Influence of Depth, Samples, and Hidden Layers on Machine Learning Bots

2631465, 2696450, and 2704183

Vrije Universiteit Amsterdam, De Boelelaan 1105, 1081 HV Amsterdam

Abstract. This paper aims to look at the impact of the number of samples, depth, and the number of hidden layers on the success of machine learning bots. The bot whose features are changed is rdeep, a bot that uses Monte Carlo sampling to determine which actions to take. The improved bot is then used to make a data set for machine learning bots to be based on. The success of the bot is measured based on the bot's performance in Schnapsen, a card game similar to sixty-six. The goal of the changed bot is to outperform the original rdeep bot. After performing many tests, the modified rdeep bot whose number of samples and depth was increased had a higher chance of winning, however, it took longer to make a move. Furthermore, when this improved bot was used to create a data set with varied hidden layers, the difference in performance between the machine learning bots was trivial at best. When testing the difference between the best machine learning bot and the original rdeep bot, there was very little difference between the two when looking at small sample sizes, and at larger sample sizes, the rdeep bot ended up winning more games.

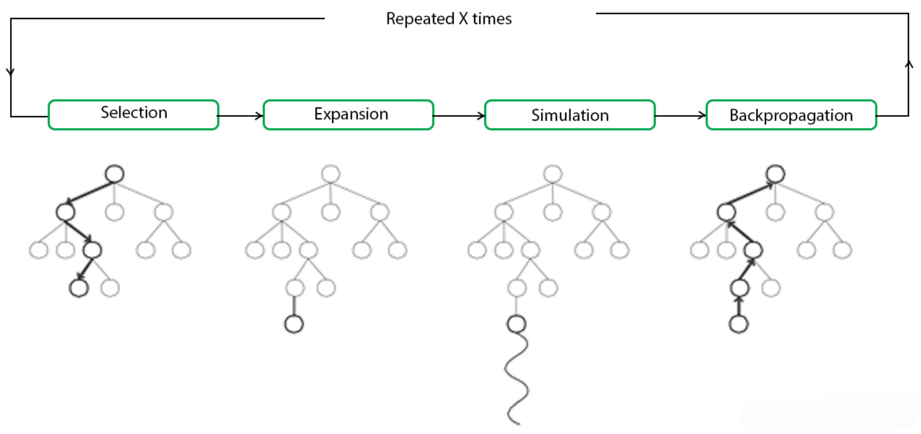
Keywords: Schnapsen · Machine Learning · Intelligent Systems · Artificial Intelligence · Hidden Layers · Monte Carlo Sampling

1 Introduction

In the past, games have been an entertaining way to display human intelligence in a way that computers were not able to partake in. This has changed, with the best GO [8] and chess players [4]both being computers. With their complexity, strict rules, and definite win states, playing games has become a spectacular way to put computer intelligence, or artificial intelligence, to the test. Both humans and computers today have the ability to look ahead to see the approximate heuristic value of playing in a specific move, however, computers have the advantage of being able to look through more moves and further into a game, more accurately and quicker than any human can [12]. This, along with specific algorithms and machine learning, has made artificial intelligence an impressive adversary to human players for nearly every game imaginable.

Machine learning is a domain of artificial intelligence that utilizes algorithms and large sets of data to recognize patterns. The data used for machine learning must be excellent, else the success of the bot's predictions may be compromised [11]. The objective of our bot is to win in games of Schnapsen, and the data of which it is trained on is the games it plays, and the success rates of those games. Based on prior knowledge and research, we hypothesize that there will be a significant difference between having a machine learning bot that is trained off of a modified rdeep bot with twenty samples and a depth of eight compared to the original rdeep bot. Our null hypothesis then is there will be no significant difference between the new machine learning bot and the original rdeep bot.

