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:
  - \( A > B \) \( A \) preferred over \( B \)
  - \( A \sim B \) indifference between \( A \) and \( B \)
  - \( A \succeq B \) \( B \) not preferred over \( A \)

---

Rational Preferences

- Preferences of a rational agent must obey constraints.
  - The axioms of rationality:
    - Orderability: \( (A > B) \land (B > A) \land (A \sim B) \)
    - Transitivity: \( (A > B) \land (B > C) \Rightarrow (A > C) \)
    - Continuity: \( A \sim B \Rightarrow \exists p \in [0,1] \) \( A \sim pB \)
    - Substitutability: \( A \sim B \Rightarrow \exists p \in [0,1] \) \( A \sim pB \)
    - Monotonicity: \( A \sim B \Rightarrow \exists p \in [0,1] \) \( A \sim pB \)

- 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:
    \[ U(A) \geq U(B) \iff A \succeq B \]
    \[ U([p_1, S_1; \ldots ; p_m, S_m]) = \sum_i p_i U(S_i) \]

- 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 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 \( S \)
  - A set of actions \( A \in S \)
  - A transition function \( T(s, a, s') \)
  - A reward function \( R(s, a, s') \)
  - 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: $P(S_{t+1} = s'|S_t = s_t, A_t = a_t, S_{t-1}, A_{t-1}, \ldots, S_0 = s_0) = P(S_{t+1} = s'|S_t = s_t, A_t = a_t)$
- Can make this happen by proper choice of state space.

Value Iteration

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

$$\forall s \in S: V^*(s) = \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$$

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 $a_t$.
  - $\arg \max_a \sum_{s'} T(s_t, a_t, s') R(s_t, a_t, s') + \gamma V^*(s')$

Policy Iteration

- Policy evaluation: with fixed current policy $\pi$, find values with simplified Bellman updates:
  - Iterate for $i = 0, 1, 2, \ldots$ until values converge
  - $V_{i+1}^\pi(s) = \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_i^\pi(s')]$
- Policy improvement: with fixed utilities, find the best action according to one-step look-ahead.
  - $\pi_{i+1}(s) = \arg \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_i^\pi(s')]$
  - Will converge (policy will not change) and resulting policy optimal.

Sample-Based Policy Evaluation?

- $V_{i+1}^\pi(s) = \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_i^\pi(s')]$
- Who needs $T$ and $R$? Approximate the expectation with samples (drawn from $T$!)
  - $sample_1 = R(s, \pi(s), s_1') + \gamma V_i^\pi(s_1')$
  - $sample_2 = R(s, \pi(s), s_2') + \gamma V_i^\pi(s_2')$
  - $\ldots$
  - $sample_k = R(s, \pi(s), s_k') + \gamma V_i^\pi(s_k')$
- $V_{i+1}^\pi(s) = \frac{1}{k} \sum_{i} sample_i$

Temporal-Difference Learning

- Big idea: learn from every experience!
  - Update $V(s)$ each time we experience $(s, a, s', r)$
  - Likely $s'$ will get more updates than $s$.
- Temporal difference learning:
  - Policy still fixed.
  - Move values toward value of whatever successor occurs: running average!
- Sample of $V(s)$:
  - $sample = R(s, \pi(s), s') + \gamma V^\pi(s')$
- Update to $V(s)$:
  - $V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + \alpha sample$
- Same update:
  - $V^\pi(s) \leftarrow V^\pi(s) + \alpha (sample - V^\pi(s))$
Exponential Moving Average

- Exponential moving average
  - Makes recent samples more important
  \[ \tilde{x}_n = \frac{x_n + (1 - \alpha) \cdot x_{n-1} + (1 - \alpha)^2 \cdot x_{n-2} + \ldots}{1 + (1 - \alpha) + (1 - \alpha)^2 + \ldots} \]
  - Forgets about the past (distant past values were wrong anyway)
  - Easy to compute from the running average
  - 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 \( i+1 \):
  \[ V_{i+1}(s) = \max_a \sum_{s'} T(s, a, s') \left( R(s, a, s') + \gamma V_i(s') \right) \]
  - 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 \( i+1 \):
  \[ Q_{i+1}(s, a) = \sum_{s'} T(s, a, s') \left( R(s, a, s') + \gamma \max_{a'} Q_i(s', a') \right) \]

