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


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


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)


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


Course Outline, Silver

Part I: Elementary Reinforcement Learning
- Introduction to RL
- Markov Decision Processes
- Planning by Dynamic Programming
- Model-Free Prediction
- Model-Free Control

Part II: Reinforcement Learning in Practice
- Value Function Approximation
- Policy Gradient Methods
- Integrating Learning and Planning
- Exploration and Exploitation
- Case study - RL in games
Introduction to Reinforcement Learning

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

observation $O_t$

reward $R_t$

action $A_t$
Branches of Machine Learning

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

Machine Learning

295, Fall 2019
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
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** $(S,A,P,R,\gamma)$
- 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 $\gamma$
- 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: $\gamma$
• 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 Functions**
  \[ Q^\pi(s, a) = \sum_{s' \in S} p(s' \mid s, a) \left[ R(s, a, s') + \gamma V^\pi(s') \right] \]

- **Optimal Value & Optimal Policy**
  \[ V^* (s_i) = \max_a \left( \sum_{s_j \in S} p(s_j \mid s_i, a) \left[ R(s, \pi(s), s') + \gamma V^* (s_j) \right] \right) \]
  \[ = \max_a Q^* (s, a) \]
  \[ \pi^* (s) = \arg\max_a Q^* (s, a) \]
Most of the story in a nutshell:

**Bellman Equation**

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

- 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 $k < \text{desired horizon}$ or (if infinite horizon) values have converged
   - For all $s$,
     
     $$V_k(s_i) = \max_a \left( \sum_{s_j \in S} p(s_j \mid s_i, a) \left[ R(s, \pi(s), s') + \gamma V_{k-1}(s_j) \right] \right)$$
     
     $$\pi_k(s_i) = \arg\max_a \left( \sum_{s_j \in S} p(s_j \mid s_i, a) \left[ R(s, \pi(s), s') + \gamma V_{k-1}(s_j) \right] \right)$$
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**

\[
\| V - V' \| = \text{Infinity norm} \\
\text{(find max diff)} \\
\text{Over all states)}
\]

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

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

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

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

\[
\leq \gamma \max_{a, s_i} \sum_{s_j \in S} p(s_j | s_i, a) \| V(s_j) - V'(s_j) \| \\
\leq \gamma \max_{a, s_i} \sum_{s_j \in S} p(s_j | s_i, a) \| V - V' \| \\
= \gamma \| V - V' \|
\]
Most of the story in a nutshell:

Properties of Contraction

• Only has 1 fixed point
  o 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
  o Fixed point doesn’t move
  o 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

Quiz!

What is the shortest sequence getting from start to goal?

Up, down, left, right
The World - 2

Quiz!

What is the reliability of our sequence:

Up Up Right Right Right Right?

Up, Down, Left, Right
- Actions executes .8
- Move at right angle .1 and .1

Goal
Start
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:

\[ S_t = f(H_t) \]
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
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
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 is a prediction of future reward
- Used to evaluate the goodness/badness of states
- And therefore to select between actions, e.g.

\[ V_\pi(s) = \mathbb{E}_\pi R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \mid S_t = s \]
A model predicts what the environment will do next
- $\mathcal{P}$ predicts the next state
- $\mathcal{R}$ predicts the next (immediate) reward, e.g.

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

Numbers represent immediate reward $R_s^a$ from each state $s$ (same for all $a$)
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
  
  \[ S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_3, R_3, S_3, \ldots \]

• The process is governed by a transition function

  \[ p(s', r | s, a) = \Pr\{S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a\}, \]

• 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
“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
For a Markov state $s$ and successor state $s'$, the state transition probability is defined by

$$P_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$$

State transition matrix $P$ defines transition probabilities from all states $s$ to all successor states $s'$,

$$P = \begin{bmatrix} P_{11} & \cdots & P_{1n} \\ \vdots \\ P_{n1} & \cdots & P_{nn} \end{bmatrix}$$

where each row of the matrix sums to 1.
A Markov process is a memoryless random process, i.e. a sequence of random states $S_1, S_2, \ldots$ 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 $S_1 = C1$

$S_1, S_2, ..., S_T$

- 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

Transition Matrix:

