Adaptive BlackJack Strategies: Leveraging Reinforcement Learning for enhanced profits

Nayan Sharma
Masters' Artificial Intelligence
Technical University of Applied Sciences Würzburg-Schweinfurt
Würzburg, Germany
nayan.sharma@study.thws.de

***Abstract*—Card games pose an attractive opportunity for performing experimentation, along with uncertainties and the partial observability of the game with reinforcement learning.This approach aims to discover the most effective strategies for winning consistently in the specific card game of Blackjack.So its obvious that people playing these games involving money would like to explore new strategies to increase winnings or minimize losses. This paper explores model-free methods such as Q Learning and SARSA[1], seeking the optimal policy for playing Blackjack against a dealer.It tries to find the policies, which indicate the best action based on the player's score and the dealer's visible card. The observed policy aligns with the fact how players with advanced techniques apply specific strategies in their games.**

I. Introduction

Blackjack is an ideal game for implementing reinforcement learning[4], especially for beginners. The game's manageable state and action spaces facilitate comprehension of policies derived from our algorithms, while still offering opportunities for policy optimization. Additionally, various works on optimising BlackJack means numerous strategies already exist[6], providing a solid basis for comparing and refining based on previous works. Existing studies have explored the application of different algorithms, such as Monte-Carlo Tree Search [5] and Temporal-Difference Control, to address strategic decision-making in games like Blackjack. By employing a fixed policy and iterative sampling, unbiased estimates of both the policy and state-value pairs can be obtained. More sophisticated reinforcement learning methods are also being used to solve for the optimal policy in Blackjack, such as a Deep Q Learning neural networks[4], and are proved to be very efficient. In Blackjack, a card game where one to more players can play against the dealer, the player aims to have a greater score than the dealer's without exceeding 21. The objective is to optimize the player's strategy to maximize their profits.

II. Description of Reinforcement Learning Player

A. Problem Definition

The objective here is to develop such a agent which can give us an optimal policy , learning from different scenarios like Basic Strategy and Complete Point Count System. This system should be then enhanced to maximise profits.

B. Implementation Details

To implement the Blackjack modelling in Python, I created a basic environment to simulate the game for required experimentation. There are two environments for Basic Strategy and Card Counting[2] respectively. A regular deck of 52 cards was used , multiple decks can also be accessed for card counting, where values for the cards is as follows: the cards between 2 and 10 have the same worth as their rank. Kings, Queens and Jacks are 10 and A is 1 or 11, depending on the situation. There are four action spaces: hit, stand, double down and insurance. In the game setup, the observation space include the total value for the players hand, the hand value for the dealer and three more individual states representing the options for usable ace, doubling down and insurance repestively. Reward of +1 for winning, -1 for losing and 0 for a draw have been setup. Also when the double down action is performed, the required behaviour of doubling the reward is implemented. For insurance, half of the reward is lost if the player loses the insurace and likewise onlny half if the reward is retrieved if the insurance pays off.

III. Approaches

To improve the players policy we use two model free approaches namely Q-learning and SARSA(State,Action, Reward, State, Action).

A. Q-learning

Q-learning is a model-free approach used to estimate the action-value function for a specific state-action pair by iterating over some data[3]. In Blackjack, we employ Q-learning to identify the best action, which is the one yielding the highest action value in a given state. Updating the Q values requires a learning rate, a discount factor, and an immediate reward.

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \quad (1)$$

The policy exclusively derived from Q-learning without incorporating exploration is greedy. Actions are chosen using an epsilon-greedy exploration strategy. Subsequently, the Q table is updated using Equation 1, following each sequence of observations, and it also returns our estimated rewards.

Over time, we modify the exploration strategy to balance the exploration-exploitation trade-off, increasingly favoring exploitation as the game progresses.

B. SARSA

State–action–reward–next state–next action (SARSA) is a reinforcement learning algorithm designed to be faster to run than Q-learning but may need more data or iterations to converge[6]. We update estimates based on almost the same type of data as for Q-learning. For our SARSA algorithm, our code structure was very similar to Q learning except that it would also use the action value for the next state and next action instead of maximizing over the actions. It would pick this next action also through an epsilon-greedy strategy and update the Q values based on the equation below.

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma Q(s',a') - Q(s,a) \right]$$

IV. Experiments and Evaluation

A. Performance Analysis

After evaluating the performance of Q-learning, SARSA by running each for a series of episodic epochs. The results, illustrated in TABLE 1, reveal clear trends in different scenarios.

Q-learning and SARSA vs. a Random Strategy: Both Q-learning and SARSA outperform the random baseline strategy. This indicates that these reinforcement learning approaches are more effective at learning and applying strategies in blackjack compared to making random decisions.

Q-learning vs. SARSA: While Q-learning shows a better performance than SARSA, the difference is not substantial or leading us to any positive values. Both methods demonstrate some levels of effectiveness, suggesting that either could be a viable option for learning strategies in blackjack depending on certain scenarios.

