COS 435/ECE 433 Week 2 Precept Notes

February 7, 2025

In this precept, we are going to review the Markov Decision Process and go over 1) Bandits 2) Model-Predictive Control

1 Review

In this class, we are primarily concerned with methods for solving *Markov Decision Process* (MDP). As such, it is important to keep in mind the definition(s) of MDPs:

Definition 1.1 (Markov Decision Process, Finite-Horizon Setting)**.** A *finite-horizon MDP* is a tuple $(\mathcal{S}, \mathcal{A}, p, r, T)$, where

- $\mathcal{S}$ is the set of *states*,
- $\mathcal{A}$ is the set of *actions*,
- p is the *transition dynamics*, such that $p(s'|s, a)$ is the distribution over the next state s', given the agent took action a from state s.
- r is the *reward function*, such that $r(s, a)$ is the reward from taking action a from state s.
- T is the *rollout horizon*, i.e. the number of actions an agent takes in every rollout. ◇

Definition 1.2 (Markov Decision Process, Discounted Infinite-Horizon Setting)**.** A *finite-horizon MDP* is a tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$, where $\mathcal{S}, \mathcal{A}, p, r$ follow the same definitions as in the finite-horizon setting, and γ is the *discount factor*. ◇

These definitions capture the notion of a particular task that an agent is trying to solve, as well as a way of measuring how well an agent is solving a task at a particular timestep (as expressed by the reward function). To quantify how well an agent is solving a task overall, we define the notion of a *(discounted) return*:

Definition 1.3 (Return)**.** Let π be a policy (a way of acting in the MDP), defined as a distribution over actions conditioned on a state. Then, the return of the policy π from a state s, denoted $V^\pi(s)$, is defined as

$$V^\pi(s) \coloneqq \mathbb{E}\left[\sum_{t=0}^{T-1} r(s_t, a_t)\right]$$

in the finite-horizon setting, or

$$V^\pi(s) \coloneqq \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$$

in the infinite-horizon setting. ◇

2 Bandit Problem

The Bandit Problem is a fundamental problem in reinforcement learning (RL) and decision-making under uncertainty. It models a scenario where an agent repeatedly chooses from a set of actions (often called "arms"), receiving a reward for each choice, and the goal is to maximize the total reward over time. In the lecture, we explore three strategies for solving the bandit problem:

1. Greedy
2. ε-greedy
3. Upper confidence bound (UCB)

2.1 Greedy strategy

The greedy strategy always chooses the action (or arm) with the highest estimated reward. We take the empirical mean, the average reward at arm a when it's been taken before.

Advantages of the greedy approach.

- Simple to implement.
- Efficient when the optimal action is well known.

Limitations of the greedy approach.

- No mechanism for exploration (If an initial estimate is incorrect, the algorithm may never discover better actions.)
- Can get stuck in a local optimum.

$$\hat{r}(a) = \frac{\text{sum of rewards when you took action } a \text{ before}}{\text{num. times taken action } a}$$

At each time step t, take the arm with greatest estimated reward:

$$\underset{a}{\text{argmax}}\, \hat{r}(a). \tag{1}$$

2.2 ε-Greedy

With probability $1-\varepsilon$, take the action $\text{argmax}_a\, \hat{r}(a)$. With probability ε, choose the action randomly.

Algorithm 1 ε-Greedy Algorithm

Input: Exploration rate ε, number of arms K
$Q(a) \leftarrow 0, N(a) \leftarrow 0$ for all arms a
for *each time step* $t = 1, 2, \ldots$ **do**
  With probability $1-\varepsilon$, select $a_t = \arg\max_a Q(a)$
  With probability ε, select a random arm a_t
  Observe reward r_t
  Update counts: $N(a_t) \leftarrow N(a_t) + 1$
  Update value estimate: $Q(a_t) \leftarrow Q(a_t) + \frac{1}{N(a_t)}(r_t - Q(a_t))$
end

2.3 Boltzmann / Softmax

How can we design a strategy so that we sample the more promising actions more frequently, but still sample the less promising actions with some frequency (on the off-chance that they are the best)?

$$\pi(a) = \textsc{SoftMax}(\hat{r}(a)) = \frac{e^{\frac{1}{\beta}\hat{r}(a)}}{\sum_{a' \in} e^{\frac{1}{\beta}\hat{r}(a')}}. \tag{2}$$

Intuition behind the temperature $\beta > 0$. For large β, we sample uniformly. For small β, we recover the greedy strategy.

Compared with ε-greedy, this approach samples actions proprtional to how promising they seem. However, this approach uses a fixed degree of exploration the entire time. If there's an action that you've sampled dozens of times and it's always been bad, this algorithm will nonetheless continue to sample it.

