Model Creation for Machine Learning in the Imperfect-Information Game Schnapsen

Project Intelligent Systems [*]

Abstract

In the imperfect-information game Schnapsen, machine learning poses a promising approach to finding an optimal game strategy. This paper focuses on the performance of the corresponding machine learning models. Our research is based on the idea that an expert-level bot can play more exceptional moves against a weaker opponent than against itself. To investigate this, a method for model creation is used in which a bot does not train on itself, but on opponents of varying strength. Different tournament approaches are then used to compare the performance of the resulting models in Schnapsen. According to our research, the machine learning models prove to be stronger when an expert-level bot trains on itself.

Keywords: schnapsen, machine learning, imperfect information

[*]Vrije Universiteit Amsterdam

1 Introduction

1.1 Strategic Challenges in Card Games

Card games have been around in Europe since the 14th century [1] and have managed to remain relevant throughout history as strategic challenges, as well as entertainment. A more modern approach to the strategic challenge that certain card games pose, is using machine learning and artificial intelligence in search for an optimally playing strategy.

Algorithms superior to human experts have already been found for a variety of card games, such as poker [6] or bridge [2]. But despite recent breakthroughs in multiple card games, a perfect winning strategy is hard to find. We have been introduced to a variety of applicable strategies during the Intelligent Systems course at the VU. The Intelligent Systems course used the trick-taking card game Schnapsen (see section 2.1) as an example to illustrate several strategies for playing the game, whether the strategy was effective at winning or not.

Schnapsen consists of two phases. During its first phase, Schnapsen forms a challenge as it is an imperfect information game. Players are “imperfectly informed” about how the cards were arranged in the opponent’s hand as well as the talon. In the second phase, both players have perfect information as all cards have been played and redistributed.

1.2 Researching Machine Learning through Schnapsen

As previously mentioned, we analyzed several strategies for playing Schnapsen in the Intelligent Systems course[1] and were allowed to use the framework for the bots (see section 2.2 for further explanation) provided in this course.

We were interested in what the model performance might be if an expert-level, for instance Rdeep-bot, was playing against amateur-opponents such as Rand-bot, since it might make even stronger moves compared to when it played against itself. Therefore our research question is:

Is it possible to significantly improve performance of a machine learning model when trained on expert-vs-amateur games compared to expert-vs-expert games?

By an "amateur-level opponent" we mean an opponent that either uses a strategy mainly based on chance (choosing a random card out of all legal moves) or having an inadequate strategy (see win-ratios compared to Rdeep-bot in appendix 8.4) compared to bots that are of "expert-level" (like the Rdeep-bot). "Performance" will be decided on how many points a player scores or how many wins a player gains over *i* number of games (depending on the experiment).

*Vrije Universiteit Amsterdam

[1]the course was held in November/December 2020 at the Vrije Universiteit Amsterdam

Hypothesis H1 is our primary hypothesis, hypothesis H0 is the null-hypothesis (which is applied in case no significant results are found).

H1: Performance of the machine learning model significantly improves when trained on expert-vs-amateur games compared to expert-vs-expert games in the game of Schnapsen

H0: Performance of the machine learning model does not significantly improve when trained on expert-vs-amateur games compared to expert-vs-expert games in the game of Schnapsen

By using several bots to create different models for the Machine-Learning-bot, we were able to compare different strategies of learning and their effectiveness on winning points. We edited the code of the model creation in such a way that the bot is not trained on itself (e.g. Rand-bot vs Rand-bot), but instead we could integrate a second player so we were able to create models based on games of opponents with varying strength (see section 2.2 for explanation of the mentioned bots).

2 Background Information

2.1 The Basics of Schnapsen

In this paper, we will make use of the game Schnapsen to analyze different bots. Schnapsen is a card game which is based on winning tricks to receive points. The game is played with 20 cards: the ace, ten, king, queen and jack of each suit (clubs, diamonds, hearts and spades). Both players start with five cards in hand and draw a card after each trick, until the talon is exhausted. When the talon is exhausted, the second phase of the game begins and players have to follow suit (as opposed to the first phase).

The cards have the following number of points assigned to them:

- Ace 11 points
- Ten 10 points
- King 4 points
- Queen 3 points
- Jack 2 points

After dealing cards to both players, a card from the talon is turned around. This is the card that determines the trump suit. Cards with a trump suit outrank all other cards: if a trump suit card is played, it can only be beaten by a trump suit card that has a higher value. When a trick is won, the sum of both cards (of the player and the opponent) is counted towards the winner's score. A marriage (having a king and queen of the same suit) is worth

20 trick-points and if it is a trump-marriage then it is worth 40 trick-points. To win a round, a player must reach 66 trick-points in the game, after which they can declare “66”. As soon as a player declares “66”, the game ends and the trick-points of both players are counted.

