Deepbot - Build a poker bot using Big Data and Deep Learning techniques

Cyril van Schreven

Supervised by:

Nguyen Thanh Tam

Prof. Karl Aberer

Electrical and Electronic Section

EPFL

Master Project Thesis

Distributed Information Systems Laboratory

5 Jul, 2019

Abstract

Texas Hold'em Poker is a challenging game for intelligent agents. The information presented to the player is partial, the outcome of actions is non-deterministic, and the number of decision points is much larger than for other board games. Moreover, the strategy of the opponent is unknown and may be deceptive. In a Sit and Go, the player should adopt different strategies as the game evolves. This report presents an agent playing No Limit 6 Handed Sit and Go against highly exploitable agents, and one novice agent. The playstyle adopted at various tables is then analyzed. The model is an end-to-end neural network featuring recurrent neural networks for game and opponent modeling. The network is evolved with a genetic algorithm. In order to develop an agent that performs well across multiple tables, heuristics are used in the selection of elites. Experimental results show that the agent successfully beats his various opponents and has a strategy that evolves as the game progresses.

Contents

1

Introduction

1.1 Motivation

Games are an important topic for research in Artificial Intelligence. They offer a properly delimited environment, which is suitable for computer programs. It permits the training and validation of novel models which could not directly be implemented in real world applications. The research in Chess and the Game of Go has brought considerable progress in the field of reinforcement learning.

Poker is different from most other board games. Mainly because it is an imperfect information game and it has non-deterministic outcomes. As the players don't have all the information of the game state, they must make an estimate of it to the best of their knowledge. As the outcome of an action is stochastic, all possible outcomes have to be taken into consideration. This makes poker one step closer to a real world environment, especially to fields which are not traditionally engineered. Indeed many decisions that humans have to take rely on imperfect information and the outcome of the decision taken is non-deterministic. For example, poker research brings a lot of content for the modeling of negotiations.

Moreover poker is a game that may be played with more than two players. This has many implications on how the game should be approached. The player loses a lot of control on the dynamic of the game. The player can be affected by interactions in which he did not partake.

Last but not least, attempting to dynamically adapt to opponents or new situations yields many interesting problems. Determining the quality of an adaptation is already a tricky problem in itself. The ideal adaptation would weight in each new piece of information perfectly. Some pieces of information may be very important. For example

if an opponent reveals cards that are way out of the estimated hand range, the opponent model should be adapted radically. This is an easier task for humans than for computer programs.

1.2 Challenges

The development of an intelligent agent for playing poker is a challenging problem in many regards.

Imperfect information. Poker is an imperfect information game: the players do not obtain all the information about the game state. In poker, the cards of the opponents are unknown. In substitution of the game state, the player relies on a belief state, whether explicit or not. Typically the belief state could express the probability that the opponent is holding a set of cards. An important implication is that the opponent will also have a belief state. If that belief state becomes too accurate, the player will get exploited. Therefore the actions of the player must not be too expressive: they should not be deterministic. In other words the Nash Equilibria are attained by mixed strategies. In addition, in order to have an accurate belief state, the player is forced to consider the sequence of action that led to a certain situation. He cannot look only to the future, but must also look at all the past actions. This is why the complexity of the problem is proportional, not to the number of game states, but to the game tree size.

Complexity. A decision point is defined as an instance of the information set: the hole cards of the agent, the public cards, the sequence of actions and the stacks of the players. Heads Up (HU) Limit poker has 10^{13} unique decision points. The version of HU No Limit poker played at the Annual Computer Poker Competition (ACPC[1]) has 10^{161} unique decision points[2]. The reason no-limit poker has so many decision points, is that each legal raise, by increments of 1 chips, are different actions. In comparison, the state space complexity of chess is 10^{47} and the state space complexity of the game of go is 10^{170}.

The number of decision points increases dramatically for certain alterations, such as: increasing the number of players, having an evolving blind structure, and maintaining an a model of the opponent.

Stochastic outcomes. Another difficulty with poker is that the outcomes are stochastic. Even when the player is playing the optimal strategy, he may lose. Thus the feedback given to the player may be deceptive. However as the law of large numbers states,

it is truthful over many events.

6 Handed Sit and Go. For an agent playing Sit and Go with multiple players, additional aspects must be taken into consideration. First regarding Nash equilibria. If the opponents are playing around a different Nash equilibrium than the agent, the theoretical guarantees of the agent are lost and it may be exploited. The agent can thus not rely on a static precomputed strategy. Second, in Sit and Go the profit of the agent depends on the finishing place. Thus the expected earning is not directly proportional to the number of chips won. This is the so-called Independent Chip Model (ICM): the aim is to survive to get to the last ranks, therefore winning a chip is worth less than it costs to lose one. Two opponents going all-in with equal odds will get negative Expected Value (EV), while the the players not involved will get positive Expected Value. Because of the ICM, the game may be influenced by the kingmaker effect. The kingmaker effect is when an opponent quickly gets a large stack by exploiting weak opponents. That opponent can then use his chip advantage to bully the agent and prevent him from making consistent profit. Therefore to be EV positive, the agent must not only be unexploitable, but it must also actively exploit his opponents.

1.3 Related work

Breakthroughs in Poker. Major breakthroughs in computer poker have been achieved by a combination of Monte-Carlo simulations and Counterfactual regret minimization (CFR). This first breakthrough was Cepheus that "essentially weakly solved" Limit HU Texas Holdem [3]. It uses a variant of counterfactual regret minimization: CFR+ [4]. Regret is measured as the incurred loss for taking a certain action at a decision point instead of the optimal one. The value of taking an action is measured by simulating the hand until it ends multiple times. Not all possible game endings are simulated as it would be intractable; Monte-Carlo simulations are used. A similar approach was used to beat professional poker players in No-Limit HU by both Libratus[5] and DeepStack[6]. Strong contributions of these works are heuristics for pruning the search tree and the use of subgame solving.

Overall the focus in the field is on 'solving' the game: finding or approaching a Nash equilibrium. This yields a Game-Theory-Optimal (GTO) agent that can by definition not be exploited. The research on modelling and exploiting opponents is also dense nevertheless. When adapting to an opponent, a strategy will naturally distance itself from the Nash equilibrium. It will therefore also become exploitable.

Probabilistic opponent modeling. One approach to model the opponents is with a statistical model[7]. The opponent's actions and showdown frequency are tracked and used in the decision making. These values are tracked separately for each decision point in a strongly reduced representation of the information set. One weakness is that a lot of hand data is required in order to have a reasonable model of the opponent. A method to alleviate this is to keep a model of players in general. The distribution of this model is then combined with the player's distribution to assist states where data is lacking[8].

Neural networks in poker. Neural networks have proven their efficiency in various games. They have been used for board games such as Chess and the Game of Go, as well as for computer games such as Super Mario and Atari Games. In poker they are not as common. DeepStack uses a neural network to solve a task within the CFR algorithm. Instead of simulating the possible endings of the hand until the end, the neural network is used at a certain depth to estimate the value. Neural networks have also been used to predict the next action of the opponent [9]. They have been deployed as end-to-end models to directly predict actions. In this case evolutionary networks have proven useful [10]. A work[11] closely related to the one presented here uses a recurrent neural network and exploits the memory of the network for opponent modelling in Heads Up.

2

Overview

In this work, the aim is to develop an agent that is capable of playing 6 Handed Sit and Go and exploiting weak opponent. Therefore the agent is not trained through self-play nor does it approach a Nash equilibrium. The agent presented is built almost entirely on a neural network. The method to model the opponents is not an explicit probabilistic model, instead it is inherent to the neural network. The agent trains through reinforcement learning: he acts in his environment and updates his decisions according to the rewards obtained.

The agent is trained and validated on two different game formats. The first format is Heads Up Ring Game. Its purpose is mainly to verify the validity of the model. The second format is 6 Handed Sit and Go. Though the two formats are presented in parallel, the reader may skip the parts relative to one or the other.

At the tables, the agent plays against various highly exploitable opponents. In addition, one rule-based bot is introduced. The rule-based bot is directly implemented from a human readable strategy. Apart from it being a better opponent, it yields an interesting study on how that strategy may be exploited.

The decision algorithm of the agent is handled by an end to end neural network. Each time the agent has to make a decision, a forward pass is performed trough the network. The features abstract the observable state of the game. The network's output represents the amount of chips the agent is willing to put in the hand. He then folds, checks, calls, or raises accordingly.

The use of a neural network presents a major advantage for the information set abstraction. Instead of treating the data as categorical, it may be treated as continuous. For example the price to call may be represented by a single value instead of what would be a large number of separate states in a regular reinforcement learning algorithm. In

addition a neural network exploits the closeness of inputs: it is able to make predictions from states that it has never encountered before.

The neural network presented is in majority composed of Long-Short-Term-Memory (LSTM) cells. Recurrent networks are powerful for spotting patterns in sequences. Intuitively the game of poker may be seen as a sequence of hands, a sequence of actions, or a sequence of information sets. This is the main motivation for using LSTMs. The memory of the network is exploited to reduce the amount of features fed to the model. The features capture the state of the table as it may be observed at the current instant. It is the recurrent network that has to do all the heavy lifting with relation to the past. This includes the previous information of the current hand as well as the previous information of the whole session.

The parameters of the neural networks are updated with a genetic algorithm (GA). This choice presents multiple advantages over the more popular backpropagation. The main difference between the two is that evolutionary algorithms optimize globally rather than locally and that their search direction is not as efficient. Backpropagated methods are not suitable for the problem at hand as actions cannot yield loss functions independently. The rewards only come once per hand in Ring Game, and once per game in Sit and Go. In addition, the stochasticity of the outcome motivates the reward function to be the result of multiple actions. Overall evolutionary algorithms tend to learn faster, but are unable to reach a minimum as closely as a gradient-based method would. Considering the complexity of the problem, this seems like a suitable trade off.

3

Environment

3.1 Games

Two different games format are played by the agents in this project. The poker variant is always Texas Hold'em poker, No Limit.

3.1.1 Heads Up Ring Game

The first format considered is Heads Up, Ring Game. There are 2 players at the table and the reward of a player is directly determined by the number of chips he won.

The initial stack is of 3'000 chips and the big blind (BB) is of 100. This means each player starts with 30 big blinds. 30 big blinds is considered short stack. There is some room for stealing blinds but hands quickly transform into all-ins. It is therefore important to control the pot-size and only get invested in hands when having good equity. For reference, a stack of 15 big blinds or less is usually advised to by played with an "all-in or fold" strategy.

The players play sessions of 500 hands. A session can have any number of games. Each time a game ends, a new one is started and the initial stacks are reset. Since the number of games is not fixed, there is no advantage in being risk averse. The action with the highest Expected Value should be taken in any situation.