Q-Learning

- Learn \( Q^*(s,a) \) values
  - Receive a sample \((s,a,s',r)\)
  - Consider your new sample estimate:
  \[ Q^*(s, a) = \sum_{s'} T(s, a, s') \left( R(s, a, s') + \gamma \max_{a'} Q^*(s', a') \right) \]
  - Incorporate the new estimate into a running average:
  \[ Q(s, a) \leftarrow (1 - \alpha) Q(s, a) + \alpha \cdot \text{sample} \]
  - 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 (\( \varepsilon \)-greedy)
  - Every time step, flip a coin
  - With probability \( \varepsilon \), act randomly
  - With probability \( 1 - \varepsilon \), 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 \( \varepsilon \) 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.
  \[ Q_{i+1}(s, a) = \left( \frac{1}{N(s, a)} \right) \left[ R(s, a, s') + \gamma \max_{a'} Q_i(s', a') \right] \]
  - 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

Probability recap

- Conditional probability: $P(x|y) = \frac{P(x,y)}{P(y)}$
- Product rule: $P(x,y) = P(x|y)P(y)$
- Chain rule: $P(X_1,X_2,\ldots,X_n) = P(X_1)P(X_2|X_1)P(X_3|X_1,X_2)\ldots$
- $X, Y$ independent iff: $\forall x,y: P(x,y) = P(x)P(y)$
  - equivalently, iff: $\forall x,y: P(x|y) = P(x)$
- $X$ and $Y$ are conditionally independent given $Z$ iff:
  - $\forall x,y,z: P(x|y,z) = P(x|z)P(y|z)$
  - equivalently, iff: $\forall x,y,z: P(x|y,z) = P(x|z)$
  - equivalently, iff: $\forall x,y,z: P(y|x,z) = P(y|z)$

Inference by Enumeration

- $P(\text{sun})$
- $P(\text{sun} | \text{winter})$
- $P(\text{sun} | \text{winter, hot})$

Bayes’ Nets Recap

- Representation: Chain rule -> Bayes’ net = DAG + CPTs
- Conditional Independences: D-separation
- Probabilistic Inference:
  - Enumeration (exact, exponential complexity)
  - Variable elimination (exact, worst-case exponential complexity, often better)
  - Probabilistic inference is NP-complete
- Sampling (approximate)

Chain Rule $\rightarrow$ Bayes net

- Chain rule: can always write any joint distribution as an incremental product of conditional distributions
  $$P(x_1, x_2, x_3) = P(x_1)P(x_2|x_1)P(x_3|x_2)$$
  $$P(x_1, x_2, \ldots, x_n) = \prod_i P(x_i|x_{i-1})$$
- Bayes nets: make conditional independence assumptions of the form
  $$P(x_i|x_{i-1}, \ldots, x_1) = P(x_i|\text{parents}(X_i))$$
  giving us:
  $$P(x_1, x_2, \ldots, x_n) = \prod_{i=1}^n P(x_i|\text{parents}(X_i))$$
Probabilities in BNs

- Bayes’ nets implicitly encode joint distributions
  - As a product of local conditional distributions
  - To see what probability a BN gives to a full assignment, multiply all the relevant conditionals together:

\[
P(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} P(x_i | \text{parents}(X_i))
\]

- Example:

\[
P(\text{+cavity}, +\text{catch}, -\text{toothache})
\]

This lets us reconstruct any entry of the full joint

- Not every BN can represent every joint distribution
  - The topology enforces certain conditional independencies

Size of a Bayes’ Net for \(P(X_1, X_2, \ldots, X_n)\)

- How big is a joint distribution over \(N\) Boolean variables?
  \(2^N\)

- Size of representation if we use the chain rule
  \(2^N\)

- How big is an \(N\)-node net if nodes have up to \(k\) parents?
  \(O(N \cdot 2^{k+1})\)

- Both give you the power to calculate

- BNs:
  - Huge space savings!
  - Easier to elicit local CPTs
  - Faster to answer queries

Bayes Nets: Assumptions

- Assumptions made by specifying the graph:

\[
P(x_i | x_1 \cdots x_{i-1}) = P(x_i | \text{parents}(X_i))
\]

- Given a Bayes net graph additional conditional independences can be read off directly from the graph

- Question: Are two nodes guaranteed to be independent given certain evidence?
  - If no, can prove with a counter example
    - i.e., pick a set of CPT's, and show that the independence assumption is violated by the resulting distribution
  - If yes, can prove with
    - Algebra (tedious)
    - D-separation (analyzes graph)

D-Separation

- Question: Are \(X\) and \(Y\) conditionally independent given evidence vars \(Z\)?
  - Yes, if \(X\) and \(Y\) “separated” by \(Z\)
  - Consider all (undirected!) paths from \(X\) to \(Y\)
  - No active paths = independence