The player that declared "66" must also have 66 trick-points. If that is true and the loser has fewer than 33 trick-points, the winner gains two game-points. If the loser has 33 trick-points or more, the winner gains one game-point. In the case that the losing player has not won a single trick throughout the entire game, the winner gains three game-points. A more detailed explanation of the rules can be found online.[4]

2.2 Bots and their Strategies

In this section we will describe the strategy the different bots use in order to determine which card to play. The framework and virtual environment for the bots were provided by the Intelligent Systems course at the VU Amsterdam. The code for the bots Rand-bot, Bully-bot, and Rdeep-bot can be found in the Schnapsen directory in the appendix 8.1 and the code for Trump-bot and Lowest-Card-bot can be found in the appendix 8.7. For clarification, we will briefly describe the strategy each bot uses for playing a move.

Rand-bot checks for all legal moves and chooses one of them at random.
Bully-bot checks for all legal moves and chooses one of them at random (just like Rand-bot), but with three added conditions: if the bot has a trump card on hand, it will be played first. If the opponent has already played a card (and the player has no trump cards), then the bot will play a card of the same suit when possible. If that is not the case, it plays the highest value card.
Rdeep-bot analyzes random extensions of the game tree using Perfect Information Monte Carlo Sampling (PIMCS) [8] to determine a heuristic for each possible move. It chooses to play the move with the best heuristic.
Trump-bot differentiates between the case that it leads and the case that the opponent leads. When Trump-bot leads, the trump cards are excluded from the hand and the highest card is played. If the opponent leads, the bot will follow suit and play a higher card (when available). If Trump-bot has a trump card available while the opponent has already played a card, then it will play the trump card. If the bot does not have a trump card available, it will play the lowest value card.
Lowest-Card-bot plays a card with the lowest available rank.
Machine-Learning-model stores a data-structure that is trained on the observation of games. All the moves made in those training games are used to teach a pattern to a deep neural network. The model can then be utilized by a **Machine-Learning-Bot** which uses the trained deep neural network. The latter recognizes certain patterns in the possible moves and returns the move that to its knowledge will likely lead to a win. Machine learning is used because the patterns might be too complex to recognise for a human programmer [7], but can still be picked up by a machine learning agent.

3 Machine Learning in Imperfect Information Games

3.1 Foundations for our Research

Within the realm of the adversarial imperfect-information game Schnapsen, researchers have made a variety of approaches aiming to produce an optimally playing algorithm. Such an algorithm would always make the best possible decision for the game state it is in.

Certain card distributions in Schnapsen (eg. only having the lowest possible cards on hand), it is impossible to win even with a perfect strategy. Therefore, the randomness involved with card-dealing makes it infeasible to create a bot that legally wins 100% of the games[5]. However in theory, it is feasible to create an optimally playing algorithm which would not be able to win every game, but it would win every game which is possible to win.

The academic prospects of games like Schnapsen include the approach of using machine learning algorithms. The latter base their moves upon a so-called "Machine-Learning-model" which stores a specified number of training games including all their respective moves as well as their final outcome (win/loss). After training, a Machine-Learning-bot (see section 2.2 for explanation) can use the created model in order to return moves that were similar to beneficial moves of the training games. Thus, the Machine-Learning-bot's performance depends heavily on the quality of the training during model-creation. It should be kept in mind that after training, the model is not being updated after each game and therefore, does not continue to learn while playing games.

Three factors play a main role while creating the training model:

1. The quality and combination of bots used in training (p1,p2)
2. The number of training games (n)
3. Whether one or both players are observed ([1,0] / [1,1])

In our research, we will focus on the first factor, the bots. In state-of-the-art machine learning for imperfect-information games, it is common to create models in which a bot plays against itself [3]. In our approach however, we want to look into different bot combinations for playing training games used for model creation. As explained in section 2.2, we already had bots and the virtual environment to be able to create some unusual combinations for training the model.
In Figure 1 the results of five tournaments between Rdeep-bot and Machine-Learning-bot using different models are shown. All models were trained on themselves, which account for the differences in performance. Performance hereby is defined by the number of points won during a typical tournament of 1000 games. The points won are also an indicator of the performance of the bots during a single game. Still, it has to be kept in mind that player performance in Schnapsen heavily relies on the random card distribution.

