Week 2: Markov Decision Process

Bolei Zhou

UCLA

January 13, 2026

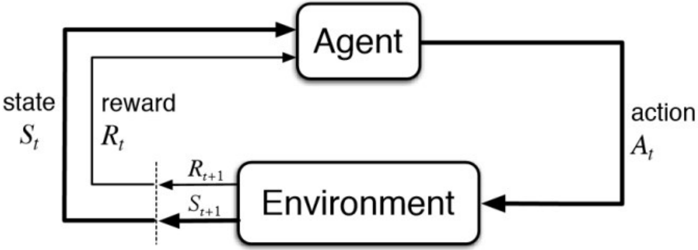
Announcement

1. Our Google Cloud Education Credits are approved.
    1. Each enrolled student will give $50 credit to use.
    2. Instruction will be sent out next week
    3. TAs will give a tutorial in one of the discussions later
2. Assignment 1 will due by the end of next week

Plan

1. Last Week
    1. Course overview
2. Basic components of RL: reward, policy, value function, action-value function, model
3. A simplified RL task: k-armed Bandit Problem
4. Markov Decision Process (MDP)
    1. Markov Chain$\rightarrow$ Markov Reward Process (MRP)$\rightarrow$ Markov Decision Processes (MDP)
    2. Policy evaluation in MDP
5. Control in MDP: policy iteration and value iteration
    1. Improving dynamic programming
6. Textbook of Sutton and Barto: Chapter 3 and Chapter 4

Reinforcement Learning: sequential decision making

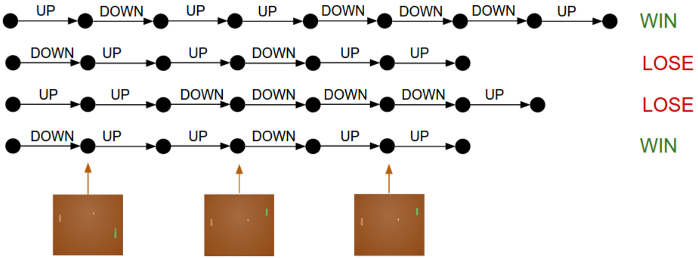


Reward

1. A reward is a scalar feedback signal
2. Reward indicates how well agent is doing at step t
3. The objective of RL is to maximize the cumulative reward
    1. Cumulative reward: $G_t = \sum_k^{\infty} R_{t+k+1}$
    2. Expected cumulative reward: $E_\pi[G_t]$
4. Example: roll a dice five times and receive the face number as the reward, what is the expected cumulative reward at the beginning? or before the third time?

Sequential Decision Making

1. Objective of the agent: select a series of actions to maximize the cumulative reward
2. Actions may have long-term consequences
   1. Reward may be delayed
   2. Trade-off between immediate reward and long-term reward



Major components of an RL agent

1. An RL agent may include one or more of these components:
    1. Policy $\pi(s)$: agent's behavior function
    2. Value function V(s): how good is each state or action
    3. Action-value function Q(s,a): how good is an action in a certain state
    4. Model: agent's state representation of the environment

Policy

1. A policy is the agent's behavior model
2. It is a map function from state/observation to action
3. Stochastic policy: action probability $\pi(a|s) = P[A_t = a|S_t = s]$
4. Deterministic policy: action $a_t^* = \arg\max_a Q(s_t, a)$

Value function

1. Value function: expected discounted sum of future rewards under a particular policy π
    1. It is used to quantify the goodness of states if following a particular policy π

$$v_\pi(s) = E_\pi[G_t|S_t = s] = E_\pi[\sum_k \gamma^k R_{t+k+1}|S_t = s] \tag{1}$$

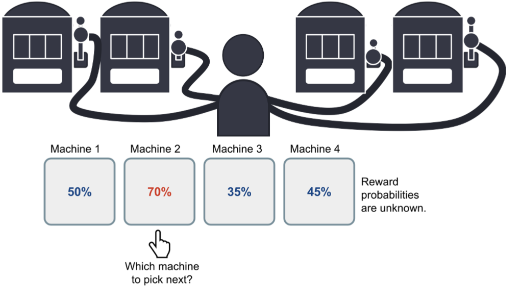
2. Action-value function, or Q function

$$Q(s,a) = E_\pi[G_t|S_t = s, A_t = a] = E_\pi[\sum_k \gamma^k R_{t+k+1}|S_t = s, A_t = a] \tag{2}$$

