Learning to Win
An AI-Driven Approach to Beating Blackjack

Kristian Feed, Wade Williams

November 2023

Abstract

This paper investigates the application of various artificial intelligence algorithms to the game of Blackjack, with the objective of approaching or surpassing the traditional win probability of 42.22%. Utilizing a simulation of the game, strategies ranging from random to basic strategy and advanced machine learning methods, including Q-Learning and Deep Q-Learning, were evaluated over numerous iterations. The basic strategy yielded a win rate close to the established benchmark, while machine learning algorithms demonstrated an increasing trend in performance, suggesting potential for further optimization. Interestingly, advanced reinforcement learning algorithms did not achieve the expected win rates, indicating a need for more sophisticated model tuning. The study confirms the capability of AI to learn and apply complex strategic decisions in Blackjack and suggests a promising direction for future research to refine these approaches.

1 Introduction

1.1 Blackjack

Blackjack, also known as 21, is one of the most common card games played in casinos worldwide. The game of blackjack is a strategic confrontation between the player and the dealer, wherein each aims to achieve a hand value closest to, but not exceeding, 21. This seemingly simple arithmetic goal belies the depth of strategic intricacies involved: players must decide whether to "hit" and take another card, "stand" with their current total, "double down" to potentially double their earnings, or "split" identical cards into two hands. These decisions are made based on an assessment of the player's current hand in relation to the dealer's visible card.

In blackjack, the outcomes are profoundly shaped by the hands players are dealt and subsequent decisions they choose to make. Each deal sets a statistical framework, as each card drawn influences the remaining deck's composition, thereby affecting the probability of future cards and potential hand outcomes. A player's initial hand can range from a winning natural blackjack to a more challenging low-value hand, demanding a different strategy. The dealer's up card is equally pivotal; certain cards like a 5 or 6 increase the dealer's odds of busting and inform the player's tactical considerations.

1.2 Machine Learning of Blackjack

Blackjack's appeal from a computational and artificial intelligence standpoint lies in its deterministic yet probabilistic essence. The game operates under fixed rules, yet the outcomes are shaped by the initial hands dealt and the strategic choices of the players. This interplay of known factors (such as rules and visible cards) and unpredictable elements (like hidden cards and future deals) makes blackjack a prime testing ground for machine learning (ML) approaches.

Machine learning in the context of blackjack involves developing algorithms that can process historical data to make predictions or decisions without being explicitly programmed to perform the task. These algorithms can identify patterns and make strategic decisions, much like a human player. By analyzing large datasets of blackjack hands, ML can learn the best responses to a variety of game situations based on the likelihood of winning or losing.

With its structured reward system (wins, losses, or draws) and the necessity for a strategy that adapts to changing card distributions, blackjack provides a fertile environment for supervised and unsupervised learning models to refine decision-making strategies. The limited and specific set of possible player actions in response to each card combination allows ML algorithms to systematically evaluate the potential outcomes of different moves. As such, the success of ML techniques in mastering blackjack strategy can offer valuable insights into their broader applications in domains that require nuanced decision-making under conditions of uncertainty.

1.3 Project Goals

The average, long-term probability of winning a classic blackjack hand is 42.22%, a figure that emerges from computer simulations of the game over millions of hands while assuming standard rules and perfect strategy based on the probability theory of blackjack. This percentage reflects a player following the basic blackjack strategy without any card counting or advanced techniques. The goal of this project was to develop algorithms to achieve blackjack's perfect-strategy win probability of 42.22% and to gain insights into the performance of different learning algorithms in blackjack.

2 Model and Methods

In our project, we developed a blackjack game simulation to test each strategy's effectiveness. We compared the effectiveness of six different strategies: random strategy, basic strategy, and a basic neural network. We also discuss the exploration of more advanced Q-Learning and Deep Q-Learning algorithms included in our code under references. To illustrate our results, we determined the win percentage for each of the strategies over numerous trials with varying numbers of simulations ran and analyze the blackjack strategy chart in the form of heat maps for the neural network over different numbers of simulations.

2.1 Random Strategy

The "Random Strategy" algorithm was devised as a baseline against which we can measure the efficacy of other strategies – essentially, to determine whether deliberate strategies offer a statistically significant improvement over decisions made by chance. This algorithm operates by arbitrarily electing to "hit" or "stand", which can culminate in:

1. Immediate "stand" without taking additional cards.
2. A "stand" after a series of "hit" actions.
3. A "bust" if the hand value exceeds 21.