The bot which we modified and tested is the rdeep bot. Rdeep is a bot that uses a Monte Carlo sampling approach to playing Schnapsen. This approach to determining which card to play next is very effective due to its use of Monte Carlo sampling. This technique has been used to outperform humans in games such as GO [7], with all of the best programs using Monte Carlo sampling since 2006 [1]. A generic Monte Carlo search tree involves four steps: selection, expansion, simulation, and backpropagation [5]. These four steps lead the algorithm to the outcome of any particular move it makes, and the value of the moves. The rdeep bot does this by performing a random number of games to the end with different assumptions of the remaining cards and then assesses the outcome of the state. The bot makes several random choices and then the heuristic value of the moves is found. The best move is then obtained by seeing which of the sequence of moves leads to the best outcome. By doing this, the bot can, with high levels of accuracy, predict the move which will have the highest favorable outcome.



[2]

The number of samples and depth of the rdeep bot is modified to observe the effect, and these bots are used to train a new machine learning bot. The machine learning bot, or ML bot, is a bot that uses machine learning strategies to train and create a model containing details about these strategies. During the process of building a ML bot, an existing bot, for example the rdeep bot, is used to play large numbers of games against itself. Then the results of these games are stored in a data set containing which moves were favorable and led to gaining points and which moves were unfavorable and led to losing points. The bot then chooses the cards that are most likely to lead to a win, as that is its goal. Training a machine learning bot takes time depending on how many games are played and the number of hidden layers in the neural networks. When a ML bot is trained using a different bot, it will generate a different data set model and therefore it will perform differently.

2 Background Information

2.1 Schnapsen

Schnapsen is a two-player card game popular in Bavaria, and similar to sixty-six. The game utilizes twenty cards, with the cards and their respective points as follows: ace worth eleven points, the 10 worth ten points, the King worth four points, the Queen worth three, and the Jack worth two points. Each player receives five cards to start, while the rest of the cards are placed face-down in the talon. One card is turned and placed under the talon in a way that the suit can be seen, and this card dictates the trump suit.

The game is played by players playing tricks, in which each player places a card onto the table. The winner is determined by the points of the cards played and whether the card is a card of a trump suit or not. The winner takes a card from the talon and plays the next trick first, while the loser takes a card from the talon after the winner. Collecting cards continues until either there are no more cards left in the talon, or a player has won. A player wins when he or she reaches 66 points.

There are also two other moves that can be played. The first is a trump exchange, where the leader of a trick can replace the Jack of the trump suit with the card dictating the trump suit. The second is a marriage, by which the leading player can receive either twenty or forty points by showing a Queen and King of the same suit, and then playing the Queen in the trick. A normal marriage is worth twenty points, whereas a marriage consisting of the trump suit is worth forty points. These points are received once the player wins either the current trick or a future trick. A final move that a player may do is to close the talon in order to force the game into Phase 2 (explained in the following paragraph), however, this aspect of the game is removed in this version of the game for the sake of simplicity.

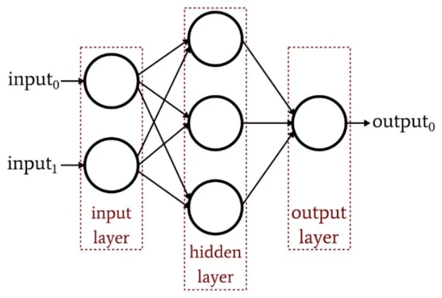
The game consists of a semi-observable and fully-observable phase, commonly referred to as Phase 1 and Phase 2. Phase 1 is before all of the cards in the talon are exhausted when the player does not have knowledge of the opposing player's cards, and as a result, the player has imperfect knowledge. Phase 2 is once all of the cards have been picked up; this phase can be solved using numerous strategies such as alpha-beta pruning as the cards of each player is now known. In Phase 1, a player does not need to follow suit, while in Phase 2, a player must follow suit. Only if this is not possible, is a player allowed to play any card.

The number of points a player receives for winning is determined by how well the opposing player played. The winning player receives three points if the opposing player won no trick, two points if the opposing player received fewer than 33 points, and one point otherwise (unless there is a draw, in which case neither player receives points).

2.2 Hidden Layers

The topic of hidden layers is notable within machine learning as it is a part of the neural network system. The reason why it is called hidden layers is because hidden layers are located between the input and output layers. Hidden also refers to the fact that hidden layers are not recognizable for public use as they are only used in neural networks. Hidden layers essentially "perform non-linear transformations of the inputs entered into the network" [6]. The difference between hidden layers is dependent on the function of the neural network and their related weights.