Model

1. A model predicts what the environment will do next
   1. Predict the next state: $P[S_{t+1} = s' | S_t = s, A_t = a]$
   2. Predict the next reward: $E[R_{t+1} | S_t = s, A_t = a]$
2. Sometimes the world model is given, like Newtonian force $F = ma$

k-Armed Bandit Problem as a simplified RL example



1. A multi-armed bandit is a tuple $< \mathcal{A}, \mathcal{R} >$
2. k actions to take at each step t
3. $\mathcal{R}^a(r) = P(r|a)$ is unknown probability distribution over rewards
4. At each step t the agent selects an action $a_t \in \mathcal{A}$, then the environment generates a reward $r_t \sim \mathcal{R}^{a_t}$
5. The goal of agent is to maximize cumulative reward $\sum_{\tau=1}^{T} r_\tau$

```
class BernoulliArm():
  def __init__(self, p):
    self.p = p
  def draw(self):
    if random.random() > self.p:
      return 0.0
    else:
      return 1.0

class NormalArm():
  def __init__(self, mu, sigma):
    self.mu = mu
    self.sigma = sigma
  def draw(self):
    return random.gauss(self.mu, self.sigma)
```

Definition of Value Function and Action-Value Function

1. The action-value is the mean reward for action a

$$Q(a) = \mathbb{E}(r|a) \tag{3}$$

2. The optimal value

$$V^* = Q(a^*) = \max_{a \in \mathcal{A}} Q(a) \tag{4}$$

3. To estimate $Q(a)$, we can compute $Q_t(a)$ at step t as

$$Q_t(a) = \frac{\text{sum of rewards when a taken prior to t}}{\text{number of times a taken prior to t}} = \frac{\sum_{i=1}^{t-1} r_i \cdot 1_{A_i=a}}{\sum_{i=1}^{t-1} 1_{A_i=a}} \tag{5}$$

Greedy Action and ϵ-Greedy Action to Take

1. The estimation of $Q(a)$ at step t

$$Q_t(a) = \frac{\text{sum of rewards when a taken prior to t}}{\text{number of times a taken prior to t}} = \frac{\sum_{i=1}^{t-1} r_i \cdot 1_{A_i=a}}{\sum_{i=1}^{t-1} 1_{A_i=a}} \tag{6}$$

2. Greedy action selection algorithm: $A_t = \arg\max_a Q_t(a)$
3. Problem with the greedy algorithm?
4. ϵ-Greedy: greedy most of the time, but with small probability ϵ select random actions (ϵ is usually as 0.1)
    1. probability $1 - \epsilon : A_t = \arg\max_a Q_t(a)$
    2. probability $\epsilon : A_t = \mathit{uniform}(\mathcal{A})$

ϵ-Greedy Algorithm

Algorithm 1 A simple *epsilon*-Greedy bandit algorithm

for a = 1 to k **do**
    $Q(a) = 0, N(a) = 0$
end for
loop
    $A = \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\epsilon \\ uniform(\mathcal{A}) & \text{with probability } \epsilon \end{cases}$
    $r = bandit(A)$
    $N(A) = N(A) + 1$
    $Q(A) = Q(A) + \frac{1}{N(A)}[r - Q(A)]$
end loop

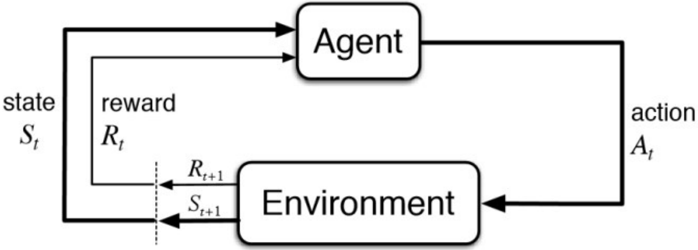
Deriving: NewEstimate = OldEstimate + StepSize[Target - OldEstimate]

$$Q_t(a_t) = Q_{t-1} + \frac{1}{N_t(a_t)}(r_t - Q_{t-1}(a_t)) \tag{7}$$

Why it is a simplified RL task

1. no delayed reward: instance feedback
2. no change of state: only one state
3. independent of consecutive behaviors

Markov Decision Process (MDP)



1. Markov Decision Process can model a lot of real-world problems. It formally describes the framework of reinforcement learning
2. Under MDP, the environment is fully observable.
    1. Optimal control primarily deals with continuous MDPs
    2. Partially observable problems can be converted into MDPs

Defining Three Markov Models

- Markov Processes
- Markov Reward Processes (MRPs)
- Markov Decision Processes (MDPs)

