The University of Hong Kong

Faculty of Engineering

Department of Computer Science

COMP7704

To Play and Cooperate in Imperfect Information Game with Reinforcement Learning

Submitted in partial fulfillment of the requirements for the admission to the degree of Master of Science in Computer Science

By
NG, Argens
UID.: 3035072143

Under the supervision of: **Dr. Dirk Schnieders**
Date of submission: 30/04/2019

Abstract

This dissertation investigates the possibility of artificial agent learning to make decisions and cooperate in Contract Bridge, a multiplayer cooperative incomplete information game. In the following chapters, we will cover the theoretic grounds of our approach, conduct an analysis of the problem, introduce our approach and discuss our experimental results. While the results are inconclusive, we hope that our experience could contribute to parties interested in similar problems in the future.

Declaration

I, Ng Argens, declare that the work presented in this dissertation is my own. I confirm that where information has been derived from other sources, it has been clearly indicated.

Acknowledgement

I would like to express my greatest gratitude towards **Dr. Dirk Schnieders**, my mentor, who has given me immeasurable amount of liberty and support throughout my project.

I would also like to thank the researchers at **Deep Mind**, whose pushing of the boundary of AI has lit up my interest in the field.

I am also very grateful for the work done by **Bo Haglund** and **Soren Hein**, authors of the double dummy solver, and **Foppe Hemminga**, author of its Python API. The two programs were used extensively throughout the experiment, saving tremendous effort and time.

Lastly, I am very thankful to the countless researchers whose findings formed the foundation of machine learning.

Table of Contents

1. Introduction 1

1.1 Contract Bridge 2

1.1.1 Contract 3

1.1.2 Bidding 4

1.1.3 Card-Play 6

1.1.4 Objective 7

1.1.5 Par Value 8

1.2 Project Scope 10

1.2.1 Motivation 10

1.2.2 Level of Autonomy 12

2. Theoretical Principles 13

2.1 Problem Abstraction 13

2.1.1 Extensive Form Game 13

2.1.2 Markov Decision Process 14

2.1.3 Best Policy 16

2.1.4 Nash Equilibrium 18

2.2 Reinforcement Learning Concepts 21

2.2.1 Exploration and Exploitation 21

2.2.2 Delayed Reward 22

2.2.3 Curriculum Learning 23

2.3 Machine Learning Algorithm 25

2.3.1 Deep Q-Learning 26

2.3.2 Monte Carlo Reinforcement Learning 28

2.3.3 Counterfactual Regret Minimization 32

3. Analysis and Algorithm 39

3.1 Analysis 39

3.1.1 Extensive Form Game 39

3.1.2 Markov Decision Process 40

3.2 Algorithm 41

3.2.1 Random Walk 41
3.2.2 First Approach (Regret Minimization) 43
3.2.3 Second Approach (MCRL) 47

4. Results 50

5. Architectural Design 53

5.1 Information Representation 53
5.2 System Architecture 56
5.2.1 Deal Generator 57
5.2.2 Knowledge Base 57
5.2.3 Game Master 59
5.2.4 Agent 60

6. Discussion 61

6.1 Difficulties 61
6.1.1 State Space 61
6.1.2 Correlation 62
6.2 Evaluation 66
6.2.1 Correct Move 66
6.2.2 Data Efficiency 67
6.2.3 Resources 67
6.3 Reward 69
6.3.1 Cooperation 69
6.3.2 Communication 69
6.4 Suggestion 72

7. Conclusion 75

References 76
Appendices 78

1. Introduction

Machine learning has been one of the hottest topics in recent years as agents using machine learning were able to generalize ideas and find patterns to solve problems that were once considered difficult. This automates and expedites the development in many fields, such as face [1] and gait recognition [2], video indexing [3], and computer gaming [4,5,6] in various games. This inspired me to investigate the possibility that an artificial agent could bring insight into the stagnant development in the game of Bridge. In particular, it was hoped that a bidding system, akin to a language in the context of Bridge, could be developed completely with artificial agents. In the following three sections, we will first cover the basic rules and elements in the game of Contract Bridge. We will then talk about the scope and goal of our project, as well as our motivation.

1.1. Contract Bridge

Contract Bridge, in essence, is a cooperative zero-sum incomplete information trick-taking game. The game is cooperative because players never play by themselves and are at least paired and sometimes teamed; It is classified as zero-sum because whenever a pair of players gain score, it is equivalent to the opposing pair losing by the same amount; It is classified as an incomplete information game because players only look at 13 cards in half the game and 26 at the beginning of the second half, although the game is played with the full deck of 52 cards (excluding Jokers). To obtain information about the other cards, players can only gather the information over the course of the game, when each player play a card in each round; Lastly, it is a trick taking game, where the player who plays the largest card in each round wins a trick for the round.

Each pair attempts to win the most tricks in the game, after which they receive scores correlated to the number of tricks taken. However, the correlation between scores and number of taken tricks is not static but instead determined by contract, which leads to the name and the two clear parts of the game. To understand such a relationship, we need to first understand the structure of a contract.

1.1.1 Contract

A *contract* consists of a level (an integer ranging from one to seven) and a trump suit (or the absence of it). The level indicates the number of additional tricks the contract-bearing side, or declaring side promises to take beyond the minimum number that is six. Hence for example, a level of three means that the declaring pair promised to take nine or more tricks, only then would they receive positive scores. Whereas a trump suit marks the suit that will trump the otherwise equal suits in the next part of the game, card-play. A usual notation of a contract is simply a concatenation of the level, in integer, and the first letter of the trump suit. For example: *1S*, *2H* and *3N* (with *3NT* being equally common) are three possible contracts, where *S* stands for *Spades*, *H* stands for *Hearts* and *N* stands for *No-Trump*.

If the trump suit is *Hearts* or *Spades* and the level is higher than or equal to four, the declaring pair would earn a game bonus if the number of tricks required is reached. The act of taking the required number of tricks is known as *making the contract*. Similar game bonuses are granted to contracts made at level three or higher without a trump suit, or five or higher with *Clubs* or *Diamonds* being trump.

If the contract is at six-level and made, the declaring pair earns an additional small slam bonus; And if the contract is at seven-level and made, they earn a grand slam bonus instead. All these bonuses are higher if the pair is at vulnerable position, occurring twice every four deals. The rate of occurrence

is strictly followed at team-based tournaments and less at pair-based tournaments.

On the other hand, if the declaring side fails to take the required number of tricks, the contract is not made and the defenders have *downed* the contract. The declaring side then take a penalty proportional to the number of tricks short. The penalty is again higher at vulnerable position and less so otherwise. For most contracts, the scale of the negative scores attainable is somewhat similar to that of positive scores, therefore the players usually could not take too much risk when competing for contracts. This is especially true at vulnerable position (due to the possibly increased penalty) but less so for game contracts (due to the possibly increased reward).

A contract may also be *doubled* and *redoubled* to vastly increase the reward and penalty of making and downing a contract respectively. *Double* has to be bid by the side not bearing the contract, whereas *redouble* by the side bearing the contract, after a *double* is suggested. Doubled and undoubled contracts are indicated by one and two lower case crosses following a contract respectively, for example *2Sx* and *3Hxx*. This concludes all the elements of a contract.

1.1.2 Bidding

The bidding phase of the game determines the declaring side and the contract. Bidding starts with the dealer putting down a bid, followed by other players

in a clockwise direction, with each bid being either a *contract*, a *pass*, a *double* or a *redouble*. This process continues until there are four consecutive passes without a contract or three consecutive passes otherwise. The last contract bid would then be the final contract, with *double* and *redouble* taking effects appropriately. Players take turns being the dealer, hence the dealer position repeats every four deals.

If a player chooses to bid a contract, such contract must be either higher in level or same in level and higher in trump suit than the previous contract. The trump suits in ascending order are *Clubs*, *Diamonds*, *Hearts*, *Spades* and *No-Trump*. For example, if the current contract bid is *1S*, then the only one-level bid that is legal would be *1NT*, with all the other bids at two-level or higher open for the player in turn.

If a player otherwise chooses to bid *pass*, *double* or *redouble*, this does not consume any bidding space. The next player who chooses to bid a contract, however, must follow the above rule governing the bidding of contracts.

To determine a successful contract, the information of both hands of the paired players should be taken into account. However, in the bidding phase, each player is only allowed to look as his/her own hand. Hence players usually bid by following a system, with each bid showing additional information besides showing intent of playing such contract. This information flow must not be secretive and players are obliged to explain fully the meaning behind each bid and partnership agreements when asked by

opponents at any time. They are also obliged to alert their opponents when a player bids with no intent of playing such contract, such bids are labelled *artificial* bids (in contrast with the *natural* bids).

1.1.3 Card-Play

If four passes are directly bid, there will be no card-play and both sides receive zero for the deal. Otherwise there must be a contract, be it doubled, undoubled, or redoubled. The pair who first bid the final contract is the declaring pair, and the player in the declaring pair who first bid the suit of the final contract is the *declarer*, whereas the declarer's partner is called *dummy*. The dummy has no right to play cards and such right is given to the declarer, who shall play two hands.

The phase of card-play starts by the player to the left of *declarer* playing a card freely on the table, followed by *dummy* revealing his/her hand. Card-play then commence by each hand playing a card in the clockwise direction, with the requirement that from each hand a card must be played with the same suit as the first card if possible. If it is not possible, any card could be played but will usually be deemed to lose, since it fails to *follow suit*. The exception to such a case is a card from the trump suit, which competes only with other trump cards and wins any card from other suits. The hand that plays the highest card in each round is the winning hand, and can freely play a card in the next turn.

The highest card in each suit is *Ace*, followed by *King*, *Queen*, *Jack* and so on. Again, there is no ranking among the suits, with the exception of trump suit.

1.1.4 Objective

The objective of the game is always to get as much score as possible. This can be broken down into the following sub-objectives, in descending order of importance, from the perspective of the declaring pair.

1) *Reach the required number of tricks decided by the contract*
2) *Reach a contract higher than or equal to game or slam level if possible*
3) *Get as many tricks as possible*

The objective of the defending pair is then to prevent the declaring pair from achieving above goals. They are, with equal priority,

1) *Get as many tricks as possible*
2) *Obstruct opponent's bidding to prevent a possible game or slam contract from being reached*

The first objective is self-explanatory, while the second one requires more game knowledge to understand. Since the basic rule of bidding enforces that no contract bid could be at the same level as the previous contract and of a

lower suit, nor could it be at a lower level as the previous contract, by bidding at a higher level, we would rob the opponents of precious bidding space to communicate. If a player has enough information to deduce that his/her side would not need the bidding space themselves, it is likely that bidding at a higher level is beneficial to the pair.

1.1.5 Par Value

Par value is a special term rarely seen in other games, except golf. For each deal, by analysing the open hands, we are able to calculate the maximum number of tricks each player can take as the declarer (with complete information and perfect play) with either *Clubs*, *Diamonds*, *Hearts* and *Spades* as trump suit or under the absence of trump suit. By looking at the highest number of tricks possibly won, we can then see which pair of players ought to win the contract. We next see if the opposite pair has a winning sacrificial bid. For example, if *4S* by NS pair would be *just made* (winning 10 tricks), gaining them 620 marks, then *5H* by EW pair might be a winning sacrificial bid even if it is impossible to be made, since *5Hx* -1 (winning one trick short) wins the NS pair 200 marks at vulnerable, *5Hx* -2 wins them 500, and less so at non-vulnerable positions.

