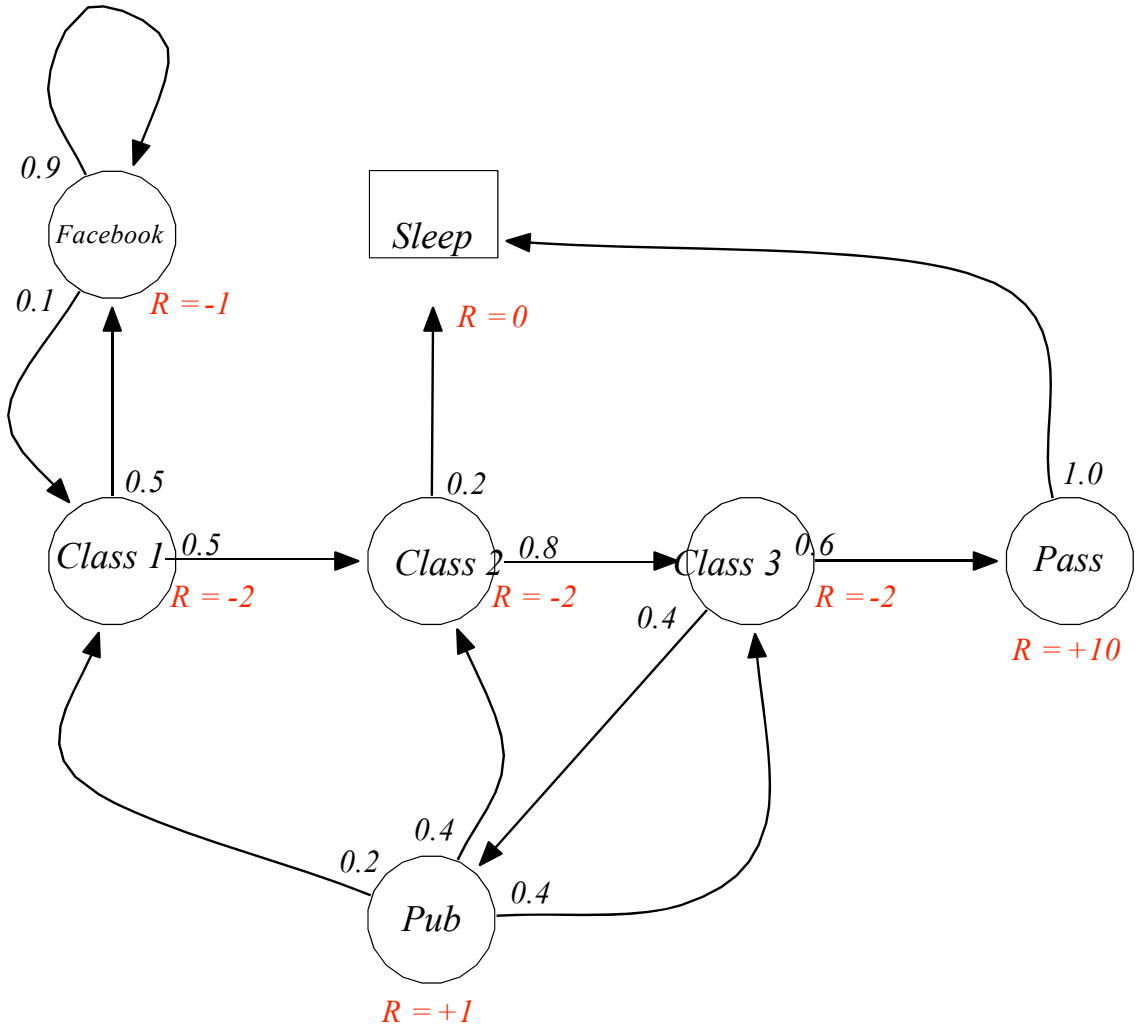
Definition

The *state value function* $v(s)$ of an MRP is the expected return starting from state s

$$v(s) = \mathrm{E}[G_t \mid S_t = s]$$

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

Example: Student MRP



Example: Student MRP Returns

Sample returns for Student MRP:
Starting from S_1 = C1 with $\gamma = \frac{1}{2}$

$$G_1 = R_2 + \gamma R_3 + \ldots + \gamma^{T-2} R_T$$

C1 C2 C3 Pass Sleep

C1 FB FB C1 C2 Sleep

C1 C2 C3 Pub C2 C3 Pass Sleep C1

FB FB C1 C2 C3 Pub C1 … FB

FB FB C1 C2 C3 Pub C2 Sleep

$$v_1 = -2 - 2 * \tfrac{1}{2} - 2 * \tfrac{1}{4} + 10 * \tfrac{1}{8} = -2.25$$

$$v_1 = -2 - 1 * \tfrac{1}{2} - 1 * \tfrac{1}{4} - 2 * \tfrac{1}{8} - 2 * \tfrac{1}{16} = -3.125$$

$$v_1 = -2 - 2 * \tfrac{1}{2} - 2 * \tfrac{1}{4} + 1 * \tfrac{1}{8} - 2 * \tfrac{1}{16} \ldots = -3.41$$

$$v_1 = -2 - 1 * \tfrac{1}{2} - 1 * \tfrac{1}{4} - 2 * \tfrac{1}{8} - 2 * \tfrac{1}{16} \ldots = -3.20$$

Bellman Equation for MRPs

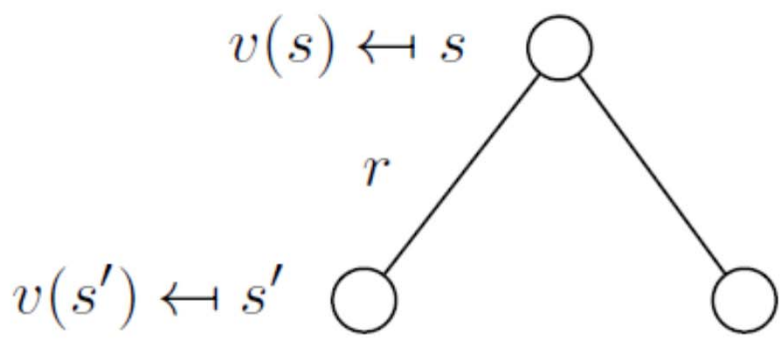
The value function can be decomposed into two parts:

- immediate reward R_{t+1}
- discounted value of successor state $\gamma v(S_{t+1})$

$$
\begin{aligned}
v(s) &= \mathrm{E}[G_t \mid S_t = s] \\
&= \mathrm{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s] \\
&= \mathrm{E}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \ldots) \mid S_t = s] \\
&= \mathrm{E}[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\
&= \mathrm{E}[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s]
\end{aligned}
$$

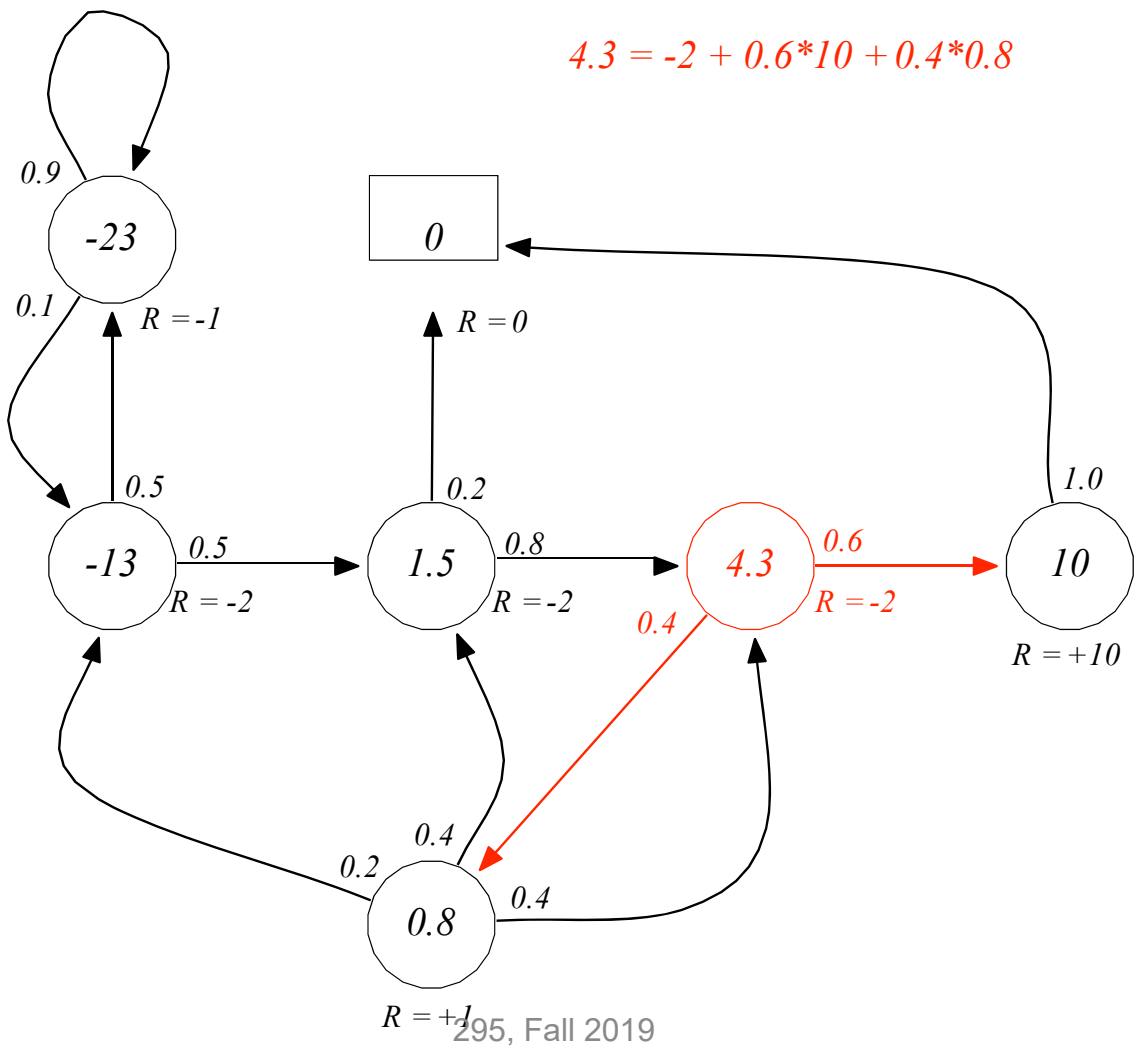
Bellman Equation for MRPs (2)

$$v(s) = \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s\right]$$



$$v(s) = \mathcal{R}_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'} v(s')$$