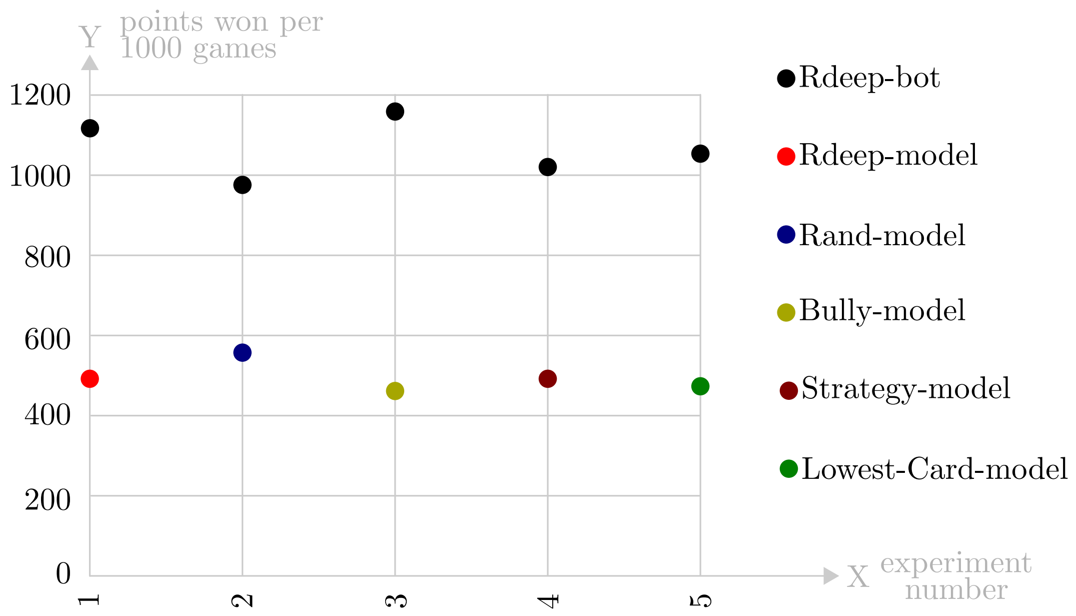

FIGURE 1: Games won by different models competing against Rdeep-bot. See appendix 8.5

Figure 1 shows how the Machine-Learning-bot's performance correlates with the strength of the bots used in model creation. As visible in the diagram, with the low amount of training games in the model (which is at 2000 for the tournament above) the Machine-Learning-bot performs poorly against Rdeep-bot no matter which model it uses.

Our approach consists of modifying the bot combination in model creation in order to improve performance without increasing training games. We used an expert and an amateur bot in model creation. This was due to the idea that with this combination, the model may afterwards 'take the best out of both worlds'. Respectively, our theory was that the model would mostly consist of good moves by the strong bot, because a good move has a stronger effect when playing against a weaker opponent. In turn, the amateur bot may by coincidence produce very few exceptional and unusual moves that would also make the model better. The Machine-Learning-bot tries to extract the most promising moves out of each prior training game situation that fits the current game state. Our hopes were that the Machine-Learning-bot would simply disregard the weak moves of the amateur bot, since they would be of no benefit to the bot. Namely, is it possible to significantly improve performance of a Machine-Learning-model when trained on expert-vs-amateur games compared to expert-vs-expert games?

4 Experiment 1

Though machine learning is often conducted on the basis of a number of games where one player performs moves against itself, our research question focuses on two different players playing the training games. During training, the features of each move are appended to a vector before the bot-function is called to move. In order to use two different bots during training, we have to alternate between the two bots instead of calling the same bot each iteration. The altered code can be seen in the appendix 8.6.

As described in section 1.2, the Rdeep-bot is seen as an expert-level bot. Therefore, we make it the "default" choice for player1 for the training games. To train the needed Machine-Learning-models, the opponent player2 was varied throughout training. In order not to create a confounding variable, our Machine-Learning-models were created in 1.000 through 10.000 training games with intervals of 1.000 games. Three different bots played against Rdeep-bot: Rand-, Bully- and Trump-bot, which were all described in section 2.2. Also, Rdeep-bot was trained on itself. A total of 40 models were created.

Further, a tournament implementation was created to eliminate the element of chance. Originally, every game played had a random game-ID[2]. This lead to the performance of bots varying significantly between the same tournaments, thereby polluting the experimental data. To avoid this, each game was hosted twice with both players alternating perspectives. Every tournament conducted during experimentation, ran the first game with the game-ID 100000. After repeating the game with players switching sides, the game-ID was set to 100001. This loop was repeated, until 1000 games had been played, setting the last game-ID to 100499. Therefore, every fair-tournament-implementation was played with the same amount and identical set of games.

4.1 Testing

The 40 created Machine-Learning-models were tested for performance (see section 3.1). This was done by using what we call "competitions", where games were rolled out using an all-versus-all pattern. The following table shows the setup of such a competition.

	Rand-model	Bully-model	Strategy-model	Rdeep-model
Rand-model		Rand vs Bully	Rand vs Strategy	Rand vs Rdeep
Bully-model	Bully vs Rand		Bully vs Strategy	Bully vs Rdeep
Strategy-model	Strategy vs Rand	Strategy vs Bully		Strategy vs Rdeep
Rdeep-model	Rdeep vs Rand	Rdeep vs Bully	Rdeep vs Strategy	

[2]A game-ID is unique and defines the exact distribution of cards of both players as well as the stack.