Assuming vulnerable position of the EW pair in the above example, and that both East and West could only win a maximum of 9 tricks with *Hearts* as trump under perfect card-play and complete information, then the par value of this deal would be NS +500, and the best contract would be EW *5Hx*. Since

once the contract has reached EW *5Hx*, any player making a move would be losing in the maximum score possibly gained (or minimum score possibly lost).

1.2 Project Scope

We will now talk about the scope of our project. While we would definitely like to develop an agent capable of handling everything from bidding to card-play to defence, this is not possible given our limited time and resources. Therefore, we will limit the scope to be focusing at the bidding phase of the game. We hope that our agents could invent a bidding system and be accustomed to it, such that they could bid while holding concealed hands and be as close to the par value and best contract, or even beat the par, if possible.

1.2.1 Motivation

There are two reasons behind choosing bidding to be our main focus.

Firstly, modern day bridge agents are significantly higher in level in terms of card-play than in bidding. This is because ultimately in card-play, humans think like computers by analysing the probability of each distribution and attempt to combine for a *play-line* that would have the highest expected value, or chance of success. This way of thinking is naturally more suitable for computer agents and hence the level of competency of the agents in such area is high. We believe this is mainly due to the abundance of information in card-play.

For example, when in a round of card-play, an opponent failed to *follow suit*, a train of thought of a bridge player might be:

1) *He/she has run out of that suit*
2) *Since he/she choose not to play a trump card, he/she might have run out of trump suit as well*
3) *This leaves the other concealed hand to be full of the two suits, leaving less space for the other two suits*

This can go on for an indefinite number of steps and as many could observe, this is a rational and sequential thinking process, where we try to digest data and deduce information, a field where computers clearly outperform humans. This is why computers perform exceptionally well, not to mention logical and rational, when the dummy's cards are revealed and card-play commences.

However, when the cards are hidden and computers try to do the bidding, they are usually rigid and uncreative, using little deduction on the information provided from unused bids and intention of players from chosen bids. For one, the simulation without considering unchosen bids might be computationally expensive already; Secondly, using hard-coded heuristics to cater all possible bidding scenarios, especially at high level, is close to impossible.

Besides the urge to attempt in the less accomplished field in computer Bridge, the second reason for choosing bidding as the area of focus is the hunger for insights in the refinement of systems in Bridge. The development of systems in Bridge has faced stagnation for years already, and people around the world mostly play around a handful of systems. Having played the game for seven

years and learnt of the existence of various systems, it becomes natural for one to wonder whether there is a system that trumps the others. Machine learning has proven itself capable of creating superhuman artificial agents at games and discovering knowledge and pattern beyond the experts' imagination. It is thus one of my hidden agenda to gain breakthrough in bidding systems via bottom up artificial intelligence.

1.2.2 Level of Autonomy

We would be providing our agents with the highest level of autonomy such that we could expect results that is free of influence from existing expert knowledge. We would be providing it with the lowest level of abstraction as well, so that the agents could surprise us with obscure insights. This principle would be upheld throughout the project and we would prioritize decreasing the level of complexity in our problems before considering the option of providing external help to our agents.

2 Theoretical Principles

2.1 Problem Abstraction

The first step to any machine learning task is choosing a suitable abstraction for the problem. This enables us to transplant algorithms that have been proven successful to such models to our problem. Also, it allows us to view the problem from the perspective of an artificial agent, and less as a player of the game.

2.1.1 Extensive Form Game

Incomplete information games are usually represented as a finite game in the extensive form due to the flexibility provided. Formally, with reference to Hart (1992), it is defined around a game tree $\boldsymbol{\Gamma}$, with a unique initial node $\boldsymbol{v_0}$, set of all nodes $\boldsymbol{V}$, terminal nodes $\boldsymbol{T}$ and non-terminal nodes, or decision nodes $\boldsymbol{D}$, such that $\boldsymbol{D + T = V}$ [7]. We also define the set of all players to be $\boldsymbol{I = \{1 \,..\, n\}}$, containing a finite number of players. Player 0 is usually reserved for nature, to model chance events as *move by nature* [7].

Since some nodes cannot be completely differentiated from one another in an incomplete information game, information set $\boldsymbol{\mathcal{H}}$ is defined, which is a partition on $\boldsymbol{D}$, such that every player should have the same strategy $\boldsymbol{\pi_i(H)}$ given decision nodes in the same information set $\boldsymbol{H \in \mathcal{H}}$. We define set of all

actions to be $\mathcal{A}$, and the action function to be $\boldsymbol{A}$, which is defined on domain $\mathcal{H}$ [7].

Lastly, the reward (utility) function is denoted $\boldsymbol{u}$, which takes in a terminal node $\boldsymbol{t} \in \boldsymbol{T}$ and returns a reward vector which is defined for each player. Hence $\boldsymbol{u_i(t)}$, would be the reward player $\boldsymbol{i}$ receives when the game terminates at node $\boldsymbol{t}$. The combination of $\mathcal{H}$, $\mathcal{A}$, $\boldsymbol{\Gamma}$ and $\boldsymbol{u}$ defines the game in a macro perspective, but we can still add in some notations that would be useful in solving the problem [7].

We know that $\boldsymbol{\pi_i(H)}$ is the strategy profile adopted by player $\boldsymbol{i}$ when faced with information set $\boldsymbol{H}$. It is defined as a probability distribution of possible actions to accommodate mixed strategies, which means that faced with $\boldsymbol{H}$, actions can be chosen not deterministically but stochastically. The strategy profile adopted by player $\boldsymbol{i}$ is usually denoted $\boldsymbol{\pi_i}$, whereas the collection of strategies adopted by all players is denoted $\boldsymbol{\Sigma}$, and $\boldsymbol{\pi_{-i}}$ is the set of strategies adopted by all players except player $\boldsymbol{i}$ [7].

2.1.2 Markov Decision Process

A game in its extensive form defines the game in a top-down manner, whereas Markov Decision Process (MDP) looks at the game from the perspective of a decision agent and is more suitable in the context of reinforcement learning.

In MDP, the learner, or the decision maker, is called an agent. Everything beyond the agent and within the problem is called the environment. At every time step $\boldsymbol{t}$, the agent is presented with some representation of the environment's state $\boldsymbol{s_t}$ and selects an action $\boldsymbol{a}$ in response [8]. One time step later, the state changes to $\boldsymbol{s_{t+1}}$ and the agent receives a reward $\boldsymbol{r_{t+1}}$ [8]. The state transition and the reward given are partly due to the agent's choice of action, but the latter need not have full influence on either of it.

In a finite MDP, the state transition and reward probability is well defined since there are only a finite number of states, actions and rewards ($\boldsymbol{S, A, R}$). That is, we could define the probability of transitioning into state $\boldsymbol{s'}$ and receiving reward $\boldsymbol{r}$, given the previous state $\boldsymbol{s}$ and action $\boldsymbol{a}$ [8].

$$p(s', r, s, a) = pr\{S_t = s, R_t = r \mid S_{t-1} = s, A_{t-1} = a\} \tag{2.1}$$

The probability given by this four-argument function $\boldsymbol{p}$ defines a Markov Decision Process just as ($\boldsymbol{\mathcal{H}}$, $\boldsymbol{\mathcal{A}}$, $\boldsymbol{\Gamma}$, $\boldsymbol{u}$) defines a game in extensive format. From this function $\boldsymbol{p}$ we could define other useful probabilities. One of the most useful ones is the expected reward of taking action $\boldsymbol{a}$ at state $\boldsymbol{s}$ [8].

$$r(s, a) = \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a] = \sum_{r \epsilon R} r \sum_{s' \epsilon S} p(s', r, s, a) \tag{2.2}$$

And by calculating the maximum expected reward among all actions at state $\boldsymbol{s}$, we get a value that quantifies how well a state is, also known as the state value.

$$v(s) = \max_{a \epsilon A} r(s, a) \tag{2.3}$$

2.1.3 Best Policy

It is now useful to define the goal of our abstracted models. What do we wish to achieve after we define our problems into models? For extended form-games, our goal is to find an action function $\boldsymbol{\pi_i(H)} : \boldsymbol{\mathcal{H}} \rightarrow \mathbb{R} + ^{|\mathcal{A}(\boldsymbol{H})|}$ that maps all possible information sets into vectors of positive real numbers, such that the *expected cumulative reward* over the course of the game is highest for player $\boldsymbol{i}$. $\boldsymbol{\pi_i(H)^{(j)}}$ is hence the probability that action $\boldsymbol{j}$ is chosen at information set $\boldsymbol{H}$ for player $\boldsymbol{i}$ and an implicit constraint of the function is that $\boldsymbol{\sum_j \pi_i(H)^{(j)} = 1}$.

For Markov Decision Process, our goal is to similarly define a policy $\boldsymbol{\pi}$ that maps the states to probabilities of selecting each possible actions. The target of such a policy could also be the *expected cumulative reward*. However, since a MDP, unlike an extensive form game, can have no terminal nodes. It is hence useful to define $\boldsymbol{G_t}$ to be the *expected return*, which is the sum of discounted expected rewards in the future. The discount factor has the notation $\boldsymbol{\gamma}$ and expected reward has the notation $\boldsymbol{r}$. $\boldsymbol{G_t}$ is then given by the

following equation [8]. (Note that we dropped the notation information set, since states in the context of MDP are already defined as the observable portion of the environment, with non-distinguishability considered.)

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k \tag{2.4}$$

Discount factor $\boldsymbol{\gamma}$ has a range of [0, 1] and $\boldsymbol{T}$ could be ∞ given that $\boldsymbol{\gamma \neq 1}$. The benefit of such a definition becomes clear when we consider an infinite MDP which gives a reward of one in every time-step as the baseline, but certain actions could increase the rate of reward. If our goal is to maximize *expected cumulative reward*, then every policy would become a solution since the *expected cumulative reward* always tends to infinity. Using *expected return* however, those with a higher rate of gaining reward would be considered superior. However, it also adds an additional variable when designing models.

We now modify eq. (2.2) and eq. (2.3) with the definition of policy $\boldsymbol{\pi}$ and expected return $\boldsymbol{G}$ [8].

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{T} \gamma^k r_{t+\,k+1} \, | S_t = s \right] \tag{2.5}$$

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s,\ A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{T} \gamma^k r_{t+\,k+1}\,|S_t = s,\ A_t = a\right]$$

(2.6)

Where $\boldsymbol{v_\pi(s)}$ is the state-value function for policy $\boldsymbol{\pi}$ and gives the expected return of following policy $\boldsymbol{\pi}$ starting from state $\boldsymbol{s}$. Whereas $\boldsymbol{q_\pi(s,a)}$ is the action-value function of policy $\boldsymbol{\pi}$ and gives the expected return of taking action $\boldsymbol{a}$ at state $\boldsymbol{s}$ and following policy $\boldsymbol{\pi}$ thereafter [8].

Therefore solving a MDP is equivalent to finding a policy $\boldsymbol{\pi}$ such that $\boldsymbol{v_\pi(s_0)}$ is largest, where $\boldsymbol{s_0}$ is the first state (faced by our agent).