Example: Bellman Equation for Student MRP



Bellman Equation in Matrix Form

The Bellman equation can be expressed concisely using matrices,

$$v = \mathrm{R} + \gamma \mathrm{P} v$$

where v is a column vector with one entry per state

$$\begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} \mathcal{R}_1 \\ \vdots \\ \mathcal{R}_n \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{11} & \dots & \mathcal{P}_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix}$$

Solving the Bellman Equation

- The Bellman equation is a linear equation
- It can be solved directly:

$$
\begin{aligned}
v &= \mathrm{R} + \gamma \mathrm{P} v \\
(I - \gamma \mathrm{P})\, v &= \mathrm{R} \\
v &= (I - \gamma \mathrm{P})^{-1}\, \mathrm{R}
\end{aligned}
$$

- Computational complexity is $O(n^3)$ for n states
- Direct solution only possible for small MRPs
- There are many iterative methods for large MRPs, e.g.
  - Dynamic programming
  - Monte-Carlo evaluation
  - Temporal-Difference learning

Markov Decision Process

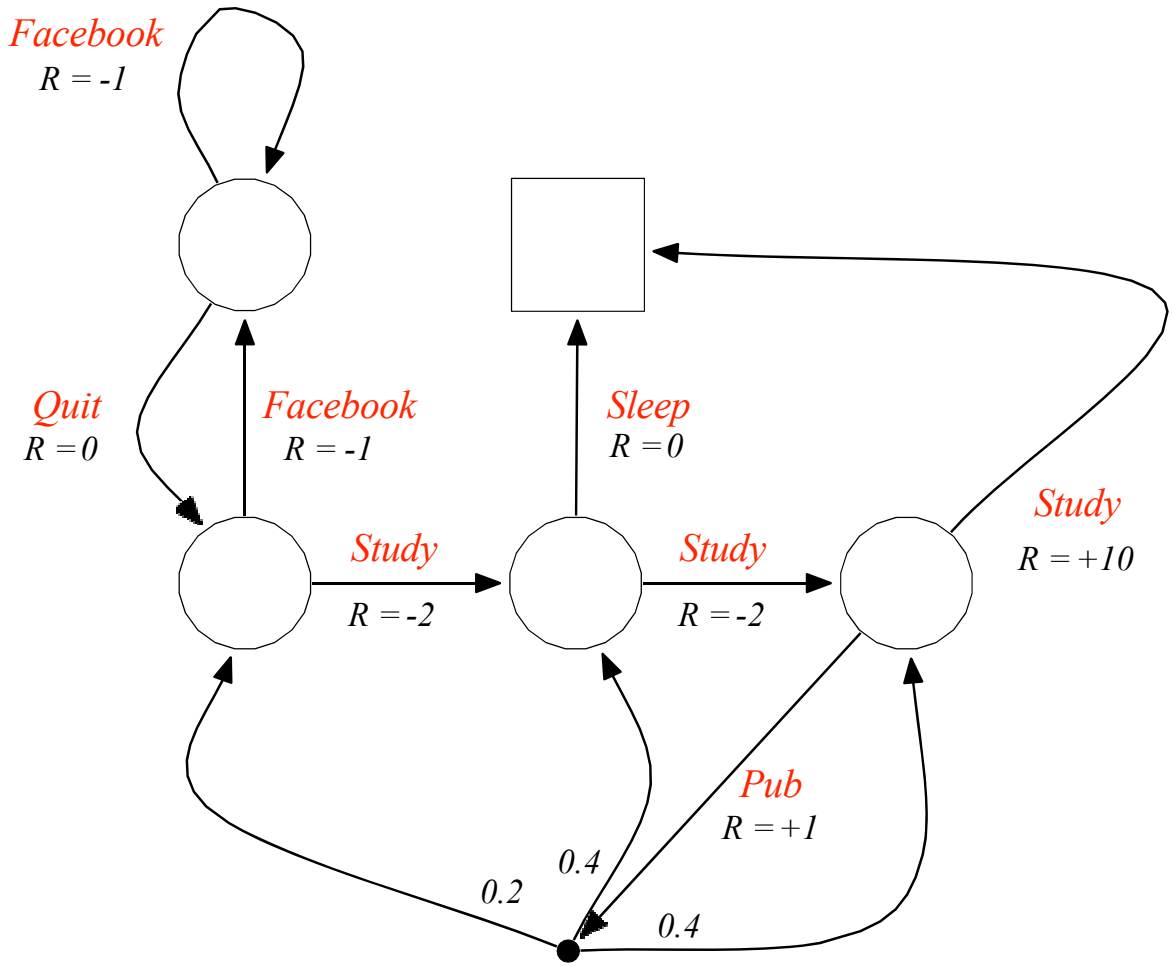
A Markov decision process (MDP) is a Markov reward process with decisions. It is an *environment* in which all states are Markov.

Definition

A *Markov Decision Process* is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability matrix, $\mathcal{P}_{ss'}^{a} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s, A_t = a\right]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}_s^a = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$
- γ is a discount factor $\gamma \in [0, 1]$.

Example: Student MDP



Policies and Value functions (1)

Definition

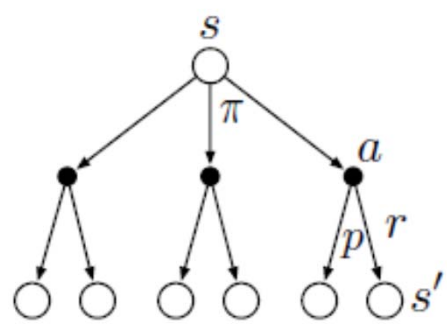
A *policy* π is a distribution over actions given states,

$$\pi(a|s) = P[A_t = a \mid S_t = s]$$

- A policy fully defines the behaviour of an agent
- MDP policies depend on the current state (not the history)
- i.e. Policies are *stationary* (time-independent),
  $A_t \sim \pi(\cdot|S_t), \ \forall t > 0$

Policy's and Value functions

$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] && \text{(by (3.9))} \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r \mid s, a) \Big[r + \gamma \mathbb{E}_\pi[G_{t+1} | S_{t+1} = s'] \Big] \\
&= \sum_a \pi(a|s) \sum_{s', r} p(s', r \mid s, a) \Big[r + \gamma v_\pi(s') \Big], \quad \text{for all } s \in \mathcal{S}, && (3.14)
\end{aligned}
$$



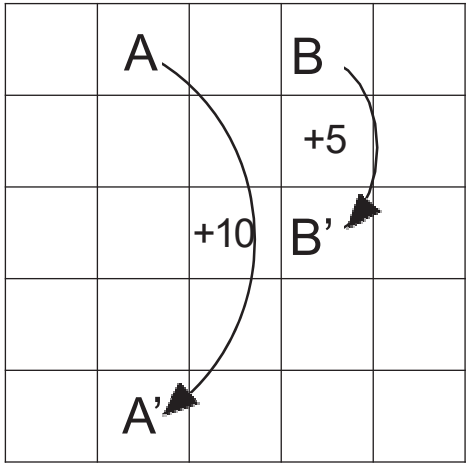
Backup diagram for v_π

Gridworld Example: Prediction

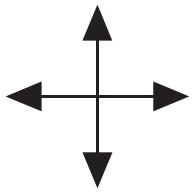
Actions: up, down, left, right. Rewards 0 unless off the grid with reward -1
From A to A', rewatd +10. from B to B' reward +5

Policy: actions are uniformly random.

Figure 3.3



(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)

What is the value function for the uniform random policy?
Gamma=0.9. solved using EQ. 3.14

Exercise: show 3.14 holds for each state in Figure (b).

Value Function, Q Functions

Definition

The *state-value function* $v_\pi(s)$ of an MDP is the expected return starting from state s, and then following policy π

$$v_\pi(s) = E_\pi[G_t \mid S_t = s]$$

Definition

The *action-value function* $q_\pi(s, a)$ is the expected return starting from state s, taking action a, and then following policy π

$$q_\pi(s, a) = E_\pi[G_t \mid S_t = s, A_t = a]$$

Bellman Expectation Equation

The state-value function can again be decomposed into immediate reward plus discounted value of successor state,

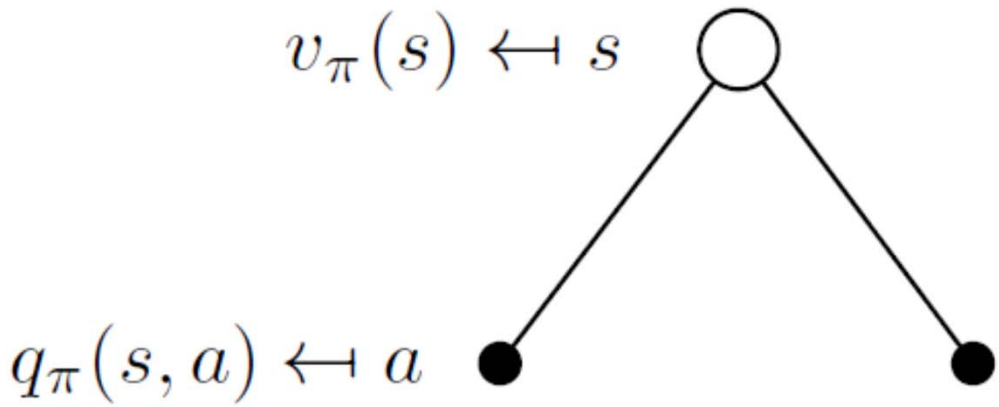
$$v_\pi(s) = \mathrm{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]$$

