Markov Decision Processes (MDPs)

Outline

- Markov Decision Processes (MDPs)
  - Formalism
  - Value iteration
    - In essence a graph search version of expectimax, but
      - there are rewards in every step (rather than a utility just in the terminal node)
      - ran bottom-up (rather than recursively)
      - can handle infinite duration games
  - Policy Evaluation and Policy Iteration
Non-Deterministic Search

How do you plan when your actions might fail?

Grid World

- The agent lives in a grid
- Walls block the agent’s path
- The agent’s actions do not always go as planned:
  - 80% of the time, the action North takes the agent North (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there is a wall in the direction the agent would have been taken, the agent stays put
- Small “living” reward each step (can be negative)
- Big rewards come at the end
- Goal: maximize sum of rewards
Grid Futures

Deterministic Grid World

Stochastic Grid World

Markov Decision Processes

- An MDP is defined by:
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - A transition function $T(s,a,s')$
    - Prob that $a$ from $s$ leads to $s'$
    - i.e., $P(s' | s, a)$
    - Also called the model
  - A reward function $R(s, a, s')$
    - Sometimes just $R(s)$ or $R(s')$
  - A start state (or distribution)
  - Maybe a terminal state

- MDPs are a family of non-deterministic search problems
  - One way to solve them is with expectimax search – but we’ll have a new tool soon
What is Markov about MDPs?

- Andrey Markov (1856-1922)
- “Markov” generally means that given the present state, the future and the past are independent
- For Markov decision processes, “Markov” means:

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

Solving MDPs

- In deterministic single-agent search problems, want an optimal plan, or sequence of actions, from start to a goal
- In an MDP, we want an optimal policy $$\pi^*: S \rightarrow A$$
  - A policy $$\pi$$ gives an action for each state
  - An optimal policy maximizes expected utility if followed
  - Defines a reflex agent

Optimal policy when $$R(s, a, s') = -0.03$$ for all non-terminals $$s$$
Example Optimal Policies

<table>
<thead>
<tr>
<th>R(s) = -0.01</th>
<th>R(s) = -0.03</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>R(s) = -0.4</th>
<th>R(s) = -2.0</th>
</tr>
</thead>
</table>

Example: High-Low

- Three card types: 2, 3, 4
- Infinite deck, twice as many 2's
- Start with 3 showing
- After each card, you say “high” or “low”
- New card is flipped
- If you’re right, you win the points shown on the new card
- Ties are no-ops
- If you’re wrong, game ends

Differences from expectimax:
- #1: get rewards as you go --- could modify to pass the sum up
- #2: you might play forever! --- would need to prune those, we'll see a better way
High-Low as an MDP

- **States:** 2, 3, 4, done
- **Actions:** High, Low
- **Model:** $T(s, a, s')$:
  - $P(s' = 4 \mid 4, \text{Low}) = 1/4$
  - $P(s' = 3 \mid 4, \text{Low}) = 1/4$
  - $P(s' = 2 \mid 4, \text{Low}) = 1/2$
  - $P(s' = \text{done} \mid 4, \text{Low}) = 0$
  - $P(s' = 4 \mid 4, \text{High}) = 1/4$
  - $P(s' = 3 \mid 4, \text{High}) = 0$
  - $P(s' = 2 \mid 4, \text{High}) = 0$
  - $P(s' = \text{done} \mid 4, \text{High}) = 3/4$
- **...**
- **Rewards:** $R(s, a, s')$:
  - Number shown on $s'$ if $s \neq s'$
  - 0 otherwise
- **Start:** 3

Example: High-Low
### MDP Search Trees

- Each MDP state gives an expectimax-like search tree

![Diagram of MDP state and transition](image)

- $(s, a)$ is a q-state
- $(s, a, s')$ is a transition
- $T(s, a, s') = P(s' | s, a)$
- $R(s, a, s')$

### Utilities of Sequences

- What utility does a sequence of rewards have?

