Outline

• Markov chains
• Markov Decision Processes
• Policies and value functions
  – Policy evaluation
  – Policy improvement
• Computing optimal value functions for MDPs
  – Policy iteration algorithm
  – Value iteration algorithm
• Function approximation
Sequential decision-making

• Utility theory provides a foundation for one-shot decisions.
  – If more than one decision has to be taken, reasoning about all of
    them in general is very expensive.

• Agents need to be able to make decisions in a repeated
  interaction with the environment over time, where the effects of
  one decision affect the next one.

• Markov Decision Processes (MDPs) provide a framework for
  modeling sequential decision-making.

Markov Chain example

• How the game works:
  – Start at state 1.
  – Roll a die, then move a number of positions given by its value.
  – If you land on square 5, you are teleported to 8.
  – Whomever gets to 12 first wins!

• Note that there is no skill or decision involved (yet!)

• Compared to bandits, here there are many states.
Markov Chain definition

- Set of states $S$

- Transition probabilities: $T: S \times S \rightarrow [0, 1]$
  \[ T(s, s') = P(s_{t+1} = s' | s_t = s) \]

- Initial state distribution: $P_0: S \rightarrow [0, 1]$
  \[ P_0(s) = P(s_0 = s) \]

---

Decision-making

- Suppose that we played the game with two dice.

- You roll both dice and then have a choice:
  - Take the roll from the first die.
  - Take the roll from the second die.
  - Take the sum of the two rolls.

- The goal is to finish the game as quickly as possible.
  - Utility function is assumed to be known (we will relax this later).

- Need to optimize a sequence of decisions.
Sequential decision-making

- At each time step $t$, the agent is in some state $s_t$.
- It chooses an action $a_t$, and as a result, it receives a numerical reward $R_{t+1}$ and it can observe the new state $s_{t+1}$.
- Similar to a Markov chain, but there are also actions and rewards.

Markov Decision Processes (MDPs)

- Set of states $S$
- Set of actions $A$
- Transition model (dynamics): $T: S \times A \times S \rightarrow [0, 1]$
  - $T(s,a,s') = P(s_{t+1} = s' | s_t = s, a_t = a)$ is the probability of going from $s$ to $s'$ under action $a$.
    
    Same as HMM model.
- Reward function $R: S \times A \rightarrow \mathbb{R}$
  - $R(s,a)$ is the short-term utility of the action.
    
    Same as bandits.
- Discount factor, $\gamma$ (between 0 and 1, usually close to 1).
The discount factor

- Two interpretations:
  - At each time step, there is a $1 - \gamma$ chance that the agent dies, and does not receive rewards afterwards.
  - Inflation rate: receiving an amount of money in a year, is worth less than today.

Applications of MDPs

- AI / Computer Science:
  - Robotic control
  - Air campaign planning
  - Elevator control
  - Computation scheduling
  - Control and automation
  - Spoken dialogue management
  - Cellular channel allocation
  - Football play selection
Applications of MDPs

- Economics / Operations Research
  - Inventory management
  - Fleet maintenance
  - Road maintenance
  - Packet retransmission
  - Nuclear plant management

- Agriculture
  - Herd management
  - Fish stock management

Planning in MDPs

- The goal of an agent in an MDP is to be rational. 
  
  *Maximize its expected utility* (i.e. respect MEU principle).

- Maximizing the immediate utility (reward) is not sufficient.
  - E.g. the agent might pick an action that gives instant gratification, even if it later makes it “die”.

- The goal is to maximize long-term utility, (also called return).
  - The return is an additive function of all rewards received by the agent.
Long-term utilities

- The utility $U_t$ for a trajectory, starting from step $t$, is defined as:

- Episodic tasks (e.g. games, trips through a maze, etc.)
  $$U_t = R_t + R_{t+1} + R_{t+2} + \ldots + R_T$$
  where $T$ is the time when a terminal state is reached.