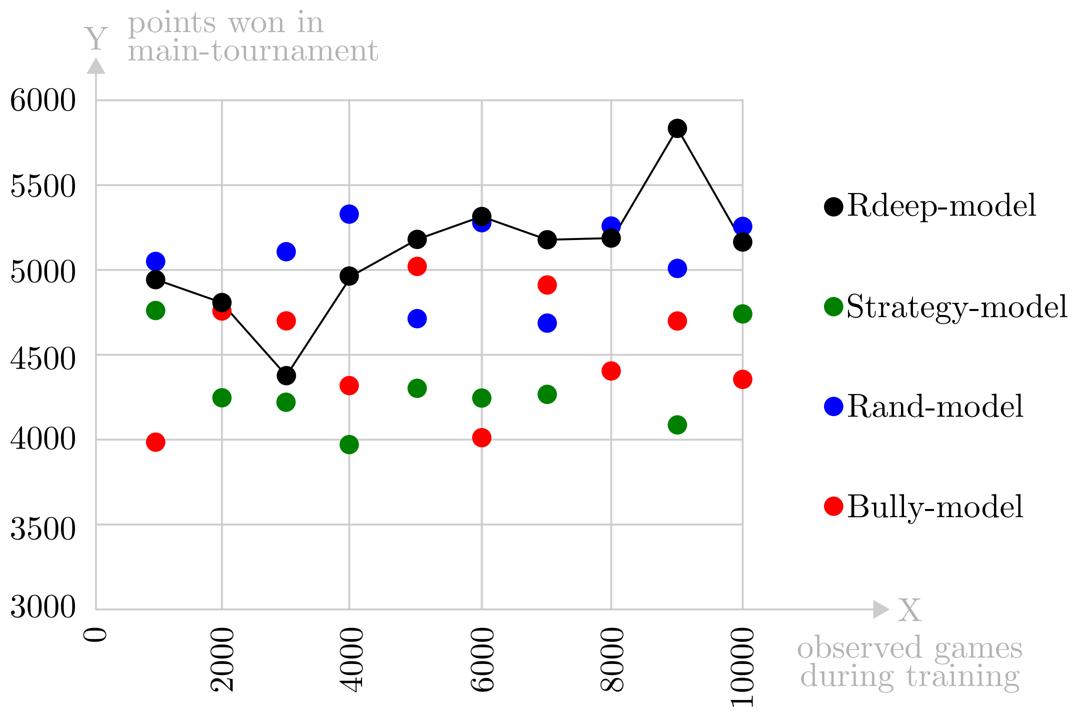

FIGURE 2: Games won by different models competing against Rdeep-bot. See appendix 8.2

4.2 Results

The goal of the competitions of experiment 1 was to compare game performance between different Machine-Learning-models. We performed 10 competitions to correlate the performance of models that were trained on the same amount of games. The comparison was realized by summing up the total game points that were won by each of the Machine-Learning-models. The results are depicted in Figure 2. Each data-point on the chart represents the total game points won (Y-axis) by a model of the corresponding color. The X-axis shows the number of training games of the four compared models, thereby representing each conducted competition. In contrast to the other data-points, Rdeep-bot is connected through a line that depicts the strength of this expert-vs-expert model.

The number of points won by the Rdeep-model (Rdeep-bot vs Rdeep-bot) in experiment 1 indicate that it is equally strong if not better than the other models trained on expert-vs-amateur games. This argues against our hypothesis H1 of an expert playing against an amateur-level opponent improving in performance compared to playing against itself.

We used a one-tailed independent T-test to statistically validate the results of experiment 1. The overall performance of the Rdeep-model was compared to all other models. Comparing the 10 Rand-models (M = 5038.7, SD = 253.1)[3] to the 10 Rdeep-models (M = 5099.5, SD = 355.6) resulted in no relevant difference in performance, t(18)[4] = 0.43, p[5] = 0.34,

[3]SD = standard deviation, M = median of all data points
[4]t(x) = t-test(degrees of freedom)
[5]p = probability that the results occurred by chance

despite the Rdeep-models earning more points overall. Meanwhile, the 10 Bully-models (M = 4517.3, SD = 343.1) performed significantly worse than the Rdeep-models, t(18) = 3.53, p = 0.001. There was also a significant difference in performance, t(18) = 5.4, p = .00002, between the 10 Strategy-models (M = 4325.2, SD = 241.1) and the Rdeep-models. Due to the fact that training on expert-vs-amateur games does not lead to an increase in performance compared to observing expert-vs-expert games, but rather the opposite, we can conclude that our hypothesis H1 is to be rejected.

5 Experiment 2

The second experiment was conducted because Rand- and Bully-bot work fundamentally differently (see section 2.2). We decided to conduct experiment 2 without Rand-bot to exclude the factor of chance to gain more reliable data.

Rdeep-bot was observed playing against itself, Strategy-, Bully- and Lowest-Card-bot (see section 2.2). For experiment 2, all of the compared bots had carefully chosen strategies with no element of chance. All of the models were again trained on 1000 to 10000 games with 1000-intervals, which resulted in 40 new models.