2.4 Upper Confidence Bound (UCB)

Let $N_a(t)$ be the number of times you've taken action a before time t. The upper confidence bound strategy says to take the action:

$$\operatorname*{argmax}_a \hat{r}(a) + \sqrt{\frac{2\log(t)}{N_a(t)}}. \tag{3}$$

The UCB algorithm follows the principle of "optimism in the face of uncertainty." The first term, $\hat{r}(a)$, ensures that the algorithm exploits actions that have yielded high rewards in the past. The second term, $\sqrt{\frac{2\log(t)}{N_a(t)}}$

Exploration-Exploitation Trade-off

- **Exploitation:** If an action a has been chosen many times and consistently yields high rewards, its exploration bonus diminishes, and it is primarily selected based on $\hat{r}(a)$.
- **Exploration:** If an action has been chosen rarely, its confidence bound remains high, prompting the algorithm to explore it further.
- Over time, UCB ensures that suboptimal actions are eventually discarded while guaranteeing sufficient exploration to identify the best action.

3 The Cross Entropy Method

The Cross Entropy Method (CEM) is a stochastic optimization technique used for solving difficult optimization and reinforcement learning problems. It is based on iteratively updating a probability distribution over candidate solutions to focus sampling on the most promising regions of the search space.

The cross entropy method (CEM).

for $t = 0, 1, \cdots, T$ **do-**

Sample $x_t^{(0)}, \cdots, x_t^{(k)} \sim p(x)$

Evaluate $f(x_t^{(i)})$ for $i = 0, \cdots, k$

Choose samples $x_t^{(0)}$ with highest values of $f(\cdot)$

Fit $p(x)$ to these samples: $\max_{\mu,\Sigma} \sum_{i \in \text{Best}} \log p(x_t^{(i)})$

return μ =0

The key idea behind CEM is to iteratively refine the sampling distribution to increasingly focus on promising regions of the search space. Initially, the distribution is broad, but as iterations progress, it becomes more concentrated around high-quality solutions. This method balances exploration and exploitation by adjusting the distribution based on observed function values.

4 Model-Predictive Control

Model-Predictive Control (MPC) is an advanced control strategy that uses a dynamic model of the system to predict future behavior. At each time step, it solves an optimization problem to find the control input sequence that minimizes a performance cost while satisfying constraints.

- **Predictive Model:** MPC assumes access to a dynamics model

$$m(s_t, a_t) \to s_{t+1} \quad \text{or} \quad p(s_{t+1} \mid s_t, a_t).$$

- **Finite-Horizon Optimization:** At each time step, it solves

$$\max_{a_{0:T}} \mathbb{E}\Big[\sum_{t=0}^{T} \gamma^t \, r(s_t, a_t)\Big],$$

  where γ is a discount factor and $r(\cdot)$ is a reward function.

- **Receding Horizon:** Only the first action from the optimal sequence is executed, then the problem is re-solved at the next step with the new state.

Dimension of the Action Sequence If each action a_t is in $\mathbb{R}^d$, then the decision variable $a_{0:T}$ is in $\mathbb{R}^{(T+1)d}$. This can become high-dimensional quickly.

Why Replan Each Time Step?

- **Adaptation to Reality:** Real systems deviate from predictions (due to stochasticity, modeling errors, etc.). By replanning, we can adjust actions based on actual observed states.

- **Error Correction:** Early inaccuracies in the model or environment can be mitigated with a new plan rather than committing to a potentially suboptimal sequence.

Why Not Use Gradient Descent?

- **Non-Differentiability:** In many cases, the model or reward function may not be differentiable (e.g., simulations with discontinuities, sampling-based methods).
- **Complicated Landscapes:** Even if differentiable, the optimization landscape can be highly non-convex, making gradient-based approaches difficult to tune or converge.

Choosing the Horizon T

- **Large** T: Better long-term planning, but more computationally expensive and often noisier.
- **Small** T: Less computation, but potentially more myopic (neglecting long-term consequences).

Open-Loop vs. Closed-Loop

- **Open-Loop:** Pre-committing to an action sequence $a_{1:T}$ without adjusting for future states.
- **Closed-Loop (MPC):** Recompute actions (or "replan") at each step, using the current state. However, planning itself often assumes open-loop future actions; this can lead to suboptimal behavior in very stochastic domains.

Challenges

- **High-Dimensional Search Space:** $(T+1) \times |\mathcal{A}|$.
- **Long-Horizon Tasks:** Short horizons can ignore significant long-term effects.
- **Model Accuracy:** Performance deteriorates if the model does not accurately capture system dynamics.

5 Discussion Problems

1. Decide whether the following scenarios are better modeled as a contextual bandit problem or a general RL problem:

   (a) Recommendation systems (e.g. Netflix)

   (b) A hypothetical system for recommending medical interventions over the course of a treatment plan.

   [**Recommender systems**] Recommender systems are often modeled as contextual bandit problems, especially when the goal is to maximize immediate engagement or satisfaction. In a contextual bandit setting, the system selects actions (e.g., recommendations) based on the current context (e.g., user profile, browsing history) without explicitly considering the long-term impact of these actions. This model is suitable for scenarios where the immediate reward is the primary focus, and the effect of the current action on future states is either negligible or not considered.

