Using Q learning to Play BlackJack

Connor Frazier
Northeastern University, Ma
Email: frazier.co@husky.neu.edu

Abstract

The reason for this research was to determine if a Reinforcement Learning agent could learn casino Blackjack through experiences. The goal of the project was to design an an agent that could beat the current Optimal Blackjack strategy.. The artificially intelligent agent in this project specifically uses Tabular Q-Learning. The states of the environment represent the situation of the player in the game. The actions of the player are the actions allowed by Blackjack rules. Using these rules the agent went through multiple learning and testing phases seeing randomly generated decks of cards throughout gameplay. Over the course of different experiments, the parameters of the Q-Learning agent were tuned and changed including the epsilon, learning rate, discount rate, rewards systems, and even state representations. The agent's performance during the testing phase was compared to three control agents in order to determine its performance. The agent was in fact able to learn the game of blackjack and even create its own strategy.

Introduction

Blackjack is a casino card game, in which the goal of the player is to get the sum of his/her cards to equal 21. In Blackjack, each player individually plays against the dealer aiming to get a higher card sum than the dealer. Each game starts by each player placing a monetary bet. Then each player and the dealer receive two cards each. Both of the player's cards are face up, and only one of the dealer's cards is face up.

Each player then takes their turn choosing of the allowed actions: hit, stand, double, or split. If the player chooses stand, the player keeps the cards currently held and their turn ends. If the player chooses to hit, the player is dealt another card and it remains their turn. If the player chooses double, the player receives is dealt one more card, the player must double the original bet, and the turn ends. If the player chooses split, which is only allowed if the cards of the player are the same value(e.g. two 8 cards), the player's hand turns into two hands. The player must put down another bet for the second hand equal to the first bet placed, and the player can take the same actions on both of their hands. While it remains the player's turn, the player is allowed to choose anyone of these four actions.

After all of the players have taken their turns, the dealer plays its hand based on the rules of the table using the actions of hit and stand. If the player's cards' sum is greater than that of the dealer or if the dealer's sum is greater than 21, the player wins the game. If the player's sum is greater 21 or less than that of the dealer, the player loses.

In the event of a player win, the player receives his/her bet back plus an extra amount equal to the bet the player placed. If the player loses, the player loses the bet that was placed. In the event where the player's sum is equal to the sum of the dealer, a push occurs, and the player takes his/her bet back. Lastly, if the player receives cards equal to 21 on the initial deal and wins the game; the player receives a Natural Blackjack payout consisting of the original bet and an additional amount equal to 3/2 the amount the player original bet.

If the player split, each hand that is being played is treated separately when determining the result of the game and the potential payouts. One final aspect of Blackjack to note is that all times, each player in the game can see all of the other players cards and only one of the dealer's cards, called the face up card. This is due to the fact that the play-

ers are only playing individually against the dealer and not each other.

This game is an interesting problem to solve because Blackjack is a game that is typically considered as a losing game for the player. However when playing optimally, the odds of the player winning are about fifty percent (Kendall and Smith) equal to the dealer's fifty percent chance. This is hard to do as a human because there are so many possible states (card positions). For example, the best decision for the player holding a king and an eight when the dealer is holding a queen as the face up card could be very different from the best move when the player has the same cards and dealer has a five as the face up card.

These situations create a good use case for an intelligent agent. The agent could learn what the best move is for any hand the agent is dealt to give it the best odds of winning the game. The problem gets even more interesting when thinking of an agent that could learn both a card playing strategy and betting strategy. The agent could learn what decisions to make both in the game and on how much the player should bet depending on the current profits or losses.

One last observation to make about Blackjack is that it is episodic. Each game of blackjack is independent of the games before and after it. The decisions of the player only affect future decisions for the current game being played. When the current game is finished, and the player decides to play again, it is a new game. This means that the player has more opportunities to recover from any previous poor decisions. However at the same time games are short, therefore the agent must learn to make good decisions at the start of the game. This is compounded by the fact that the cards being dealt have no particular order. This means that the player's actions are indeterministic. Therefore the player will not know what exactly is going to result from actions taken. Therefore one way to learn Blackjack is to play many games and learn from the results of actions taken. Hence, the use of tabular Q-Learning in this project.

Q-Learning is an appropriate approach to the problem of Blackjack because the decisions are made based on the state of the game. Each choice is determined by the idea: given a certain situation, there is an optimal action to take to improve the player's chance of winning. This approach to playing the game presents the perfect situation for applying a Q-Learning agent.

