CS188 Midterm1

Yibing Chen

MDP

- **Definition**:An MDP is defined by: A set of states $s \in S$, A set of actions $a \in A$, A transition fxn $T(s,a,s') \leftarrow$ probability that a from s leads to s': $P(s'|s,a)$. Also called the model, A reward fxn $R(s,a,s')$, A start state (or distribution), maybe a terminal state

Fundamental operation: compute the values (optimal expectimax utilities) of states s. Optimal values define optimal policies!

- Define value of a state s: $V^*(s)$= expected utility starting in s and acting optimally
- define the value of a q-state (s,a): $Q^*(s,a)$=expected utility starting in s, taking action a and then acting optimally
- Define the optimal policy: $\pi^*(s)$ = optimal action from state s

Bellman Equations: definition of "optimal utility" leads to a simple one-step lookahead relationship amongst optimal utility values: Optimal rewards = maximize over first action and then follow optimal policy:

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

Value Estimates: Calculate estimates $V_k^*(s) \leftarrow$ not the optimal value of s! It's the optimal value considering only next k time steps. As $k \to \infty$, approaches optimal value

Value Iteration $V_i^*(s)$:the expected discounted sum of rewards accumulated when starting from state s and acting optimally for a horizon of i time steps, Start with $V_0^*(s) = 0$, then $V_{i+1}^*(s) \leftarrow max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_i^*(s')] \leftarrow$ this is called a value update. Repeat until convergence. Approximations get refined towards optimal values. Computer optimal values for all states all at once using successive approximations (before you start moving)

We can also compute the utility of a state s under a fix (general non-optimal) policy. Similar definition for $V^\pi(s)$: Solve with modifying Bellman updates. $V_0^\pi(s) = 0$ Then $V_{i+1}^\pi(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V_i^\pi(s')]$ OR solve it as a linear system

Policy Iteration

- 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 until policy converges

In value iteration: every pass updates both utilities and policy. In policy iteration: several passes to update utilities with frozen policy with occasional passes to update policies

For **reinforcement learning** still assume an MDP, but don't know T or R. Don't know which states are good or what the actions do.

- Learn the model empirically through experience. Solve for values as if the learned model were correct
- Count outcomes for each s,a . Normalize to give estimate of T(s,a,s'). Discover R(s,a,s') when we experience (s,a,s')
- Model-based RL: first act in MDP and learn the transition model and reward fxn, then run value iteration or policy iteration with the learned models and fxns. Advantage: efficient use of data. Disadvantage: requires building a model for T, R
- Model-free RL: bypass the need to learn the model and fxn! Approaches: direct evaluation, temporal difference learning, q-learning

Direct Evaluation: repeatedly execute the policy π, estimate the value of the state s as the average over all times the state s was visited of the sum of discounted rewards accumulated from state s onwards. (limitations:) assume random initial state, assumes the value of a state is known perfectly based on past runs.

Temporal Difference Learning

- learn from every experience! Update V(s) each time we experience (s,a,s',r). Likely s' will contribute updates more often
- 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): $V^\pi(s) \leftarrow (1-\alpha)V^\pi(s) + (\alpha) \cdot sample$
- Same update: $V^\pi(s) \leftarrow V^\pi(s) + \alpha(sample - V^\pi(s))$ (α is the learning rate)
- If we want to turn values into a new policy we can't do this. Try learning Q-values directly

Q-value Iteration

- Value iteration: find successive approx optimal values. Start with $V_0(s) = 0$. Given V_i, calculate the values for all states for depth i+1 $v_{i+1}(s) \leftarrow max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_i(s')]$

- But Q-values are more useful! $Q_0(s,a) = 0$. Given Q_i calculate q-values for all q-states for depth i+1: $Q_{i+1}(s,a) \leftarrow \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma max_{a'} Q_i(s',a')]$