To alleviate the variance, a so-called match consists of two mirrored sessions. After the first session the players swap their places and the decks are repeated. A player thus gets exactly the same hands the second game as the opponent had the first game. The opponent modelling that the agent may have developed is wiped at the end of each session.

virtual time	small blind	big blind	ante
$0 \rightarrow 90$	10	20	0
$90 \rightarrow 180$	15	30	0
$180 \rightarrow 270$	25	50	0
$270 \rightarrow 360$	50	100	0
$360 \rightarrow 450$	100	200	0
$450 \rightarrow 540$	100	200	25
$540 \rightarrow 630$	200	400	25
$630 \rightarrow 720$	300	600	50
$720 \rightarrow 810$	400	800	50
$810 \rightarrow 900$	600	1200	75

Table 3.1: Blind structure

3.1.2 6 handed Sit and Go

The second format played is 6 handed Sit and Go. A game starts with 6 players and finishes when one player has all the chips. This can be seen as a one-table tournament. The reward is determined by the finishing place; the rank. The buy-in, or investment, is of 1 token. The reward for finishing first is 4 tokens, and the reward for finishing second is 2 tokens. As the Independent Chip Model highlights, the players are encouraged to take fewer risk in order to make it to the later ranks. This is especially true when only 3 players remain at the table.

The initial stack is of 1'500, and here the blinds are ascending. In live poker the blinds increase over time. However time is not relevant for bots. A virtual time is thus introduced. The virtual time is proportional to the number of hands played and the number of remaining players in each hand. Each player's first action in a hand is a tick. The effect is that if fewer players are remaining, they will be able to play more hands before the blinds increase. When the virtual time passes a certain threshold, the blinds increase on the next hand. The blinds increase each 90 ticks. If 6 players remain, this corresponds to 15 hands. The blind structure is defined in Table 3.1. A game takes in most cases between 100 and 200 hands. This is similar to an online game where blinds increase every 10 minutes.

The first three levels of blinds are relatively low and leave space for more actions in each hand. At the fourth level, a player that remained at its initial stack will feel a lot of pressure from the blinds and should make plays. The purpose of the antes, appearing at the sixth level, is to force players with tiny stacks to get in the hand.

To alleviate the luck factor, a match consists of 6 games such that the player sits

at each position. The decks are the same for each game. The opponent modelling that the agent may have developed is wiped at the end of each game.

3.2 Opponents

The agent faces various opponents.

3.2.1 Highly-exploitable bots

These opponents are defined to cover various extreme strategies.

- The Caller: This player always calls (or checks). He is very Loose-Passive.
- The Maniac: This player always raises. If the street is not raised yet, he will raise by 1 time the pot size. If the street is already raised, he will raise by 2 times the pot size. He is very Loose-Aggressive.
- The Conservative: This player folds a large majority of his hands. Every now and then he calls or performs a small raise. The amount of chips he wants to commit to the pot is:

$$des\_wager = ini\_stack * eq^7 \tag{3.1}$$

  where des_wager is the desired wager, ini_stack is the initial stack (1500), and eq is his equity. He is very Tight-Passive.
- The Attached: This player also folds a majority of his hands. When his hand suits him, he will call with it until the showdown. The hands he plays are an Ace or King combined with an Ace, King, Queen or Jack. Just like the Conservative he is Tight-passive. The difference is that when he joins a hand, he will not leave it.
- The Equity bot: This player acts according to his equity. When he is confident, he is aggressive and cannot be made to fold. If his equity is higher than 80%, he raises by two times the pot size. If his equity is higher than 70%, he raises by one time the pot size. If his equity is higher than 60%, he performs a minimum raise. If his equity is higher than 50%, he calls. And otherwise he folds. Empirically this bot encouraged the agent to play in a proper manner. Indeed the bot has some control over the pot, and cannot be bluffed out of the hand. Therefore the approach to exploit him is to be smarter on the equity and bet sizing then he is.

3.2.2 Rule-based bot

The rule-based bot, referred to as the Ruler, follows a more complex strategy than the other bots. He plays Tight-Aggressive: he folds often, never limbs or checks, and makes strong raises when holding a valuable hand. It is a direct implementation of the Sit and Go beginner strategy from PokerStrategy[12]. Such a strategy is suitable for beginners as it does not lead to difficult decisions. It is based on human knowledge and heuristics that have been developed over time.

Apart from the the hole cards and the public cards, the Rule-based bot uses four pieces of of information.

- Wether he has more or less than 12 big blinds in his stack.
- His position at the table.
- Wether the street has been raised. If it has, from what position.
- The current street

The positions at the table are separated in three groups: blinds, late and middle. The blind positions are from where the blinds are posted: Small-blind and Big-blind. The late positions are the two positions preceding that, the Button (or Dealer) and Cut-off. And finally the middle positions are the two positions before that, so-called Under-the-gun 1 and Under-the-gun 2.

The strategy can then be separated in two sub-strategies. One is used whenever the player has more than 12 big blinds, and the other one is used whenever the player has 12 big blinds or less. The first sub-strategy differentiates preflop from other situation, whereas the second one uses an 'all-in or fold' approach and thus never makes decisions past the preflop. The bot also plays differently if the street has been raised before him. Finally the bot looks at the cards and takes a decision.

Above 12 big blinds, preflop. When the agent has more than 12 big blinds and is on the preflop he plays according to Table 3.2.

When the street has already been raised, the second action in the cell is taken. The amount to which the wager is raised also depends on whether it is the first raise on the street. The raise amount is described in Table 3.3, where opp_calls is the number of opponents that called into the hand and $prev\_raise$ is the value of the previous raise.

Above 12 big blinds, postflop. When the agent has more than 12 big blinds and is after the flop, he will choose his action depending on the combination of cards he has or might make. The decision taken are summarized in Table 3.4

Hole cards	Middle	Late and Blinds
AA, KK, QQ, AK	Raise	Raise
JJ	Raise / Fold	Raise
TT, 99, 88, AQ	Raise / Fold	Raise / Fold
77, AJ, AT, KQ, KJ	Fold	Raise / Fold
Other cards	Fold	Fold

Table 3.2: Above 12 big blinds, preflop: decision table

Is first raise	Raise amount
True	$(3 + opp\_calls) * BB$
False	$3 * prev\_raise$

Table 3.3: Above 12 big blinds, preflop: raise amount

Top pair and above includes: a top pair, an overpair, two pairs, three of a kind and any combination above that. Strong draws are always made of 4 cards. It includes: flush draws where the agent holds an Ace or King of the suit, flush draws where the agents holds two cards of the suits, straight draws where the 4 card ranks are consecutive.

The amount to which the wager is raised is detailed in Table 3.5

Below 12 big blinds. When the agent has 12 big blinds or less, he plays an all-in or fold strategy. Table 3.6 determines the hands that are raised all-in if no one entered the hand before. Table 3.7 presents the hands that are raised all-in if an opponent entered the hand before.

3.3 Information set abstraction

At each decision point, the player has access to the current information set. The information set is the hole cards and the public state. As the number of unique decision points is very large, it is important to abstract the information set to a simpler form without losing too much information. As explained in chapter 2, the memory of the LSTMs is exploited to this end.

Hand	Flop	Turn	River
Top pair or better	Bet / Raise	Bet / Raise	Bet / Raise
Strong draw	Bet / Raise	Bet / Raise	Check / Fold
Any other	Check / Fold	Check / Fold	Check / Fold

Table 3.4: Above 12 big blinds, postflop: decision table

Is first raise	Raise amount
True	$(2/3) * pot$
False	$3 * prev\_raise + pot$

Table 3.5: Above 12 big blinds, postflop: raise amount

Player's position	Stack in BBs	Offsuit cards	Suited cards	Pairs
Middle	$10 \rightarrow 12$	AK - AQ	AK-AT	AA-66
	$6 \rightarrow 10$	AK-AT KQ-KT QJ-QT JT	AK-A2 KQ-K4 QJ-Q8 JT-J7 T9-T7 98	AA-22
	$0 \rightarrow 6$	AK-A2 KQ-K8 QJ-QT JT	AK-A2 KQ-K4 QJ-Q6 JT-J7 T9-T6 98-96 87-86 76-75 65	AA-22
Late and Blinds	$0 \rightarrow 12$	AK-A2 KQ-K4 QJ-Q9 JT-J9 T9	AK-A2 KQ-K2 QJ-Q2 JT-J7 T9-T6 98-96 87-86 76-75 65	AA-22

Table 3.6: Below 12 big blinds, unactive opponents: all-in hands

Opponent's position	Stack in BBs	Offsuit cards	Suited cards	Pairs
Middle	$7 \rightarrow 12$	AK-AQ	AK-AQ	AA-77
	$0 \rightarrow 7$	AK-AJ	AK-AJ	AA-77
Late	$7 \rightarrow 12$	AK-AT KQ	AK-A7 KQ	AA-55
	$0 \rightarrow 7$	AK-A5 KQ-KT QJ	AK-A5 KQ-KT QJ	AA-55
Blinds	$7 \rightarrow 12$	AK-A8 KQ	AK-A2 KQ	AA-22
	$0 \rightarrow 7$	AK-A2 KQ-KT QJ JT	AK-A2 KQ-KT QJ JT	AA-22

Table 3.7: Below 12 big blinds, active opponent: all-in hands

Heads Up Ring Game.

In the Heads Up Ring Game, the model gets 8 features:

- The current street (preflop, flop, turn or river) as a one-hot-encoding. It costs four features. It has to be a one-hot-encoding as there is no continuity between the streets and each has to be considered separately.
- The equity of the hand. The equity is the probability of winning the hand if all active players went to the showdown. It is the only information about the cards that the player gets. It calculated by Monte-Carlo simulations. In each simulation a random hand is given to the opponent, and random community cards are added to have a full board. The winner of the hand is then determined. These results are then averaged to estimate the equity. The simulations are repeated until a satisfying standard deviation on the estimated equity is achieved. The tolerance is set to 0.1%. Note that the hand range of the opponent is not explicitly modelled here as random cards are distributed to him. The equity calculation is the most time consuming operation in the agent's decision making, thus also in the generation of game data. In the implementation used it takes approximately $10ms$. The repeated estimation and comparison of hand strength is the reason for this relatively long computation.
- The investment of the player in the hand, normalized by the initial stack.

- The investment of the opponent in the hand, normalized by the initial stack. The investments summarize the importance of the current hand and may describe the strength of the hand the opponent is representing.

- The pot odds. The pot odds are the amount of chips necessary to call divided by the total pot. It is an important measure often used by professional players. In combination with the equity it can give a simple estimate of the Expected Value of calling. Indeed with the assumption that the player's action is the last until showdown, the action of calling is EV positive if and only if the pot odds are lower than the equity.

All the features are between 0 and 1.

6 handed Sit and Go.

In the 6 handed Sit and Go the features are separated in two groups: the general features and the opponent features.

There are 12 general features:

- The current street (preflop, flop, turn or river) as a one-hot-encoding.

- The number of active opponents, normalized by the initial number of players; 6.

