PROJECT REPORT: IMPROVING RBC AGENT

Puranjay Datta
19D070048

Shubh Kumar
200050134

Introduction

Reconnaissance Blind Chess (RBC) is a unique and challenging chess variant that introduces a new level of uncertainty to the game. Unlike traditional chess, RBC requires players to perform a Sense move before each play, allowing them to gather information on a 3x3 board. This added layer of complexity makes decision-making more difficult and calls for creative and adaptive mixed strategy policies. The RBC competition, with a 15-minute time control and a 5-second increment, as well as a 50-move draw rule, showcases the skills and strategy of participating bots. In the 2021 NeurIPS competition, Fianchetto, developed by Professor Shivaram and his PhD student Anvay, emerged as the winner. The following year, in 2022, Fianchetto came in second, losing to Strangefish2. You may see a sample game here.

Background

Fianchetto, models the RBC game as a Partially Observable Markov Decision Process (POMDP) and updates its belief at each move using Bayes' rule. The opponent's move policy, represented by the probability distribution $Pr(a|s)$, is estimated through a combination of LC0 and RBC-specific heuristics. These heuristics include giving higher incentives to pawn moves due to their lower detection risk, as well as penalizing moves that result in pieces being placed on unprotected squares, which are deemed high-risk. The implementation of these heuristics helps Fianchetto make more informed decisions and achieve its winning performance in the competition.

$$Pr(s'|b,z) = \sum_a Pr(s'|b,a,z) \sum_s Pr(a|s)Pr(s|b) = \sum_a Pr(z|s',a) \sum_s Pr(s'|s,a)Pr(s/b) \sum_s Pr(a|s)Pr(s|b) \tag{1}$$

$$Pr(a|s) = softmax(Lc0LastLayer(s) + c) \tag{2}$$

c :incentive vector for all moves
a :opponent's move
s' :new state(after opponent's move)
s :old state(before opponent's move)
b :belief(probability) over the old states
z :observation

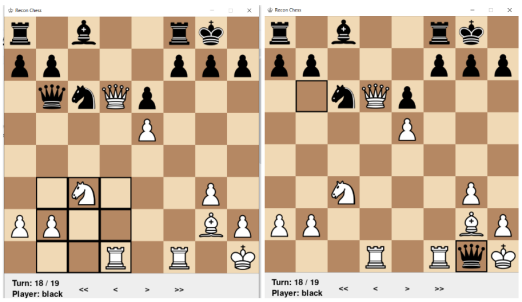
Past Blunders

Looking at the games of the previous year's Reconnaissance Blind Chess matches between fianchetto and strangefish2, we can see that our bot was :

1. Making some silly mistakes in positions that we do not expect
2. Failing to predict some of the moves made by strangefish, such as sneak attack moves (explained later). Therefore, the goal of this research is to analyze Fianchetto's peculiar behavior and improve our ability to anticipate our adversaries' moves in to deal with them more effectively.

Let's look at some of the blunders that fianchetto made

1. In a game of Reconnaissance Blind Chess (RBC), Fianchetto was caught off guard by a sneaky move from its opponent, Strangefish. Despite playing the best move based on its current understanding of the game, Fianchetto was checkmated on the next move as it was unable to predict Strangefish's queen sacrifice. Had Fianchetto taken the time to sense near its king position, it could have easily captured Strangefish's queen and won the game. This example highlights the importance of using the Sense move in RBC to gather crucial information and make informed decisions.



(a) Example 1

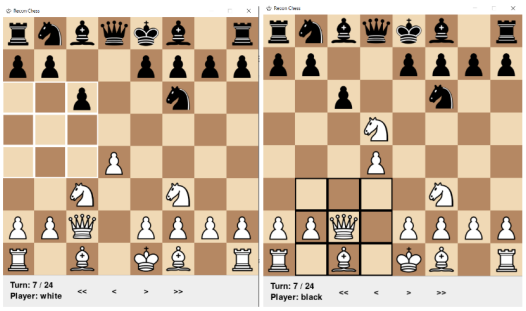
(b) Example 2

Figure 1: Sneak Attack

2. In this game below, the pawn was not poisoned and we let the black queen take our b2 pawn and hence had a worse position. Then later in the game, we had the chance to take the black queen with our rook but we did not.Finally, we had the chance to have an advantage by playing h5 which was the best move but we played Qf3.