<table>
<thead>
<tr>
<th></th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>Pass</th>
<th>Pub</th>
<th>FB</th>
<th>Sleep</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>0.5</td>
<td>0.8</td>
<td>0.6</td>
<td>0.4</td>
<td></td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td>0.8</td>
<td>0.6</td>
<td>0.4</td>
<td>0.4</td>
<td></td>
<td>0.9</td>
<td></td>
</tr>
<tr>
<td>C3</td>
<td>0.6</td>
<td>0.4</td>
<td>0.4</td>
<td>0.9</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Graph:

- States: Class 1, Class 2, Class 3, Pass, Sleep, Facebook, Pub
- Transitions:
  - From Class 1 to Class 2: 0.5
  - From Class 1 to Class 3: 0.2
  - From Class 1 to Sleep: 0.4
  - From Class 2 to Class 3: 0.8
  - From Class 2 to Sleep: 0.4
  - From Class 3 to Pass: 0.6
  - From Sleep to Class 1: 0.9
  - From Sleep to Class 2: 0.1
  - From Sleep to Class 3: 0.4
  - From Sleep to Pass: 0.4
  - From Facebook to Sleep: 0.9
  - From Facebook to Pub: 0.1
  - From Pub to Class 1: 0.4
  - From Pub to Class 2: 0.4
  - From Pub to Class 3: 0.4
  - From Pub to Sleep: 0.4

295, Fall 2019
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: $\gamma$
- 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:

\[ G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T, \]
**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.
  - $\gamma$ close to 0 leads to “myopic” evaluation.
  - $\gamma$ close to 1 leads to “far-sighted” evaluation.
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.
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 $s$

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

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

The diagram illustrates a Markov Decision Process (MRP) for a student's daily activities. The student can choose to sleep, go to class 1, class 2, class 3, Facebook, or the pub. The rewards for each action are as follows:

- Sleep: $R = -1$
- Class 1: $R = -2$
- Class 2: $R = -2$
- Class 3: $R = -2$
- Facebook: $R = +1$
- Pub: $R = +10$
- Pass: $R = +10$

The probabilities for transitioning between states are also shown in the diagram.
Example: Student MRP Returns

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

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

<table>
<thead>
<tr>
<th>State Sequence</th>
<th>Value $v_1$</th>
<th>$\gamma^{T-2}$</th>
<th>$v_1$ Final Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1 C2 C3 Pass Sleep</td>
<td>$-2 - 2 \times \frac{1}{2} - 2 \times \frac{1}{4} + 10 \times \frac{1}{8}$</td>
<td>$\frac{1}{2}$</td>
<td>$-2.25$</td>
</tr>
<tr>
<td>C1 FB FB C1 C2 Sleep</td>
<td>$-2 - 1 \times \frac{1}{2} - 1 \times \frac{1}{4} - 2 \times \frac{1}{8} - 2 \times \frac{1}{16}$</td>
<td>$\frac{1}{4}$</td>
<td>$-3.125$</td>
</tr>
<tr>
<td>C1 C2 C3 Pub C2 C3 Pass Sleep C1</td>
<td>$-2 - 2 \times \frac{1}{2} - 2 \times \frac{1}{4} + 1 \times \frac{1}{8} - 2 \times \frac{1}{16} \ldots$</td>
<td>$\frac{1}{8}$</td>
<td>$-3.41$</td>
</tr>
<tr>
<td>FB FB C1 C2 C3 Pub C1 ... FB</td>
<td>$-2 - 1 \times \frac{1}{2} - 1 \times \frac{1}{4} - 2 \times \frac{1}{8} - 2 \times \frac{1}{16} \ldots$</td>
<td>$\frac{1}{16}$</td>
<td>$-3.20$</td>
</tr>
</tbody>
</table>
The value function can be decomposed into two parts:

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

$$v(s) = E \left[ G_t \mid S_t = s \right]$$

$$= E \left[ R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s \right]$$

$$= E \left[ R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \ldots) \mid S_t = s \right]$$

$$= E \left[ R_{t+1} + \gamma G_{t+1} \mid S_t = s \right]$$

$$= E \left[ R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s \right]$$
Bellman Equation for MRPs (2)

