# Reinforcement Learning

# Blackjack

Jasper Grootscholten - 20-744-553
Andrianos Michail - 20-745-931

Robert Earle

December 2020

**Abstract**

Blackjack is one of the most widely played casino games in the world. It has relatively simple rules, yet finding an optimal strategy is a nontrivial task. We explore this subject through the use of Reinforcement Learning. We show that the usage of SARSA with table lookup improves on the simple but effective mimic the dealer strategy. We also show how neural networks can be used as a function approximator, which leads to slightly worse results. Moreover, investigation is done whether the results can be improved through a technique called card counting, but only the neural network benefits significantly from the technique, making it the best performing strategy.

# Contents

# 1 Introduction

Blackjack is one of the most widely played casino games in the world, and consequently a huge source of income for the gambling industry. For example, Macao casino, one of the largest casino's in the world, made a revenue of $365 million from blackjack alone (Statista (2020)). Naturally the game has been of interest for mathematicians. Baldwin, Cantey, Maisel, and McDermott (1956) published a paper titled The Optimum Strategy in Blackjack in the Journal of the American Statistical Association, in which they were the first to try to mathematically compute the ideal strategy one should follow during the game of blackjack.

However, Baldwin et al. (1956) did not have access to the computational power available today. So, in this project we try solving the ideal blackjack strategy using the modern toolbox of reinforcement learning within our somewhat less limited computational resources. This machine learning paradigm is specifically created to solve games, and is therefore the ideal technique to solve this problem. Even though the use of a computer allows for very complex strategies, the strategies are simplified so that they could be employed by a human.

Reinforcement learning has proven to be a very successful technique in a variety of games. For example, the AlphaZero, developed by DeepMind, is possibly the strongest player in the complicated games Chess and Go (Silver et al. (2018)), and the program is entirely self-taught through reinforcement learning techniques.

The outline of this project is as follows. Initially, a discussion of the rules of the simplified version of blackjack, for which we will train a reinforcement learning agent, is layed out in Section 2. Afterwards, the methods used to train the agent are outlined in Section 3, and the performance of the trained agents is shown and discussed in Section 4. Lastly, a brief conclusion is presented in Section 5.

# 2 Blackjack

## 2.1 Rules

There is a lot of different versions of blackjack, the rules of a simple version are presented, on which our focus will be. The explanation is in detail, as we do not want to leave any ambiguity regarding the setup, but for a more comprehensive explanation of the game, we recommend finding a different source.

Our version will be played between a single player (which we control) and the dealer. Initially, 6 decks of cards are shuffled to be one collective deck of 312 cards, from which cards are dealt during each game. The cards used during play are not shuffled back into the deck, instead they are kept on a separate pile, and only when the deck does not have the minimal cards required, are all 312 cards shuffled again. This fact will be important later in Section 2.2. The minimal cards required is set to be 6 per player. With this overall setup, we can focus on individual games:

At the start of each game, the player is dealt 2 cards face up, while the dealer is dealt 1 card face up and 1 card face down. The value of the cards is 2 through 10 for the number cards, and 10 for all picture cards, aces can be worth either 1 or 11 (whichever is most advantageous to the player, if the most advantageous value is 11, the hand is referred to as "soft", as opposed to "hard"). The goal is to obtain a score as close to 21, without exceeding it (going "bust"). Here score simply refers to the sum of value of the

cards in hand, with aces having the value that leads to the best score. The score can be improved by requesting additional cards ("hitting"), until the player is satisfied and "sticks".

Once all players have completed their hands (either have a "blackjack" (explained later), went bust or sticked), it is the dealer's turn. The dealer follows a set strategy, namely the dealer will keep hitting until his hand value is 17 or higher, after which the dealer will stick.

