A Reinforcement Learning Agent Plays Blackjack

INM426 Software Agents: Adam Gaventa and Asaf Gazit

1. Introduction

This paper will apply reinforcement learning techniques to the game of Blackjack. Our goal is to explore an agent's acquisition of the optimal policy, to test convergence in a stochastic environment and to examine how attitude towards risk might change the action mapping for the best learnt policy.
Blackjack is a game of chance that is played against the "house", represented by a dealer. The goal is to end the game with the total value of dealt cards higher than those dealt to the dealer. After each card is dealt a player may request another card, to "hit", or "stay" with his current total. If a dealt card causes a player's (or the dealers') total to go over 21, they are "bust" and lose the hand.

As blackjack is a casino game, a house edge is built into the game, such that even a perfect player will lose money over the long run. As such, our goal in this paper will be to find a strategy that minimises a player's expected loss. The average player, who does not play an optimal strategy, can expect to lose £4 for every £100 staked (expected value of 0.96). A 1956 paper showed that an optimal strategy can achieve a loss of £0.62 for every £100 staked (expected value of 0.9938) [1].

For our analysis, we will use a modified version of the rules, without "splits", "insurance", and with a "blackjack" pay-out of 1:1. An optimal strategy in our variation will be expected to lose more often than the optimal strategy for the "full" version of blackjack.

1.1. Game rules

1.1.1. Definitions

"Card value" is the face value of the card. Royal cards all have a value of ten and an ace may be either one or eleven, as best fits the hand.
"Hand value" is the sum of the card values for a player's hand.
"Hit" is the term for asking for another card to be dealt.
"Stay" is the term for confirming you would not like to be dealt any more cards. This will end the hand.
"Bust" is the term for when the hand value is 22 or above. The player automatically loses the game.

1.1.2. Gameplay

Episode sequence:
1) Two cards are dealt to both the player and the dealer. The player sees his own cards and one of the dealer's cards (the "hole card"), which is dealt face up.
2) The agent plays the game until the game reaches a terminal state. These are either: the player chooses to "stay", the player's hand value is 21, or the player is "bust".
3) If the player is not "bust" the dealer then plays out his hand.
4) Hand values are compared, the winner is decided, and the winnings are paid.

The dealer's game follows a predefined set of rules: as long as the dealer's hand value is not between 17 and 21, the dealer will "Hit". If the hand value is in that range, the dealer will stay.
It is important to note that the dealer (i.e. the casino) has a double advantage; the dealer plays after the player, so that when the player busts the dealer wins without playing, and the dealer wins a tie.

1.2. Stochastic elements

Unlike a deterministic application of the Q-learning algorithm, such as for finding the best path through a network, blackjack introduces unknowns into the agent's environment so that the agent cannot guarantee a choice of any particular hand value following a given action.
From the agent's perspective, there are two stochastic elements to overcome in order to learn an optimal policy:

- The "Hit" action calls for an additional card, however the agent does not know which card will be dealt.
- Since the agent plays before the dealer the winning criteria (comparison to the dealer's hand) is unknown while the agent plays the game.

This work reviews how we have applied reinforcement learning, **specifically, Q-learning**, first described by Watkins [2], to a problem of a stochastic nature in order to investigate the stochastic nature of the game.

2. Basic Case

2.1. State Transition Function

We will define a *state* as the hand value of the agent's cards, which can take values from 4 to 21. For ease of analysis, we give every "bust" a value of 22. We include the dealer's face-up card in the agent's state.

The agent's *actions* will be one of {"hit", "stay"}.

The state transition function is defined as $s_{t+1} = \delta(s_t, a_t)$.

In other words, it defines the next state for the agent given a state and action. In our case, any "stay" action will move the agent back to its current state, and the hand (episode) will terminate. A "hit" action will move the agent to another state depending on the value of the next card dealt.