5.1 Testing

Not using cross-testing on models of different competitions might lead to a confounding variable. During testing of experiment 2, this was avoided by playing every single model against the Rdeep-bot as an expert-level player. By having the same opponent each time, the results would be comparable across all groups. We applied the fair-tournament-implementation using the same set of game-IDs as in experiment 1 but this time measuring won games instead of won points.

It was interesting to see that in experiment 1 the Machine-Learning-model trained on Rdeep-bot vs Rand-bot (expert-vs-amateur) was outperforming the Machine-Learning-bot trained with both Bully- and Trump-bot. Instead of Rand-bot, we created a third strategy bot, the Lowest-Card-bot as described in chapter 2.2. Geared towards ranking the three bots by their performance, we individually tested their won games against the normal Rdeep-bot in a fair-tournament-implementation (same game-IDs; switching perspectives) of 10,000 games.

1. Trump-bot: 22.8% win-ratio
2. Bully-bot: 14.1% win-ratio
3. Lowest-Card-bot: 12.0% win-ratio

See appendix 8.4 for a more detailed overview.

5.2 Results

In contrast to experiment 1, we simplified the process of comparing the trained models. Instead of playing the models against each other, we tested their individual performance against the Rdeep-bot. This resulted in fewer testing steps as well as the possibility to compare performance across different numbers of training-games. There were two options we could use to test performance in Schnapsen: counting won points or won games. In order to use a binomial test, which only allows for dichotomous response variables (eg. win/loss), we now counted wins instead of points like in experiment 1.

The results of experiment 2 are depicted in Figure 3. Similarly to the chart of experiment 1, the data-points of the 10 Rdeep-models are again connected for an easier visual differentiation of the expert-vs-expert model and the three expert-vs-amateur models of Strategy-, Bully- and Lowest-Card-bot. All data-points show the number of wins (Y-axis) that were won by the model of the corresponding color. The X-axis differentiates the models by the amount of training games they observed.

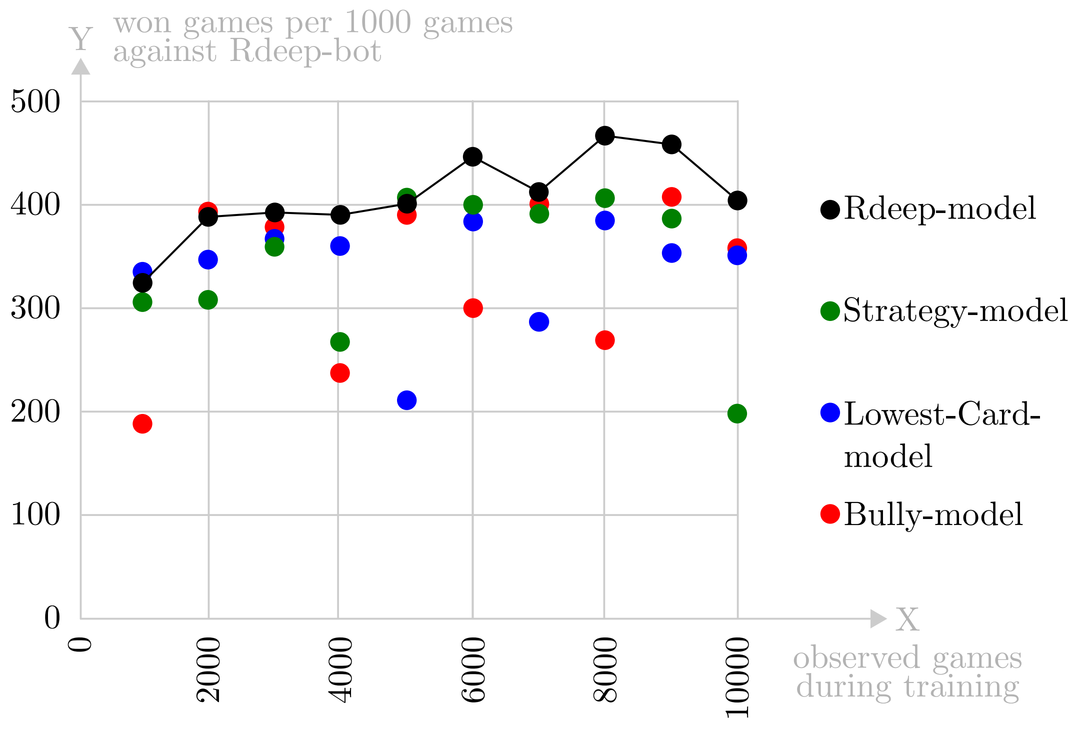

FIGURE 3: Won games by different models competing against the Rdeep-bot. See appendix 8.3

In contrast to experiment 1, there are a considerable amount of outliers [6] in the testing data. It is interesting to note that all of the strategy-bots[7] have such outliers differing significantly from the main distribution pattern while Rdeep-model does not:

[6]An outlier is a data point that varies abnormally from other values

[7]We define strategy-bots as bots that follow a set strategy with no element of chance.