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

![](image)

\[ v(s) = R_s + \gamma \sum_{s' \in S} P_{ss'} v(s') \]
Example: Bellman Equation for Student MRP

\[ 4.3 = -2 + 0.6 \times 10 + 0.4 \times 0.8 \]
The Bellman equation can be expressed concisely using matrices,

$$v = R + \gamma P v$$

where $v$ is a column vector with one entry per state.
The Bellman equation is a linear equation. It can be solved directly:

\[ v = R + \gamma P v \]

\[ (I - \gamma P) v = R \]

\[ v = (I - \gamma P)^{-1} R \]

- Computational complexity is \( O(n^3) \) for \( n \) states
- Direct solution only possible for small MRP\( s \)
- There are many iterative methods for large MRP\( s \), e.g.
  - Dynamic programming
  - Monte-Carlo evaluation
  - Temporal-Difference learning
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 \( \langle S, A, P, R, \gamma \rangle \)

- \( S \) is a finite set of states
- \( A \) is a finite set of actions
- \( P \) is a state transition probability matrix,
  \[ P_{ss'}^a = \mathbb{P} [ S_{t+1} = s' \mid S_t = s, A_t = a ] \]
- \( R \) is a reward function,
  \[ R_s^a = \mathbb{E} [ R_{t+1} \mid S_t = s, A_t = a ] \]
- \( \gamma \) is a discount factor \( \gamma \in [0, 1] \).
Example: Student MDP
Definition

A policy \( \pi \) 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),
  \( A_t \sim \pi(\cdot \mid S_t), \quad \forall t > 0 \)
Policy’s and Value functions

\[ v_\pi(s) \equiv E_\pi[G_t \mid S_t = s] = E_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] = \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[ r + \gamma E_\pi[G_{t+1} \mid S_{t+1} = s'] \right] = \sum_a \pi(a \mid s) \sum_{s', r} p(s', r \mid s, a) \left[ r + \gamma v_\pi(s') \right], \text{ for all } s \in \mathcal{S}, \] (by (3.9))

(3.14)
Gridworld Example: Prediction

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

Policy: actions are uniformly random.

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 $\pi$

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

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

$$q_\pi(s, a) = E_\pi [G_t \mid S_t = s, A_t = a]$$
The state-value function can again be decomposed into immediate reward plus discounted value of successor state,

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

The action-value function can similarly be decomposed,

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

Expressing the functions recursively,
Will translate to one step look-ahead.
Bellman Expectation Equation for $V^π$

\[
v_\pi(s) = \sum_{a \in A} \pi(a|s)q_\pi(s, a)
\]
Bellman Expectation Equation for $Q^\pi$

\[ q_\pi(s, a) = R_s + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') \]
The Bellman Expectation Equation for $v_\pi$ is given by:

$$v_\pi(s) = \sum_{a \in A} \pi(a | s) \left( R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') \right)$$
Bellman Expectation Equation for $q_\pi$ (2)

$$q_\pi(s, a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a'|s') q_\pi(s', a')$$
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

$v^*(s)$ for $\gamma = 1$

States and Transitions:
- State 6: Study, $R = -2$
  - Transitions: Facebook ($R = -1$), Quit ($R = 0$)
- State 8: Study, $R = -2$
  - Transitions: Sleep ($R = 0$)
- State 10: Study, $R = +10$
  - Transitions: Pub ($R = +1$)

Rewards:
- Facebook: $R = -1$
- Quit: $R = 0$
- Sleep: $R = 0$
- Pub: $R = +1$
Optimal Action-Value Function for Student MDP

$q^*(s,a)$ for $\gamma = 1$

- **Facebook**
  - $R = -1$
  - $q^* = 5$

- **Quit**
  - $R = 0$
  - $q^* = 6$

- **Sleep**
  - $R = 0$
  - $q^* = 0$

- **Study**
  - $R = -2$
  - $q^* = 6$

- **Pub**
  - $R = +1$
  - $q^* = 8.4$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
  - $q^* = 10$

- **Study**
  - $R = +10$
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 \( \pi^* \) 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, \( q_{\pi^*}(s, a) = q_*(s, a) \)
An optimal policy can be found by maximising over $q_*(s, a)$,

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

- 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**

\[ \pi^*(a|s) \text{ for } \gamma = 1 \]

