1 Introduction

In this lab, you will be building reinforcement learning (RL) agents for partially-observable worlds: i.e., worlds in which states are unobservable, but noisy sensors provide information suggestive of the underlying states. In a decision-making context, such worlds are modeled by **partially observable Markov decision processes (POMDPs)**, a generalization of MDPs. To solve POMDPs, agents maintain beliefs about the underlying state; these beliefs are updated over time, as noisy observations are collected. A POMDP can be reduced to an MDP via a reduction in which these beliefs are the MDP's state, at which point the POMDP can be solved using **belief-state reinforcement learning**, as you will see in this lab.

You will be learning about belief-state RL in the context of the **Lemonade Stand Game**,[1] a generalization of a famous game of electoral politics studied by Hotelling.[2] In Hotelling's game, two ice cream vendors set up shop on a linear beach. A continuum of beachgoers uniformly spreads themselves out on the beach, and each beachgoer buys ice cream from the nearest vendor. So rather than spread themselves out, the vendors creep towards the center, because each time they do, they steal a bit more of the customer base from the other vendor. Ultimately, both vendors find themselves at the dead center. The Lemonade Stand game is played by three lemonade stand vendors on a circular beach.

2 Setup

You can find and clone the stencil code for Lab 4 here. The stencil code includes:

- `belief_bayesian_agent.py`*
- `belief_q_learning_agent.py`*
- `cslogin.txt`*
- `mysterious_agents.py`*

The star annotations indicate which files you will edit during this lab. The others are purely support code and/or already-implemented opponent bots for the simulations.

Please be sure to read the `README.md`. Here is an abridged version of the setup guide described in there:

1. Clone the repository with the command
   `git clone https://github.com/brown-agt/lab-stencils.git`
2. Create a python virtual environment and activate it. Be sure to use `python 3.10` or higher.
3. Run `pip install --upgrade agt_server`

IMPORTANT: The `README.md` will also contain helpful methods for implementing your agent.

3 The Lemonade Stand Game

The Lemonade Stand Game is a three-player game, in which the players are lemonade stand proprietors trying to maximize their revenue (i.e., sell as much lemonade as possible).

The three players have been issued permits to set up their lemonade stands at one of twelve possible evenly-spaced spots on a circular beach around a lake—imagine the possible lemonade-stand locations as the hours on a clockface.

Every morning, 12 beachgoers spread themselves out evenly on the beach, one at each hour. Over the course

[1] Martin Zinkevich, Michaels Bowling and Wunder. Solving Unsolvable Games, *ACM SIGecom Exchanges*, 10(1):35–38, 2011.
[2] Harold Hotelling. Stability in Competition. *Economic Journal*, 39:41–57, 1929.

of the day, they all buy two cups of lemonade, one from the closest stand to their left, and another from the closest stand to their right. If there is one lemonade stand at a beachgoer's location, she buys both cups from that one stand. If there are two lemonade stands at a beachgoer's location, she buys one from each. If all three lemonade stands are at her location, she buys from only two of them, selecting those two at random.

A player's payoff equals the total number of cups of lemonade it sells. (When two or more players choose the same location, the proceeds are split evenly among all players at that location.)

The game is illustrated below, along with several examples.

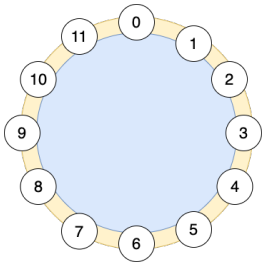

Figure 1: The setup for the Lemonade Stand Game. Players submit actions numbered between 0 and 11 to indicate their chosen location.

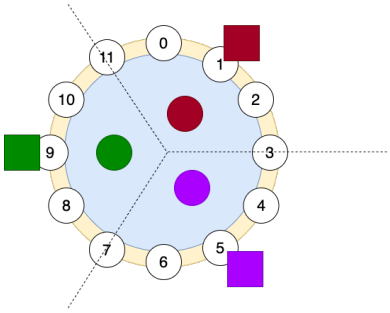

Figure 2: A possible outcome in the Lemonade Stand Game. The players are evenly spaced along the lake, so all earn payoffs of 8.

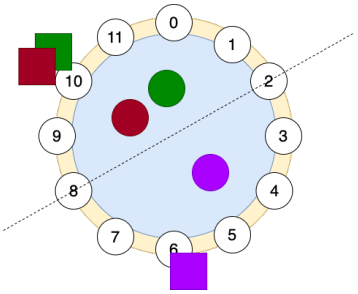

Figure 3: Another possible outcome in the Lemonade Stand Game. The Green and Red players both choose 10, while the Purple player chooses 6. Together, the Red and Green players earn 12, which they split evenly, while the Purple player earns 12.

Question: Imagine you are the third player in the Lemonade Stand Game, with your opponents' moves as shown in Figure 4. What is your best response? Is there a unique answer? What is your best possible payoff?