Standard deviation over the 10 models of a bot: SD = $\sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$
where SD = population standard deviation, N = the size of the population, x_i = each value from the population, mu = population mean

- SD(Rdeep-models) = 39.2
- SD(Lowest-Card-models) = 49.9
- SD(Trump-models) = 67.2
- SD(Bully-models) = 74.4

We used a binomial test to statistically validate the results of experiment 2. The total summed points of the Rdeep-models were compared to the best-performing strategy bot models, which were the ones of Trump-bot. The following formula validates our results, because it calculates the low likelihood by which they might have occurred by chance. P(0.5) = $\sum_{k=3428}^{7517}\binom{7517}{k}\cdot p^k q^{7517-k} < 0.00001$

The result lies far below the typical 5% significance-level. The best strategy-bot model performs significantly worse than the Rdeep-model. We can therefore conclude that all of our strategy-bot models (expert-vs-amateur) are weaker than the expert-vs-expert Rdeep-model. Again, this is an indication against our hypothesis H1.

6 Findings

Both of our experiments have indicated that there is no evidence supporting our hypothesis H1. It seems to be the case that training on an amateur-opponent in model creation does not significantly improve the model. Instead it might be worse than if the model had been created by self-play of the stronger bot. This has been the case for nearly all of our model combinations. The only data supporting our hypothesis is that occasionally, the combination of Rdeep-bot and Rand-bot for the model outperforms a model of Rdeep-bot and Rdeep-bot (see Figure 3). As this is out of line with our other results, we came up with a possible explanation.

As Rdeep-bot's strategy features the sampling of various random game trees originating from the current move and picking the one with the most promising heuristic, it correlates with Rand-bot's strategy. Because Rdeep-bot models an opponent that plays exactly like Rand-bot, this may influence the training during model creation. However, despite the outliers of Rand-bot, our results were consistent and statistically valid. One confounding variable may also be the implementation of the provided bots (see appendix 8.1)

6.1 Future Work

In future research, we would like to go more in-depth as to why Rand-bot might outperform Rdeep-bot sometimes and how often this happens for tournaments of respectively 3000 and

4000 games. We are unsure as to whether this effect occurred by chance, if the experiment was not set up well enough despite careful checking and including several methods to ensure statistical validity, or if there is an underlying explanation for this that is yet to be discovered.

Also, as seen in section 3.1, Rdeep-bot won about twice as many points than the Machine-Learning-bot using a model which was based on training games in which Rdeep-bot played against itself. This is due to the fact that the number of training games in model creation were comparatively low (2000 games), which decreases the quality of the model. It might be of interest for future research if the performance of a Machine-Learning-bot using a model where Rdeep-bot plays against itself gets closer to reaching Rdeep's level of performance as the number of training games approach infinity.

Additionally, we encountered the very interesting effect that a Machine-Learning-model increases in performance if only one of the two players is observed during model creation compared to observing both. This is still completely counter-intuitive to us and also poses an appealing subject for future research.

7 Conclusion

No significant difference in strength was found between training on an amateur-opponent and training on an expert-level opponent in the game of Schnapsen. Machine learning was implemented in such a way that we were able to train a bot on a different bot (instead of the bot performing moves against itself). We conducted two different experiments to test our H1 hypothesis. Four different bots were trained on the expert-level Rdeep-bot in experiment 1: Bully-bot, Rand-bot, Trump-bot and the Rdeep-bot also trained on itself. 40 Machine-Learning-models were created to compare overall performance of the expert-level Rdeep-bot and the other bots. By using a one-tailed independent T-test we were able to statistically validate our results for experiment 1 and we must conclude that no significant increase in performance has been found between training on an expert-level compared to training on an amateur-level opponent.

To exclude the possibility of a factor of chance interfering with the data, we decided to exclude Rand-bot from experiment 2. Instead of Rand-bot we created a new bot called Lowest-Card-bot, which always plays the lowest card available. By using a binomial test we were able to statistically validate our results for experiment 2, which were far below the standard 5% significance level. Therefore, the results of experiment 2 indicate that no significant result was found to support our H1 hypothesis. The game-ID's were created and kept the same for each round, after the round was finished the players switched perspectives to make the tournaments fair and exclude any confounding variables. Therefore, we reject the H1 hypothesis and conclude that the H0 hypothesis is likely to be correct, since it seems that the performance of the machine learning model does not significantly improve when trained on expert-vs-amateur games compared to expert-vs-expert games in the game of Schnapsen.

References

[1] Michael Dummett. “The history of card games”. In: *European Review* 1.2 (1993), pp. 125–135.

[2] Matthew L. Ginsberg. “GIB: Imperfect information in a computationally challenging game”. In: *Journal of Artificial Intelligence Research* 14 (2001), pp. 303–358.

[3] Johannes Heinrich and David Silver. “Deep reinforcement learning from self-play in imperfect-information games”. In: *arXiv preprint arXiv:1603.01121* (2016).

[4] John McLeod Keith Waclena. *Schnapsen*. URL: https://www.pagat.com/marriage/schnaps.html.