- The position at the table. It is represented by a single value, measuring how many active opponents there are between the player and the big blind. It is also normalized by the number of players. In postflop play the big blind is the last to take action and is considered the best position. Therefore the lower this value the better.

- The equity of the player. In this game format it is multiplied by the number of active players. This way it represents the strength of the hand and the neural network does not need to figure it is always lower when there are more players.

- The equity of the player if the showdown was on the flop, also multiplied by the number of active players. If the current street is later than the flop (turn or river), it is set to 0. This features adds information about the cards. Mainly it helps to distinguish made hands from drawing hands. Tight players typically play made hands where they can be confident of their strength on an early street. Having this measure for the flop and not for other street is based on heuristics. This feature is not theoretically grounded as there is never a showdown on the flop. A stronger measure would be the standard deviation of the equity of the hands the

player could obtain on the flop. However this would require a very large increase in equity calculations.

- The player's current stack. Indeed short stack and deep stacks should be played differently, even more so in Sit and Gos. This feature is also normalized by the initial stack.
- The amount of chips that must be added to call, normalized by the initial stack.
- The total pot, normalized by the initial stack.
- The value of the big bling, normalized by the initial stack.

Note that the feature describing the chips and stacks is slightly different from the ones used in Heads Up. The pot-odds are not given explicitly but the information necessary to calculate it is available.

All features are between 0 and 1 except the equities. These can range from 0 to 6, but are more than often between 0.5 and 1.5.

The 4 opponent features only describe the actions of the opponent:

- The action taken by the opponent, as a one hot-encoding. It takes 3 features for Fold, Call and Raise.
- The amount of chips the opponent added to the hand, normalized by the initial stack.

4

Model

The model developed is an end-to-end neural network. A forward pass is done through the network at each turn of the agent. The input is the information set abstraction as described in section 3.3. The output is a single value between 0 and 1 that determines the proportion of the initial stack that the agent is willing to set the wager to. The agent then folds, checks, calls or raises by increments of the big blinds accordingly. The network is trained through neuroevolution. Overall the approach is strongly inspired by ASHE [11].

4.1 The neural network

Two different neural networks are presented: one for each game format. Though they are different, they follow a general structure that is similar and have some modules in common. They are composed of recurrent networks in parallel which are followed by a feed-forward network. The recurrent networks are composed of multiple Long-Short-Term-Memory modules (LSTM).

All the weights and biases are initialized from a uniform distribution:

$$param_i = U(-k, k) \tag{4.1}$$

where $k = 1/sqrt(layer\_size)$

The initial cell state and initial hidden state are initialized from a normal distribution with mean 0 and variance 1.

$$
\begin{aligned}
i_t &= \sigma(W_{ii}x_t + W_{hi}h_{(t-1)} + b_i) \\
f_t &= \sigma(W_{if}x_t + W_{hf}h_{(t-1)} + b_f) \\
g_t &= \tanh(W_{ig}x_t + W_{hg}h_{(t-1)} + b_g) \\
o_t &= \sigma(W_{io}x_t + W_{ho}h_{(t-1)} + b_o) \\
c_t &= f_t * c_{(t-1)} + i_t * g_t \\
h_t &= o_t * \tanh(c_t)
\end{aligned}
\tag{4.2}
$$

Table 4.1: LSTM equations

4.1.1 Long-Short-Term-Memory

Each LSTM is a vanilla implementation as presented in Figure 4.1. They are composed of only one layer. The relevant equations are defined in Table 4.1 where, h_t is the hidden state at time t, c_t is the cell state at time t, x_t is the input at time t, $h_{(t-1)}$ is the hidden state at time $t-1$ or the initial hidden state at time 0, i_t is the input gate, f_t is the forget gate, g_t is the cell gate and o_t is the output gate.

4.1.2 Heads-Up

The structure of the neural network used in Heads Up is summarized in Figure 4.2.

There are two recurrent networks: one models the game of poker in general while the other one models the opponent. The difference lies in that the cells of the first are reset every hand, while the cells of the second are reset every session. The one modelling the game is referred to as the game network and the one modelling the session is referred to as the opponent network. The game network has memory of the current hand only. The opponent network has memory of all hands played against the opponent in the current session. The feed-forward network that follows is referred to as the decision network.

The recurrent networks are composed of 10 LSTMs. The LSTMs have a cell state, and a hidden state, of size 10. The decision network is composed of three fully-connected layers. The number of hidden neurons are, in order: 50 and 10. Each layer is followed by a hyperbolic tangent as activation function.

The 8 input features, as defined in section 3.3 are sent to the LSTMs in the recurrent networks. Their 200 outputs are concatenated and sent to the feed-forward network.

The LSTMs have 720 weights and 40 biases. In addition, the initial cell state c_0 and the initial hidden state h_0, each of size 10, are treated as trainable parameters.

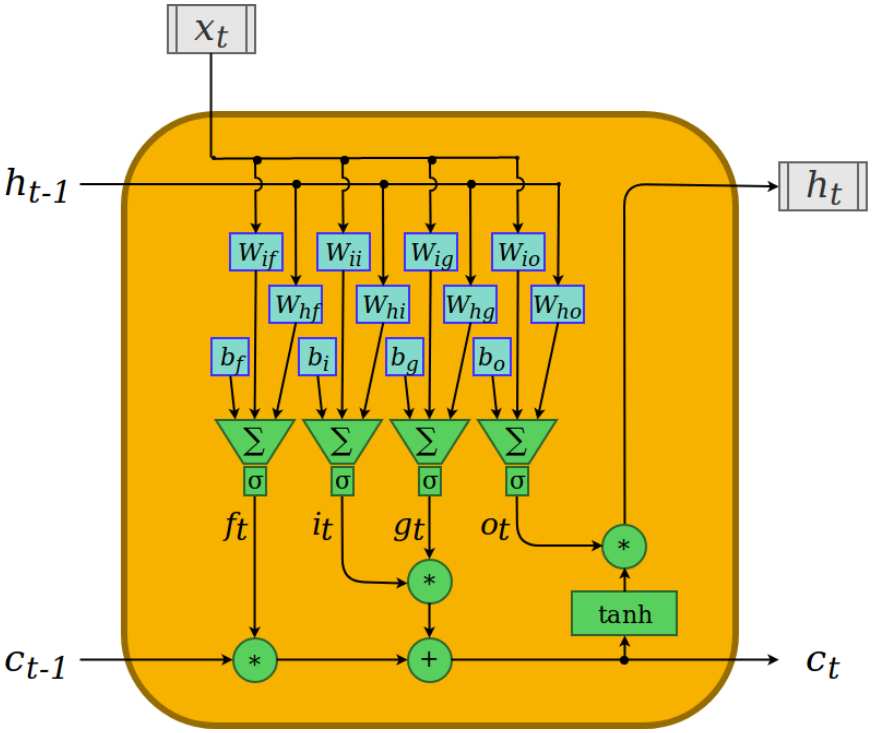

Figure 4.1: Long-Short-Term-Memory

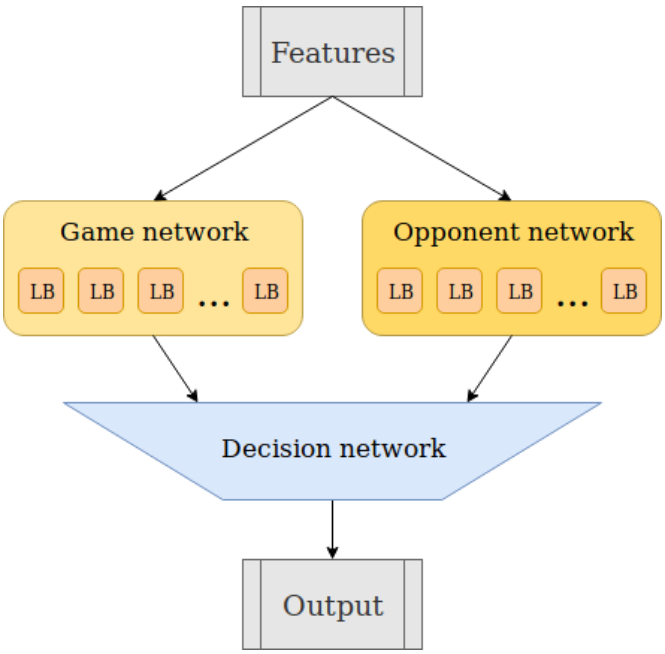

Figure 4.2: Heads-Up network overview

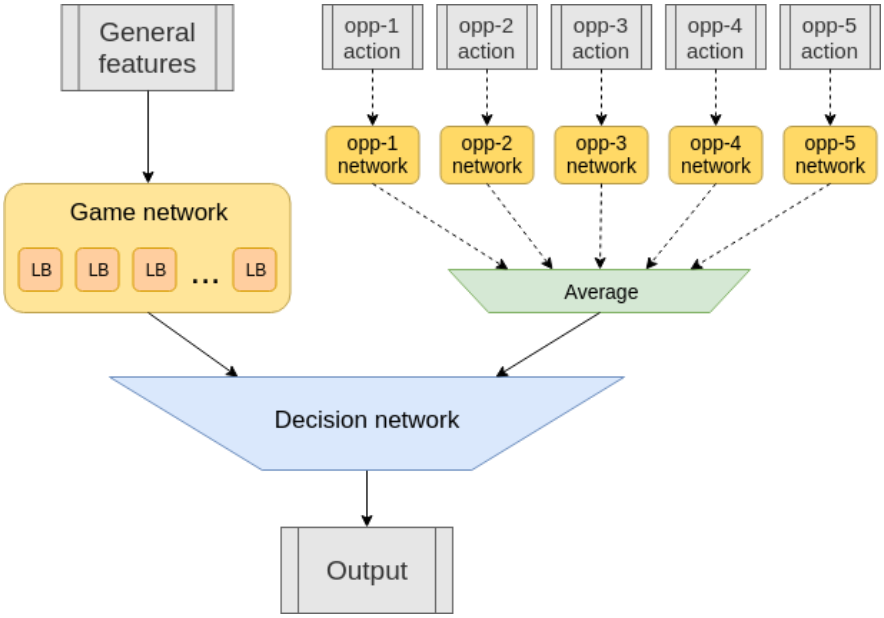

Figure 4.3: 6-Handed network overview

Therefore each LSTM has 780 parameters; each recurrent network has 7'800 parameters. The layers of the decision network have respectively 10'050, 510 and 11 trainable parameters. Thus the presented architecture has a total of 26'171 trainable parameters.

4.1.3 6-Handed

The structure of the neural-network used in 6 handed is summarized in Figure 4.3.

The game network and the decision network are the same as for heads-up. The opponent modeling however is handled differently. There is one network per opponent, and these networks are siamese: their parameters (weights, biases, initial cell state and initial hidden state), are all the same. However, as hands are played and forward passes are performed through them, the cell state and hidden state develops differently. The arrows connecting these networks in Figure 4.3 are dotted as they are not always active. First because some opponents will not have performed an action since the last forward pass: they have already folded or are eliminated. Second because the outputs of the opponents that fold are not forwarded further. As they have left the hand, modelling them is not relevant anymore. For clarity an example is depicted in Figure 4.4. Each opponent can be in one of three cases:

- He has already left the hand, therefore no action is recorded from him. This is the case of opponents 1 and 3.