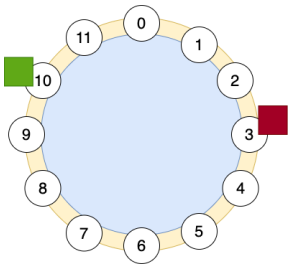

Figure 4: If you are the third player facing these two opponents, what would you do?

As you can see, your payoff in this game depends heavily on what your opponents do. You might even reason that without any idea of how your opponents will play, it is impossible to design a meaningful strategy![3]

Question: Think about some possible strategies for this game. If you know how your opponents tend to play, how would you respond to their strategies? How would you try to learn how your opponents are playing?

4 Partially Observable Markov Decision Processes

Recall that an MDP is characterized by:

- a set of states S
- a set of actions A
- a Markov transition function $T : S \times A \times \Delta(S)$ that gives the probability of transitioning from state s to state s' via action a
- a reward function $R : S \times A \to \mathbb{R}$ that gives reward $R(s, a)$ for taking action a at state s
- an initial state, or a probability distribution over initial states

A **partially observable Markov decision process (POMDP)** includes all of the above, and in addition:

- a set of observations O
- an observation function: $Z : S \times A \times \Delta(O)$ that gives the probability of observing o after taking action a and transitioning to state s'

Since an agent cannot observe its state in a POMDP, it encodes its uncertainty about the state in a so-called **belief state**, which is a probability distribution over states. This belief state is updated via Bayes' rule based on the noisy observations the agent collects as it takes actions and explores its environment. Suppose the current belief state is $b(s)$, and the agent takes action a and receives observation o. The new belief state $b(s')$ is computed as follows:

$$\begin{aligned} b(s') &= \Pr(s' \mid b, a, o) \\ &= \eta \Pr(o \mid b, a, s') \Pr(s' \mid b, a) && \text{Bayes' rule} \\ &= \eta \, Z(o \mid a, s') \sum_{s \in S} T(s' \mid s, a) \, b(s) && (1) \end{aligned}$$

Here η is a normalization constant, which ensures that $\sum_{s' \in S} b(s') = 1$.

So long as beliefs are computed recursively via this update rule, the belief state is a sufficient statistic for the full history of actions and observations. As a result, the transition function from one belief state to another

[3] Garry Kasparov, chess grandmaster and perhaps the greatest chess player of all time, claims that this was part of (or perhaps even entirely) the reason he ultimately lost his match to IBM's Deep Blue. Just like pitchers know the batters they face, Kasparov always studied the past plays of his other opponents. But he was not permitted access to Deep Blue's algorithmic workings, so when the machine did something unusual, Kasparov was caught off guard.

is Markov: i.e., the current belief state encodes all information relevant to the next belief state. In this way, a POMDP can be reduced to a belief-state MDP, whose states are precisely these belief states.

5 Belief-State Q-Learning

Even when the underlying POMDP's state space is discrete, the state space in a belief-state MDP is still continuous, because belief states are probability distributions. Thus, more sophisticated RL algorithms than tabular Q-learning (Lab 3) are required to solve this MDP. Indeed, in this lab, you will implement **belief-state** Q-learning, which represents the Q-function using a linear function approximator instead of a table.

Given a function $f : \mathbb{R}^m \to \mathbb{R}^n$, a **linear function approximator** models f as a linear function of **features** of the inputs. These features are an alternative representation of the inputs, highlighting its key "features." Given an input $\boldsymbol{x} \in \mathbb{R}^m$ and a feature vector $\phi(\boldsymbol{x}) \in \mathbb{R}^d$, with $d \ll m$ (usually), $\hat{f}(\boldsymbol{x}) = \boldsymbol{w}^T\phi(\boldsymbol{x}) = \sum_{i=1}^{d} w_i\phi_i(x)$ is a linear function approximator for f, where $\boldsymbol{w} \in \mathbb{R}^d$ is a parameter vector called the **weights**.

Applying this logic, we define a linear approximator $\hat{Q}(s, a; \boldsymbol{w})$ for the Q-function of an MDP as follows:

$$\hat{Q}(s, a; \boldsymbol{w}) = \boldsymbol{w}^T\phi(s, a)$$

Note that $\phi(s, a)$ is the concatenation of $\phi(s)$, a feature representation of the state and a one-hot encoding of the action a.[4] For example, if there are d features and c actions, then $\phi(s, a)$ is a vector of length $c + d$.