Markov Property

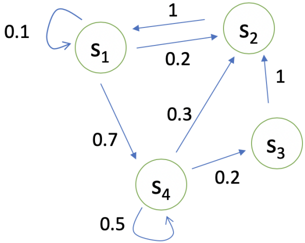
1. The history of states: $h_t = \{s_1, s_2, s_3, ..., s_t\}$
2. State s_t is Markovian if and only if:

$$p(s_{t+1}|s_t) = p(s_{t+1}|h_t) \tag{8}$$

$$p(s_{t+1}|s_t, a_t) = p(s_{t+1}|h_t, a_t) \tag{9}$$

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

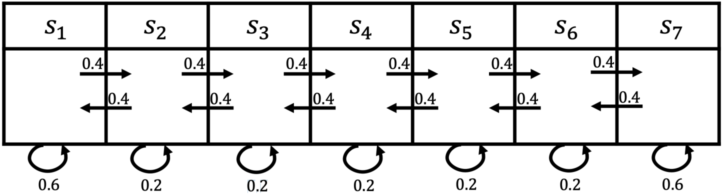
Markov Process/Markov Chain



1. State transition matrix P specifies $p(s_{t+1} = s' | s_t = s)$

$$P = \begin{bmatrix} P(s_1|s_1) & P(s_2|s_1) & \dots & P(s_N|s_1) \\ P(s_1|s_2) & P(s_2|s_2) & \dots & P(s_N|s_2) \\ \vdots & \vdots & \ddots & \vdots \\ P(s_1|s_N) & P(s_2|s_N) & \dots & P(s_N|s_N) \end{bmatrix}$$

Example of MP

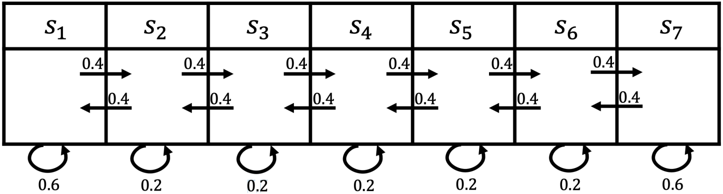


1. Sample episodes starting from s_3
   1. s_3, s_4, s_5, s_6, s_6
   2. s_3, s_2, s_3, s_2, s_1
   3. s_3, s_4, s_4, s_5, s_5

Markov Reward Process (MRP)

1. Markov Reward Process is a Markov Chain + reward
2. Definition of Markov Reward Process (MRP)
   1. S is a (finite) set of states ($s \in S$)
   2. P is dynamics/transition model that specifies $P(S_{t+1} = s' | s_t = s)$
   3. R is a reward function $R(s_t = s) = \mathbb{E}[r_t | s_t = s]$
   4. Discount factor $\gamma \in [0, 1]$
3. If finite number of states, R can be a vector
   1. We focus on tabular representation first
   2. What happen if there are infinite number of states?

Example of MRP



Reward: $+5$ in s_1, $+10$ in s_7, 0 in all other states. So that we can represent $R = [5, 0, 0, 0, 0, 0, 10]$

Return and Value Function

1. Definition of Horizon
   1. Number of maximum time steps in each episode/trajectory
   2. Can be infinite, otherwise called finite Markov (reward) Process
   3. Per game: 100 moves for Go, 80 moves for chess
2. Definition of Return
   1. Discounted sum of rewards from time step t to horizon

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + ... + \gamma^{T-t-1} R_T$$

3. Definition of state value function $V_t(s)$ for a MRP
   1. Expected return from t in state s

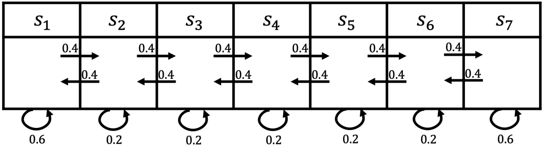
$$\begin{aligned} V_t(s) =& \mathbb{E}[G_t | s_t = s] \\ =& \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... + \gamma^{T-t-1} R_T | s_t = s] \end{aligned}$$

   2. Present value of accumulated future rewards

Why Discount Factor γ

1. Avoid infinite returns in cyclic Markov processes
2. Uncertainty about the future
    1. If the reward is financial, immediate rewards may earn more interest than delayed rewards
3. Animal/human behavior shows a preference for immediate reward
4. It is sometimes possible to use undiscounted Markov reward processes (i.e. $\gamma = 1$), e.g. if all sequences terminate.
    1. $\gamma = 0$: Only care about the immediate reward
    2. $\gamma = 1$: Future reward is equal to the immediate reward.

Example of MRP



1. Reward: $+5$ in s_1, $+10$ in s_7, 0 in all other states. So that we can represent $R = [5, 0, 0, 0, 0, 0, 10]$
2. Sample returns G for a 3-step episodes with $\gamma = 1/2$
   1. return for $s_4, s_5, s_6, s_7 : 0 + \frac{1}{2} \times 0 + \frac{1}{4} \times 10 = 2.5$
   2. return for $s_4, s_3, s_2, s_1 : 0 + \frac{1}{2} \times 0 + \frac{1}{4} \times 5 = 1.25$
   3. return for s_4, s_5, s_6, s_6: 0
3. How to compute the value function? For example, the value of state s_4 as $V(s_4) = \mathbb{E}[G_t | s_t = s_4]$

Computing the Value of a Markov Reward Process

1. Value function: expected return from starting in state s

$$V(s) = \mathbb{E}[G_t | s_t = s] = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... | s_t = s]$$