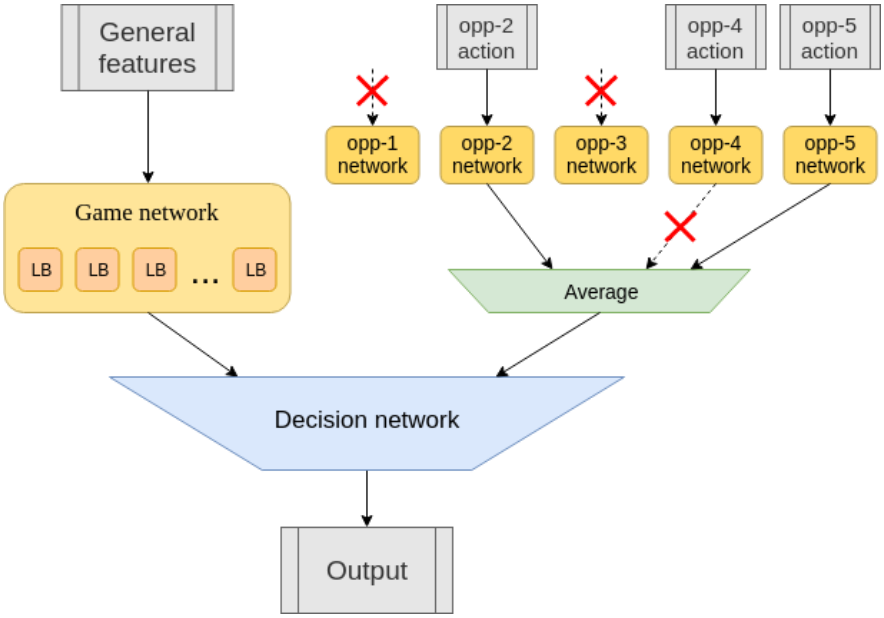

Figure 4.4: 6-Handed network activity example

- He left the hand, by folding. His action is transmitted to his network but not further. This is the case of opponent 4.
- He stayed in the hand, by calling or raising. His action is transmitted to his network, and the output of his network is transmitted further. This is the case of opponents 2 and 5.

The outputs coming from the remaining opponents are averaged and transmitted to the decision network.

The opponent network also has a structure that can be separated in two sub-networks. It's structure is illustrated in Figure 4.5. The distinction between the two sub-networks is the same as what has been done previously: the hand sub-network only has memory of the current hand while the session sub-network has memory of all actions of the current game. The idea is that the hand sub-network will model how aggressive the opponent is in the current hand, and the session sub-network will model how aggressive the opponent is overall. The sub-networks have 10 LSTMs vanilla cells. The cell state and the hidden state of these LSTMs are of size 5.

The 12 general features, as defined in section 3.3 are sent to the LSTMs in the game network. The 5 opponent features are sent to the LSTMs in the relevant opponent networks. These output 50 values for each of the two sub-network, which are then averaged. The 200 intermediary outputs are concatenated and sent to the feed-forward

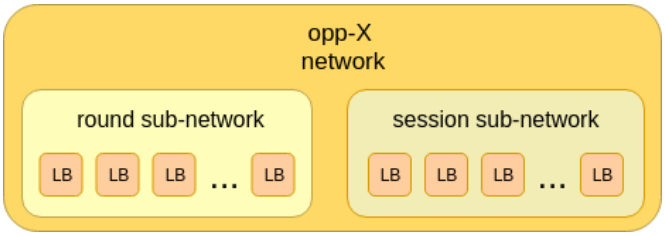

Figure 4.5: 6-Handed opponent network

network.

The LSTMs of the game network have 880 weights, 40 biases and 20 initial state values. The LSTMs of the opponent networks have 180 weights, 20 biases and 10 initial state values. With the 10'571 parameters of the decision network, the presented architecture thus has a total of 22'071 trainable parameters.

4.2 Neuroevolution: Genetic algorithm

4.2.1 Overview

The neuroevolution can be split in three parts: execution, selection and genetic operations. Each part is performed once per generation.

4.2.2 Execution and Selection

During the execution, the agents act in their environment. They play matches of poker against the 4 predefined tables, as detailed in section 3.1.

During the selection, the agents that performed the best are determined. The quality of their play is defined by the fitness function. A simple approach would be to sum all the gains (or losses) that the agent made. The aim however is to beat all tables. Therefore the fitness function is normalized. As the agents are losing at the beginning of the training, it is important to also properly reward them for making less losses. A different normalization factor is used for positive and negative values.

$$f(i) = \begin{cases} \frac{1}{m} \sum_{j=1}^{m} \frac{e_{ij}}{npos_j} & if\ e_{ij} >= 0 \\ \frac{1}{m} \sum_{j=1}^{m} \frac{e_{ij}}{nneg_j} & if\ e_{ij} < 0 \end{cases} \tag{4.3}$$

where in heads-up, the minimum is the big-blind:

$$\begin{aligned} npos_j &= max(bigBlind, max_i(e_{ij})) \\ nneg_j &= max(bigBlind, -min_i(e_{ij})) \end{aligned} \tag{4.4}$$

and in sit-and-go, the minimum is a tenth of the initial investment:

$$\begin{aligned} npos_j &= max(investment/10, max_i(e_{ij})) \\ nneg_j &= max(investment/10, -min_i(e_{ij})) \end{aligned} \tag{4.5}$$

From this fitness function, the top 30% agents are selected as survivors. The survivors are separated in two tiers. The elites are the agents performing better than the average of the survivors. The rest are second tier survivors. The elites will reproduce and pass to the next generation without mutation. The second tier agents will be added to the next generation after being mutated. This approach, introduced by [11], permits the strongest genes to persist without mutation, and the second-tier genes to not be fully dismissed. In the special case that there is only one elite, another random survivor is selected to be an elite. This is to enable cross-over.

In this setup the learning process may still suffer of neglect for one or multiple opponents. Typically the algorithm will learn how to beat 3 out of 4 opponents. The exploration will then continue in that direction and the 4th agent will never be beat. For this reason a heuristic is introduced. If an opponent is not beaten by any elite, the next best agent that does beat him is added to the list of elites. First the algorithm looks for an agent whose earnings, summed up with the earnings of the best elite, are positive for all opponents. If that fails, the algorithm looks for any agent that beats the opponent in question.

4.2.3 Genetic operations

To generate a new generation of agents, genetic operations are applied: cross-over and mutation. The heuristics implemented in these two methods will greatly affect the success of the neuro-evolution. Cross-over is the process of mixing genes of two or more agents in order to create a child. The aim is to find and assemble the genes that are the strongest on each parents. Mutation is the process of mutating genes in order to explore new solutions. In neuro-evolution, the genes are the parameters of the neural network.

Mutation. Mutation is performed by adding a Gaussian noise to randomly selected

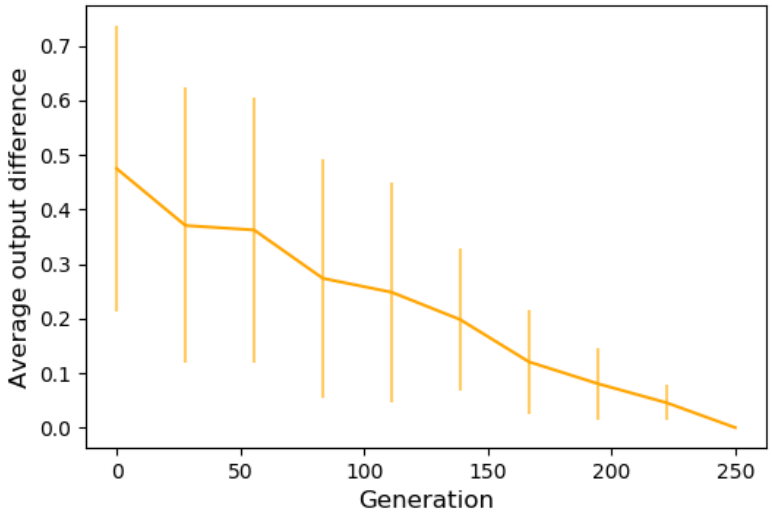

Figure 4.6: Difference in model's output caused by mutation

genes. In the first generations, the mutation must sufficiently affects the parameters to explore new solutions. As the algorithm reaches the later generations, the mutation must slightly change the parameters such that the information from the previous generation is preserved. The exploration is then more localized. For this reason, the mutation rate is set to linearly decrease from 0.3 to 0.05 while the mutation strength linearly is set to decrease from 0.5 to 0.1. The mutation rate is the probability that a gene is mutated. The mutation strength determines the standard deviation of the Gaussian noise added.

To verify that the mutation has the exploration rate required, a basic experiment is performed on the neural network of the Heads Up games. 1000 different random agents are generated, with 5 different hands to play. Each of these bots will be mutated by the mutation strengths and mutation rates that the model will encounter through the generations of his training. The mutated bots play the same 5 hands as their original version and the outputs are compared. Figure 4.6 summarizes the results of this analysis. The vertical bars represent the standard deviation.

The output difference caused by the mutations satisfies the requirements.

Crossover. Crossover is performed on two parents. It may be implemented in very various ways. It is important for the children's outputs to be correlated to that of both of the parents. If there is no correlation, it becomes a random search and the main purpose of crossover is defeated. However it should also leave room for children to be

better than their parents. Three different methods have been considered.

- The first method is the uniform crossover. Each parameter is chosen from either parent with equal probability. This method will properly mix the weight but does not take into account the underlying architecture; the neural network.

- The second method consist in taking the mean of the weight of both parents. This is not formally a crossover method as new genes are created. In fact the learning process is expected to behave more as an Evolution Strategy would. It also presents the disadvantage of there only being one possible child for each pair of parents. Nevertheless it is interesting to see how such this approach will affect the output.

- The third method takes the architecture of the neural network into account. It separates the parameters into groups. Then it randomly selects a group of parameter from each parent with equal probability. There are 5 groups of parameters for each LSTM module: the 320 weights that are applied to the input (W_{ii},W_{if},W_{ig},W_{io}), the 400 weights that are applied to the hidden state (W_{hi},W_{hf},W_{hg},W_{ho}), the 40 biases ((b_i,b_f,b_g,b_o)), the 10 initial cell states and finally the 10 initial hidden states. There are naturally 6 different groups of parameters in the decision network: the 10'000 weights and 50 biases of the first fully-connected layer, the 500 weights and 10 biases of the second layer, and the 10 weights and 1 bias of the last layer. As taking 10'000 parameters at once appears excessive, in a second version this group is split in 50 subgroups of 200 parameters. This results in 155 different groups in total.