2.2. Reward Function

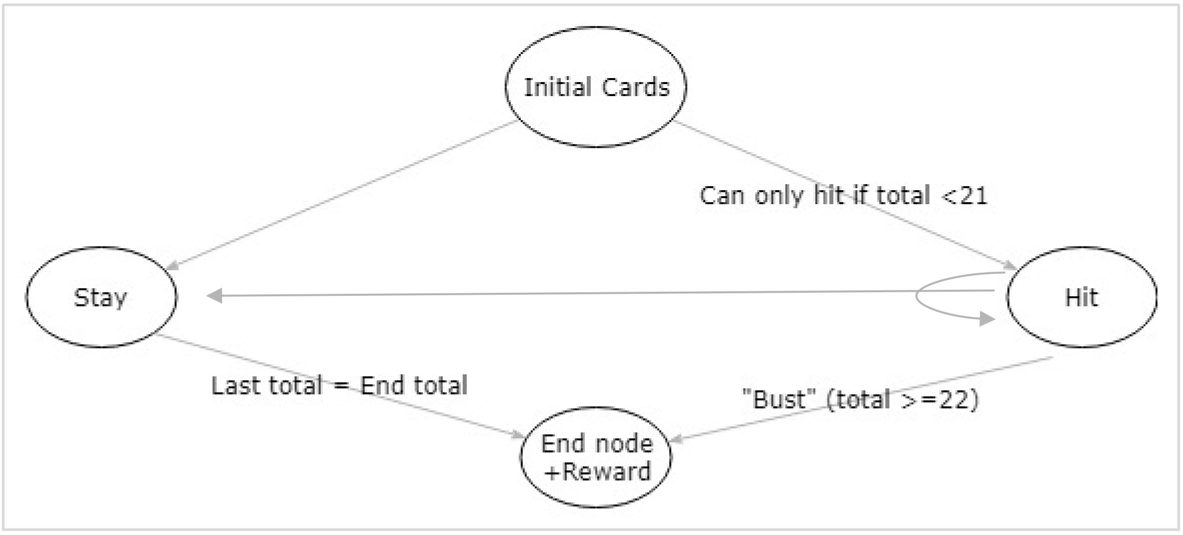
The agent receives a reward only for a limited set of states and actions. Receipt of a reward is dependent on the outcome of the dealer's hand. Rewards are based on the actual winnings. Assuming a starting bet of £100, the agent will receive a reward of £100 for beating the dealer and -£100 for losing to the dealer. The outcome of the game, however, will only be revealed when the agent arrives in a terminal state (stay, bust or 21), at which time the agent's hand is compared to the dealer's. There is no reward (zero reward) for the "hit" action.

2.3. Choosing a Policy

We initially chose to implement an "ε-greedy" policy, which ensures the agent initially "explores" its environment through a random choice of action more often than it "exploits" its knowledge by choosing an action which maximises future rewards. The "ε-greedy" policy progressively adjusts a parameter such that the agent moves from exploring (random actions) to exploiting (maximise expected reward) as the number of episodes increases.
Due to some surprising results (discussed later), we decided to use an 'explore-only' strategy.

2.5. Graphical Representation

Graphical representation of the game, in terms of states, actions, and rewards:



2.6. Initialisation

To begin implementation of Q-learning, we require a Reward Matrix to assign rewards for agent's actions, an initial (zero) Q-matrix, and initial parameters for the algorithm.
Default parameters are: Gamma = 0.9, alpha = 0.2, epsilon-greedy=False, number of episodes = 50,000.

2.6.1. Reward Matrix

The reward matrix contains the states and the rewards for each possible action. Rewards are only given on termination of the hand, when the agent's hand is compared to the dealer's. Once the agent is "bust", a "hit" is no longer a valid action.

State (agent's total)	State (Dealer's card)	Hit	Stay
4	2	0	R
5	2	0	R
6	2	0	R
7	2	0	R
8	2	0	R
9	2	0	R
10	2	0	R
11	2	0	R
12	2	0	R
13	2	0	R
14	2	0	R
15	2	0	R
16	2	0	R
17	2	0	R
18	2	0	R
19	2	0	R
20	2	0	R
21	2	0	R
22	2	n/a	R

2.6.2. Discount Factor

The discount factor, γ, determines the extent to which the agent values expected *future* rewards as compared to the immediate reward of an action. A single blackjack hand is short, since a terminal state is usually quickly reached, so we wish to associate the last value with the action leading to it. We prefer the agent to strive for long-term reward, since a short-sighted agent may bias an outcome which is merely due to random chance. Therefore, we set a high default discount factor.

2.6.3. Learning Rate

The learning rate, α, determines the extent to which the agent takes into account the error between the estimate of future return and the actual return. As mentioned earlier, we wish our agent to try and capture the statistical nature of the game. Specifically, we aim for our Q-matrix to represent a preference for the "stay" action with higher state values and "hit" with lower hand values.
In the case of a game with stochastic elements, even a "perfect" player will lose a game he is expected to win. A large learning rate will associate mostly the latest game result to the Q-matrix value and override values the agent has learnt over a longer period. We prefer to try and capture a more holistic perspective. Therefore, we set a low default learning rate.