The action-value function can similarly be decomposed,

$$q_\pi(s, a) = \mathrm{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$$

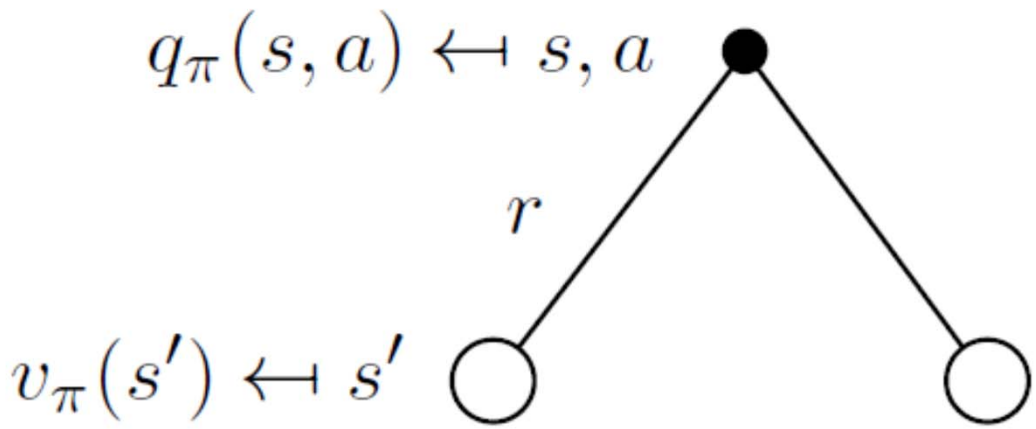
Expressing the functions recursively,
Will translate to one step look-ahead.

Bellman Expectation Equation for V^{π}



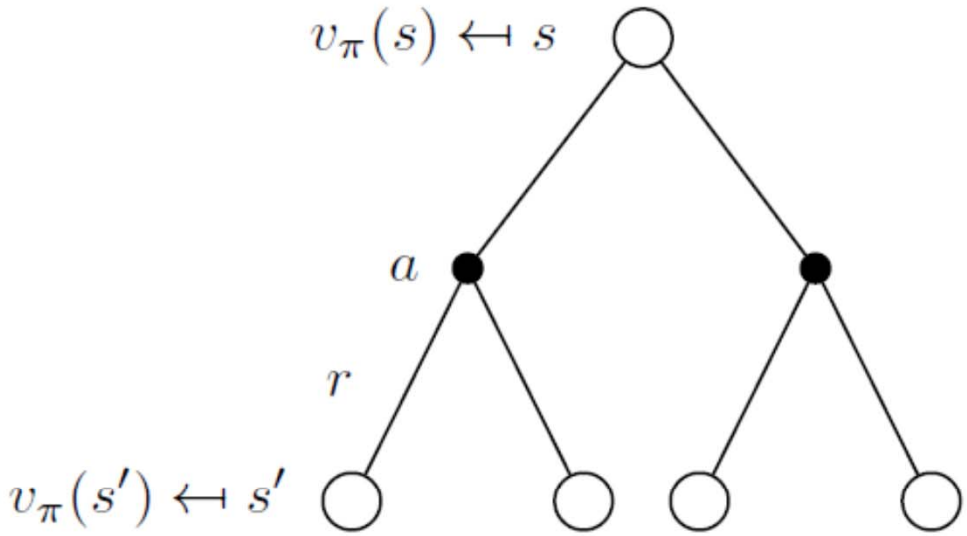
$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$$

Bellman Expectation Equation for Q^π



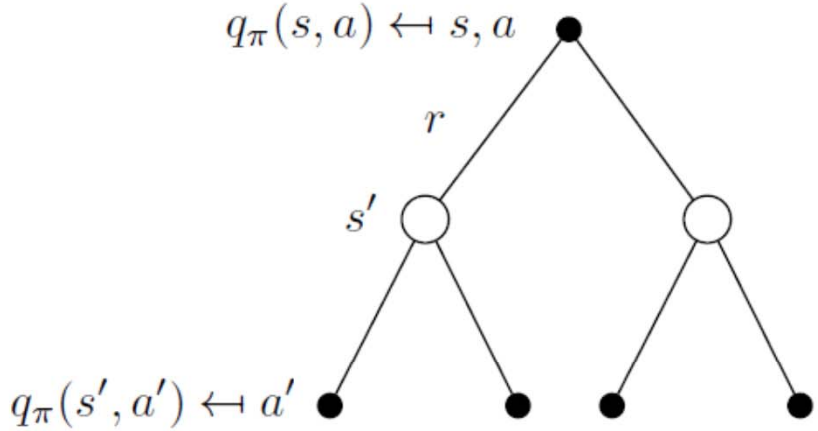
$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$

Bellman Expectation Equation for v_π (2)



$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right)$$

Bellman Expectation Equation for q_π (2)



$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s',a')$$

Optimal Policies and Optimal Value Function

Definition

The *optimal state-value function* $v_*(s)$ is the maximum value function over all policies

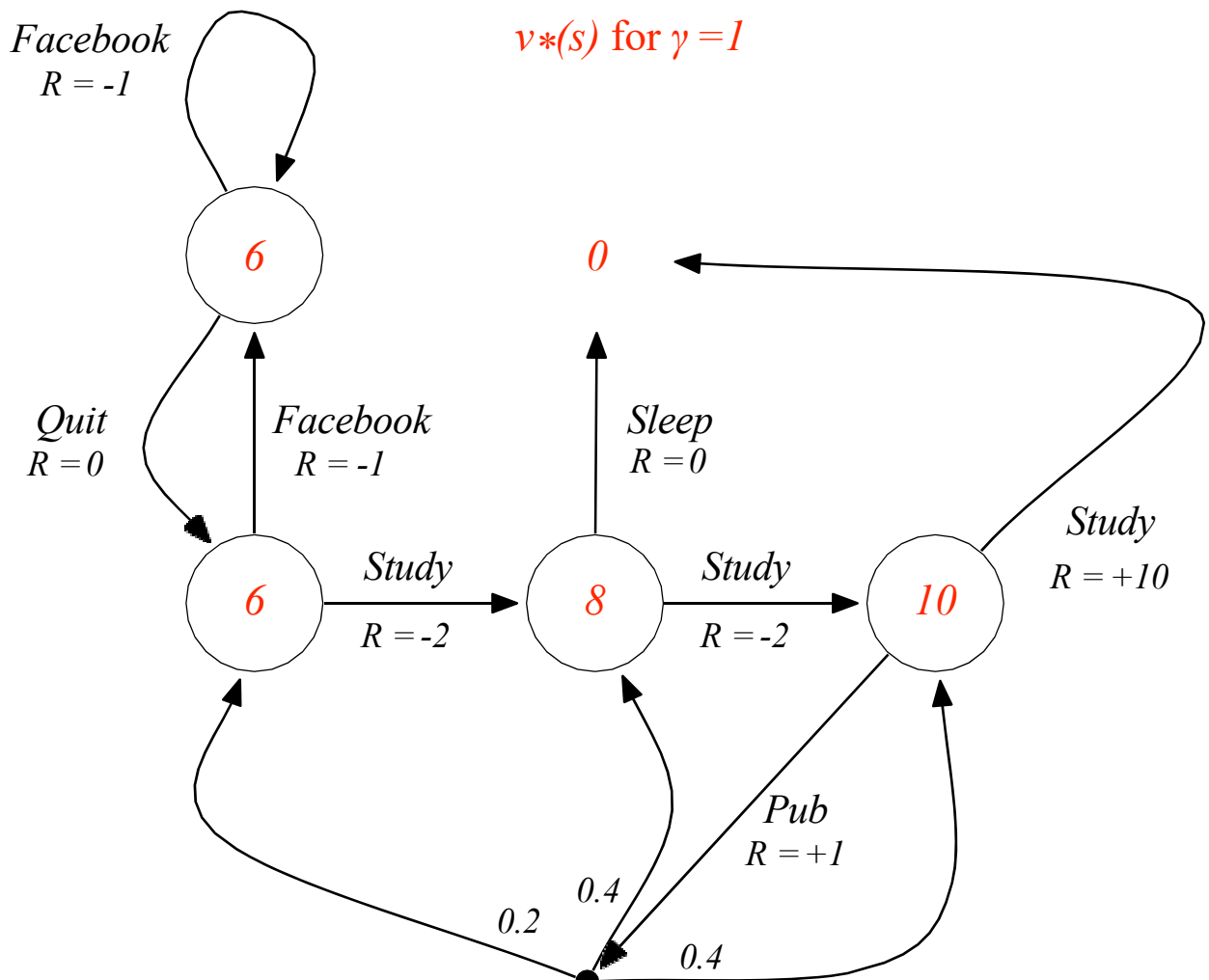
$$v_*(s) = \max_{\pi} v_\pi(s)$$

The *optimal action-value function* $q_*(s, a)$ is the maximum action-value function over all policies