To verify how closely related the children are to the parents, an experience is performed on each method. This is done on the neural network of Heads Up again. 1'000 pairs of random agents are generated and mutated once with a mutation rate of 0.175 and a mutation strength of 0.3. This is the mutation at generation 125. After that a child is generated with one of the methods. Finally the parents and the child face the same 5 hands. In Figure 4.7 the distance between the childs output, and the range stretching between parents' outputs, is measured. If the childs output is between that of the parents, the value is 0. The average difference between the parents outputs is added for reference. This value should not be compared directly to the values of the methods. In practice, as the training is in the later generations, the distance between the two parents becomes much smaller.

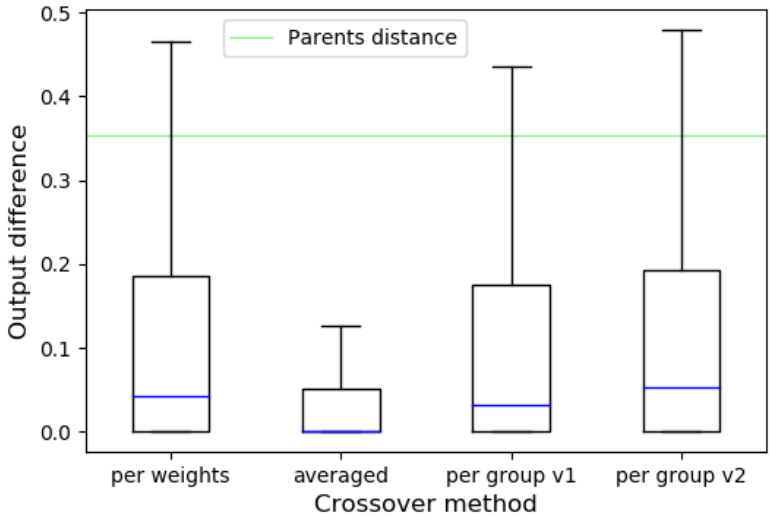

Figure 4.7: Difference of children's output to closest parent

The first observation is that the outputs of the second method are significantly closer to that of its parents than the other methods. More than 60% of its outputs are in between that of its parents. The first method and the two version of the third method perform similarly. Approximately 40% of the children outputs lie between that of their parents. Overall the crossover yields agents that act closely to their parents. Due to details given previously, the second version of the third method is retained and developed in the rest of this work.

5

Experiments

The performance of the developed agent is now measured and analyzed. The measures are performed on the best agent of the last generation. Better selection methods exist. However as the effect of mutation is small towards the last generations, the last agents have roughly the same performance.

5.1 Experiments' measures

Milli-big blinds per hand (mbb/hand). For Ring Games, the measure of performance is in milli-big blinds per hand, as detailed in Equation 5.1. In a real setting, solid players can expect to win about 50mbb/H. A performance above 100mbb/hand is considered outstanding.

$$mbb/hand = \frac{chips\_earned}{big\_blind * nb\_hands} \tag{5.1}$$

Return on investment (roi). In Sit and Go the measure of performance is the return on investment. It is defined in equation Equation 5.2. The roi, measured in percentage, is the net reward normalized by the investment. In the 6-handed setting presented in this work, the cost of participating is one token, and the reward for the first place is 4 tokens. Therfore the upper bound on the roi is of 300%. In a real setting, an roi of 30% is considered very solid.

$$roi = \frac{reward - investment}{investment} \tag{5.2}$$

In-game measures. When analyzing the strategy of the agent used in-game, a couple

of other measures will be used:

- Action Frequency: The frequency at which an action is chosen for a given situation.
- Raise amount: The amount to which the player raises. Not to be misinterpreted with the amount by which the player raises.
- Relative position: it is the position with relation to the big blind, taking only active players into account. A similar value is fed to the neural network.
- Blind-normalized Action Frequency (baf): This measure is introduced to cancel the effect of the blind level. It is used when assessing the effect of the position of the player. It is necessary because the strategy goes through various states during the game. The issue is that positions that are further from the big blind will mostly, if not only, appear early in the game. This would add a lot of bias. The Blind-normalized action frequency is measured as detailed in Equation 5.3. The effect of an action is normalized by the average frequency of that action at its corresponding blind level.

$$BNAF(a\_type) = \frac{\sum_{i \in dec\_pts}(\mathbf{1}_{action(i)==a\_type} * (a\_freq(bb\_lvl(i), a\_type) + eps))}{size(dec\_pts)} \tag{5.3}$$

  where a_type is the action type (raise, call, check or fold), dec_pts is the set of decision points, bb_lvl is a blind level, a_freq is the average action frequency for a given blind level and action type, and eps is added to avoid excessive influence of outliers, it is set to 0.1.
- The adapted equity: the equity multiplied by the number of players remaining in the hand. This is also a feature of the neural network.

5.2 Heads Up Ring Game

In this first experiment the agent is trained to play Heads Up Ring Games. It is referred to as the HU-agent. The opponents are: the Caller, the Conservative, the Equity bot, and the Maniac. This experiment shows whether the agent is capable of beating weak but very different opponents in a Heads Up game. The details of a match is given in subsection 3.1.1.

The neuroevolution has 250 generations and a population size of 60 agents. The execution phase consists of one match of 1'000 hands versus each opponent. The training thus requires the simulation of 60 million hands.

The average mbb/hand of elite agents during training is visualized in Figure 5.1.

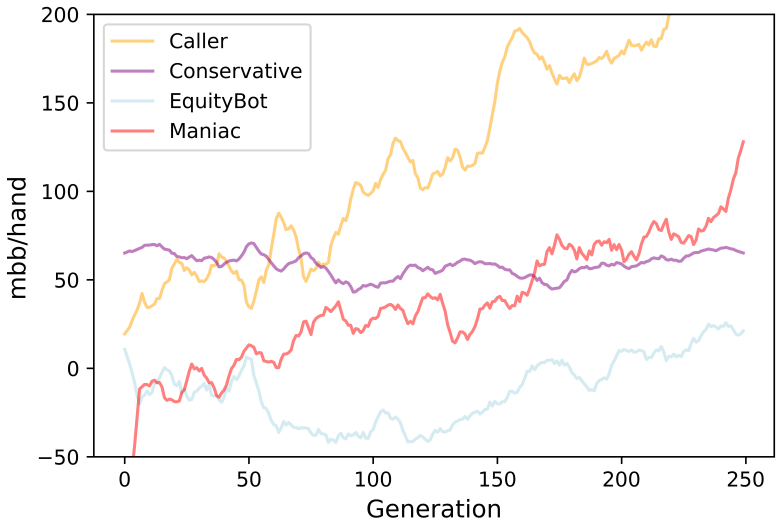

Figure 5.1: HU-agent: elite's average earnings

For validation, the trained agent realizes 1'000 matches against each opponent, for a total of 1'000'000 hands per opponent. The results of these games are summarized in Table 5.1.

Performance	Caller	Conservative	Equity	Maniac
mBB/hand	2'162	620	106	872

Table 5.1: HU-agent: mbb/hand, generation 250

The performance of the player against these opponents is very good. As the opponents are highly exploitable, it is on average much higher than what could be expected against real opponents. One value to keep in mind is size of the initial stacks: 30 big blinds. Indeed this is how much chips can be earning in an all-in. The performance is the best against the Caller, for more than 2 big blinds per hand. This is no surprise as the agent is free to control the pot size as much as he wants. Against the Conservative bot the earnings are around 600 mbb/hand. During training it seems as the agent does not learn much against this opponent. To exploit him it is only necessary to raise a small amount every time. One thing to note is that because the Conservative bot folds

a lot, the maximum exploitation is reduced. For comparison, if the conservative bot was always folding, and the agent always stealing the blinds, the earnings would be at 750 mbb/hand. As expected the Equity bot is the most difficult to exploit. Indeed he does not participate in hands where he is weak, and will increase the wager according to his confidence. However the performance is still high with about 100 mbb/hand. Finally the Maniac bot is also strongly exploited for around 850 mbb/hand. Though is is impossible to steal the blinds against him, he will accept to go all-in on any hand, like the Caller, which leaves a lot of room for exploitation.

Overall the model is capable of learning to beat multiple opponents. The main purpose of this experiment is to verify the validity of the model. The game data is thus not further analyzed.

5.3 6-Handed Sit and Go: fixed-agent

In this experiment, an agent is trained to play 6-Handed Sit and Go without opponent modeling. This version is referred to as the fixed-agent because his strategy at each hand is fixed for a given input. The agent plays only against one opponent: the rule-based bot. There are thus 5 instances of the rule-based bot at the table. As the opponent modeling is removed, only the game network and the decision network are used. This experiment yields a study of how much and how the rule-based bot can be exploited.

5.3.1 Performance

The neuroevolution has 250 generations and a population size of 70 agents. In the execution phase, the agent plays 4 full matches, resulting in 24 games. In total the training thus requires the simulation of 420'000 games.

The average roi of the agents during training is presented in Figure 5.2. The full population and the elites follow the same trend. The growth of the performance also suddenly increases around the generation 170. The hypothesis is that the mutation strength has to reach a small enough value for the agent to play a more precise game. From an non-formal analysis it appeared that the agent struggles to control his raise sizing at earlier generations.

For validation the agent plays 1'000 full matches, equivalent to 6'000 games. The performance measures are presented in Table 5.2. The density of the finishing rank is presented in Figure 5.3. For reference, the performance of the best elite at generation 200 is also presented.

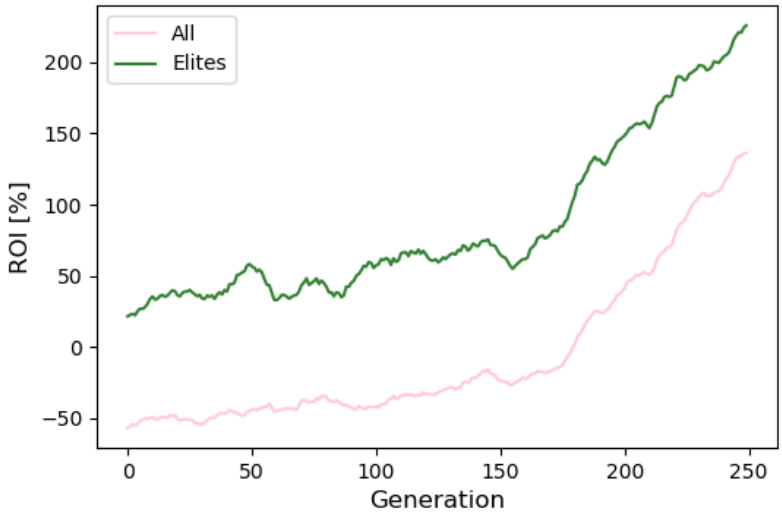

Figure 5.2: Fixed-agent: average ROI

Measure	Value
Average roi	204%
STD match roi	79%
Average rank	1.69

Table 5.2: Fixed-agent: Performance, generation 250

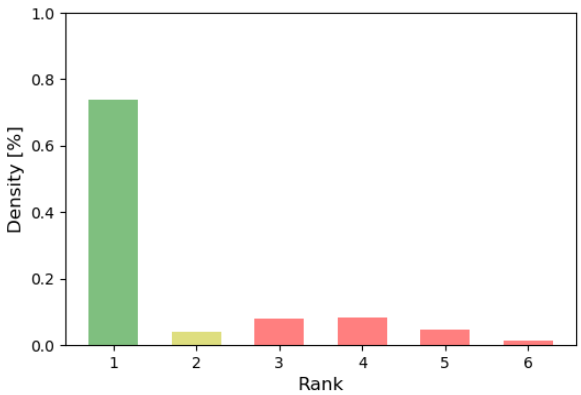

Figure 5.3: Fixed-agent: Rank density, generation 250

The trained agent performs extremely well and successfully exploits the rule-based bot.