[5] Jeffrey Long et al. “Understanding the success of perfect information monte carlo sampling in game tree search”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 24. 1. 2010.

[6] Matej Moravčık et al. “Deepstack: Expert-level artificial intelligence in heads-up no-limit poker”. In: *Science* 356.6337 (2017), pp. 508–513.

[7] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[8] Florian Wisser. “An expert-level card playing agent based on a variant of perfect information Monte Carlo sampling”. In: *Twenty-Fourth International Joint Conference on Artificial Intelligence*. 2015.

8 Appendix

8.1 Schnapsen directory

https://github.com/ursulean/schnapsen

8.2 Data of Experiment 1

The following 10 tables display the results of all Competitions. The top left number depicts how many training games were used to train the models. Points are displayed using the following system: row-player-points - coloumn-player-points

1000	Rand-model	Bully-model	Strategy-model	Rdeep-model	**points**
Rand-model		947 - 618	947 - 618	763 - 777	5043
Bully-model	659 - 966		655 - 886	701 - 879	3980
Strategy-model	732 - 814	908 - 677		700 - 815	4761
Rdeep-model	769 - 798	930 - 670	785 - 758		4953

2000	Rand-model	Bully-model	Strategy-model	Rdeep-model	**points**
Rand-model		786 - 772	822 - 705	785 - 794	4708
Bully-model	768 - 780		809 - 711	792 - 796	4762
Strategy-model	710 - 794	699 - 801		722 - 840	4249
Rdeep-model	825 - 741	745 - 820	822 - 702		4822

3000	Rand-model	Bully-model	Strategy-model	Rdeep-model	**points**
Rand-model		782 - 699	849 - 655	919 - 712	5105
Bully-model	687 - 828		771 - 696	882 - 725	4702
Strategy-model	702 - 791	699 - 777		682 - 820	4221
Rdeep-model	659 - 936	714 - 886	745 - 787		4375

4000	Rand-model	Bully-model	Strategy-model	Rdeep-model	**points**
Rand-model		901 - 640	900 - 654	861 - 777	5325
Bully-model	662 - 904		775 - 715	715 - 825	4316
Strategy-model	638 - 886	674 - 792		663 - 847	3967
Rdeep-model	768 - 873	799 - 735	716 - 623		4959

5000	Rand-model	Bully-model	Strategy-model	Rdeep-model	**points**
Rand-model		734 - 867	825 - 783	806 - 882	4719
Bully-model	884 - 709		895 - 660	806 - 797	5026
Strategy-model	761 - 843	648 - 820		760 - 889	4309
Rdeep-model	894 - 802	806 - 754	919 - 697		5187

6000	Rand-model	Bully-model	Strategy-model	Rdeep-model	**points**
Rand-model		925 - 623	904 - 668	830 - 823	5280
Bully-model	662 - 934		707 - 802	639 - 925	4006
Strategy-model	701 - 896	779 - 737		619 - 960	4241
Rdeep-model	835 - 791	903 - 638	891 - 672		5337

7000	Rand-model	Bully-model	Strategy-model	Rdeep-model	**points**
Rand-model		696 - 836	892 - 758	777 - 879	4683
Bully-model	865 - 686		862 - 726	747 - 771	4917
Strategy-model	747 - 881	651 - 884		719 - 897	4269
Rdeep-model	881 - 751	804 - 723	948 - 668		5180

8000	Rand-model	Bully-model	Strategy-model	Rdeep-model	**points**
Rand-model		892 - 666	898 - 705	858 - 807	5256
Bully-model	697 - 827	790 - 774		716 - 870	4407
Strategy-model	747 - 881	651 - 884		719 - 897	4408
Rdeep-model	881 - 751	804 - 723	948 - 668		5187

9000	Rand-model	Bully-model	Strategy-model	Rdeep-model	**points**
Rand-model		809 - 807	933 - 683	778 - 935	5015
Bully-model	810 - 827		876 - 725	686 - 931	4704
Strategy-model	686 - 924	763 - 813		604 - 1013	4087
Rdeep-model	960 - 744	965 - 712	1030 - 626		5834

10000	Rand-model	Bully-model	Strategy-model	Rdeep-model	**points**
Rand-model		849 - 710	827 - 771	900 - 793	5253
Bully-model	723 - 845		735 - 835	685 - 929	4353
Strategy-model	728 - 916	809 - 742		802 - 873	4740
Rdeep-model	836 - 916	871 - 758	859 - 795		5161

8.3 Data of Experiment 2

The following table depicts the total wins of different Machine-Learning-models playing against the Rdeep-bot.

training games	Lowest-Card-model	Bully-model	Strategy-model	Rdeep-model
1000	335	188	306	326
2000	347	393	308	389
3000	368	379	359	392
4000	360	238	267	390
5000	211	391	407	401
6000	384	300	399	447
7000	287	402	391	413
8000	385	269	407	467
9000	353	408	386	459
10000	351	358	198	405
total	3381	3326	3428	4089

8.4 Bot Ranking Data

The following table depicts how many games the different bots win against the Rdeep-bot in a fair tournament of 10,000 games.