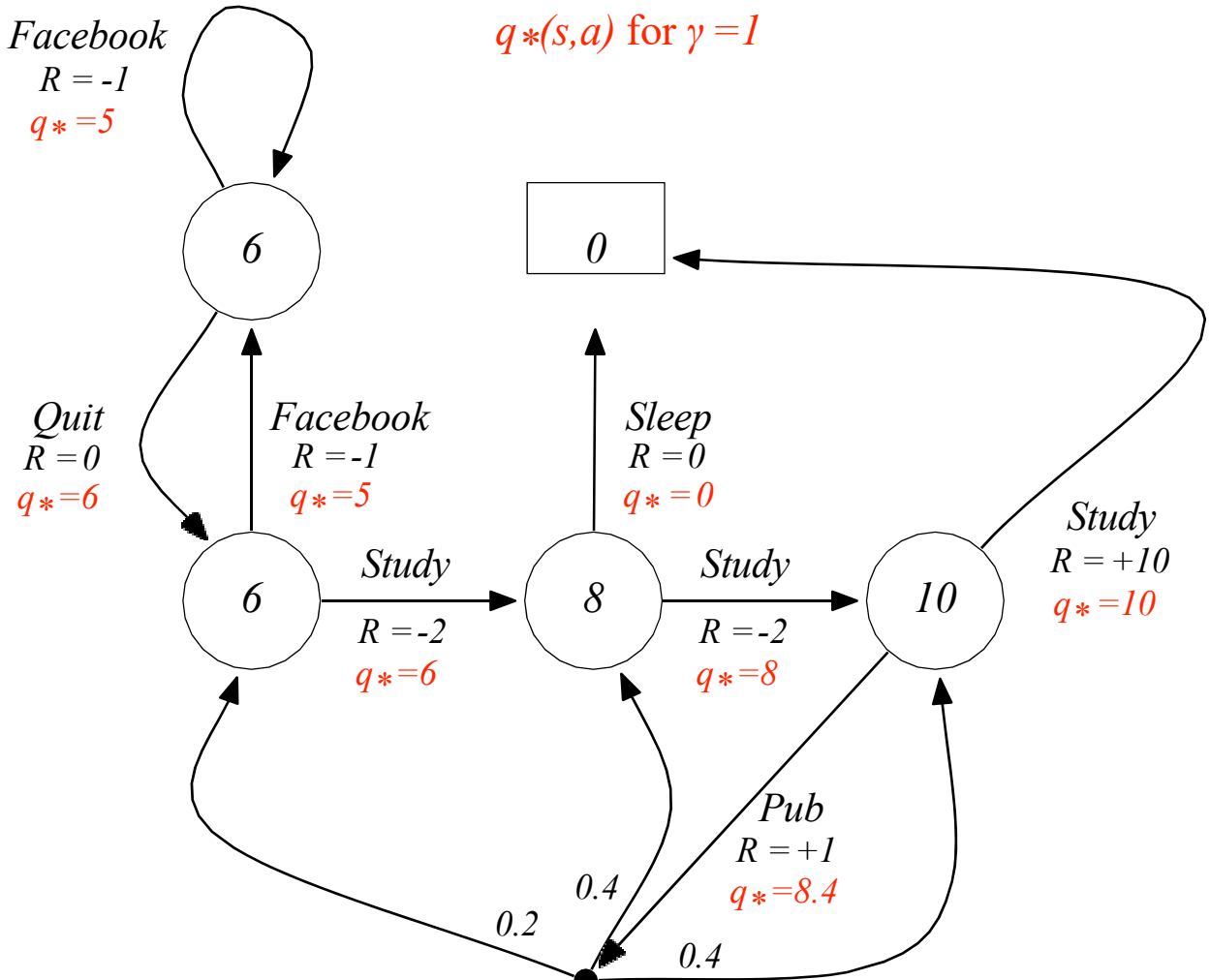
$$q_*(s, a) = \max_{\pi} q_\pi(s, a)$$

- The optimal value function specifies the best possible performance in the MDP.
- An MDP is "solved" when we know the optimal value function.

Optimal Value Function for Student MDP



Optimal Action-Value Function for Student MDP



Optimal Policy

Define a partial ordering over policies

$$\pi \geq \pi' \text{ if } v_\pi(s) \geq v_{\pi'}(s), \forall s$$

Theorem

For any Markov Decision Process

- *There exists an optimal policy π_* that is better than or equal to all other policies, $\pi_* \geq \pi, \forall \pi$*
- *All optimal policies achieve the optimal value function, $v_{\pi_*}(s) = v_*(s)$*
- *All optimal policies achieve the optimal action-value function, $q_{\pi_*}(s, a) = q_*(s, a)$*

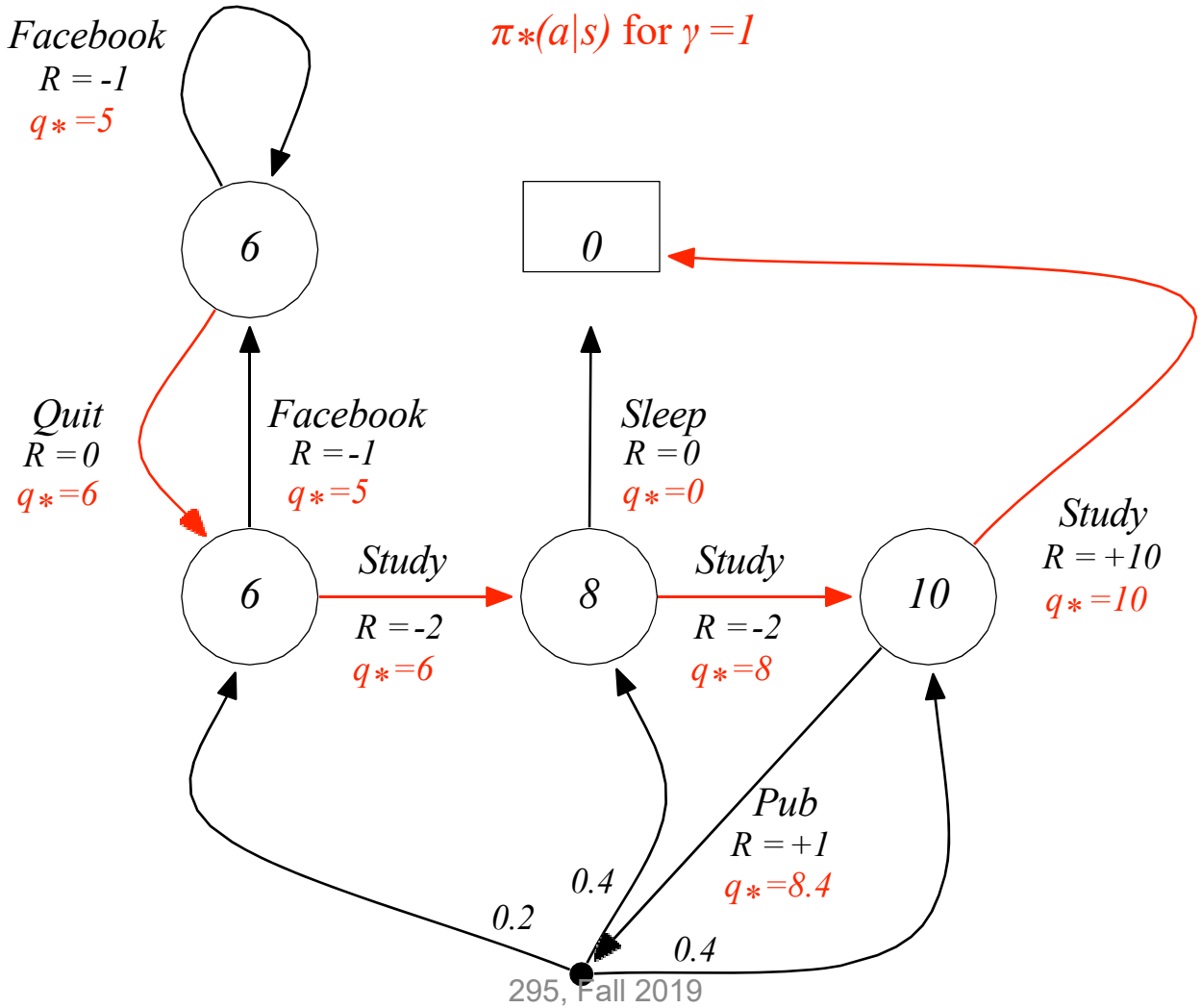
Finding an Optimal Policy

An optimal policy can be found by maximising over $q_*(s, a)$,

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, q_*(s, a) \\ 0 & otherwise \end{cases}$$

- There is always a deterministic optimal policy for any MDP
- If we know $q_*(s, a)$, we immediately have the optimal policy

