CS 188: Artificial Intelligence

Review of Utility, MDPs, RL, Bayes’ nets

DISCLAIMER: It is insufficient to simply study these slides, they are merely meant as a quick refresher of the high-level ideas covered. You need to study all materials covered in lecture, section, assignments and projects!

Pieter Abbeel – UC Berkeley
Many slides adapted from Dan Klein

Preferences

- An agent must have preferences among:
  - Prizes: $A$, $B$, etc.
  - Lotteries: situations with uncertain prizes

\[ L = [p, A; (1 - p), B] \]

- Notation:
  \[ A \succ B \quad A \text{ preferred over } B \]
  \[ A \sim B \quad \text{indifference between } A \text{ and } B \]
  \[ A \succeq B \quad B \text{ not preferred over } A \]
Rational Preferences

- Preferences of a rational agent must obey constraints.
  - The axioms of rationality:
    - Orderability
      \[(A \succ B) \lor (B \succ A) \lor (A \sim B)\]
    - Transitivity
      \[(A \succ B) \land (B \succ C) \Rightarrow (A \succ C)\]
    - Continuity
      \[A \succ B \succ C \Rightarrow \exists p \ [p, A; 1 - p, C] \sim B\]
    - Substitutability
      \[A \sim B \Rightarrow [p, A; 1 - p, C] \sim [p, B; 1 - p, C]\]
    - Monotonicity
      \[A \succ B \Rightarrow (p \geq q \iff [p, A; 1 - p, B] \succeq [q, A; 1 - q, B])\]

- Theorem: Rational preferences imply behavior describable as maximization of expected utility

MEU Principle

- Theorem:
  - [Ramsey, 1931; von Neumann & Morgenstern, 1944]
  - Given any preferences satisfying these constraints, there exists a real-valued function \(U\) such that:
    \[U(A) \geq U(B) \iff A \succeq B\]
    \[U([p_1, S_1; \ldots ; p_n, S_n]) = \sum_i p_i U(S_i)\]

- Maximum expected utility (MEU) principle:
  - Choose the action that maximizes expected utility
  - Note: an agent can be entirely rational (consistent with MEU) without ever representing or manipulating utilities and probabilities
  - E.g., a lookup table for perfect tic-tac-toe, reflex vacuum cleaner
Recap MDPs and RL

- **Markov Decision Processes (MDPs)**
  - Formalism: \( (S, A, T, R, \gamma) \)
  - Solution: policy \( \pi \) which describes action for each state
  - Value Iteration (vs. Expectimax --- VI more efficient through dynamic programming)
  - Policy Evaluation and Policy Iteration

- **Reinforcement Learning (don’t know \( T \) and \( R \))**
  - Model-based Learning: estimate \( T \) and \( R \) first
  - Model-free Learning: learn without estimating \( T \) or \( R \)
    - Direct Evaluation [performs policy evaluation]
    - Temporal Difference Learning [performs policy evaluation]
    - Q-Learning [learns optimal state-action value function \( Q^* \)]
    - Policy Search [learns optimal policy from subset of all policies]
  - Exploration
  - Function approximation --- generalization

---

**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
  - Reinforcement learning: MDPs where we don’t know the transition or reward functions
What is Markov about MDPs?

- “Markov” generally means that given the present state, the future and the past are independent.

- For Markov decision processes, “Markov” means:
  \[ P(S_{t+1} = s'|S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \ldots, S_0 = s_0) = P(S_{t+1} = s'|S_t = s_t, A_t = a_t) \]

- Can make this happen by proper choice of state space.

Value Iteration

- Idea:
  - \( V_i(s) \): the expected discounted sum of rewards accumulated when starting from state \( s \) and acting optimally for a horizon of \( i \) time steps.

- Value iteration:
  - Start with \( V_0(s) = 0 \), which we know is right (why?)
  - Given \( V_i \), calculate the values for all states for horizon \( i+1 \):
    \[
    V_{i+1}^*(s) \leftarrow \max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_i^*(s') \right]
    \]
  - This is called a value update or Bellman update.
  - Repeat until convergence.

- Theorem: will converge to unique optimal values.
  - Basic idea: approximations get refined towards optimal values.
  - Policy may converge long before values do.
  - At convergence, we have found the optimal value function \( V^* \) for the discounted infinite horizon problem, which satisfies the Bellman equations:
    \[
    \forall s \in S : \quad V^*(s) = \max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V^*(s') \right]
    \]
Complete Procedure

- 1. Run value iteration (off-line)
  - This results in finding $V^*$

- 2. Agent acts. At time $t$ the agent is in state $s_t$ and takes the action $a_t$:

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

Policy Iteration

- Policy evaluation: with fixed current policy $\pi_i$, find values with simplified Bellman updates:
  - Iterate for $i = 0, 1, 2, \ldots$ until values converge

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

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

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

- Will converge (policy will not change) and resulting policy optimal
Sample-Based Policy Evaluation?