Win-Loss Dynamics: It's crucial to acknowledge that despite the relative success of Q-learning and SARSA, all the tested agents—including the random strategy are more likely to lose than win. This underscores the inherent difficulty of the game and reflects the house edge present in blackjack.

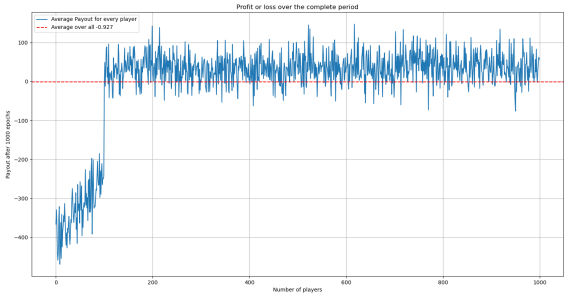

Fig. 1: q-learning plot for profit or loss(dd=Yes,ia=Yes,cc,nd=1)

TABLE I: Performance of Q-learning and SARSA in Blackjack

algo*	DD*	IA*	CC*	AP*	ND*
Q-learning	No	No	No	-47.3	
SARSA	No	No	No	-196.92	
Q-learning	No	No	Yes	-23.7	4
SARSA	No	No	Yes	-193.7	4
Q-learning	Yes	No	Yes	-71.7	4
SARSA	Yes	No	Yes	-221.2	4
Q-learning	Yes	Yes	Yes	-40.0	4
SARSA	Yes	Yes	Yes	-208.5	4
Q-learning	Yes	Yes	Yes	-0.927	1
SARSA	Yes	Yes	Yes	-195.0	1

- **algo***: Algorithm
- **DD***: Double Down
- **IA***: Insurance Action
- **CC***: Card Counting
- **AP***: Average Payout
- **ND***: Number of Decks

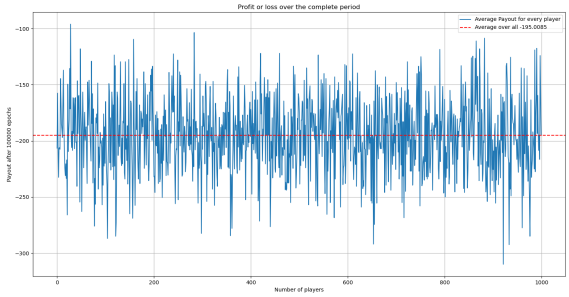

Fig. 2: SARSA plot for profit and loss(dd=Yes,ia=Yes,cc,nd=1)

B. Hyper Parameter Tuning

The hyper-parameters that could be adjusted included the discount factor (γ), learning rate (α), and epsilon (ϵ) (refer to Table 2).Epsilon-greedy exploration strategy is implied for both approaches. Initially, the epsilon was kept constant, but after evaluating performance (average reward return per episode), i decided to use a decay factor of 0.9. Established a minimum epsilon to ensure some level of exploitation after iterating through each episode, though less than at the beginning. We maintained the learning rate at 0.01, which proved effective for both approaches. Regarding the discount factor,started with 0.99 and manually tuned it for optimal performance, ultimately settling on 0.9. The primary performance metric was the comparison of average rewards per episode (reflecting wins, ties, and losses) from each approach against the random baseline. Evaluating the performance repeatedly, guided the manual tuning of the hyper-parameters.

TABLE II: Hyperparameters

Hyperparameter	Value
Epsilon (ϵ)	0.1
Learning Rate (α)	0.01
Gamma (γ)	0.9

C. State and State-Action Space Analysis

The state-action space in our Blackjack implementation is extensive due to the numerous possible states and actions. The state space is defined by the tuple (sum of player's hand, dealer's visible card, presence of usable ace, double down availability, and insurance availability). Given:

- Player's hand can sum from 0 to 30 (31 values)
- Dealer's visible card can range from 1 to 10 (11 values)
- Usable ace can be True or False (2 values)
- Double down can be available or not (2 values)
- Insurance can be available or not (2 values)

This results in a state space of 31 * 11 * 2 * 2 * 2 = 2,728 states. Each state can have one of four actions (stick, hit, double down, insurance), leading to a state-action space of 2,728 * 4 = 10,912.

Given the large size of the state-action space, achieving stable estimates $Q(s, a)$ can be challenging. Strategies such as epsilon-greedy exploration with decaying epsilon address this issue to balance exploration and exploitation.Also tuning hyperparameters like the learning rate and discount factor helps to optimize learning stability and convergence.

D. Policy Changes Due to Rule Adjustments

The rule changes introduced in the Blackjack game, specifically the inclusion of double down and insurance actions, significantly impacted the learned policies. These additional actions provide more strategic depth and options for the player, which the reinforcement learning algorithms needed to account for. Reference TABLE 1.

For instance, the availability of double down affects the risk-reward balance for the player. The model had to learn when doubling down would maximize the expected reward, considering the increased risk of going bust. Similarly, the insurance action introduced another layer of decision-making, where the model had to evaluate the likelihood of the dealer having a blackjack and the potential payout.