Example: Optimal Policy for Student MDP



Bellman Equation for V* and Q*

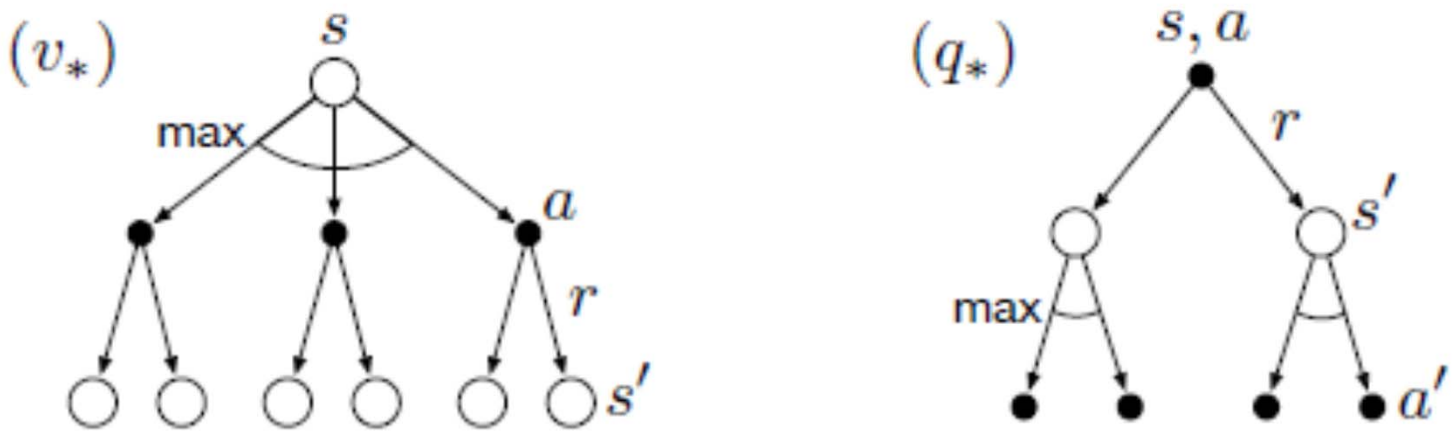
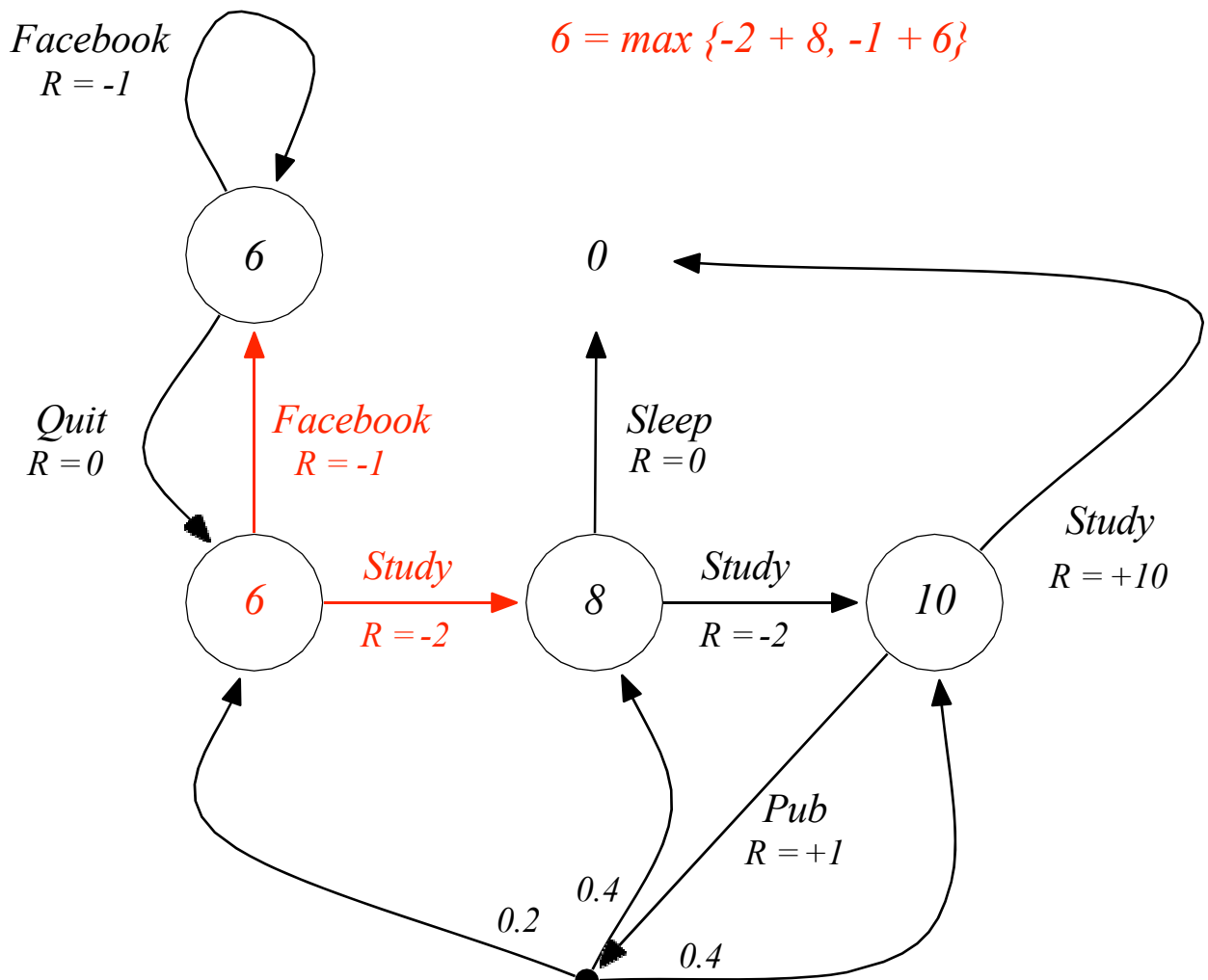


Figure 3.5: Backup diagrams for v_* and q_*

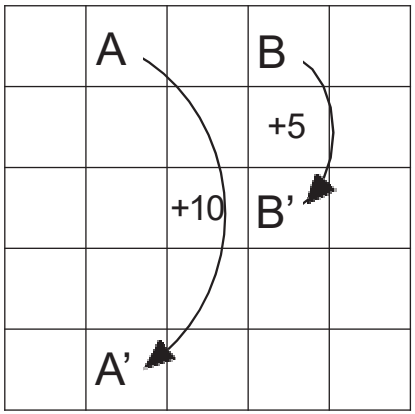
V*(s) $= \max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma v_*(s')\big].$

q*(s; a)= $\sum_{s',r} p(s',r|s,a)\Big[r + \gamma \max_{a'} q_*(s',a')\Big].$

Example: Bellman Optimality Equation in Student MDP



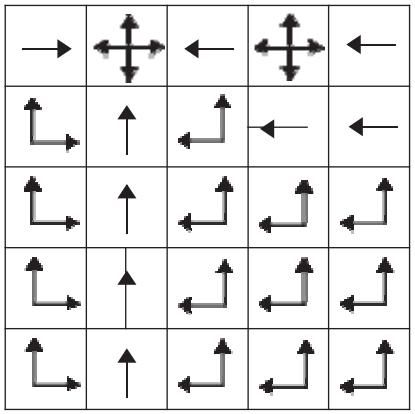
Gridworld Example: Control



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 is the optimal value function over all possible policies?
What is the optimal policy?

Figure 3.6

Solving the Bellman Optimality Equation

- Bellman Optimality Equation is non-linear
- No closed form solution (in general) Many
- iterative solution methods
  - Value Iteration
  - Policy Iteration
  - Q-learning
  - Sarsa

Planning by Dynamic Programming

Sutton & Barto,
Chapter 4

Outline

Planning by Dynamic Programming

- Dynamic programming assumes full knowledge of the MDP
- It is used for *planning* in an MDP
- For prediction:
  - Input: MDP $(\mathrm{S}, \mathrm{A}, \mathrm{P}, \mathrm{R}, \gamma)$ and policy π
  - or: MRP $(\mathrm{S}, \mathrm{P}^{\pi}, \mathrm{R}^{\pi}, \gamma)$
  - Output: value function v_{π}
- Or for control:
  - Input: MDP $(\mathrm{S}, \mathrm{A}, \mathrm{P}, \mathrm{R}, \gamma)$
  - Output: optimal value function v_{*}
  - and: optimal policy π_{*}

Policy Evaluation (Prediction)

- Problem: evaluate a given policy π
- Solution: iterative application of Bellman expectation backup
- $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_\pi$
- Using *synchronous* backups,
  - At each iteration $k + 1$
  - For all states $s \in S$
  - Update $v_{k+1}(s)$ from $v_k(s')$
  - where s' is a successor state of s
- We will discuss *asynchronous* backups later
- Convergence to v_π will be proven at the end of the lecture

Iterative Policy Evaluations

$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big],
\end{aligned}
$$

These is a simultaneous linear equations in ISI unknowns and can be solved.

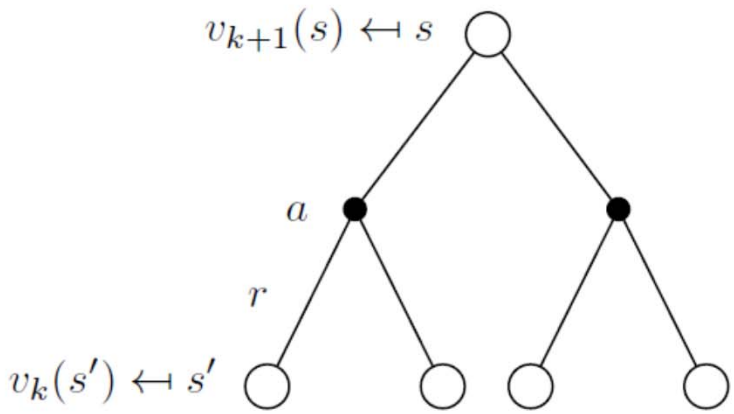
Practically an iterative procedure until a foxed-point can be more effective

$$
\begin{aligned}
v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_k(s')\Big],
\end{aligned}
$$

Iterative policy evaluation.

Iterative Policy Evaluation (2)

Iterative Policy Evaluation



$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

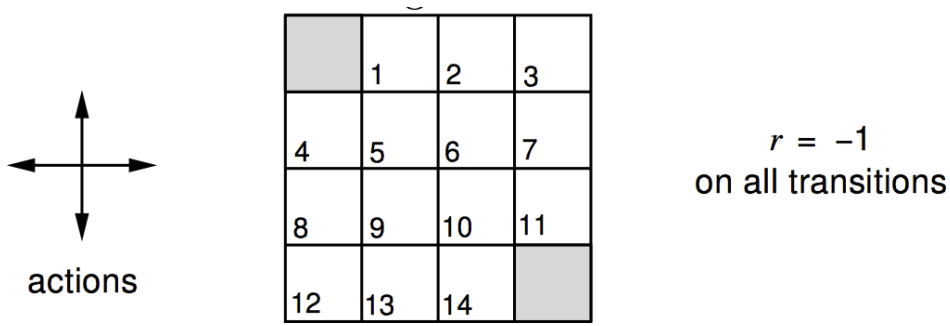
$$\mathbf{v}^{k+1} = \boldsymbol{\mathcal{R}}^{\pi} + \gamma \boldsymbol{\mathcal{P}}^{\pi} \mathbf{v}^k$$

Iterative policy Evaluation

Iterative policy evaluation

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 $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