Figure 2: a) Not a poisoned pawn-b) Forgot to take the Queen-c)Why not h5

3. We sacrificed our knight thinking that there was a queen on that d5 square as seen from the top board set at that instant which we were able to obtain from the replay buffer.



(a) Sacrificed our knight for no compensation

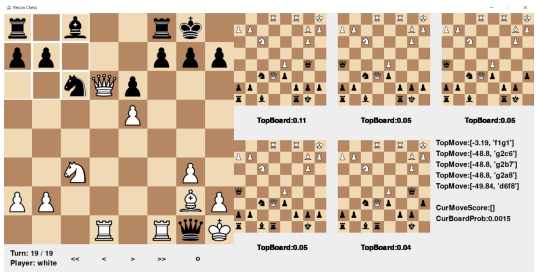
(b) Example 2

Figure 3: Top board in the set with probability=0.309

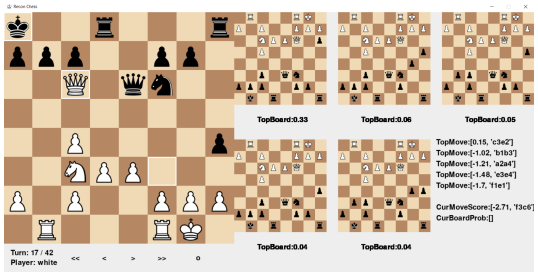
Blunder analysis using Replay Buffer

Replay buffer:

The Replay Buffer will play back the whole game we want to view while also keeping track of the board probabilities and the score for each move. For instance, to replay the game $G : w_1, b_1, w_2, b_2, ...$, we attempted to mimic the normal game scenario with the sole exception that the opponent bot was replaced by a dummy bot that would simply read the moves from the game history and play them out exactly as they were, and the fianchetto bot was replaced by a fianchettobuffer that would calculate the board probabilities, move, and sense score in the same manner but would play the move obtained from the history in place of the best move according to the scenario. Hence we were able to replay the game and obtain the board sets and move the score from which it had decided while playing the particular game.To better visualize the board set we created a GUI that displays the top 5 boards along with their probabilities at every move and along with it a list of move scores and the probability of true board state.

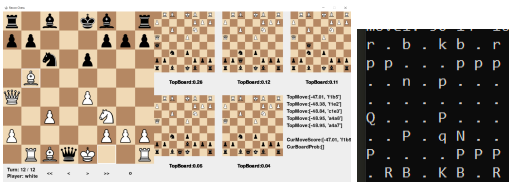


(a) Sneak Attack correctly identified by Fianchetto

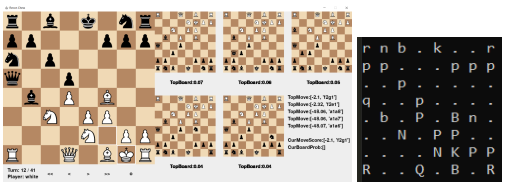


(b) Should have played Ne2 instead of Qxc6

1. After some thorough inspection of the sneak attack using the replay buffer we have created, we found that, ignoring the time situation, fianchetto was indeed able to detect it and hence the top move suggested by it is f1g1 as seen in fig (a). Astonishingly it is the only move in the list of best moves and all the other moves have a score(≈ 48) below the threshold.
2. In fig (b), the queen sacrifice was not intended and is clearly indicated by the move score of f3c6(-2.71) which is significantly smaller than the best move c3e2(0.15).Hence like the previous case the move was not even in the best move list which was testified using the replay buffer that was created.
3. The next example is an interesting case where a possibility of checkmate in next move dominates its score. We found that the compound score was being calculated the following way- $compound\_score = 0.7mean(score, board\_prob) + 0.3min(score)$. score assigned to a possibility of a checkmate is -160 hence the score is being dominated by the factor $-0.3 * 160 = -48$.
   In the case (a) below there are two boards that threaten checkmate but a single sense move can detect both the threats i.e if we sense the squares given by (c1,e1,e3,c3).This can detect both queen threats and after some analysis we found that the sense square suggested by fianchetto is indeed the same.
   In the case (b) there is checkmate threat which supersedes the chance to capture the black queen with the rook.Hence the move f2g1 is the best move whereas Ra1a6 is the third best move as seen in the replay buffer.