2.1.4 Nash Equilibrium

Up until now, we have defined the problem mostly from a single agent's perspective. We now introduce opponents by first defining $\boldsymbol{v_{\pi_i}(s,\pi_{-i})}$ as the expected return of player $\boldsymbol{i}$ under policy $\boldsymbol{\pi_i}$ at state $\boldsymbol{s}$ given policies of all players except player $\boldsymbol{i}$, denoted $\boldsymbol{\pi_{-i}}$. We also simplify the notations by introducing utility function $\boldsymbol{u_i(\pi_i,\pi_{-i})}$, which gives the expected return of player $\boldsymbol{i}$ at state $\boldsymbol{s_0}$ given player $\boldsymbol{i}$ follows $\boldsymbol{\pi_i}$ and other players follow $\boldsymbol{\pi_{-i}}$ [10].

$$v_{\pi_i}(s,\pi_{-i}) = \mathbb{E}_\pi[G_t|S_t = s,\pi_{-i}]$$

(2.7)

$$u_i(\pi_i,\pi_{-i}) = v_{\pi_i}(s_0,\pi_{-i})$$

(2.8)

Then we can define best response $\boldsymbol{BR_i(\pi_{-i})}$ to be the best policy player i could have, given the strategies followed by other players $\boldsymbol{\pi_{-i}}$

.

$$BR_i(\pi_{-i}) = \arg\max_{\pi_i} u_i(\pi_i, \pi_{-i})$$

(2.9)

We can also define $\mathcal{E}$-best response to be:

$$\mathcal{E}BR_i(\pi_{-i}) = \left\{\pi_i \in \Sigma_i : u_i(\pi_i, \pi_{-i}) \geq \max_{\pi_i} u_i(\pi_i, \pi_{-i}) - \mathcal{E}\right\}$$

(2.10)

Which is the set of policies player $\boldsymbol{i}$ could have that would bring an expected return no more than $\boldsymbol{\mathcal{E}}$ less than the best policy [10]. When every player is playing a best response, the game-state is known as Nash Equilibrium.

$$u_i(\pi_i, \pi_{-i}) \geq u_i(\pi_i', \pi_{-i}) \text{ , for all player } i$$

(2.11)

Similarly, when every player is playing a $\mathcal{E}$-best response, the game-state is known as the $\mathcal{E}$-Nash Equilibrium.

$$u_i(\pi_i, \pi_{-i}) \geq u_i(\pi_i', \pi_{-i}) - \mathcal{E} \text{ , for all player } i$$

(2.12)

Nash Equilibrium has the special property that at such a game-state, no rational agent would have the intention to deviate from their strategy. This gives rise to a stable solution otherwise computationally costly to find.

Without the Nash Equilibrium, solving a MDP would require us to provide an optimal strategy given every combination of possible strategies used by opponents. With the Nash Equilibrium, however, it would be sufficiently strong proof of an agent's success if it could reach Nash Equilibrium or $\mathcal{E}$-Nash Equilibrium with small enough $\mathcal{E}$ under self-play.

Nash Equilibrium also takes up the form of par value in the game of Bridge and provides an objective evaluation of the performance of each pair in each deal. Recall that par value is the optimal score that either pair could win (or lose) under perfect play, we could then evaluate the success of our agents by measuring the mean absolute difference or mean squared difference between par value and the results of simulated self-play.

2.2 Reinforcement Learning Concepts

2.2.1 Exploration and Exploitation

Exploration and exploitation is at the core of reinforcement learning. The struggle is based off a simple question – "How much should I experiment?"

Imagine a scenario where there are two buttons, which when pressed, have different effect on the score shown on the screen. This effect, however, is unknown to us. Assume that we start experimenting by pressing button one, and the score raises by 100, and we soon found out this is a repeatable action, what are we going to do?

Naturally, we would exploit this strategy at the beginning, but sooner or later, we would try button two, at least for once, to see if it raises our score by more than 100. We might even press it for a second time, if it was rewarding or not too punishing. Now, how rewarding or not punishing should it be before we went back to the promising first strategy? This explains the interaction between exploration and exploitation.

In everyday scenario, we might have a varying exploration rate by setting certain scenarios as merely test grounds. However, as an online training paradigm, reinforcement learning agents often faces the exact same problem a handful of times, or even once for problems with large state space and little abstraction. It is then a hard task for a reinforcement learning agent to pick

between exploring choices that were not chosen before and exploiting choices that are known to be good in this small window of opportunity. In Counterfactual Regret Minimization (CFR Minimization), the solution of such a problem was simply to override chance and generate the required training data according to the abstraction model.

Whereas in Monte Carlo Reinforcement Learning (MCRL), while the MCTS is supplemented with the transition probability vector from neural network to find the most useful subtree, this information is not the sole determinant of the next node, value feedback, noise and most importantly visit count are also given credit when selecting the next node to explore. This ensure that those with a low visit count has certain priority over those with a high visit count.

2.2.2 Delayed Reward

Delayed reward is another important concept in reinforcement learning. If agents are not able to take into account delayed reward, it would simply go for the greedy approach of receiving the highest reward for the next action, also known as being reward myopic. Reward in this context should not be confused with value, or have any association with the final outcome. Reward only means the immediate reward as an agent chooses an action and act on it, as in the Markov Decision Process model.

Here's an example to better illustrate the importance of delayed reward. Imagine an agent playing the arcade game Tetris, where eliminating each line

wins it 40 points but eliminating four lines at once wins it 1200 points, with the trade-off that setting up such clear requires four times the time on average. Any rational human players would naturally stack tiles and wait for the opportunity of clearing four lines at once, yet we have to convince an artificial agent that getting 40 points with move A is not as beneficial with getting 0 points with move B for three rounds. This is the mentality of delayed reward, which usually involves looking forward and doing simulations when artificial agents decide.

2.2.3 Curriculum Training

In the paper titled *Curriculum Training*, the authors discussed how machine learning should take reference of human learning, which is usually structured and organized in a way such that easier concepts and knowledge are learnt before the more complex ones [11]. In human learning, this way of learning allows the easier concepts to ease the learning of the harder ones; Whereas in machine learning, using such technique, coined curriculum learning, the authors were able to show faster convergence and a better local minimum in the case of non-convex problems.

To show a faster convergence, the authors, Bengio, Louradour, Collobert and Weston, used a perceptron to do the test. In the test, the perceptron was asked to classify each data into two classes, with each data only having certain features as relevant, the rests are just noises that could be turned on or off. Easier learning, in this context, would be the training data that have more

noises turned off, whereas harder ones would be the noisier ones. It was then shown that training in an ascending order of difficulty did end up with a faster convergence in total [11].

In the test to show that a better local minimum is achievable with curriculum training, the authors trained a deep neural network predictor that returns the next word given a sentence segment, and rank this returned word by its actual likelihood of appearing. They designed a curriculum for such network by first ignoring training examples with words that are not the 5k most common, followed by ignoring examples with words that are not the 10k most common. By doing so, while having a slower start, the network converges to a lower minimum than network without a curriculum after 1000 million updates [11]. These tests all showed the promising property of curriculum training in reinforcement learning.

2.3 Machine Learning Algorithms

We will now discuss several machine learning algorithms we considered during planning. Deep Q-Learning is the algorithm used by DeepMind in constructing the agent that plays Atari games with low-level sensory input; Monte Carlo Reinforcement Learning (MCRL) is the name we give to the general reinforcement learning algorithm DeepMind developed for Chess and Shogi through self-play; Counterfactual Regret Minimization (CFR Minimization) is the algorithm researchers in Carnegie Mellon University develop to play Texas Hold 'Em. Since all the algorithms are designed for other games, we will only focus on the parts that we could take reference of in the game of Bridge.

2.3.1 Deep Q-Learning

Deep Q-Learning is the main technique used by DeepMind when training an agent to play the Atari games using raw pixel inputs [6]. Deep Q-Learning is basically performing reinforcement learning on a network that predicts the Q-value, which is the optimal state-action reward defined over possible combinations of state $\boldsymbol{s}$ and action $\boldsymbol{a}$. It is similar to the optimal state-action value by substituting the optimal strategy $\boldsymbol{BR_i(\pi_{-i})}$ from eq. (2.9) into $\boldsymbol{q_\pi(s,a)}$ from eq. (2.6) in the absence of $\boldsymbol{\pi_{-i}}$, since there are no opponents in the Atari environment. The resulting $\boldsymbol{Q^*(s,a)}$ is defined as follow.

$$q_\pi(s,a) = \mathbb{E}_\pi\left[\sum_{k=0}^{T} \gamma^k r_{t+\,k+1}\,|S_t = s,\ A_t = a\right]$$

from (2.6)

$$Q^*(s,a) = \max_\pi \mathbb{E}_\pi[R_t|S_t = s,\ A_t = a, \pi]$$

(2.13)

Where $\boldsymbol{R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}}$. Then by the *Bellman equation* [12], we know that the optimal strategy at this state $\boldsymbol{s}$ is to find an action $\boldsymbol{a}$ that maximizes the expected value of $\boldsymbol{r + \gamma Q^*(s,a)}$. One way of doing so is to use iterative update [8], from which we can find the best action $\boldsymbol{a^*}$ as well as all the $\boldsymbol{Q^*(s,a)}$. This is however not very informative nor practical, since each action-state value function is estimated separately. The approach that DeepMind takes (as well as in the successful case of *TD-gammon*) is to use a neural network (or perceptron in the case of *TD-gammon*) to approximate

$\boldsymbol{Q}^{*}(\boldsymbol{s}, \boldsymbol{a})$. The function shall be denoted $\boldsymbol{Q}(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{\theta})$, where θ is the set of weights of the network.

The training process of the network is as follows [6]. The weight $\boldsymbol{\theta}$ of the network is first randomly initialized. To ensure exploration the agent randomly selects a move a with a probability $\boldsymbol{\epsilon}$, otherwise it greedily selects the action with the highest value of $\boldsymbol{Q}(\emptyset, \boldsymbol{a}; \boldsymbol{\theta})$, where $\emptyset_{\boldsymbol{t}}$ is the processed state of raw state $\boldsymbol{s_t}$. The transition $(\emptyset_{\boldsymbol{t}}, \boldsymbol{a_t}, \boldsymbol{r_t}, \emptyset_{\boldsymbol{t+1}})$ is then stored into experience pool $\boldsymbol{\mathcal{D}}$, after which a random transition is sampled and trained on. If $\emptyset_{\boldsymbol{t+1}}$ is terminal then $\boldsymbol{r_t}$ is the target value for our function $\boldsymbol{Q}(\emptyset_{\boldsymbol{t}}, \boldsymbol{a_t})$; Otherwise, the target value is $\boldsymbol{r_t} + \boldsymbol{\gamma} \max_{\boldsymbol{a'}} \boldsymbol{Q}(\emptyset_{\boldsymbol{t+1}}, \boldsymbol{a'})$, by the definition of Q-value. The network is then trained by off-policy stochastic gradient decent on the squared error between predicted value and target value [6].

Experience replay should be a large part in the algorithm since Q-Learning has been known to oscillate and even diverge with non-linear function approximators or even off-policy learning [13]. By utilising experience replay [14], the correlation between consecutive training samples is reduced and there is also less chances of creating feedback loop when, for example, the chosen action leads to next state $\boldsymbol{s'}$ identical to current state $\boldsymbol{s}$. This allows Deep Q-Network to reduce cost compared with neural fitted Q-learning [15], (NFQ) which uses batch update and has a computation cost proportional to the size of the data set.

DeepMind also claims that they used a model architecture where the output vector is all the $\boldsymbol{Q}(\emptyset_t, \boldsymbol{a_t})$ for all possible action $\boldsymbol{a_t}$ and the input vector is $\emptyset_t$. This allows them to update the values in one forward pass of the network and reduces cost. This is confusing since it should be impossible to sample all the transition reward $\boldsymbol{r_t}$ in one simulation of game and near-impossible to encounter the same $\emptyset_t$ over multiple episodes to collect enough data for all $\boldsymbol{a_t}$, especially since a constant memory size $\boldsymbol{N}$ is set for the replay memory. However, this may also be possible if the state space of Atari games is sufficiently small.