- Formally, we generally assume stationary preferences:

\[
[r, r_0, r_1, r_2, \ldots] \succ [r, r'_0, r'_1, r'_2, \ldots]
\]

\[
[r_0, r_1, r_2, \ldots] \Leftrightarrow [r'_0, r'_1, r'_2, \ldots]
\]

- Theorem: only two ways to define stationary utilities
  - Additive utility:
    \[
    U([r_0, r_1, r_2, \ldots]) = r_0 + r_1 + r_2 + \cdots
    \]
  - Discounted utility:
    \[
    U([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots
    \]
Infinite Utilities?!

- **Problem**: infinite state sequences have infinite rewards

- **Solutions**:
  - **Finite horizon**:
    - Terminate episodes after a fixed $T$ steps (e.g. life)
    - Gives nonstationary policies ($\pi$ depends on time left)
  - Absorbing state: guarantee that for every policy, a terminal state will eventually be reached (like “done” for High-Low)
  - Discounting: for $0 < \gamma < 1$

  $U([r_0, \ldots r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq R_{\text{max}}/(1 - \gamma)$

  - Smaller $\gamma$ means smaller “horizon” – shorter term focus

---

Discounting

- **Typically discount rewards by $\gamma < 1$ each time step**
  - Sooner rewards have higher utility than later rewards
  - Also helps the algorithms converge

- **Example: discount of 0.5**
  - $U([1,2,3]) = 1*1 + 0.5*2 + 0.25*3$
  - $U([1,2,3]) < U([3,2,1])$
Recap: Defining MDPs

- Markov decision processes:
  - States $S$
  - Start state $s_0$
  - Actions $A$
  - Transitions $P(s' | s, a)$ (or $T(s,a,s')$)
  - Rewards $R(s,a,s')$ (and discount $\gamma$)

- MDP quantities so far:
  - Policy = Choice of action for each state
  - Utility (or return) = sum of discounted rewards

Our Status

- Markov Decision Processes (MDPs)
  - Formalism
  - Value iteration
    - In essence a graph search version of expectimax, but
      - there are rewards in every step (rather than a utility just in the terminal node)
      - ran bottom-up (rather than recursively)
      - can handle infinite duration games
  - Policy Evaluation and Policy Iteration
Expectimax for an MDP

Example MDP used for illustration has two states, $S = \{A, B\}$, and two actions, $A = \{1, 2\}$.
Example MDP used for illustration has two states, $S = \{A, B\}$, and two actions, $A = \{1, 2\}$. 

$i = \text{number of time-steps left}$

$i = 1$

$i = 2$

$i = 3$
Example MDP used for illustration has two states, $S = \{A, B\}$, and two actions, $A = \{1, 2\}$.
Example MDP used for illustration has two states, $S = \{A, B\}$, and two actions, $A = \{1, 2\}$.
Expectimax for an MDP

Value Iteration Performs this Computation Bottom to Top
Value Iteration for Finite Horizon H and no Discounting

- Initialization: \( \forall s \in S : V^0_i(s) = 0 \)
- For \( i = 1, 2, \ldots, H \)
  - For all \( s \in S \)
    - For all \( a \in A \):
      - \( Q^*_i(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + V^*_{i-1}(s')] \)
      - \( V^*_i(s) = \max_{a \in A} Q^*_i(s, a) \)
    - \( \pi^*_i(s) = \arg \max_{a \in A} Q^*_i(s, a) \)

- \( V^*(s) \): the expected sum of rewards accumulated when starting from state \( s \) and acting optimally for a horizon of \( i \) time steps.
- \( Q^*(s) \): the expected sum of rewards accumulated when starting from state \( s \) with \( i \) time steps left, and when first taking action and acting optimally from then onwards.
- How to act optimally? Follow optimal policy \( \pi^*(s) \) when \( i \) steps remain:
  \[
  \pi^*_i(s) = \max_a Q^*_i(s, a) = \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + V^*_{i-1}(s')]
  \]

Value Iteration for Finite Horizon H and with Discounting

- Initialization: \( \forall s \in S : V^0_i(s) = 0 \)
- For \( i = 1, 2, \ldots, H \)
  - For all \( s \in S \)
    - For all \( a \in A \):
      - \( Q^*_i(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*_{i-1}(s')] \)
      - \( V^*_i(s) = \max_{a \in A} Q^*_i(s, a) \)
    - \( \pi^*_i(s) = \arg \max_{a \in A} Q^*_i(s, a) \)

