MTL 106 Assignment 1

Shreeraj Jambhale
2023CS50048

September 20, 2024

1 Theory

Setup

Alice and Bob play a Mind game in which each round they choose one of three strategies: aggressive, balanced, or defensive. The result of each round is a win, draw, or loss. A win gives +1 point, a draw gives +0.5 points, and a loss gives 0 points.

The probability of Alice winning, drawing, or losing in a round can or cannot depends on the current points of both players, where n_A is Alice's score and n_B is Bob's score. The probabilities are represented by the matrix $[p_1, p_2, p_3]$, where:

- p_1 is the probability of Alice winning,
- p_2 is the probability of a draw,
- p_3 is the probability of Bob winning.

Below is the matrix that shows the probabilities of each outcome based on the strategies of Alice and Bob:

	Attack Bob	**Balanced Bob**	**Defense Bob**
Attack Alice	$\left(\frac{n_B}{n_A+n_B}, 0, \frac{n_A}{n_A+n_B}\right)$	$\left(\frac{7}{10}, 0, \frac{3}{10}\right)$	$\left(\frac{5}{11}, 0, \frac{6}{11}\right)$
Balanced Alice	$\left(\frac{3}{10}, 0, \frac{7}{10}\right)$	$\left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$	$\left(\frac{3}{10}, \frac{1}{2}, \frac{1}{5}\right)$
Defense Alice	$\left(\frac{6}{11}, 0, \frac{5}{11}\right)$	$\left(\frac{1}{5}, \frac{1}{2}, \frac{3}{10}\right)$	$\left(\frac{1}{10}, \frac{4}{5}, \frac{1}{10}\right)$

2 Part 1: Probability, Expectation, Variance

2.1 Probability

```
memo = {}
def calc_prob(alice_wins, bob_wins):
    if (alice_wins, bob_wins) in memo:
        return memo[(alice_wins, bob_wins)]

    if alice_wins == 1 and bob_wins == 1:
        return 1
    p = 0
    total_wins = alice_wins + bob_wins - 1

    if alice_wins > 1:
        alice_prob = mod_divide(bob_wins, total_wins)
        p = mod_add(p, mod_multiply(alice_prob, calc_prob(alice_wins - 1,
        bob_wins)))

    if bob_wins > 1:
        bob_prob = mod_divide(alice_wins, total_wins)
        p = mod_add(p, mod_multiply(bob_prob, calc_prob(alice_wins,
        bob_wins - 1)))

    memo[(alice_wins, bob_wins)] = p
    return p
```

2.1.1 Base Case

$$\text{if alice\_wins} = 1 \text{ and bob\_wins} = 1 \text{ then probability} = 1$$

This is the terminating statement, which signifies when the code will come to end . as we are going from higher values of alice_wins and bob_wins so when these both reach 1 we need to stop the recursion as this is the starting point given in question statment.