Measure	Value
Average roi	93%
STD match roi	92%
Average rank	2.58

Table 5.3: Fixed-agent: Performance, generation 200

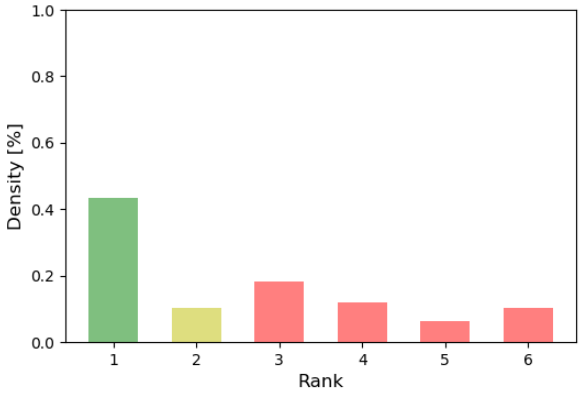

Figure 5.4: Fixed-agent: Rank density, generation 200

5.3.2 Strategy analysis

The strategy that the fixed-agent developed is now analyzed. As many different situation arise and there are many factors influencing the agent's decision, the analysis cannot be exhaustive.

One thing to point out is that the observations in this section are obtained through induction. This may yield erroneous conclusions. For example towards the end of the game, the amount the agent raises is higher. As both the average stack and the blind level is higher, and cannot be determined which of these features caused the model to raise higher.

The data presented here is based on 1'000 games. To look at the data presented it is important to keep in mind the blind structure presented in Table 3.1.

First the game will be analysed as a timeline where the x-axis is the hand. Figure 5.5 presents the frequency at which each action is taken. Figure 5.6 presents the average value to which the wager is raised, and the average stack of the agent. As the number of data points per hand is not huge, the data presented on the figures have been smoothed by a filter of size 7 and 5 respectively. On the edges the outer most value is repeated to fill the filter. The data after 200 hands is not presented as it is noisy. The noise is

there because there are fewer data points; most games have finished.

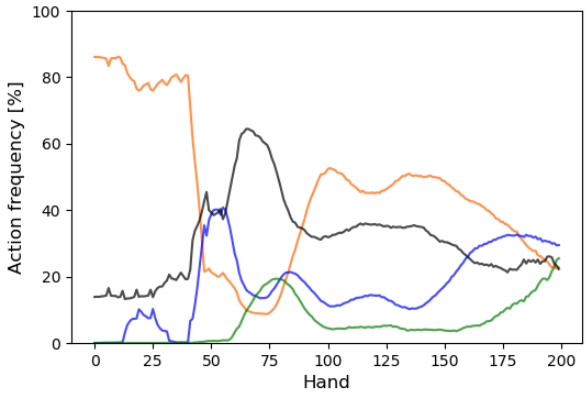

Figure 5.5: Fixed-agent: Action frequency

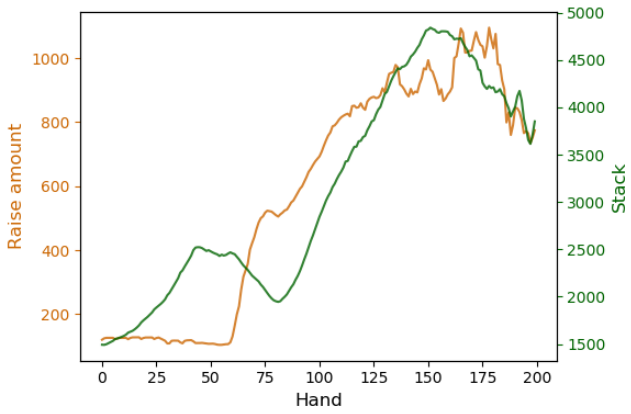

Figure 5.6: Fixed-agent: Raise amount and stack

The strategy of the player can roughly be split in three phases.

- From hand 0 to 50, the agent earns a lot of chips. The opponents more than often have more than 12 big blinds and therefore use their tighter strategy. The agent is raising about 80% of his hands and his raises are very small. This is a solid strategy to steal the blinds, which is efficient against tight players.
- From hand 50 to 80 the agent is losing on average. At this point the big blind level has increased to at least 100 and the antes start to kick in. Some of the opponent will have less than 12 big blinds and deploy their all-in or fold strategy. The agent raises about 20% of his hands. The size of his raises increases considerably

however during this phase, going from a 100 to 500. A hypothesis for this sudden change of strategy is that the agent is attempting to set the wager to the same amount as in the previous phase, however, since the blinds are lower he will be calling or folding instead of raising. In the meantime the stack of the agent decreases and the situation changes overall, therefore he moves to a new strategy with stronger raise values.

- From hand 80 to the end the agent plays an overall aggressive strategy. His average raise value increases and so does his stack. Towards the end he starts replacing raises by calls. In the meantime, the opponent Rulers also become more aggressive because their stacks becomes smaller with relation to the big blind.

The next three figures analyse whether the agent smartly selects the hands on which to raise. Figure 5.7 presents the raise frequency, depending on whether an opponent already raised on this street or not. As can be clearly observed the agent raises more often when no opponent raised before him.

Figure 5.8 presents the action frequency with relation to the adapted equity. No clear pattern can be spotted and it seems as if the fixed-agent does not take equity into account in his decision making.

Figure 5.9 presents the blind normalized action frequency with relation to the position at the table. The positions -5 and -6 are not presented as their data is too scarce. In order to reduce the noise from other factors as much as possible, this measure is performed in a specific setup. Only actions on an unraised preflop with at least 3 players remaining in the game are considered. The phase with 2 player is removed because the logic of the position swaps at this point: the dealer is first to speak on the preflop. The most noticeable aspects of this figure are the increased raise frequency on the big blind and the reduced raise frequency on the position before that. The differences are not high though; the fixed-agent only take slight advantage of his position.

Finally Figure 5.10 presents the action frequencies with relation to the street. The agent applies a different strategy on each street. Overall the aggression is higher after the flop. One thing to note is that the agent has very unbalanced action frequencies for the turn and the river which make it seems as if the strategy is less developed. This may be due to the fact that the later the street, the less it has been encountered. During training the agent could have neglected these streets. It may also be a strategy intentionally developed: the agent does fold a large % of hands when the opponent raises. If he reaches the river it usually means the opponent did not show aggression, thus he can place a raise with a very wide range.

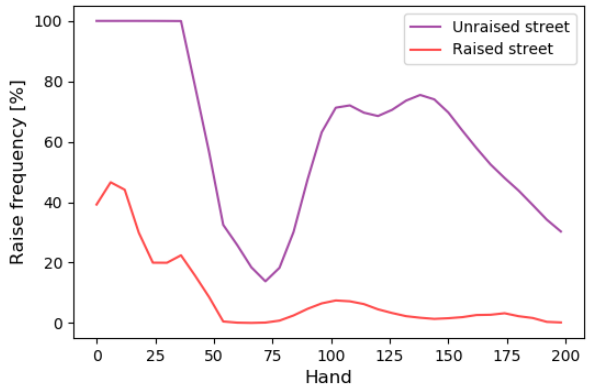

Figure 5.7: Fixed-agent: Raise frequency with relation to opponent action

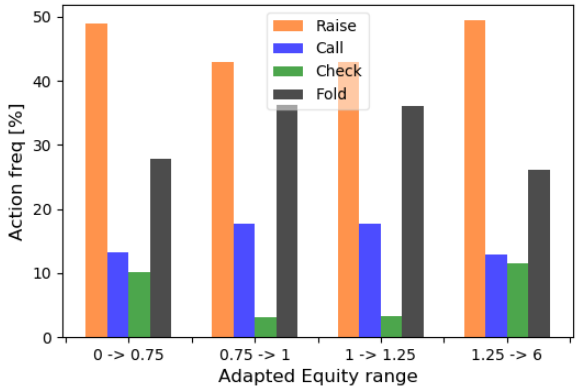

Figure 5.8: Fixed-agent: Action frequency with relation to equity

A summary of the analysis is given in Table 5.4. It presents whether an aspect of the game is taken into account. Each of the 6 columns corresponds to the insight retrieved from the 6 figures presented figures in this subsection. Note that the data may be presented in many more ways and underlying patterns can be missed. The most surprising observation from the strategy analysis is that the agent strongly beats the Ruler without noticeably taking his equity into account. This is possible against a tight player as he leaves a lot of blinds uncontested.

Agent	Action freq	Raise amount	First raise	Equity	Position	Postflop
Fixed-agent	Yes	Yes	Yes	No	Slight	Unclear

Table 5.4: Strategy analysis summary

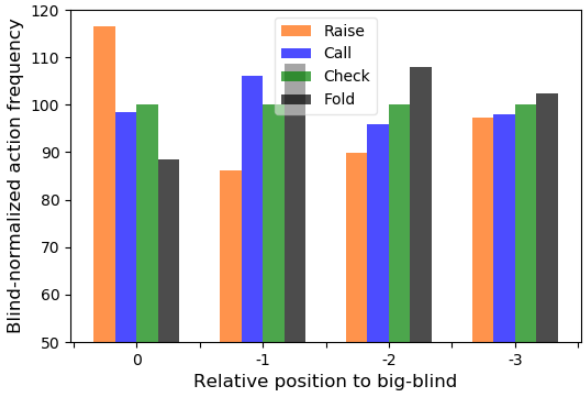

Figure 5.9: Fixed-agent: Action frequency with relation to position

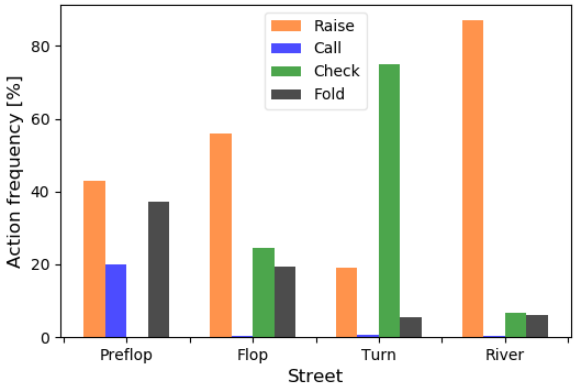

Figure 5.10: Fixed-agent: Action frequency with relation to street

5.4 6-Handed Sit and Go: poly-agent

In this experiment the agent is trained to play 6-Handed Sit and Go against different opponents. This version of the agent is referred to as the poly-agent because of its capacity of playing versus various opponents. The model used is the one described in subsection 4.1.3. The opponents that the agent will encounter are: the Caller, the Attached, the Maniac and the Rule-based bot. Four different table setups are played, each with one of the opponent being predominant. The tables are named according to their predominant strategy: Loose-Passive (LP), Tight-Passive (TP), Loose-Agressive (LA) and Tight-Agressive (TA). For the games to make sense, the tables never have a single opponent type. This is especially necessary for the loose opponents. For example 5 Maniacs at a table would go all-in in the first hand. The table compositions

are presented in Table 5.5.