In blackjack, you are betting that you do better than the dealer. This will lead to the following exhaustive reward scheme:

- If the player has an ace and a 10-value card, he has a "blackjack", if the dealer does not have a "blackjack", the player wins (reward 1)
- Else if the player's hand exceeds 21, the player loses (reward -1)
- Else if the dealer's hand is above 21, the player wins (reward 1)
- Else if the player's hand value is strictly above that of the dealer, the player wins (reward 1)
- Else if the player's hand value is equal to that of the dealer, it's a draw (reward 0)
- Else if the player's hand value is strictly below that of the dealer, the player loses (reward -1)

Note that this reward scheme is not completely symmetric, since we check whether the player busts before we check whether the dealer busts. This asymmetry leads to a house edge, that is the dealer, who works for the casino, is more likely to win, and therefore is expected to make a long-term profit. Consequently, as this is a zero-sum game, the player is expected to make a long-term loss.

### 2.2 Card Counting

As highlighted previously, the deck is actually not reshuffled after every game. Therefore, the player could improve his odds by keeping track of the state of the deck, through the cards that have been revealed during the game. This technique is known as card counting, first introduced by Thorp (1966), the father of card counting, and is actually not permitted by most casino's, but it is of theoretical interest for us. As humans are generally not capable of remembering exactly which cards have been revealed, systems have been developed that only require remembering a single number. We focus on the High-Low system, where the card count score variable is set to 0 whenever the deck is shuffled, then we add -1 if a card between 2 and 6 is revealed, and add +1 if a card between 10 and 11 is revealed. Within this project it is of interest if such a system could swing the house edge to a player edge. Statistically, a higher amount of high cards, which corresponds to a high score, should benefit the player.

## 3 Strategies

### 3.1 Learning in a Probabilistic Environment

Due to the nature of card games, the order in which the cards appear plays a huge role in the states the agent learns and of course the outcome. For example, if the agent the

first few times he is on an 18 against a low card decides to hit and gets rewarded for it repeatedly, the agent initially learns that it may be the correct behaviour. To minimise this effect, the agent is trained on a large amount of games, so that the same states are explored multiple times and the correct approximation function based on probabilities is learned (Sutton and Barto (2018)).

## 3.2 Heuristic strategies

To have a benchmark for our reinforcement learning based strategies, 2 simple strategies are introduced. First, the player hits or stick with a probability of 50% in all situations, which we will refer to as the "Random player". Second, and much more refined, we will let the player mimic the dealer. Dealers, always hit until their hand value is 17 or higher, and then stick. As this is the strategy most casino dealers use, it should already be fairly close to optimal, therefore it is a useful benchmark.

## 3.3 Reinforcement Learning strategies

### 3.3.1 MDP description

Before discussing the specific implementation, the general setup is introduced necessary for reinforcement learning, namely a MDP type description of the problem, that is the actions, state, rewards and episodes. The player can only choose between 2 actions, hit (denoted 0) and stick (denoted 1), hence we have our action set $A = \{0, 1\}$. Next the state $s \in S$ should describe the information available to the player, which we model by a vector containing the following variables:

- An integer containing the agent's score, ranging from 11 to 21[1]
- An integer containing the value of the revealed dealer card, ranging from 1 to 10
- A binary variable that tracks if we have at least one "useful" ace (meaning it can still take either 1 or 11 as value)
- (If we do card counting) An integer containing the card count score, ranging from -50 to 50[2]

It is important to remark that this model does not capture all information available to the player. Obviously, we lose the full history of revealed cards in this setup, and only have the card count score, but this is done purposefully because, as discussed before, a human would not be capable of tracking this fully history, and within the scope of this project, it was deemed important that the strategy can be employable by a human player. More subtly, we also lose exactly what cards the player is holding, and only have his score. Though we could technically make some inference on the dealers hand with the cards we hold (for example if we are holding 3 two's, the dealer is less likely to hold a two), this effect should be negligible. Lastly, the rewards are simply $R = \{-1, 0, 1\}$, as defined in Section 2, and we do not discount the rewards, i.e. $\gamma = 1$.