States:
- **6**: Facebook, \( R = -1 \), \( q^* = 5 \)
- **8**: Study, \( R = -2 \), \( q^* = 6 \)
- **10**: Study, \( R = +10 \), \( q^* = 10 \)

Actions:
- **Facebook**
  - \( R = -1 \)
  - \( q^* = 5 \)
- **Quit**
  - \( R = 0 \)
  - \( q^* = 6 \)
- **Study**
  - \( R = -2 \)
  - \( q^* = 6 \)
- **Sleep**
  - \( R = 0 \)
  - \( q^* = 0 \)
- **Pub**
  - \( R = +1 \)
  - \( q^* = 8.4 \)

Transitions:
- **6** to **6** with probability \( 0.4 \)
- **6** to **8** with probability \( 0.2 \)
- **6** to **10** with probability \( 0.4 \)
- **8** to **6** with probability \( 0.4 \)
- **8** to **10** with probability \( 0.6 \)
- **10** to **6** with probability \( 0.5 \)
- **10** to **8** with probability \( 0.5 \)
Bellman Equation for $V^*$ and $Q^*$

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

$$q^*(s; a) = \sum_{s', r} p(s', r | s, a) \left[ r + \gamma \max_{a'} q^*(s', a') \right] .$$
Example: Bellman Optimality Equation in Student MDP

\[ 6 = \max \{-2 + 8, -1 + 6\} \]
Gridworld Example: Control

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

Figure 3.6
- Bellman Optimality Equation is non-linear
- No closed form solution (in general)
- Many iterative solution methods
  - Value Iteration
  - Policy Iteration
  - Q-learning
  - Sarsa
Outline

1. Introduction
2. Policy Evaluation
3. Policy Iteration
4. Value Iteration
5. Extensions to Dynamic Programming
6. Contraction Mapping
Dynamic programming assumes full knowledge of the MDP

It is used for planning in an MDP

For prediction:
- Input: MDP \((S, A, P, R, \gamma)\) and policy \(\pi\)
- or: MRP \((S, P^\pi, R^\pi, \gamma)\)
- Output: value function \(v^\pi\)

Or for control:
- Input: MDP \((S, A, P, R, \gamma)\)
- Output: optimal value function \(v^*\)
- and: optimal policy \(\pi^*\)
Policy Evaluation (Prediction)

- Problem: evaluate a given policy \( \pi \)
- Solution: iterative application of Bellman expectation backup
  \[
  v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_\pi
  \]
  Using \textit{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 \textit{asynchronous} backups later
- Convergence to \( v_\pi \) will be proven at the end of the lecture
Iterative Policy Evaluations

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

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

\[
v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]
= \sum_a \pi(a \mid s) \sum_{s', r} p(s' \mid s, a) \left[ r + \gamma v_\pi(s') \right],
\]

Iterative policy evaluation.
Iterative Policy Evaluation (2)

\[
v_{k+1}(s) = \sum_{a \in A} \pi(a | s) \left( R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s') \right)
\]