2.7. Q-learning Implementation

The Q-learning algorithm updates the Q-matrix with each episode of the implementation. For each step within an episode the agent chooses an action and updates the Q-matrix with the expected future returns of that action, according to the following formula:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma max Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

A demonstration of how the Q-matrix is updated is shown below:

State	Dealer	Hit	Stay
4	2	0	0
5	2	0	0
6	2	0	0
7	2	0	0
8	2	0	0
9	2	0	0
10	2	0	0
11	2	0	0
12	2	0	0
13	2	0	0
14	2	0	0
15	2	0	0
16	2	0	0
17	2	0	0
18	2	0	0
19	2	0	0
20	2	0	0
21	2	0	0
22	2	n/a	n/a

Agent is dealt a 5 and 8, for a total of 13. The dealer is dealt a 2.

The agent chooses to "hit" and receives a 10. The agent has a hand value over 22, so is "bust" and loses the hand.
The reward received is -100.

The Q-matrix value to update is highlighted.

The new value is (assume α = 0.2, γ = 0.9):
0 + 0.9(-100 + 0.2*0) – 0 = -90

State	Dealer	Hit	Stay
4	2	0	0
5	2	0	0
6	2	0	0
7	2	0	0
8	2	0	0
9	2	0	0
10	2	0	0
11	2	0	0
12	2	0	0
13	2	-90	0
14	2	0	0
15	2	0	0
16	2	0	0
17	2	0	0
18	2	0	0
19	2	0	0
20	2	0	0
21	2	0	0
22	2	n/a	n/a

The new Q-matrix value will be used whenever an agent returns to the same state (hand value=13, dealer's card=2). The agent has learned (tentatively – we hope the agent will visit this state many times in order to learn the true value) that a "hit" in this state carries a negative expectation.

2.8. Varying the Parameters

A full grid search (every combination of parameters) would take prohibitively long to run, so instead for each experiment we vary a single parameter, holding all others at their default values. We vary parameters as follows:

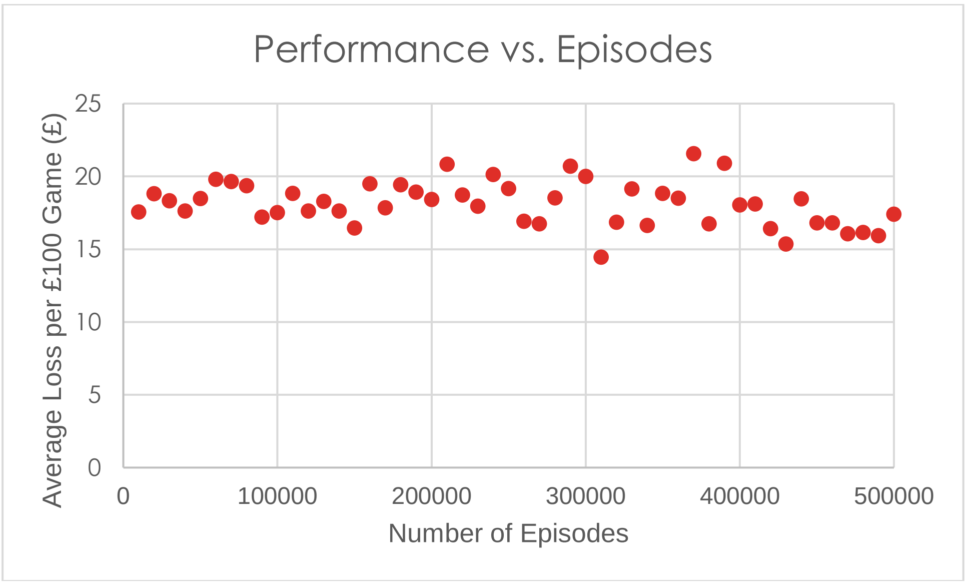
Parameter	Default Value	Values
Alpha	0.2	0.4, 0.6, 0.8, 1.0
Gamma	0.9	0.2, 0.4, 0.6, 0.8, 1.0
Epsilon policy	FALSE	TRUE
Number of Episodes	50,000	10k, 100k, 200k, 500k

In total there are fourteen experiments, excluding the default values. A full grid search would have been three hundred experiments.

To measure performance, after the agent has completed its learning we will allow the agent to play 10,000 simulated games. We will record the proportion of wins out of the 10,000 games, and the average winnings per game in real terms. We will also consider the final learnt matrix of actions, and we will explore convergence of this matrix i.e. how often the preferred action changes for a given state.