Evaluating a Random Policy in the Small Gridworld



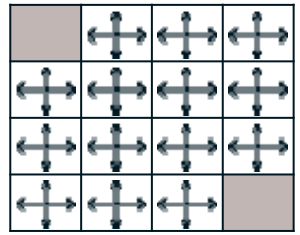
- Undiscounted episodic MDP ($\gamma = 1$)
- Nonterminal states 1, …, 14
- One terminal state (shown twice as shaded squares)
- Actions leading out of the grid leave state unchanged
- Reward is -1 until the terminal state is reached
- Agent follows uniform random policy

$$\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = 0.25$$

Iterative Policy Evaluation in Small Gridworld

	V_k for the Random Policy	Greedy Policy w.r.t. V_k
$k = 0$		

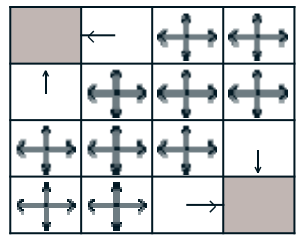
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



random policy

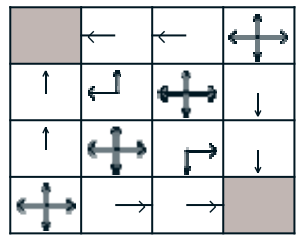
$k = 1$

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



$k = 2$

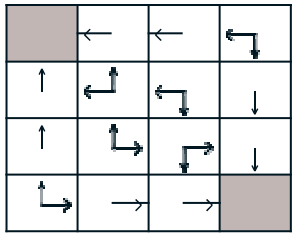
0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0



Iterative Policy Evaluation in Small Gridworld (2)

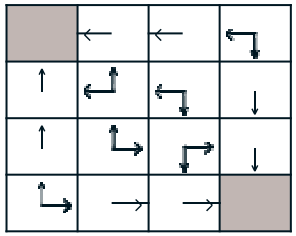
$k = 3$

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



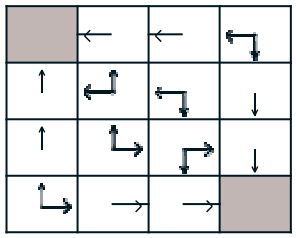
$k = 10$

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



optimal policy

Policy Improvement

- Given a policy π
  - Evaluate the policy π

$$v_\pi(s) = \mathrm{E}\,[R_{t+1} + \gamma R_{t+2} + \ldots | S_t = s]$$

  - Improve the policy by acting greedily with respect to v_π

$$\pi' = \text{greedy}(v_\pi)$$

- In Small Gridworld improved policy was optimal, $\pi' = \pi^*$
- In general, need more iterations of improvement / evaluation
- But this process of policy iteration always converges to $\pi*$

Policy Iteration

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*,$$

where $\xrightarrow{\text{E}}$ denotes a policy *evaluation* and $\xrightarrow{\text{I}}$ denotes a policy *improvement*. Each policy is guaranteed to be a strict improvement over the previous one (unless it is already optimal). Because a finite MDP has only a finite number of policies, this process must converge to an optimal policy and optimal value function in a finite number of iterations.

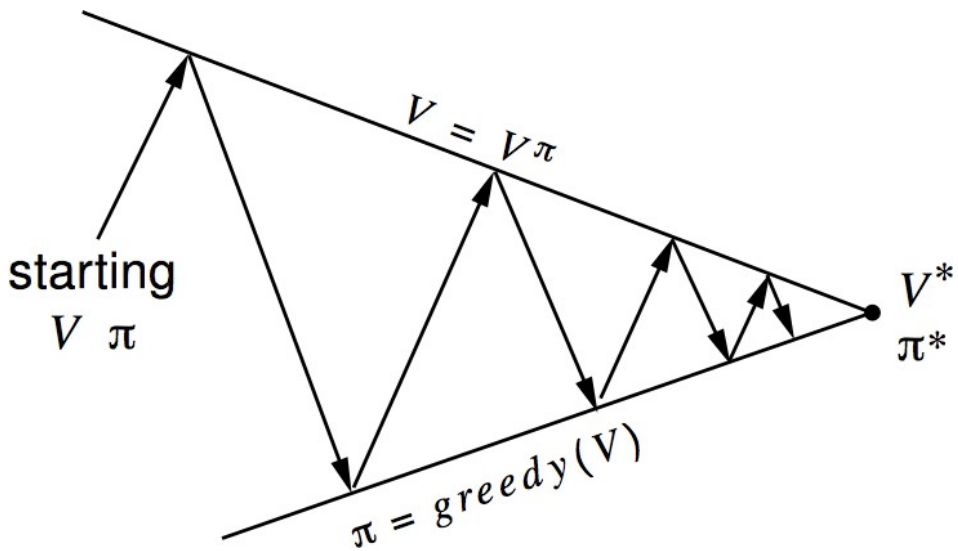
Policy iteration (using iterative policy evaluation)

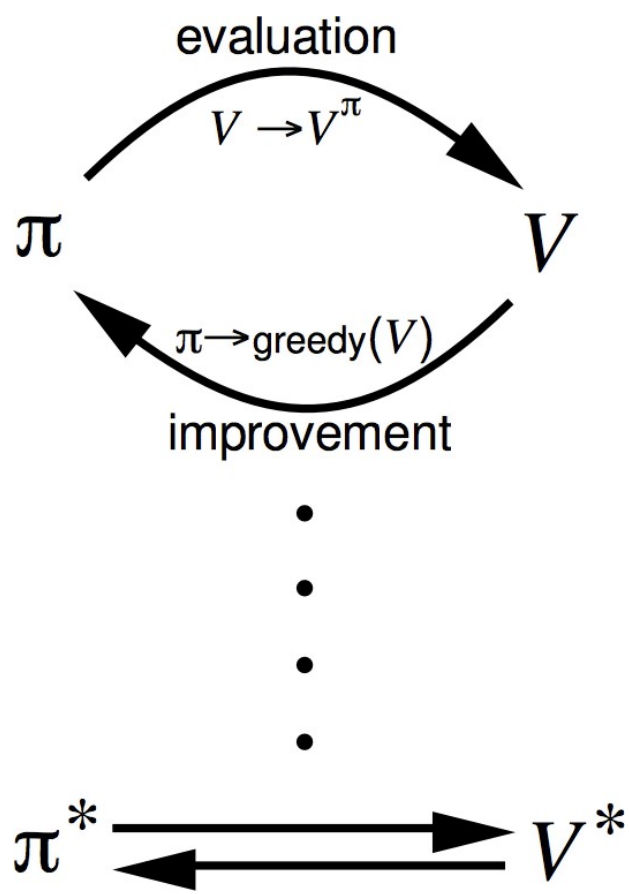
1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

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

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

Policy Iteration





Policy evaluation Estimate v_π

Iterative policy evaluation

Policy improvement Generate $\pi^{l} \geq \pi$

Greedy policy improvement

Policy Improvement

Policy Improvement

- Consider a deterministic policy, $a = \pi(s)$
- We can *improve* the policy by acting greedily

$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, q_\pi(s, a)$$

- This improves the value from any state s over one step,

$$q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) \geq q_\pi(s, \pi(s)) = v_\pi(s)$$

- It therefore improves the value function, $v_{\pi'}(s) \geq v_\pi(s)$

$$\begin{aligned} v_\pi(s) &\leq q_\pi(s, \pi'(s)) = \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s\right] \\ &\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s\right] \\ &\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi(S_{t+2}, \pi'(S_{t+2})) \mid S_t = s\right] \\ &\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \ldots \mid S_t = s\right] = v_{\pi'}(s) \end{aligned}$$

Policy Improvement (2)

- If improvements stop,

$$q_\pi(s, \pi'(s)) = \max_{a \in A} q_\pi(s, a) = q_\pi(s, \pi(s)) = v_\pi(s)$$

- Then the Bellman optimality equation has been satisfied

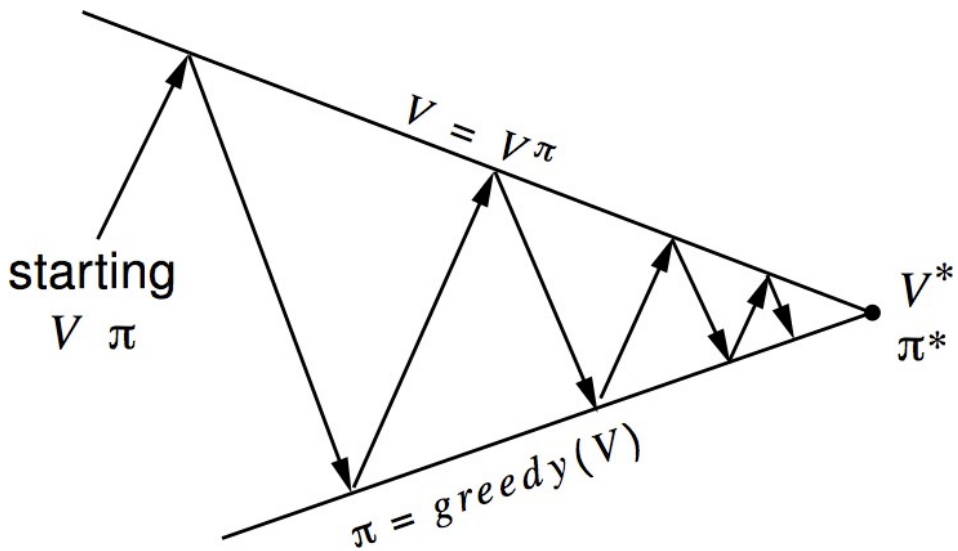
$$v_\pi(s) = \max_{a \in A} q_\pi(s, a)$$