2.3.2 Monte Carlo Reinforcement Learning

As mentioned in the introduction to this section, Monte Carlo Reinforcement Learning (MCRL) in a name given by us, since DeepMind did not officially name the algorithm that they use. This algorithm is a general reinforcement learning algorithm that is suitable for (at least) zero-sum complete information games, in particular Chess, Go and Shogi as proven by AlphaZero [4], the program using this algorithm in their project.

MCRL is based off Monte Carlo Tree Search (MCTS), a heuristic search algorithm that explores a game tree in a manner somewhat similar to a random walk. At each iteration of MCTS, a game is played to the end by selecting moves (pseudo-)randomly, prioritizing moves that were never tried before. If all the moves, relative to current state, have been visited before, then the ones that have a higher average value would be visited with a higher probability.

After the search reaches the end of the game, the value at termination and visit count of each move at each node is updated. This allows the future iterations to focus on the more promising sub-trees and is one of the main reason MCTS has received great success in games with a high branching factor.

The performance of a MCTS largely depends on successfully finding the promising subtrees in a short amount of time [16], which is why DeepMind uses reinforcement learning network to supplement the search of the tree. DeepMind also uses the network to truncate the search depth of MCTS. Hence the input of the network is the features of the state and the output is a prediction of the probability of each move being chosen as well as value of the state. Such predicted values will be referred as policy vector $\boldsymbol{p}$ and expected value $\boldsymbol{v}$ hereafter.

At each episode of self-play, the network undergoes the process of MCTS, followed by choosing a move to play and repeats until the game terminates or if the expected value $\boldsymbol{v}$ is lower than a $\boldsymbol{v}_{\text{threshold}}$, where the losing network will then withdraw. The value of $\boldsymbol{v}_{\text{threshold}}$ is automatically selected to ensure that the false positive is less than 5% and false positive is measured by disabling withdrawal 10% of the time. The number of MCTS simulations in each of these iterations is 1500 for AlphaGo Zero and 800 for AlphaZero [4,5].

At each simulation of MCTS, a move is continuously selected by choosing the argument $\boldsymbol{a_t}$ that maximizes the sum of $\boldsymbol{Q(s_t, a)}$ and $\boldsymbol{U(s_t, a)}$,

where $\boldsymbol{Q(s_t, a)}$ is the average action value of previous searches and $\boldsymbol{U(s_t, a)}$ is a bias term based of policy vector $\boldsymbol{p}$. This continues until $(\boldsymbol{s_t, a_t})$ leads to an unexpanded leaf node [4,5].

$$a_t = \arg\max_{a_t} \left(Q(s_t, a) + U(s_t, a)\right) \tag{2.14}$$

$$Q(s_t, a) = \frac{1}{N(s_t, a)} \sum_{s'|s,a \to s'} V(s') \tag{2.15}$$

Where $\boldsymbol{s'}$ is the next state reached from state s following action $\boldsymbol{a}$; And $\boldsymbol{V(s)}$ is the expected value v given s, or the scalar output of the network [5].

$$U(s_t, a) = c_{puct} P(s_t, a) \frac{\sqrt{\sum_b N(s_t, b)}}{N(s_t, a)} \tag{2.16}$$

Where $\boldsymbol{c_{puct}}$ [17] is a constant determining the level of exploration, and $\boldsymbol{P(s_t, a)}$ is a modified probability of what is returned from policy vector p [5].

$$P(s_t, a) = (1 - \varepsilon) p_a + \varepsilon \eta_a \tag{2.17}$$

Where $\boldsymbol{p_a}$ is the probability of action $\boldsymbol{a}$ being chosen (predicted by the network); $\boldsymbol{\varepsilon}$ is the exploration constant set at 0.25; And $\boldsymbol{\eta} \sim$ Dir (0.3) is the Dirichlet distribution generating noise to ensure all moves have a probability of being chosen (but the effect can be overridden with a large enough $\boldsymbol{p_a}$) [5].

After an unexpanded leaf node is encountered, it is expanded and evaluated using the network. The statistics are then back-propagated, incrementing visit counts and updating $\boldsymbol{Q(s,a)}$ values. After reaching the count of simulation, a move is selected by $\boldsymbol{\pi(a|s_0)}$ ($\boldsymbol{s_0}$ since the current node is always the root) [5]. The subtree below the selected action is kept along with the statistics whereas the other subtrees are discarded.

$$\pi(a|s_0) = \frac{N(s_0,a)^{1/\Gamma}}{\sum_b N(s_0,b)^{1/\Gamma}} \tag{2.18}$$

Where $\boldsymbol{\Gamma}$ is the temperature parameter, again controlling the level of exploration. It is set to $\boldsymbol{\Gamma = 1}$ at the first 30 moves of each game, and infinitesimal otherwise. It is set to infinitesimal at the beginning of the game during evaluation of the network.

After each iteration of self-play the actual game outcome $\boldsymbol{z}$ is added to the search probabilities $\boldsymbol{\pi}$ and state $\boldsymbol{s}$ to update the training samples, which is a collection of the samples of the most recent 500,000 games [5]. The network is asynchronously updated using stochastic gradient decent of the loss function $\boldsymbol{l}$ to minimize mean squared error between predicted value v and actual game outcome z, as well as to maximize likelihood between search probability $\boldsymbol{\pi}$ and policy vector $\boldsymbol{p}$, with L2 regularization in mind [4,5].

$$l = (z - v)^2 - \pi^T \log p + c\|\theta\|^2$$

(2.19)

Where $\boldsymbol{c}$ is the L2 regularizing constant and $\boldsymbol{\theta}$ is the set of parameters of the network.

The main principle behind MCRL is policy iteration, a classic algorithm in reinforcement learning [5,8]. It alternates between policy evaluation and policy improvement to generate a sequence of improving policies. In the case of MCRL, policy evaluation is conducted by MCTS and self-play, whereas policy improvement is conducted on the network primarily, then projected onto MCTS due to its extensive use of the network.

One main architectural takeaway from the MCRL is the use of various techniques to ensure exploration, including exploration constant, noise and ratio of root total count over action visit count [5]. It is also impressive that DeepMind uses virtual loss [9] to prevent duplicated traversal of game tree in the case of parallel computing.

2.3.3 Counterfactual Regret Minimization

Counterfactual Regret Minimization [10, 18] (CFR minimization, hereafter denoted as CFR) is a technique introduced by the creators of Libratus [18], a poker playing AI which won four human professional players by a total of $1.7 million (in chips) over the course of 20 days. The big blind in the match was set to $100 and thus the win rate was 14.7 big blinds per 100 hands. The

technique was based on regret aversion in decision theory and psychology and is especially suitable to play incomplete information games.

Suppose we were to make a decision on something we do not have complete information nor complete control on, such as investment. For example, we had predicted the stock price of a particular company to go down and we thus sold all our stocks, which was higher than what we had bought it for, making net gain all in all. When the stock price instead continued to go up, we would feel regret even when no loss was incurred. The emotion of regret is hence best defined as "I could be better off if …" A quantitative definition of regret is the utility (or expected return) difference between any two legal actions at one state, and regret minimization is the technique that mimics regret aversion to steer artificial agents towards optimal policy.

For example, imagine our agent is playing a rock paper scissors game [19], where the rock beats scissors, scissors beat paper and paper beats rock. In one particular game, it may have played paper, losing to an opponent playing scissors. This incurred a sense of regret of not playing the other moves. This regret can be quantified as:

$$R_i(a) = U(a) - U(a_i) \tag{2.20}$$

By defining the utility as one if player wins, zero if player draws and negative one if player loses, we get $R_i(\text{Rock}) = 2$, $R_i(\text{Scissors}) = 1$, $R_i(\text{Rock}) = 0$.

Then, by summing up all the positive regrets over a substantial period of games, we receive a strategy of playing each move by the probability:

$$P(a) = \frac{\sum_i^n R_i^+(a)}{\sum_b \sum_i^n R_i^+(b)} \tag{2.21}$$

Where the numerator is the sum of $\boldsymbol{R_i^+(a)}$, the positive regret of not playing action a in game i, across all games; And the denominator is the sum of such sums over all possible actions. Hence moves with higher past history regret have a higher chance of being played, which minimizes regret in the future and leads to the name regret minimization.

Regret minimization has the advantage of not having to expand a game tree online but instead learns after each play, which fits into the schema of reinforcement learning. It is also more suitable to playing imperfect information games, since expanding a game tree prior would be very costly. Despite its greedy nature, it has been proven to converge into the Nash equilibrium [10]. In the case of 2-player zero-sum game, given that both players have an average overall regret less than **ε**, regret minimization converges into a 2ε-Nash equilibrium, which means that both players could not receive more than 2ε extra utility if they change strategy [10,12].

The main disadvantage of such method, however, is the vast amount of storage space needed for the different states, which is why creators of Libratus introduced the idea of immediate counterfactual regret $\boldsymbol{R_{i,imm}^T(I)}$ and

counterfactual utility $\boldsymbol{u_i(\pi, I)}$ [10]. Before discussing these ideas, we need to define several basic terms used in CFR: A history $\boldsymbol{h}$ is a sequence of actions leading to the current state, $\boldsymbol{H}$ is the set of all histories and $\boldsymbol{Z}$ is the set of all terminal histories; $\mathbb{H}_{\boldsymbol{i}}$ is the information partition for player $\boldsymbol{i}$ and a set $\boldsymbol{I_i} \in \mathbb{H}_{\boldsymbol{i}}$ is an information set for player $\boldsymbol{i}$. For other terms, we shall use notations used earlier in our paper.

We also now define that $\boldsymbol{p^{\pi}(h)}$ as the probability of history $\boldsymbol{h}$ occurring given that all players follow strategy profile $\boldsymbol{\pi}$, $\boldsymbol{p_i^{\pi}(h)}$ be the probability component contributed by player $\boldsymbol{i}$ and $\boldsymbol{p_{-i}^{\pi}(h)}$ be the component contributed by all other players (including chance) [10]. Probability of information set *I* occurring when all players follow strategy profile $\boldsymbol{\pi}$ is $\boldsymbol{p^{\pi}(I)}$ and similarly we define $\boldsymbol{p_i^{\pi}(I)}$ and $\boldsymbol{p_{i\text{-}1}^{\pi}(I)}$ [10].

$$p^{\pi}(I) = \sum_{h \in I} p^{\pi}(h)$$

(2.22)

It is then easy to see that the utility of the strategy profile $\boldsymbol{\pi}$ to player $\boldsymbol{i}$ is given by $\boldsymbol{u_i(\pi)}$.

$$u_i(\pi) = \sum_{h' \in Z} p^{\pi}(h')\, u_i(h')$$

(2.23)

And the utility to player of the strategy profile $\boldsymbol{\pi}$ to player $\boldsymbol{i}$ given that history $\boldsymbol{h}$ is reached is $\boldsymbol{u_i(\pi, h)}$.

$$u_i(\pi, h) = \sum_{h' \in Z} \frac{p^\pi(h \cap h') u_i(h')}{p^\pi(h)} = \sum_{h' \in Z} p^\pi(h, h')\, u_i(h')$$

(2.24)

We can then define counterfactual utility, the utility given that information set $\boldsymbol{I}$ is reached to be $\boldsymbol{u_i(\pi, I)}$.