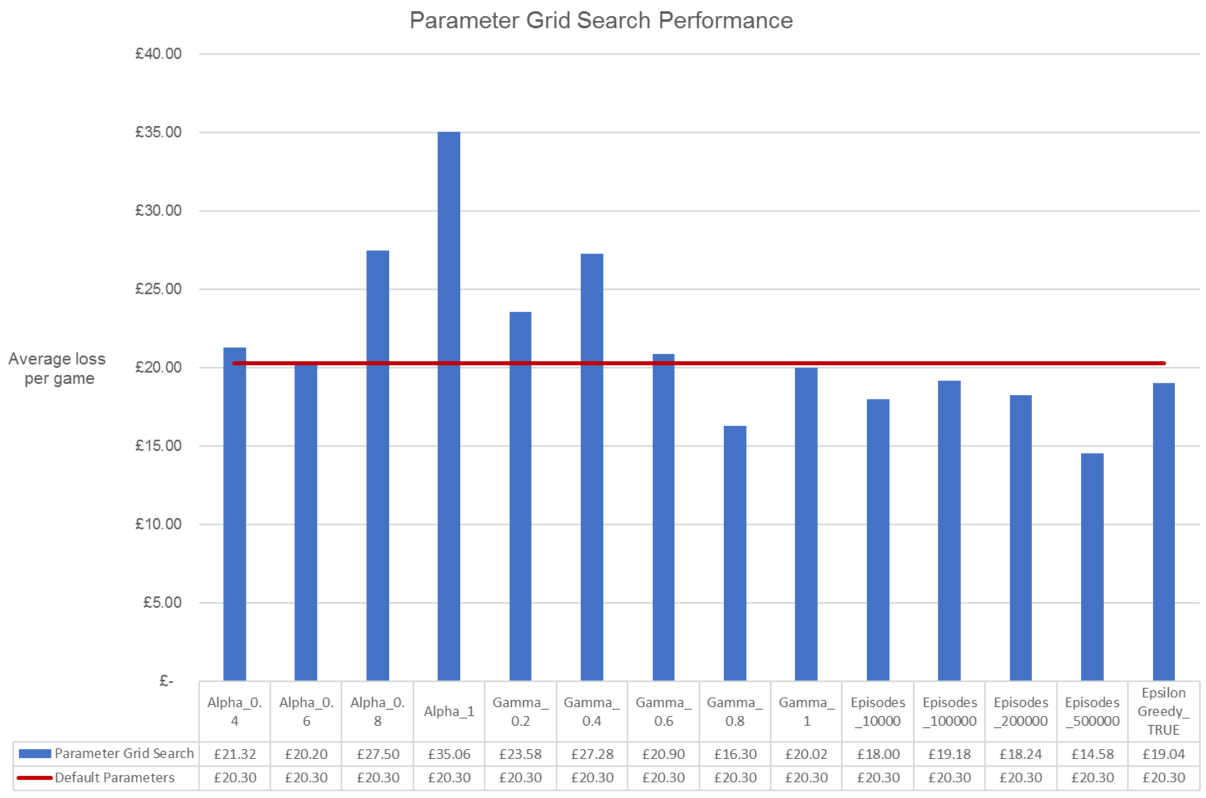
2.9. Results & Discussion

2.9.1. Performance vs. episodes



As the number of episodes increases, performance *does not* increase significantly. We see high variance in the performance throughout. There is no direct correlation between the number of episodes and the performance measure. We investigate this phenomenon in Section 3 – Advanced Cases.

2.9.2. Finding the best parameters



	Alpha_0.4	Alpha_0.6	Alpha_0.8	Alpha_1	Gamma_0.2	Gamma_0.4	Gamma_0.6	Gamma_0.8	Gamma_1	Episodes_10000	Episodes_100000	Episodes_200000	Episodes_500000	Epsilon Greedy_TRUE
🟦 Parameter Grid Search	£21.32	£20.20	£27.50	£35.06	£23.58	£27.28	£20.90	£16.30	£20.02	£18.00	£19.18	£18.24	£14.58	£19.04
🟥 Default Parameters	£20.30	£20.30	£20.30	£20.30	£20.30	£20.30	£20.30	£20.30	£20.30	£20.30	£20.30	£20.30	£20.30	£20.30

In the table above, lower values are indicative of better performance i.e. lower average loss per game.

We observed several interesting correlations between the parameter values and the agent's performance. For example, high gamma values seem to have worked better than lower ones and lower alphas performed better than higher ones. This confirms our hypothesis regarding those parameters (sections 2.5.2 & 2.5.3) and justifies our choice of default values.

Perhaps the most surprising result relates to the implementation of "ε-greedy". In an ε-greedy implementation the agent spends about approximately half the time exploring randomly and half the time exploiting the best policy. The resulting Q-matrix of this strategy surprised us, suggesting a strategy of "hit" in all states except for 21. This is clearly a losing strategy.
Upon further investigation we noticed that the agent appears to learn a reasonable policy in the exploration stage, but learnt a losing policy as described after exploiting.
We suggest that this is due to the nature of blackjack as game with *negative* expected value, even in an optimal strategy. The agent continued to lose even when it had learnt the optimal strategy, backpropagating negative values to the Q-matrix for "stay" actions. The Q-matrix slowly changed to favour "hit" over "stay", resulting in our anomalous Q-matrix described above.
We addressed this issue by changing our algorithm and performance measure: to only explore during the agent's learning phase; and to measure performance using the final Q-matrix, without further updating the matrix.

