MDP & Reinforcement Learning

CS 4804 Fall 2020
Virginia Tech
Today’s Topics

- MDP
- Reinforcement Learning (RL)
Recap: Definition of MDP

- A set of states $S$
- A start state
- A set of actions $a$
- Possibly one or more terminal states
- A transition function $T(s,a,s')$ from $s$ to $s'$ with an action $a$
- A reward function $R(s,a,s')$ from $s$ to $s'$ with an action $a$. It may be positive or negative
- Possibly a discount factor $0 \leq \gamma \leq 1$
MDP Quantities

- Policy: A **solution** to the MDP problem. Tell an agent what to do in each state
- Utility: Sum of discounting rewards
Optimal Quantities

- Utility of a state $s$: expected utility starting from $s$ and acting optimally
  $$U^*(s) = \max_a Q^*(s, a)$$

- Utility of a q-state $(s, a)$: expected utility starting out having taken action $a$ from state $s$ and (thereafter) acting optimally
  $$Q^*(s, a) = \sum_{s'} P(s'|s, a) [R(s, a, s') + \gamma U^*(s')]$$
  $$= \sum_{s'} P(s'|s, a) [R(s, a, s') + \gamma \max_a Q^*(s', a')]$$

- $U^*(s) = \max_a \sum_{s'} P(s'|s, a) [R(s, a, s') + \gamma U^*(s')]$
Value Iteration

- Start with initial values of zeros. $U_0(s) = 0$
- Do Bellman update
- Repeat until convergence
- Complexity of each iteration: $O(S^2A)$
Bellman Update

• Let $U_i(s)$ be the utility value for state $s$ at the $i$th iteration

• Bellman update:

$$U_{i+1}(s) \leftarrow \max_{a \in A(s)} \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma U_i(s')]$$

• The update is to be applied simultaneously to all the states at each iteration
Policy from Value Iteration

VALUES AFTER 0 ITERATIONS

VALUES AFTER 100 ITERATIONS

Q-VALUES AFTER 100 ITERATIONS
Observation from Value Iteration

• Value iteration repeats the Bellman update
• The “max” at each state rarely changes
• It looks like it does more than it should be
  – The policy often converges long before the values
  – It is slow
• It is possible to get an optimal policy even when the utility function estimate is inaccurate
• If one action is clearly better than others, then we don’t need the precise utilities on the states
K=12 vs K=100

\[ \gamma = 0.9 \text{ and } r = 0 \]
Policy Iteration

• Alternative approach for optimal policy:
  – Step 1: **Policy evaluation**: calculate utilities for some fixed policies until convergence
  – Step 2: **Policy improvement**: update policy using one-step look-ahead with resulting converged utilities as future values
  – Repeat steps until policy converges (policy improvement step yields no change in the utilities)

• It is still optimal
• Can converge (much) faster under some conditions
• Efficiency: $O(S^2)$ per iteration
Policy Iteration

function POLICY-ITERATION(mdp) returns a policy

inputs: mdp, an MDP with states $S$, actions $A(s)$, transition model $P(s' | s, a)$
local variables: $U$, a vector of utilities for states in $S$, initially zero
$\pi$, a policy vector indexed by state, initially random

repeat

$U \leftarrow$ POLICY-EVALUATION($\pi$, $U$, mdp)
unchanged? $\leftarrow$ true

for each state $s$ in $S$ do

$\alpha^* \leftarrow \arg \max \limits_{a \in A(s)} Q$-VALUE($mdp$, $s$, $a$, $U$)

if $Q$-VALUE($mdp$, $s$, $\alpha^*$, $U$) $>$ $Q$-VALUE($mdp$, $s$, $\pi[s]$, $U$) then

$\pi[s] \leftarrow \alpha^*$; unchanged? $\leftarrow$ false

until unchanged?

return $\pi$
Fixed Policies

Do the optimal action

- Expectimax trees max over all actions to compute the optimal values

Do what $\pi$ says to do

- Expectimax trees max one action per state to compute the optimal values
Utilities for a Fixed Policy