- Continuing tasks (e.g. tasks which may go on forever)
  $$U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k}$$
  Discount factor $\gamma < 1$ ensures that return is finite if rewards are bounded.

Example: Mountain-Car

- **States**: position and velocity
- **Actions**: accelerate forward, accelerate backward, coast
- **Goal**: get the car to the top of the hill as quickly as possible.
- **Reward**: -1 for every time step, until car reaches the top (then 0)
  (Alternately: reward = 1 at the top, 0 otherwise, $\gamma < 1$)
Policies

- The goal of the agent is to find a way of behaving, called a policy (similar to a universal plan, or a strategy) that maximizes the expected value of $U_t, V_t$.

- Two types of policies:
  1. **Deterministic policy**: in each state the agent chooses a unique action.
     $$\pi: S \rightarrow A, \quad \pi(s) = a$$
  2. **Stochastic policy**: in the same state, the agent can “roll a die” and choose different actions.
     $$\pi: S \times A \rightarrow [0, 1], \quad \pi(s, a) = P(a_t=a \mid s_t=s)$$

- Once a policy is fixed, the MDP becomes a Markov chain with rewards.

Value Functions

- Because we want to find a policy that maximizes the utility, it is a good idea to estimate the expected utility.

- We can search through the space of policies for one that is good.

- The value function, $V(s)$, represents the expected utility, for every state, given a certain policy.

- Computing accurate value functions is an intermediate step towards computing good policies.
State Value Function

- The **value function of a policy** $\pi$ is a function: $V^\pi: S \rightarrow \mathbb{R}$

- The **value of state $s$ under policy $\pi$** is the expected return if the agent starts from state $s$ and picks actions according to policy $\pi$.
  
  $$V^\pi(s) = E_{\pi}\left[U_t \mid s_t = s\right]$$

  - For a finite state space, we can represent this as an array, with one entry for every state.
  - We will talk later about methods used for very large or continuous state spaces.

Bellman Equations for Evaluating Policy

- Recall our definition of the return:
  
  $U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \ldots$
  
  $= R_t + \gamma (R_{t+1} + \gamma R_{t+2} + \ldots)$
  
  $= R_t + \gamma U_{t+1}$

- Based on this observation, $V^\pi(s)$ becomes:
  
  $$V^\pi(s) = E_{\pi}\left[U_t \mid s_t = s\right] = E_{\pi}\left[R_t + \gamma U_{t+1} \mid s_t = s\right]$$

- By writing the expectation explicitly, we get:
  
  - Deterministic policy: $V^\pi(s) = \left( R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V^\pi(s') \right)$
  
  - Stochastic policy: $V^\pi(s) = \sum_{a \in A} \pi(s, a) \left( R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^\pi(s') \right)$

  This is a system of linear equations (one per state) with unique solution $V^\pi$. 
Policy evaluation in matrix form

- Bellman’s equation in matrix form:
  \[ V = R + \gamma T V \]

- What are \( V, R \) and \( T \)?
  - \( V \) is a vector containing the value of each state under policy \( \pi \).
  - \( R \) is a vector containing the immediate reward at each state: \( R(s, \pi(s)) \).
  - \( T \) is a matrix containing the transition probability at each state: \( T(s, \pi(s), s') \).

- In some cases, we can solve this exactly:
  \[ V = (I - \gamma T)^{-1} R \]

- Can we do this iteratively?

Iterative Policy Evaluation

Main idea: turn Bellman equations into update rules.

1. Start with some initial guess \( V_0 \).
2. During every iteration \( k \), update the value function for all states:
   \[ V_{k+1}(s) \leftarrow \left( R(s, \pi(s)) + \gamma \sum_{s'} T(s, \pi(s), s') V_k(s') \right) \]
3. Stop when the maximum changes between two iterations is smaller than a desired threshold (the values stop changing.)

This is a bootstrapping idea: the value of one state is updated based on the current estimates of the values of successor states.