The number of hidden layers in a neural network depends on the complexity of the problem. This means that a large number of hidden layers is required when dealing with a complex problem [10]. One of the main drawbacks of many hidden layers, however, is that it will result in a long wait time to produce a solution. Nonetheless, for many problems, "especially those that are linearly separable, one hidden layer can suffice"[9].



[9]

3 Research Question

How does the success rate of a machine learning bot trained off of rdeep change when the number of samples, depth, and the number of hidden layers is modified?

4 Experimental Set-up

4.1 Experiment 1: rdeep bot versus rdeep1 bot

The original rdeep bot, which was provided by the Intelligent Systems course, uses a depth of eight and four samples. This works fairly well within the given restrictions, as its time to complete a move typically falls within the time limit. Limiting the depth can lead to the bot only evaluating the heuristic of a move in a top-down approach based on the current, prior, and possible upcoming positions. The upcoming positions are somewhat limited however, leading to the possibility of the horizon effect, where an obviously poor move is chosen as the depth is limited, and therefore the knowledge of this poor move goes unknown [3]. Knowing this, we decided to create a bot very similar to rdeep, where the number of samples and the depth were expanded.

Rdeep1 is a bot we introduced as an adversary to the original rdeep bot. Rdeep1 increases both the number of samples and the depth of the bot as compared to rdeep. Rather than four samples and a depth of eight, rdeep1 uses twenty for its number of samples and a depth of ten. In order to see the effectiveness of the new rdeep1 bot, three tournaments were conducted, each consisting of a different number of games. Each tournament had rdeep and rdeep1 playing against each other, with game 1 consisting of 10 games, game 2 consisting of 100 games, and game 3 consisting of 500 games.

Due to what we know about the effect of increasing the number of samples and depth for top-down Monte Carlo sampling, we would expect to see rdeep1 to outperform the original rdeep. Rdeep1 will be able to explore more options and as a result, we would expect rdeep1 to have a higher chance of finding an optimal move. The null hypothesis would be if there was no statistical difference between the two bots' performance. Although we would assume that rdeep1 has a higher chance of succeeding based on our prior knowledge, the additional options could create an overload of choice, leading to poor information. This can consequently negatively influence the time it takes for rdeep1 to make a move.

4.2 Experiment 2: ML-bots tournament

In order to test what, if any, difference there is between machine learning bots that are trained off of different numbers of hidden layers, we tested three different bots. Rdeep1 was used to train machine learning bots and create different models. The only difference between each model is the number of hidden layers inside the neural networks.

A data set was created by using sklearn to train machine learning bots on rdeep1. To certify that the machine learning bots would be a satisfactory opponent, it was trained by playing 10,000 games, which ended up taking about thirteen hours. This is not unexpected, however, as rdeep1 was already shown to take longer to make a move as compared to rdeep, in addition to the large number of games stored in the data set. Once the generation of the data set was complete, three different numbers of hidden layers were considered.

The number of hidden layers was either one, two, or three hidden layers. These each produced a data set model that was stored in a ML-1, ML, and ML-3 bot respectively. The final step of this experiment was to play a tournament consisting of the three machine learning bots. The best machine learning bot would then be tested against the rdeep bot to see how well this trained bot could perform against an already impressive bot with no machine learning implementation.

4.3 Experiment 3: ML-3 versus rdeep

The performance of the three machine learning bots with different numbers of hidden layers is analyzed, and the best out of the three is chosen. This ML bot then plays against the original rdeep bot. This allows the bot to be analyzed for its final point score against one other opponent rather than a tournament style of counting wins.

The tournament is played with three different sample sizes. These are 10, 100, and 500 game tournaments. These games measure the performance of the best-performing version of the machine learning bot trained with the improved version of rdeep (rdeep1). Additionally, this experiment will provide the answer to whether this improved machine learning bot can beat the original rdeep bot.

5 Results

5.1 Experiment 1: rdeep bot versus rdeep1 bot

The experiment which is demonstrated in graph 1 shows that in the first two tournaments the 10 and 100 games there was not such a significant difference in performance between rdeep and rdeep1, even in the tournament of 100 games rdeep was able to gain more points than rdeep1. Nevertheless, in the 500 games tournament, it was clear that rdeep1 is a better version of rdeep where the difference was evident in the 100 extra points rdeep1 has scored.