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

- Who needs T and R? Approximate the expectation with samples (drawn from T!)

\[
\text{sample}_1 = R(s, \pi(s), s'_1) + \gamma V_i^{\pi}(s'_1) \\
\text{sample}_2 = R(s, \pi(s), s'_2) + \gamma V_i^{\pi}(s'_2) \\
\vdots \\
\text{sample}_k = R(s, \pi(s), s'_k) + \gamma V_i^{\pi}(s'_k) \\
V_{i+1}^{\pi}(s) \leftarrow \frac{1}{k} \sum_{i} \text{sample}_i
\]

Almost! (i) Will only be in state s once and then land in s' hence have only one sample \( \rightarrow \) have to keep all samples around? (ii) Where do we get value for s'?

Temporal-Difference Learning

- Big idea: learn from every experience!
  - Update V(s) each time we experience \((s,a,s',r)\)
  - Likely s' will contribute updates more often

- Temporal difference learning
  - Policy still fixed!
  - Move values toward value of whatever successor occurs: running average!

**Sample of V(s):**

\[ \text{sample} = R(s, \pi(s), s') + \gamma V^{\pi}(s') \]

**Update to V(s):**

\[ V^{\pi}(s) \leftarrow (1 - \alpha) V^{\pi}(s) + \alpha \text{sample} \]

**Same update:**

\[ V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha (\text{sample} - V^{\pi}(s)) \]
Exponential Moving Average

- Exponential moving average
  - Makes recent samples more important
  \[
  \bar{x}_n = \frac{x_n + (1 - \alpha) \cdot x_{n-1} + (1 - \alpha)^2 \cdot x_{n-2} + \ldots}{1 + (1 - \alpha) + (1 - \alpha)^2 + \ldots}
  \]
  - Forgets about the past (distant past values were wrong anyway)
  - Easy to compute from the running average
  \[
  \bar{x}_n = (1 - \alpha) \cdot \bar{x}_{n-1} + \alpha \cdot x_n
  \]
  - Decreasing learning rate can give converging averages

Detour: Q-Value Iteration

- Value iteration: find successive approx optimal values
  - Start with \( V_0(s) = 0 \), which we know is right (why?)
  - Given \( V_i \), calculate the values for all states for depth \( i+1 \):
  \[
  V_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_i(s') \right]
  \]
  - But Q-values are more useful!
    - Start with \( Q_0(s, a) = 0 \), which we know is right (why?)
    - Given \( Q_i \), calculate the q-values for all q-states for depth \( i+1 \):
  \[
  Q_{i+1}(s, a) \leftarrow \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma \max_{a'} Q_i(s', a') \right]
  \]
Q-Learning

- Learn Q*(s,a) values
  - Receive a sample (s,a,s',r)
  - Consider your new sample estimate:
    \[ Q^*(s, a) = \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma \max_{a'} Q^*(s', a') \right] \]
    \[ \text{sample} = R(s, a, s') + \gamma \max_{a'} Q(s', a') \]
  - Incorporate the new estimate into a running average:
    \[ Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha \text{[sample]} \]
  - Amazing result: Q-learning converges to optimal policy
    - If you explore enough
    - If you make the learning rate small enough but not decrease it too quickly!
- Neat property: off-policy learning
  - learn optimal policy without following it

Exploration Functions

- Simplest: random actions (ε greedy)
  - Every time step, flip a coin
  - With probability ε, act randomly
  - With probability 1-ε, act according to current policy
  - Problems with random actions?
    - You do explore the space, but keep thrashing around once learning is done
    - One solution: lower ε over time
- Exploration functions
  - Explore areas whose badness is not (yet) established
  - Take a value estimate and a count, and returns an optimistic utility, e.g. \( f(u, n) = u + k/n \) (exact form not important)
  - \( Q_{i+1}(s, a) \leftarrow (1 - \alpha)Q_i(s, a) + \alpha \left( R(s, a, s') + \gamma \max_{a'} Q_i(s', a') \right) \)
    - now becomes:
    \[ Q_{i+1}(s, a) \leftarrow (1 - \alpha)Q_i(s, a) + \alpha \left( R(s, a, s') + \gamma \max_{a'} f(Q_i(s', a'), N(s', a')) \right) \]
Feature-Based Representations

- Solution: describe a state using a vector of features
  - Features are functions from states to real numbers (often 0/1) that capture important properties of the state
  - Example features:
    - Distance to closest ghost
    - Distance to closest dot
    - Number of ghosts
    - \(1 / (\text{dist to dot})^2\)
    - Is Pacman in a tunnel? (0/1)
    - …… etc.
  - Can also describe a q-state \((s, a)\) with features (e.g. action moves closer to food)

Linear Feature Functions

- Using a feature representation, we can write a q function (or value function) for any state using a few weights:

  \[ V(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) \]

  \[ Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \ldots + w_n f_n(s, a) \]

- Advantage: our experience is summed up in a few powerful numbers
- Disadvantage: states may share features but be very different in value!
Overfitting

Degree 15 polynomial

Policy Search

- Problem: often the feature-based policies that work well aren’t the ones that approximate V/Q best
- Solution: learn the policy that maximizes rewards rather than the value that predicts rewards
- This is the idea behind policy search, such as what controlled the upside-down helicopter
- Simplest policy search:
  - Start with an initial linear value function or Q-function
  - Nudge each feature weight up and down and see if your policy is better than before
- Problems:
  - How do we tell the policy got better?
  - Need to run many sample episodes!
  - If there are a lot of features, this can be impractical