2.1.2 Recursive Calculation

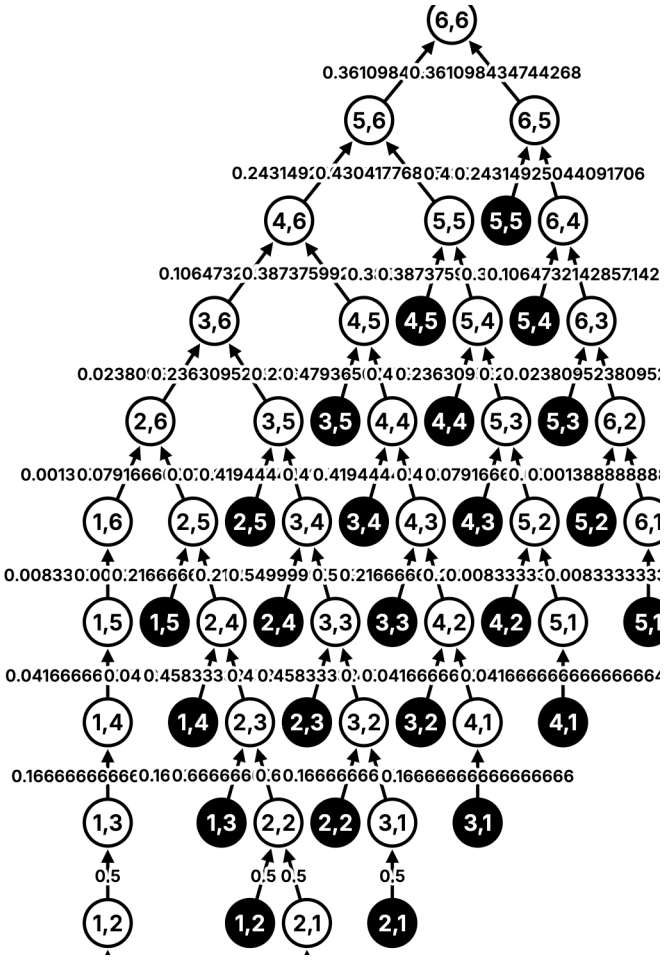
$$p = \left(\frac{\text{bob\_wins}}{\text{total\_wins}} \times \text{calc\_prob}(\text{alice\_wins} - 1, \text{bob\_wins})\right) + \left(\frac{\text{alice\_wins}}{\text{total\_wins}} \times \text{calc\_prob}(\text{alice\_wins}, \text{bob\_wins} - 1)\right)$$

basically the expression gives the probability of reaching till current points that is by 2 paths until the previous round is won by Alice or Bob, and the same is depicted in code by checking for both the probability ad adding them in p .

2.1.3 Memoization

Store p in memo[(alice_wins, bob_wins)]

This can be done in 2 ways either by using python inbuilt function importing from $functool$ and using @$cache$ decorator or can be implemented by creating a dictionary and storing the newly created probability values at different parameters. This helps in reducing the computation time by not computing same parameterised function again and again below is the pictorial depiction of the concept used for small input parameters.



2.2 Expectation

2.2.1 Theory

Let X_i be defined as follows:

$$X_i = \begin{cases} 1 & \text{if Alice wins round } i, \\ 0 & \text{if it's a draw,} \\ -1 & \text{if Alice loses.} \end{cases}$$

to find

$$\mathbb{E}\left[\sum_{i=1}^{T} X_i\right] = \sum_{i=1}^{T} \mathbb{E}[X_i]$$

where $T = 48$ (last two digits of the entry number).

Function

```
def calc_expectation(t):
    expectation = 0
    for x in range(1,t + 1):
        p_xy = calc_prob(x,t-x)
        expectation = mod_add(expectation, mod_multiply(2*x-t, p_xy))
    return expectation
```

2.2.2 Calculation Procedure

The expected value is:

$$\mathbb{E}[X_i] = (i - (t - i)) \cdot calc\_prob(i, t - i)$$

where t is the total games played and i is the games won by alice and so is $t - i$ the games won by bob. Hence the total Expectation is just summation of all such values from i =1 to i = t

1. Loop Through Possible Outcomes

The function iterates over all possible values of i from 1 to t. Here, i represents a specific outcome in the context of a probabilistic model where t is the total number of trials or rounds.

2. Calculate Probability

For each value of i, the function calculates the probability $p_{i(t-i)}$ of i wins and $t - i$ losses using the `calc_prob` function. This function likely computes the probability distribution of outcomes based on the given parameters.

$$p_{i,t-i} = \text{calc\_prob}(i, t - i)$$

3. Update Expectation

The function updates the variance using the formula for variance:

$$\text{expectation} = \text{expectation} + ((2i - t) \times p_{i,t-i})$$

It is observed that expectation is always yielding 0 as the function is symmetric so output is 0.