$$\begin{aligned} u_i(\pi, I) &= \sum_{h' \in Z} \frac{p^\pi(I \cap h') u_i(h')}{p^\pi(I)} \\ &= \sum_{h \in I, h' \in Z} \frac{p^\pi(h) p^\pi(h, h') u_i(h')}{p^\pi(I)} \\ &= \sum_{h \in I, h' \in Z} \frac{p_i^\pi(h) p_{-i}^\pi(h) p^\pi(h, h') u_i(h')}{p_i^\pi(I) p_{-i}^\pi(I)} \\ &= \frac{\sum_{h \in I, h' \in Z} p_{-i}^\pi(h) p^\pi(h, h') u_i(h')}{p_{-i}^\pi(I)} \end{aligned}$$

(2.23)

The terms $\boldsymbol{p_i^\pi(h)}$ and $\boldsymbol{p_i^\pi(I)}$ are cancelled out since they are essentially identical and are both the product of move probabilities by player $\boldsymbol{i}$ specified by $\boldsymbol{\pi}$ for all the moves chosen. The authors then define counterfactual regret [10] $\boldsymbol{R_i^T(I, a)}$ as the average regret at information set $\boldsymbol{I}$ for not choosing action $\boldsymbol{a}$ over $\boldsymbol{T}$ rounds of the game, with a special weighing term for cancelling out the denominator at eq. (2.23).

$$R_i^T(I, a) = \frac{1}{T} \sum_{t=1}^{T} p_{-i}^{\pi^t}(I)(u_i(\pi^t|_{I \to a}, I) - u_i(\pi^t, I))$$

(2.24)

Where $\boldsymbol{\pi^t|_{I\to a}}$ is defined as a strategy identical to $\boldsymbol{\pi^t}$ except that action $\boldsymbol{a}$ is always chosen at information set $\boldsymbol{I}$; $\boldsymbol{\pi^t}$ is defined as the strategy $\boldsymbol{\pi}$ at round $\boldsymbol{t}$, accommodating a changing strategy between each round. Note that the weighing term $\boldsymbol{p_{-i}^{\pi^t}(I)}$ was explained by the authors as the "counterfactual probability that $\boldsymbol{I}$ would be reached if *the player had tried to do so*" [10], which fits mathematically since the $\boldsymbol{p_i^{\pi^t}(I)}$ would then become 1 under such scenario, but the purpose of such weighing term is still unclear. We hence assume the real purpose of such term to be cancelling out the hard to obtain $\boldsymbol{p_{-i}^{\pi^t}(I)}$ term in counterfactual utility.

The researchers also defined a similar term for not choosing the optimal action at information set $\boldsymbol{I}$, independent on action space $\boldsymbol{A}$. It was proved that minimizing such term leads to minimizing overall regret. Therefore, by repeated self-play and updating strategies defined by eq. (2.25), a strategy that approaches Nash Equilibrium is obtained [10].

$$P(a) = \frac{\sum_i^n R_i^+(a)}{\sum_b \sum_i^n R_i^+(b)}$$

From (2.21)

Substituting the regret term by the one defined in Eq. (2.24), we get:

$$\pi^{T+1}(I,a) = \frac{R_i^{T,+}(I,a)}{\sum_b R_i^{T,+}(I,b)}, if \sum_b R_i^{T,+}(I,b) \neq 0$$

(2.25)

If the denominator term is equal to zero, then $\boldsymbol{\pi^{T+1}(I, a)}$ chooses a move randomly.

Similar to MCRL, the approach CFR taken can be seen as a variation of policy iteration [8]. Policy evaluation is the process of self-play and gathering regrets, and policy improvement is the direct update to the policy through dynamic programming. Besides the algorithm, an architectural takeaway from the project is that the researchers generate representational sets instead of raw states during training and self-play, to ensure a more meaningful distribution of states encountered [10]. This is done by fixing a sample hand for each abstracted state. The results proved that this has little drawback.

3 Analysis and Algorithm

In this chapter, we will analyze our problem using theories discussed above. We will also talk about the two algorithms we have designed after taking reference of the ones from literature.

3.1 Analysis

3.1.1 Extensive Form Game

We shall now define the bidding phase of the game of Bridge using extensive form game. The game can been seen as having game tree $\boldsymbol{\Gamma}$ with initial node $\boldsymbol{v_0}$, which is empty. The set of all nodes $\boldsymbol{V}$ is defined as the Cartesian product of the set of all bidding sequences $\boldsymbol{S}$ and the set of all possible distributions of hands $\boldsymbol{C}$ togther with $\boldsymbol{v_0}$, i.e. $\boldsymbol{V = C \times S + v_0}$; The set of terminal nodes $\boldsymbol{T}$ is the subset of $\boldsymbol{V}$ with bidding sequence component $\boldsymbol{seq} \in \boldsymbol{S}$ having "Pass, Pass, Pass" as proper suffix; We can also define $\boldsymbol{V_1}$ as the subset of V where all nodes $\boldsymbol{v} \in \boldsymbol{V}$ has an empty bidding sequence component. This subset consists of all nodes $\boldsymbol{v}$ where the cards are dealt but no one has bided.

The first move of game tree $\boldsymbol{\Gamma}$ is always by player 0 (nature), which follows a fixed strategy $\boldsymbol{\pi_0}$ and leads to a node in $\boldsymbol{V_1}$. After which player plays based on strategy $\boldsymbol{\pi_i(H)}$, where $\boldsymbol{H}$ has the feature that for any arbitrary information set $\boldsymbol{H}$, all nodes $\boldsymbol{v}$ in $\boldsymbol{H_i}$ has identical hand $\boldsymbol{h}$ and bidding sequence $\boldsymbol{seq}$.

Lastly, the utility function $\boldsymbol{u}$ is defined in space $\boldsymbol{C} \times \boldsymbol{T}$. It takes in terminal node $\boldsymbol{t} \in \boldsymbol{T}$ and hand distribution $\boldsymbol{h} \in \boldsymbol{C}$ and returns a 4-vector $\boldsymbol{s}$ with the property that $\boldsymbol{s_1} = \text{-}\boldsymbol{s_2} = \boldsymbol{s_3} = \text{-}\boldsymbol{s_4}$.

3.1.2 Markov Decision Process

We can also view the bidding of Bridge as a MDP from the perspective of one player. Initial state $\boldsymbol{s_0}$ consists of a set of cards and a bidding sequence with length less than 4. At every time step $\boldsymbol{t}$, an action is chosen which advances $\boldsymbol{s_t}$ to $\boldsymbol{s_{t+1}}$. The state $\boldsymbol{s_{t+1}}$ will always differ from $\boldsymbol{s_t}$ by having a bidding sequence $\boldsymbol{seq_{t+1}}$ that has $\boldsymbol{seq_t}$ as proper prefix and $|\boldsymbol{seq_{t+1}}| = |\boldsymbol{seq_{t+1}}| + \boldsymbol{4}$, since three other players would have bided by the time our focusing player can make another move.

3.2 Algorithm

3.2.1 Random Walk

While more accurately this should be called an experiment, we have decided to put the analysis on the results of random walk here for a better comparison with the other approaches. It will also help explain some of our incentives when designing our algorithms.

The idea of random walk is simply randomly returning a bid with equal opportunity regardless of the information provided in the current information set. The first observation we made in the game is that it is very easy for the bidding to overshoot – that is, to pass the point of no return.

We define two different points of no return for each pair. The first one is the internal point of no return which can be defined for both pairs, one pair or none for each deal. It is the highest possible contract bid that each pair can bid with a chance of getting positive score, given idle opponents and perfect card-play. For example, imagine a deal where every hand is mediocre in terms of strength and shape, such that the only makeable contract is *1NT* for the NS pair and EW pair has no possible contracts. Then a bid of *1C*, *1D*, *1H*, *1S* or *1NT* from the NS pair all has a chance of getting positive score in the end, that is: $\max_{\pi_{NS}} \boldsymbol{u}\ (\boldsymbol{a}, \boldsymbol{\pi_{NS}}) > \mathbf{0}$, where $\boldsymbol{\pi_{NS}}$ is the strategy of the NS.

On the other hand, any bid from *2C* onwards can only receive negative scores and there is no legal action to avoid it. (Note that in this project, we will always assume unflawed perfect information card-play.) The point of no return for NS pair is thus *1NT*, and the point of no return for EW pair is undefined since they do not have a profitable contract. In some cases, where neither pair has a makeable contract, the point of no return for both pairs are undefined.

The second point of no return is the external point of no return, which is in essence the best contract, the contract that gives the par value of each deal. For example, if the best contract for a deal is *4Hx by NS*, then any bid from *4Hxx* all the way to *7NT* passes the external point of no return, in that the best contract is no longer reachable, or that the Nash equilibrium is no longer approachable. Note that *4H* bid by the EW pair also passes the point of no return.

The observation that we made regarding overshooting the contract is true for both definitions, where the bidding always ends at the 7-level, well pass the makeable contract for either pair. We then observe this is due to the rule of bidding in Bridge, where the bidding phase does not end until three passes are bid. With each bid at time $\boldsymbol{t}$ being equally likely, the probability of the bidding phase ending in $\boldsymbol{t+2}$ is always the lowest among possible outcomes.

$$\pi_t(pass) \times \pi_{t+1}(pass) \times \pi_{t+2}(pass) \leq \pi_t(a) \times \pi_{t+1}(b) \times \pi_{t+2}(c) \tag{3.1}$$

This is because contract bids reduce the action space of the next node, whereas *Pass* does not. This explains why any contract below *7C* is rarely passed out for both random walk and undertrained networks.

3.2.2 First Approach (Regret Minimization)

Let us recall that the counterfactual utility at an information state is given by $\boldsymbol{u_i(\pi, I)}$ and the cumulative counterfactual regret of an action $\boldsymbol{a}$ at information state $\boldsymbol{I}$ is given by $\boldsymbol{R_i^T(I, a)}$.

$$u_i(\pi, I) = \frac{\sum_{h \in I, h' \in Z} p_{-i}^{\pi}(h) p^{\pi}(h, h') u_i(h')}{p_{-i}^{\pi}(I)}$$

From Eq. (2.23)

$$R_i^T(I, a) = \frac{1}{T} \sum_{t=1}^{T} p_{-i}^{\pi^t}(I)(u_i(\pi^t|_{I \to a}, I) - u_i(\pi^t, I))$$

From Eq. (2.24)

While in CFR Minimization, the regrets of all information states and actions $\boldsymbol{R_i^T(I, a)}$ are stored and updated, we did not take this approach since the memory requirement of this approach is large and some terms are easier to calculate in the game of Poker than Bridge.

For example in Poker, $\boldsymbol{u_i(h')}$ has an output of either the pool money or zero with inputs being the card strength of both (or more) sides; In Bridge, an external agent is needed to find the most number of tricks of a specific trump suit, and returns a value $\boldsymbol{u_i(h')}$ ranging from -7600 to 3160. Another example

is the term $\sum_{h \in I} p_{-i}^{\pi}(h) / p_{-i}^{\pi}(I)$, which in Poker requires iteration through the combination of two hidden cards with a known policy π; Whereas in Bridge, 39 hidden cards need to be considered even with a known policy π. The sheer function space of the game makes dynamic programming near impossible.

This is why we attempt to train a network which returns the normalized regret value of all actions a (illegal actions included), given information set I. That is, $f_{\theta}(I) \approx R(I)^{[a]}$. Instead of incrementing the regret values at each time step t, we decrease the regret of the chosen less optimal action. In order to prevent the change of policies from opponents from affecting the assumption in Eq. (2.23) and Eq. (2.24), actions and states encountered by only one player are used to train the network after each episode of self-play.