2.2 Simple Strategy

The "Simple Strategy" in blackjack is guided by the pivotal threshold of 17, reflecting the dealer's rule to hit until reaching this hand value. It's a strategic fulcrum balancing the risk of busting against the prospect of improving the hand. As the player's total nears 21, the risk of busting increases, informing the decision to hit or stand.

The strategy leverages the dealer's constraints—hitting below 17 and standing at 17 or higher—as a player standing with less than 17 often relies on the dealer busting to win. This simple rule presumes that hitting below 17 generally offers a better expected value, aiming to maximize the player's chances without complex calculations.

The simple blackjack strategy can be mathematically expressed using a piece-wise function that prescribes a specific action based on the player's total hand value. This strategy hinges on a single threshold: the total value of 17, which is the same value at which the dealer is typically required to stand. The rationale behind this is that hitting on a total less than 17 carries a lower risk of busting, while standing on a total of 17 or higher avoids the high risk of exceeding 21. The decision rule is succinctly captured by the function:

$$\text{action} = \begin{cases} \text{“hit”} & \text{if player\_total} < 17, \\ \text{“stand”} & \text{if player\_total} \geq 17. \end{cases}$$

In this equation, player_total represents the sum of the values of the player's cards. If this total is less than 17, the function dictates that the player should take the action "hit," which means to take another card. Conversely, if the player's total is 17 or more, the function prescribes the action "stand," indicating that the player should not take any additional cards. This strategy, while simplistic, follows the logic of minimizing the chances of busting while trying to reach a hand value that is competitive against the dealer's potential total.

2.3 Basic Strategy

The "Basic Strategy" in blackjack is a set of decisions for playing hands that has been optimized through mathematical analysis. The strategy varies depending on whether the player's hand is a hard total (no Ace or an Ace that must be counted as 1 to avoid busting) or a soft total (an Ace that can be counted as 11 without busting). The blackjack strategy

can be mathematically expressed as a set of piece-wise functions that determine the player's action based on their hand total and the dealer's up card value. We define the action function $A(p, d)$ for hard totals, where p represents the player's total and d the dealer's up card value, as follows:

$$A(p,d) = \begin{cases} \text{“hit”} & \text{if } p < 12, \\ \text{“stand”} & \text{if } p = 12 \text{ and } d \in \{4,5,6\}, \\ \text{“hit”} & \text{if } p = 12 \text{ and } d \notin \{4,5,6\}, \\ \text{“stand”} & \text{if } 13 \leq p < 17 \text{ and } d \in \{7,8,9,10,11\}, \\ \text{“hit”} & \text{if } 13 \leq p < 17 \text{ and } d \notin \{7,8,9,10,11\}, \\ \text{“stand”} & \text{if } p \geq 17. \end{cases}$$

For soft totals, where the hand includes an Ace that can be counted as 11, we define a separate function $A_{\text{soft}}(p, d)$:

$$A_{\text{soft}}(p,d) = \begin{cases} \text{“hit”} & \text{if } p \leq 17, \\ \text{“hit”} & \text{if } p = 18 \text{ and } d \in \{9,10,11\}, \\ \text{“stand”} & \text{if } p = 18 \text{ and } d \notin \{9,10,11\}, \\ \text{“stand”} & \text{if } p > 18. \end{cases}$$

The overall strategy function that incorporates both hard and soft totals is given by:

$$A_{\text{overall}}(p, d, \text{is\_soft}) = \begin{cases} A_{\text{soft}}(p,d) & \text{if is\_soft} = \text{True}, \\ A(p,d) & \text{if is\_soft} = \text{False}. \end{cases}$$

Here, is_soft is a boolean variable indicating whether the player's hand is soft, meaning it includes an Ace that can be valued as 11 without busting.