2. MRP value function satisfies the following **Bellman equation**:

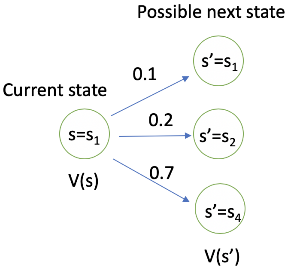
$$V(s) = \underbrace{R(s)}_{\text{Immediate reward}} + \underbrace{\gamma \sum_{s' \in S} P(s'|s) V(s')}_{\text{Discounted sum of future reward}}$$

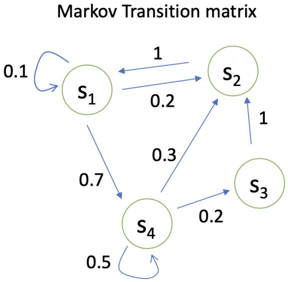
3. Practice: To derive the Bellman equation from the definition of V(s)
   1. Hint: $V(s) = \mathbb{E}[R_{t+1} + \gamma \mathbb{E}[R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + ...] | s_t = s]$

Understanding Bellman Equation

1. **Bellman equation** describes the iterative relations of states

$$V(s) = R(s) + \gamma \sum_{s' \in S} P(s'|s)V(s')$$





Matrix Form of Bellman Equation for MRP

Therefore, we can express V(s) using the matrix form:

$$\begin{bmatrix} V(s_1) \\ V(s_2) \\ \vdots \\ V(s_N) \end{bmatrix} = \begin{bmatrix} R(s_1) \\ R(s_2) \\ \vdots \\ R(s_N) \end{bmatrix} + \gamma \begin{bmatrix} P(s_1|s_1) & P(s_2|s_1) & \dots & P(s_N|s_1) \\ P(s_1|s_2) & P(s_2|s_2) & \dots & P(s_N|s_2) \\ \vdots & \vdots & \ddots & \vdots \\ P(s_1|s_N) & P(s_2|s_N) & \dots & P(s_N|s_N) \end{bmatrix} \begin{bmatrix} V(s_1) \\ V(s_2) \\ \vdots \\ V(s_N) \end{bmatrix}$$

$V = R + \gamma PV$

1. Analytic solution for value of MRP: $V = (I - \gamma P)^{-1} R$
    1. Matrix inverse takes the complexity $O(N^3)$ for N states
    2. Only possible for small MRPs
2. Other methods to solve this?

Iterative Algorithm for Computing Value of a MRP

1. Dynamic Programming
2. Monte-Carlo evaluation
3. Temporal-Difference learning

Monte Carlo Algorithm for Computing Value of a MRP

Algorithm 2 Monte Carlo simulation to calculate MRP value function

$i \leftarrow 0, G_t \leftarrow 0$
while $i \neq N$ **do**
  generate an episode, starting from state s and time t
  Using the generated episode, calculate return $g = \sum_{i=t}^{H-1} \gamma^{i-t} r_i$
  $G_t \leftarrow G_t + g, i \leftarrow i + 1$
end while
$V_t(s) \leftarrow G_t / N$

1. For example: to calculate $V(s_4)$ we can generate a lot of trajectories then take the average of the returns:
   1. return for $s_4, s_5, s_6, s_7 : 0 + \frac{1}{2} \times 0 + \frac{1}{4} \times 10 = 2.5$
   2. return for $s_4, s_3, s_2, s_1 : 0 + \frac{1}{2} \times 0 + \frac{1}{4} \times 5 = 1.25$
   3. return for s_4, s_5, s_6, s_6: 0
   4. more trajectories

Iterative Algorithm for Computing Value of a MRP

Algorithm 3 Iterative algorithm to calculate MRP value function

for all states $s \in S, V'(s) \leftarrow 0, V(s) \leftarrow \infty$
while $||V - V'|| > \epsilon$ **do**
    $V \leftarrow V'$
    For all states $s \in S, V'(s) = R(s) + \gamma \sum_{s' \in S} P(s'|s)V(s')$
end while
return $V'(s)$ for all $s \in S$

Markov Decision Process (MDP)

1. Markov Decision Process is Markov Reward Process with decisions.
2. Definition of MDP
   1. S is a finite set of states
   2. A is a finite set of actions
   3. P is dynamics/transition model for each action a under certain state s as $P(s_{t+1} = s' | s_t = s, a_t = a)$
   4. R is a reward function $R(s_t = s, a_t = a) = \mathbb{E}[r_t | s_t = s, a_t = a]$
   5. Discount factor $\gamma \in [0, 1]$
3. MDP is a tuple: (S, A, P, R, γ)

Policy in MDP

1. Policy specifies what action to take in each state
2. Given a state, specify a distribution over actions
3. Policy: $\pi(a|s) = P(a_t = a|s_t = s)$
4. Policies are stationary (time-independent), $A_t \sim \pi(a|s)$ for any $t > 0$