To simplify the task and have faster training, some practical adaptations were done on the algorithm. Firstly, dealer of every deal is always North and the position is always vulnerable for all players. We set the position to be always vulnerable with the intention of deterring overbidding. Secondly, the statistics of other players' hands are always the real time exact statistics, to take away the unknown influence in generating a bidding knowledge base. Thirdly, only two networks are trained in one episode instead of the maximum four to reduce complexity. The EW pair always bid *Pass* instead.

Algorithm 1 - Adaptation of Regret Minimization

```
1   Initialize networks with random parameters θ
2   for each episode do
3       Generate random deal
4       Initialize empty bidding sequence
5       while bidding not ended
6           Provide playing network with statistics of other players' hands,
            bidding sequence and cards
7           Receive a vector of normalized regret value
8           Calculate probabilities among legal moves
9           Choose a bid according to probability
10          Append the bid to bidding sequence
11          Check if bidding ended
12          Repeat with next player
        end while
13      Decrease regret value of ensured wrong moves of learning network
14      Repeat with next learning network
    end for
```

We shall now clarify our design choice of decreasing regret value and the meaning of *ensured wrong move* on line 13.

Consider a bidding sequence *3C*, *Pass*, *5N*, *Pass*, *Pass*, *Pass*, by extracting the bid by NS pair and changing the notation, we get *3C-5N-P*. Let us assume the result is -400 since the pair could only make 1NT ideally. What should we do in a traditional regret minimization framework?

We would compare the counterfactual utility of the bid *3C* with all the 35 other legal opening bids given by the player's hand and public information and update the regret value accordingly. To find the counterfactual utility, for each opening bid, we would need to generate a combination of 13 cards (for the partner's hand) to 39 cards (for all three hands), then construct and traverse a game tree with maximum branch factor of 35 and maximum depth of 35. Although the tree should be skewed in general, total searches should be of comparable scale of Poker, without considering opponent's strategy.

Therefore, we instead chose to reduce the regret of wrong moves, ensuring that the correct moves have more probability of being chosen. The definition of wrong move is given as the move that leads to a state where $\max_{\pi} \boldsymbol{u}\,(\boldsymbol{a}, \boldsymbol{\pi}) < \boldsymbol{par}$, with par value adjusted according to pair orientation. By this definition, a simple indicator of wrong move should be any bid beyond the external point of no return. However, this indicator is considered sub-optimal, since we would like to encourage our agent to beat the par. Our final choice is hence the combination of external and internal point of no

return. If the agent bids a bid that is beyond both external and internal point of no return, such move is considered a bad move.

In the above example, *3C-5N-P*, since *3C* is beyond both internal and external point of no return, the regret value of action *3C* given the information state (the hand held by North and an empty bidding sequence) in hence steered towards zero. It is done by setting a target vector identical to the original vector except replacing the value corresponding to *3C* to zero. All the other bids are not evaluated, since we do not wish to deter bidding *Pass* and *saving attempts* of subsequent bids, that is the attempt to reduce expected loss by changing contract.

3.2.3 Second Approach (MCRL)

Our second approach is based on the MCRL used by AlphaZero. We learnt from MCRL that a reinforcement learning network can be combined with MCTS to form a policy interaction cycle. This allows the network to automatically find the correct answer and learn from it. Hence our approach is an adaption of MCRL. Procedures for simplifying the task are adopted in a manner similar to the first approach, in particular, the statistics are provided and EW pair always bids *Pass*.

For each episode, an MCTS is conducted and assisted by network $\boldsymbol{f_\theta(I) = p}$, and we then train the network using the MCTS visit probability $\boldsymbol{\pi}$. We discarded value $\boldsymbol{v}$ and outcome $\boldsymbol{z}$ from MCRL because it would ultimately be

useless in actual gameplay – it is practically impossible to look forward in actual gameplay when policies and cards are all unknown.

Algorithm 2 - Adaptation of MCRL

```
1   Initialize networks with random parameters θ
2   for each episode do
3       Generate random deal
4       Initialize empty bidding sequence
5       for each iteration do
6           Initialize empty simulation bidding sequence
7           while bidding not ended
8               Provide playing network with statistics of other players' hands,
                bidding sequence and cards
9               Receive a policy vector p
10              if playing network equals learning network
11                  Adjust policy vector p with sample values and visit count
                end if
12              Choose the bid with maximum value in p
13              Append the bid to simulation bidding sequence
14              Check if bidding ended
15              Repeat with next player
            end while
16          Update visit count and value
        end for
17      Choose the bid with maximum visit count
18      Append the bid to bidding sequence
19      Provide actual visit probability π as training data for learning network
20      Repeat with next learning network
    end for
```

4 Results

To compare the performances of different algorithms, deals are randomly generated and then bided by agents. The score of each resulting contract (under perfect card-play and defence) is then compared with the par value. The result of random walk is also provided as comparison. It should be noted that besides *Players*, *Deals Tested* and *Lvl 7 Boards*, all other gathered results are based off par value of each deal. That is, if our agent gains 1430 points for a deal, it means that it outperforms the par value for 1430 points.

	Regret Minimize	*MCRL*	*Random Walk*
Players	2	2	2
Deals Tested	1,000	1000	1,000
Max Gain	1,420	2,220	1,420
Max Loss	-2,920	-2,220	-2,820
Total G/L	-638,846	**18,490**	-626,400
Squared Total Error	701,394,744	557,534,500	655,661,400
Total IMP	-9,740	**324**	-9,753
Lvl 7 Boards	**805**	0	858

Remarks: Gain, Loss, Total G/L and Total Imp are in NS perspective

For Regret Minimization, the first thing we noticed is that the loss in our trained network is even higher than a random walk, it lost 12,446 marks more than randomly generated biddings over 1000 boards, on average losing 124

marks more per deal. Other than that, the statistics are mostly similar. However, we did manage to reach the less 7-level contracts by a reasonable margin, showing that the algorithm brought some improvement in its area of focus.

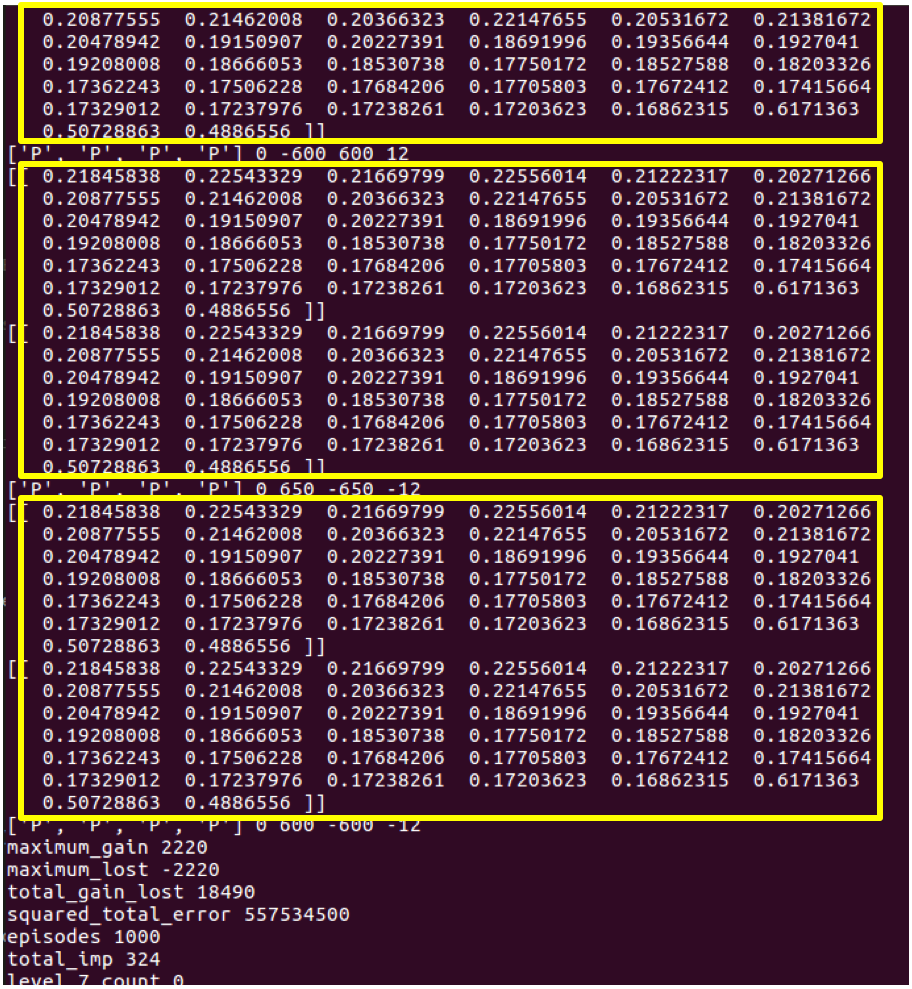

Fig 4.1 Raw output vector of MCRL network during testing

For MCRL, the results are very different from the control as well as the first algorithm. In particular, the agents beat the par value in terms of both raw score and International Match Point (IMP) [20]. It also never reaches a 7-level contract. The reason becomes clear when we look into the bidding process of the agents. For every deal, the agents would always bid *pass* and

end the bidding process in one round of bidding. The fluctuation comes merely from the randomness of each deal.

Looking further into the details, we realise that the network always produces the same output vector regardless of input vector provided (Fig 4.1). We hence suggest that the large state space and the low correlation between state-action pair and resulting score results in a network that over-generalizes.

5 Architectural Design

In this chapter, we will go into even more details concerning the application of algorithm. We will first discuss the information representation actually employed, for example what is a state $\boldsymbol{s}$ or an information state $\boldsymbol{I}$ in our codes. We will then talk about the overview and some components in our program.

5.1 Information Representation

The observed state (or the information set) at each decision making step is made up of three components, the player's cards, previous bids and information encoded in those bids.

To encode the cards of each player, we use the effective and simple one-hot encoding, where the presence of any one of the 52 cards would be coded as one, and zero otherwise. This method is chosen since we do not wish to limit or provide bias to how our agents should perceive each card using our own experience.

As for previous bids and output bids, they are once again encoded with one-hot encoding. The main design choice is then the number of previous bids to remember. AlphaZero uses an 8-step history [4,5], and we chose to use a 10-step history. This is because in most complete information games, the domain AlphaZero is trying to tackle, the current state of the game holds the most

information. Whereas in the bidding phase of Bridge, the bidding history contains most of the information.

While except at the very early stages of sandboxing, bidding base was no longer employed, its mechanism was still carefully considered. We chose to keep the approximate average of past statistics of each player's hand given the player bid such way. For example, if a player bids *1C* on an empty bidding sequence, the statistics we obtain from his hand is used to update the statistics that we stored away using the formula.

$$prev_{stat} \times (1 - \eta) + stat \times (1 - \eta) \rightarrow prev_{stat} \tag{5.1}$$

Besides saving memory space, such method will also ensure our statistics react to the latest changes due to constant step size. This means that our bidding base would constantly adapt to new changes in the change of strategy by the agents.

The statistics that we chose to maintain is the approximate average of the length of each suit, number of top cards (from Ace to 9) and the presence of specific top card in each suit [Appendix 1]. This is the part with the most expert knowledge involved. As a reference, in human bidding system, a bid usually conveys information such as the length of one or two long suit, the suit quality of the named suit, high card point and the presence of control in a suit. The way information is conveyed is usually by promising a minimum in any combination of the above quantity although there may also be

exceptions. One such common exception is the asking bid, asking partner to further clarify his/her hand in the shorter suits or the range of high card point. This is the hardest to code into a bidding system.