2.4 Machine Learning (ML) Algorithm

The Machine Learning (ML) algorithm employed in our study is based on a simple feedforward neural network, constructed using the Keras Sequential model, comprising five layers. The network architecture delineates a progression of neurons across these layers with the following counts: 16, 128, 32, 8, and a final output layer harboring a singular neuron. The pivotal activation function for the output layer is the sigmoid, expressed mathematically as $\sigma(z) = \frac{1}{1+e^{-z}}$, which is particularly suited for binary classification tasks. The default activation for the remaining layers is linear, which is implicitly assumed in the absence of explicit specifications in Keras. The network employs binary crossentropy as its loss function, formalized as

$$L = -\frac{1}{N} \sum_{i=1}^{N} \left[y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i) \right], \tag{1}$$

where N is the number of samples, y_i denotes the true label, and p_i represents the predicted probability output by the network for the i-th sample. For optimization, the model utilizes

stochastic gradient descent (SGD), iteratively adjusting the model's weights to enhance prediction accuracy through 20 epochs with update rule $\theta \leftarrow \theta - \eta \nabla_\theta L(\theta)$, where η is the learning rate. The action-selection mechanism is contingent on the neural network's output; a prediction exceeding 0.5 prompts a "hit" action, whereas a lower value suggests "stand". It is noteworthy that this ML algorithm diverges from Q-learning, primarily as it does not engage in explicit Q-value updates. Rather, it is adeptly trained to forecast the accurate action rooted in the input features. This model is refined through training on a dataset with predefined correct actions, distinguishing it from online learning paradigms like Q-Learning or DQN.

The trained model's performance is then assessed through the Receiver Operating Characteristic (ROC) curve, which plots the true positive rate against the false positive rate at different thresholds. The Area Under the ROC Curve (AUC) serves as a measure of the model's ability to classify correctly, with higher AUC values indicating superior predictive performance.

In practical game play, the trained network uses the game state—including the player's total hand value and whether an ace is present, along with the dealer's visible card—to predict the likelihood of benefitting from a "hit." A probability greater than 0.5 suggests that taking another card would be favorable ("hit"), while a lower probability advises against it ("stand"). This approach, leveraging predictive analytics, aspires to refine game-play decisions in a manner akin to the strategic thinking of experienced blackjack players.

2.5 Q-Learning (QN) Algorithm

The Q-Learning (QN) Algorithm is a quintessential model-free reinforcement learning method that operates without the need for a neural network. It leverages a Q-table to explicitly store the Q-values corresponding to each state-action pair. In the absence of a neural network, the concept of activation functions is rendered inapplicable.

Q-values are updated using the Temporal Difference (TD) error, encapsulated in the equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right], \tag{2}$$

where α is the learning rate, γ is the discount factor, r is the reward, s and s' denote the current and subsequent state, respectively, and a and a' represent the current and subsequent action, respectively.

Given the algorithm's reliance on Q-values rather than gradient-based optimization, traditional optimizers are not utilized. Instead, action selection is driven by an epsilon-greedy strategy, where the probability of choosing the best-known action is $1 - \epsilon$, and the probability of selecting a random action is ϵ. This strategy balances exploration and exploitation, ensuring that the algorithm does not become too greedy based solely on current knowledge.

Q-Learning's objective is to learn the value of actions through direct interaction with the environment, without the need for a model of the environment's dynamics. It is particularly effective in scenarios where the possible states and actions are known, but their outcomes are unpredictable.

2.6 Deep Q-Learning (DQN) Algorithm

The Deep Q-Learning (DQN) Algorithm is an extension of the classic Q-Learning, utilizing a neural network to approximate the Q-value function. The network comprises multiple layers, with batch normalization and ReLU activations introducing non-linearity into the model.

Batch normalization is applied to each layer's inputs, normalizing the output to have a mean of zero and a standard deviation of one, stabilizing the learning process:

$$\text{BN}(x) = \gamma \left(\frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \right) + \beta, \tag{3}$$

where x is the input to a layer, μ and σ^2 are the mean and variance of x, respectively, γ and β are parameters to be learned, and ϵ is a small constant for numerical stability.

ReLU activation functions are used within the hidden layers and are defined as:

$$\text{ReLU}(x) = \max(0, x), \tag{4}$$

which introduces non-linearity by outputting the input directly if it is positive, and zero otherwise.

The loss function for the DQN algorithm is the mean squared error between the predicted and target Q-values, formalized as:

$$L(\theta) = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta) \right)^2 \right], \tag{5}$$

where θ denotes the parameters of the current network, θ^- the parameters of the target network, γ the discount factor, r the immediate reward, s and s' the current and next state, and a and a' the current and next action, respectively.