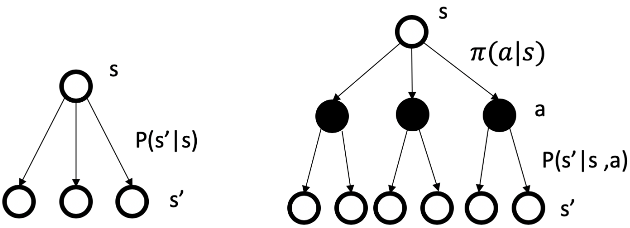
Policy in MDP

1. Given a MDP (S, A, P, R, γ) and a policy π
2. The state and reward sequence $S_1, R_2, S_2, R_2, ...$ is a Markov reward process $(S, P^\pi, R^\pi, \gamma)$ where,

$$P^\pi(s'|s) = \sum_{a \in A} \pi(a|s) P(s'|s, a)$$

$$R^\pi(s) = \sum_{a \in A} \pi(a|s) R(s, a)$$

Comparison of MP/MRP and MDP



Value function for MDP

1. The state-value function $v^\pi(s)$ of an MDP is the expected return starting from state s, and following policy π

$$v^\pi(s) = \mathbb{E}_\pi[G_t | s_t = s] \tag{10}$$

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

$$q^\pi(s, a) = \mathbb{E}_\pi[G_t | s_t = s, A_t = a] \tag{11}$$

3. We have the relation between $v^\pi(s)$ and $q^\pi(s, a)$

$$v^\pi(s) = \sum_{a \in A} \pi(a|s) q^\pi(s, a) \tag{12}$$

Bellman Expectation Equation

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

$$v^{\pi}(s) = E_{\pi}[R_{t+1} + \gamma v^{\pi}(s_{t+1})|s_t = s] \tag{13}$$

2. The action-value function can similarly be decomposed

$$q^{\pi}(s, a) = E_{\pi}[R_{t+1} + \gamma q^{\pi}(s_{t+1}, A_{t+1})|s_t = s, A_t = a] \tag{14}$$

Bellman Expectation Equation for V^π and Q^π

$$v^\pi(s) = \sum_{a\in A} \pi(a|s) q^\pi(s,a) \tag{15}$$

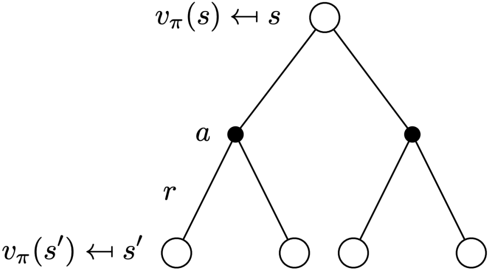
$$q^\pi(s,a) = R_s^a + \gamma \sum_{s'\in S} P(s'|s,a) v^\pi(s') \tag{16}$$

Thus

$$v^\pi(s) = \sum_{a\in A} \pi(a|s)\Big(R(s,a) + \gamma \sum_{s'\in S} P(s'|s,a) v^\pi(s')\Big) \tag{17}$$

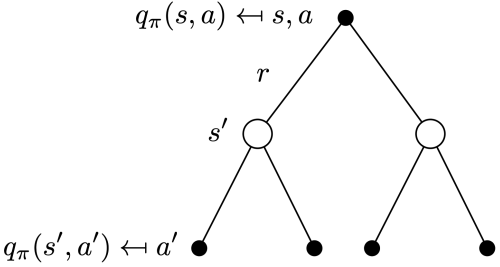
$$q^\pi(s,a) = R(s,a) + \gamma \sum_{s'\in S} P(s'|s,a) \sum_{a'\in A} \pi(a'|s') q^\pi(s',a') \tag{18}$$

Backup Diagram for V^π



$$v^\pi(s) = \sum_{a \in A} \pi(a|s)(R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)v^\pi(s')) \tag{19}$$

Backup Diagram for Q^π



$$q^\pi(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \sum_{a' \in A} \pi(a'|s') q^\pi(s',a') \quad (20)$$

Policy Evaluation

1. Evaluate the value of state given a policy π: compute $v^{\pi}(s)$
2. Also called as (value) prediction

Example: Navigate the boat

Figure: Markov Chain/MRP: Go with river stream

Figure: MDP: Navigate the boat

Example: Policy Evaluation

s_1	s_2	s_3	s_4	s_5	s_6	s_7
						

1. Two actions: *Left* or *Right*
2. For all actions, reward: $+5$ in s_1, $+10$ in s_7, 0 in all other states. So that we can represent $R = [5, 0, 0, 0, 0, 0, 10]$
3. Let's have a deterministic policy $\pi(s) = Left$ and $\gamma = 0$ for any state s, then what is the value of the policy?
    1. $V^\pi = [5, 0, 0, 0, 0, 0, 10]$ since $\gamma = 0$