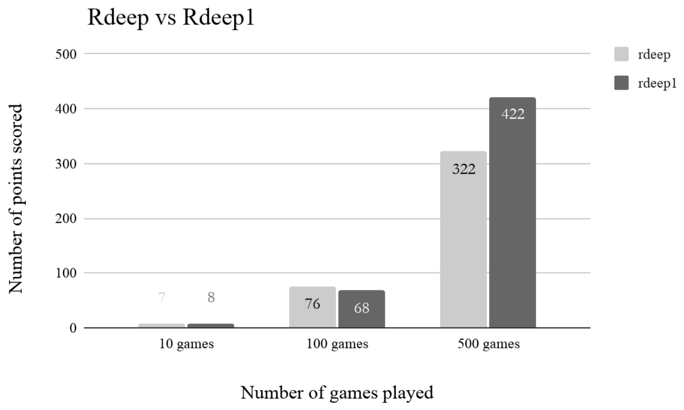


Fig. 1. Graph displays how many points each bot scored for tournaments consisting of 10, 100, and 500 games

Table 1. Comparison between the depth and number of samples between the two bots, and how this changes the average time needed to complete a move.

	rdeep	rdeep1
Depth	8	10
Sampels	4	20
Average time to complete a move	236.244 millisecond	610.446 millisecond

5.2 Experiment 2: ML-bots tournament

In this experiment, three machine learning bots played a tournament of 300 games. As the graph below shows, the performance of the three bots was not noticeably different. This result is to be expected since all three bots are different training models for the same data set. ML-3 bot won the tournament with a slightly better score than ML-1. These results are reasonable because ML-3 bot uses three hidden layers in the neural network while training a data set model compared to only one neural network used by ML-1 bot. Surprisingly, ML-2 bot had the weakest performance in the tournament despite the fact it uses two hidden layers in the neural network while training a data set model. Based on the p-values for a one-tailed test with an alpha of 0.05, the only statistically significant difference in the bots is the difference between the performance of ML-2 and ML-3, with 89 and 110 points respectively. They have a p-value of 0.03438, which is under the threshold for significance as defined earlier. Based on these results, ML-3 bot is the best performing bot that qualifies to play against rdeep in Experiment 3.

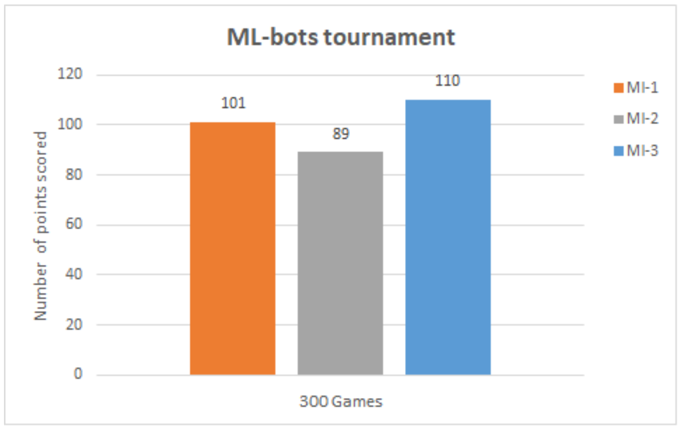


Fig. 2. Graph shows the difference in performance between the three bots, each with a different number of hidden layers

Table 2. p-values of a one-tailed test with $\alpha = 0.05$, based on how it performs in a tournament.

	Ml-1	Ml-2	Ml-3
ML-1	-	0.14686	0.22065
ML-2	0.1468	-	0.03438
ML-3	0.22065	0.03438	-

5.3 Experiment 3: ML-3 versus rdeep

Unfortunately, as seen from table 4 below, the machine learning bot could at best, win in small sample sizes. When looking at larger sample sizes, the machine learning bot could tie at best, and lose by a sizable margin otherwise. This indicates that the rdeep bot is still able to outperform the machine learning bot and that the improvements made to the training bot (rdeep1), the increase in hidden layers, and the machine learning did not contribute to an improvement in the bot's performance against the rdeep bot.

