CS 188 Discussion 9:
MDP Sampling, Decision Networks, VPI

Kenny Wang (kwkw@berkeley.edu)
Wed Oct 25, 2023

Slides based on Sashrika + Joy

Administrivia

- Project 4 due on Mon, Nov 6
- Homework is due on Tuesdays
- We have office hours pretty much all day every weekday (12-7), come to Soda 341B! (my hours are 1-3 PM on Mondays)
- Discussion slides are on Ed

Today’s Topics

- MDP Sampling
  - Prior Sampling
  - Rejection Sampling
  - Likelihood Weighting
  - Gibbs Sampling
- Decision Networks
- Value of Perfect Information (VPI)

Prior Sampling

- Algorithm:
    - Generate samples

    - For i=1, 2, …, n (in topological order)
        - Sample X_i from $P(X_i \mid parents(X_i))$
    - Return $(x_1, x_2, \ldots, x_n)$

    - Discard samples inconsistent with evidence
    - Calculate probability of the query
- Pros:
    - Easy!
- Cons:
    - Requires a large number of samples, especially when we condition on unlikely scenarios

Rejection Sampling **Worksheet Q1a**

- Algorithm:
  - Generate samples

    - Input: evidence $e_1,..,e_k$
    - For i=1, 2, …, n
      - Sample X_i from $P(X_i \mid parents(X_i))$
      - If x_i not consistent with evidence
        - Reject: Return, and no sample is generated in this cycle
    - Return $(x_1, x_2, \ldots, x_n)$

  - Calculate probability of the query
- Pros:
  - Bad samples take less time to create, so more efficient than Prior Sampling
- Cons:
  - Still requires a large number of samples, especially when we condition on unlikely scenarios

Likelihood Weighting

Worksheet Q1b

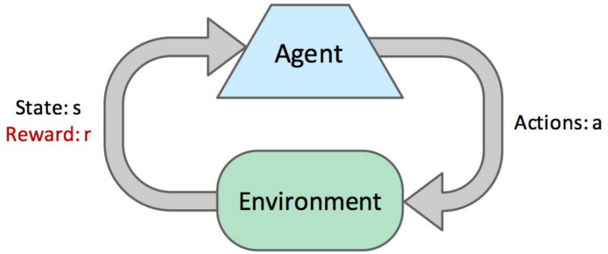
- Idea: Set all your variables to agree with evidence, then sample
  - **Weight** all samples by the probability of the evidence given the sampled variables

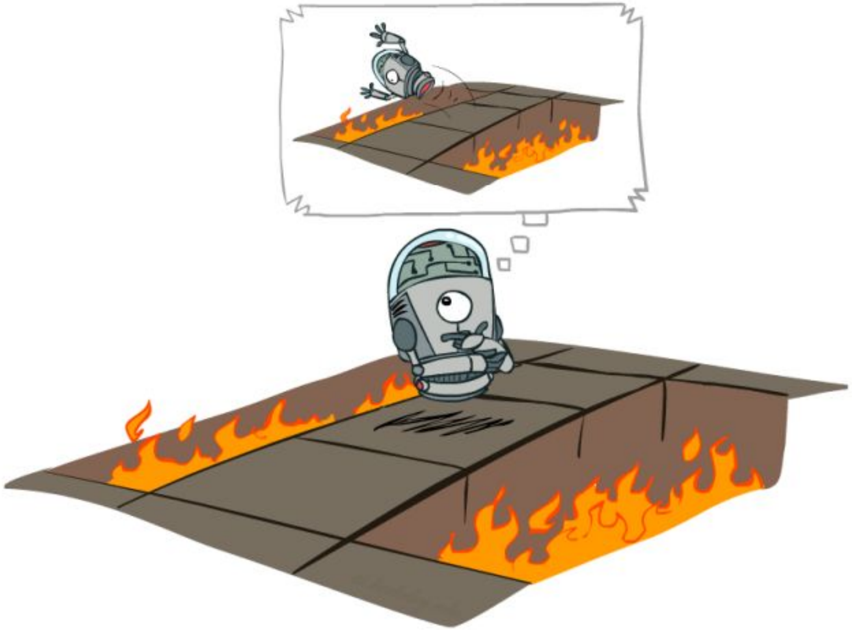
- Pros:
  - Never generate a bad sample!
- Cons:
  - Evidence influences the choice of downstream variables, but not upstream ones. Samples could have low weight!