The goal of Q-learning with (even non-linear) function approximation is to learn a parameterized Q-function that satisfies Bellman's equation:

$$\hat{Q}(s, a; \boldsymbol{w}) = \mathbb{E}_{s'}\left[r + \max_{a'} \hat{Q}(s', a'; \boldsymbol{w})\right]$$

But during learning, the agent sees only, say T, sample trajectories (s, a, r, s') through its environment, so it cannot compute this expectation. As a result, we treat this problem like a regression problem, and aim to minimize (half) the (mean squared) empirical **temporal difference error**:

$$L(\boldsymbol{w}) = \frac{1}{T}\sum_{(s,a,r,s')} \frac{1}{2}\delta^2$$

where

$$\delta = r + \max_{a' \in A} \hat{Q}(s', a'; \boldsymbol{w}) - \hat{Q}(s, a; \boldsymbol{w})$$

As is usual when solving a regression problem whose loss function is mean squared error, we follow the negative gradient of the loss. Assuming the target $r + \max_{a' \in A} \hat{Q}(s', a'; \boldsymbol{w})$ is constant, $\nabla_{\boldsymbol{w}} L(\boldsymbol{w}) = -\nabla_{\boldsymbol{w}} \hat{Q}(s, a; \boldsymbol{w})$. Furthermore, assuming a linear function approximator, $\nabla_{\boldsymbol{w}} \hat{Q}(s, a; \boldsymbol{w}) = -\phi(s, a)$. Therefore, the rule for updating weights when using Q-learning with a linear function approximation is:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha\delta\phi(s, a) \ ,$$

where α is a learning rate.

Belief-state Q-learning is a method for solving POMDPs that incorporates these two ideas: building a belief-state MDP and solving that MDP using Q-learning and function approximation.

As games can be *very* hard to solve, it is common for researchers to develop agents that solve greatly simplified models of games of interest to them. A common simplification is to treat a game as a POMDP (i.e., as an

[4] A **one-hot** encoding of a categorical variable is a vector whose length is the number of categories, with all entries set to 0, except for a single 1, indicating the category.

optimization problem) even though it is not—it is a game! Not only does an agent's performance depend on other agents' decisions, environments in which multiple agents are learning simultaneously are not stationary!

In this lab, you will apply belief-state Q-learning to a greatly simplified model of the Lemonade Stand game, which we model as a POMDP. In particular, we assume the behaviors of the two agents' other than your own are stationary. In next week's lab, you will build an agent to play against agents built by other students. Those agents are very likely to be learning agents themselves, so the environment is not likely to be stationary.

6 Bayesian Filtering

The aforementioned belief-state update formula applied a technique called Bayesian filtering, by which better and better guesses about the value of a hidden state H are made as more and more evidence E is observed via Bayes' rule:

$$\Pr(H \mid E) = \frac{\Pr(E \mid H)\Pr(H)}{\Pr(E)}$$

Here, $\Pr(H \mid E)$ is called the posterior probability (or the updated belief); $\Pr(E \mid H)$ is called the likelihood; $\Pr(H)$ is called the prior probability (or belief); and $\Pr(E)$ is called the evidence.

Imagine you are building an agent to play the Lemonade game, and you know that one agent is sticky (he plays 0 forever), and the other is "mysterious." The mystery agent might be of type 3, or she might be of type 9; her true type is unknown. Moreover, observations are noisy: when her true type is 3, she plays 3 with probability 0.7 and 9 with probability 0.3; likewise, when her true type is 9, she plays 9 with probability 0.7 and 3 with probability 0.3. Assume that *a priori* you have no reason to believe she is one type or the other.

Task Imagine you observe the sequence of actions 9 3 3 3 for the mysterious agent after four rounds of play. Use Bayesian filtering to update your beliefs about her type. Work with paper and pencil.

7 Implementation Tasks

Task 1 Implement recursive Bayesian filtering as per Equation 1 for the mystery agent. Observe how your implementation of recursive Bayesian filtering learns to predict the mystery agent's strategy.

Task 2 Modify your agent to learn a best response to the prediction learned by recursive Bayesian filtering.
Hint Since the transition function is the identity for the mystery agent (i.e., the agent's behavior is static), fictitious play is sufficient for learning a best response.

Task 3 Modify the mystery agent in various ways. Here are some ideas for how to modify the mystery agent:

- Allow the mystery agent to randomize over more than two actions.
- Allow the mystery agent to cycle through actions deterministically: e.g., cycle from 3 to 9 and back.
- Build a Markov chain to represent the mystery agent's actions. For example, after playing 3, it might play 9 with probability 0.7 and 0 and 6, each with probability 0.3. And so on.

After each modification, observe how your implementation of recursive Bayesian filtering learns the mystery agent's strategy. Under what condition can Fictitious Play still learn a best response?
Hint While static behavior is sufficient, it is not necessary.

Task 4 Modify your agent to use Q-learning with a linear function approximator to learn a best response to a mystery agent whose behavior (i.e., transition function) is *stationary*: e.g., a Markov chain mystery agent.

Task 5 Finally, modify the mystery agent to play Tit-for-Tat. How does Q-learning perform?
Hint Is Tit-for-Tat a stationary strategy?