Next, we define an episode. An episode consists of an initial state, the terminal state and the inbetween states. The agent's goal is to maximise the total reward it receives in

[1]When the score is below 10, the agent should always hit (which is hardcoded)

[2]With our setup, the card count score will not go beyond this range

an episode. The next episode begins independently of how the previous one ended (Sutton and Barto (2018)). Within this project, an episode is a single game of blackjack, ending in a win, loss or draw. Multiple episodes are packed together in a set of episodes, namely all games until the deck is reshuffled, named a shuffle. Within the card counting experiments, the counter is preserved between episodes of the same set making the episodes of a set dependent on each other, but the different sets are independent. In general, 1,000 sets of episodes, corresponding to about 50,000 individual episodes, are used to train the agent in order to allow the agent to explore the different states regardless of probabilities. This amount of sets leads to a training time in the order of 1 minute for the Neural Network discussed later, which is about ideal for our research. Working with sets of episodes instead of individual episodes gives some extra freedom, as the amount of games within a shuffle can vary per strategy.

### 3.3.2 Training algorithms

With this description of the model, we can move to the training of the agent. Training of the agent is done using the SARSA (Rummery and Niranjan (1994)) algorithm, which can be described as follows:

**Algorithm 1:** SARSA

***initialize*** $Q(s,a)$ $\forall s \in S$, $a \in A$
**for** *each game* **do**
  $s$,$a \leftarrow$ initial state s $\in S$ and ***choose*** action $a(s)$
  **while** $s$ *is not terminal* **do**
    take action $a$, observe reward $R$ and state $s'$
    **if** $s'$ *is terminal* **then**
      ***update*** Q with $s$,$a$,$R$
      break
    **end**
    ***choose*** action $a'(s')$ ***update*** $Q$ with $s$,$a$,$R$,$s'$,$a'$
    $s \leftarrow$s'
    $a \leftarrow$a'
  **end**
**end**

Here ***choose***, ***initialize*** and ***update*** have to be further elaborated. We start with how we choose an action $a(s)$ in a given state $s$, which is done through the $\varepsilon$-greedy algorithm, which helps with the exploration versus exploitation dilemma and is given by:

**Algorithm 2:** $\varepsilon$-greedy

r $\leftarrow \max(\frac{\text{shuffles}*0.6-\text{current shuffle}}{\text{shuffles}*0.6},0)$
$\varepsilon \leftarrow 0.1 * r$
$u \sim U(0,1)$
**if** $u < \varepsilon$ **then**
  $a$ is chosen uniformly random
**else**
  $a \leftarrow \arg\max Q(s,a)$
**end**

We let $\varepsilon$ decay (as suggested in Koppula (2020)) for the first 60% of the shuffles, after which we set it to 0

We let the player train until the deck is shuffled 1,000 times, unless specified otherwise, which corresponds to about 50,000 games (or episodes). Next we look at 2 different ways of doing initialization and updating of the $Q(s, a)$ function, *Table Lookup* and *Neural Network*.

**Table Lookup** As the dimensionality of the state-action space is not too large, it is possible to create a full table containing the value of $Q(s, a)$ for every state $s$ and action $a$. This table is initialised to follow $U(-0.1, 0.1)$, for a terminal state the update is done by

$$Q(s, a) \leftarrow Q(s, a) + \alpha(R - Q(s, a)),$$

and for a non terminal state it is done by

$$Q(s, a) \leftarrow Q(s, a) + \alpha(R + Q(s', a') - Q(s, a)),$$

where $\alpha = 0.05$ is used.

**Neural Network** Instead of creating a full lookup table, a standard fully connected Feed Forward Neural Network (FFNN) can be used to represent the function $Q(s, a)$. For this purpose TensorFlow (2015) in Python is used, which allows the creation and training of the neural network. There are multiple different ways to create the neural network, as an estimator, as a classifier, an optimizer through usage of different activation functions, which could all result in a better model, which could be found through multiple runs in a validating format. In an optimal world, the desired plan would be to experiment with different setups (layers, layer size, activation functions, dropout) and architectures of neural networks (FFNN, CNN, RNN) and train all of them through multiple iterations of training and validation to choose the best one.

Unfortunately, due to lack of computational resources within this project, we could only conduct experiments on limited amount of training shuffles with multiple neural network setups. The usage of varying activations, such as the sigmoid or tanh, or amount of layers or layer sizes did not indicate greater results. Varying the learning rate would lead to divergence of the model if it was set higher, or slow to no learning if it was set lower. After these initial experiments, only one version of Neural Network is further investigated and presented within this project.