\[
v^{k+1} = R^\pi + \gamma P^\pi v^k
\]
Iterative policy Evaluation

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

<table>
<thead>
<tr>
<th>$k = 0$</th>
<th>$V_k$ for the Random Policy</th>
<th>Greedy Policy w.r.t. $V_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0 0.0 0.0 0.0</td>
<td><img src="image1" alt="Random Policy diagram" /></td>
<td><img src="image2" alt="Greedy Policy diagram" /></td>
</tr>
<tr>
<td>0.0 0.0 0.0 0.0</td>
<td><img src="image3" alt="Random Policy diagram" /></td>
<td><img src="image4" alt="Greedy Policy diagram" /></td>
</tr>
<tr>
<td>0.0 0.0 0.0 0.0</td>
<td><img src="image5" alt="Random Policy diagram" /></td>
<td><img src="image6" alt="Greedy Policy diagram" /></td>
</tr>
<tr>
<td>0.0 0.0 0.0 0.0</td>
<td><img src="image7" alt="Random Policy diagram" /></td>
<td><img src="image8" alt="Greedy Policy diagram" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$k = 1$</th>
<th>$V_k$ for the Random Policy</th>
<th>Greedy Policy w.r.t. $V_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0 -1.0 -1.0 -1.0</td>
<td><img src="image9" alt="Random Policy diagram" /></td>
<td><img src="image10" alt="Greedy Policy diagram" /></td>
</tr>
<tr>
<td>-1.0 -1.0 -1.0 -1.0</td>
<td><img src="image11" alt="Random Policy diagram" /></td>
<td><img src="image12" alt="Greedy Policy diagram" /></td>
</tr>
<tr>
<td>-1.0 -1.0 -1.0 -1.0</td>
<td><img src="image13" alt="Random Policy diagram" /></td>
<td><img src="image14" alt="Greedy Policy diagram" /></td>
</tr>
<tr>
<td>-1.0 -1.0 -1.0 0.0</td>
<td><img src="image15" alt="Random Policy diagram" /></td>
<td><img src="image16" alt="Greedy Policy diagram" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$k = 2$</th>
<th>$V_k$ for the Random Policy</th>
<th>Greedy Policy w.r.t. $V_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0 -1.7 -2.0 -2.0</td>
<td><img src="image17" alt="Random Policy diagram" /></td>
<td><img src="image18" alt="Greedy Policy diagram" /></td>
</tr>
<tr>
<td>-1.7 -2.0 -2.0 -2.0</td>
<td><img src="image19" alt="Random Policy diagram" /></td>
<td><img src="image20" alt="Greedy Policy diagram" /></td>
</tr>
<tr>
<td>-2.0 -2.0 -2.0 -1.7</td>
<td><img src="image21" alt="Random Policy diagram" /></td>
<td><img src="image22" alt="Greedy Policy diagram" /></td>
</tr>
<tr>
<td>-2.0 -2.0 -1.7 0.0</td>
<td><img src="image23" alt="Random Policy diagram" /></td>
<td><img src="image24" alt="Greedy Policy diagram" /></td>
</tr>
</tbody>
</table>
Iterative Policy Evaluation in Small Gridworld (2)

\[ k = 3 \]

\begin{array}{cccc}
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 \\
\end{array}

\[ k = 10 \]

\begin{array}{cccc}
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 \\
\end{array}

\[ k = \infty \]

\begin{array}{cccc}
0.0 & -14. & -20. & -22. \\
-22. & -20. & -14. & 0.0 \\
\end{array}
Given a policy $\pi$

- Evaluate the policy $\pi$

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

- Improve the policy by acting greedily with respect to $v_\pi$

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

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

\[ \pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots \xrightarrow{I} \pi_* \xrightarrow{E} v_* , \]

where \( \xrightarrow{E} \) denotes a policy evaluation and \( \xrightarrow{I} \) 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
   
   \[ V(s) \in \mathbb{R} \text{ and } \pi(s) \in \mathcal{A}(s) \text{ arbitrarily for all } s \in \mathcal{S} \]

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

3. Policy Improvement
   
   \( policy-stable \leftarrow true \)
   
   For each \( s \in \mathcal{S} \):
   
   \[ old-action \leftarrow \pi(s) \]
   
   \[ \pi(s) \leftarrow \text{argmax}_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')] \]
   
   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 $v_{\pi}$
   Iterative policy evaluation
Policy improvement Generate $\pi^l \geq \pi$
   Greedy policy improvement