Another issue that our grid search tried to address was the number of episodes required to capture a reasonably effective policy for Blackjack. Notably, our best performing Q-matrix was found after half a million experiments. We could not find a statistically significant result that showed us that a larger number of episodes will result in a more optimal policy. We attribute this to the stochastic nature of the agent's environment (motivating our 'advanced' variation, to be discussed below). This phenomenon also influences the 'performance vs. results' (section 2.8.1), where we see high variance in performance.

We found that certain parameter values significantly decrease the performance of the algorithm. For example, high alpha and low gamma yielded exceptionally poor results. At this point we are very cautious in drawing definite conclusions about the behaviour of the agent under different parameters – as we can see in section 2.8.1, performance is similar, and changes are likely not statistically significant. Further analysis will be discussed after introducing the advanced methods, and their purpose, as an extension to this section.

3. Advanced Cases

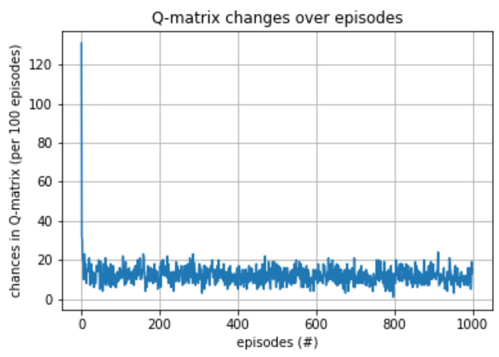
3.1. Variable Learning Rate

3.1.1. Motivation: Convergence and Stability

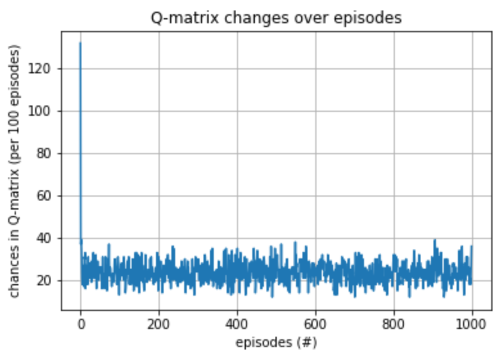
While running the above experiments and examining the Q-matrix associated, we noticed that the Q-matrices did not became "stable", in the sense of converging to a consistent optimal game policy, even for the same experiments under the same parameters. The stochastic nature of the agent's environment is affecting the stability of the Q-matrix.
We wanted to measure the convergence of our matrix and explore whether there will be a given number of episodes, or indeed any other key parameter, that leads to a convergent Q-matrix.

We define a "convergence index" to measure the changes of exploiting actions per state in our matrix, per number of episodes. In other words, we count the number of times an episode has updated the Q-matrix in such a way that the best action (maximum Q-value for that state) has changed (i.e. from "Hit" to "Stay", or vice versa). As a single episode can only change a single value and our goal is to understand the system's convergence, we have aggregated those changes per 100 episodes.
NB: this type of convergence is different to convergence to the optimal policy, representing rather the number of changes to the Q-matrix as learning progresses.

The results of this index measurement revealed that the Q-matrix does not become entirely stable for *any* number of episodes, under any set of parameters (see figure below).



Number of changes in the Q-matrix, using constant alpha=0.2, per 100 episodes (100,000 in total)



Number of changes in the Q-matrix, using constant alpha=0.6, per 100 episodes (100,000 in total)

We observe that the number of matrix changes is relatively stable. Lack of convergence is partly due to the stochastic nature of blackjack, where an unlikely loss in the final episode of a given state will back-propagate a negative reward and may change the preferred action for the agent. Simply lowering the starting alpha of alpha can mitigate this effect somewhat, however, a more considered approach is to lower alpha progressively throughout the learning phase. We investigate a number of approaches to implement dynamic alpha, described below.