(For those interested, high card point (HCP) is a quantity to represent hand strength. An Ace is worth four points, a King is worth three points, a Queen two points and a Jack one. Hence the highest HCP obtainable in a hand is 37.)

(Control refers to the presence of Ace/King in a suit, which usually allows the declarer to win the first trick following the attacking lead by the defenders. This allows the declarer to perform other desirable actions. In gaming terms, this opens up the possibility of the declarer to choose a move most beneficial to his/herself. Since the purpose of a control is to win the liberty of playing the next card, a void/singleton in a suit in a contract with trump is also considered control.)

5.2 System Architecture

The system is written in Python, backed by **dds** [21], **dds-py** [22], **tensorflow** [23] and **Keras** [24]. It is formed by four main components, with some commonly used codes isolated out in a separate module called ***Helper***. The four components are ***deal generator***, ***knowledge base***, ***agent*** **and** ***game master.*** *Deal generator* generates deals required for training, along with the par value and result table needed to assess the performance of agents; *Knowledge bas*e is used to store the meaning of each bid for each agent; *Agent* is the class which translates the output of network into biddings and translates observation of states into inputs; Lastly, *game master* is the main program where the codes are bound together with various learning algorithms implemented. Now we would illustrate what a full version of the program looks like.

Prior to any training, the *game master* loads in or initializes one or more networks and *knowledge bases*. It then loads in four *agents* and specifies the network and knowledge base each one will be using. They would become the players at the table. In one episode of training or playing, the *game master* asks for a deal from the *deal generator*, which passes on the four hands, the par value and the result table. It then passes the hands to the *agents* and asks for a bid, starting from the dealer. The public knowledge is then updated, followed by the update of bidding sequence and the meaning of the bid. After that, the next player is asked for a bid, this continues until three passes are bid. When the bidding process is completed, the resulting score is obtained

from the result table, which is then used to train the agent in ways defined in different learning algorithms.

Unfortunately, we soon realise such an algorithm would take forever to converge even if it does converge and hence we adopted a simpler version, which only gives the agents the responsibility of giving meanings to biddings and liberty of making a bid. Some common simplifications are static dealer, static vulnerability and the removal of obstruction from opponents.

5.2.1 Deal Generator

Generating deal is a trivial task which could be done simply by shuffling a deck of cards and distributing evenly. The main workload of the *deal generator* is actually getting the par value of each deal and the optimal result of each possible contract. There are a lot of card abstraction, game tree expansion and heuristics involved. Tremendous amount of time is saved with the help of double dummy solver by **Haglund** and **Hein**, as well as its Python API by **Hemminga**. I am truly grateful for their contribution to the community.

5.2.2 Knowledge Base

Communication is a key part of the project and this could not be possible without *knowledge base*. At each state, an agent is prompted to provide a bidding based off its observed state. For the other agents, partner and

opponents alike, the information provided should be comprehensible in some way for the bidding process to be meaningful. If we instead let two networks train as inputs their own cards and the output of each other, regardless of whether or not they converge, this is regarded unfair gameplay since they are communicating through a secret channel – the information is comprehensible only to themselves. Hence the use of a knowledge base is to translate such output (bid) into information of an agent's cards.

There have been two different approaches to knowledge base in this project. The first one is to construct the meaning of all biddings manually, using expert knowledge. With the help of a Python snippet, such a bidding system has actually been constructed in the form of a dictionary. The system used is the ***Hong Kong Youth Team System 2018*** and it takes in biddings that were translated into strings in a strict format as key, and gives out strict keywords as outputs. For example: System ["2D"] would return "5-11 HCP; 5+ Hearts OR 5+ Spades". (HCP here means *High Card Point*, a system devised by human players to calculate hand strength. Naturally we did not plan to explain it to our program, only expecting it to understand the meaning behind eventually.)

It was hoped that at each bidding instance, the agent would match the bidding history with the system, hoping to find the longest matching string to obtain information about the hands. The problem, however, arises as we try to translate such tokenized information into network input. Minimum and maximum of HCP and cards in each suit could be entered into the network in

10 input nodes, yet the task became harder and harder if we were to consider cases with multiple meanings, such as various forcing bids, bidding of opponents' suits, or change of suit at high level. These biddings introduce a combination of scenarios and the variation makes the design of input nodes very challenging.

The second approach we used and were ultimately adopted (and abandoned) is the from-scratch bottom-up approach, where agents input statistics about their hands after making a bid, updating the statistics of such a bid in the process. Although past statistics were ultimately not used in any agent's training and we provide accurate current statistics instead, we hope that this approach can be used when they are sufficiently trained and bidding actually carries meaning.

5.2.3 Game Master

Game master allows us to conduct various reinforcement learning experiments in one place. To load and save a network, we used the built-in library of Keras [24] to save it as h5 format. As for bidding base, we used JSON to dump the dictionary and load it as per required.

For our RL network, various versions have been tested (all with inconclusive results). The version as of now is a neural network with 486 input nodes, 38 output nodes and 4 hidden fully connected layers with batch normalization between most layers [Appendix 2].

5.2.4 Agent

The *agent* is an object responsible for providing a bidding given the network and observed state. It is also responsible for all the works associated with the main goal, such as: Aid in the learning process by keeping record of input and output vector; Translating the state information into input vector as per the network's requirement; Selecting a bid from the output vector after modifying it with values, visit counts and explore coefficient.

It is also responsible for the calculation of the card statistics and updating the bidding knowledge base. Although this functionality is obsolete since in our experiments the agents always have the current statistics provided.

6 Discussion

In this chapter, we would discuss the difficulties faced in solving the problem of bidding in Bridge, followed by various reasons for failure of our algorithms. Lastly, we would talk about the incentives and possible reward solving this problem could provide. We hope that future researches in similar topic could take these challenges and rewards into account before commencement.

6.1 Difficulties

6.1.1 State Space

The main development direction of our program is to use neural network to reduce the memory requirement of an imperfect information agent. Instead of mapping every information set or distinguishable states, to strategies deterministically, we were hoping that this function could be approximated and different observable states generalized with a neural network. It was obviously much harder to train a network under this circumstance, since each information set contains multiple states, which introduces greater variation and fluctuation to the training.

The high variance in bridge worsens the situation still. In poker, a two-player, limit Texas Hold'em has just under 10^{18} game states [10], whereas in bridge, even when we fixed the hand that the decision maker holds, there are already holdings in the magnitude of 10^{16}; And by simply considering possible

opening bids a dealer can make, the number of possible game states increases to more than 10^{18}. The sheer number of different combinations make the problem extremely challenging.

In the making of Libratus [18], abstractions were made on various levels, including bet sizes and cards. In comparison, our network did not receive any abstraction and was receiving raw hands and biddings as inputs. This is one of the main reasons our network fail to converge.

The failure seems even more unavoidable when we take into account that human players usually require at least a year of training before bidding in bridge begins to make sense – when we understand the rationale and benefit behind each bid. In comparison, if we were to play chess, poker or even go, even if we have a low level of play due to our poor planning or exploitable strategy, at least the game makes sense to us. This marks the level of abstractness humans employ in the game before it starts to be playable.

6.1.2 Correlation

The second difficulty that we face is the weak correlation of cards, biddings and results. Consider a real world scenario, where two beginner bridge players practice their bidding skills by only bidding and checking with the result table generated by the computer. Both players have the identical mind-sets and there is no misunderstanding between the bids of the two players. Now the dealer bids *1NT* and the partner bids *2C*, how should we evaluate

this position? Are we in a good spot or a bad spot? What is the expected return of such a bid compared with opening with *1H*? We simply do not know.

This is vastly different from abstract complete information games at the core of the game. In games such as Chess, the board position provides every information needed to evaluate the state, it is relatively clear what losing a Castle would mean and what luring opponent's Queen would do to us. Same goes with poker, if we hold a pair of pocket Aces and the pot never grew and we are at the forth public card, or turn already, we know we were in a bad spot for not utilizing our cards fully. From the difference in how humans play the game, we suspect that Bridge is less correlated than other games in terms of states and results.

This is a similar argument to the large state space of the game itself, but it is different in its core. A problem with a large state space could have high correlation between its states and its values, or expected utility, then the main obstacle of such a problem is to go through all the representative cases in an effective manner. A problem with a low correlation, however, would always have a slow convergence rate since it would be harder for the agents to pick up the determining details.

Some may suggest that once the communication between agents becomes more informative, they would train at an exponentially faster rate since the information from both hands are crucial in determining the best action. However, an experiment done with supervised learning showed otherwise.

	First Approach	*NT_by_N*
Players	2	1*
Deals Tested	1,000	1,000
Max Gain	1,420	1610
Max Loss	-2,920	-1,960
Total G/L	-638,846	-176,330
Squared Total Error	701,394,744	449,294,610
Total IMP	-9,740	**-3,036**
Lvl 7 Boards	805	**0**

Remarks: Gain, Loss, Total G/L and Total Imp are in NS perspective

(Table 6.1)

The algorithm NT_by_N is simply an algorithm that uses a trained network to predict the maximum number of tricks North can get without using a trump suit. The data passed to train such network is the raw cards of all four hands, and it is thus also the input of the algorithm. The algorithm assumes that opponents and partner all pass after the dealer, North, places down the guess. It also assumes perfect information scenario. The result, as we can see, is not as overwhelming as we would expect.

While the algorithm bid no 7-level contracts, it on average only performs 72% better than our first approach. It performs about 69% better in IMP lost. While No-Trump contracts might not be ideal for every hand, we would still expect

to see positive gains if our network can avoid overbidding. While some may suspect that an undertrained network might be the main cause of poor performance, our network only had a mean squared error of 3.79 on test data and is the output of a 3-generation genetic algorithm. We argue that the network is well-trained and the low correlation between cards, biddings, and results is one of the main difficulties future similar researches may face.

6.2 Evaluation

6.2.1 Correct Move

Our first algorithm was simple and brutal. The idea was that if a move led to a bad outcome, it should be punished, and in some implementations of the algorithm, rewarded if it led to a good outcome. We were hoping that by pushing to network to move in the general direction of a correct move, we could increase the probability that a good move is produced, similar to regret minimization.

What we did not notice however, was the importance of a reference move, or the so called correct move. In most algorithms that utilize game tree, this is the move found producing the highest expected return by iterating the tree; In CFR Minimization, all the unchosen moves are basically reference moves, and the regret of not choosing each of them is taken into account. While this is tedious and computationally expensive, it lowers the branch factor and reduces that number of searches we need, by focusing on the promising moves.

This is analogous to humans learning with a correct answer to guide them. In the early stages, our learning is often supplemented by the presence of a correct answer, and our goal is to receive the reward associated with attaining such an answer. We could also fine-tune our approach if we missed by a

fraction of a margin, or review our decision making further back in time if the answer was missed by a long shot.

6.2.2 Data Efficiency

The data efficiency of our first approach is also low, which ultimately leads to a poor performance. Take CFR as an example, a visit to a state gives us information about the best move, how good it is compared with our chosen move, as well as implicitly how good it is compared with other moves. On the other hand, by using our algorithm, we eliminate a poor choice in every visit and provide no extra information. While ultimately the best move should come out by elimination, this requires at least $\boldsymbol{n}$ number of visits per observable state, where $\boldsymbol{n}$ is the number of legal bids.