- Compute the utility of a state $s$ under a fixed initial random policy
- Expected total discounted rewards starting from $s$ and following policy $\pi$

$$U^\pi(s) = E \left[ \sum_{t=0}^{\infty} \gamma^t R(S_t, \pi(S_t), S_{t+1}) \right]$$

- $U^\pi(s) = \sum_{s'} P(s' | s, \pi(s))[R(s, \pi(s), s') + \gamma U^\pi(s')]$
Policy Evaluation

- Given a policy $\pi_i$, calculate $U_i = U^{\pi_i}$, the utility of each state if $\pi_i$ were to be executed.

$$ U_i(s) = \sum_{s'} P(s'|s, \pi_i(s))[R(s, \pi_i(s), s') + \gamma U_i(s')] $$
Demo: Policy Evaluation
Policy Improvement

- Calculate a new MEU policy $\pi_{i+1}$ using one-step look-ahead based on $U_i$
- Get a better policy using policy extraction
- $\pi^*(s) = \max_{a \in A(s)} \sum_{s'} P(s'|s,a) [R(s,a,s') + \gamma U_i(s')]$
Demo: Policy Improvement

-0.12  -0.12  0.65
-0.12  -0.21  -1
-0.12  -0.12  -0.12

-0.12  -0.12  0.57
-0.08  0.80
-0.12

-0.12  -0.12  -0.12  -0.12
-0.12  -0.21  -1
-0.12  -0.12  -0.12

-0.12  -0.12  0.57
-0.08  0.80
-0.12

-0.12  -0.12  -0.12  -0.12
-0.12  -0.21  -1
-0.12  -0.12  -0.12

0.57  -0.08  0.80
-0.12
### Comparison

<table>
<thead>
<tr>
<th>Value Iteration</th>
<th>Policy Iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Every iteration updates the values (Bellman update)</td>
<td>Every iteration updates the utilities with fixed policy (simplified Bellman update)</td>
</tr>
<tr>
<td>Compute Q-Values</td>
<td>Compute Q-Values</td>
</tr>
<tr>
<td></td>
<td>After the policy is evaluated, a new policy is chosen (one-step lookahead)</td>
</tr>
<tr>
<td>Stop until value converges</td>
<td>Stop until policy converges</td>
</tr>
<tr>
<td>Policy extraction (one-step lookahead)</td>
<td>Dynamic program</td>
</tr>
<tr>
<td>Dynamic program</td>
<td>Dynamic program</td>
</tr>
</tbody>
</table>
Convergence of Value Iteration

![Graph showing convergence of value iteration with different points labeled (3,3), (1,3), (1,1), (3,1), and (4,1). The x-axis represents the number of iterations, ranging from 0 to 40, and the y-axis represents utility estimates, ranging from -0.2 to 1.]
Convergence of Value Iteration

- Bellman update:
  \[ U_{i+1}(s) \leftarrow \max_{a \in A(s)} \sum_{s'} P(s' | s, a) [R(s, a, s') + \gamma U_i(s')] \]
- \( U_{i+1} \leftarrow BU_i \)
- Max norm: measures the “length” of a vector by the absolute value of its biggest component
  \[ ||U|| = \max_s |U(s)| \]
- \( ||BU_i - BU'_i|| \leq \gamma ||U_i - U'_i|| \) \( \gamma < 1 \)
- \( ||BU_i - U|| \leq \gamma ||U_i - U|| \)
- Total utilities of all states are bounded:
- Maximum initial error:
  \[ ||U_0 - U|| \leq \frac{2R_{max}}{(1 - \gamma)} \]
Utility Error Bound

- Error at step 0: \[ \|U_0 - U\| \leq \frac{2R_{\text{max}}}{(1 - \gamma)} \]
- Error at step N: \[ \|U_N - U\| = \gamma^N \cdot \frac{2R_{\text{max}}}{(1 - \gamma)} < \varepsilon \]
- Steps for error below \( \varepsilon \):
  \[ N = \frac{\log\left(\frac{2R_{\text{max}}}{\varepsilon(1-\gamma)}\right)}{\log\left(\frac{1}{\gamma}\right)} \]
Iterations required to guarantee an error of at most $\epsilon$, $\epsilon = cR_{\text{max}}$.

