Artificial Intelligence Programming

Markov Decision Processes

Chris Brooks

Department of Computer Science
University of San Francisco
Previously, we’ve talked about:

- Making one-shot decisions in a deterministic environment
- Making sequential decisions in a deterministic environment
  - Search
  - Inference
  - Planning
- Making one-shot decisions in a stochastic environment
  - Probability and Belief Networks
  - Expected Utility

What about sequential decisions in a stochastic environment?
Recall that the expected utility of an action is the utility of each possible outcome, weighted by the probability of that outcome occurring.

More formally, from state $s$, an agent may take actions $a_1, a_2, \ldots, a_n$.

Each action $a_i$ can lead to states $s_{i1}, s_{i2}, \ldots, s_{im}$, with probability $p_{i1}, p_{i2}, \ldots, p_{im}$.

$$EU(a_i) = \sum p_{ij} s_{ij}$$

We call the set of probabilities and associated states the state transition model.

The agent should choose the action $a'$ that maximizes EU.
We can extend this idea to sequential environments.

Problem: How to determine transition probabilities?
- The probability of reaching state $s$ given action $a$ might depend on previous actions that were taken.

The Markov assumption says that state transition probabilities depend only on a finite number of parents.

Simplest: a first-order Markov process. State transition probabilities depend only on the previous state.
- This is what we’ll focus on.
We’ll also assume a *stationary distribution*. This says that the probability of reaching a state $s'$ given action $a$ from state $s$ with history $H$ does not change. Different histories may produce different probabilities given identical histories, the state transitions will be the same. We’ll also assume that the utility of a state does not change throughout the course of the problem.
Utility will depend on a sequence of states $s_1, s_2, \ldots, s_n$.

Let’s assign each state a reward $R(s_i)$.

Agent wants to maximize the sum of rewards.

We call this formulation a Markov decision process.

- An initial state $s_0$
- A discrete set of states and actions
- A transition model: $T(s, a, s')$ that indicates the probability of reaching state $s'$ from $s$ when taking action $a$.
- A reward function: $R(s)$
Agent moves in the “intended” direction with probability 0.8, and at a right angle with probability 0.2

What should an agent do at each state to maximize reward?
Since the environment is stochastic, a solution will not be an action sequence.

Instead, we must specify what an agent should do in any reachable state.

We call this specification a **policy**

- “If you’re below the goal, move up.”
- “If you’re in the left-most column, move right.”

We denote a policy with $\pi$, and $\pi(s)$ indicates the policy for state $s$. 
We can compare policies according to the expected utility of the histories they produce.

The policy with the highest expected utility is the *optimal policy*.

Once an optimal policy is found, the agent can just look up the best action for any state.
As the cost of being in a nonterminal state changes, so does the optimal policy.

Very high cost: Agent tries to exit immediately, even through bad exit.

Middle ground: Agent tries to avoid bad exit

Positive reward for nonterminals: Agent doesn’t try to exit
In solving an MDP, an agent must consider the value of future actions.

There are different types of problems to consider:

- Horizon - does the world go on forever?
  - Finite horizon: after $N$ actions, the world stops and no more reward can be earned.
  - Infinite horizon; World goes on indefinitely, or we don’t know when it stops.
    - Infinite horizon is simpler to deal with, as policies don’t change over time.
More on reward functions

We also need to think about how to value future reward.

$100 is worth more to me today than in a year.

We model this by discounting future rewards.

\[ U(s_0, s_1, s_2, s_3, \ldots) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \gamma^3 R(s_3) + \ldots, \gamma \in [0, 1] \]

If \( \gamma \) is large, we value future states

if \( \gamma \) is low, we focus on near-term reward

In monetary terms, a discount factor of \( \gamma \) is equivalent to an interest rate of \( (1/\gamma) - 1 \).
Discounting lets us deal sensibly with infinite horizon problems.

- Otherwise, all EUs would approach infinity.

Expected utilities will be finite if rewards are finite and bounded and $\gamma < 1$.

We can now describe the optimal policy $\pi^*$ as:

$$\pi^* = \arg \max_{\pi} EU \left( \sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi \right)$$
How to find an optimal policy?

We’ll begin by calculating the expected utility of each state and then selecting actions that maximize expected utility.

In a sequential problem, the utility of a state is the expected utility of all the state sequences that follow from it.

This depends on the policy $\pi$ being executed.

Essentially, $U(s)$ is the expected utility of executing an optimal policy from state $s$. 
Notice that utilities are highest for states close to the +1 exit.
The utility of a state is the immediate reward for that state plus the expected discounted utility of the next state, assuming that the agent chooses the optimal action.

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U(s')$$

This is called the Bellman equation.

Example:

$$U(1, 1) = -0.04 + \gamma \max (0.8U(1, 2) + 0.1U(2, 1) + 0.1U(1, 1))$$
The Bellman equation is the basis of dynamic programming.

In an acyclic transition graph, you can solve these recursively by working backward from the final state to the initial states.

Can’t do this directly for transition graphs with loops.
Since state utilities are defined in terms of other state utilities, how to find a closed-form solution?

We can use an iterative approach:

△ Give each state random initial utilities.
△ Calculate the new left-hand side for a state based on its neighbors’ values.
△ Propagate this to update the right-hand-side for other states,
△ Update rule:

\[ U_{i+1}(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U_i(s') \]

This is guaranteed to converge to the solutions to the Bellman equations.
Value Iteration algorithm

\[
\begin{align*}
\text{do} & \\
\text{for } s \text{ in states} & \\
U(s) &= R(s) + \max_{a} T(s, a, s') \ U(s') \\
\text{until} & \\
\text{all utilities change by less then delta}
\end{align*}
\]

where \( \delta = error \times (1 - \gamma) / \gamma \)
20-18: Value Iteration example
20-19: Discussion

6 Strengths of Value iteration
   ▶ Guaranteed to converge to correct solution
   ▶ Simple iterative algorithm

6 Weaknesses:
   ▶ Convergence can be slow
   ▶ We really don’t need all this information
   ▶ Just need what to do at each state.
Policy iteration helps address these weaknesses.

Searches directly for optimal policies, rather than state utilities.

Same idea: iteratively update policies for each state.

Two steps:
- Given a policy, compute the utilities for each state.
- Compute a new policy based on these new utilities.
Policy iteration algorithm

Pi = random policy vector indexed by state
do
    U = evaluate the utility of each state for Pi
    for s in states
        a = find action that maximizes expected utility for that state
        Pi(s) = a
    while some action changed
20-22: Policy iteration example
Policy and value iteration assume a fully observable environment.

- Partially observable MDPs deal with partially observable environments.
- Also assume a fixed goal/reward structure.
- Very effective in finding optimal action sequences.
- Assumes that transition model $T(s, a, s')$ is known.

Wednesday: How to learn $T$ and an optimal policy.