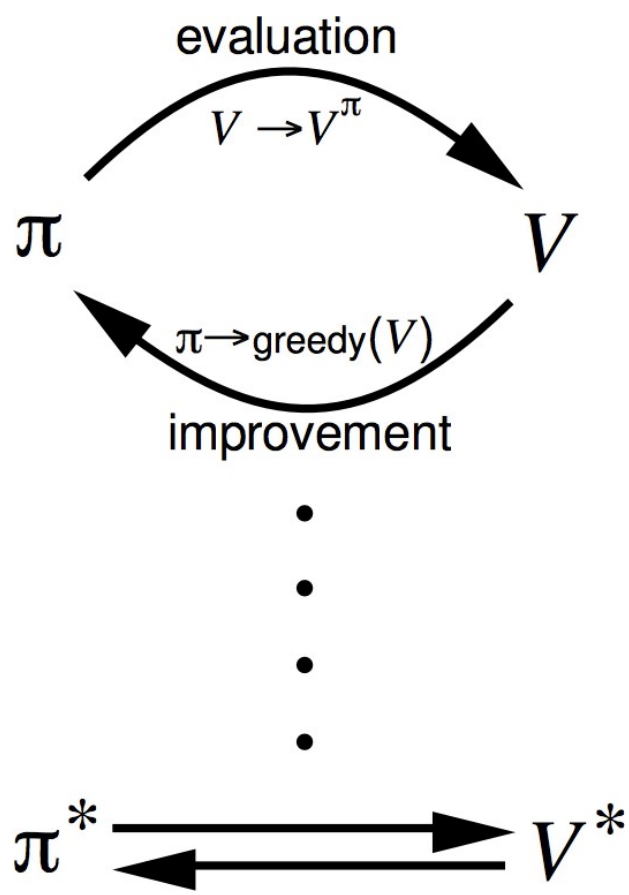
- Therefore $v_\pi(s) = v_*(s)$ for all $s \in S$
- so π is an optimal policy

Modified Policy Iteration

- Does policy evaluation need to converge to v_π?
- Or should we introduce a stopping condition
  - e.g. ε-convergence of value function
- Or simply stop after k iterations of iterative policy evaluation?
- For example, in the small gridworld $k = 3$ was sufficient to achieve optimal policy
- Why not update policy every iteration? i.e. stop after $k = 1$
  - This is equivalent to *value iteration* (next section)

Generalised Policy Iteration





Policy evaluation Estimate v_π

Any policy evaluation algorithm

Policy improvement Generate $\pi' \geq \pi$

Any policy improvement algorithm

Principle of Optimality

Any optimal policy can be subdivided into two components:

- An optimal first action A_*
- Followed by an optimal policy from successor state S'

Theorem (Principle of Optimality)

A policy $\pi(a|s)$ achieves the optimal value from state s, $v_\pi(s) = v_*(s)$, if and only if

- *For any state s' reachable from s*
- *π achieves the optimal value from state s', $v_\pi(s') = v_*(s')$*

Deterministic Value Iteration

- If we know the solution to subproblems $v_*(s')$
- Then solution $v_*(s)$ can be found by one-step lookahead

$$v_*(s) \leftarrow \max_{a \in \mathcal{A}} \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

- The idea of value iteration is to apply these updates iteratively
- Intuition: start with final rewards and work backwards
- Still works with loopy, stochastic MDPs

Value Iteration

$$
\begin{aligned}
v_{k+1}(s) &\doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_k(s')\Big], \qquad (4.10)
\end{aligned}
$$

for all $s \in \mathcal{S}$. For arbitrary v_0, the sequence $\{v_k\}$ can be shown to converge to v_* under the same conditions that guarantee the existence of v_*.

Value Iteration

Value iteration

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

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

Output a deterministic policy, $\pi \approx \pi_*$, such that
  $\pi(s) = \arg\max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$

Example: Shortest Path

g			

Problem

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

V_1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

V_2

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

V_3

0	-1	-2	-3
-1	-2	-3	-3
-2	-3	-3	-3
-3	-3	-3	-3

V_4

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-4
-3	-4	-4	-4

V_5

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-5

V_6

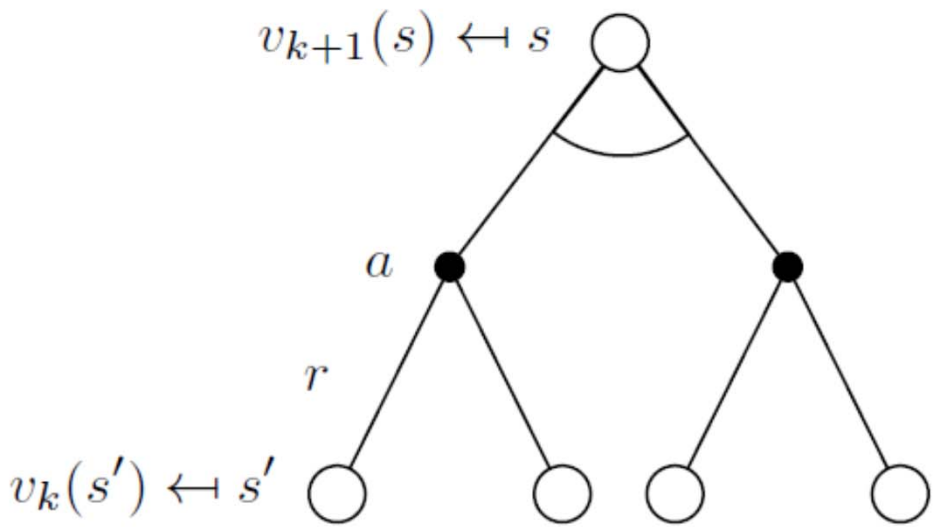
0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

Value Iteration

- Problem: find optimal policy π
- Solution: iterative application of Bellman optimality backup
- $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_*$
- Using synchronous backups
  - At each iteration $k + 1$
  - For all states $s \in S$
  - Update $v_{k+1}(s)$ from $v_k(s')$
- Convergence to v_* will be proven later
- Unlike policy iteration, there is no explicit policy
- Intermediate value functions may not correspond to any policy

Value Iteration (2)



$$v_{k+1}(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

$$\mathbf{v}_{k+1} = \max_{a \in \mathcal{A}} \boldsymbol{\mathcal{R}}^a + \gamma \boldsymbol{\mathcal{P}}^a \mathbf{v}_k$$

Synchronous Dynamic Programming Algorithms

Summary of DP Algorithms

Problem	Bellman Equation	Algorithm
Prediction	Bellman Expectation Equation	Iterative Policy Evaluation
Control	Bellman Expectation Equation + Greedy Policy Improvement	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

- Algorithms are based on state-value function $v_\pi(s)$ or $v_*(s)$
- Complexity $O(mn^2)$ per iteration, for m actions and n states
- Could also apply to action-value function $q_\pi(s, a)$ or $q_*(s, a)$
- Complexity $O(m^2n^2)$ per iteration

Asynchronous Dynamic Programming

Asynchronous Dynamic Programming

- DP methods described so far used *synchronous* backups
- i.e. all states are backed up in parallel
- *Asynchronous DP* backs up states individually, in any order
- For each selected state, apply the appropriate backup
- Can significantly reduce computation
- Guaranteed to converge if all states continue to be selected

Asynchronous Dynamic Programming

Three simple ideas for asynchronous dynamic programming:

- *In-place* dynamic programming
- *Prioritised sweeping*
- *Real-time* dynamic programming

In-Place Dynamic Programming

- Synchronous value iteration stores two copies of value function

  for all s in $\mathcal{S}$

$$v_{new}(s) \leftarrow \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_{old}(s') \right)$$

  $v_{old} \leftarrow v_{new}$

- In-place value iteration only stores one copy of value function

  for all s in $\mathcal{S}$

$$v(s) \leftarrow \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v(s') \right)$$

Prioritised Sweeping

- Use magnitude of Bellman error to guide state selection, e.g.

$$\left| \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v(s') \right) - v(s) \right|$$

- Backup the state with the largest remaining Bellman error
- Update Bellman error of affected states after each backup
- Requires knowledge of reverse dynamics (predecessor states)
- Can be implemented efficiently by maintaining a priority queue

Real-Time Dynamic Programming

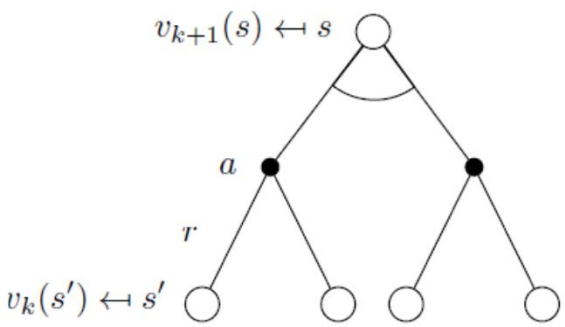
Asynchronous Dynamic Programming

- Idea: only states that are relevant to agent
- Use agent's experience to guide the selection of states
- After each time-step S_t, A_t, R_{t+1}
- Backup the state S_t

$$v(S_t) \leftarrow \max_{a \in \mathcal{A}} \left(\mathcal{R}^a_{S_t} + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{S_t s'} v(s') \right)$$

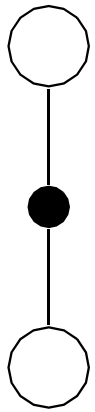
Full-Width Backups

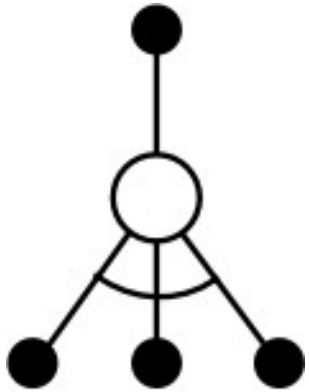
- DP uses *full-width* backups
- For each backup (sync or async)
  - Every successor state and action is considered
  - Using knowledge of the MDP transitions and reward function
- DP is effective for medium-sized problems (millions of states)
- For large problems DP suffers Bellman's *curse of dimensionality*
  - Number of states $n = |S|$ grows exponentially with number of state variables
- Even one backup can be too expensive



Sample Backups

- In subsequent lectures we will consider *sample backups*
- Using sample rewards and sample transitions (S, A, R, S')
- Instead of reward function R and transition dynamics P
- Advantages:
  - Model-free: no advance knowledge of MDP required
  - Breaks the curse of dimensionality through sampling
  - Cost of backup is constant, independent of $n = |S|$





Approximate Dynamic Programming

- Approximate the value function
- Using a *function approximator* $\hat{v}(s, \mathbf{w})$
- Apply dynamic programming to $\hat{v}(\cdot, \mathbf{w})$
- e.g. Fitted Value Iteration repeats at each iteration k,
  - Sample states $\tilde{\mathcal{S}} \subseteq \mathcal{S}$
  - For each state $s \in \tilde{\mathcal{S}}$, estimate target value using Bellman optimality equation,

$$\tilde{v}_k(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \hat{v}(s', \mathbf{w_k}) \right)$$

  - Train next value function $\hat{v}(\cdot, \mathbf{w_{k+1}})$ using targets $\{\langle s, \tilde{v}_k(s) \rangle\}$

Csaba slides,

The fundamental theorem and the Bellman (optimality) operator

Theorem

Assume that $|\mathcal{A}| < +\infty$. Then the optimal value function satisfies

$$V^*(x) = \max_{a \in \mathcal{A}} \left\{ r(x,a) + \gamma \sum_{y \in \mathcal{X}} \mathcal{P}(x,a,y) V^*(y) \right\}, \qquad x \in \mathcal{X}.$$

and if policy π is such that in each state x it selects an action that maximizes the r.h.s. then π is an optimal policy.