   [**Recommending medical interventions over a treatment plan**] In this case, the problem is inherently sequential and involves significant considerations for the future state of the patient. Each intervention has consequences that affect the patient's health, which influences the appropriateness of

future interventions. This is a classic scenario for general RL, where each action (medical intervention) affects the environment (patient's health state) in a way that must be considered for future decisions. The objective is not just to maximize the immediate outcome of an intervention but to optimize the patient's health over the entire treatment plan. This requires a model that can account for the temporal dependencies between actions and their long-term effects on the state, making general RL the suitable approach.

2. Formulate finding the shortest path between two nodes on a given graph $G = (V, E)$ as an RL problem. Give three equivalent ways: (1) letting the state be the current node, (2) having the state maintain a stack of visited states (3) similar state space as (2), but computing the reward using the stack only at termination (sparse reward).

   [(1) State = Current Node]

   - States: Each state represents the current node the agent is located at within the graph G.
   - Actions: Actions correspond to moving from the current node to an adjacent node.
   - Transitions: Determined by the graph structure; moving from one node to another changes the state based on the edge E between them.
   - Reward: A negative reward (e.g., -1) for each move encourages the agent to find the shortest path (recall that in RL we aim to maximize total reward). A large positive reward could be given upon reaching the destination node, incentivizing completion.
   - Horizon: We can set the horizon to $|V|$, the number of nodes in the graph, since no shortest path will visit a node twice.

   [(2) State = Stack of Visited States]

   - States: State maintains the history of visited states so far.
   - Actions: Actions correspond to moving from the current node to an adjacent node.
   - Transitions: Determined by the graph structure; moving from one node to another adds a new node to the state stack.
   - Reward: A negative reward (e.g., -1) for each move encourages the agent to find the shortest path (recall that in RL we aim to maximize total reward). A large positive reward could be given upon reaching the destination node, incentivizing completion.
   - Horizon: We can set the horizon to $|V|$, the number of nodes in the graph, since no shortest path will visit a node twice.

   [(3) State = Stack of Visited States, Sparse Rewards]

   - States: State maintains the history of visited states so far.
   - Actions: Actions correspond to moving from the current node to an adjacent node.
   - Transitions: Determined by the graph structure; moving from one node to another adds a new node to the state stack.
   - Reward: Reward is only given once the stack length reaches $|V|$ or when the agent reaches the goal state, whichever one comes first. The reward is computed using the stack (and computed in a way to be equivalent with the earlier two formulations).

- Horizon: We can set the horizon to $|V|$, the number of nodes in the graph, since no shortest path will visit a node twice.

Remark 5.1. Observe that there are multiple ways to formulate the same MDP. However, as we will learn throughout this course, the way we formulate the MDP can have a huge impact on how easy/difficult it is to find an optimal policy in an MDP. In the example above, note that the first formulation is straightforward and models the problem directly in terms of the graph's structure. The continuous reward signal (negative reward at each timestep) helps guide the agent by providing immediate feedback on its actions, making the learning process potentially easier. On the other hand, the latter two formulations consider a much larger state space (with $n!$ possible states worst case, depending on the graph), and the final formulation has extremely sparse reward. ⌟

3. (Markov chain, return computation practice) Consider an MDP whether the state space is structured as a binary tree of depth d with two actions, left and right (which traverses down the tree appropriately). Assume the agent receives a reward of 1 if and only if it takes an action leading to a particular fixed leaf. Compute the return of a policy that chooses between left and right uniformly at random.

   [**Expected Reward**] We compute the distribution of states at every timestep, considering that from any given state, the agent has a $1/2$ chance of moving left or right. At each depth k of the tree, the agent can be in 2^k possible states (nodes) since the tree branches out by a factor of 2 at each level. The probability of being in any particular state at depth k is $(1/2)^k$ because to arrive at that state, the agent must make a particular sequence of k choices, each with a probability of $(1/2)$. To compute the expected return, we aggregate the probabilities of reaching the rewarding state over all possible paths. However, since there's only one rewarding path and the reward is located at one leaf, the expected return is $(1/2)^d$. This is because regardless of the path distribution across the tree, only the path leading to the reward contributes to the expected return.

   Remark 5.2. Observe that an MDP, together with a fixed policy, induces a particular Markov chain. ⌟

   Remark 5.3 (On Reward Choice)**.** This example underscores how crucial the choice of reward and state space structure is in reinforcement learning. In a binary tree with a single reward at one leaf and a uniform random policy, the expected return diminishes exponentially with the depth of the tree. This setup illustrates a near-worst-case scenario for an RL agent, where the environment's structure and reward distribution make learning efficient policies highly challenging without additional strategies or information. It highlights the balance between exploration and exploitation and the importance of reward shaping or structure to guide learning in complex environments. ⌟