- \( V^*(s) \): the expected sum of discounted rewards accumulated when starting from state \( s \) and acting optimally for a horizon of \( i \) time steps.
- \( Q^*(s) \): the expected sum of discounted rewards accumulated when starting from state \( s \) with \( i \) time steps left, and when first taking action and acting optimally from then onwards.
- How to act optimally? Follow optimal policy \( \pi^*(s) \) when \( i \) steps remain:
  \[
  \pi^*_i(s) = \arg \max_a Q^*_i(s, a) = \arg \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*_{i-1}(s')]
  \]
Value Iteration Rewritten

- Initialization: \( \forall s \in S: V_0^*(s) = 0 \)
- For \( i = 1, 2, ..., H \)
  - For all \( s \in S \)
    - For all \( a \in A: Q_i^*(s, a) = \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V_{i-1}^*(s')] \)
    - \( V_i^*(s) = \max_{a \in A} Q_i^*(s, a) \)

Maps more directly to how you would code value iteration

Having done so, makes it very explicit that we can think of Value Iteration as computing the sequence \( V_0^*, V_1^*, V_2^*, ... \)

Rewritten version is convenient for our ensuing discussion of convergence properties

Convergence

- Question we are about to answer is whether this procedure converges, i.e., what happens for \( H \rightarrow \infty \)?
Convergence

Both are the optimal expected sum of rewards when acting for H+1 time steps in the same MDP, except that for V_\*^H, the rewards are set to zero for the transition H→H+1

- In the best possible scenario for V_\*^{H+1}, one is able to achieve V_\* in the first H time steps, and then \( \gamma^{H+1} \max_{s,a,s'} R(s,a,s') \) in the last time step
  - [you can’t do better than that, make sure you understand why]

- In the worst possible scenario for V_\*^{H+1}, one is able to achieve V_\* in the first H time steps, and then \( \gamma^{H+1} \min_{s,a,s'} R(s,a,s') \) in the last time step
  - [you can’t do worse than that, make sure you understand why]

Hence we have: 

\[ |V_\*^H(s) - V_\*^{H+1}(s)| \leq \gamma^{H+1} \max_{s,a,s'} |R(s,a,s')| \]

Hence the difference decays exponentially, and hence the series V_\*^1, V_\*^2, V_\*^3, ... converges to a limit, which we call V_\*.
Now we know how to act for infinite horizon with discounted rewards!

- Run value iteration till convergence.
- This produces $V^*$, which in turn tells us how to act, namely following:

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

- Note: the infinite horizon optimal policy is stationary, i.e., the optimal action at a state $s$ is the same action at all times. (Efficient to store!)

**Example: Bellman Updates**

$$V_{i+1}(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V_i(s')]$$

$$V_2((3,3)) = \sum_{s'} T((3,3), \text{right}, s')[R((3,3)) + 0.9 V_1(s')]$$

- max happens for $a=\text{right}$, other actions not shown

$$= 0.9 [0.8 \cdot 1 + 0.1 \cdot 0 + 0.1 \cdot 0]$$
Convergence (from Contraction Perspective)*

- Define the max-norm: \( ||U|| = \max_s |U(s)| \)

- Theorem: For any two approximations \( U \) and \( V \)

\[
||U_{i+1} - V_{i+1}|| \leq \gamma ||U_i - V_i||
\]

- I.e. any distinct approximations must get closer to each other, so, in particular, any approximation must get closer to the true \( U \) and value iteration converges to a unique, stable, optimal solution

- Theorem:

\[
||U_{i+1} - U_i|| < \epsilon, \Rightarrow ||U_{i+1} - U|| < 2\epsilon\gamma/(1 - \gamma)
\]

- I.e. once the change in our approximation is small, it must also be close to correct

Reminder: Computing Actions

- Which action should we chose from state \( s \):
  - Given optimal values \( V^* \)?

\[
\arg \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')] 
\]

  - Given optimal q-values \( Q^* \)?

\[
\arg \max_a Q^*(s, a)
\]