The version investigated is an estimator Neural Network structure. The architecture consists of, an input layer, 2 hidden layers consisting of 60 nodes with a bias node, connected by a ReLu (Glorot, Bordes, and Bengio (2011)) non-linear activation function. The output layer uses a linear activation with the LASSO (Tibshirani (1996)) regularizer with parameter $\lambda = 0.01$ in order to predict the value of $Q(s, a)$. The chosen loss function is MSE, which is optimized using ADAM (Kingma and Ba (2014)), with learning rate $\alpha = 0.00005$.[3] This network architecture is visualized in Figure 1.

Initialization is done by feeding a few arbitrary features with arbitrary label. The update is done by feeding the network the feature $(s, a)$ with label $R$ for a terminal state, and label $R + Q(s', a')$ for a non terminal state.

[3]As mentioned before, the model is a simple model as the optimisation of the neural network architecture is deemed out of the scope of the project due to low computational resources

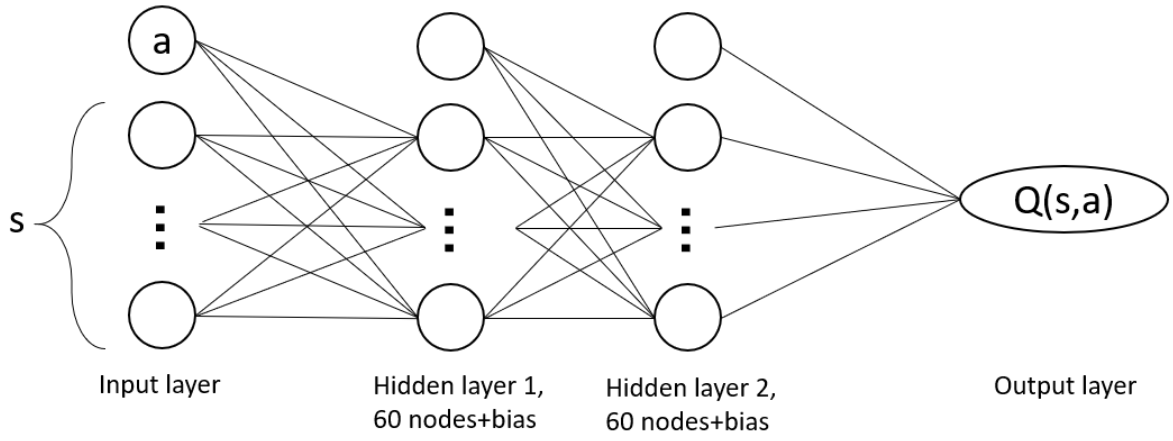


Figure 1: Neural network architecture

## 3.4 Performance measurement

As the game of blackjack involves playing and winning/losing money, the most natural measure for performance is how much money the strategy makes. After every game $i$ a reward $R_i$ is obtained, which corresponds to the money won in a real game of blackjack where the bet size is fixed at 1. During the training, we obtain a set of these rewards $R_{training}$, however this does not fully reflect the performance of the models playing optimally. Therefore, having trained the model, the agent is allowed to play for 100 extra validation shuffles[4] for which we obtain another set of rewards $R_{performance}$, the average of which is defined to be:

$$\bar{r}_{performance} = \sum_{i=1}^{n} R_{i,performance} \tag{1}$$

where $R_{i,performance}$ is the reward from game $i$ in the performance set, and $n$ is the amount of validation games played. This is what is used to asses the performance.

Convergence of the model is checked by allowing model to train on 100 additional shuffles and then doing another 100 validation shuffles, which gives another set of rewards $R_{convergence}$. This was deemed an appropriate validation environment, as over our experiments it appeared to provide consistent enough results through the large quantity of games it involves. This, combined with the rewards from the previous paragraph, gives 2 sets of rewards, $R_{performance}$ and $R_{convergence}$, which if the model has converged, should be very close as the effect of training for a converged model should be negligible. This leads to checking whether

$$|\bar{r}_{performance} - \bar{r}_{convergence}| < \eta,$$

where $\bar{r}$ is the mean of the sets $R$, as defined in Equation 1, and $\eta = 0.02$.