$$\mathbb{E}\left[\sum_{i=1}^{48} X_i\right] = 0$$

2.3 Variance

2.3.1 Theory

$$\text{Var}\left(\sum_{i=1}^{48} X_i\right)$$

as the $X_{i's}$ are not independent we can't swap the Variance and Summation and hence have to calculate $\mathbb{E}\left[\sum_{i=1}^{T} X_i^2\right]$ and at the end substract $\mathbb{E}\left[\sum_{i=1}^{T} X_i\right] = 0$ which gives us result $= \mathbb{E}\left[\sum_{i=1}^{T} X_i^2\right] = \sum_{i=1}^{T} \left[\mathbb{E}\left(X_i^2\right)\right]$

Function

```
def calc_variance(t):
    variance = 0
    for x in range(1, t+1):
        p_xy = calc_prob(x, t-x)
        variance = mod_add(variance, mod_multiply((2*x - t)**2, p_xy))
    return variance
```

2.3.2 Calculation

1. Loop Through Possible Outcomes

The function iterates over all possible values of x from 1 to t. Here, x represents a specific outcome in the context of a probabilistic model where t is the total number of trials or rounds.

2. Calculate Probability

For each value of x, the function calculates the probability p_{xy} of x wins and $t - x$ losses using the `calc_prob` function. This function likely computes the probability distribution of outcomes based on the given parameters.

$$p_{xy} = \text{calc\_prob}(x, t - x)$$

3. Update Variance

The function updates the variance using the formula for variance:

$$\text{variance} = \text{variance} + \left((2x - t)^2 \times p_{xy}\right)$$

here the value $2x - t$ comes form $x - (t - x)$ and p_{xy}: This is the probability associated with the outcome x and $t - x$.

The function uses modular arithmetic functions `mod_add` and `mod_multiply` to perform addition and multiplication, ensuring the operations stay within certain numerical limits, if applicable.

3 Part 2: Alice Against Bob's Predefined Strategy

In this scenario, Bob's strategy is dependent on his performance in the previous round. His choices are determined as follows:

- **Defensive Play:** If Bob won the previous round, he chooses to play defensively.
- **Balanced Play:** If the previous round resulted in a draw, Bob chooses to play balanced.
- **Aggressive Play:** If Bob lost the previous round, he chooses to play aggressively.

3.1 2a) Optimal Strategy for Alice Also Known as Greedy

Given Bob's fixed strategy, Alice can adjust her play style to maximize her probability of winning. For example:

- If Bob plays *Defensively*, Alice should play *Balanced*, as the probability matrix suggests that *Balanced* play has a higher expected value of points gained in the current round. In *Attack*, the expected value is $\frac{5}{11}$, while in *Balanced*, the expected value is $\frac{3}{10} + \frac{1}{2} \cdot \frac{1}{2} = \frac{11}{20}$.
- If Bob plays *Balanced*, Alice should play *Aggressively*, as the expected points in that round are higher for the *Attack* strategy, i.e., $\frac{7}{10}$ is greater than *Balanced* which is $\frac{1}{3} + \frac{1}{2} \cdot \frac{1}{3} = \frac{1}{2}$ and also greater than *Defensive* strategy which is $\frac{1}{5} + \frac{1}{2} \cdot \frac{1}{2} = \frac{9}{20}$.
- If Bob plays *Aggressively*, Alice should choose a more *Aggressively* or *Defensively* based on the current points of alice and bob. To check which value is bigger, $\frac{n_B}{n_A + n_B}$ or $\frac{6}{11}$.

3.1.1 Result

After running the Monte Carlo simulation on the given code for 10^5 rounds, we obtain approximate results as follows:

- Alice's Points: 57132.5
- Bob's Points: 42869.5

- Alice's Average Points: 0.57132

So we can conclude that on avg. Alice can maximise her points on any Round $\equiv 0.57/1$.

