ECE433/COS435 Introduction to RL
Assignment 1: MDP
Spring 2025

Fill me in

Your name here.

Due February 10, 2025

Collaborators

Fill me in

Please fill in the names and NetIDs of your collaborators in this section.

Instructions

Writeups should be typeset in Latex and submitted as PDF. You can work with whatever tool you like for the code, but **please submit the asked-for snippet and answer in the solutions box as part of your writeup. We will only be grading your writeup.**

Grading. Questions 1-3 will collectively be worth total 50 points, and the coding assignment will also be worth 50 points, making the total score 100 points.

Question 1. Markov Chain (30 Points)

We have a two-state Markov chain with two states, s_1 and s_2. The probability of transitioning from s_1 to s_2 is $1-p$, and vice versa. We can summarize the transition probabilities in the table shown below.

$$P = \begin{bmatrix} p & 1-p \\ 1-p & p \end{bmatrix},$$

the value of $P_{i,j}$ indicates the probability of transitioning from state i to state j, for any $i, j \in [1, 2]$.

Question 1.a (5 Points)

Use the principle of induction[1] to show that

$$P^{(n)} = \begin{bmatrix} \frac{1}{2} + \frac{1}{2}(2p-1)^n & \frac{1}{2} - \frac{1}{2}(2p-1)^n \\ \frac{1}{2} - \frac{1}{2}(2p-1)^n & \frac{1}{2} + \frac{1}{2}(2p-1)^n \end{bmatrix}.$$

Solution

Your solution here...

Question 1.b (10 Points)

Expectation of State Occupancy.

- Compute the expected number of times the process is in s_1 after n transitions (including the starting state $t = 0$), starting from s_1.
- Compute the expected number of times the process is in s_2 after n transitions, starting from s_1.
- For $p \neq 0$, discuss how the expectations change as n approaches infinity.

Solution

Your solution here...

Question 1.c (5 Points)

Probability of First Visit.

- Compute the probability that the process visits s_2 for the first time on the k-th transition, given it starts in s_1. What happens if $k \to \infty$?

Solution

Your solution here...

Question 1.d (5 Points)

Conditional Expectations.

- Given that the chain is in s_2 at the n-th step, compute the conditional expectation of the number of visits to s_1 in the next m steps.

Solution

Your solution here...

[1] `https://en.wikipedia.org/wiki/Mathematical_induction`

Question 1.e (5 Points)

Expected rewards. When transitioning from one state to another, assume we receive a reward of 1 for reaching s_2 and -1 for reaching s_1.

- Compute the expected total reward after n transitions (i.e., the summation of rewards), starting from s_1.

Solution

Your solution here...

Question 2. Grid World Example (10 Points)

In this exercise, you will work with a simple reinforcement learning environment called "Gridworld." Gridworld is a 4x4 grid where an agent moves to reach a goal state. The agent can take four actions at each state (up, down, left, right) and receive a reward for each action. Moving into a wall (the edge of the grid) keeps the agent in its current state.

Grid Layout:

- The grid is a 4x4 matrix.
- Start state (S): Top left cell (0,0).
- Goal state (G): Bottom right cell (3,3).

The agent receives a reward of -1 for each action until it reaches the goal state.

Question 2.a (5 Points)

Formulate the problem as a Markov Decision Process (MDP). Define the states, actions, transition probabilities (assume deterministic transitions), rewards, and policy.

Solution

Your solution here...

Question 2.b (5 Points)

How many unique (deterministic) policies are there in total?

Solution

Your solution here...

Question 3. Bandits vs MDPs (10 Points)

Decide whether the following scenario is better modeled as a contextual bandit problem or a general RL problem, and explain the reason.

1. A doctor recommends a treatment plan to a patient. The doctor needs to prescribe medication each time the patient comes in.

Solution

Your solution here...

Question 4. Coding (50 points)

In this assignment, you will explore various strategies for balancing exploration and exploitation in a multi-armed bandit setup. Understanding how different strategies navigate the exploration-exploitation trade-off is crucial in reinforcement learning. You will implement and compare the following strategies:

1. Random Exploration
2. Epsilon-Greedy (with varying epsilon values)
3. Epsilon-Greedy with Decay (You can design their own decay schedule)
4. Upper Confidence Bound (UCB)

The question descriptions and code templates are provided in the provided `hw1.ipynb` notebook. Fill out the indicated code segments below after solving each problem.

Question 4.a (10 points)

The Bandit Class. We start by defining a "Bandit" class to represent each slot machine. This class will have methods to simulate pulling the machine's arm and updating our estimate of its win rate.