Loose-Passive	Tight-Passive
- Caller	- Attached
- Caller	- Attached
- Caller	- Attached
- Attached	- Caller
- Ruler	- Ruler
Loose-Agressive	Tight-Agressive
- Maniac	- Ruler
- Maniac	- Ruler
- Maniac	- Ruler
- Conservative	- Caller
- Ruler	- Attached

Table 5.5: Table compositions

As this section presents the same figures as section 5.3, less details are given regarding the methodology.

5.4.1 Performance

The neuroevolution has 250 generations and a population size of 70 agents. During the execution phase, the agent plays 4 full matches at each table. This is equivalent to 96 games. The training thus requires the simulation of 1'680'000 games.

Figure 5.11 presents the average roi of the elites during training at each of the tables. The agent still appears to have room for learning at generation 250. Therefore, the training is continued until generation 300, bringing the mutation rate down to 0 and the mutation strength down to 0.02. The performance of the agents keeps increasing. However to keep consistency with the previous section, the agent at generation 250 is retained for analysis.

For validation the agent plays 1'000 full matches at each table, equivalent to 6'000 games per table composition. The performance measures are presented in Table 5.6. The density of the finishing rank is presented in Figure 5.12.

Measure	LP	TP	LA	TA
Average roi	232%	231%	134%	159%
STD match roi	65%	68%	98%	80%
Average rank	1.47	1.58	1.89	1.97

Table 5.6: Poly-agent: Performance

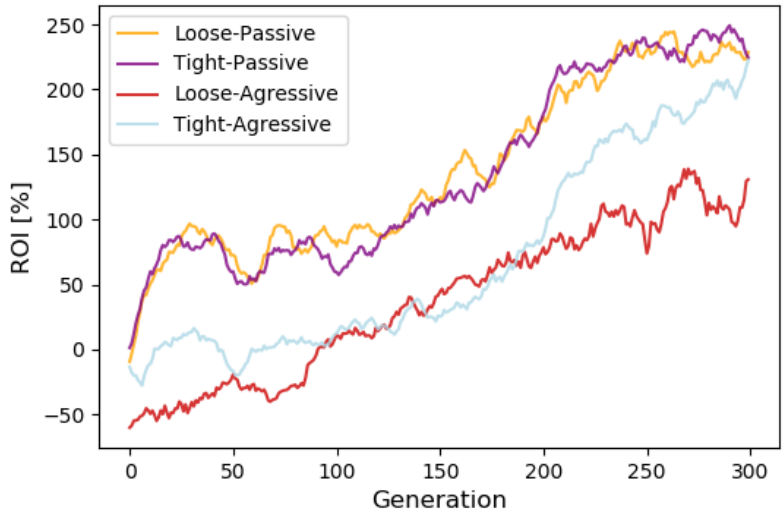

Figure 5.11: Poly-agent: average ROI of elites

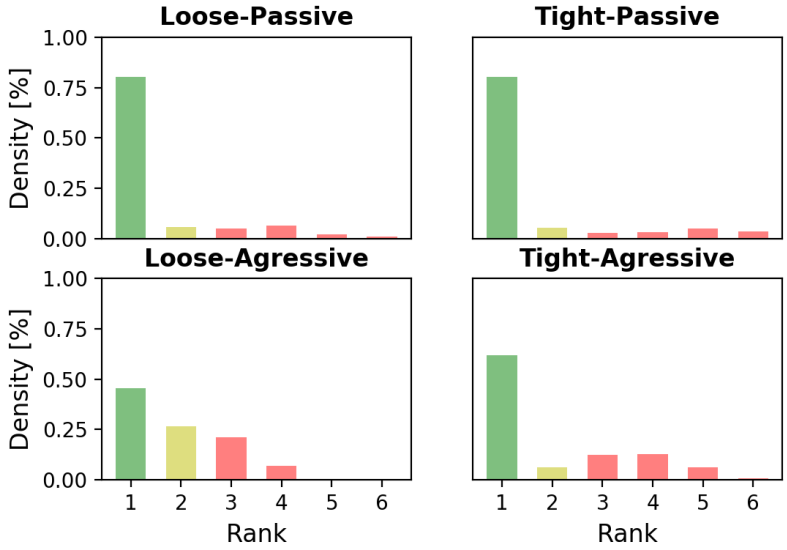

Figure 5.12: Poly-agent: Rank density

Overall the poly-agent has a great return on investment. Against the two passive table he gets first position on more than 75% of the games. It is against the Loose-Aggressive table that he struggles the most. This is not too surprising: against such a high aggression he is forced to play bigger pots and thus take risks.

5.4.2 Strategy analysis

The strategy of the poly-agent is now analyzed. The data presented here is based on 1000 games on each of the 4 tables. Figure 5.13 presents the frequency at which each action is taken. Figure 5.14 presents the average value to which the wager is raised, and the average stack of the agent.

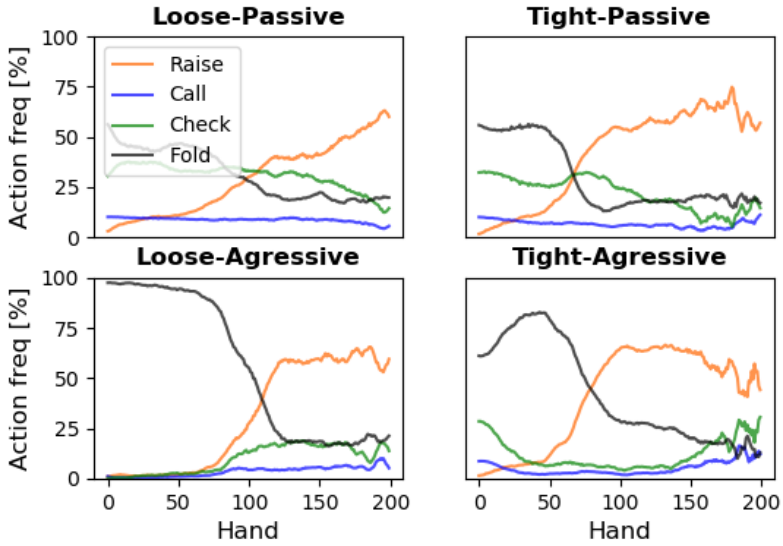

Figure 5.13: Poly-agent: Action frequency

The strategy of the agent can roughly be split in two phases.

- From hand 0 to 75, the agent is rather tight as he folds often. His raise amount however, is very high compared to the blind level. His raises in the first hands are often close to 1'000 though the big blind is at 20. At the Loose-Passive table the poly-agent directly plays more hands, and he increases his stack. At the Loose-Agressive table his fold frequency is close to 100%. This is the only table where he makes losses during this phase.

- From hand 75 to the end the agent develops a strategy that is more and more loose: At hand 125 his raise frequency is around 50%. During this whole phase the raise amount grows almost linearly. At the Tight-Agressive table is average raise amount continues to grow strongly until the end and reaches considerably higher values than at the other tables.

Figure 5.15 presents the raise frequency according to opponent activity. As was the case for the fixed-agent, the poly-agent raises more often when no opponent raised

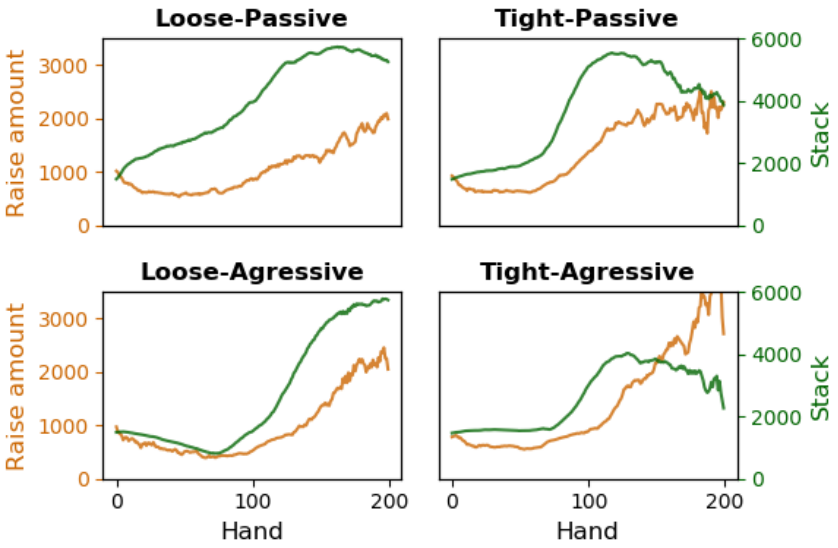

Figure 5.14: Poly-agent: Raise amount and stack

before him. At the Tight-Aggressive table the poly-agent takes huge advantage from this factor during the midgame. He raises up to 80% of the unraised streets while still raising only around on 20% of the raised streets.

Figure 5.16 presents the action frequency with relation to the adapted equity. A pattern is visible where hands with high equity are raised more often. This is a satisfying observation as the fixed-agent did not appear to take equity into account. The difference may be due to the fact that equity is more relevant against passive opponents. Figure 5.17 presents the blind normalized action frequency with relation to the position at the table. The setup is the same as in

section 5.3. The poly-agent takes advantage of his position. Indeed the agent raises more often on the two latest positions and much less often on the two positions after that. This is especially true at the Loose-Passive table.