Function

Alice move

```
        def play_move(self):
            if self.results[-1] == 0:
                return 1
            elif self.results[-1] == 0.5:
                return 0
            elif self.results[-1] == 1:
                if (len(self.results) - self.points) / len(self.results)
                    > 6 / 11:
                    return 0
                else:
                    return 2
```

Simulate round

```
        def simulate_round(alice, bob, payoff_matrix):
            alice_move = alice.play_move()
            bob_move = bob.play_move()

            probabilities = payoff_matrix[alice_move][bob_move]

            result = np.random.choice([1, 0.5, 0], p=probabilities)

            alice.observe_result(alice_move, bob_move, result)
            bob.observe_result(bob_move, alice_move, 1 - result)
```

Monte carlo

```
        def monte_carlo(num_rounds):
            alice = Alice()
            bob = Bob()

            payoff_matrix = [
                [[1/2, 0, 1/2], [7/10, 0, 3/10], [5/11, 0, 6/11]],
                [[3/10, 0, 7/10], [1/3, 1/3, 1/3], [3/10, 1/2, 1/5]],
                [[6/11, 0, 5/11], [1/5, 1/2, 3/10], [1/10, 4/5, 1/10]]
            ]

            for _ in range(num_rounds):
                simulate_round(alice, bob, payoff_matrix)
                payoff_matrix[0][0] = [(bob.points / (alice.points + bob.
                    points)), 0,
                                       (alice.points / (alice.points + bob
                                           .points))]
            print("Alice's Points:", alice.points)
            print("Bob's Points:", bob.points)
            print("Alice Avg Points ",alice.points / num_rounds)
```

3.2 2b) More Optimal Strategy for Alice in fewer Rounds

Can't come up with more optimal strategy so implemented same code as in 2a in this section

4 Part 3: Alice Against Bob's Random Strategy

In this problem, Bob is playing his strategy uniformly at random, making it difficult to predict his next move. To maximize her points Alice will play fix strategy irrespective of what bob played in previous round this is similar to memory-less property studied in MTL-106 . We will check which among all the combination

```
    def play_move(self):
        if (len(self.results) - self.points) / len(self.results) > 15/44:
            return 0
        else:
            return 2
```

4.1 Decision Analysis

As we have to choose only one strategy that Alice will follow and continue checking some variable against constant value, then play a move.

We will check all Strategies for expected values of Alice Points .

$$\text{Balanced} = \frac{1}{3}\left(\frac{3}{10} + \frac{1}{3} + \frac{1}{3}\cdot\frac{1}{2} + \frac{3}{10} + \frac{1}{2}\cdot\frac{1}{2}\right) = \frac{1}{3}\cdot\frac{27}{20} = 0.450$$

$$\text{Defensive} = \frac{1}{3}\left(\frac{6}{11} + \frac{1}{5} + \frac{1}{2}\cdot\frac{1}{2} + \frac{1}{10} + \frac{4}{5}\cdot\frac{1}{2}\right) = \frac{1}{3}\cdot\frac{329}{220} = 0.498$$

$$\text{Aggressive} = \frac{1}{3}\left(\frac{n_B}{n_A+n_B} + \frac{7}{10} + \frac{5}{11}\right) = \frac{1}{3}\cdot\left(\frac{n_B}{n_A+n_B} + \frac{127}{110}\right) = \frac{n_B}{3\cdot(n_A+n_B)} + 0.251$$

here it is evident that Balanced is the worst of all and we now have to check what of the remaining defensive or aggressive we have to use in next play, which is dependent on current points of Alice and Bob.

$$\text{check for } \frac{n_B}{n_A+n_B} > \frac{329}{220} - \frac{127}{110}$$

$$\frac{n_B}{n_A+n_B} > \frac{15}{44}$$

if this condition is satisfied then we will use Aggressive Strategy and if it is not valid then use Defensive Strategy

4.2 Results

After running the Monte Carlo simulation on the given code for 10^5 rounds, we obtain approximate results as follows:

- Alice's Points: 53937.0
- Bob's Points: 46065.0
- Alice's Average Points: 0.53937

So we can conclude that on avg. Alice can maximise her points on any Round $\equiv 0.54/1$.