Example: Policy Evaluation

s_1	s_2	s_3	s_4	s_5	s_6	s_7
						

1. $R = [5, 0, 0, 0, 0, 0, 10]$
2. Practice 1: Deterministic policy $\pi(s) = Left$ with $\gamma = 0.5$ for any state s, then what are the state values under the policy?
3. Practice 2: Stochastic policy $P(\pi(s) = Left) = 0.5$ and $P(\pi(s) = Right) = 0.5$ and $\gamma = 0.5$ for any state s, then what are the state values under the policy?
4. Iteration t:
$v_t^\pi(s) = \sum_a P(\pi(s) = a)(r(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v_{t-1}^\pi(s'))$

Decision Making in Markov Decision Process (MDP)

1. Prediction (evaluate a given policy):
   1. Input: MDP $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$ and policy π or MRP $< \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma >$
   2. Output: value function v^π
2. Control (search the optimal policy):
   1. Input: MDP $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$
   2. Output: optimal value function v^* and optimal policy π^*
3. Prediction and control in MDP can be solved by dynamic programming.

Dynamic Programming

Dynamic Programming is a very general solution method for problems which have two properties:

1. Optimal substructure
    1. Principle of optimality applies
    2. Optimal solution can be decomposed into subproblems
2. Overlapping subproblems
    1. Subproblems recur many times
    2. Solutions can be cached and reused

Markov decision processes satisfy both properties

1. Bellman equation gives recursive decomposition
2. Value function stores and reuses solutions

Prediction: Policy evaluation on MDP

1. Objective: Evaluate a given policy π for a MDP
2. Output: the value function under policy v^{π}
3. Solution: iteration on Bellman expectation backup
4. Algorithm: Synchronous backup
   1. At each iteration t+1
      update $v_{t+1}(s)$ from $v_t(s')$ for all states $s \in \mathcal{S}$ where s' is a successor state of s

$$v_{t+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s)(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v_t(s')) \tag{21}$$

5. Convergence: $v_1 \rightarrow v_2 \rightarrow ... \rightarrow v^{\pi}$

Policy evaluation: Iteration on Bellman expectation backup

Bellman expectation backup for a particular policy

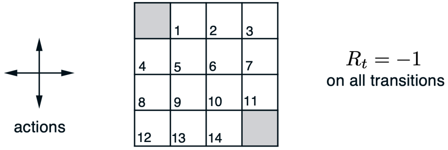
$$v_{t+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s)(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v_t(s')) \tag{22}$$

Or if in the form of MRP $< \mathcal{S}, \mathcal{P}^{\pi}, \mathcal{R}, \gamma >$

$$v_{t+1}(s) = R^{\pi}(s) + \gamma \sum_{s' \in \mathcal{S}} P^{\pi}(s'|s) v_t(s') \tag{23}$$

Evaluating a Random Policy in the Small Gridworld

Example 4.1 in the Sutton RL textbook.



1. Undiscounted episodic MDP ($\gamma = 1$)
2. Nonterminal states $1, ..., 14$
3. Two terminal states (two shaded squares)
4. Action leading out of grid leaves state unchanged, $P(7|7, right) = 1$
5. Reward is -1 until the terminal state is reach
6. Transition is deterministic given the action, e.g., $P(6|5, right) = 1$
7. Uniform random policy $\pi(l|.) = \pi(r|.) = \pi(u|.) = \pi(d|.) = 0.25$

Evaluating a Random Policy in the Small Gridworld

1. Iteratively evaluate the random policy

v_k for the Random Policy

$k = 0$

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

$k = 1$

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

$k = 2$

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

$k = 3$

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

$k = 10$

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

$k = \infty$

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

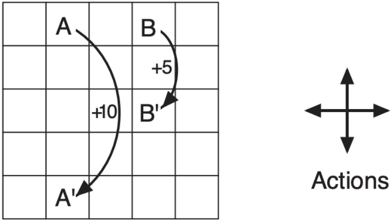
A live demo on policy evaluation

$$v^{\pi}(s) = \sum_{a \in A} \pi(a|s)(R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v^{\pi}(s')) \tag{24}$$

1. `https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html`

Practice: Gridworld

Textbook Example 3.5:GridWorld



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

MDP Control

1. Compute the optimal policy

$$\pi^*(s) = \arg\max_{\pi} v^{\pi}(s) \tag{25}$$

2. Optimal policy for a MDP in an infinite horizon problem (agent acts forever) is
   1. Deterministic
   2. Stationary (does not depend on time step)
   3. Unique? Not necessarily, may have state-actions with identical optimal values

Optimal Value Function

1. The optimal state-value function $v^*(s)$ is the maximum value function over all policies

$$v^*(s) = \max_{\pi} v^{\pi}(s)$$

2. The optimal policy

$$\pi^*(s) = \arg\max_{\pi} v^{\pi}(s)$$

3. An MDP is "solved" when we know the optimal value
4. There exists a unique optimal value function, but could be multiple optimal policies (two actions that have the same optimal value function)