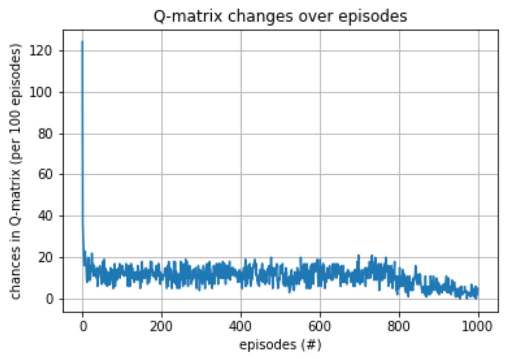
We also would like to note that lowering alpha leads to Q-matrix convergence only because the update values become too small to change the matrix. The stochastic elements are still affecting the agent in the same way, however we are reducing the effect of the error on the matrix. Therefore, this is not a "natural" convergence or a truly stable Q-matrix (which would represent the optimal policy). Variable alpha strives to reach a convergent matrix under stochastic elements by trying to mitigate the effect of those elements over time, as the matrix would not converge otherwise.

3.1.2. Linear variable alpha with (large) grace period (function linear_alpha)

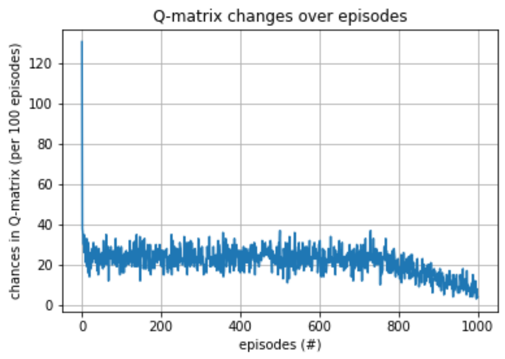
Linear alpha reduces alpha each episode by a constant rate. The method assumes a grace period in which the alpha is constant and not depreciated. The constant rate is defined as 1/(number of episodes-number of grace episodes).
The agent registered a 40.1% win percentage and a £18.80 average loss per game.

3.1.3. Asymptotic variable alpha (function linear_alpha_for_convergence)

This approach assumes a constant alpha for the most part of the learning phase and starts reducing alpha only towards the end of that phase. A parameter is set to specify the portion of the game in-which alpha should be depreciated. The depreciation factor also diminishes over time, thus resulting in a non-zero very low alpha, asymptotically approaching zero. The rate of alpha depreciation is defined as (current alpha)/(number of episodes * diminish range).



Number of changes in the Q-matrix, using asymptotic variable alpha, per 100 episodes.Starting alpha=0.2



Number of changes in the Q-matrix, using asymptotic variable alpha, per 100 episodes.Starting alpha=0.6

Algorithm performance over default parameters, reducing alpha in the last 25% of episodes: the agent registered a 41.56% win percentage and a £15.88 average loss per game.

3.1.4. Adaptive alpha (function adaptive_alpha)

Adaptive alpha changes alpha each episode according to a small performance check. Adaptive alpha is somewhat inspired by neural networks' adaptive learning rate: if an increase of performance is registered between episodes, the alpha value will be reduced (and vice versa). This method is computationally expensive as it requires a performance check each episode.

Algorithm performance over default parameters with a 20-game performance measure on every episode: the agent registered a 32.67% win percentage and a £33.66 average loss per game.

3.1.5. Stochastic adaptive alpha (function stochastic_adaptive_alpha)

This is a very similar to the adaptive alpha algorithm, but instead of running a performance check each episode, the algorithm makes a performance check every N episodes. This approach tries to mitigate the high cost of adaptive alpha.
Algorithm performance over default parameters with 20 games as performance measure every 100 games: the agent registered a 41.22% win percentage and a £16.56 average loss per game.

3.1.6. Summary

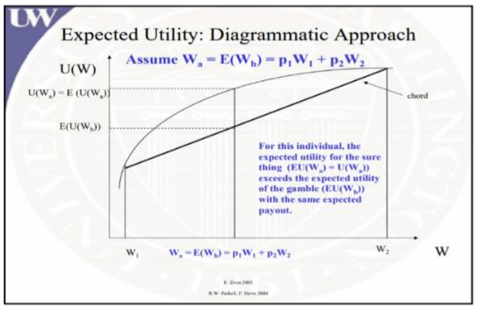
Overall, the asymptotic variable alpha performs marginally better than the stochastic adaptive alpha. The worst performance was registered by the adaptive alpha, which was also the most expensive method implemented. Comparing those results to the constant alpha results (section 2.8) we can see that the better adaptive alpha methods are second only to constant alpha with half a million episodes registered. This drop in the number of episodes required represents the increase of efficiency through the use of those methods. Notably, of the adaptive alpha algorithms, the asymptotic variable alpha is the best performing algorithm and is also the most cost efficient.