Optimization is performed using Stochastic Gradient Descent (SGD) with momentum. The update rule for SGD with momentum is given by:

$$v_t = \mu v_{t-1} - \eta \nabla_\theta L(\theta), \tag{6}$$

$$\theta = \theta + v_t, \tag{7}$$

where v_t is the velocity at time t, μ is the momentum factor, η is the learning rate, and $\nabla_\theta L(\theta)$ is the gradient of the loss with respect to the parameters.

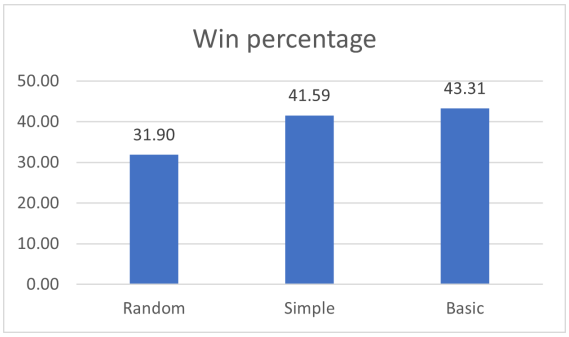
For action selection, the DQN algorithm employs an epsilon-greedy policy, balancing the trade-off between exploration of the state space and exploitation of the known Q-values. The Q-values are not stored in a table but are predicted by the neural network, and the weights are updated via backpropagation.

DQN improves the stability of applying Q-Learning with neural networks by incorporating techniques such as experience replay and fixed Q-targets.

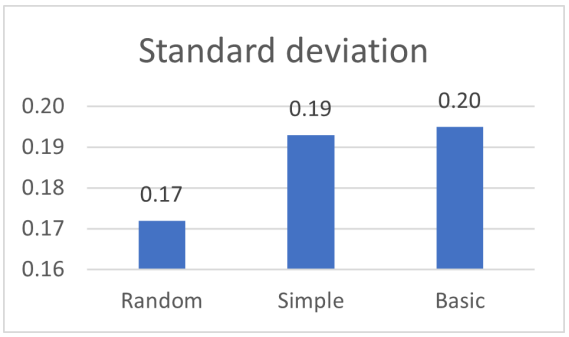
3 Numerical Results

3.1 Human Strategy Results

Figure 1 displays the win percentage and standard deviation for the human strategies (random, simple, and basic) run for 10 trials of 100,000 simulations. As depicted in Figure 1(a), the win percentage for a random strategy slightly exceeds 30%, while a simple strategy yields just over 40%. Utilizing a basic/optimal strategy, the average, long-term win percentage reaches 43.31%, slightly differing from the known average, long-term win probability of 42.22% due to our simulation not incorporating the option to surrender. Figure 1(b) demonstrates that through sufficient simulations, the standard deviation for each strategy is reduced to less than 0.2%.



(a) Strategy Win Percentage

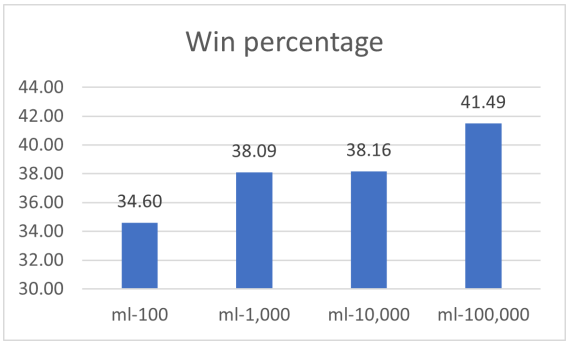


(b) Strategy Standard Deviation

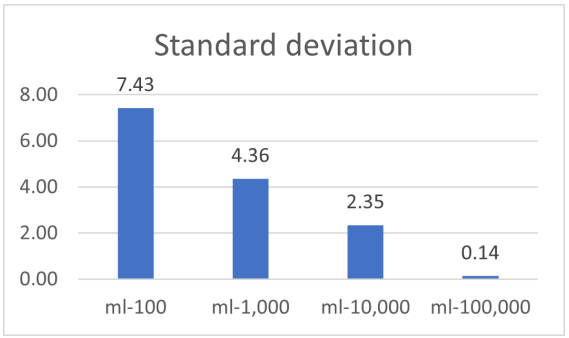
Figure 1: Human Strategy Comparison

3.2 Machine Learning Results

The focus then shifts to the potential of a machine learning algorithm in achieving higher win percentages. As Figure 2(a) indicates, win percentage increases with more simulations, while Figure 2(b) shows a corresponding decrease in standard deviation, aligning with expectations.