(a) Sneak Attack Undetected

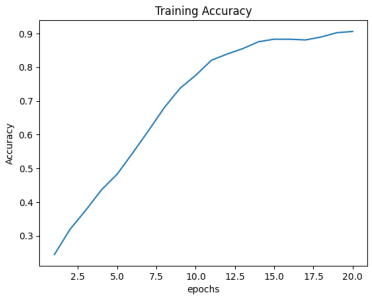


(b) Ra1a6 third best move with a very bad score

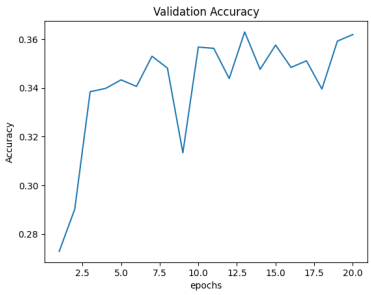
Opponent Modelling

Opponent Model Training

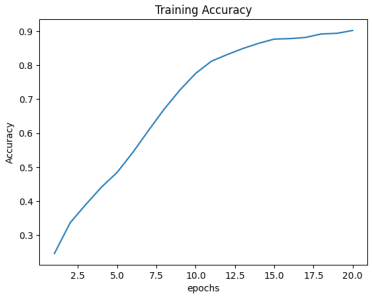
The figure below shows the training using Gurnoor and Dhruva's training pipeline [1] and validation accuracy while training lc0 weights as per three different opponents -SF2,ROOKie,Oracle. The validation accuracy was fluctuating between 0.3 and 0.36 indicating may be we need more data to learn the opponent model better.



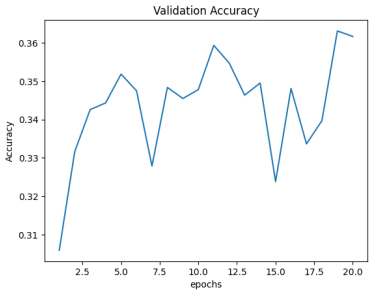
(a) SF2 Training Accuracy



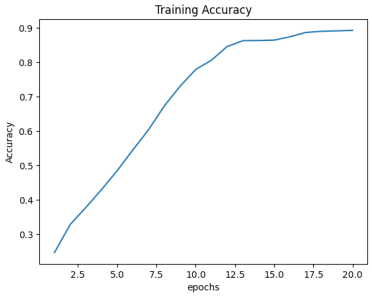
(b) SF2 Validation Accuracy



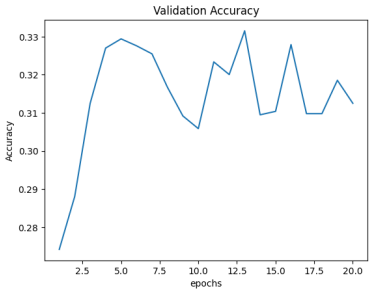
(c) ROOKie Training Accuracy



(d) ROOKie Validation Accuracy



(e) Oracle Training Accuracy



(f) Oracle Validation Accuracy

Results

After training the weights of lc0 network for specific opponents using the last years NeurIPS 2022 Tournament [2] which had 1020 games for each opponent. The games were played on the server on a span of 8hrs which paired Fianchetto with various bots which were available at the time.Results are listed as win-loss-draw for weights trained according to SF2, ROOKie and Oracle.