Finding Optimal Policy

1. An optimal policy can be found by maximizing over $q^*(s, a)$,

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

2. There is always a deterministic optimal policy for any MDP
3. If we know $q^*(s, a)$, we immediately have the optimal policy

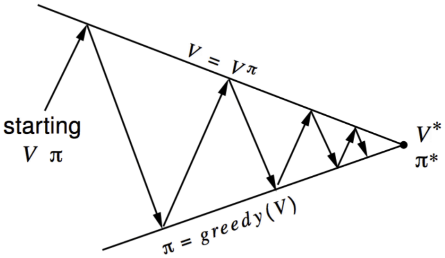
Policy Search

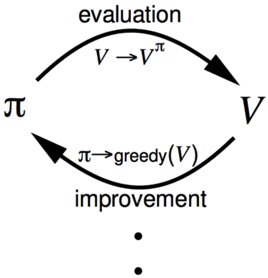
1. One option is to enumerate search the best policy
2. Number of deterministic policies is $|\mathcal{A}|^{|\mathcal{S}|}$
3. Other approaches such as policy iteration and value iteration are more efficient
    1. Policy iteration
    2. Value iteration

Improving a Policy through Policy Iteration

1. Iterate through the two steps:
   1. Evaluate the policy π (computing v given current π)
   2. Improve the policy by acting greedily with respect to v^π

$$\pi' = \text{greedy}(v^\pi) \tag{26}$$





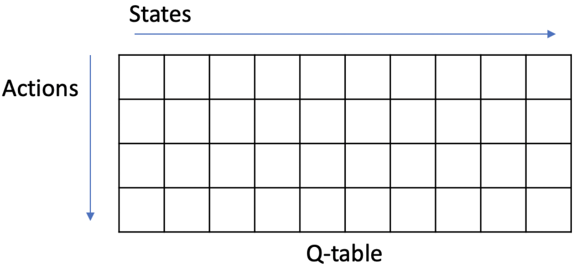
Policy Improvement

1. Compute the state-action value of a policy π:

$$q^{\pi_i}(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v^{\pi_i}(s') \tag{27}$$

2. Compute new policy π_{i+1} for all $s \in \mathcal{S}$ following

$$\pi_{i+1}(s) = \arg\max_a q^{\pi_i}(s,a) \tag{28}$$



Monotonic Improvement in Policy

1. Consider a determinisitc policy $a = \pi(s)$
2. We improve the policy through

$$\pi'(s) = \arg\max_a q^\pi(s, a)$$

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

$$q^\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q^\pi(s, a) \geq q^\pi(s, \pi(s)) = v^\pi(s)$$

4. It therefore improves the value function, $v^{\pi'}(s) \geq v^\pi(s)$

$$\begin{aligned} v^\pi(s) \leq & q^\pi(s, \pi'(s)) = \mathbb{E}_{\pi'}[R_{t+1} + \gamma v^\pi(S_{t+1}|S_t = s)] \\ \leq & \mathbb{E}_{\pi'}[R_{t+1} + \gamma q^\pi(S_{t+1}, \pi'(S_{t+1}))|S_t = s] \\ \leq & \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 q^\pi(S_{t+2}, \pi'(S_{t+2}))|S_t = s] \\ \leq & \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + ...|S_t = s] = v^{\pi'}(s) \end{aligned}$$

Monotonic Improvement in Policy

1. If improvements stop,

$$q^{\pi}(s, \pi'(s)) = \max_{a \in \mathcal{A}} q^{\pi}(s, a) = q^{\pi}(s, \pi(s)) = v^{\pi}(s)$$

2. Thus the Bellman optimality equation has been satisfied

$$v^{\pi}(s) = \max_{a \in \mathcal{A}} q^{\pi}(s, a)$$

3. Therefore $v^{\pi}(s) = v^{*}(s)$ for all $s \in \mathcal{S}$, so π is an optimal policy

Bellman Optimality Equation

1. The optimal value functions are reached by the Bellman optimality equations:

$$v^*(s) = \max_a q^*(s, a)$$
$$q^*(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) v^*(s')$$

thus

$$v^*(s) = \max_a R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) v^*(s')$$
$$q^*(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) \max_{a'} q^*(s', a')$$

Value Iteration by turning the Bellman Optimality Equation as update rule

1. If we know the solution to subproblem $v^*(s')$, which is optimal.
2. Then the solution for the optimal $v^*(s)$ can be found by iteration over the following Bellman Optimality backup rule,

$$v(s) \leftarrow \max_{a \in \mathcal{A}} \Big(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v(s') \Big)$$

3. The idea of value iteration is to apply these updates iteratively

Algorithm of Value Iteration

1. Objective: find the optimal policy π
2. Solution: iteration on the Bellman optimality backup
3. Value Iteration algorithm:
    1. initialize $k = 1$ and $v_0(s) = 0$ for all states s
    2. For $k = 1 : H$
        1. for each state s

$$q_{k+1}(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v_k(s') \tag{29}$$

$$v_{k+1}(s) = \max_a q_{k+1}(s,a) \tag{30}$$

        2. $k \leftarrow k + 1$
    3. To retrieve the optimal policy after the value iteration:

$$\pi(s) = \arg\max_a R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v_{k+1}(s') \tag{31}$$

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

After the optimal values are reached, we run policy extraction to retrieve the optimal policy.