Figure 5.18 presents the action frequencies with relation to the street. The clear observation is that the agent almost checks 100% of his hands postflop. Such a radical strategy highlights a lack of development of his postflop play. As most of his decision points are preflop, the learning process may have neglected the other streets. Checking is more than often chosen when the agent wants to fold but is allowed to check. In other words the agent does not want to pursue the hand. In addition, it implies he is not facing aggression. As the poly-agent calls very few hands preflop, a vast majority of his postflop play is indeed against passive opponents. This is nevertheless a counterintuitive

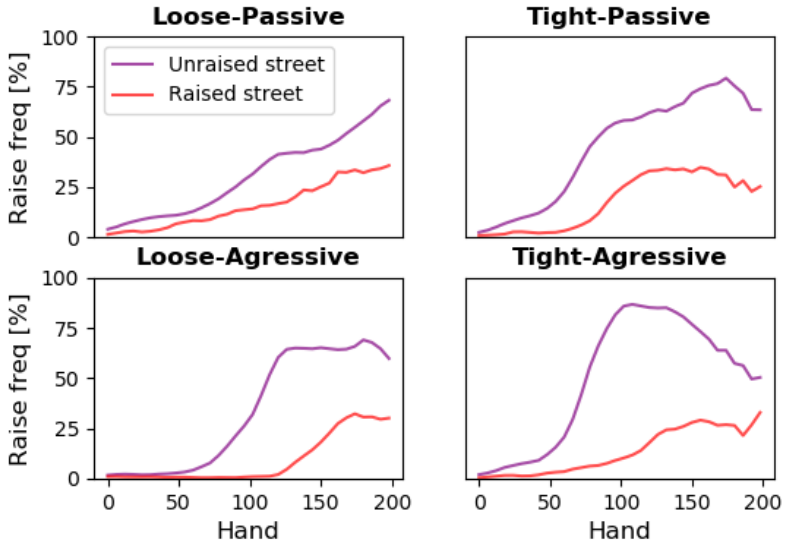

Figure 5.15: Poly-agent: Raise frequency with relation to opponent action

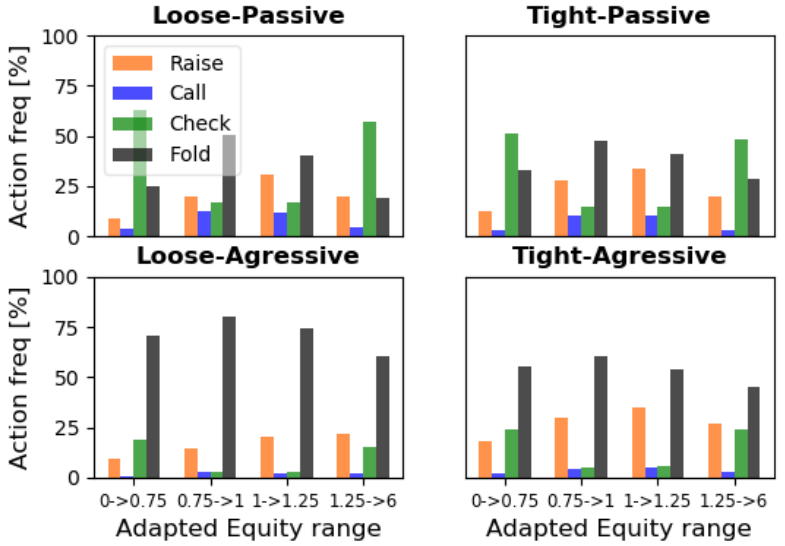

Figure 5.16: Poly-agent: Action frequency with relation to equity

strategy: if playing against an opponent that calls often, one would want to play more often on the postflop to take advantage of the added information. However, as the poly-agent already exploits the passive opponents almost to the maximum, he is not incentivized to learn any further.

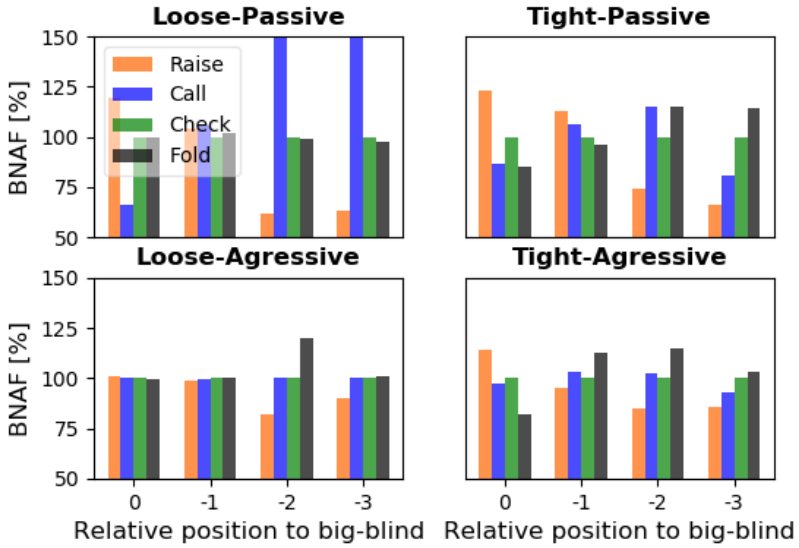

Figure 5.17: Poly-agent: Action frequency with relation to position

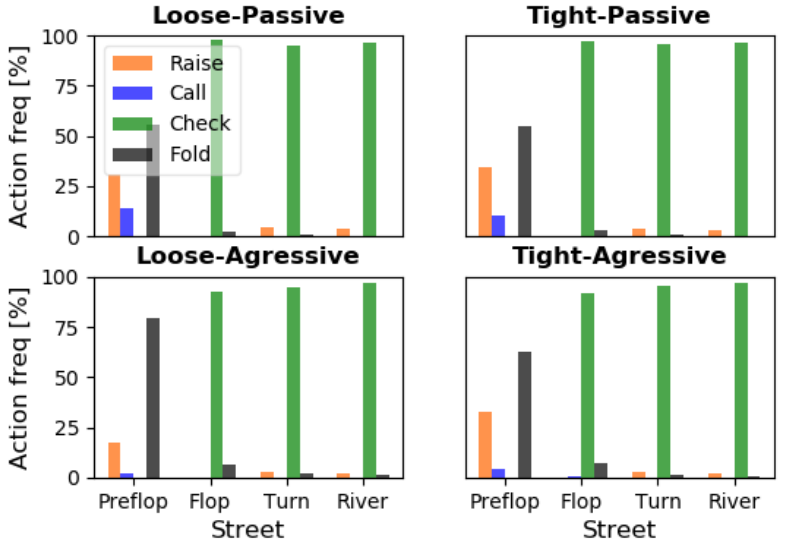

Figure 5.18: Poly-agent: Action frequency with relation to street

A summary of the analysis is given in Table 5.4. The main observation is that the agent does not control his raise amount very well and appears to have an underdeveloped postflop play. However he takes good advantage of his equity and position.

Regarding opponent modelling: the agent plays a different game at each of the 4

tables. It is however difficult to measure how much of that difference is related to opponent modeling and how much of that difference is related to the static part of the model.

Agent	Action freq	Raise amount	First raise	Equity	Position	Postflop
Poly-agent	Yes	Half	Yes	Yes	Yes	No

Table 5.7: Poly-agent: Analysis summary

6

Conclusions

In this report an agent to play 6 Handed Sit and Go poker against various weak opponent is presented. The performance is on average extremely high at all the tables played. This validates the quality of the model for the task it was presented.

The fixed-agent, that trained and played only against the rule-based bot, has an average return on investment higher than 200%. It shows that the strategy of the rule-based bot presented can very strongly be exploited. This is achieved over the course of a Sit and Go lasting only one or two hundred hands without making noticeable use of the equity. With the study on the in-game data, a human player may have a good idea of how to effectively exploit such a strategy as well. The fixed-agent can however not be generalised to other opponents: his strategy, though reasonable, is aimed at exploiting the rule-based bot only.

The poly-agent, playing against multiple table compositions, also achieved a very high return on investment. The table where the lowest roi is achieved is the Loose-Agressive table, with approximately 130%. On the Tight-Agressive table is has an roi around 160%. On the Loose-Passive table and the Tight-Passive table it has an roi around 230%. The in-game analysis shows that the agent plays differently at different tables. However it is not clear whether the agent does so thanks to the opponent modeling or because he is confronted to different situations. When confronting the strategy developed to expert knowledge, it appears the poly-agent plays wrong on a couple of aspects. Mainly the raise are sized way too strongly in early game and the postflop play is not developed. An important note is that the agent almost exploits the tables to the maximum. Thus it is not incentivized to develop a smarter strategy. This highlights how the quality of the strategy is bounded to some extent by the opponents faced. To further validate the potential of the model it has to confront

stronger opponents.

This report leaves a lot of potential for future work. Multiple valuable experiments can still be performed on this model. One would be to make the opponent validate against an opponent it has never met before. Another one would be to set up a special testing environment to measure how much each part of the network independently influences the final decision. A couple of changes may also make the model stronger. One is to increase the number of features fed to the opponent network. It would be valuable for the model to have more information about the state in which the opponents find themselves. Also, for a given a hand, information about the opponent should be obtained even before he has played. Finally, the output of the network could have more dimension than one. This would help the model to select specific action such as calling or minimum raising.

References

[1] N. Bard. **The Annual Poker Competition webpage**. Technical report, 2010. 2

[2] Michael Johanson. **Measuring the Size of Large No-Limit Poker Games**. Technical Report TR13-01, Department of Computing Science, University of Alberta, 2013. 2

[3] Oskari Tammelin, Neil Burch, Michael Johanson, and Michael Bowling. **Solving Heads-up Limit Texas Hold'Em**. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 645–652. AAAI Press, 2015. 3

[4] Oskari Tammelin. **Solving Large Imperfect Information Games Using CFR+**. *CoRR*, **abs/1407.5042**, 2014. 3

[5] Noam Brown and Tuomas Sandholm. **Superhuman AI for heads-up no-limit poker: Libratus beats top professionals**. *Science*, **359**(6374):418–424, 2018. 3

[6] Matej Moravcík, Martin Schmid, Neil Burch, Viliam Lisý, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael H. Bowling. **DeepStack: Expert-Level Artificial Intelligence in No-Limit Poker**. *CoRR*, **abs/1701.01724**, 2017. 3

[7] Finnegan Southey, Michael Bowling, Bryce Larson, Carmelo Piccione, Neil Burch, Darse Billings, and D. Chris Rayner. **Bayes' Bluff: Opponent Modelling in Poker**. *CoRR*, **abs/1207.1411**, 2012. 4

[8] Marc J. V. Ponsen, Jan Ramon, Tom Croonenborghs, Kurt Driessens, and Karl Tuyls. **Bayes-Relational Learning of Opponent Models from Incomplete Information in No-Limit Poker. 3**, pages 1485–1486, 01 2008. 4

[9] Aaron Davidson, Darse Billings, Jonathan Schaeffer, and Duane Szafron. **Improved Opponent Modeling in Poker**. pages 493–499. AAAI Press, 2000. 4

[10] Garrett Nicolai and Robert J. Hilderman. **Algorithms for Evolving No-Limit Texas Hold'em Poker Playing Agents.** pages 20–32, 09 2010. 4

[11] Xun Li and Risto Miikkulainen. **Evolving Adaptive Poker Players for Effective Opponent Exploitation**. In *AAAI-17 Workshop on Computer Poker and Imperfect Information Games*, San Francisco, CA, USA, 2017. 4, 16, 22

[12] PokerStrategy.com. **Sit and Go beginner strategy**. 10