Card Counting Black Jack: Phase 1

Michalis Lamprakis 2020030077
Dimitris Ilia 2020030200

Project context

The goal of this problem is to try to find an optimal Black Jack policy that beats the dealer in a single player game (agent against the dealer). A simplified version of Blackjack is simulated without betting, "card splitting" or other more advanced rules. The project is structured in two main phases: i) learning a basic policy without card counting and ii) extending the agent to exploit card counting using the "Hi-Lo" system.

Implementantion

The whole Implementantion is based on **BlackJackEnv** class that we made and contains all the necessary functions to simulate the game. The provided code contains detailed comments.

In **task 0** a simple game setup is implemented where the user can play against the dealer with a simple visualization enviroment(console prints). As a baseline, we evaluated a completely random policy and a simple threshold one where the agent hits if the player's hand is below 17, and sticks otherwise. For a number of 100000 games, we observe the following results:

Policy	Win	Draw	Lose
Random (Uniform)	28.42%	4.25%	67.33%
Threshold (Stick $\geq$ 17)	40.99%	10.14%	48.88%

As regards the threshold policy, while it is better than a purely random strategy, still underperforms compared to our learned Q-agent, as we will see below. The threshold policy fails to incorporate the dealer's upcard or ace usability, which are critical in optimal decision-making.

For **task 1** we train an agent to find an optimal action policy using tabular Q-learning with the following update formula:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right)$$

After experimenting with different values we decided to use $\alpha = 0.1$ (avoids too fast or too slow convergence) and $\gamma = 1.0$ (because future rewards are just as valuable as immediate ones). Also for the exploration rate we satarted with $\epsilon = 1.0$ and we decay it up to $\epsilon = 0.05$, linearly over episodes.

We model the game using a standard reinforcement learning framework. The state space for this environment is the tuple $(\mathit{player\ hand\ total}, \mathit{dealer\ upcard}, \mathit{usable\ ace})$. After training the agent for 500000 episodes and playing 100000 games with the learned policy (large enough numbers for the action space, the agent's win rate stabilizes) we observe a win rate of $\approx 42\%$, a loss rate of $\approx 48\%$ and $\approx 9\%$ rate for draw.

After searching online ,there is a proven theory for the optimal strategy of BlackJack (without card counting in this case) and using this as a benchmark we compare our learned policy and observe a mismatch rate of less than 10% which is acceptable. We are indeed convinced that our agent has learned an optimal policy, but even with this the loss rate is higher than the win rate (so we do not yet beat the casinos).

This is the strategy we used as a reference:

Hard Totals (No usable ace)

Player Total	Dealer Showing	Action
17–21	Any	STICK
13–16	2–6	STICK
13–16	7–A	HIT
12	4–6	STICK
12	2–3 or 7–A	HIT
5–11	Any	HIT

Soft Totals (Usable Ace)

Soft Total	Dealer Showing	Action
19–21	Any	STICK
18	2, 7, 8	STICK
18	3–6, 9, 10, A	HIT
13–17	Any	HIT

For **task 2**, the goal is to try to imporve our win rate by using "basic" card counting with a simple "Hi-Lo" system. The new action space is extended to include the state of the game, which can be Low, Neutral, or High. As mentioned in the project description, we assume that we are playing with a single deck of cards and the deck is reshuffled when 10 or less cards are left. A slight modification is made due to this and we change states based on > -3 or < 3 (not -5 and 5 that was initially suggested).

Now because the action space is bigger we train our agent for 1500000 episodes. After evaluating for 1000000 episodes, for our bad luck, we do not observe any improvement in the win rate.(maybe +0.5% but is not constant). This result aligns with expectations, while card counting is theoretically advantageous, its power lies primarily in adjusting bet size based on the count, not just in modifying the HIT/STICK decisions. Since our game does not include betting, the potential of card counting cannot be fully realized in this phase.

Execution Details

All the above are implemented in **BlackJack_PHASE_1.ipynb** file. There is a menu with the following options:

- 1: Play a game manually against the dealer.
- 2: Evaluate random policy.
- 3: Evaluate threshold policy.
- 4: Train agent without card counting.
- 5: Evaluate the trainted agent without card counting.
- 6: Compare the trained agent (from option 2) with the optimal policy.
- 7: Train agent with card counting.
- 8: Evaluate the trainted agent with card counting.
- 9: Exit.

To evaluate each policy (except the random ones), we need to train the agent first, otherwise Q-tables will be empty.