- A path is active if each triple is active:
  - Causal chain \(A \to B \to C\) where \(B\) is unobserved (either direction)
  - Common cause \(A \leftarrow B \to C\) where \(B\) is unobserved
  - Common effect (aka \(v\)-structure) \(A \to B \leftarrow C\) where \(B\) or one of its descendents is observed

- All it takes to block a path is a single inactive segment

Example: Alarm Network

Bayes Nets: Assumptions
Example

\[
\begin{align*}
L \perp T' | T & \quad \text{Yes} \\
L \perp B & \quad \text{Yes} \\
L \perp B | T & \\
L \perp B | T' & \\
L \perp B | T, R & \quad \text{Yes}
\end{align*}
\]

All Conditional Independences

- Given a Bayes net structure, can run d-separation to build a complete list of conditional independences that are necessarily true of the form

\[X_i \perp \perp X_j | \{X_{k_1}, \ldots, X_{k_n}\}\]

- This list determines the set of probability distributions that can be represented by Bayes’ nets with this graph structure

Topology Limits Distributions

- Given some graph topology G, only certain joint distributions can be encoded
- The graph structure guarantees certain (conditional) independences
- (There might be more independence)
- Adding arcs increases the set of distributions, but has several costs
- Full conditioning can encode any distribution

Inference by Enumeration

- Given unlimited time, inference in BNs is easy
- Recipe:
  - State the marginal probabilities you need
  - Figure out ALL the atomic probabilities you need
  - Calculate and combine them
- Example:

\[
P(+b \mid +j, +m) = \frac{P(+b, +j, +m)}{P(+j, +m)}
\]

Example: Enumeration

In this simple method, we only need the BN to synthesize the joint entries

\[
P(+b \mid +j, +m) = \\
P(+b)P(+c)P(+a \mid +b, +c)P(+j \mid +a)P(+m \mid +a) + \\
P(+b)P(+c)P(-a \mid +b, +c)P(+j \mid -a)P(+m \mid -a) + \\
P(+b)P(-c)P(+a \mid +b, -c)P(+j \mid +a)P(+m \mid +a) + \\
P(+b)P(-c)P(-a \mid +b, -c)P(+j \mid -a)P(+m \mid -a)
\]

Variable Elimination

- Why is inference by enumeration so slow?
  - You join up the whole joint distribution before you sum out the hidden variables
  - You end up repeating a lot of work!
- Idea: interleave joining and marginalizing!
  - Called “Variable Elimination”
  - Still NP-hard, but usually much faster than inference by enumeration
Variable Elimination Outline

- Track objects called factors
- Initial factors are local CPTs (one per node)
  
\[
\begin{array}{c|c|c|c}
  P(R) & P(T|R) & P(L|T) \\
  \hline
  +r & 0.1 & 0.9 \\
  -r & 0.9 & 0.1 \\
  +t & 0.8 & 0.2 \\
  -t & 0.2 & 0.8 \\
  +\ell & 0.1 & 0.9 \\
  -\ell & 0.9 & 0.1 \\
  +r & +t & 0.15 \\
  +r & -t & 0.05 \\
  -r & +t & 0.05 \\
  -r & -t & 0.85 \\
\end{array}
\]

- Any known values are selected
  - E.g. if we know \( L = +\ell \), the initial factors are

\[
\begin{array}{c|c|c|c}
  P(R) & P(T|R) & P(L|T) \\
  \hline
  +r & 0.1 & 0.9 \\
  -r & 0.9 & 0.1 \\
  +t & 0.8 & 0.2 \\
  -t & 0.2 & 0.8 \\
  +\ell & 0.1 & 0.9 \\
  -\ell & 0.9 & 0.1 \\
\end{array}
\]

- VE: Alternately join factors and eliminate variables

Variable Elimination Example

\[
P(R) \\
\hline
+r & 0.1 \\
-r & 0.9 \\
\]

Join \( R \)

\[
P(T|R) \\
\hline
+T & +r & 0.8 \\
+T & -r & 0.2 \\
-r & +t & 0.09 \\
-r & -t & 0.91 \\
\]

Sum out \( R \)

\[
P(T) \\
\hline
+T & +r & 0.17 \\
+T & -r & 0.83 \\
-r & +t & 0.051 \\
-r & -t & 0.119 \\
\]

Example

Choose \( A \)

\[
P(A|B, E) \\
\hline
+A & +B & +E & 0.3 \\
+A & +B & -E & 0.7 \\
-A & +B & +E & 0.1 \\
-A & +B & -E & 0.9 \\
\]

Finish with \( B \)

\[
P(B) \\
\hline
+B & 0.134 \\
-B & 0.886 \\
\]