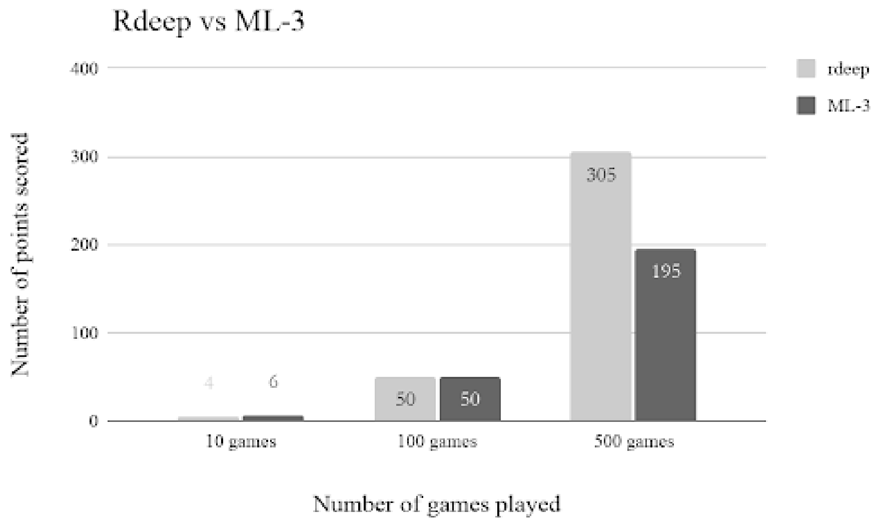


Fig. 3. Graph shows the performance of rdeep and ML-3 in a tournament against one another.

6 Findings

The results of our first experiment indicate that increasing the depth of the rdeep1 bot to eight and the number of samples to twenty does increase the odds of rdeep1 winning against the rdeep bot. This however comes at the cost of looking through more game states, and as such, the time it takes to make a move is increased. This means that it is not really practical for games that have a time limit for making a move.

Our second experiment with the hidden layers indicates that there is not a large difference between the bots with one, two, or three hidden layers. This is not necessarily surprising as all bots were trained off of the same data set. There was a statistically significant difference between ML-2 and ML-3 when looking at the p-values of a one-tailed test with an alpha value of 0.05, which would indicate the difference seen between the performance of the two bots is not due to chance. Other than those two bots, the other p-values indicate that there is no statistically significant difference between the bots.

Our final experiment showed that in our games with the original rdeep bot and the modified machine learning bot (ML-3), the original rdeep bot would still outperform the machine learning bot. The machine learning bot would at best, tie or slightly outperform the rdeep bot in small sample sizes, however when looking at larger sample sizes and therefore more accurate data, the rdeep bot would typically quite easily outperform the machine learning bot.

7 Conclusions

From this research, it can be concluded that increasing the number of samples to twenty and the depth to eight in the rdeep bot improves the bot's performance against the original rdeep bot. This comes with the cost of time, with the rdeep1 bot taking significantly longer (over double the time of the rdeep bot, or 374.202 milliseconds longer on average) to make a move when compared to rdeep. Furthermore, we can conclude that the number of hidden layers does not have a significant role in the performance of a bot when played against bots with different numbers of hidden layers but based on the same data set. This indicates that the determiner for success when it comes to these machine learning bots has very little to do with the number of hidden layers it has. Finally, while we were able to create a machine learning bot that performed well when played against the original rdeep bot, the improved machine learning bot (ML-3) would perform well enough to only win about half of the time at best.