Background

Reinforcement Learning is a form of learning that relies on the agent interacting with an environment. The agent takes an action or sequence of actions and receives rewards for those actions. The agent then updates its knowledge of those states and actions values. Each action taken is paired with the state it was taken from. This allows the agent to learn what policy(action) to take from each state it may encounter in the environment.

This implementation of Reinforcement Learning is called Tabular Q-Learning. The agent keeps a table of state-action pairs in its memory. As the agent interacts with the environment, it saves each state-action pair it encounters with the rewards received. This allows the agent to pick the action with the best value from its table when it is in a certain state. To update the state-action pair values after it is experienced, this formula is used:

$$Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$$

Figure 1: Q-Learning update formula (Russel & Norvig)

This formula updates the state action pair value with the difference between the current experience's value and the currently saved value of the state-action pair. The size of the difference taken is controlled by the learning rate and discount rate.

The Q-Learning update formula can be applied to a sequence of experiences that each include a state, action, next state, and reward, if the rewards are only received at a terminal state. As this is the case for Blackjack due to only terminal rewards being received. This means the agent must use this update formula to give credit to earlier experiences.

The update formula is used as a apart of this general Q learning algorithm:

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state},\cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$
  until S is terminal

Figure 2: Q Learning pseudo code (Russel & Norvig)

This algorithm is used during the learning phase for the agent to learn the proper Q-values. During the execution phase the agent uses the maximum state-action pair from each state to choose an action.

This style of Reinforcement Learning applies to Blackjack because the states can be represented as the cards held by the agent, and the dealer's face up card as an example. The possible actions of the states are the actions allowed by the rules of Blackjack. The agent may take multiple actions during a game, and can update its value table with the reward from either winning or losing the game..

The last important part of Reinforcement Learning is exploration. For the agent to learn optimal actions, it must

choose random actions or at least suboptimal actions during learning. The agent can not only follow the best valued action from a state because it may miss the opportunity to learn a better choice. As an example, the agent could make a poor choice, choosing a risky action that returns high rewards only sometimes. The agent could get lucky, receiving a high reward on the first try. However, there exists a better choice, one with consistently high returns. If the agent never chooses an action other than the one it already has, it will never learn its full potential. For this reason, an epsilon greedy approach was taken for agent exploration this project. With this approach, the agent will take random actions at different points during its learning, based on a factor epsilon. Epsilon is a probability that when selecting an action, the agent will choose a random action or the best action based on epsilon. This allows the agent to explore other actions in hopes of the discovering the true Q-values as well as the optimal policy of a state.

Related Work

There have been other attempts to use Reinforcement Learning. Tabular Q-Learning has been used before but on a simplified game and state space. The actions allowed are typically slimmed down to take out the action split and even double due to simulation difficulties. Removing these actions changes the game drastically as it eliminates useful options for the player.

Another approach is to decrease the size of the state space of the player. The serves as a way to limit the information the agent must keep track of. One typical state representation is the sum of the player's card and the dealer's face card. Some approaches also include whether or not one of the player's cards is an ace. This allows the Q-table size to remain small. This approach has been used not only in tabular Q-Learning but also in a Monte Carlo approach. Monte Carlo reinforcement Learning approach has been used with the aim to learn Blackjack's probabilistic nature using a more statistical approach than Tabular Q-Learning. The idea is to average the values of the state action pairs over the number of times they were tried to create an inherit weighting towards the actions that provided high rewards more consistently.

Lastly, Deep Q-Learning approaches have been attempted to increase the state space to include not only the previously mentioned features but also: whether or not the player has an ace, if the cards are a pair, and if the action being taken is the first action. At the same time, the recent historic experiences would be used to influence the value of actions from the different state. The aim of this deep learning approach was to be able to se more information to make better informed decisions based not only on the general game play but also how actions in the current game were differing from the agent's knowledge. This idea is similar to counting cards to figure out if the deck is currently in the player's favor or the dealer's favor.

The approach for this project came from a combination of these ideas. Tabular Q-Learning was chosen with the idea of using a larger state state space containing the sum of the agent's cards, the dealer's face up card, whether or not the player has an ace, and whether or not the hand is splittable. The agent is also allowed to take all of the actions allowed by the rules of the game. This approach was taken in hopes of combining the best features of related works.