General Variable Elimination

- Query: \( P(Q|E_1, \ldots, E_k = e_k) \)
- Start with initial factors:
  - Local CPTs (but instantiated by evidence)
- While there are still hidden variables (not \( Q \) or evidence):
  - Pick a hidden variable \( H \)
  - Join all factors mentioning \( H \)
  - Eliminate (sum out) \( H \)
- Join all remaining factors and normalize
Another (bit more abstractly worked out) Variable Elimination Example

Query: \( P(X_1, X_2, X_3, Y_1, Y_2, Y_3 | Z) \)

Start by knowing evidence, which gives the following initial factors:
- \( P(Y_1 | X_1, X_2, X_3) \)
- \( P(Y_2 | X_1, X_2, X_3) \)
- \( P(Y_3 | X_1, X_2, X_3) \)

Remove \( X_1 \), this introduces the factor \( P(Y_1, Y_2, Y_3 | X_2, X_3) \) and \( P(Y_1, Y_2, Y_3 | X_2, X_3) \)

Remove \( X_2 \), this introduces the factor \( P(Y_1, Y_2, Y_3 | X_1, X_3) \) and \( P(Y_1, Y_2, Y_3 | X_1, X_3) \)

Eliminate \( Z \), this introduces the factor \( P(Y_1, Y_2, Y_3 | X_1, X_2, X_3) \) and \( P(Y_1, Y_2, Y_3 | X_1, X_2, X_3) \)


Computational complexity critically depends on the largest factor being generated in this process. Size of factor = number of entries in table. In example above (assuming binary) all factors generated are of size 2 — as they all only have one variable (Z, Z, and X3 respectively).

Variable Elimination Ordering

For the query \( P(X_1, Y_1, \ldots, Y_n | Z) \) work through the following two different orderings as done in previous slide: Z, X1, ..., Xn, and X1, ..., Xn, Z. What is the size of the maximum factor generated for each of the orderings?

Answer: 2^n versus 2 (assuming binary)

In general: the ordering can greatly affect efficiency.

Computational and Space Complexity of Variable Elimination

- The computational and space complexity of variable elimination is determined by the largest factor.
- The elimination ordering can greatly affect the size of the largest factor.
  - E.g., previous slide’s example 2^n vs. 2
- Does there always exist an ordering that only results in small factors?
  - No!

Worst Case Complexity?

- Consider the 3-SAT clause:

Approximate Inference: Sampling

- Basic idea:
  - Draw N samples from a sampling distribution S
  - Compute an approximate posterior probability
  - Show this converges to the true probability P
- Why? Faster than computing the exact answer
- Prior sampling:
  - Sample ALL variables in topological order as this can be done quickly
  - Rejection sampling for query \( P(Q | E_1 = e_1, \ldots, E_k = e_k) \)
  - If like prior sampling, but reject when a variable is sampled inconsistent with the query, in this case when a variable E_i is sampled differently from e_i
  - Likelihood weighting for query \( P(Q | E_1 = e_1, \ldots, E_k = e_k) \)
  - If like prior sampling but variables E_i are not sampled, when it’s their turn, they get set to e_i, and the sample gets weighted by \( P(Q | \text{value of parent}(e_i) \text{ in current sample}) \)
  - Gibbs sampling: repeatedly samples each non-evidence variable conditioned on all other variables \( \rightarrow \) can incorporate downstream evidence

Polytrees

- A polytree is a directed graph with no undirected cycles
- For poly-trees you can always find an ordering that is efficient
  - Try it!!
  - Cut-set conditioning for Bayes’ net inference
    - Choose set of variables such that if removed only a polytree remains
    - Think about how the specifics would work out?
Prior Sampling

We’ll get a bunch of samples from the BN:
- +c, -s, +r, +w
- -c, +s, +r, +w
- -c, -s, -r, -w
- +c, -s, +r, +w
- -c, +s, -r, +w

If we want to know P(W)
- We have counts <+w:4, -w:1>
- Normalize to get P(W) = <+w:0.8, -w:0.2>
- This will get closer to the true distribution with more samples
- Can estimate anything else, too
- What about P(C| +w)? P(C| +r, +w)? P(C| -r, -w)?
- Fast: can use fewer samples if less time

Likelihood Weighting

Sampling distribution if z sampled and e fixed evidence

Now, samples have weights

Together, weighted sampling distribution is consistent

Gibbs Sampling

Idea: instead of sampling from scratch, create samples that are each like the last one.

Procedure: resample one variable at a time, conditioned on all the rest, but keep evidence fixed.

Properties: Now samples are not independent (in fact they’re nearly identical), but sample averages are still consistent estimators!

What’s the point: both upstream and downstream variables condition on evidence.