The maximum error of the utility estimates and the policy loss.
**Bandit Problem**

- Unknown probability distribution of winnings
- Gambler must exploit best action to obtain rewards
- Gambler must explore previous unknown states and actions to gain information
Partially Observable MDPs (POMDPs)

• Fully observable environment: the agent always knows which state it is in

• Partially observable environment:
  – The agent does not necessarily know which state it is in
  – It cannot execute the action $\pi(s)$ recommended for that state
  – How to calculate the utility of a state $s$?
  – How to make optimal action in state $s$?
Definition of POMDPs

- A set of states $S$
- A set of actions $a$
- A start state
- Possibly one or more terminal states
- A **transition model** $T(s,a,s')$ from $s$ to $s'$ with an action $a$. (or $P(s'|s,a)$)
- A **reward function** $R(s,a,s')$ from $s$ to $s'$ with an action $a$. It may be positive or negative
- A **sensor model** $P(e|s)$: the probability of perceiving evidence $e$ in state $s$
- Possibly a discount factor $0 \leq \gamma \leq 1$
- The optimal action depends only on the agent’s current belief state
- Optimal policy: $\pi^*(b)$
MDP Summary

• Offline algorithm for solving MDPs
  – Value iteration
  – Policy iteration
  – Linear programming
• Bellman equation
• Convergence
• Partially Observable MDPs (POMDPs)
MDP Summary

- Value iteration: compute the optimal values of states, by iterative updates until convergence
- Policy iteration:
  - Policy evaluation: compute the values of states under a specific policy
  - Policy improvement: update policy using one-step look-ahead with resulting converged utilities as future values
- Policy iteration tends to outperform value iteration
- Policy extraction: determine a policy from state values
  - Value iteration: compute an optimal policy from the optimal state values
  - Policy iteration: compute the best policy for the currently estimated state values
Machine Learning

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning
- Deep Learning
Google DeepMind DQN plays Atari Breakout

Agent57: Outperforming the human Atari benchmark

2015
Capture the Flag

Gamemode: Harvester
Map: Future Crossings
3 agents vs. 3 agents

Human-level performance in Quake III Arena Capture the Flag using reinforcement learning
AlphaStar

AlphaStar: Mastering the Real-Time Strategy Game StarCraft II
AI Learns to Park - Deep Reinforcement Learning

Unity ML-Agents Toolkit
Emergent Tool Use from Multi-Agent Interaction

We've observed agents discovering progressively more complex tool use while playing a simple game of hide-and-seek. Through training in our new simulated hide-and-seek environment, agents build a series of six distinct strategies and counterstrategies, some of which we did not know our environment supported. The self-supervised emergent complexity in this simple environment further suggests that multi-agent co-adaptation may one day produce extremely complex and intelligent behavior.
Reinforcement Learning

- Offline planning
  - Markov decision process
  - Agents have full knowledge of both the transition function and the reward function
  - Precompute optimal actions in the world encoded by the MDP without ever actually taking any actions

- Online planning
  - Markov decision process
  - An agent may not know the reward function or the transition model
  - An agent must try actions and states out to learn and get feedbacks
  - The agent uses feedbacks to estimate an optimal policy through a process known as reinforcement learning
Reinforcement Learning

• Still a Markov decision process (MDP)
  – A set of states $S$
  – A set of actions $A$
  – A transition model $T(s,a,s')$ or $P(s'|s,a)$
  – A reward function $R(s,a,s')$
• Still looking for a policy $\pi(s)$
• Agents don’t know $T$ and $R$
Reinforcement Learning

- Agents actively learn from their own experience
- Receive feedback in the form of rewards
- Agent’s utility is defined by the reward function
- Agents must act to maximize expected rewards
- All learning is based on observed samples of outcomes
- With deep learning, RL systems do a lot more!
Reading and Next Class

- MDPs: AIMA 17-3-17.5
- Reinforcement Learning: AIMA 22.1
- Next:
  - Reinforcement Learning: AIMA 22.1 – 22.2