$V = V\pi$
$\pi = \text{greedy}(V)$

$V \rightarrow V^{\pi}$
$\pi \rightarrow \text{greedy}(V)$

$V^* \rightarrow V^*$
Policy Improvement

- Consider a deterministic policy, \( a = \pi(s) \)
- We can improve the policy by acting greedily
  \[
  \pi'(s) = \arg \max_{a \in A} q_{\pi}(s, a)
  \]
- This improves the value from any state \( s \) over one step,
  \[
  q_{\pi}(s, \pi'(s)) = \max_{a \in A} q_{\pi}(s, a) \geq q_{\pi}(s, \pi(s)) = v_{\pi}(s)
  \]
- It therefore improves the value function,
  \[
  v_{\pi'}(s) \geq v_{\pi}(s)
  \]
  \[
  v_{\pi}(s) \leq q_{\pi}(s, \pi'(s)) = E_{\pi'} \left[ R_{t+1} + \gamma v_{\pi}(S_{t+1}) \mid S_t = s \right]
  \leq E_{\pi'} \left[ R_{t+1} + \gamma q_{\pi}(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s \right]
  \leq E_{\pi'} \left[ R_{t+1} + \gamma R_{t+2} + \gamma^2 q_{\pi}(S_{t+2}, \pi'(S_{t+2})) \mid S_t = s \right]
  \leq E_{\pi'} \left[ R_{t+1} + \gamma R_{t+2} + \ldots \mid S_t = s \right] = v_{\pi'}(s)
  \]
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 \( \pi \) is an optimal policy.
Modified Policy Iteration

- Does policy evaluation need to converge to $v_\pi$?
- Or should we introduce a stopping condition
  - e.g. $E$-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 $v_{\pi}$
- Any policy evaluation algorithm

**Policy improvement** Generate $\pi' \geq \pi$
- Any policy improvement algorithm

$$V = V^\pi$$
$$\pi = \text{greedy}(V)$$

Starting

$V \pi$

$V^*$

$\pi^*$

Evaluation

$V \rightarrow V^\pi$

Improvement

$\pi \rightarrow \text{greedy}(V)$

$\pi^* \leftrightarrow V^*$
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 $s$
- $\pi$ 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

$$v_*(s) \leftarrow \max_{a \in A} R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s')$$

- 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

\[ v_{k+1}(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] \]
\[ = \max_a \sum_{s', r} p(s', r \mid s, a) \left[ r + \gamma v_k(s') \right], \quad (4.10) \]

for all \( s \in S \). 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

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

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

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \arg\max_a \sum_{s', r} p(s', r | s, a) [r + \gamma V(s')]$
## Example: Shortest Path

<table>
<thead>
<tr>
<th>Problem</th>
<th>$V_1$</th>
<th>$V_2$</th>
<th>$V_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$V_4$</th>
<th>$V_5$</th>
<th>$V_6$</th>
<th>$V_7$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>-3</td>
<td>-3</td>
<td>-3</td>
<td>-3</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>-3</td>
<td>-3</td>
<td>-3</td>
<td>-3</td>
</tr>
<tr>
<td>-3</td>
<td>-4</td>
<td>-4</td>
<td>-4</td>
</tr>
<tr>
<td>-3</td>
<td>-4</td>
<td>-5</td>
<td>-5</td>
</tr>
<tr>
<td>-3</td>
<td>-4</td>
<td>-5</td>
<td>-6</td>
</tr>
</tbody>
</table>
**Value Iteration**

- Problem: find optimal policy $\pi$
- Solution: iterative application of Bellman optimality backup
  - $v_1 \to v_2 \to \ldots \to v_*$
- 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)

\[ v_{k+1}(s) = \max_{a \in A} \left( R_s + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s') \right) \]

\[ v_{k+1} = \max_{a \in A} R^a + \gamma P^a v_k \]
### Synchronous Dynamic Programming Algorithms

<table>
<thead>
<tr>
<th>Problem</th>
<th>Bellman Equation</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prediction</td>
<td>Bellman Expectation Equation</td>
<td>Iterative Policy Evaluation</td>
</tr>
<tr>
<td>Control</td>
<td>Bellman Expectation Equation + Greedy Policy Improvement</td>
<td>Policy Iteration</td>
</tr>
<tr>
<td>Control</td>
<td>Bellman Optimality Equation</td>
<td>Value Iteration</td>
</tr>
</tbody>
</table>

- 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

- 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
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 \( S \)

\[
    v_{new}(s) \leftarrow \max_{a \in A} \left( R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_{old}(s') \right)
\]

\( v_{old} \leftarrow v_{new} \)

- In-place value iteration only stores one copy of value function for all \( s \) in \( S \)

\[
    v(s) \leftarrow \max_{a \in A} \left( R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v(s') \right)
\]
Prioritised Sweeping

- Use magnitude of Bellman error to guide state selection, e.g.
  \[
  \max_{a \in A} \left( R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \nu(s') \right) - \nu(s)
  \]