Project Description

For this project the following rules of blackjack have been implemented in order to define the characteristics of the environment that the agent learns and tests in:

-Cards hold their numerical value (1-11)
- Aces can be 1 or 11, Face cards are 10
-Goal: Gain a hand up to 21 or more than the dealer
-Actions: Hit, Stand, Double, Split
-Payouts: Natural Blackjack 3:2, Win 1:1, Push: Bet back, Loss 0
- 8 decks, reshuffled when 2/3 of cards have been used
- Dealer hits on soft 17

These rules are the standard rules of Blackjack except for two of them. The number of decks used and the Dealer hitting on a soft 17 (when the sum is equal to 17 with an ace as one of the cards) are rules that vary slightly depending on the casino. These two respective variations were chosen because they make the game harder for the player.

For each experiment, the agent first go throughs a learning phase and then a testing phase. The learning phase consists of one hundred thousand consecutive blackjack where the agent is using the Q-Learning algorithm to learn how to play the game. The agent then enters the testing phase, in which the agent plays through ten rounds of two hundred and fifty games using the Q-Table as is. The agent does not explore during the testing phase. Instead the agent uses the knowledge it has gained during learning. This style of testing was chosen in order to minimize the affect of the cards dealt on the agent's performance. Since the cards are dealt at random, the agent could be dealt poor hands during one round of two hundred and fifty hands and may lose more than it normally would. By playing ten rounds, the agents performance can be judged over a much larger test set. 250

hands per round was chosen because it is equal to about 3 hours of play in a casino.

During both phases, there are three other players playing in the game: an Optimal player, a Basic player, and a Random player. These players act as the control players so that the Q-Learning agent's performance can be tested against the performance of the control players. The Random player always chooses random actions. The Basic player plays a similar strategy to the dealer in that the player decides to hit if its card sum is less than seventeen and stay otherwise. However, if the hand is splittable, the player always chooses split. Lastly, the Optimal player plays the best known strategy to date defined by Edward Thorp's book "Beat the Dealer". An example of the strategy is understandable by this chart:

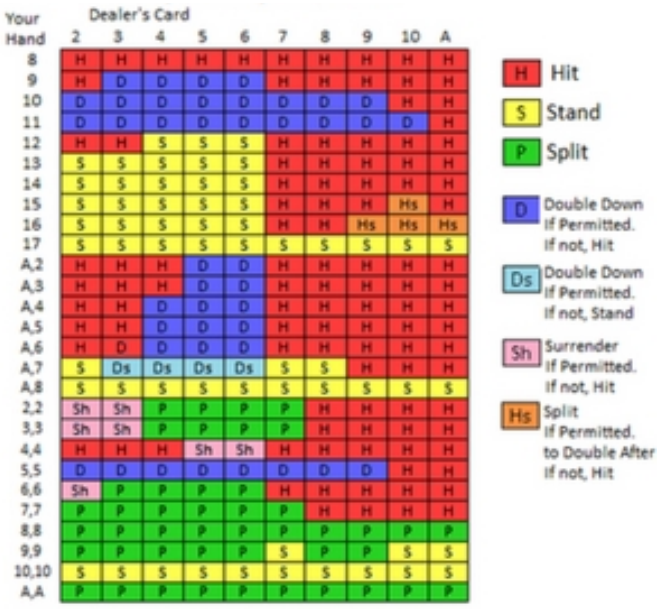

Figure 3: Optimal Play Chart Example (onlineblackjack-realmoney.org)

This chart serves as a target for this project. The Q-Learning agent should o find Q-values associated with the appropriate actions that are close to or better than the actions displayed in the chart above.

After the environment, all of the players, and the Q-Learning agent were implemented, there were numerous experiments run. The goal was to test the different approaches to designing the Q-Learning agent in order to improve its performance while also determining the affects of the different approaches. In these experiments, the parameters of the agent(including alpha, discount, and epsilon), the rewards system of the environment, the state representations used by the agent, and the betting strategy of the agent were all altered to achieve different results. Therefore the best Q-Learning agent design could be found.

Experiments

A note to begin that applies for the following experiments but is subject to change: all of the players for each of these experiments bet 2 dollars, the minimum bet, for every game played. Also, the approximate win percentage for each of the control players in all experiments were as follows: the Optimal player won 41% of its games, the Basic player won 30% of its games, and the Random player won 28% of its games across the different experiments. Observations made about experiments explained below were made with these performance results in mind.