Solution

```
class Bandit:
    def __init__(self, p):
        self.p = p  # The true win rate
        self.p_estimate = 0.0 # The estimated win rate
        self.N = 0  # number of samples collected so far

    def pull(self):
        pass
        ### START CODE HERE ###
        # TODO: Simulate pulling the arm. Recall that True is also
   regarded as 1 and False is regarded as 0 ###
```

```
        # Hint: Use np.random.random() and compare with self.p to simulate a dense probability distribution
        ### END CODE HERE ###

    def update(self, x):
        pass
        ## START CODE HERE ###
        # TODO: Update the win rate estimate and increment N ###
        # Hint: Calculate the new estimate as a weighted average of the old estimate and the new sample `x`
        ### END CODE HERE ###
```

Question 4.b (10 points)

The Upper Confidence Bound (UCB) Function. The UCB algorithm selects the bandit to play based on both the estimated win rates and the uncertainty or variance in these estimates. Implement the UCB calculation in the function below.

Solution

```
def ucb(mean, n, nj):
    pass
    ### START CODE HERE ###
    # TODO: Implement the UCB formula to calculate the upper confidence bound ###
    # Hint: UCB = estimated win rate + exploration factor. The exploration factor can be calculated using np.sqrt(2 * np.log(n) / nj)
    ### END CODE HERE ###
```

Question 4.c (20 points)

Running the Experiment. We will now set up our experiment to compare how different exploration strategies affect the learning performance and rewards. Focus on implementing the selection logic for each strategy within the `run_experiment` function.

Solution

```
def run_experiment(bandit_probs, N, strategy="ucb", epsilon=0.1, decay_rate=0.99):
    bandits = [Bandit(p) for p in bandit_probs]
    rewards = np.zeros(N)
    total_plays = 0

    # Strategy selection
    if strategy == "random":
        selection_strategy = lambda: np.random.choice(len(bandits))
```

```
    elif strategy == "epsilon-greedy":
        selection_strategy = lambda: np.random.choice([np.random.choice(len(bandits)), np.argmax([b.p_estimate for b in bandits])], p=[epsilon, 1-epsilon])
    elif strategy == "epsilon-greedy-decay":
        selection_strategy = lambda: np.random.choice([np.random.choice(len(bandits)), np.argmax([b.p_estimate for b in bandits])], p=[current_epsilon, 1-current_epsilon]) if np.random.random() < current_epsilon else np.argmax([b.p_estimate for b in bandits])
    elif strategy == "ucb":
        selection_strategy = lambda: np.argmax([ucb(b.p_estimate, total_plays, b.N) for b in bandits])
    else:
        raise ValueError("Strategy not implemented!")

    # Initialization: Play each bandit once - No changes needed
    for j in range(len(bandits)):
        x = bandits[j].pull()
        bandits[j].update(x)
        total_plays += 1
        rewards[total_plays - 1] = x

    # Run the Main loop to play N rounds

    if strategy == "epsilon-greedy-decay":
        current_epsilon = epsilon

    for i in range(N):
        if strategy == "epsilon-greedy-decay":
            # Update current_epsilon using the decay rate
            current_epsilon *= decay_rate

        j = selection_strategy()
        x = bandits[j].pull()
        bandits[j].update(x)
        total_plays += 1
        rewards[i] = x

        if strategy == "epsilon-greedy-decay" and i < N - 1:
            # Optionally adjust epsilon for the next round
            current_epsilon = max(current_epsilon * decay_rate, 0.01)  # Ensure epsilon does not become too small

    # Plotting the results - No changes needed
    cumulative_rewards = np.cumsum(rewards)
    win_rates = cumulative_rewards / (np.arange(N) + 1)

    plt.plot(win_rates, label=strategy)

    # Displaying the estimated probabilities - No changes needed
    for b in bandits:
```

```
        print(f'Strategy {strategy}: Estimated probability of bandit, {b.p_estimate}')
    print('Total Reward:', rewards.sum())
```

Question 4.d (5 points)

Results. With the aforementioned `run_experiment` function, you should able to simulate these strategies. Let `bandit_probs = [0.2, 0.5, 0.75]` and `bandit_probs = [0.1, 0.6, 0.4]`, run experiments and attach the two plots that compare the performances of four strategies below. Note that you do not need to change other hyperparameters, such as rounds of play, and the script for making plots is already provided in the notebook.

For `bandit_probs = [0.2, 0.5, 0.75]`.

Solution

For `bandit_probs = [0.1, 0.6, 0.4]`.

Solution

Question 4.e (5 points)

Reflection and Analysis. After running the experiments, reflect on the performance of each strategy.

Solution