### 3.4.1 Validity of Performance measurement

Unfortunately, due to the stochasticity of validating a Reinforcement Learning agent on blackjack, it is very hard to guarantee results and whether it would perform in the same

[4]Without training during validation shuffles

effectiveness in the next experiment, due to the large effect randomness in both training and validation runs.

One way to make this effect lesser is to have more games as validation of the agent or repeat the whole training and validation experiment and average the results. 100 validation shuffles was deemed the appropriate balance between efficiency and effectiveness, however we were still restricted to choose $\eta$ quite generously.

# 4 Results

Having implemented the models as described in the previous section, their results are now discussed. Figure 2 shows the rewards obtained by the different models during the training phase, i.e. the vector $R_{training}$ as described in Section 3.4. Here one can see the typical convex shape for reinforcement learning, where as the models become better trained, the rewards loss decreases.

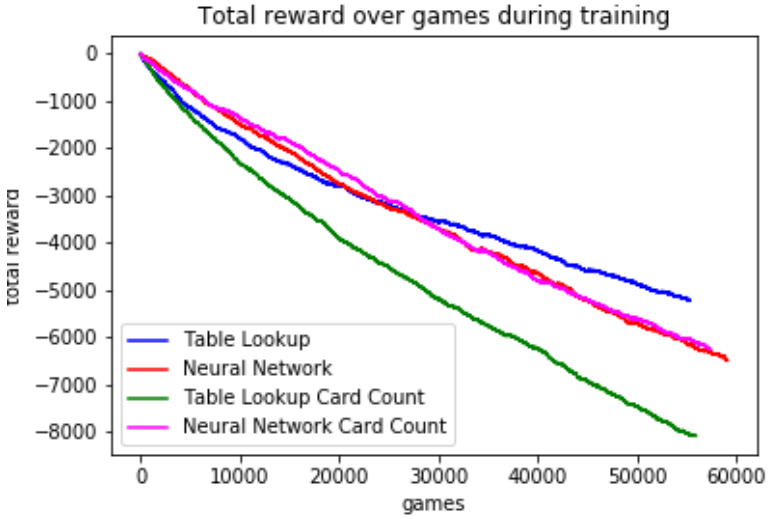


Figure 2: Reward over games using different models during training

With the trained models, we obtain the set $R_{performance}$ and its average $\bar{r}_{performance}$, as also described in Section 3.4, shown in Table 1 and plotted in Figure 3. All models converged, in the sense described in Section 3.4. Some interesting observations can be made about these results, but first we should preface that due to the stochastic nature of the game, the results are somewhat volatile. For our experiment, the best performing model turned out to be the neural network that does card counting. However, even following the best model, the house still has a marginal edge.

If card counting was not allowed, then the table lookup model performed best, still slightly outperforming the simple benchmark of mimicking the dealer. Since strategies without card counting are very simple, it is possible to show the entire strategy found by the table lookup model, which is shown in Figure 4 (since the table lookup model is computationally fast, it was trained on 100,000 shuffles to obtain these results). From this figure, some intuitive ideas, e.g. that it is safer for the player to hit if he has a useful ace, are confirmed.

A curios result is that the table lookup model performs better when it is not counting cards. From a theoretical perspective, this should not be possible, as no card counting is a restriction, and the table lookup model with card counting can always learn the same strategy as the model without card counting, hence its results should be at least as good. However, from a practical viewpoint, it turns out that with this amount of training, the card counting variable does more damage as noise, than it does good. If the model is trained a lot longer (which is possible again due to the low computation time), it will eventually identify a better strategy, as shown in Table 2, where it starts to match, and even improve on, the performance of the neural network with card counting.

| Model | Random | Mimic Dealer | Table Lookup | Neural Network | Table Lookup Card Count | Neural Network Card Count |
|---|---|---|---|---|---|---|
| Average reward | -0.3609 | -0.0787 | -0.0738 | -0.0938 | -0.1189 | -0.0594 |

Table 1: Average reward over 100 validation shuffles for the trained models using different models, all models converged (in the sense described in Section 3.4)