Q-Learning: sample-based Q-value iteration

- Learn $Q^*(s,a)$ values. Receive a sample (s,a,s',r). Consider your old estimate $Q(s,a)$
- New sample estimate: $Q^*(s,a,) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma max_{a'} Q^*(s',a')]$
- incorporate new estimate into running average: $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)[sample]$ (sample $= R(s,a,s') + \gamma max_{a'} Q(s',a')$)
- Q-learning converges to optimal policy. Learn optimal policy w/o following it
- In realistic situations, we can't learn about every single state, so we generalize: learn about some small number of training states through experience, and generalize that experience to new, similar states
- Feature-based representations are our solution: Features are functions from states to real numbers that capture important properties of the state. Can write a q or value fxn for any state using a few weights: $V(s) = \omega_1 f_1(s) + \omega_2 f_2(s) + \cdots + \omega_n f_n(s) \quad Q(s,a) = \omega_1 f_1(s,a) + \omega_2 f_2(s,a) + \cdots + \omega_n f_n(s,a)$

Policy search: often feature-based policies that work well aren't the ones that approximate V and Q best. We should learn the policy that maximizes rewards rather than the value that predicts rewards

- Start with an initial linear value fxn or Q-fxn
- Nudge each feature weight up and down and see if your policy is better than before
- need to run many sample episodes!

Chain rule

Inference by enumeration: chain rule: $P(X_1,\dots,X_n) = \prod_{i=1}^{n} P(x_i|\text{parents}(X_i))$. sum out the hidden variables.

$$P(B|j,m) \propto P(B,j,m) = \sum_e \sum_a P(B,j,m,e,a)$$
$$= \sum_e \sum_a P(b)P(e)P(a|b,e)P(j|a)P(m|a)$$

Variable elimination: do a calculation once and save it for later use

$$P(B,j,m) = \underbrace{P(B)}_{f_1(B)} \sum_e \underbrace{P(e)}_{f_2(E)} \sum_a \underbrace{P(a|B,e)}_{f_3(A,B,E)} \underbrace{P(j|a)}_{f_4(A)} \underbrace{P(m|a)}_{f_5(A)}$$

Sum out A from the product of f_3, f_4, f_5 to make a new 2×2 factor $f_6(B,E)$:

$$f_6(B,E) = \sum_a f_3(A,B,E) \times f_4(A) \times f_5(A) =$$

$$f_3(a,B,E) \times f_4(a) \times f_5(a) + f_3(\neg a,B,E) \times f_4(\neg a) \times f_5(\neg a)$$

Now we have

$$P(B,j,m) = f_1(B) \times \sum_e f_2(E) \times f_6(B,E)$$

(remember that if you have an entry $P(A,B) \times P(B,C) \Rightarrow P(A,B,C) = P(A,B) \cdot P(B,C)$)

Probability:

Conditional probability: $P(x|y) = \frac{P(x,y)}{P(y)} = \frac{P(y|x)P(x)}{P(y)}$

Chain rule: $P(X_1,\dots,X_n) = \prod_{i=1}^{n} P(x_i|\text{parents}(X_i))$

$XY|Z$ if $P(x,y|z) = P(x|z)P(y|z)$ or $P(x|y,z) = P(x|z)$or$P(y|x,z) = P(y|z)$

DBNs

- We want to track multiple variables over time, using multiple sources of evidence. Repeat a fixed Bayes net structure at each time. Variables from time t can condition on those from $t-1$. Discrete valued dynamic Bayes nets are also HMMs
- Exact inference in DBNs: variable elimination applies to dynamic Bayes nets. Procedure: "unroll" the network for T time steps, then eliminate variables until $P(X_t|e_{1:T})$ is computed. Online belief updates: Eliminate all variables from the previous time step; store factors for current time only
- DBN Particle Filters: a particle is a complete sample for a time step. **Initialize**: Generate prior samples for the $t=1$ Bayes net. **Elapse time**: sample a successor for each particle **Observe**: weight each entire sample by the likelihood of the evidence condition on the sample

n	Number of states in the problem
b	the average branching factor (# of successors)
C^*	Cost of least cost solution
s	depth of the shallowest solution
m	Max depth of the search tree

Algorithm	Modifiers	Complete?	Optimal?	Time	Space
DFS		N	N	infinite	infinite
DFS	w/ cycle checking	Y	N	$O(b^m)$	$O(bm)$
BFS		Y	N^*	$O(b^{s+1})$	$O(b^{s+1})$
ID		Y	N^*	$O(b^{s+1})$	$O(bs)$
UCS		Y^*	Y	$O(b^{C^*/\epsilon})$	$O(b^{C^*/\epsilon})$