Difference between Policy Iteration and Value Iteration

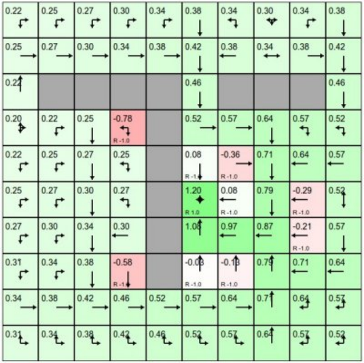
1. Policy iteration includes: **policy evaluation** + **policy improvement**, and the two are repeated iteratively until policy converges.
2. Value iteration includes: **finding optimal value function** + **one policy extraction**. There is no repeat of the two because once the value function is optimal, then the policy out of it should also be optimal (i.e. converged).
3. Finding optimal value function can also be seen as a combination of policy improvement (due to max) and truncated policy evaluation (the reassignment of v(s) after just one sweep of all states regardless of convergence).

Summary for Prediction and Control in MDP

Table: Dynamic Programming Algorithms

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

Demo of policy iteration and value iteration



1. Policy iteration: Iteration of policy evaluation and policy improvement(update)
2. Value iteration
3. https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html

Policy iteration and value iteration on FrozenLake

1. `https://github.com/ucla-rlcourse/RLexample/tree/master/MDP`

Improving Dynamic Programming

1. A major drawback to the DP methods is that they involve operations over the entire state set of the MDP, that is, they require sweeps of the state set.
2. If the state set is very large, for example, the game of backgammon has over 10^{20} states. Thousands of years to be taken to finish one sweep.
3. Asychronous DP algorithms are in-place iterative DP that are not organized in terms of systematic sweeps of the state set
4. The values of some states may be updated several times before the values of others are updated once.

Improving Dynamic Programming

Synchronoous dynamic programming is usually slow. Three simple ideas to extend DP for asynchronous dynamic programming:

1. In-place dynamic programming
2. Prioritized sweeping
3. Real-time dynamic programming

In-Places Dynamic Programming

1. Synchronous value iteration stores two copies of value function:
   for all s in $\mathcal{S}$
   $$v_{new}(s) \leftarrow \max_{a \in \mathcal{A}} \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v_{old}(s') \right)$$
   $v_{old} \leftarrow v_{new}$
2. In-place value iteration only stores one copy of value function:
   for all s in $\mathcal{S}$
   $$v(s) \leftarrow \max_{a \in \mathcal{A}} \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v(s') \right)$$

Prioritized Sweeping

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

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

2. Backup the state with the largest remaining Bellman error
3. Update Bellman error of affected states after each backup
4. Can be implemented efficiently by maintaining a priority queue

Real-Time Dynamic Programming

1. To solve a given MDP, we can run an iterative DP algorithm at the same time that an agent is actually experiencing the MDP
2. The agent's experience can be used to determine the states to which the DP algorithm applies its updates
3. We can apply updates to states as the agent visits them. So focus on the parts of the state set that are most relevant to the agent
4. After each time-step S_t, A_t, backup the state S_t,

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

Sample Backups

1. The key design for RL algorithms such as Q-learning and SARSA in next lectures
2. Using sample rewards and sample transition pairs $< S, A, R, S' >$, rather than the reward function $\mathcal{R}$ and transition dynamics $\mathcal{P}$
3. Benefits:
    1. Model-free: no advance knowledge of MDP required
    2. Break the curse of dimensionality through sampling
    3. Cost of backup is constant, independent of $n = |\mathcal{S}|$

Approximate Dynamic Programming

1. Using a function approximator $\hat{v}(s, \mathbf{w})$
2. Fitted value iteration repeats at each iteration k,
   1. Sample state s from the state cache $\tilde{\mathcal{S}}$

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

   2. Train next value function $\hat{v}(s', \mathbf{w}_{k+1})$ using targets $< s, \tilde{v}_k(s) >$.
3. Key idea behind the Deep Q-Learning

End

1. Summary: MDP, policy evaluation, policy iteration, and value iteration
2. Check out the code example of MDP at `https://github.com/ucla-rlcourse/RLexample/tree/master/MDP`
3. In this week's discussion session, TAs will give you more examples
4. Next Week: Model-free methods
5. Reading: Textbook Chapter 5 and Chapter 6