	Linear variable alpha with grace	Asymptotic variable alpha	Adaptive alpha	Stochastic adaptive alpha	Constant alpha with 500,000 episodes
Average loss per game	£18.80	£15.88	£33.66	£16.56	£14.58
% wins	40.1%	41.56%	32.67%	41.22%	42.21%

3.2. Risk

In order to approach a more realistic simulation of blackjack, we explore how a player's perception of risk may influence their play-style and winnings
Our implementation of "risk attitude" is based on the expected-utility theory in which people are assumed to change preferences or actions in cases of uncertainty based on their attitude towards risk.

For example, a risk-neutral attitude will perceive the value of an uncertain event as the weighted average of the probability of the outcome times its' reward. A risk-prone attitude will value the expected reward higher than the actual expectancy of the uncertain event and a risk averse attitude would show a lower expectancy than the neutral one. An example of a risk averse attitude can be seen in the figure opposite [3].

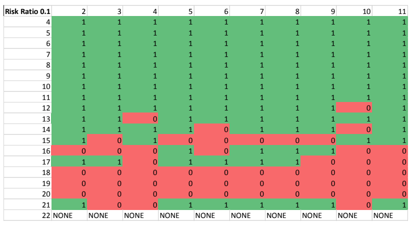

Expected Utility: Diagrammatic Approach
Assume $W_a = E(W_b) = p_1W_1 + p_2W_2$
U(W)
$U(W_a) = E(U(W_a))$
$E(U(W_b))$
chord
For this individual, the expected utility for the sure thing $(EU(W_a) = U(W_a))$ exceeds the expected utility of the gamble $(EU(W_b))$ with the same expected payout.
W_1
$W_a = E(W_b) = p_1W_1 + p_2W_2$
W_2
W

Our strategy for exploring risk perception is to modify the reward function for the agent. In our basic case we have a reward of +100 for winning a game, and -100 for losing a game. A risk-averse player with a 'risk-ratio' of 0.1 will perceive the 'reward' of losing a game as greater than its true value, and vice versa. Therefore we assign a reward of +10 for winning a game, and -100 for losing a game. A risk-prone player with a 'risk-ratio' of 2.0 will perceive the reward of winning as greater than its true value, and will therefore receive a reward of +200 for winning a game, and -100 for losing a game.
It is important to note that this is the reward function for the agent, and not the actual winnings. The game is left unchanged and so are the measures of performance.

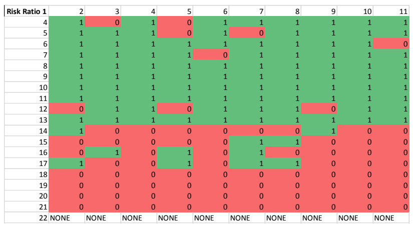
	0.1 (risk-averse)	0.5	1 (neutral)	2	10 (risk-prone)
Average loss per game	£26.90	£17.28	£13.72	£16.22	£15.28
% wins	36.5%	40.9%	42.3%	41.4%	41.9%

The agent performed best with a neutral perception of risk. A risk-prone agent performed better than a risk-averse agent.

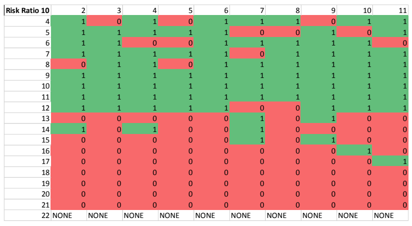
We show below the agent’s preferred actions after learning for various risk profiles. Rows represent the agent’s current state, and columns represent the dealer’s face-up card. Green (1) indicates “hit” is preferred, and red (0) indicates “stay” is preferred.



Risk Ratio 0.1	2	3	4	5	6	7	8	9	10	11
4	1	1	1	1	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1	1
6	1	1	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
11	1	1	1	1	1	1	1	1	1	1
12	1	1	1	1	1	1	1	1	0	1
13	1	1	0	1	1	1	1	1	1	1
14	1	1	1	1	0	1	1	1	0	1
15	1	0	1	0	0	0	0	0	1	1
16	0	0	0	1	0	1	1	1	0	0
17	1	1	0	1	1	1	1	0	0	0
18	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0
21	1	0	0	1	1	1	1	1	0	1
22	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE



Risk Ratio 1	2	3	4	5	6	7	8	9	10	11
4	1	0	1	0	1	1	1	1	1	1
5	1	1	1	0	1	0	1	1	1	1
6	1	1	1	1	1	1	1	1	1	0
7	1	1	1	1	0	1	1	1	1	1
8	1	1	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
11	1	1	1	1	1	1	1	1	1	1
12	0	1	1	0	1	1	1	0	1	1
13	1	1	1	1	1	1	1	1	1	1
14	1	0	0	0	0	0	0	1	0	0
15	0	0	0	0	0	1	1	0	0	0
16	0	1	0	1	0	1	0	0	0	0
17	1	0	0	1	0	1	1	0	0	0
18	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0
22	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE



Risk Ratio 10	2	3	4	5	6	7	8	9	10	11
4	1	0	1	0	1	1	1	0	1	1
5	1	1	1	1	1	0	0	1	0	1
6	1	1	0	0	1	1	1	1	1	0
7	1	1	1	1	1	0	1	1	1	1
8	0	1	1	0	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
11	1	1	1	1	1	1	1	1	1	1
12	1	1	1	1	1	0	0	1	1	1
13	0	0	0	0	0	1	0	1	0	0
14	1	0	1	0	0	1	0	0	0	0
15	0	0	0	0	0	1	0	1	0	0
16	0	0	0	0	0	0	0	0	1	0
17	0	0	0	0	0	0	0	0	0	1
18	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0
22	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE

Preferred actions: risk-ratio 0.1, 1, 10

We would expect that a ‘risk-averse’ player (i.e. risk-ratio<1) would have a more conservative play-style, choosing to “stay” more often than choosing to “hit”, since every “hit” action risks going “bust”. We find instead the opposite: as the risk-ratio increases, the agent prefers to “stay” on lower hand values. The implication is that of the two stochastic elements in the game – the agent’s next card and the final total required to beat the dealer – the latter has more of an impact on play-style than the former.

4. Conclusion

4.1. Conclusions

In this paper we implemented a Q-learning algorithm to the game of Blackjack. We showed that Q-learning can be successfully applied to a game of chance such as Blackjack, where the agent’s environment includes stochastic elements.
We investigated a number of methods to improve an agent’s expected winnings, such as varying algorithm parameters. Our investigation of the convergence stability of the agent in a stochastic environment such as ours motivated our suggestion to vary the learning rate throughout the agent’s learning phase. We discovered that the issue is caused by the dominance of one of the environment’s stochastic elements: the dealer’s hand.
The most effective method to vary the learning rate was ‘asymptotic variable alpha’, where the learning rate reduces after a given proportion of episodes has passed, and decreases thereon, approaching zero asymptotically.
Variable alpha did not fully address the issue, but it achieved faster ‘convergence’ than the standard constant learning rate.
We investigated how a player’s perception of risk affects their play-style, concluding counter-intuitively that a risk-averse agent will choose to “hit” more often, and on lower hand values, than a risk-prone agent.

4.2. Further Research

4.2.1. Further Blackjack gameplay options and optimal performance

The performance measured in our experiments is much lower than registered in the optimal cases for the game [1]. We partially attribute this to the implementation of the game. Our implementation of Blackjack was a fairly simplistic one. It represents the basic gameplay options and the game sequence, but doesn’t include more advance actions (such as “Split”, “Double” and “Insurance”) from the agent. A Q-matrix that captures all the state-action mapping of such a game would be a very large one and might require a large number of episodes to capture. We have demonstrated that the large number of episodes required could be reduced by a variable alpha method, as proposed in this paper.

4.2.2. Other Reinforcement Learning Techniques

Other reinforcement learning techniques have been proposed, such as the “Functional Systems Network”, whose performance in stochastic environments surpasses the Q-learning algorithm [4]. Further research would be able to apply more up-to-date methods to our blackjack challenge, and perhaps achieve a policy closer to the optimal.

4.2.3. Risk in games of chance

Any game of chance, such as Blackjack, may hold more than one stochastic element to them. Aggregating those effects might not be as straight forward as we think. Our understanding of utility in light of an uncertain events might be sometimes counterintuitive, thus resulting in an unexpected, or even reversed strategy. The agent's risk management is systematic in nature and allows a more coherent understanding of the uncertainties of the agent's environment. Those can be utilised to understand risk management strategies under uncertainty. We think that further research on risk attitude of agents in stochastic environment might reveal more unintuitive insights [5].

5. Bibliography

[1] R. R. Baldwin, W. E. Cantey, H. Maisel, and J. P. McDermott, “The Optimum Strategy in Blackjack,” *Journal of the American Statistical Association*, vol. 51, no. 275, pp. 429–439, Sep. 1956.

[2] C. Watkins, “Learning from Delayed Rewards,” King's College London, PhD Thesis, 1989.

[3] “https://faculty.washington.edu/ezivot/econ422/Expected%20Utility%20Theory%20EZ.pdf [accessed 11/04/2018].” .

[4] A. Y. Sorokin and M. S. Burtsev, “Functional Systems Network Outperforms Q-learning in Stochastic Environment,” *Procedia Computer Science*, vol. 88, pp. 397–402, Jan. 2016.

[5] D. Hillson and R. Murray-Webster, *Understanding and Managing Risk Attitude*. Gower Publishing, 2007.