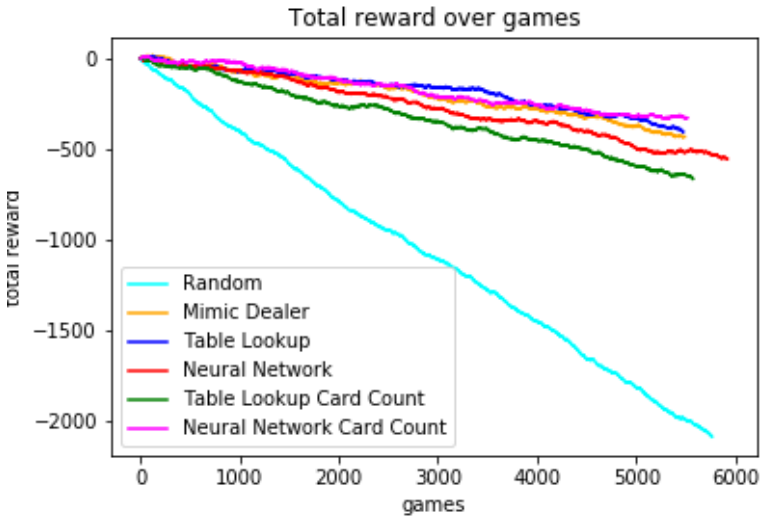


Figure 3: Reward over games for 100 validation shuffles using different models

| | | hand value | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 |
| dealer showing | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| | 2 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| | 3 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | 4 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| | 5 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | 6 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | 7 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| | 8 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| | 9 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| | 10 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |

| | | hand value | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 |
| dealer showing | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

Figure 4: Action table for Table Lookup with 100,000 shuffles, where 0 means hit and 1 stick, without useful ace (left) and with useful ace (right)

| Training Shuffles | 1,000 | 10,000 | 100,000 |
|---|---|---|---|
| Average reward | -0.1189 | -0.0574 | -0.0528 |

Table 2: Average reward over 100 shuffles using table lookup with card counting trained on more shuffles

# 5 Conclusion

In this report the power of reinforcement learning techniques for the game of blackjack is investigated. Though only limited time and computational resources were available, it has shown the potential of table lookup as well as neural network function approximations to beat out simple heuristic strategies. Specifically, combining the neural network with a simple card counting scheme performed best. However, the house edge for the casino remained, with our best strategy losing on average approximately 6 cents per 1 dollar bet game.

Though these results are hopeful, they should be taken with a grain of salt. With the blackbox nature of neural networks, it is hard to conclude the models have converged in a meaningful way, as our method for assessing convergence is quite rough. Further research could be done on the stability of strategies generated through the neural network.

There are 2 major directions in which these results could be improved. First, there is still a lot of room for improvement in the setup of the models, such as more computation resources for training, different choices for and optimization of the chosen hyper parameters or a different network architecture all together, to name a few. Second, different versions of the game of blackjack, for example with a bonus for having a blackjack, or with additional actions such as doubling or card splitting, which have a theoretically lower house edge, could be considered. Also, the perspective of the game can be switched, and optimal (i.e. harder to exploit) strategies for the dealer can be explored.

Overall, this report has shown the potential reinforcement learning has, and that it should definitely be one of the top considerations for tools to investigate different games.

# References


Baldwin, R. R., Cantey, W. E., Maisel, H., & McDermott, J. P. (1956). The optimum strategy in blackjack. *Journal of the American Statistical Association*, *51*(275), 429–439.

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier neural networks. In (Vol. 15, pp. 315–323).

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization.

Koppula, R. (2020). exploration vs. exploitation in reinforcement learning.

Rummery, G. A., & Niranjan, M. (1994). *On-line q-learning using connectionist systems* (Vol. 37). University of Cambridge, Department of Engineering Cambridge, UK.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., ... others (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, *362*(6419), 1140–1144.

Statista. (2020). Gross revenue from black jack in macao from 2009 to 2019.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. Cambridge, MA, USA: A Bradford Book.

TensorFlow. (2015). Tensorflow: Large-scale machine learning on heterogeneous systems, version 1.13.1.
(Available at `tensorflow.org`)

Thorp, E. O. (1966). *Beat the dealer: a winning strategy for the game of twenty one* (Vol. 310). Vintage.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, *58*(1), 267–288.