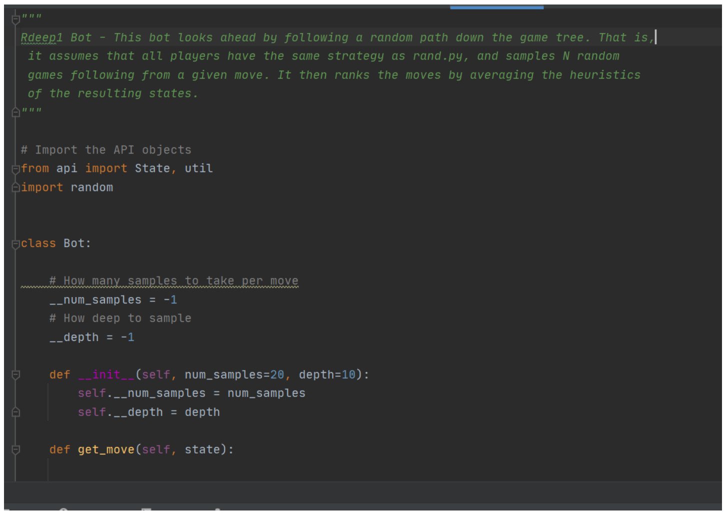
While the number of games played by each bot was in the hundreds, and therefore a good sample size, the number of variables tested could be expanded on. Future research could focus on looking at the impact of different combinations of depths and number of samples, and how this affects the machine learning bots. This would however likely take a lot of time, as just making one data set already took around thirteen hours. Furthermore, this research could be expanded further to look at the effect of different combinations of depths and number of samples on machine learning bots on bots other than rdeep, or even looking at if there is any difference in the performance of bots other than the rdeep bot when the number of hidden layers in a data set is changed. With the findings of our study, this could be considered cursory research as the number of hidden layers did not seem to have a significant role in the performance of the rdeep machine learning bot, however, there is the possibility that further research shows that hidden layers have a more significant effect for other bots. To test our bots, we compared our bot's performance to the performance of rdeep. Future research could also look into how these new bots compare to other bots.

References

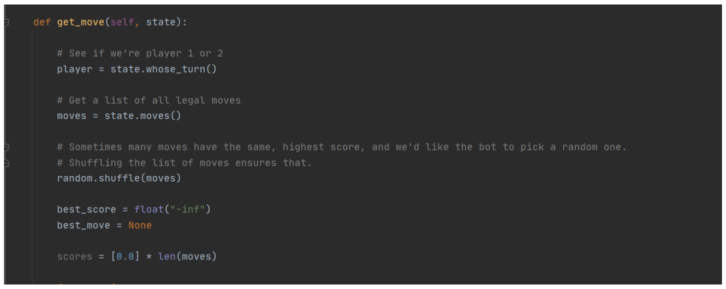
1. (2011), https://senseis.xmp.net/?KGSBotRatings
2. (Jan 2019), https://www.geeksforgeeks.org/ml-monte-carlo-tree-search-mcts/
3. (2021), https://www.oxfordreference.com/view/10.1093/oi/authority.20110803095944934
4. Cazenave, T., Winands, M.H., Saffidine, A.: Computer games. Springer (2018)
5. Chaslot, G., Winands, M., Uiterwijk, J., Van Den Herik, H., Bouzy, B., Wang, P.: Progressive strategies for monte-carlo tree search. In: Proceedings of the 10th Joint Conference on Information Sciences (JCIS 2007). pp. 655–661 (2007)
6. DeepAI: Hidden layer (May 2019), https://deepai.org/machine-learning-glossary-and-terms/hidden-layer-machine-learning
7. Gelly, S., Silver, D.: Monte-carlo tree search and rapid action value estimation in computer go. Artificial Intelligence **175**(11), 1856–1875 (2011)
8. Grant, T.D., Wischik, D.J.: On the path to AI: Law's prophecies and the conceptual foundations of the machine learning age. Springer Nature (2020)
9. https://www.facebook.com/MachineLearningMastery: How to configure the number of layers and nodes in a neural network (Jul 2018), https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/
10. Michelucci, U.: Applied deep learning—a case-based approach to understanding deep neural networks; apress media, llc: New york, ny, usa, 2018. Tech. rep., ISBN 978-1-4842-3789-2
11. Mohri, M., Rostamizadeh, A., Talwalkar, A.: Foundations of machine learning. MIT press (2018)
12. Yannakakis, G.N., Togelius, J.: Artificial intelligence and games, vol. 2. Springer (2018)