A shorter way to write this is

$$V^* = T^* V^*,$$

$$(T^* V)(x) = \max_{a \in \mathcal{A}} \left\{ r(x,a) + \gamma \sum_{y \in \mathcal{X}} \mathcal{P}(x,a,y) V(y) \right\}, \quad x \in \mathcal{X}.$$

Policy evaluation operator

Definition (Policy evaluation operator)

Let π be a stochastic stationary policy. Define

$$\begin{aligned}(T^\pi V)(x) &= \sum_{a\in\mathcal{A}} \pi(a|x)\left\{r(x,a) + \gamma \sum_{y\in\mathcal{X}} \mathcal{P}(x,a,y)V(y)\right\} \\ &= \sum_{a\in\mathcal{A}} \pi(a|x) T_a V(x), \quad x \in \mathcal{X}.\end{aligned}$$

Corollary

T^π is a contraction, and V^π is the unique fixed point of T^π.

Greedy policy

Definition (Greedy policy)

Policy π is greedy w.r.t. V if

$$T^{\pi} V = T^{*} V,$$

or

$$\sum_{a \in \mathcal{A}} \pi(a|x) \left\{ r(x,a) + \gamma \sum_{y \in \mathcal{X}} \mathcal{P}(x,a,y) V(y) \right\} = \max_{a \in \mathcal{A}} \left\{ r(x,a) + \gamma \sum_{y \in \mathcal{X}} \mathcal{P}(x,a,y) V(y) \right\}$$

holds for all states x.

A restatement of the main theorem

Theorem

Assume that $|\mathcal{A}| < +\infty$. Then the optimal value function satisfies the fixed-point equation $V^* = T^* V^*$ and any greedy policy w.r.t. V^* is optimal.

Action-value functions

Corollary

Let Q^* be the optimal action-value function. Then,

$$Q^* = T^* Q^*$$

and if π is a policy such that

$$\sum_{a \in \mathcal{A}} \pi(a|x) Q^*(x,a) = \max_{a \in \mathcal{A}} Q^*(x,a)$$

then π is optimal. Here,

$$T^* Q(x,a) = r(x,a) + \gamma \sum_{y \in \mathcal{X}} \mathcal{P}(x,a,y) \max_{a' \in \mathcal{A}} Q(y,a'), \quad x \in \mathcal{X}, a \in \mathcal{A}.$$

Finding the action-value functions of policies

Theorem

Let π be a stationary policy, T^π be defined by

$$T^\pi Q(x,a) = r(x,a) + \gamma \sum_{y \in \mathcal{X}} \mathcal{P}(x,a,y) \sum_{a' \in \mathcal{A}} \pi(a'|y)\, Q(y,a'), \quad x \in \mathcal{X}, a \in \mathcal{A}.$$

Then Q^π is the unique solution of

$$T^\pi Q^\pi = Q^\pi.$$

Value iteration

Note

- *If V_t is the value-function computed in the tth iteration of value iteration then*

$$V_{t+1} = T^* V_t.$$

- *The key is that T^* is a contraction in the supremum norm and Banach's fixed-point theorem gives the key to the proof the theorem mentioned before.*

Note

One can also use $Q_{t+1} = T^* Q_t$, or value functions with post-decision states. What is the advantage?

Policy iteration

```
function PolicyIteration(π)
repeat
    π' ← π
    V ← GetValueFunction(π')
    π ← GetGreedyPolicy(V)
until π ≠ π'
return π
```

What if we stop early?

Theorem (e.g., Corollary 2 of Singh and Yee 1994)

Fix an action-value function Q and let π be a greedy policy w.r.t. Q. Then the value of policy π can be lower bounded as follows:

$$V^{\pi}(x) \geq V^{*}(x) - \frac{2}{1-\gamma} \|Q - Q^{*}\|_{\infty}, \quad x \in \mathcal{X}.$$

Some Technical Questions

- How do we know that value iteration converges to v_*?
- Or that iterative policy evaluation converges to v_π?
- And therefore that policy iteration converges to v_*?
- Is the solution unique?
- How fast do these algorithms converge?
- These questions are resolved by *contraction mapping theorem*

Value Function Space

- Consider the vector space $\mathbb{V}$ over value functions
- There are $|\mathbb{S}|$ dimensions
- Each point in this space fully specifies a value function $v(s)$
- What does a Bellman backup do to points in this space?
- We will show that it brings value functions *closer*
- And therefore the backups must converge on a unique solution

Value Function ∞-Norm

- We will measure distance between state-value functions u and v by the ∞-norm
- i.e. the largest difference between state values,

$$||u - v||_\infty = \max_{s \in S} |u(s) - v(s)|$$

Bellman Expectation Backup is a Contraction

- Approximate the value function
- Using a *function approximator* $\hat{v}(s, \mathbf{w})$
- Apply dynamic programming to $\hat{v}(\cdot, \mathbf{w})$
- e.g. Fitted Value Iteration repeats at each iteration k,
  - Sample states $\tilde{\mathcal{S}} \subseteq \mathcal{S}$
  - For each state $s \in \tilde{\mathcal{S}}$, estimate target value using Bellman optimality equation,

$$\tilde{v}_k(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \hat{v}(s', \mathbf{w_k}) \right)$$

  - Train next value function $\hat{v}(\cdot, \mathbf{w_{k+1}})$ using targets $\{\langle s, \tilde{v}_k(s) \rangle\}$

Contraction Mapping Theorem

Theorem (Contraction Mapping Theorem)

For any metric space $\mathbb{V}$ *that is complete (i.e. closed) under an operator* $T(v)$*, where* T *is a* γ*-contraction,*

- T *converges to a unique fixed point*
- *At a linear convergence rate of* γ

Bellman Operator is a Contraction

|| V-V'|| = Infinity norm
(find max diff
Over all states)

$$\|BV - BV'\| = \left\| \begin{array}{l} \max_a \left[R(s,a) + \gamma \sum_{s_j \in S} p(s_j \mid s_i, a) V(s_j) \right] \\ - \max_{a'} \left[R(s,a') - \gamma \sum_{s_j \in S} p(s_j \mid s_i, a') V'(s_j) \right] \end{array} \right\|$$

$$\leq \left\| \max_a \left[R(s,a) + \gamma \sum_{s_j \in S} p(s_j \mid s_i, a) V(s_j) - R(s,a) + \gamma \sum_{s_j \in S} p(s_j \mid s_i, a) V'(s_j) \right] \right\|$$

$$\leq \gamma \left\| \max_a \left[\sum_{s_j \in S} p(s_j \mid s_i, a) V(s_j) - \sum_{s_j \in S} p(s_j \mid s_i, a) V'(s_j) \right] \right\|$$

$$= \gamma \max_a \left\| \left[\sum_{s_j \in S} p(s_j \mid s_i, a) (V(s_j) - V'(s_j)) \right] \right\|$$

$$\leq \gamma \max_{a, s_i} \sum_{s_j \in S} p(s_j \mid s_i, a) \left| V(s_j) - V'(s_j) \right|$$

$$\leq \gamma \max_{a, s_i} \sum_{s_j \in S} p(s_j \mid s_i, a) \|V - V'\|$$

$$= \gamma \|V - V'\|$$

Convergence of Iter. Policy Evaluation and Policy Iteration

- The Bellman expectation operator T^{π} has a unique fixed point
- v_{π} is a fixed point of T^{π} (by Bellman expectation equation)
- By contraction mapping theorem
- Iterative policy evaluation converges on v_{π}
- Policy iteration converges on v_*

Bellman Optimality Backup is a Contraction

- Define the *Bellman optimality backup operator* T^*,

$$T^*(v) = \max_{a \in A} \mathrm{R}^a + \gamma \mathrm{P}^a v$$

- This operator is a γ-contraction, i.e. it makes value functions closer by at least γ (similar to previous proof)

$$||T^*(u) - T^*(v)||_\infty \leq \gamma ||u - v||_\infty$$

Convergence of Value Iteration

- The Bellman optimality operator T_* has a unique fixed point
- v_* is a fixed point of T_* (by Bellman optimality equation) By
- contraction mapping theorem
- Value iteration converges on v_*

Will Value Iteration Converge?

- Yes, if discount factor is < 1 or end up in a terminal state with probability 1

- Bellman equation is a contraction
- If apply it to two different value functions, distance between value functions shrinks after apply Bellman equation to each

Properties of Contraction

- Only has 1 fixed point
  - If had two, then would not get closer when apply contraction function, violating definition of contraction
- When apply contraction function to any argument, value must get closer to fixed point
  - Fixed point doesn't move
  - Repeated function applications yield fixed point

Value Iteration Converges

- If discount factor < 1
- Bellman is a contraction
- Value iteration converges to unique solution which is optimal value function