These changes required the model to adapt its policies to the new dynamics, leading to observed policy adjustments. By expanding the action space, the agent had to learn more nuanced strategies, which could initially result in higher variance in performance. Over time, with sufficient training, the models were able to converge to policies that effectively utilized these additional actions to improve the overall strategy.

From Fig.3 , it is evident that the Q-values for the DOUBLE DOWN action exhibit significant variability across different player and dealer hand values. Peaks in the Q-values suggest scenarios where doubling down is particularly advantageous, such as when the player has a hand value around 12 to 16 and the dealer shows a lower hand value, typically between 2 and 6. This aligns with basic blackjack strategy, where doubling down is often recommended when the player has a favorable position against the dealer's weaker hand.

Conversely, the troughs in the graph indicate situations where doubling down is less favorable or even detrimental. These scenarios are marked by negative Q-values, suggesting that the expected rewards from doubling down in these cases are lower than not taking this action. Such situations often arise when the player has a higher risk of busting or when the dealer's hand is strong, such as a 9, 10, or Ace.

The second plot(Fig. 4)represents the Q-values for the INSURANCE action, again with the player's hand value on the x-axis and the dealer's hand value on the y-axis. The color gradient here shows a more constrained range of Q-values compared to the DOUBLE DOWN action, with a generally flatter surface punctuated by a few notable peaks and valleys.

The relatively flat surface suggests that the INSURANCE action typically holds less variability in its expected rewards across different hand values. This can be attributed to the nature of insurance in blackjack, which is a side bet on whether the dealer has a blackjack when showing an Ace. The limited positive Q-values suggest that there are very few scenarios where taking insurance is beneficial, as indicated by the isolated peaks. These peaks are likely rare and specific cases where the player's potential loss is substantially mitigated by taking insurance.

The prevalent negative Q-values reflect the generally unfavorable nature of the insurance bet in blackjack. Since the insurance bet statistically favors the house, the Q-learning algorithm correctly identifies that, in most situations, opting for insurance does not maximize the player's expected reward.

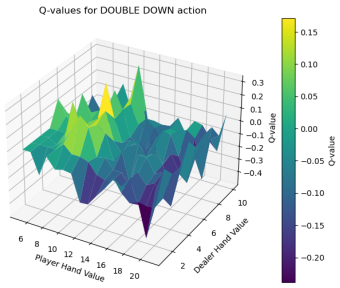

Fig. 3: Q-values for double down action(q-learing)

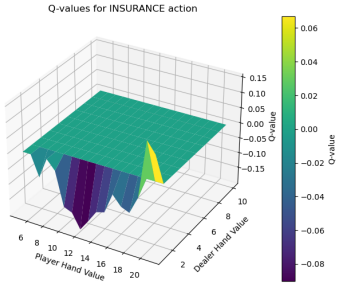

Fig. 4: Q-values for insurance action(q-learning)

V. Conclusion

Exploring with reinforcement learning in the card game Blackjack offers valuable insights into optimizing player strategies.Methods such as Q-learning and SARSA are employed to determine the optimal policy for a player in Blackjack games against a dealer. The represen- tation of policies provides players with a clearer visualization of optimal decision-making. Two reinforcement learning approaches, Q-learning and SARSA, are considered in this paper to estimate the action- value function and update policies based on observed states and actions. Hyper-parameter tuning, involving factors like the discount factor, learning rate, and epsilon, allowed for the flexible adjustment of the designed algorithms. Extending the observation space with actions like doubling down and taking insurance led furthur insight on implications of the action. There is also potential for future work to explore scenarios involving more than these actions expading the action space furthur. We can also look into more advanced algorithms like Deep Q which performs significantly better, Experiments done on this project can be transitioned to other algorithms. By delving deeper in theoritical and practical aspects of these algorithms, we can expand the application of these reinforcement learning techniques to a broader range of strategic decision-making situations beyond card games.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed., MIT Press, Cambridge, MA, 2018.

[2] E. O. Thorp, *Beat the Dealer*, Vintage, New York, 1966.

[3] Watkins, Christopher and Dayan, Peter. *(*1992) Technical Note: Q-Learning. Machine Learning. 8. 279-292. 10.1007/BF00992698.

[4] A. Perez-Uribe and E. Sanchez, "Blackjack as a test bed for learning strategies in neural networks," 1998 IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98CH36227), Anchorage, AK, USA, 1998, pp. 2022-2027 vol.3, doi: 10.1109/IJCNN.1998.687170.

[5] Srinivasaiah, Raghavendra and Biju, Vinai and Jankatti, Santosh and Channegowda, Ravikumar and S J, Niranjana. (2024).*R*einforcement learning strategies using Monte-Carlo to solve the blackjack problem. International Journal of Electrical and Computer Engineering (IJECE). 14. 904. 10.11591/ijece.v14i1.pp904-910.

[6] Schiller, Marvin and Gobet, Fernand. (2012). *A* Comparison between Cognitive and AI Models of Blackjack Strategy Learning.