Opponent	attacker	StrangeFish2	ROOKie	Oracle	trout	random
Baseline	43-2-0	18-16-5	32-9-2	37-7-0	43-1-0	44-0-0
(SF2)	27-0-0	0-22-0	2-23-0	0-26-0	10-15-1	25-1-0
(ROOKie)	30-13-0	1-42-0	5-38-0	2-39-0	40-1-0	43-0-0
(Oracle)	48-1-0	1-48-0	3-46-0	4-44-0	29-17-1	49-0-0

Conclusion

Future work and Discussion

As you can see the win percentage dropped significantly(even for Strangefish2 according to which the weights were modelled). We also tried other opponent model like ROOKie and Oracle but the results i.e the win percentage dropped. Hence on analyzing 4 positions where blunders happened we understood that the root cause of the problem was move-score. Currently, the move score is being calculated as $score = board\_prob.move\_score\_for\_that\_board$).So there

was a situation where we blundered a rook as there were two possible boards one where the same move would result in a checkmate(board1) and the other(board2) where it was a blunder. The checkmate was assigning it a $score = 120$ and the rook blunder was assigned a $score = -6$.$score = board\_prob[1] * 120 + board\_prob[2] * (-6)$, hence dominated by checkmate. Even if we correctly identify the board probability(by opponent modeling) we have to ensure that $board\_prob[1] <<< board\_prob[2]$ so that the entire score is weighted properly.We believe that instead of opponent modeling(which could work if we can train on a lot of games) we can try reward/score learning or assign a risk measure with every move and do a risk minimization scheme.

Plausible Bugs in Gurnoor/Dhruva's Model

They wanted to fine-tune the LCZero model for RBC but the problem was that the outputs given by the lc0 python API and the output given by the lc0 model loaded in tensorflow (for the same weights) were not equal (or even close) for some cases. They found it difficult to emulate the lc0 encoding for the game state and the lc0 model loading in tensorflow so they used the lc0 python API encoding function for getting the input tensors (check out lc0-prediction/lc_encoder.py in the github repo) and the model loading function present in lc0 python code. The above ensured that the input tensors and model loaded were same through both the python API for lc0 and the colab notebook. So, ideally, the output tensors given by both the API and the tensorflow model should have been identical. But, the output tensors only agreed on about 60% of the cases. For the remaining 40%, the outputs were not close.

Owing to the change in the lc0 output when the same piece of code was ran twice. This suggests that there is some randomness in the lc0 evaluate function.We feel that the lc0 python API has both a model and Monte Carlo Tree Search algorithm [3] to find the best move using a Upper Confidence Bound algorithm which imparts some randomness. Hence we cannot expect just the model to predict as good as the lc0 API which has a search algorithm also. The colab notebook has just a model which(no MCTS) which results in a much poor output without training,whereas the python API had a poor prediction at those moves where there were multiple good moves for the opponent.

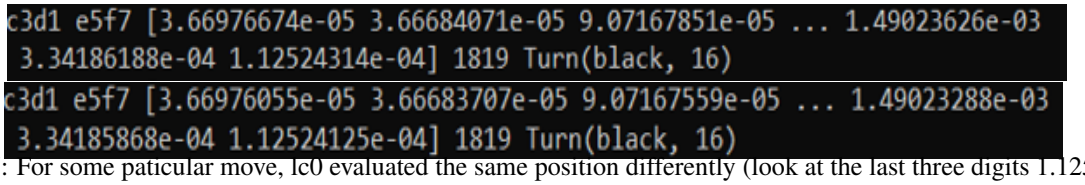


Figure 7: For some paticular move, lc0 evaluated the same position differently (look at the last three digits 1.12524314 changed to 1.12524125)

Acknowledgments

We would like to thank Gurnoor and Dhruva for providing the training pipeline which helped us greatly in learning the lc0 weights and Anvay for helping us get familiar with the Fianchetto framework and debugging the replay buffer.

References

[1] Gurnoor and dhruva's last year cs 748 report and code repository.

[2] Reconnaissance blind chess. (n.d.). retrieved april 28, 2023, from https://rbc.jhuapl.edu/.

[3] lczero retrieved april 28, 2023, from https://lczero.org/.