(a) ML Win Percentage



(b) ML Standard Deviation

Figure 2: ML Strategy Results

3.3 Heat Map Comparisons

To discern the nuances between the basic strategy and the machine learning approach, we used heat maps, displaying win percentages across various player hands and dealer cards. Along the x-axis we have different values of the player's hand, and along the y-axis we have the value of the dealer's card. In the rightmost column we have the range of colors depending on the win percentage. Figure 3 presents a heat map from running 100,000 simulations using the basic strategy, with a resulting win probability of 43.08%, marginally lower than the 43.3% average.

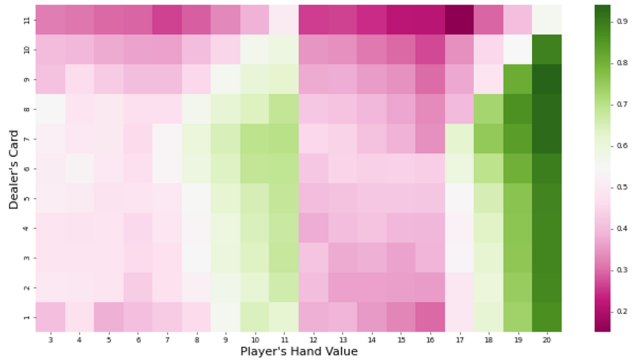


Figure 3: Basic Strategy Heat Map (100,000 simulations)

Additionally, the heat maps for the simple and basic strategies exhibit a high degree of similarity, suggesting that the basic strategy—the optimal approach in blackjack when card counting is not employed—is only slightly more effective than the simple strategy.

Insights gleaned from Figures 4 and 5, which display results from two distinct 100,000-simulation runs using the machine learning algorithm, reveal subtle yet telling variations. At a cursory glance, these heat maps appear similar; however, a detailed examination uncovers notable differences in the gradation of colors for certain hand values and in the gradation patterns. For instance, Figure 3 demonstrates a more gradual color transition between adjacent cells compared to Figures 4 and 5. Specifically, Figure 4 features an anomalously green cell at position (3,6) and a stark transition from purple to nearly white between the cells at (6,1) and (7,1). Similarly, Figure 5 showcases a cell at (4,6) with a markedly deeper shade of purple than its neighboring cells, and a conspicuous shift from (3,11) to (4,11) in the top left corner. While Figure 3 also presents significant color variations, they are more intuitively aligned with blackjack logic. For instance, the substantial shift between x=11 and x=12 is understandable, as moving from a player hand value of 11 to 12 represents a pivotal strategic inflection point. Conversely, it is less intuitive why a player's hand of 4 would perform markedly worse against a dealer's 6 than a hand of 3 or 5.

The pronounced disparities between Figures 4 and 5 raise intriguing considerations. If the color patterns were consistent across both figures, it might indicate that the learning model was trained in a manner slightly divergent from the basic strategy. However, the significant contrasts between the two iterations of the same algorithm might hint that the algorithm has yet to reach its full potential. With a greater number of simulations, surpassing 100,000, it is conceivable that the model could improve its win rate beyond 41.5% and approach—or even exceed—the optimal threshold of 43.3%.

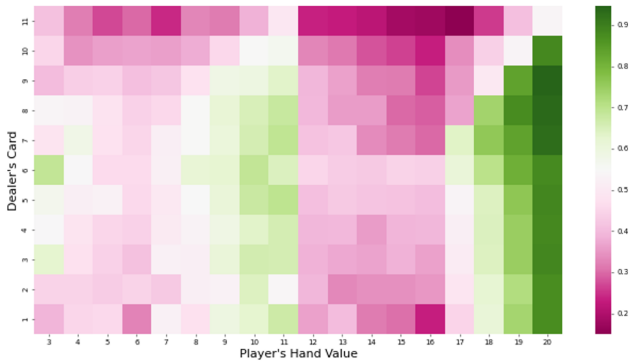


Figure 4: ML Heat Map: Trial 1 (100,000 Simulations)