  - Lesson: actions are easier to select from Q’ s!
Our Status

- Markov Decision Processes (MDPs)
  - Formalism
  - Value iteration
    - In essence a graph search version of expectimax, but
      - there are rewards in every step (rather than a utility just in the terminal node)
      - ran bottom-up (rather than recursively)
      - can handle infinite duration games
  - Policy Evaluation and Policy Iteration

Policy Evaluation

- Another basic operation: compute the utility of a state $s$ under a fix (general non-optimal) policy

- Define the utility of a state $s$, under a fixed policy $\pi$:
  $$ V^\pi(s) = \text{expected total discounted rewards (return) starting in } s \text{ and following } \pi $$

- Recursive relation (one-step look-ahead / Bellman equation):
  $$ V^\pi(s) = \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V^\pi(s')] $$
### Policy Evaluation

- **How do we calculate the $V'$ s for a fixed policy?**
- **Idea one: modify Bellman updates**
  \[
  V_0^\pi(s) = 0
  \]
  \[
  V_{i+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_i^\pi(s')]
  \]
- **Idea two: it’s just a linear system, solve with Matlab (or whatever)**

### Policy Iteration

- **Alternative approach:**
  - **Step 1: Policy evaluation:** calculate utilities for some fixed policy (not optimal utilities!) until convergence
  - **Step 2: Policy improvement:** update policy using one-step look-ahead with resulting converged (but not optimal!) utilities as future values
  - Repeat steps until policy converges
- **This is policy iteration**
  - It’s still optimal!
  - Can converge faster under some conditions
Policy Iteration

- Policy evaluation: with fixed current policy \( \pi \), find values with simplified Bellman updates:
  - Iterate until values converge

  \[
  V_{i+1}^\pi_k(s) \leftarrow \sum_{s'} T(s, \pi_k(s), s') \left[ R(s, \pi_k(s), s') + \gamma V_i^\pi_k(s') \right].
  \]

- Policy improvement: with fixed utilities, find the best action according to one-step look-ahead

  \[
  \pi_{k+1}(s) = \arg \max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V^\pi_k(s') \right].
  \]

Policy Iteration Guarantees

Theorem. Policy iteration is guaranteed to converge and at convergence, the current policy and its value function are the optimal policy and the optimal value function!

Proof sketch:

(1) \textbf{Guarantee to converge:} we will not prove this, but the proof proceeds by first showing that in every step the policy improves. This means that a given policy can be encountered at most once. This means that after we have iterated as many times as there are different policies, i.e., \((\text{number actions}) \times (\text{number states})\), we must be done and hence have converged.

(2) \textbf{Optimal at convergence:} by definition of convergence, at convergence \( \pi_{i^*}(s) = \pi(s) \) for all states \( s \). This means

  \[
  V_{i^*}^\pi(s) = \max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_{i^*}^\pi(s') \right].
  \]

  Hence \( V_{i^*}^\pi \) satisfies the Bellman equation, which means \( V_{i^*}^\pi \) is equal to the optimal value function \( V^* \).
Comparison

- **In value iteration:**
  - Every pass (or “backup”) updates both utilities (explicitly, based on current utilities) and policy (possibly implicitly, based on current policy)

- **In policy iteration:**
  - Several passes to update utilities with frozen policy
  - Occasional passes to update policies

- **Hybrid approaches (asynchronous policy iteration):**
  - Any sequences of partial updates to either policy entries or utilities will converge if every state is visited infinitely often

Asynchronous Value Iteration*

- **In value iteration, we update every state in each iteration**

- **Actually, any** sequences of Bellman updates will converge if every state is visited infinitely often

- **In fact, we can update the policy as seldom or often as we like, and we will still converge**

- **Idea:** Update states whose value we expect to change: If $|V_{i+1}(s) - V_i(s)|$ is large then update predecessors of $s$
MDPs recap

- Markov decision processes:
  - States $S$
  - Actions $A$
  - Transitions $P(s' | s, a)$ (or $T(s, a, s')$)
  - Rewards $R(s, a, s')$ (and discount $\gamma$)
  - Start state $s_0$

- Solution methods:
  - Value iteration (VI)
  - Policy iteration (PI)
  - Asynchronous value iteration*

- Current limitations:
  - Relatively small state spaces
  - Assumes $T$ and $R$ are known