Parameters:

For the first experiment, the goal was determine the best parameters (alpha, discount, and epsilon) that allowed the agent to perform the best. All three of parameters were adjusted between the range of greater than 0 and less than or equal to 1. The results were quite clear that the smaller the values of the parameters the better the agent did. For these tests, the agent's state space was equal to the sum of the cards held, whether or not there was an ace in the hand, the dealer's face up card, and whether or not the hand was splittable. The rewards for a win was +1, 0 for a push, and -1 for a loss. Some example results are below:

Alpha	Discount	Epsilon	Win %	Profits
0.1	0.25	0.25	39.16	-35.1
0.5	0.5	0.5	35.04	-82.5
0.9	0.9	0.9	30.96	-115.5

Figure 4: alpha, discount, epsilon experiment data

There is a clear correlation between lowering the values of parameters and the increased performance. The reasoning for this is due to the fact that the smaller values allow the agent to learn slowly. Adjusting its Q-Table with small changes allows it to get closer to the true Q values of its actions based on consistency. With the higher parameter values, the agent's Q-Values swung greatly with every experience. Therefore the most recent experiences always had the greatest affect on the associated Q-Value. With these results, for the next experiment the agent's parameters were set to an alpha equal to 0.1, discount equal to the 0.25, and epsilon set to 0.25 and the state space remained the same.

Rewards:

The next experiment involved adjusting the reward system for wins, pushes, and losses. Similar to the adjusting of the parameters, the rewards for a win were adjusted between a range of greater than or equal to +1 and less than or equal to +10. The rewards for a loss were adjusted between a range of greater than or equal to -1 and less than or equal to -10. The rewards for a push were adjusted between a range of less than or equal to 0 and greater or equal to -5. The reason for this push reward range was due to the fact that a push result is not a win result for the agent, but it is also not as much a negative result as a loss. Some example results are below:

Win	Push	Loss	Win %	Profits
5	0	5	41.32	-19.7
10	0	-10	40.0	-32.6
5	-2	-5	39.6	-52.7

Figure 5: reward experiment data

The results from these test show that increasing the scalar value of the rewards for a win or loss helped the player marginally win more games. However, the win rate rate increased to a limit of a reward value of 5 and -5 respectively, but when raised above or below these values, the win percentage began dropping. This means that the rewards need to be tuned to represent the true value of the game result. If the results are weighted too much or too little, it affects the agent's learning in a negative way. Lastly, adjusting the push reward in the negative direction seemed to also reduce the win percentage of the agent. This also agrees with the idea that the rewards must match the value of the result in the game. In Blackjack, the player neither gains or loses money when the result is a push. Therefore the agent learns nothing from the experience. With these results, for the following experiments, the win reward was set to +5, push reward set 0, and the loss reward set to -5.

Decreasing Epsilon:

The next experiment took another look at the epsilon parameter, this time decreasing it over the learning phase. Since the learning phase was 100,000 games long, the epsilon factor was reduced every 10,000 games or every 1/10 of learning. With the knowledge gained from the previous two experiments, the aim of this test was to see if by decreasing the epsilon over time, the agent could attempt to learn everything it can at first and then begin to focus more on improving its early experiences. The idea was to simulate human reinforcement learning; a human would try all actions at first and then slowly focus on the actions that achieved the desired goal. Three decreasing epsilon tests were run based on observations made in the parameters experience. Those three tests run: epsilon equal to 1 reduced by 0.1, epsilon equal to 0.5 reduced by 0.05, and epsilon equal to 0.25 reduced by 0.025. All three of these tests resulted in the epsilon being reduced to 0 by the end of the learning phase. With the rewards set by observations in the previous experiment and the state space remaining the same; the results of the three tests are:

Epsilon	Delta	# of games till change	Win %	Profits
0.25	0.025	10,000	47.12	-52.4
0.5	0.05	10,000	44.2	-10.2
1	0.1	10,000	43.68	-12.0

Figure 6: decreasing epsilon experiment data

The reason for retrying a higher epsilon from the start was to see if the idea of trying everything then reducing exploration would allow it to learn any better than a standard low epsilon. Following the idea that the agent doesn't understand the game in the beginning then learns over time; therefore the epsilon should reflect this. Interestingly, the epsilon starting at 0.25 again was the best, but the reduction over time increased the agent's performance greatly. This is due to the fact that as the agent learns more about the game over time, it does not need to explore as much and can focus on improving the knowledge it already has.