This is a dynamic programming algorithm. It’s guaranteed to converge!
Convergence of Iterative Policy Evaluation

- Consider the absolute error in our estimate $V_{k+1}(s)$:

$$|V_{k+1}(s) - V^\pi(s)| = \left| \sum_a \pi(s, a)(R(s, a) + \gamma \sum_{s'} T(s, a, s') V_k(s')) - \sum_{s'} \pi(s, a)(R(s, a) + \gamma \sum_{s'} T(s, a, s') V^\pi(s')) \right|$$

$$= \gamma \left| \sum_a \pi(s, a) \sum_{s'} T(s, a, s') (V_k(s') - V^\pi(s')) \right|$$

$$\leq \gamma \sum_a \pi(s, a) \sum_{s'} T(s, a, s') |V_k(s') - V^\pi(s')|$$

- Let $\varepsilon_k$ be the worst error at iteration $k-1$:

$$\varepsilon_k = \max_{s' \in S} |V_k(s') - V^\pi(s')|$$

- From previous calculation, we have:

$$|V_{k+1}(s) - V^\pi(s)| \leq \gamma \sum_a \pi(s, a) \sum_{s'} T(s, a, s') |V_k(s') - V^\pi(s')|$$

$$\leq \gamma \varepsilon_k \sum_a \pi(s, a) \sum_{s'} T(s, a, s')$$

$$= \gamma \varepsilon_k \sum_a \pi(s, a) \cdot 1$$

$$= \gamma \varepsilon_k, \forall s \in S$$

- Because $\gamma < 1$, this means that $\lim_{k \to \infty} \varepsilon_k = 0$

- We say that the error contracts by a contraction factor of $\gamma$. 
Searching for a Good Policy

- We say that $\pi \succeq \pi'$ if $V^\pi(s) \succeq V^{\pi'}(s)$, $\forall s \in S$.

- This gives a partial ordering of policies.
  - If one policy is better at one state but worse at another state, the two policies are not comparable.

- Since we know how to compute values for policies, we can search through the space of policies.

- Local search seems like a good fit.

Policy Improvement

- Recall Bellman’s eqn: 
  $$V'(s) \leftarrow \sum_{a \in A} \pi(s,a) \left( R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V'(s') \right)$$

- Suppose that there is some action $a^*$, such that:
  $$\left( R(s,a^*) + \gamma \sum_{s' \in S} T(s,a^*,s') V'(s') \right) > V'(s)$$

- Then if we set $\pi(s,a^*) \leftarrow 1$, the value of state $s$ will increase.
  - Because we replaced each element in the sum in $V'(s)$ with a bigger value.
  - The values of states that can transition to $s$ increase as well.
  - The values of all other states stay the same.

- So the new policy using $a^*$ is better than the initial policy $\pi$. 
Policy Iteration

• More generally, we can change the policy \( \pi \) to a new policy \( \pi' \) which is greedy with respect to the computed values \( V^\pi \):

\[
\pi'(s) = \arg\max_{a \in A} ( R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^\pi(s') )
\]

• This gives us a local search through the space of policies.

• We stop when the values of two successive policies are identical.

• Because we only look for deterministic policies, and there is a finite number of them, the search is guaranteed to terminate.

Policy Iteration Algorithm

• Start with an initial policy \( \pi_0 \) (e.g. random)

• Repeat:
  – Compute \( V^\pi \), using policy evaluation.
  – Compute a new policy \( \pi' \) that is greedy with respect to \( V^\pi \)

• Terminate when \( V^\pi = V^{\pi'} \)
What you should know

- Definition of MDP framework.

- Differences/similarities between MDPs and other AI approaches (e.g. general search, game playing, STRIPS planning).

- Basic MDP algorithms and their properties:
  - Policy evaluation
  - Policy iteration

Final notes

- Final project: Don’t forget you need to submit working code (Thursday) and a written report (Friday).
  - See the instructions for information on what the report should contain.