	Rdeep-bot
Rdeep-bot	5061
Trump-bot	2282
Bully-bot	1414
Lowest-Card-bot	1198

8.5 Model Ranking Data

The following table depicts how many points different models win against the Rdeep-bot in a fair tournament of 1,000 games.

	Rdeep-bot
Rdeep-model	486 - 1117
Rand-model	561 - 984
Bully-model	454 - 1069
Strategy-model	487 - 1015
Lowest-Card-model	461 - 1067

8.6 Fair Tournament Implementation

This code was implemented so we could eliminate the element of chance when cross validating tournament-performance. The tournament uses the same set of game IDs every time and makes the participating two players switch perspective per game-ID. This lets both players have equal chance of winning as they play the same games from both perspectives.

```python
game_ID = 100000
print(f'Playing {int(total_games)} game(s):')
for a, b in matches:
    alternating_counter = 1
    for r in range(my_repeats):

        if alternating_counter == 1:
            p = [a, b]
            alternating_counter = 2
            print(f"new game ID: {game_ID}")
        else:
            alternating_counter = 1
            p = [b, a]

        # Generate a state
        state = State.generate(phase=my_phase, id=game_ID)

        # change game_ID every second game
        if alternating_counter == 1:
            game_ID += 1
        # run the game
        winner, score = engine.play(bots[p[0]], bots[p[1]],
                                    state, 1000, verbose, True)

        if winner is not None:
            winner = p[winner - 1]
            points[winner] += score
            wins[winner] += 1

        played_games += 1
```

8.7 Bots: get_move functions[8]

Trump-bot

```
def get_card_value(card):
    card = Deck.get_rank(card)
    if card == "10":
        value = 10
    elif card == "J":
        value = 2
    elif card == "Q":
        value = 3
    elif card == "K":
        value = 4
    else:
        value = 11
    return value

def remove_trump_from_hand(hand, state):
    trump_suit = state.get_trump_suit()
    index_of_all_trumps = []

    for x in range(len(hand)):
        if Deck.get_suit(hand[x]) == trump_suit:
            index_of_all_trumps.append(x)
    for index in sorted(index_of_all_trumps, reverse=True):
        del hand[index]
```

[8]The framework for these bots was provided by the Intelligent Systems course 2020 at the VU Amsterdam. The implementations of the Trump-bot and Lowest-Card-bot were written by us. The get_move functions are specified in this section (along with other relevant functions) since it is the most important function the bot uses for deciding on a move to play.

```
    return hand

def get_move(self, state):
    moves = state.moves()
    while len(moves) > 5:
        moves.pop()

    possible_cards = []

    for i in range(len(moves)):
        possible_cards.append(moves[i][0])

    # if Trump-bot leads
    if state.get_opponents_played_card() is None:

        # exclude trump from cards
        remove_trump_from_hand(possible_cards, state)

        # play highest card
        value = 0
        highest_card_index = 0
        for x in range(len(possible_cards)):
            if value > get_card_value(possible_cards[x]):
                pass
            else:
                value = get_card_value(possible_cards[x])
                highest_card_index = x
        chosen_move = moves[highest_card_index]
        return chosen_move

    # if other player leads
    else:
        # follow suit and play higher card
        for i in range(len(moves)):

                if Deck.get_suit(moves[i][0]) == \
                        Deck.get_suit(state.get_opponents_played_card()):
                    if get_card_value(moves[i][0]) > \
                            get_card_value(state.get_opponents_played_card()):
                    chosen_move = moves[i]
                    return chosen_move
                else:
                    continue

        # play trump if trump available
        for k in range(len(moves)):
            if Deck.get_suit(moves[k][0]) == state.get_trump_suit():
                chosen_move = moves[k]
                return chosen_move

        # play lowest card
        value = 6
        lowest_card_index = 0
        for x in range(len(possible_cards)):
            if value < get_card_value(possible_cards[x]):
                continue
            else:
                value = get_card_value(possible_cards[x])
                lowest_card_index = x
        chosen_move = moves[lowest_card_index]
        return chosen_move
```

Lowest-Card-bot

```
def get_card_value(card):
    rank = Deck.get_rank(card)
    if rank == "J":
        value = 2
    elif rank == "Q":
        value = 3
    elif rank == "K":
        value = 4
    elif rank == "10":
        value = 10
    else:
        value = 11
    return value

def get_move(self, state):
    moves = state.moves()

    possible_cards = []
```

```
for move in moves:
    possible_cards.append(move[0])

# play lowest card
value = float("inf")
lowest_card = None
for card in possible_cards:
    if card is not None:
        if get_card_value(card) < value:
            value = get_card_value(card)
            lowest_card = card

index = possible_cards.index(lowest_card)
chosen_move = moves[index]
return chosen_move
```