- 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

- 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 $S_t$

$$v(S_t) \leftarrow \max_{a \in A} \left( R_{S_t}^a + \gamma \sum_{s' \in S} P_{S_tS_t'}^a v(s') \right)$$
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
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 \( \hat{v}(s, w) \)
- Apply dynamic programming to \( \hat{v}(\cdot, w) \)
- e.g. Fitted Value Iteration repeats at each iteration \( k \),
  - Sample states \( \tilde{S} \subseteq S \)
  - For each state \( s \in \tilde{S} \), estimate target value using Bellman optimality equation,

\[
\tilde{v}_k(s) = \max_{a \in A} \left( R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \hat{v}(s', w_k) \right)
\]

- Train next value function \( \hat{v}(\cdot, w_{k+1}) \) using targets \( \{\langle s, \tilde{v}_k(s) \rangle\} \)
The fundamental theorem and the Bellman (optimality) operator

Theorem

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

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

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

A shorter way to write this is

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

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

Definition (Policy evaluation operator)
Let $\pi$ be a stochastic stationary policy. Define

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

Corollary

$T^\pi$ is a contraction, and $V^\pi$ is the unique fixed point of $T^\pi$. 

Greedy policy

Definition (Greedy policy)
Policy $\pi$ is greedy w.r.t. $V$ if

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

or

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

holds for all states $x$. 
A restatement of the main theorem

**Theorem**

Assume that $|A| < +\infty$. 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 $\pi$ is a policy such that

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

then $\pi$ is optimal. Here,

$$T^* Q(x, a) = r(x, a) + \gamma \sum_{y \in \mathcal{X}} \mathcal{P}(x, a, y) \max_{a' \in A} Q(y, a'), \quad x \in \mathcal{X}, a \in \mathcal{A}.$$
Finding the action-value functions of policies

**Theorem**

Let $\pi$ be a stationary policy, $T^\pi$ be defined by

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

Then $Q^\pi$ is the unique solution of

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

Note
- If \( v_i \) is the value-function computed in the \( i^{th} \) iteration of value iteration then
  \[ v_{i+1} = T^* v_i. \]

- 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 \( Q_{i+1} = T^* Q_i \), or value functions with post-decision states. What is the advantage?
Policy iteration

function POLICYITERATION(\(\pi\))
1: repeat
2: \(\pi' \leftarrow \pi\)
3: \(V \leftarrow \text{GETVALUEFUNCTION}(\pi')\)
4: \(\pi \leftarrow \text{GETGREEDYPOLICY}(V)\)
5: until \(\pi \neq \pi'\)
6: return \(\pi\)
What if we stop early?

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

Fix an action-value function $Q$ and let $\pi$ be a greedy policy w.r.t. $Q$. Then the value of policy $\pi$ 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 $v_{\pi}$?
- 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
Consider the vector space $V$ over value functions

- There are $|S|$ 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
We will measure distance between state-value functions $u$ and $v$ by the $\infty$-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 $\hat{v}(s, w)$
- Apply dynamic programming to $\hat{v}(\cdot, w)$
- e.g. Fitted Value Iteration repeats at each iteration $k$,
  - Sample states $\tilde{S} \subseteq S$
  - For each state $s \in \tilde{S}$, estimate target value using Bellman optimality equation,
    \[
    \hat{v}_k(s) = \max_{a \in A} \left( R^a_s + \gamma \sum_{s' \in S} P^a_{ss'} \hat{v}(s', w_k) \right)
    \]
- Train next value function $\hat{v}(\cdot, w_{k+1})$ using targets \{$(s, \hat{v}_k(s))$\}
Theorem (Contraction Mapping Theorem)

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

- $T$ converges to a unique fixed point
- At a linear convergence rate of $\gamma$
Bellman Operator is a Contraction

\[ \| V-V' \| = \text{Infinity norm} \]

(find max diff)

Over all states)

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

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

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

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

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

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

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

\[ = \gamma \| V - V' \| \]
The Bellman expectation operator $T^\pi$ has a unique fixed point $v^\pi$ is a fixed point of $T^\pi$ (by Bellman expectation equation).

By contraction mapping theorem,

Iterative policy evaluation converges on $v^\pi$.

Policy iteration converges on $v^*$.
Define the *Bellman optimality backup operator* $T^*$,

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

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

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