- Input: evidence $e_1,..,e_k$
- $w = 1.0$
- for i=1, 2, …, n
  - if X_i is an evidence variable
    - x_i = observed value$_i$ for X_i
    - Set $w = w * P(x_i \mid parents(X_i))$
  - else
    - Sample x_i from $P(X_i \mid parents(X_i))$
- return $(x_1, x_2, \ldots, x_n)$, w

Reinforcement Learning Overview

- We solved MDPs using **offline planning**
  - Knew exact transition and reward functions, could precompute optimal actions without trying anything
- Now we move to **online planning**
  - Agent has no prior knowledge of transitions and rewards
  - Try **exploration** to receive feedback
  - Estimate an optimal policy to use for **exploitation** (maximizing rewards)





Offline Solution:

Compute policy ahead of time

Online Learning:

Compute policy as experience comes in

Types of Reinforcement Learning

- **Passive RL:** learn values from experience using a given policy, then use that to inform better policies
  - **Model-based:** learn the MDP model (transitions and rewards) from experiences, then solve the MDP
  - **Model-free:** forego learning the MDP model, directly learn V(s) or Q(S, a)
    - **Direct Evaluation**
    - **TD-learning (Temporal Difference)**
- **Active RL:** learn policy from our experiences directly
  - **Q-learning:** learns Q(state, action) values of the optimal policy
  - **Approximate Q-learning**

Model-Based Learning

- **Model-based learning:** observe a bunch of actions, then estimate transition probabilities T(s, a, s’) and rewards R(s, a, s’) by averaging.
    - T(s, a, s’) is the number of times you end up in s’ after being in state s and taking action a, divided by the total number of times you took a from s.
    - R(s, a, s’) is the average reward you got from starting in s, taking action a, and ending up in s’.

Worksheet 1(a)

TD-Learning (model-free, passive RL)

- Idea: learn values of states, given that you're following some policy
- **Algorithm**
  - Initialize value estimates for policy π $\boxed{\forall s,\ V^{\pi}(s) = 0}$
  - Each timestep:
    - Take action π(s) and receive reward R(s, π(s), s')
    - Obtain sample $\boxed{\text{sample} = R(s,\pi(s),s') + \gamma V^{\pi}(s')}$
    - Update value estimate with **exponential moving average**

$$\boxed{V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + \alpha \cdot \text{sample}}$$

    - Learning rate α, 0 ≤ α ≤ 1

Q-Learning (active RL)

- Idea: learn Q(s, a) values - which gives you the optimal policy (maximize Q)
- **Algorithm**
  - Initialize Q(state, action) estimates to 0
  - Each timestep:
    - Take some action, any action! It doesn't need to be optimal
    - Obtain sample $\boxed{\text{sample} = R(s,a,s') + \gamma \max_{a'} Q(s',a')}$
    - Update Q-value estimate with exponential moving average

$$\boxed{Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha \cdot \text{sample}}$$

  - If we explore enough and decrease α appropriately, learn optimal Q-values

Worksheet 1(b)

Value Iteration

Value Iteration

- A dynamic programming algorithm we use to compute values until convergence ($\forall s, U_{k+1}(s) = U_k(s)$)
- Algorithm
  - $\forall s \in S$, initialize $U_0(s) = 0$ [initial value estimate is 0]
  - Repeat rule until convergence (U-values stop changing)

$$\forall s \in S,\ U_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma U_k(s')]$$

- Convergence is when
  - $\forall s \in S, U_k(s) = U_{k+1}(s) = U^*(s)$

Policy Iteration

Policy Iteration

- Issues with value iteration
  - $O(|S|^2|A|)$ runtime
  - Overcomputes, policy tends to converge faster than values
- Policy iteration: preserve the optimality from value iteration but with better performance by iterating until only the *policy* converges instead of the U-values

Policy Iteration

- Algorithm
  - Define initial policy $\boldsymbol{\pi_0}$ (can be arbitrary)
  - Repeat until convergence:
    - **Policy evaluation:** compute expected utility of starting in state s when following policy π, for all states s

$$U^{\pi}(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma U^{\pi}(s')]$$

    - **Policy improvement:** generate a better policy

$$\pi_{i+1}(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma U^{\pi_i}(s')]$$

  - Convergence when $\boldsymbol{\pi_{i+1} = \pi_i}$ (policy stops changing)

Summary

- **Value Iteration**
  - $\forall s \in S$, initialize $U_0(s) = 0$ [initial value estimate is 0]
  - Repeat rule until convergence (U-values stop changing)

$$\forall s \in S,\ U_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma U_k(s')]$$

  - Convergence is when $\forall s \in S$, $U_k(s) = U_{k+1}(s) = U^*(s)$
- **Policy Iteration**
  - Define initial policy $\boldsymbol{\pi_0}$ (can be arbitrary)
  - Repeat until convergence:
    - **Policy evaluation:** compute expected utility of starting in state s when following policy π, for all states s

$$U^\pi(s) = \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma U^\pi(s')]$$

    - **Policy improvement:** generate a better policy

$$\pi_{i+1}(s) = \operatorname*{argmax}_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma U^{\pi_i}(s')]$$

  - Convergence when $\boldsymbol{\pi_{i+1} = \pi_i}$ (policy stops changing)
- **Q*(s,a): the optimal value of (s, a)** [state, action pair]

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

- **U*(s): the optimal value of state s** $U^*(s) = \max_a Q^*(s,a)$

Rest of the Worksheet

Thank you for attending!

Attendance link:

- https://tinyurl.com/cs188fa23

Discussion No: 6

Remember my name is Kenny

My email: kwkw@berkeley.edu