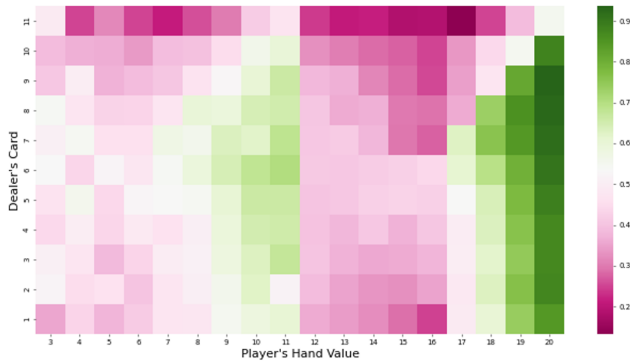


Figure 5: ML Heat Map: Trial 2 (100,000 Simulations)

In sum, the differences between Figures 4 and 5 underscore the potential for the machine learning algorithm to improve with further training, potentially surpassing the 41.5% win probability and edging closer to the optimal strategy's benchmark.

Additionally, while not included in our main analysis of our results, we were able to implement Q-Learning and Deep Q-Learning Algorithms to run on our simulation. However, these algorithms displayed a surprising inability to achieve a win probability close to the optimal value. When ran on 100,000 simulation for 10 trials, the Q-Learning and Deep Q-Learning algorithms obtained an average win probability of 32.52% and 36.82% respectively.

Algorithm	Structure	Activation	Loss Function	Optimizer	Results
Random	N/A	N/A	N/A	N/A	31.9%
Simple	N/A	N/A	N/A	N/A	41.6%
Basic	N/A	N/A	N/A	N/A	43.3%
Machine Learning	Feedforward NN	Sigmoid	Binary Crossentropy	SGD	41.5%
Q-Learning	Q-Table	N/A	Temporal Difference	N/A	32.52%
DQN	Neural Net	ReLU	MSE	SGD w/ momentum	36.82%

Table 1: Comparison of Algorithms Used

4 Conclusion

Simulations employing the basic strategy culminated with a win probability of 43.31%. The machine learning algorithm displayed progressive improvement with increasing simulations, attaining its peak performance of 41.5% at the 100,000 simulation mark. Although this falls short of the basic strategy's efficacy, disparities observed in the heat maps for the 100,000 simulations suggest untapped potential in the learning model beyond the 41.5% win probability. The smoother color transitions in the basic strategy's heat map, as opposed to the two machine learning algorithm runs, imply that the latter may require further simulations for optimization.

Notably, while the Q-Learning and Deep Q-Learning algorithms did not converge to the optimal policy for blackjack, they nonetheless managed to learn a comparatively effective strategy. The preliminary nature of our model and algorithm could account for the underachievement. The difficulty faced by the Deep Q-Learning algorithm in mastering probabilistic transitions and rewards, particularly those hinging on thresholds characteristic of blackjack, suggests that a more nuanced approach to tuning and hyper-parameter optimization may yield the sought-after optimal policy.

In essence, our project has substantiated the potential for basic strategy to marginally surpass a foundational neural network model and has illustrated the prowess of AI algorithms to assimilate strategies that verge on the level of skilled human play. This underscores the feasibility of applying AI to reach or even exceed the established win probability benchmarks in blackjack and emphasizes the strategic acumen inherent in learning algorithms.

Nonetheless, the realm of blackjack strategy and AI interplay remains ripe for further inquiry.

Future research endeavors may delve into more sophisticated adjustments of the Q-Learning and Deep Q-Learning algorithms, apply advanced reinforcement learning methodologies such as Q-λ, and broaden the state-action space to encompass betting strategies, thereby amplifying the comprehensive nature of the game model.

References

[1] Liu, Z., & Spil, G. (2021). Learning Explainable Policy For Playing Blackjack Using Deep Reinforcement Learning (Reinforcement Learning). Stanford University. https://cs230.stanford.edu/projects_fall_2021/reports/103066753.pdf

[2] Wu, A. (2018). Playing Blackjack with Deep Q-Learning. Stanford University. https://cs230.stanford.edu/files_winter_2018/projects/6940282.pdf

[3] Geiser, J., & Hasseler, T. (2020). Beating Blackjack - A Reinforcement Learning Approach. Stanford University. https://web.stanford.edu/class/aa228/reports/2020/final117.pdf

[4] Pandey, S. (2022). blackjack-AI. GitHub. https://github.com/sakshampandey27/blackjack-AI

[5] Feed, K. & Williams, W. (2023). Blackjack-AI-Exploration. Github. https://github.com/wadewilliamsw1234/Blackjack-AI-Exploration