Appendices

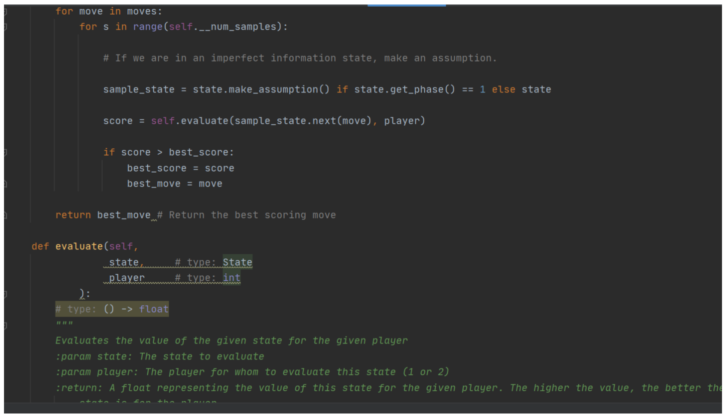
Below shows the rdeep1 code whose only modifications are the depth and the number of samples. This is the code which was used to train all of the ML bots.



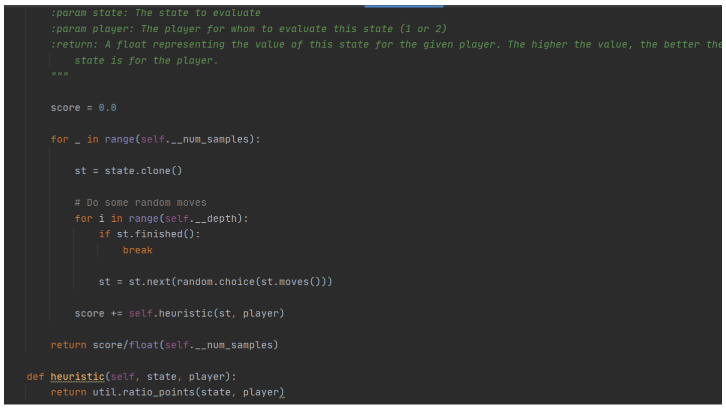
```python
"""
Rdeep1 Bot - This bot looks ahead by following a random path down the game tree. That is,
 it assumes that all players have the same strategy as rand.py, and samples N random
 games following from a given move. It then ranks the moves by averaging the heuristics
 of the resulting states.
"""

# Import the API objects
from api import State, util
import random


class Bot:

    # How many samples to take per move
    __num_samples = -1
    # How deep to sample
    __depth = -1

    def __init__(self, num_samples=20, depth=10):
        self.__num_samples = num_samples
        self.__depth = depth

    def get_move(self, state):
```



```python
    def get_move(self, state):

        # See if we're player 1 or 2
        player = state.whose_turn()

        # Get a list of all legal moves
        moves = state.moves()

        # Sometimes many moves have the same, highest score, and we'd like the bot to pick a random one.
        # Shuffling the list of moves ensures that.
        random.shuffle(moves)

        best_score = float("-inf")
        best_move = None

        scores = [0.0] * len(moves)
```



```python
        for move in moves:
            for s in range(self.__num_samples):

                # If we are in an imperfect information state, make an assumption.

                sample_state = state.make_assumption() if state.get_phase() == 1 else state

                score = self.evaluate(sample_state.next(move), player)

                if score > best_score:
                    best_score = score
                    best_move = move

        return best_move # Return the best scoring move

    def evaluate(self,
                 state,     # type: State
                 player     # type: int
             ):
        # type: () -> float
        """
        Evaluates the value of the given state for the given player
        :param state: The state to evaluate
        :param player: The player for whom to evaluate this state (1 or 2)
        :return: A float representing the value of this state for the given player. The higher the value, the better the
```



```python
        :param state: The state to evaluate
        :param player: The player for whom to evaluate this state (1 or 2)
        :return: A float representing the value of this state for the given player. The higher the value, the better the
            state is for the player.
        """

        score = 0.0

        for _ in range(self.__num_samples):

            st = state.clone()

            # Do some random moves
            for i in range(self.__depth):
                if st.finished():
                    break

                st = st.next(random.choice(st.moves()))

            score += self.heuristic(st, player)

        return score/float(self.__num_samples)

    def heuristic(self, state, player):
        return util.ratio_points(state, player)
```