During the learning phase, all of the agents' performance was tracked in order to see the growth of the agent's knowledge over its learning time. The chart below shows the average rewards for ever 1000 games played by the players:

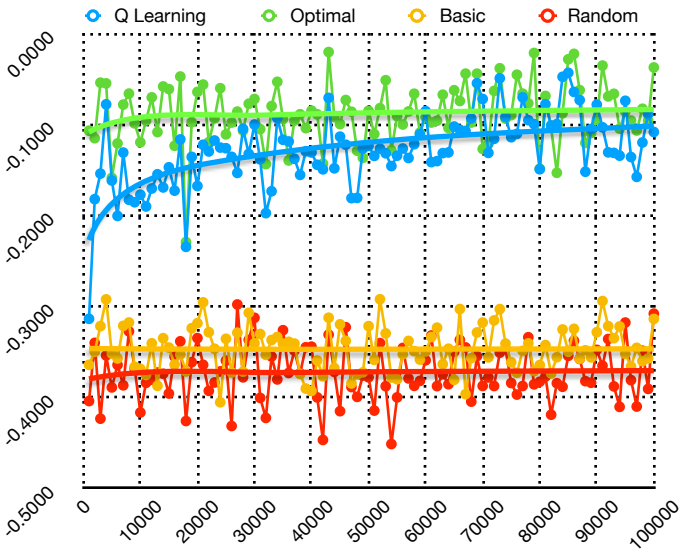

Figure 7: decreasing epsilon average rewards graph

From this figure, the Q-Learning agent starts out performing the same as the Random and Basic players but it quickly starts learning the Q-Values of the state action pairs. At 40,000 games the Q-Values begin their convergence and the agent nears the performance of the Optimal player. At about 80,000 games of learning, the agent is almost matching the Optimal player in terms of average rewards,. In the finally 20,000 games the Q-Values have converged and average rewards become extremely close to that of the Optimal player. Even more interesting is that when looking at the trendlines, it appears that the Q-Learning agent never matches the performance of the Optimal agent. However, as previously seen the Q-Learning agent takes. This is most likely because the Optimal agent is consistent, where as the Q-Learning grows over time.

Smaller State Space:

At this point, having an agent that could learn blackjack well, the Q-Values learned by the agent were analyzed. The agent's Q-Table was very large with 2738 Q-Values held at the end of learning. For this reason, the next experiment attempted to shorten the state space in order to reduce the memory consumption and increase the speed of the agent's decisions. The state space used by the agent was reduced from the previous four attributes to only two attributes which included the sum of the cards held by the agent and the dealer's face up card. Keeping the best parameters, rewards, and decreasing epsilon from the previous experiments, the results of the smaller state space were:

State Space	Win %	Profits
(Sum of cards, ace?, Dealer's card, split?)	47.12	-52.4
Sum of cards, Dealer's card)	43.4	-19.8

Figure 8: collapsed state experiment data

The result of reducing the state space was that the win percentage and the profit loss were decreased. This means that with the smaller state space the agent did not learn how to win as well but it played in a risk averse manner; allowing it to not lose as much money. The resulting Q-Table count was 710 values; a much smaller table than before. However, there was no apparent playing speed difference than the larger state space, the learning and testing phase each remained approximately at 20 seconds and 1 second respectively. However the convergence rate appears to be quicker than the larger state space. The graph of the average rewards of learning phase is here:

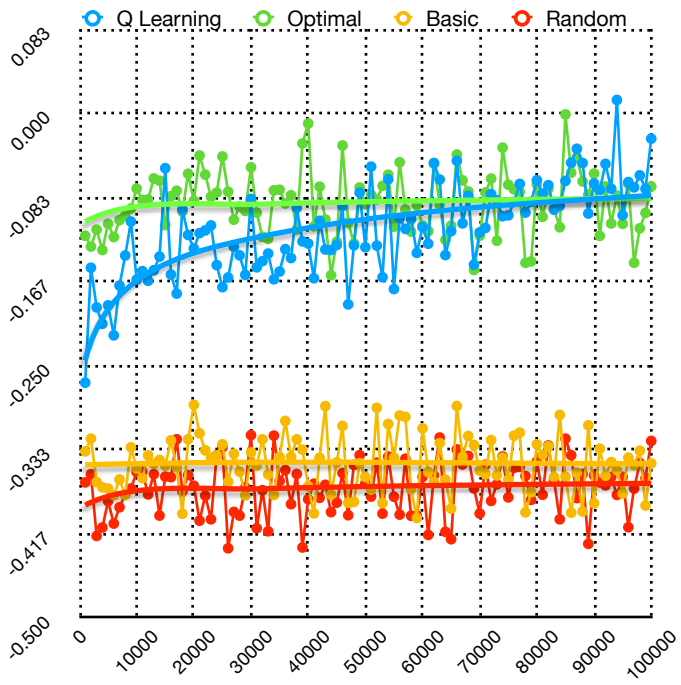

Figure 9: smaller state space average rewards graph

This graph of the average rewards during learning shows that the agent matches the optimal agent much more closely than that of the increased state space agent. When looking at the trendlines: at about 40000 games played, the agent is much closer to the Optimal agent performance and the Q-Values are converging. At 80000 games, the agent is now matching the Optimal player with converged Q-Values. For the last 20000 games, the Q-Learning agent and the Optimal agent are playing almost identically. This shows that with the smaller state space, the Q-Learning agent is able to learn Blackjack in the same way the Optimal player knows how to play the game. While this was an interesting result, the goal of this project was to attempt to learn a better strategy than the current optimal strategy. For this reason, the choice was made to continue to explore the ability of the Q-Learning agent with the enlarged state space as the betting strategy of the agent was examined.

Q-Value Analysis(Large State Space):

The other part of Blackjack that these experiments did not focus on was the betting aspect of Blackjack. The results show that even though the Q-Learning agent wins more games; it is still losing more money than the Optimal player. This is due to the difference in the policies learned by the agent and the policies of the Optimal player. When comparing the policies, it shows that the agent is not learning risk vs. reward. When the agent has a strong hand and can double to increase profits, it chooses to hit, while the optimal agent doubles. Similarly, the Optimal player

chooses to stand instead of split on a splittable hand, because it favors the probable win over the chance of losing both hands after the split. The visual representation of the examples explained:

Player Sum	Ace?	Dealer Up Card	Split?	Agent Policy	Optimal Policy
10	No	6	No	Hit	Double
18	No	7	Yes	Split	Stand

Figure 10: example Q-Values found by agent

Both of hands are strong from the point of view of the player and therefore the player has a high chance of winning the game. However, the strategy of the Q-Learning agent and the player are different. The Q-Learning agent in the first hand in table chooses to Hit while the Optimal player Doubles, the agent thereby misses out on an opportunity to make a bigger profit. The Q-Learning agent doesn't see the small risk higher reward feature of the hand. At the same time in the second hand, the sum of the cards is already high but the hand is splittable. If the player decides to split it could have two chances to win the game. The Optimal player chooses to stay because there is already a high probability the player will win the game. The Q-Learning agent chooses to split instead in attempt to give itself an extra chance to win. This is a risker choice as the agent now has two weaker hands. With this hand the Q-learning agent doesn't see the value of taking the safe bet.

This missing risk vs. reward knowledge explains why the profits of the agent are lower than that of the optimal agent even when the agent wins more games than the Optimal agent. An example of the profits and win percentages using the best decreasing epsilon agent from the earlier experiment compared to the rest of the control players is here:

Players	Win %	Profits
Q-Learning	47.12	-52.4
Optimal	40.64	-17.5
Basic	29.24	-31.0
Random	27.92	-151.6

Figure 11: decreasing epsilon results for all players

Here the table shows that winning more hands does not necessarily translate to higher profits. The policy in which a player uses in certain states can both decrease profits and increase the game win percentage as well as in the opposite direction. Since an agent who wins games but loses money in Blackjack is not a very affective player, the next experiment involved giving the Q-Learning agent the ability to learn how to bet.

Betting:

For this experiment, there were two adjustments made to the Q-Learning agent to examine the affects of betting on the agent's profits. The first adjustment was to increase the agent's standard bet from 2 to 4. While a simple change, the idea was to see if betting a small amount more could reduce the loss of the player and take advantage of the higher winning percentage. For comparisons, all of the other agents bets were increased from 2 to 4 as well. The second adjustment was giving the Q-Learning agent a second Q-table to hold the Q-Values of its state-action pairs for betting. The state space for this learning was simple in that it only had a discretized level of profit or loss. For example, all of the points where the agent had lost money greater than or equal to -2000 and less than or equal to -1000 were considered one state. This made the state space into ranges of losses and gains, so that the agent could learn the best policy for the profit/loss range. The actions of the agent were as follows: double the current bet, bet the minimum allowed bet, bet the maximum allowed bet, bet more than the current bet, and bet less than the current bet. Where the starting bet of the agent was 15, betting more and betting less increased or decreased the agent's bet by 15. If the player bet the minimum, the player's bet was set to 15, and betting the maximum set the player's bet to 1000. Finally, the rewards for the betting were equal to the profit/loss the player received from playing the bet.