The probability of visiting the same state $\boldsymbol{n}$ times is already low enough with randomly distributed hands. By adding in a second agent however, the probability that the same state is visited again is even lower. As observed in our agent, every attempt to bid is basically random and opens up a new state way more often than necessary. The low learning rate directly causes the poor quality of learning data, which leads to even lower learning rate.

6.2.3 Resources

We have stated in project scope that we hope to give full liberty to our neural networks, such that they would not be limited by our limitations and bring

new insights to the game and to machine learning. This is why it had minimal abstractions and inputs from human. Such an approach will naturally deal with a much larger state space and as such require a powerful machine to increase the chance of success. However due to the ease of implementation, the project was conducted solely on a desktop computer accelerated with a 2015 GPU. This constitutes a large part of failure.

Take our first approach as an example, its correctness lies largely on the probability that every action is experienced at least once for every information state $\boldsymbol{I}$. This is because it eliminates bad moves as episode count $\boldsymbol{T}$ increases. With a small $\boldsymbol{T}$, the chance that our agent would similar states after training is significantly reduced.

6.3 Reward

6.3.1 Cooperation

One substantial feature in our project is the element of cooperation in reinforcement learning. As more and more aspects of our lives suggest machine learning to be a solution for automation, the lack of cooperative elements in these agents becomes more and more apparent. We can see agents solving problems intelligently without explicit directions in the area of speech recognition [25], chess [26], scheduling [27], but we rarely hear news about two agents cooperatively learn from scratch and achieve substantial feats.

The ability to cooperate, however, is crucial to pushing the boundary of artificial agents to the next level. Not only because this is the natural way of development, taking reference of various living beings, but also because the limitation of a huge, powerful entity has always proven to be less impactful than distributed competent components with examples such as the Internet, Prosumer and Bitcoin. It is envisioned that one day, agents trained for different tasks could work together to solve greater tasks without explicit controls from humans.

6.3.2 Communication

Bidding in essence is a form of communication, where players selectively disclose important parts of their hands to their partners, and sadly their

opponents too. Bidding systems can hence be regarded as a language, despite having difficulties only faced by its kind. For example, in everyday languages such as English, we have a wide range of vocabulary as well as limitless amount of space, granting it the capability to express any abstract and concrete part of our world, albeit occasionally inefficient.

This is contrasted by the limited number of tokens and amount of space in bidding. While the number of tokens, 36, is not exactly small, each token comes with two hidden costs when being bid. The first one is that each bid robs the partner of linearly increasing amount of bidding space. For example, a 6NT bid leaves only five biddings possible responses for the partner. This is why bidding sequences rarely starts at the 4-level, leaving effective starting bids to be 16. Secondly, each bid comes with the cost that the pair will end up playing at the named level, at the very least. For example, staring from 1C-1S-3C, the pair has committed themselves of playing at the 3-level.

These two constraints together limit the usual bidding sequences to a length of six. And in 3 bids, the pair has to convey information about their shape (length of each suit), strength (usually in the form of High Card Point, calculated from face cards) and controls (presence of Aces and Kings). At least this is the case in existing human bidding systems.

If an artificial agent is able to construct and use a language in this sandbox-like environment, a language that captures the essence of each hand, conveying in a manner and sequence such that most scenarios could be

covered as partner branches off in each node, this would hopefully point to a new direction in knowledge representation and compression, easing the workload of devising a new communication paradigm as we introduce AI to other aspects of the world.

6.4 Suggestion

If other researchers would like to continue on our work, and that existing bidding systems are not provided, then it is suggested that the problem not be taken as a decision making or game tree solving problem but as a language construction problem. However, if an existing bidding base is explicitly provided, then the problem should be better defined as the former two categories.

The reason is that most decision making or game tree solving models do not have such a high focus on information. Incomplete information games are either tackled with rewards or by dynamic programming and the concept of information in perfect information game is obviously obsolete. However, if we could view the problem as the combined problem of language construction and MDP, then a bid can be considered both a token of communication and an action, whereas the information conveyed could be separated from the umbrella term information set. This should be more beneficial in developing a solution to the problem.

Either way, we believe that training should be organized in an ascending level of difficulty. For example, extreme hands should be introduced first and only opening bids should be expected from our agents. It should be trained with supervised learning at first. For example, a hand of 8 solid Hearts and 10 winning tricks should return an extremely high favourability on the opening bid “4H”; Or the hand with the shape of 4333 (where the longest suit has a

length of 4 and shortest 3) and 0 HCP should bid “Pass” with high probability as well. The difficulty then lies in obtaining such samples.

After training on opening bids, it could then be expected that the statistics in the bidding base is accurate to a certain extent. The agents could then practice bidding as partners with the condition that bidding phase immediately terminates after both bid at least once. In this stage, the agents should learn how to evaluate their own hands and combine it with the information provided by partner’s bidding.

This process continues and expands until each pair of agents is given adequate space to develop the ability of gradual disclosure of their hands as well as premature termination. After which they could then bid in the presence of opponents, which could simply be copies of themselves since playing against the same network does not necessarily leads to overfitting into itself, as shown by the results of AlphaZero [4].

For the actual update algorithm and knowledge base, we believe that there could be various viable changes. For example, for the update algorithm, the experience to be replayed could be stored using circular buffers [6,28] to reduce correlation between training data and increase stability of the training of the network. Also, the expected value of each bid could be retrieved to see if there were unfound correlation between player’s cards, bidding history and the chosen bid. As for the knowledge base, it would perhaps be more beneficial if a mechanism to encode scenarios or bidding sequences could be

implemented. More complex biddings could then be put to the test and be observed through our agents.

7 Conclusion

Solving incomplete information games is a hard task, more so in the presence of cooperative element. While not much experimental data was produced in this project, the experience learnt is crucial and valuable. I hope that one day in the future, I could witness the advent of cooperative RL agents and the birth of a bottom-up Bridge bidding system and agent. I also hope that the idea of information and communication could be focused more by researchers in the field of automatic decision making agents.

References

1. Zhang, H. and Xu, S. (2016). The Face Recognition Algorithms Based on Weighted LTP. *Journal of Image and Graphics*, 4(1), pp.11-14.

2. Sayed, M. (2018). Biometric Gait Recognition Based on Machine Learning Algorithms. Journal of Computer Science, 14(7), pp.1064-1073.

3. S., S. and Kamade, P. (2011). Video OCR for Video Indexing. *International Journal of Engineering and Technology*, 3(3), pp.287-289.

4. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K. and Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419), pp.1140-1144.

5. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T. and Hassabis, D. (2017). Mastering the game of Go without human knowledge. Nature, 550(7676), pp.354-359.

6. Minh, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. arXiv:1312.5602v1

7. Hart, S. (n.d.). *Games in extensive and strategic forms*. [S.l.]: [s.n.], pp.20-40.

8. SUTTON, R. (2017). *REINFORCEMENT LEARNING*. 2nd ed. London, England: MIT Press.

9. Mirsoleimani, S., Plaat, A., van den Herik, J. and Vermaseren, J. (2017). An Analysis of Virtual Loss in Parallel MCTS. *Proceedings of the 9th International Conference on Agents and Artificial Intelligence*.

10. Zinkevich, M., Johanson. M., Bowling, M., Piccione, C. (2007). Regret Minimization in Games with Incomplete Information. Proceedings of the 20th Advances in Neural Information Processing System

11. Bengio, Y., Louradour, J., Collobert, R., Weston, J. Curriculum Training. (2009) International Conference on Machine Learning

12. Blackwell, D. (1956). An analog of the minimax theorem for vector payoffs. *Pacific Journal of Mathematics*, 6(1), pp.1-8.

13. John N Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. Automatic Control, IEEE Transactions on , 42(5):674–690, 1997.

14. Long-Ji Lin. Reinforcement learning for robots using neural networks. Technical

report, DTIC Document, 1993.

15. Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In Machine Learning: ECML 2005 , pages 317–328. Springer, 2005.

16. En.wikipedia.org. (2019). *Monte Carlo tree search*. [online] Available at: https://en.wikipedia.org/wiki/Monte_Carlo_tree_search [Accessed 28 Apr. 2019].

17. Rosin, C. D. Multi-armed bandits with episode context. Annals of Mathematics and Artificial Intelligence 61, 203–230 (2011).

18. Brown, N. and Sandholm, T. (2017). Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374), pp.418-424.

19. fledsu (2019). *关于德州扑克 AI 中 Counterfactual Regret Minimization 的介绍*. [online] 知乎. Available at: https://zhuanlan.zhihu.com/p/30438383 [Accessed 29 Apr. 2019].

20. Pi.math.cornell.edu. (2019). *IMPs and Matchpoints*. [online] Available at: http://pi.math.cornell.edu/~belk/impmp.htm [Accessed 29 Apr. 2019].

21. Haglund, B. and Hein, S. (2019). *dds-bridge/dds*. [online] GitHub. Available at: https://github.com/dds-bridge/dds [Accessed 29 Apr. 2019].

22. Hemminga, F. (2019). *atolab/pydds*. [online] GitHub. Available at: https://github.com/atolab/pydds [Accessed 29 Apr. 2019].

23. TensorFlow. (2019). *TensorFlow*. [online] Available at: https://www.tensorflow.org [Accessed 29 Apr. 2019].

24. Keras.io. (2019). *Home - Keras Documentation*. [online] Available at: https://keras.io/ [Accessed 29 Apr. 2019].

25. Geitgey, A. (2019). *Machine Learning is Fun Part 6: How to do Speech Recognition with Deep Learning*. [online] Medium. Available at: https://medium.com/@ageitgey/machine-learning-is-fun-part-6-how-to-do-speech-recognition-with-deep-learning-28293c162f7a [Accessed 29 Apr. 2019].

26. David, E., Netanyahu, N. and Wolf, L. (2017). End-to-End Deep Neural Network for Automatic Learning in Chess. *International Conference on Artificial Neural Networks (ICANN)*, [online] 9887, pp.88-96. Available at: https://arxiv.org/abs/1711.09667.

27. Sądel, B. and Śnieżyński, B. (2017). Online Supervised Learning Approach for Machine Scheduling. *Schedae Informaticae*, 1/2016.

28. Heinrich, J. (2017). Reinforcement Learning from Self-Play in Imperfect-Information Games.

Appendix 1 – Input Features

As the basis of our proposed knowledge base and bidding system, the information conveyed by each bid / sequence of bid has the following format:

1. *[# of Spades, # of Hearts, # of Diamonds, # of Clubs]*
2. *[# of Aces, # of Kings, # of Queens, # of Jacks, # of Tens, # of Nines]*
3. *[# of Spade Ace, # of Spade King, # of Heart Ace, # of Heart King, # of Diamond Ace, # of Diamond King, # of Club Ace, # of Club King]*

The three vectors are then concatenated in the listed order.

Appendix 2 – Network Structure

The network used to produce the results and apply algorithms in general have the following structure:

1. *Input layer of 486 input nodes*
2. *A fully connected linear layer to a hidden layer of size 128*
3. *Batch normalization*
4. *A fully connected linear layer to a hidden layer of size 128*
5. *Batch normalization*
6. *A fully connected linear layer to a hidden layer of size 128*
7. *Batch normalization*
8. *A fully connected linear layer to a hidden layer of size 128*
9. *Output layer of 38 output nodes*

The network, unless otherwise specified, uses mean squared error as its loss function and stochastic gradient decent as its optimizer. It also applies L2 regularization to all fully connected hidden layers.