The same agent that learned the card playing strategy in the earlier experiments is now learning a betting strategy at the same time. Using the best parameters, the two reward systems, and decreasing epsilon for both the card strategy and the betting strategy. The results of the two adjustments to the agent are below:

Betting	Win %	Profits
Standard	47.12	-52.4
Increased Standard	46.36	226.6
Q-Learning	46.48	1496.65

Figure 12: betting adjustments experiment data

Also included are the results of the control players standard bets being increased:

Player	Win %	Profits
Optimal	41.04	232.2
Basic	30.72	23
Random	28.20	-54

Figure 13: control players increased betting data

Increasing the bets of all the players had positive results. Increasing the betting amount allowed the Q-Learning agent and the Optimal player to increase their profits. One observation to note is that the Q-Learning agent's win rate was a little lower during this experiment which possibly explains the lower profits than the Optimal player. The definitive result of betting more allowed the agent to take advantage of its higher win percentage.

When given the ability to learn a betting strategy, the profits increased greatly. However upon deeper examination, two negative indicators were found. First, the Q-Values never converged. As similar profit/loss ranges not experienced enough to come up with a real strategy. The agent simply just tried to bet bigger amounts in hopes of gaining a bigger profit. Secondly, the agent never learned the negative value of losing large amounts of money. At some points, the losses of agent fall into the negative millions. Meaning that the agent would need a large amount money in order to come out profitable. With these negative results, it seems that the agent's best chance in its current state is to simply bet larger amounts than the minimum bet it was previously using. More experiments on the state space and the available actions are needed to give the agent the ability to learn how to affectively bet.

Conclusion

The results of this project were overall positive. The Q-Learning agent was able to learn a strategy to effective play Blackjack. The experiments show that the different parts in the make up of the agent and the rewards of the environment affect the ability of the agent to both learn and execute during testing. The part of this project that needs future work is the betting strategy. The major question out of this project is whether or not learning a betting strategy is better than adjusting betting amounts to the effectiveness of the card strategy learned. Another possible topic for future research could be to alter the state space of the agent to not only contain card information but also betting information.

The results of this project are partly due to human bias, as the main focus of the project was get the agent to learn how to play the cards in order to win more games than the Optimal agent. The secondary goal of the project was for the agent to learn how to earn profits. Lastly more trials are needed to determine the effectiveness of the Q-Learning agent. The results of this project were gathered over a moderate amounts of samples. More in depth testing with larger amounts of games in both learning and testing would provide more concrete results and analysis. Lastly, a good test for the agent would be to learn and play in Blackjack environments with different sets of rules. This would test the agent's adaptability to different Blackjack games and in which games the agent is mot affective.

At the very leas this project shows that Reinforcement learning can be applied to Blackjack. There is room for more research into the effectiveness of artificially intelligent agents playing in casinos and compare them to the strategies used by humans. With more research, there could be a new strategy to blackjack found that is definitively better than Thorpe's current optimal strategy.

References

Dawson, R. 2018. *Playing Blackjack With Machine Learning.*

Granville, Charles de. *Applying Reinforcement Learning to Blackjack Using Q-Learning,* University of Oklahoma, Norman OK.

Howard, P. *Optimal Blackjack Strategy*, Department of Math, Texas A&M University, College Station, TX.

Kendall G., & Smith C. *The Evolution of Blackjack Strategies*, School of Computer Science & IT, University of Nottingham, Jubilee Campus, Nottingham, UK.

Russel, S. & Norvig, P. 2010. *Artificial Intelligence: A Modern Approach*, Pearson.

Sutton, R., & Barto, A. 1998. *Reinforcement Learning An introduction.* MIT Press.

Wong, S. 1993. *Blackjack Secrets.* Pi Yee Press.

Wu A. *Playing Blackjack with Deep Q-Learning*, Management Science & Engineering, Stanford University, Stanford, CA.

Xi, H. *Basic Strategy*, Department of Computer Science, Boston University, Boston MA.